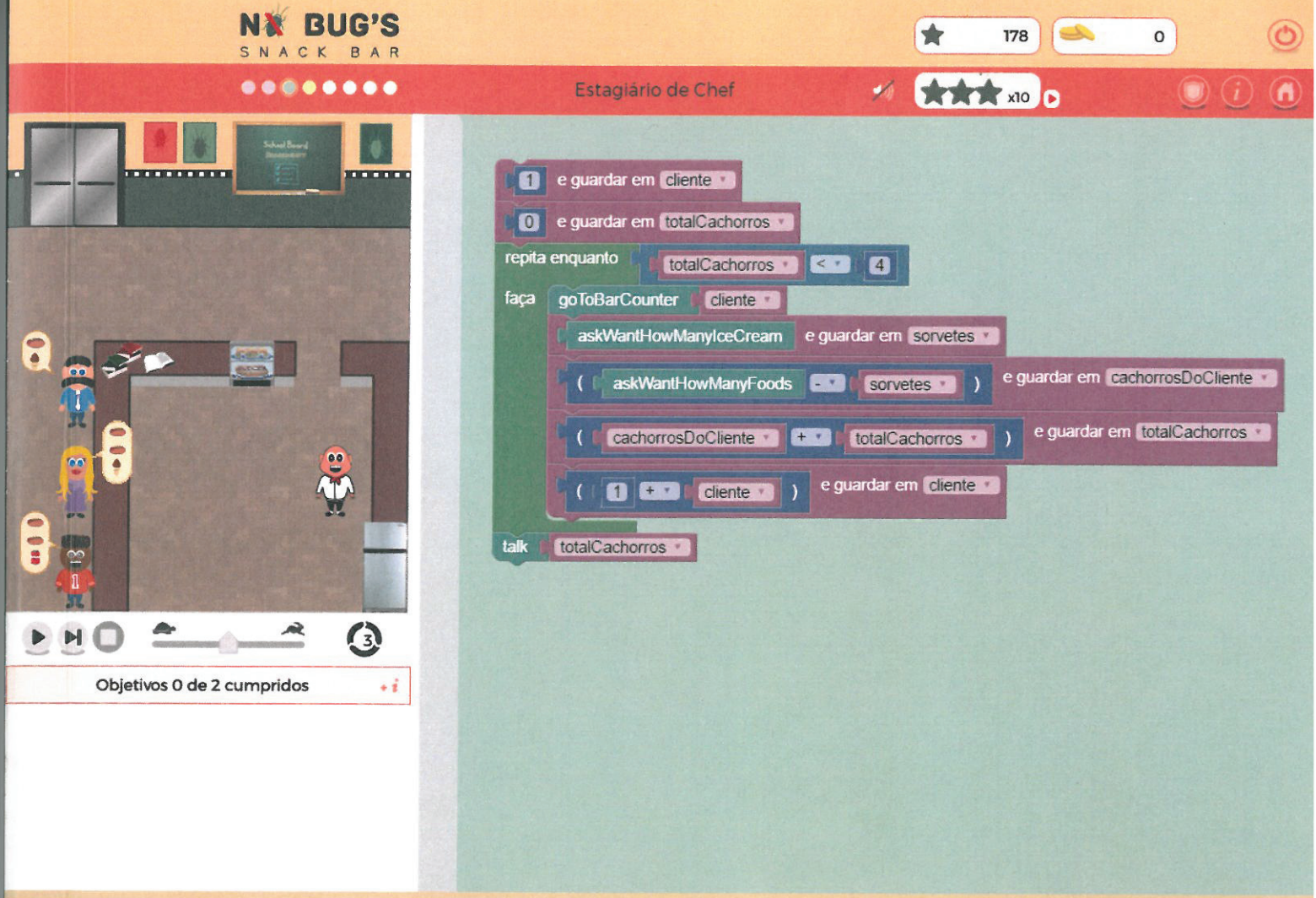


Adilson Vahldick

Aperfeiçoamento das competências de resolução de problemas na aprendizagem introdutória de programação de computadores usando um jogo sério digital

Tese de doutoramento do Programa de Doutoramento em Ciências e Tecnologias da Informação, orientada pelo Professor Doutor António José Nunes Mendes e Professora Doutora Maria José Patrício Marcelino e apresentada à Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Novembro/2017

· U C ·

UNIVERSIDADE DE COIMBRA

Adilson Vahldick

APERFEIÇOAMENTO DAS COMPETÊNCIAS DE RESOLUÇÃO DE PROBLEMAS NA APRENDIZAGEM INTRODUTÓRIA DE PROGRAMAÇÃO DE COMPUTADORES USANDO UM JOGO SÉRIO DIGITAL

Tese apresentada como requisito parcial para obtenção do grau de doutor em Ciências e Tecnologias da Informação pela Universidade de Coimbra.

Orientador: Prof. Dr. António José Nunes Mendes
Co orientador: Prof. Dra. Maria José Patrício Marcelino

Coimbra
2017

“Sem sacrifício não há vitória”
Optimus Prime

À minha família.

AGRADECIMENTOS

Agradeço à Deus, pela dádiva da vida e todas as graças alcançadas durante os quatro anos desse trabalho.

Aos meus amigos e orientadores, Professor Doutor António José Mendes e Professora Doutora Maria José Patrício Marcelino, que desempenharam o papel de artesãos, permitindo o meu desenvolvimento como investigador, combinando elogios e reprovações na medida certa sempre pensando na minha integridade.

De forma especial, reconheço a importância da família, minha esposa Darlane, e meus filhos Bruce e Celine, pelos momentos de paciência e apoio.

Tenho que agradecer ao Professor Paulo Roberto Farah por ter aceitado disponibilizar as suas turmas para as experiências no Brasil, e pelo nascimento de seu filho Eduardo Crestani Farah ter aberta a oportunidade de fazer uma experiência diferenciada. Tenho que agradecer aos quase 150 alunos que experimentaram o jogo, e os professores que dispuseram a permitir os testes, pois sem a contribuição deles, também não teríamos esse trabalho concluído.

Não posso deixar de agradecer a oportunidade do financiamento dessa investigação pelo CNPq/CAPES – Programa Ciência sem Fronteiras – CsF (6392-13-0) e pelo afastamento para capacitação pela Udesc (688/13 e 970/16).

Muita gente fez parte da minha vida durante o desenvolvimento desse trabalho, e que de alguma forma, contribuiu com alguma linha de código, ponto de vista ou pensamento. Do fundo do coração, agradeço muito a todas essas pessoas.

RESUMO

Os investigadores que se dedicam à área do ensino e aprendizagem inicial de programação continuam a classificar e identificar as dificuldades dos estudantes, e a tentar desenvolver metodologias e ferramentas que os possam ajudar, especialmente aqueles que apresentam mais dificuldades para alcançar as competências necessárias.

Neste trabalho investigamos o uso de jogos sérios para suportar estudantes que iniciam a aprendizagem de programação. Foi desenvolvido um jogo sério, chamado NoBug's SnackBar, com elementos de diversão que normalmente não são encontrados em outros jogos de mesma natureza, com intuito de atrair e motivar os alunos nas fases iniciais de aprendizagem para que tomem gosto pelo jogo, e com isso, desenvolvam suas competências de programação. A mecânica e as regras do jogo são simples, de modo a que seja fácil aprender como o jogo, facilitando a sua introdução e integração como ferramenta nas disciplinas introdutórias de programação. O jogo inclui um conjunto de missões, que se podem cumprir através de pequenos programas. É usada uma abordagem baseada em blocos, para tornar o jogo independente da linguagem de programação, permitindo aos estudantes focar-se nos problemas a resolver e não na sintaxe de alguma linguagem de programação. O jogo é executado num navegador, não requerendo grandes requisitos de *hardware* e nem a instalação de *software* adicional. O professor conta com uma área administrativa que destaca os alunos com maior dificuldade em progredir, e assim permite que ele(a) monitore a evolução de cada aluno, em cada uma de suas tentativas.

O projeto seguiu a metodologia de investigação Design-Based Research, tendo sido realizados quatro ciclos, o primeiro foi um piloto para validar a ideia, e os três seguintes de maior duração, cada um deles cobrindo praticamente um semestre inteiro, para que fosse possível verificar mudanças significativas na aprendizagem e na adesão ao jogo, bem como identificar alterações e/ou melhorias que era necessário incluir no jogo ou na sua forma de utilização. Os alunos tiveram a liberdade de usar o jogo quando e quanto queriam, e assim acreditamos ter uma avaliação mais verídica sobre a sua utilidade pedagógica. No final de cada ciclo, foram identificadas algumas ideias a serem testadas no ciclo seguinte, levando mudanças no jogo. Em cada um

dos três ciclos foram testadas formas distintas de organização e disponibilização das tarefas de aprendizagem, bem como de integração do jogo com as aulas.

Como produto dessa tese, além do jogo, todas as observações efetuadas durante os ciclos de avaliação e desenvolvimento nos permitiram formular um conjunto de ideias e recomendações para o desenvolvimento e utilização de jogos para o suporte à aprendizagem de programação.

O jogo foi desenvolvido seguindo os princípios de motores de jogos. Esta opção foi tomada para permitir que o jogo suportasse as distintas formas de organização e disponibilização das tarefas de aprendizagem experimentada durante os ciclos. Igualmente foi objetivo permitir que os professores que venham a usar o jogo em suas aulas possam definir a sua própria organização e tarefas de aprendizagem. Com isso, como um terceiro produto da tese, definimos um modelo de arquitetura construcionista de jogos sérios para aprendizagem de programação, que procura orientar e inspirar outros projetistas no desenvolvimento dos seus jogos.

Palavras-chave: Jogos sérios. Ensino de programação. Programação baseada em blocos. Jogos casuais. Motor de jogos.

ABSTRACT

Researchers who focus on introductory programming teaching and learning continue to classify and identify students' difficulties, and to try to develop methodologies and tools that can help them, especially those who are struggling to develop the required competencies.

In this work, we investigate the use of serious games to support students who begin computer programming learning. We developed a serious game, called NoBug's SnackBar, with fun elements that are not usually found in other games of a similar nature. The idea is to attract and motivate the students in the early learning stages to use the game and with that develop their programming skills. The mechanics and rules of the game are simple; therefore, it is easy to learn how it works, facilitating its introduction and integration as a tool in the introductory computer programming courses. The game includes a set of missions that can be fulfilled through small programs. A block-based approach is used to make the game independent of the computer programming language, allowing students to focus on the problems to be solved and not on the syntax of any computer programming language. The game runs in a browser, requiring no major hardware requirements or installing additional software. The teacher has an administrative area that highlights the students with more difficulties in progressing, and thus allows the teacher to monitor the evolution of each student in each of his or her attempts.

The project followed the Design-Based Research methodology, with four cycles being performed. The first one was a pilot to validate the idea. The next three were longer, each covering almost a whole semester, thus it was possible to verify significant learning changes and adherence to the game, as well as identifying changes and / or improvements that needed to be included in the game or its use. Students had the freedom to use the game when and how much they wanted. Therefore, we believe we have a more truthful assessment of the fun and its pedagogical usefulness. At the end of each cycle, some ideas were identified to be tested in the following cycle, leading to changes in the game. In each of the three cycles, we used different forms of organizing and making available the learning tasks, as well as integrating the game with the classes.

As a product of this thesis, in addition to the game, all the observations in the development cycles allowed us to formulate a set of ideas and recommendations for the development and use of games to support programming learning.

The game was developed following the principles of game engines. This option was taken to allow the game to support the different forms of organization and availability of the learning tasks experienced during the cycles. It was also intended to allow teachers who use the game in their courses to define their own organization and learning tasks. Therefore, as a third product of the thesis, we define a constructionist serious games architecture model for programming learning, which seeks to guide and inspire other designers in the development of their games.

Keywords: Serious games. Programming teaching. Block based programming. Casual games. Game engine.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AIA	App Inventor for Android
API	Application Programming Interface
APV	Ambiente de Programação Visual
CSGE	Construcionist Serious Game Engine
DBR	Design-Based Research
ES	Bacharelado em Engenharia de Software
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
IU	Interface com Usuário
JC	Jogos Casuais
JH	Jogos Hardcore
LA	Learning Analytics
ME	Mestrado em Ciências da Educação
MMORPG	Multiplayer Massive Online Role-Playing-Game
PBB	Programação Baseada em Blocos
PC	Pensamento Computacional
RPG	Role-Playing-Game
SR	Sistema de Recomendação
UC	Universidade de Coimbra
Udesc	Universidade do Estado de Santa Catarina
UFSC	Universidade Federal de Santa Catarina
XML	eXtensible Markup Language

Capítulo 1

INTRODUÇÃO

1.1 Enquadramento e Motivação

A demanda mundial por programadores é crescente. Segundo o relatório da Burning Glass Technologies (2016), nos Estados Unidos no ano de 2015 abriram-se 7 milhões de postos de trabalho para programadores, o que corresponde a 20% das carreiras que ganham pelo menos U$ 15 a hora. A tarefa de programar é essencial nas ciências da computação. A codificação deve fazer parte da alfabetização atual e ajuda a praticar as habilidades do século XXI, como a resolução de problemas, o trabalho em equipa e o pensamento analítico (European Union Digital Single Market, 2016). O principal propósito das disciplinas de programação não é o desenvolvimento e a compreensão da gramática de uma linguagem de programação, mas desenvolver as habilidades computacionais de resolução de problemas, chamado como Pensamento Computacional (PC). PC é uma abordagem para resolver problemas, projetar sistemas, e entender o comportamento humano que se baseia nos conceitos fundamentais da computação (Wing, 2006). Para atender essa necessidade da alfabetização do século XXI, 11 países da União Europeia (DK, FR, FI, HR, IT, MT, PL, TR, UK-EN, UK-SCT) concluíram recentemente uma reforma escolar em que adicionaram PC e outros conceitos relacionados em seus currículos (Bocconi et al., 2016). Investimentos são feitos nos Estados Unidos, pelo governo (White House, 2016), e por empresas privadas através da iniciativa Code.org. Em Portugal, está a decorrer um projeto-piloto de formação de professores e aprendizagem por alunos do 3º e 4º ano de projetos articulados com as demais componentes curriculares (Equipa

de Recursos e Tecnologias Educativas, 2015). No Brasil, as iniciativas ainda são isoladas e promovidas pelas Universidades (Moser, 2017; Wangenheim, 2017).

Essas são iniciativas de preparação para a geração atual de crianças e jovens. Entretanto, existem ainda alguns anos em que os iniciantes dos cursos de graduação não tiveram quaisquer contactos prévios com esses assuntos introdutórios de programação. Na educação básica os alunos aprendem paulatinamente esses conceitos. No ensino superior, o ritmo é mais intenso, o que exige mais disciplina, organização e persistência no tempo de estudo (Gomes & Mendes, 2007). A complexidade desse processo faz dele fonte de dificuldades na sua aprendizagem, e por consequência na alta desistência e reprovação nessas disciplinas nos primeiros anos dos cursos superiores (Robins et al., 2003). Em um estudo recente, Portugal e Brasil lideraram entre os três – dos quinze países da amostra – com maior índice de reprovação (62,1% e 55% respetivamente), porém, de modo geral os índices ficaram entre 30% a 40% (Watson & Li, 2014).

Esses altos índices de reprovação cunham a disciplina como difícil de aprender. Existem várias razões para os estudantes apresentarem tanta dificuldade em se qualificarem nessa disciplina, entre eles, o processo de ensino, em que muitas vezes o foco está na linguagem de programação e não nas estratégias de resolução de problemas (Robins et al., 2003), e em que se usam materiais instrucionais estáticos, o que não corresponde a natureza dinâmica inerente na aprendizagem de programação (Watson et al., 2011), a diversidade de estilos de aprendizagem dos estudantes e a natureza e dificuldade dos exercícios (Gomes & Mendes, 2015). Esses são alguns fatores que interferem na manutenção da motivação em aprender programação. As tarefas são realizadas com maior entusiasmo e empenho com o estudante motivado e, por consequência, melhora o aproveitamento e o sucesso nas suas aprendizagens (Figueiredo, 2011). Aumentar a motivação dos estudantes leva a uma aprendizagem melhor e mais eficaz para toda a turma, independente do nível de conhecimento individual, maximiza o potencial de todos (Carter et al., 2011).

Os estudantes da nova geração, habituada com jogos e outras médias eletrónicas, não sentem-se motivados com exercícios para calcular e imprimir no console, ou em uma janela, números, mas estão acostumados a consumir animações, gráficos e sons com ritmo acelerado, e provavelmente esse é um tipo de média que eles gostariam de produzir (Guzdial & Soloway, 2002). O uso de jogos vem integrando os currículos para simular atividades do mundo real (Johnson et al., 2015) e para

proverem oportunidades significativas de aprendizagem com a esperança de aumentar o interesse dos alunos nos conteúdos educacionais (Weintrop & Wilensky, 2016b). Os alunos podem aprender de forma personalizada (os jogos podem ser configurados pelo jogador) e auto assistida (os jogadores percecionam seus erros ao falharem nas missões, ou o jogo instrui como fazer determinada ação) (Prensky, 2001). A motivação é promovida pela programação de pequenos jogos como forma alternativa aos tradicionais exercícios de programação (Hijon-Neira et al., 2014; Soares et al., 2015; Black, 2016), ou o uso de jogos para reforçar conceitos, auxiliar na concretização de conceitos abstratos, encurtar o tempo entre teoria e prática e também como opção aos exercícios comuns (Lee, Bahmani, et al., 2014; López et al., 2016; Malliarakis et al., 2017).

Os jogos são experiências interativas fáceis e divertidas (Aldrich, 2009). Normalmente não exigem um treino ou ajuda externa, a não ser pelos comentários das pessoas que os apreciaram jogando (Schell, 2008). As pessoas gostam de jogar porque através dos jogos podem cumprir os seus desejos interiores (Radoff, 2011), estimulam a curiosidade pela descoberta de novos locais ou meios de interagir com as personagens e os objetos, e através da fantasia ao simplificar ou desconstruir elementos do mundo real (Malone, 1980). Os jogos sérios (ou jogos educacionais) são efetivos quando se deseja imergir o aluno em um conteúdo e provê-lo de uma experiência para ensinar conceitos e ideias (Boller & Kapp, 2017).

1.2 Objetivos do Estudo

Este trabalho de investigação procurou contribuir para melhorar as condições de aprendizagem inicial da programação. A aplicação de atividades instrucionais de PC são normalmente voltadas para o ensino básico (1º ao 12º ano) (Grover & Pea, 2013). Como o uso do PC tem o apelo da computação para todos (Wing, 2008), os esforços são voltados para a introdução desses conceitos computacionais em quaisquer cursos (Malan, n.d.). Entretanto, com o insucesso nas disciplinas introdutórias de programação em cursos de informática, poderíamos utilizar as práticas e ferramentas do PC com essas cadeiras facilitando a aprendizagem dos estudantes.

Existem ambientes educacionais que usam a abordagem do PC, como Scratch[1] e Alice[2]. Esses ambientes utilizam alguma representação gráfica na metáfora do Lego para os alunos construírem suas soluções, porém o Scratch tem uma diferenciação maior quanto ao uso de cores e formatos dos blocos conforme o tipo de instrução. O foco da tarefa do aluno é resolver os problemas, e não se preocuparem com a gramática da linguagem. Entretanto, apesar desses ambientes serem de fácil utilização e os componentes da linguagem de fácil aprendizagem, a didática a ser aplicada exige que mesmo assim o professor planeie e explique o conteúdo das aulas, e monitore a aprendizagem dos alunos com a avaliação dos exercícios. Além disso, como o objetivo da disciplina é formar solucionadores de problemas com linguagens de programação reais, ainda existe o inconveniente da migração da linguagem em blocos para a linguagem de programação textual.

A pesquisa realizada através desta tese teve como objetivos (i) identificar requisitos para que um jogo educativo promova a aprendizagem de técnicas de resolução de problemas em programação introdutória e (ii) apresentar uma metodologia para usar esse jogo integrada numa disciplina de introdução à programação. Consequentemente, a principal questão de investigação foi

> **Como se pode melhorar a aprendizagem das habilidades de resolução de problemas, em programação de computadores introdutória, usando um jogo digital?**

Para podermos explorar e avaliar as características dos jogos na aprendizagem de resolução de problemas e operacionalizar a investigação para responder a questão de pesquisa, decidimos subdividi-la em três novas perguntas. Na primeira questão desejamos reconhecer o contexto de aplicação dos jogos sérios:

> **Q1 – Quais são as razões para usar jogos sérios no ensino e aprendizagem das habilidades de resolução de problemas em disciplinas de introdução à programação de computadores?**

[1] https://scratch.mit.edu
[2] http://www.alice.org/

Ao respondê-la, teremos identificado (i) os benefícios de usar jogos; (ii) o género, mecânica e elementos essenciais de jogos melhor aplicáveis para as habilidades que se pretende desenvolver; (iii) os hábitos, motivações e perfil dos alunos quanto à adoção de jogos.

Uma vez que identificamos o género do jogo, e sua mecânica, pudemos investigar aspetos específicos para o jogo, e assim chegamos à segunda questão:

> **Q2 – Quais os recursos que devem fazer parte de um jogo sério para que promova a aprendizagem de resolução de problemas em disciplina de introdução à programação de computadores?**

A resposta a esta pergunta pôde-nos esclarecer sobre o jogo quanto (i) às atividades relacionadas às habilidades que se pretende promover nas tarefas; (ii) como podem ser recompensadas e pontuadas as tarefas; (iii) como pode evoluir o nível do jogo, e o que vai implicar nas tarefas; (iv) como perceber as dificuldades que o jogador enfrenta e fornecer pareceres e comentários para auxiliá-lo e promover o seu progresso; e (v) quais são as formas, seja através de elementos visuais ou codificação, em que o aluno fornecerá a solução.

A terceira questão diz respeito às formas de usar, aplicar e avaliar o jogo integrado no currículo da disciplina:

> **Q3 – Como pode ser integrado um jogo sério em disciplinas de introdução à programação de computadores?**

Espera-se que com a resposta desta questão possamos sugerir (i) os momentos mais propícios para uso do jogo; (ii) a sua integração com os conteúdos e exercícios tradicionais; (iii) o momento de evoluir do uso de uma linguagem gráfica livre de erros de sintaxe para uma linguagem codificada; e (iv) como integrar a avaliação do desempenho no jogo com a avaliação do desempenho da disciplina.

Em conjunto, estas três questões direcionam a resposta à questão principal, identificando porquê usar um jogo (questão 1), quais os elementos e como os jogos se devem comportar (questão 2) e como usá-los no contexto da disciplina (questão 3).

1.3 Estrutura da Tese

A tese está dividida em seis capítulos. O capítulo 1 apresenta uma breve descrição sobre a problemática a ser abordada na investigação, as questões de investigação e a estrutura da tese. Os capítulos 2 e 3 formam a base teórica para o desenvolvimento do jogo digital. O capítulo 2 relata as competências a serem apropriadas pelos alunos para se tornarem hábeis na programação, e as dificuldades para que eles as conquistem. Também apresenta algumas estratégias para a aprendizagem de programação, principalmente no que refere à abordagem de uso de linguagens com gramáticas livres de falhas, que é a ferramenta que usamos no jogo. O capítulo 3 descreve a abordagem do uso de jogos para o ensino e aprendizagem de programação, e destaca alguns conceitos que seguimos no desenvolvimento, como motores de jogos, jogos casuais e abordagem construcionista no uso de jogos. O capítulo é finalizado com um levantamento dos jogos para a aprendizagem de programação identificados em publicações e/ou disponíveis para descarga e uso. O capítulo 4 apresenta o desenvolvimento do jogo e a experiência de teste com quatro turmas. O capítulo inicia relatando o projeto inicial do jogo, relacionando os objetivos pedagógicos, o modelo e cenário inicial para o jogo. Em seguida, descreve a metodologia de experimentação adotada e depois cada um dos quatro ciclos, sempre apresentando as mudanças no jogo entre uma experiência e outra. O capítulo 5 explana os resultados da investigação, relacionando os elementos que constatamos terem relevância nas experiências, e encerra delineando um modelo de arquitetura de jogos sérios para que as lições aprendidas possam ser replicadas por outros investigadores. O último capítulo inicia retornando as questões de pesquisa e aponta suas respostas e finaliza com sugestões para continuidade da investigação.

Capítulo 2

APRENDIZAGEM DE PROGRAMAÇÃO E PENSAMENTO COMPUTACIONAL

2.1 Considerações Iniciais

Aprender programação exige determinadas atitudes dos estudantes, que podem ser melhoradas com a aplicação de vários métodos e técnicas pedagógicas e o uso de ferramentas computacionais. Há várias décadas que a comunidade científica vem desenvolvendo esses instrumentos para melhorar a aprendizagem da programação. O Pensamento Computacional (PC) é um desses instrumentos que permite tornar acessível o processo de resolver problemas com conceitos e práticas da computação. Neste capítulo serão discutidos os resultados de algumas propostas que fomentaram o desenvolvimento da nossa abordagem pedagógica quanto a aplicação do jogo em sala de aula e os requisitos para o seu desenvolvimento.

Este capítulo está estruturado em sete secções. A segunda secção apresenta o conjunto de competências e habilidades necessárias para se tornar um bom programador, relacionando-as com os conceitos de PC dissertados na terceira secção. Esse assunto de PC é descrito com foco no domínio das habilidades para torná-lo pedagogicamente operacional. Na quarta secção são enumeradas algumas dificuldades na aprendizagem da programação. Na quinta secção é descrita a programação baseada em blocos, recurso muito comum como estratégia para praticar o PC, e são listados alguns ambientes de programação visual, que surgem como uma alternativa para motivar os estudantes e impulsionar a sua aprendizagem nas

primeiras semanas. Na sexta secção são abordadas algumas estratégias de ensino-aprendizagem usando esses ambientes em programação. A última secção apresenta um resumo que destaca os principais pontos deste capítulo.

2.2 Competências e Capacidades em Programação

A programação não é simplesmente um conjunto de conhecimentos, mas uma competência (Jenkins, 2002). Para dominá-la é necessário tanto recordar os detalhes da sintaxe da linguagem de programação usada, como a aplicação dessa linguagem para produzir uma solução funcional para problemas concretos. Essa aplicação depende das estratégias de resolução de problemas baseando-se em capacidades de compreensão, codificação e depuração de programas (Pea et al., 1987; Winslow, 1996; Robins et al., 2003; Lahtinen et al., 2005; Fuller et al., 2007). A compreensão de programas envolve a leitura, interpretação e manutenção de produtos existentes. Para a geração de programas, são necessárias capacidades de projeto e criação de novos produtos. A depuração exige rever e compreender o código, para então se efetuarem as modificações necessárias. Elas fazem parte de uma hierarquia de habilidades, onde o estudante aprende gradualmente a partir da sintaxe, passando para a semântica, a estrutura e o estilo de codificação (Jenkins, 2002).

A **compreensão** do funcionamento de um programa requer que o aluno primeiramente esteja familiarizado com a forma como o computador se comporta em relação ao programa. Os alunos principiantes têm problemas em abstrair a noção de como a máquina interpreta um programa (Du Boulay, 1986; Pea et al., 1987). As máquinas imaginárias são abstrações projetadas para fornecer um modelo que auxilia na compreensão de uma construção particular de uma linguagem ou a execução de um programa (Berry & Kölling, 2014). Elas apresentam um alto nível conceitual ao prover uma camada metafórica sobre as máquinas reais e assim se espera que sejam mais fáceis de serem compreendidas pelos alunos iniciantes. Ao habituar-se às noções de funcionamento da máquina, o aluno pode então conhecer a notação (sintaxe e semântica) de uma linguagem de programação (Winslow, 1996; Mannila et al., 2006; Rosbach & Bagge, 2013), o que lhe permitirá ler e entender o código. Saber ler um código é tão importante como saber escrever (Striewe & Goedicke, 2014): serve

tanto para aprender com exemplos, como entender e descrever o seu próprio código, e o dos outros quando trabalhar em equipas, para futuras manutenções.

A **escrita**, ou geração de código, é quando o sujeito cria uma parte ou um programa inteiro para resolver um problema (Robins et al., 2003). O aluno, munido de ferramentas (linguagem de programação e representações visuais) e de boas práticas, cria os seus programas. É o momento de treinar a transformação das soluções mentais em programas de computador (Winslow, 1996). A sua tarefa não é escrever programas simples, mas resolver problemas. Para isso o aluno precisa de desenvolver habilidades ou capacidades essenciais como abstração, generalização, decomposição e estratégias de resolução de problemas. Escrever programas envolve compreender o problema e abstrair um modelo, dividir o problema em partes menores, decidir pela melhor estratégia para resolver cada uma das partes, e aplicar ou adaptar soluções já conhecidas. O aluno não consegue aprender essas capacidades sem realmente fazê-lo e praticar muito (Winslow, 1996; Lahtinen et al., 2005). Na literatura é descrita essa competência como a principal fonte de problemas dos alunos (Du Boulay, 1986; Pea et al., 1987; Robins et al., 2003). Os alunos demonstram conhecimento da sintaxe e semântica da linguagem, mas apresentam deficiências em estratégias de resolução de problemas e planeamento insuficiente (Caspersen & Bennedsen, 2007).

Uma situação natural quando se desenvolvem programas é a ocorrência de erros que fazem com que o resultado do programa não seja conforme o previsto. Consequentemente, os programadores são frequentemente defrontados com a necessidade de resolver erros de código ou até de projeto (Robins et al., 2003). A atividade de **depuração** requer a competência de ler e entender o código, capacidades de teste, identificação dos erros e refatoração do código (Du Boulay, 1986; Winslow, 1996). Nessa tarefa, os alunos podem usar ferramentas de suporte para criar artefactos adicionais tais como visualizações e rastreamentos, que também necessitam de ser compreendidos pelo estudante (Striewe & Goedicke, 2014). Se eles não dominarem esta capacidade, podem adicionar novos erros num programa (Murphy et al., 2008). O sucesso na depuração de código depende mais de um aluno com melhor capacidade de compreensão do que de escrita: os programadores sentem maior confiança na escrita quando têm maior domínio em capacidades de depuração (Ahmadzadeh et al., 2005; Lopez et al., 2008; Lister et al., 2009).

2.3 Pensamento Computacional

O uso dos conceitos, métodos e ferramentas da computação podem transformar o comportamento de todas as disciplinas, profissões e setores (Wing & Stanzione, 2016). O Pensamento Computacional (PC) é um conjunto de capacidades fundamentais usadas por toda a gente, como a escrita, a leitura e a aritmética, que serve para formular e resolver problemas de forma algorítmica – como uma série ordenada de passos – e usando representações abstratas de dados, que possam ser executadas por um agente de processamento da informação (Wing, 2010; Barr et al., 2011). Esse agente pode tanto ser um humano como um computador, ou até ambos. Resolver problemas com os conceitos do PC fornece familiaridade com constructos algorítmicos, como as estruturas básicas de controlo e, por isso, esforços devem ser feitos para que os alunos tenham contacto com o PC antes de experimentarem a sua primeira linguagem de programação (Lu & Fletcher, 2009). A programação concretiza os conceitos do PC e este vem como uma ferramenta para a aprendizagem da programação (Bocconi et al., 2016).

O processo de PC é composto essencialmente por quatro passos: decomposição, padrões, abstração e algoritmo (Google, n.d.). Entretanto, outros conceitos estão relacionados com esses passos, como a automação, simulação, paralelização, recolha, análise e representação de dados.

Decomposição é a tarefa de dividir um problema em pequenas partes muito mais simples de entender, resolver, desenvolver e avaliar (Curzon et al., 2014). Esse passo faz com que sistemas complexos sejam mais fáceis de serem resolvidos, situações novas sejam melhor entendidas e grandes sistemas sejam mais fáceis de projetar (Csizmadia et al., 2015). Essa habilidade é reconhecida como a mais difícil de dominar, e os alunos somente conseguem fazê-la perfeitamente quando entendem muito bem o problema (Selby, 2015). Essa tarefa utiliza duas ferramentas: a recolha de dados e a simulação. A **recolha de dados**, através da observação e mensuração, é responsável pela identificação, compreensão e aprofundamento do problema para dar suporte ao processo (Weintrop et al., 2016). A **simulação** do problema, o imitar processos reais, permite explicar os eventos e situações do problema, assim como instrumentaliza a recolha dos dados (Shodiev, 2015).

É possível **reconhecer padrões** nos processos e dados dessas partes, ou seja, situações recorrentes no novo problema ou em anteriores (Chande, 2015). A **análise de dados** é a ferramenta utilizada para identificar esses padrões, através da categorização, identificação de tendência e correlações (Weintrop et al., 2016). Uma vez identificadas essas recorrências, podem ser reconhecidos os seus atributos e características para a criação de modelos, regras, princípios, e assim **generalizar os padrões** (Csizmadia et al., 2015). A generalização permite que se expresse a solução do problema em termos gerais, permitindo que se possa aplicar a diferentes problemas que compartilham as mesmas características (Selby, 2014). Visando facilitar o trabalho no próximo passo do processo, como esses termos detetados durante a generalização se transformarão em dados, é preciso atenção para que eles estejam bem **representados** (como gráficos, imagens ou textos) (Csizmadia et al., 2015).

A parte mais importante do processo é a **abstração** (Wing, 2010). Ela serve para identificar propriedades importantes do problema, ignorando os seus detalhes irrelevantes, para identificar a ideia principal, e poder ser reutilizada noutras situações (Wing, 2008).

Com todas as partes devidamente identificadas e relacionadas, é possível começar a **projetar o algoritmo**, ou seja, escrever a solução de uma forma que possa ser executada passo-a-passo para se atingir os objetivos (Curzon et al., 2014). A **automação** é a execução dessa solução por máquinas ou computadores, é a computação da abstração (Wing, 2008). Por vezes é possível a aplicação da **paralelização** que é a execução simultânea de pequenas partes para atingir um objetivo comum, em que se obtém um melhor desempenho na automação (Cesar et al., 2017).

2.4 Dificuldades na Aprendizagem da Programação

Diagnosticar os motivos que levam os alunos ao fracasso na aprendizagem da programação é uma tarefa primordial para sugerir mudanças na forma de ensinar, praticar e avaliar, ou seja, propor estratégias para melhorar o cenário da alta reprovação e desistência nas disciplinas introdutórias. Por exemplo, o maior problema

do aluno é a capacidade de resolver problemas, pois ele acaba por usar muitas vezes a mesma estratégia independentemente do problema, ou nem sabe como começar (Winslow, 1996; Lahtinen et al., 2005; Pears et al., 2007; Piteira & Costa, 2013). Mais de metade dos alunos não entendem que resolver problemas e pensar logicamente fazem parte do aprender a programar, e muitos deles não vislumbram a verdadeira característica do que é um programa correto (Stamouli & Huggard, 2006). Du Boulay (1986) identificou três principais dificuldades na aprendizagem da programação:

1. Entendimento incorreto de como funciona um computador (o aluno acredita que o computador entenderá uma expressão que possa ser malformada), e da execução passo-a-passo de um programa (o aluno acredita que o computador subentende o que precisa ser feito, mesmo com um conjunto incompleto de instruções);
2. Uso de analogias inapropriadas para atribuições, variáveis e *arrays*; por exemplo, uma variável mantém armazenado todos os seus valores, ou uma atribuição de variável para variável remove o conteúdo da origem;
3. Compreensão insuficiente sobre o uso e aplicação das estruturas de controlo, falhando o código quando aninhado nessas estruturas, ou mesmo após as mesmas. Pode conseguir resolver pequenas partes, mas quando junta o todo, não consegue encaixar os trechos de código.

Existem alguns poucos erros pontuais de programação que ocorrem com frequência – como a definição dos limites nas instruções condicionais e nos ciclos (Sirkiä & Sorva, 2012; Rosbach & Bagge, 2013). Contudo, a maioria desses erros ocorre porque os alunos não tem a compreensão completa da semântica de algumas construções das linguagens (Spohrer & Soloway, 1986) e, por consequência, não conseguem identificar onde estão esses erros (Lahtinen et al., 2005; Kinnunen & Malmi, 2006). Ainda, para encontrar e resolver esses erros muitos alunos não tentam entender o código antes de começar a resolver o erro e limitam-se a fazer testes com as entradas da especificação do exercício (Murphy et al., 2008).

Rosbach & Bagge (2013) classificaram os problemas dos alunos em duas categorias: problemas baseados em conceitos e problemas baseados em conhecimento. A primeira categoria incluiu erros que geralmente o estudante consegue resolver com a combinação de leitura, execução passo-a-passo, e observação dos resultados no final da execução do programa. Existem quatro causas para esse tipo de problema, duas delas já mencionadas anteriormente: (1) incorreta

compreensão de como funciona um computador (conforme afirmado por Du Boulay); (2) dados insuficientes para testar (corroborado por Murphy et al.), ou falta de imaginação e habilidade para fazê-lo, e assim não abranger consideravelmente todos os caminhos do seu programa; (3) a tradução da solução de uma linguagem natural para a linguagem de programação; (4) a aplicação de um padrão de solução para o seu problema específico, por exemplo, reutilizar um ciclo onde aprendeu a calcular a média para realizar outros tipos de cálculos. A outra categoria representa a falta de conhecimento ou compreensão para resolver o problema. As dificuldades nessa categoria são (1) a incorreta ou incompleta interpretação do enunciado, não conseguindo extrair todos os requisitos do problema; (2) falta de conhecimento das construções e recursos da linguagem de programação; (3) dificuldades em expressar suas intenções usando as partes relevantes do programa, ao reutilizar recursos já existentes nele, como as variáveis e as funções ou métodos.

O caminho da aprendizagem da programação não é uma progressão linear, a sequência dos tópicos não serve para todos da mesma forma e consome-se muito tempo (Eckerdal et al., 2007). O tempo é um elemento natural da aprendizagem, e para cumprir o tempo fixo de uma disciplina, o aluno precisa de um atendimento individualizado com exercícios, tarefas e atividades que vão ao encontro das suas deficiências.

Os alunos podem desanimar quando consecutivamente não conseguem vencer as suas dificuldades. A disciplina de programação exige muito mais trabalho e tempo do que as demais disciplinas (Kinnunen & Malmi, 2006). Existe uma grande relação entre a motivação do estudante e o seu sucesso na aprendizagem de programação (Carter et al., 2011). Quando um aluno quer aprender a fazer algo, os obstáculos não são tão intransponíveis assim. A motivação pode ser influenciada pelo material instrucional, por exercícios divertidos e úteis, pela interação com os colegas e professores, e pela visibilidade do progresso do estudante (Kelleher & Pausch, 2005; Settle et al., 2014). Por outro lado, desmotivar-se-á se lhe forem fornecidos problemas para os quais ele ainda não possui as habilidades necessárias para os resolver (Gomes & Mendes, 2007). Existem várias alternativas descritas na literatura, por exemplo, suporte individualizado pelos professores, seja pessoalmente ou replicada sua presença através de ambientes inteligentes (Piech et al., 2012) e contextos pedagógicos significativos, focando os exercícios de programação nas áreas de

interesse dos alunos (Gomes & Mendes, 2015). Na próxima secção abordaremos outra alternativa que é o uso de ambientes de programação visual.

2.5 Programação Baseada em Blocos

Conforme Xinogalos et al. (2017), um ambiente ideal para programação introdutória deve proporcionar uma interface simples que suporte a visualização dos objetos, que tenha um editor baseado em blocos, relate as mensagens de erro de forma simples e instrua como corrigi-las, e que tenha a capacidade de executar passo-a-passo um programa. Os Ambientes de Programação Visual (APV) apresentam a execução dos programas como uma animação (Sorva et al., 2013) e usam notações gráficas para a produção das soluções (Ben-Ari, 2013), por exemplo com uma abordagem de Programação Baseada em Blocos (PBB) (Price & Barnes, 2015; Weintrop & Wilensky, 2015). Esses ambientes facilitam ao estudante a compreensão da noção de máquina (observam a execução passo-a-passo e a mudança de estados das variáveis/objetos), e permitem-lhe concentrar-se em resolver os problemas e não na sintaxe da solução (previnem erros de sintaxe). Existem jogos que também usam a abordagem da PBB. Enquanto nos jogos o aluno é guiado a resolver um problema específico, os APV são abertos para produção de animações, estórias e pequenos jogos. Os APV possibilitam a concretização e instrumentalização do PC ajustável ao currículo.

A PBB popularizou-se com o uso de APV como o Scratch. Nesses ambientes as ações, manipulação de variáveis, e estruturas de controlo são representadas através de blocos coloridos que se encaixam, como uma metáfora do Lego (Kelleher & Pausch, 2005). Os estudantes preferem aprender com artefactos computacionais em que conseguem ver algum valor, que sejam interessantes e relevantes (Guzdial, 2004). Essa abordagem tem vindo a ser utilizada para introduzir a programação aos estudantes pela capacidade de mapear visualmente conceitos complexos, a acessibilidade de todos os comandos disponíveis, a facilidade de uso com o sistema de arrastar-e-soltar, e a descrição dos blocos com linguagem natural (DiSalvo, 2014; Weintrop & Wilensky, 2015). Com esses recursos o estudante foca-se na lógica e na

estrutura da programação em vez de nas mecânicas que envolvem a escrita de programas (Kelleher & Pausch, 2005).

O estudante constrói o seu programa unindo os blocos e bastando utilizar um rato. Os blocos são coloridos de acordo com a sua funcionalidade, e tem um formato visual que sugere como podem ser conectados dentro do programa e os que podem ser conectados com eles. A Figura 2.1 ilustra um pedaço de programa em Scratch que usa uma estrutura de repetição aninhando uma estrutura de condição. No exemplo pode-se observar: (1) as estruturas (*repete* e *se*) podem conter um conjunto de outros blocos; (2) na definição do condicional é exigido um tipo de bloco que retorna valores verdadeiro ou falso (losango); (3) existe uma espécie de dente abaixo de alguns blocos e uma ranhura acima, indicando quando são aceites blocos conectados posterior ou anteriormente. Por exemplo, o bloco de repetição perpétuo (*repete para sempre*) não aceita blocos posteriores.

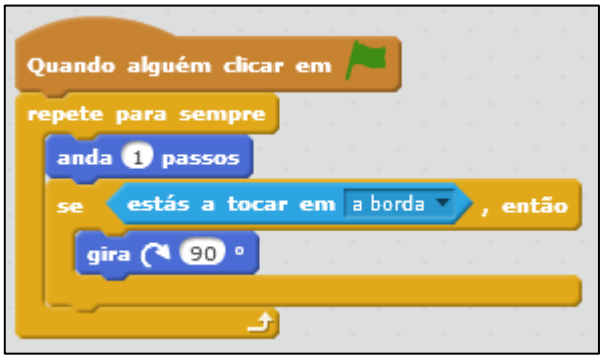


Figura 2.1 – Exemplo de um programa com PBB

Com as linguagens textuais, o professor necessita de discutir primeiro a sintaxe, os recursos da linguagem, a compilação e a execução, enquanto com a PBB os alunos podem começar logo a produzir programas (Weintrop & Wilensky, 2016a). Isso permite que o professor gaste menos tempo com explicações para toda a sala, e mais tempo com atenção individualizada, principalmente naqueles alunos que mais precisam do seu apoio. Entretanto, alguns professores têm o sentimento de falta de controlo sobre esses tipos de ambientes educacionais e são relutantes na sua adoção (Levy & Ben-Ari, 2008). Um dos motivos é a descentralização do conhecimento, pois esses ambientes promovem a aprendizagem sem a intervenção do professor. Outro é a incompatibilidade do estilo didático-pedagógico do professor. Finalmente, os professores costumam aprender esses ambientes autonomamente, e por vezes não têm confiança suficiente para usá-los com os seus alunos.

Em virtude dos ambientes com PBB serem de natureza pedagógica, algumas desvantagens foram sentidas pelos alunos mais avançados, quando comparado com a codificação textual (Weintrop & Wilensky, 2015). As modificações em programas com blocos são mais difíceis de ser feitas (Green & Petre, 1996). O uso da PBB é reconhecido como distante das ferramentas e práticas profissionais em relação ao modelo convencional (DiSalvo, 2014), arrastar-e-soltar leva mais tempo do que digitar (Schanzer et al., 2015) e, como os alunos estão limitados a seguir os templates dos blocos, muitas vezes uma declaração torna-se incompreensível pelo encadeamento de tantos blocos (Weintrop & Wilensky, 2015).

Apesar dessas desvantagens, mais alunos resolveram problemas usando blocos do que texto num estudo de Price e Barnes (2015), e conseguem envolver-se mais porque a produção de jogos e animações é um processo divertido (Armoni et al., 2015). Como consequência, praticam com mais frequência e naturalmente os princípios do PC, e assim apropriam-se das bases para a produção de programas.

Acompanhando a tendência no crescimento da oferta, pesquisa e desenvolvimento de APV, existem algumas bibliotecas que facilitam a implementação de novos ambientes baseados em blocos como Blockly[3], OpenBlocks[4] e Waterbear[5]. Blockly e Waterbear são bibliotecas em JavaScript que permitem adicionar um editor de blocos em aplicações web. O projeto Blockly foi iniciado pela Google com o intuito de migrar o App Inventor for Android (AIA) – será descrito na secção 2.5.3 – de Java para JavaScript (Ibanez, 2015). Atualmente, tanto o AIA quanto o Code Studio – será descrito na secção 2.5.7 – usam o Blockly para a edição de blocos. OpenBlocks é uma biblioteca em Java para criar aplicações gráficas com o objetivo de tornar extensível o editor de blocos do StarLogo TNG (Roque, 2007) – será descrito na secção 2.5.6. A Figura 2.2 ilustra a aparência dos blocos e menus das três bibliotecas.

[3] https://developers.google.com/blockly/
[4] http://web.mit.edu/mitstep/openblocks.html
[5] http://waterbearlang.com/

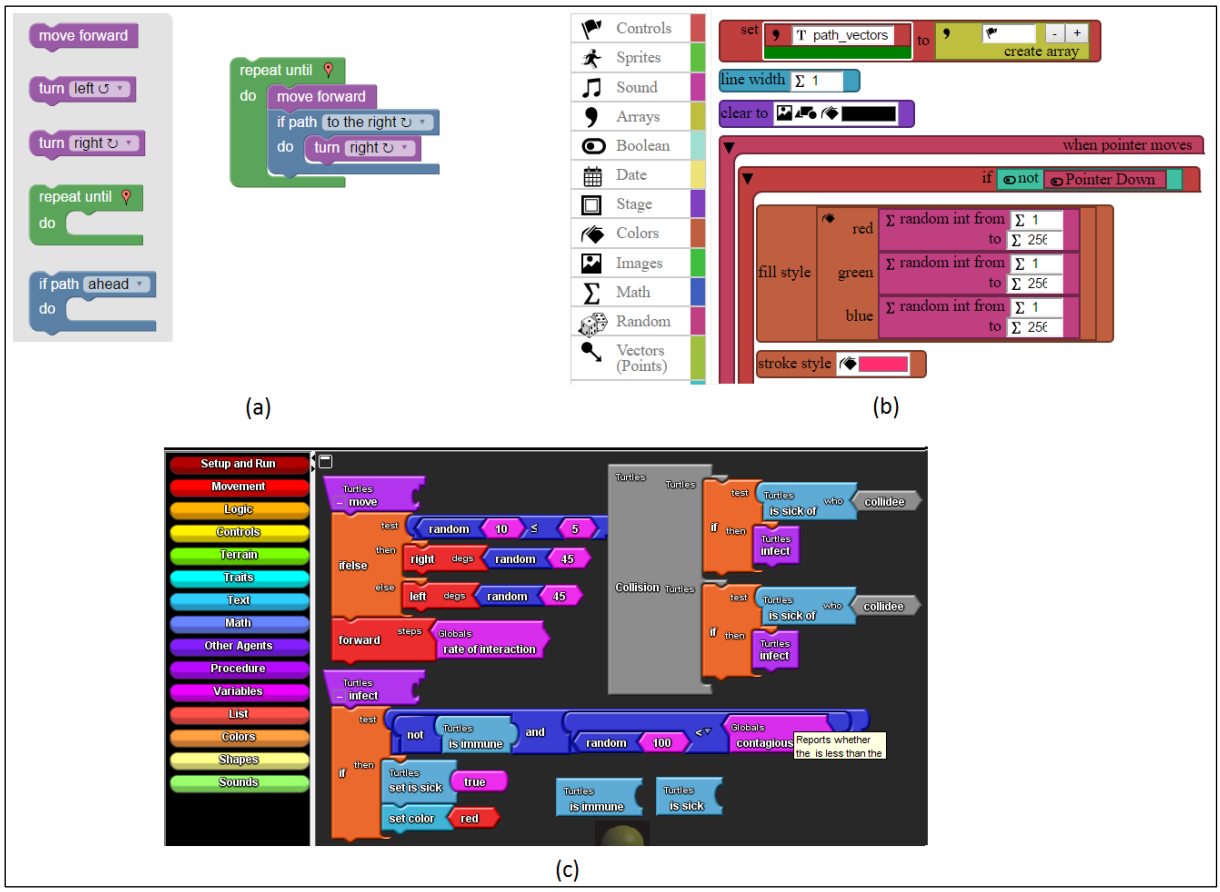


Figura 2.2 – Aparência dos blocos e menus (a) Blockly; (b) Waterbear e (c) OpenBlocks

Nas subsecções seguintes serão apresentados alguns APV mais conhecidos e descritos na literatura. Consideramos apenas os ambientes em que se pode produzir livremente programas utilizando como linguagem uma representação por blocos. Os jogos são apresentados no capítulo três. A utilização (prática pedagógica) de APV com alunos de curso superior será descrita na secção 2.6.

2.5.1 Scratch

Scratch[6] é um dos APV mais conhecidos. Em janeiro de 2017, o Scratch estava na 23ª posição no TIOBE[7]. A versão 2 em modo *offline* está disponível como código aberto. É executado num navegador e permite rapidamente desenvolver aplicações 2D baseadas em eventos através de processos assíncronos concorrentes (Wolz et al., 2009). Os alunos criam e manipulam *sprites* (personagens gráficos), adicionam música, animação e interatividade. Um projeto no Scratch consiste numa cena fixa e

[6] https://scratch.mit.edu
[7] http://www.tiobe.com/ é uma tabela de classificação das linguagens de programação mais populares.

num conjunto de *sprites* móveis. Cada objeto contém o seu próprio conjunto de imagens, sons, variáveis e *scripts*. Isso permite facilmente intercambiar *sprites* entre projetos. Além disso, tem elementos sociais no sentido em que no seu *website* os utilizadores conseguem enviar, compartilhar e alterar programas entre si (Price & Barnes, 2015).

A Figura 2.3 ilustra o ambiente do Scratch. A área (1) representa a parte de animação que pode ser gerida clicando na bandeira para iniciar e no botão vermelho para interromper. A área (2) contém a lista de objetos do programa e o pano de fundo. Ao clicar num desses elementos, na área (4) são apresentados os seus *scripts*. A área (3) contém todos os comandos disponíveis para a programação.

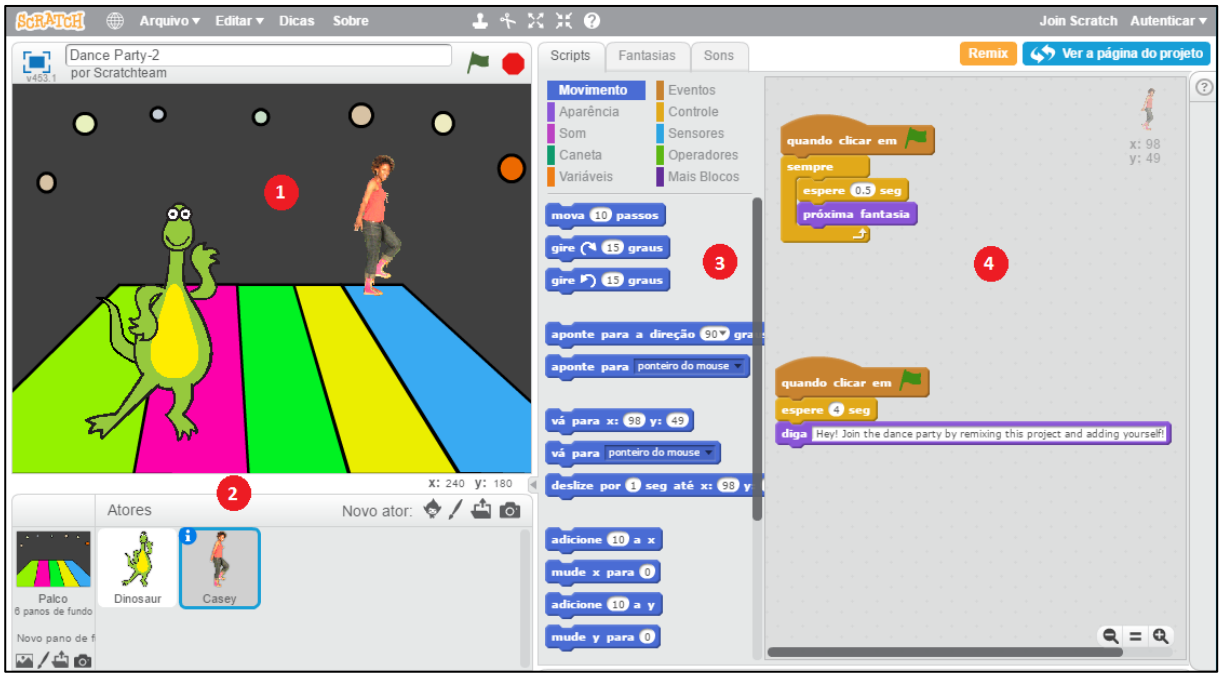


Figura 2.3 – Ambiente de programação do Scratch

2.5.2 Snap!

Snap![8] é uma reimplementação, em código aberto, do Scratch com uma aproximação para as linguagens de programação convencionais, tornando-a mais apropriada para a aprendizagem de conceitos de computação no ensino superior. Inicialmente o Snap! cobria alguns recursos inexistentes antes da versão 2 do Scratch, como a criação de novos blocos e manipulação de listas. Entretanto, o Snap! ainda permanece com algum diferencial como a possibilidade de criar funções

[8] https://snap.berkeley.edu/

(procedimentos com retorno), passar como parâmetro um conjunto de blocos (Figura 2.4), além de recursos para definir classes, instâncias e usar herança (Harvey & Mönig, 2010).

O ambiente de programação (Figura 2.5) é muito similar ao Scratch, com as mesmas quatro áreas já explicadas anteriormente.

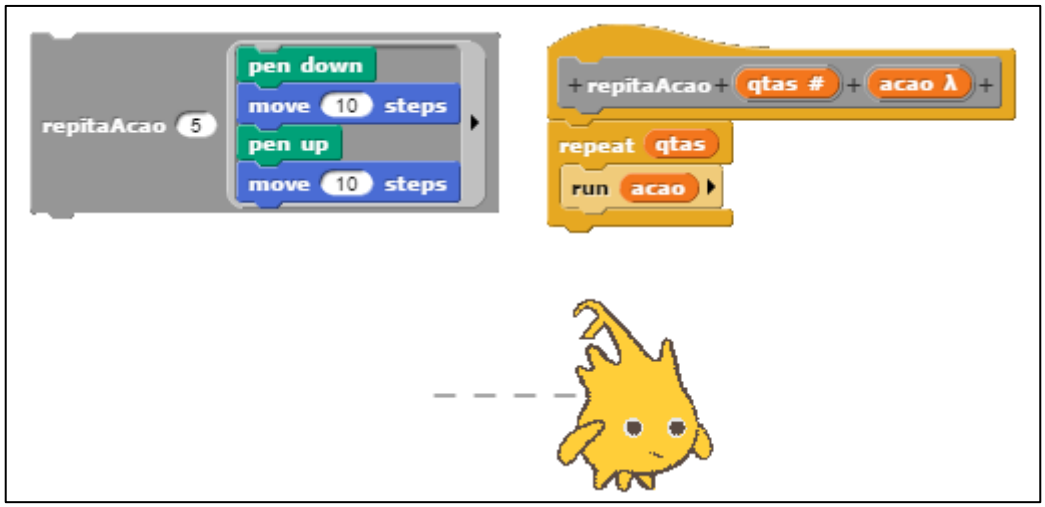


Figura 2.4 – Exemplo de passagem de um conjunto de blocos como parâmetro

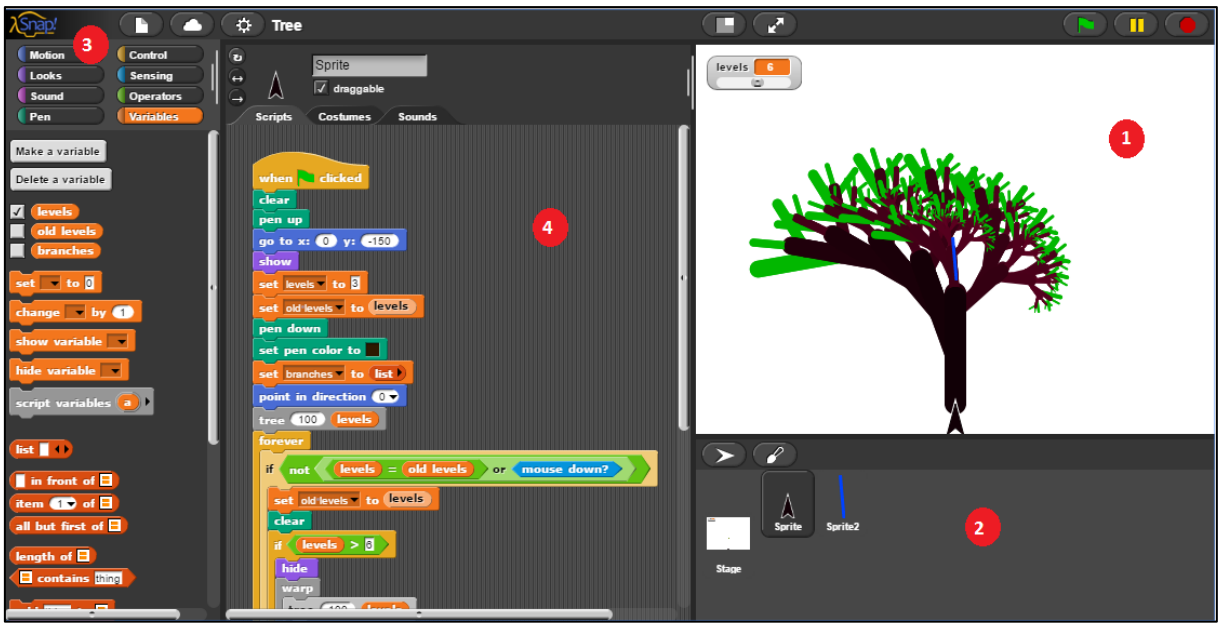


Figura 2.5 – Ambiente de programação do Snap!

2.5.3 App Inventor for Android

App Inventor for Android[9] (AIA) é um APV web de código aberto destinado ao desenvolvimento de aplicações para a plataforma Android. Possui duas abas: uma para o desenho da interface com o utilizador e outra para programação. Na aba de desenho (Figura 2.6) o aluno projeta a interface arrastando os componentes para o ecrã virtual do dispositivo. Na aba de codificação (Figura 2.7) o aluno programa os eventos dos componentes. Para testar o programa pode ser usado um emulador ou conectar-se ao dispositivo do aluno.

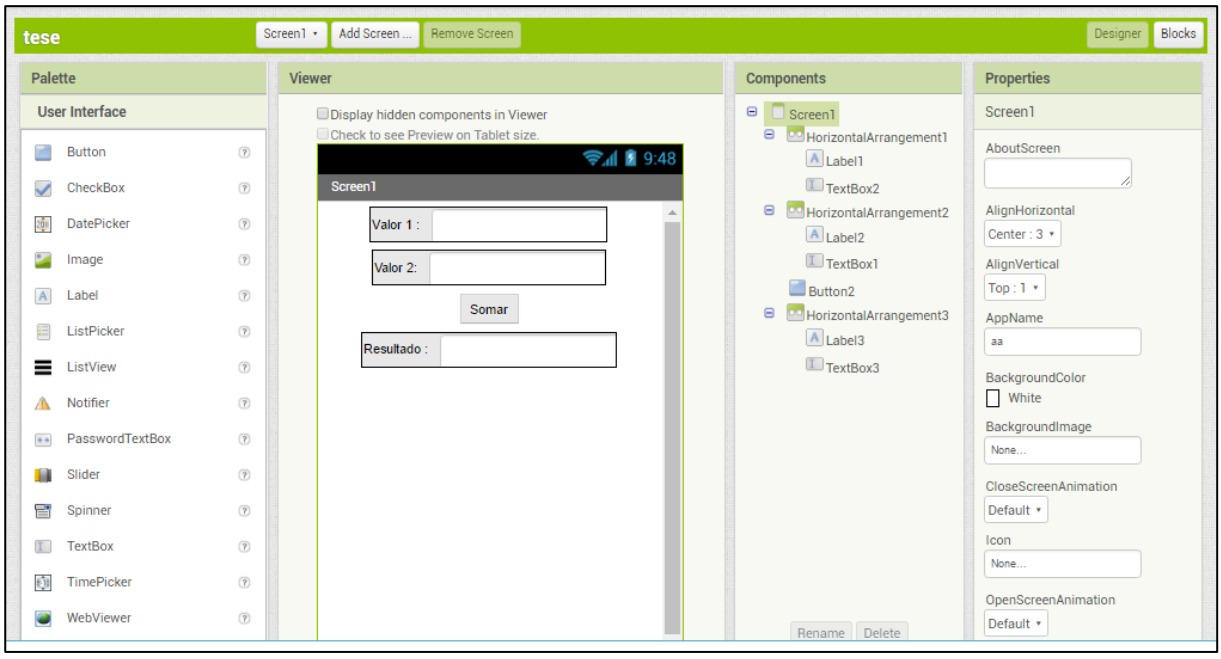


Figura 2.6 – Ambiente de programação do App Inventor: aba de projeto da interface

[9] http://appinventor.mit.edu/

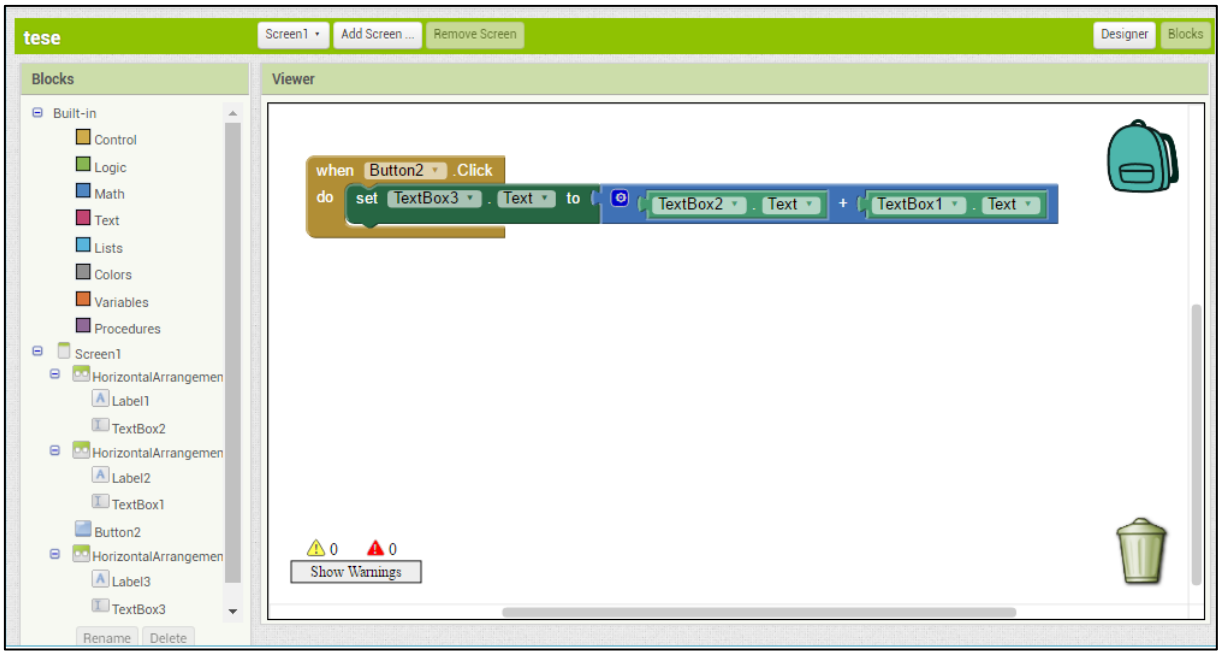


Figura 2.7 – Ambiente de programação do App Inventor: aba de programação

2.5.4 Pencil Code

Pencil Code[10] é um APV de código aberto baseado no Logo, ou seja, o aluno programa os movimentos de uma tartaruga. O ambiente permite alternar entre a programação com blocos ou textual com Coffeescript ou JavaScript, o que pode amenizar a dificuldade com a transição entre blocos e a sintaxe de uma linguagem de programação (Bau et al., 2015). Além disso, possui um modo interativo onde o aluno digita um comando de cada vez para ser interpretado. O ambiente de programação (Figura 2.8) é dividido em três partes: à esquerda os blocos disponíveis, no centro a área de codificação e à direita a de animação.

[10] http://pencilcode.net

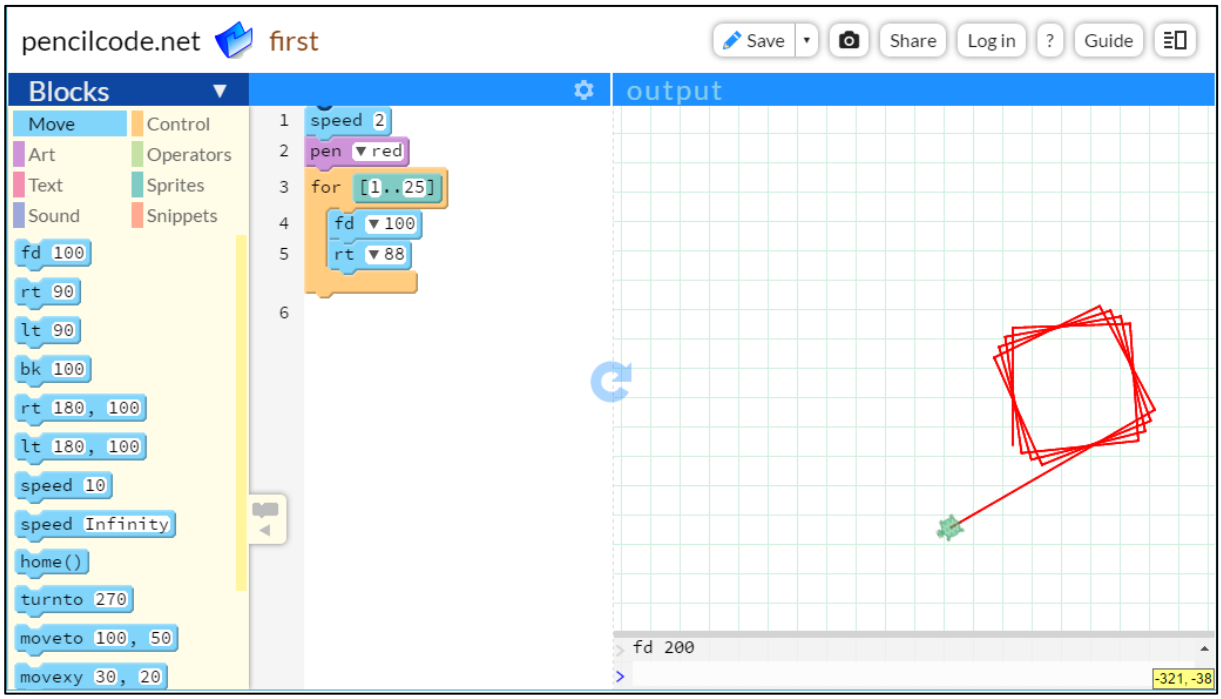


Figura 2.8 – Ambiente de programação do Pencil Code

2.5.5 Alice 3

Alice 3 é um APV que é necessário descarregar[11] e instalar no computador. A programação no ambiente segue a abordagem Orientada a Objetos. A linguagem Alice encontrava-se em janeiro de 2017 na 49ª posição do TIOBE. É um ambiente muito rico quanto à qualidade gráfica e quantidade de classes que podem ser usadas para adicionar personagens e objetos nas cenas. Implementa um sistema baseado em eventos, que permite aos alunos adicionar objetos na cena, manipular os seus atributos e invocar os seus métodos. Os objetos e o cenário são numa perspetiva 3D. O objeto principal é o cenário e todos os demais objetos são instanciados e guardados como atributos desse objeto. Ao executar o programa, são disparados os eventos de inicialização desse objeto cenário. A Figura 2.9 ilustra o ambiente de programação: a área (1) é o cenário onde podem ser selecionados e reposicionados os objetos, e ao clicar no botão [Run] observar-se-á a animação; a caixa (2) é outra forma de selecionar o objeto, e uma vez selecionado são apresentados todos os métodos da classe na área (3); por fim, na área (4) são codificados os métodos. Pode-se observar que os blocos não usam cores e formatos diferentes, como nos APV apresentados anteriormente, mas as estruturas de controlo são representadas com linguagem

[11] http://www.alice.org/

natural. Existem recursos no ambiente para definir a sintaxe dos blocos seguindo Java ou Alice, e ainda apresentar o código em Java.

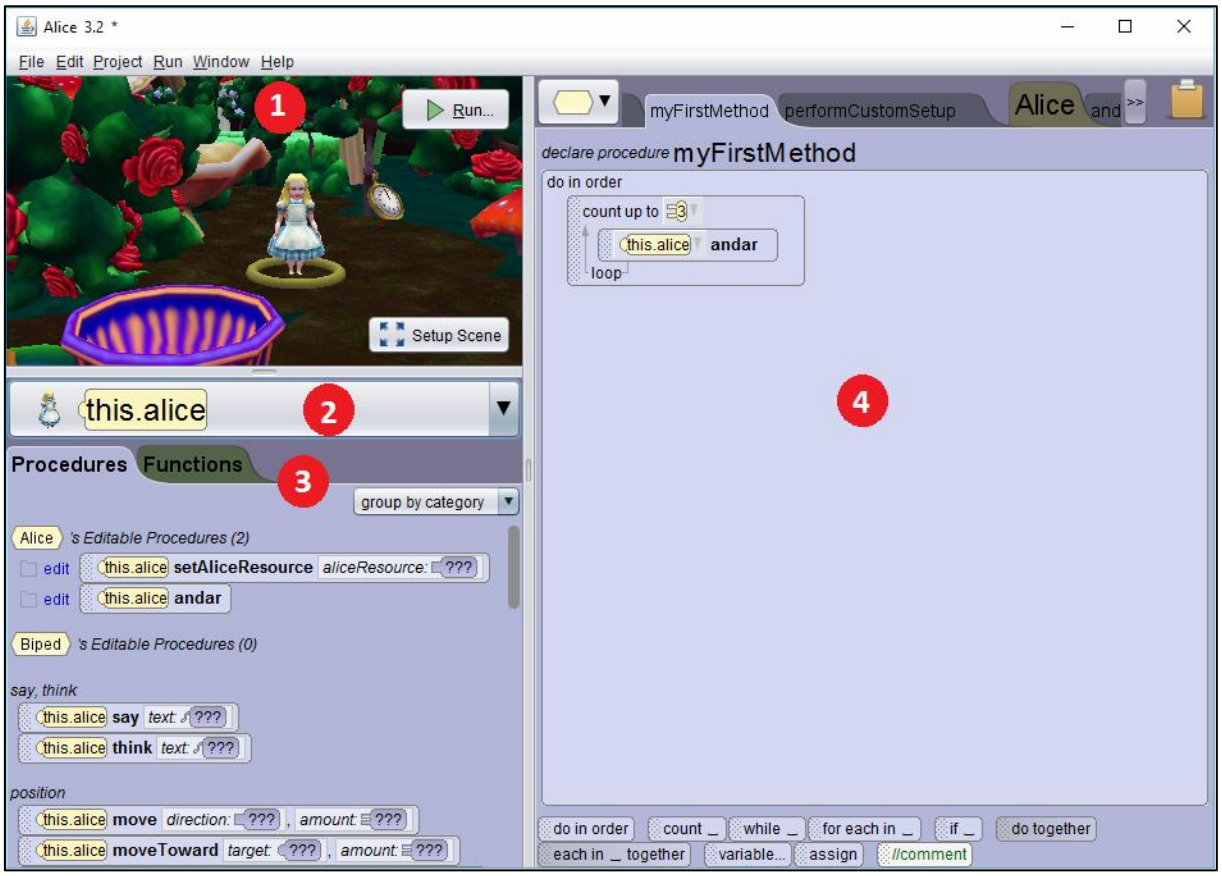


Figura 2.9 – Ambiente de programação do Alice 3

2.5.6 StarLogo TNG

StarLogo TNG é outro APV que necessita de ser descarregado[12] e instalado onde o aluno também manipula objetos numa perspetiva 3D. O ambiente foi criado para desenvolvimento de jogos 3D do tipo plataforma (p.e., Super Mario) (Klopfer & Begel, 2007). O ambiente (Figura 2.10) está organizado em duas janelas: uma para edição e outra para visualização. Cada tipo de objeto tem uma área (1) no editor para programar os seus comportamentos. Os objetos podem invocar os métodos de outros objetos ou aceder aos seus atributos. Uma pequena janela (2) permite aceder rapidamente a cada uma dessas áreas. Além das áreas de cada objeto, ainda existe uma para definir comportamento comum para todos, outra para definir como será a

[12] http://education.mit.edu/portfolio_page/starlogo-tng/

inicialização do cenário, outra para execução comum e finalmente uma outra para o tratamento de colisões.

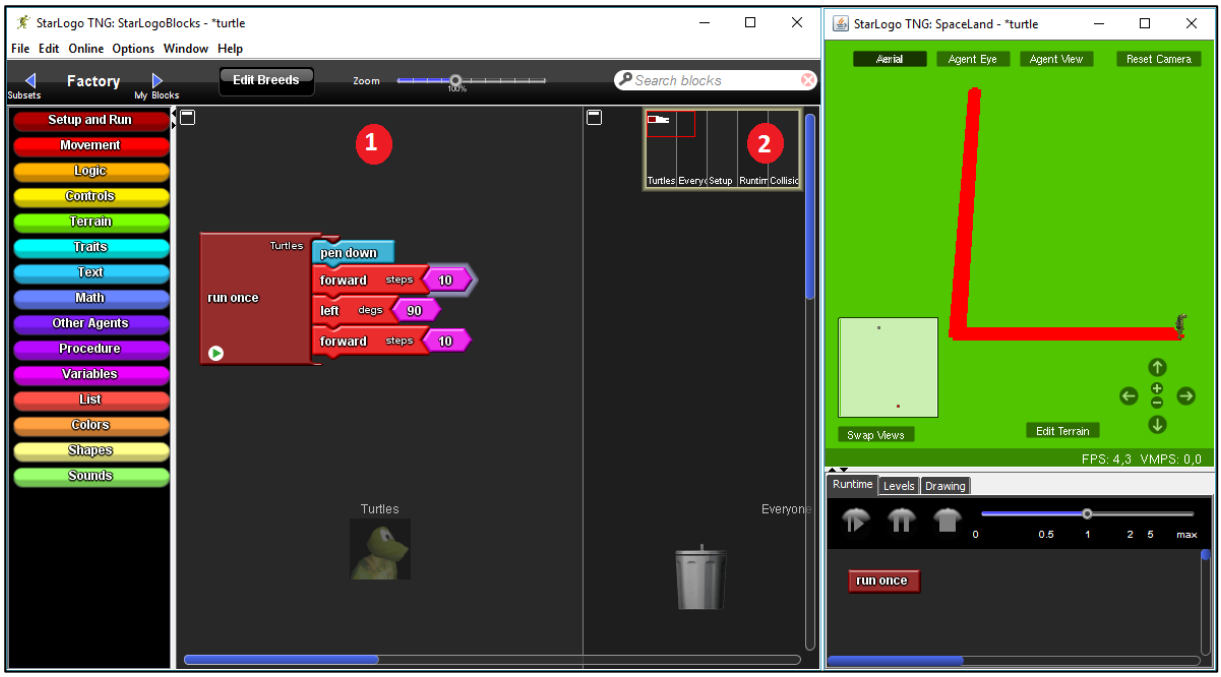


Figura 2.10 – Ambiente de programação do StarLogo TNG

2.5.7 Code Studio

Code Studio[13] é um APV de código aberto, com aprendizagem guiada, financiado pela iniciativa Code.org iniciada em 2013, patrocinada por mais de 200 entidades, e é destinado a promover a aprendizagem da programação em massa para crianças e jovens antes do ensino superior (Wilson, 2015; Wing & Stanzione, 2016). A iniciativa promove oficinas para que os professores introduzam atividades com o ambiente nas suas aulas. No ambiente são oferecidos uma série de cursos, de acordo com a faixa etária. A Figura 2.11 ilustra um curso de Ciência da Computação acelerado para a faixa de 10 a 18 anos. Basicamente, cada assunto é composto de duas fases: a primeira é uma atividade passiva, onde o aluno assiste a uma vídeo-aula, e a segunda consiste em resolver um conjunto de missões, que corresponde à parte prática do tópico. O ambiente é livre para que o aluno possa escolher a atividade que deseja fazer ou a missão que deseja jogar, independentemente de ter concluído ou não as tarefas anteriores.

[13] https://studio.code.org/

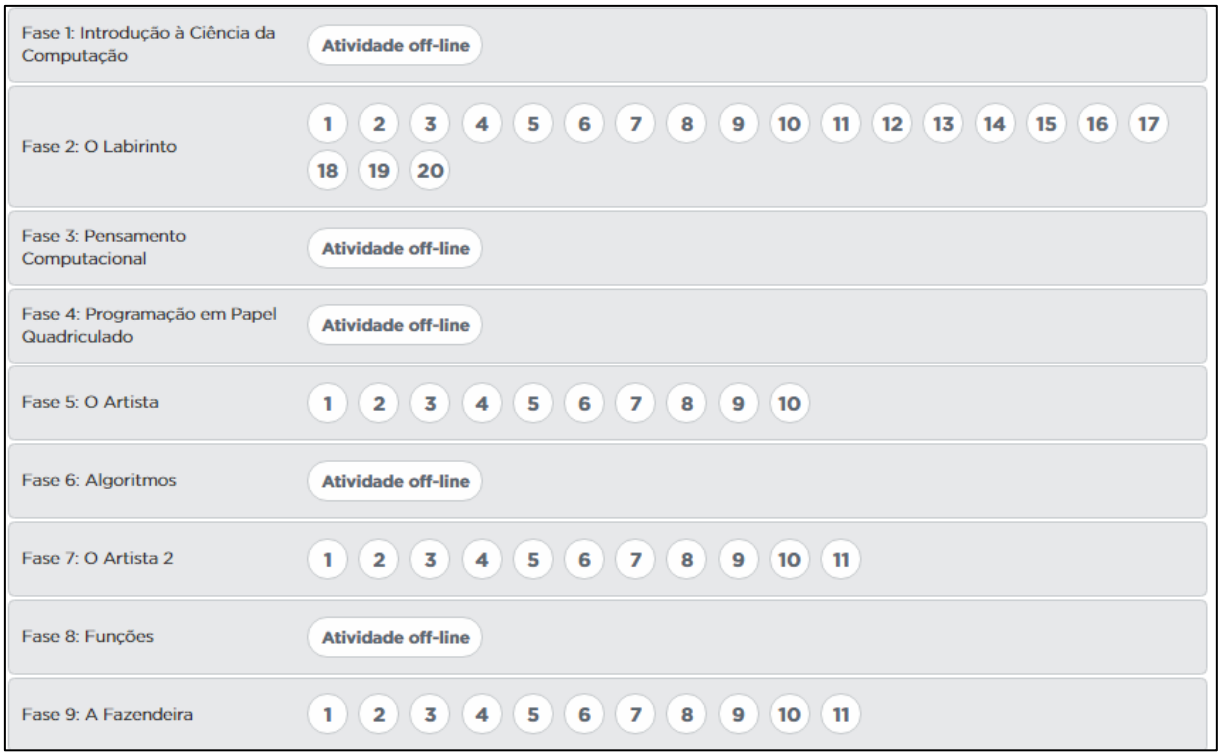


Figura 2.11 – Estrutura do curso de introdução à ciência da computação

O ambiente de programação (Figura 2.12) é basicamente dividido em duas áreas: à esquerda fica a animação e à direita a área de codificação. O Code Studio difere dos outros APV por oferecer atividades direcionadas, como o curso apresentado anteriormente, e atividades livres, como programar uma interface com o utilizador com linguagem JavaScript (13 anos ou mais), ou fazer desenhos no estilo da tartaruga do Logo. No caso de programar a interface com JavaScript, o ambiente possui duas abas: uma para o código e outra para o projeto da interface. Na aba de código (Figura 2.13) o aluno visualiza a interface, e, a partir das várias opções de menu, monta o seu programa em blocos ou em texto, além disso pode executar ou depurar o programa. Na aba de projeto (Figura 2.14) o aluno adiciona os componentes da interface e altera as suas propriedades.

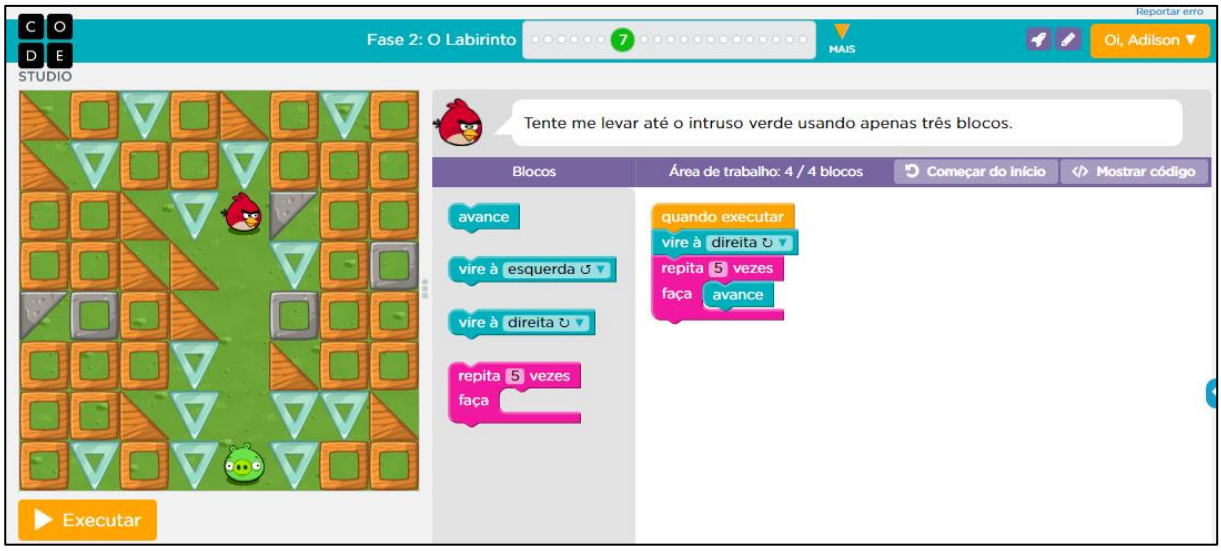


Figura 2.12 – Ambiente de programação do Code Studio (atividade direcionada)

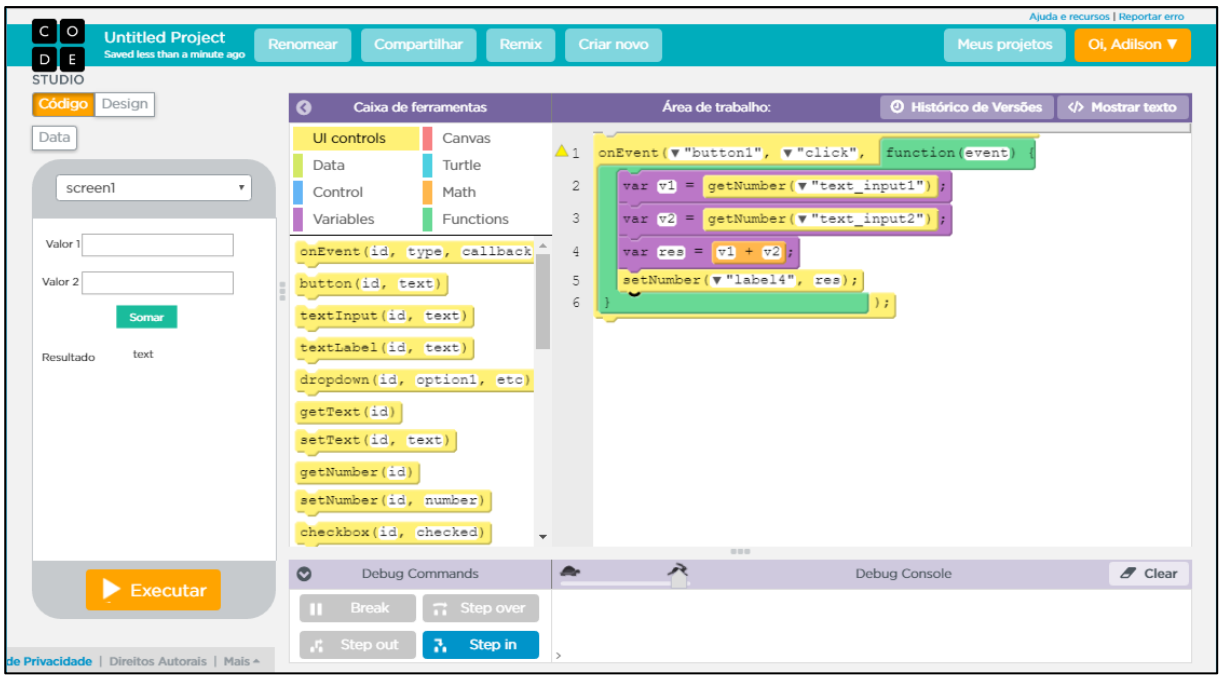


Figura 2.13 – Ambiente de programação do Code Studio (atividade livre – aba código)

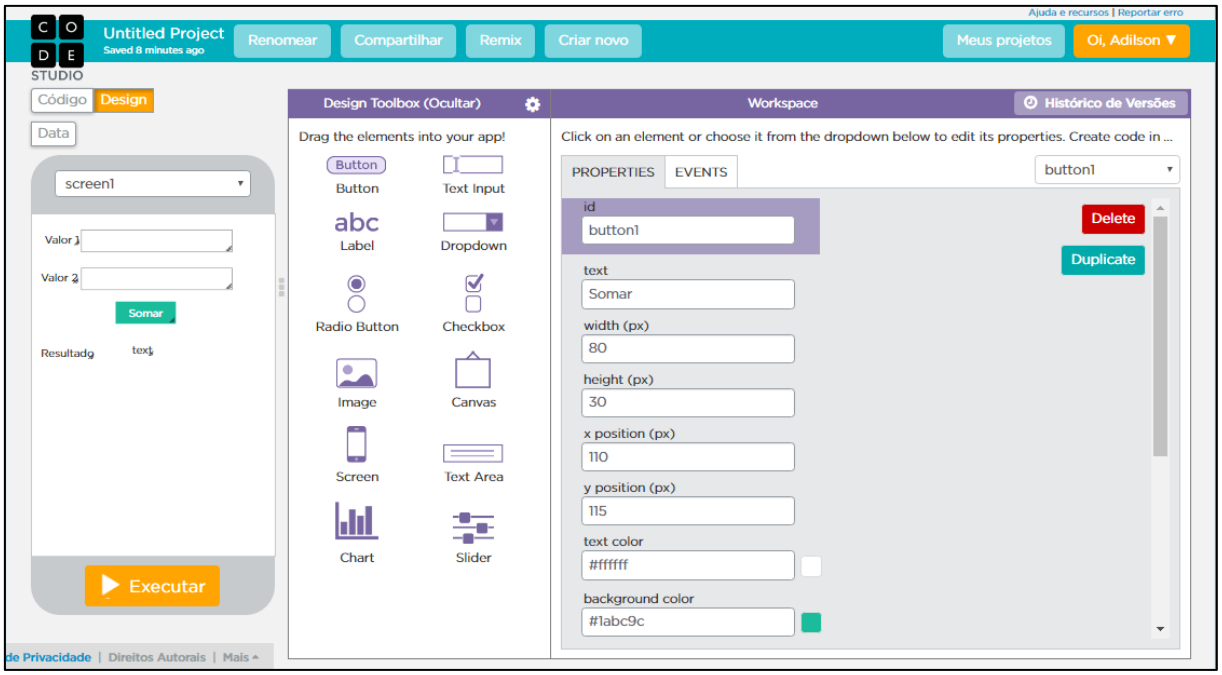


Figura 2.14 – Ambiente de programação do Code Studio (atividade livre – aba projeto)

O professor pode criar turmas para um determinado curso e acompanhar com estatísticas simples (quantidade de missões concluídas e blocos nas soluções) o progresso dos alunos (Figura 2.15).

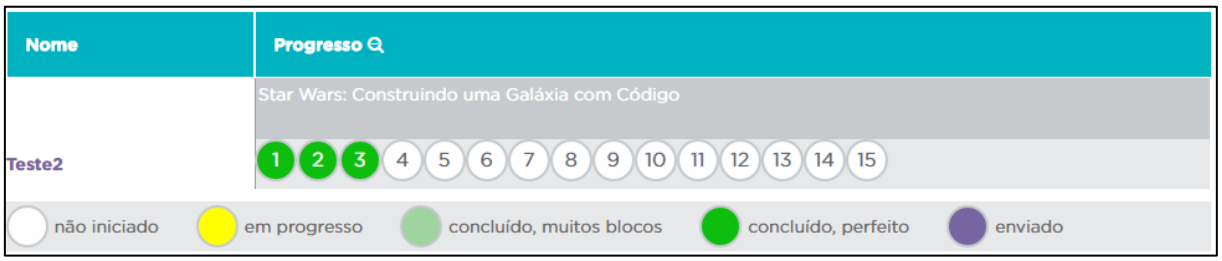


Figura 2.15 – Acompanhamento do progresso dos alunos no Code.org

2.5.8 Comparação entre os Ambientes

O Code Studio tem três tipos de atividades: 1) direcionadas, 2) programação de interfaces e 3) fazer desenhos livres. A comparação a seguir fará essa diferenciação adicionando o modo como sobrescrito.

Dos sete ambientes comparados, somente o Alice e o StarLogo são descarregados e instalados. Simultaneamente são os únicos que não são de código aberto. Os demais são executados diretamente num navegador. Três destes ambientes permitem codificar em texto como alternativa aos blocos: Pencil Code, com Coffeescript e JavaScript, Code Studio[3] tem um tipo de atividade que pode alternar

com JavaScript, e Alice que permite configurar os blocos com sintaxe Java, e visualizar o código fonte enquanto se montam os blocos. Inclusive, é nesse modo que o Code Studio é o único dos ambientes que permite a depuração do código. O Scratch e Snap! permitem que o utilizador faça um duplo clique sobre um bloco para executá-lo imediatamente. O Pencil Code tem o equivalente a esse recurso, mas digitando numa consola.

AIA e Code Studio[3] têm como atividades desenvolver uma interface com o utilizador e o aluno programar os eventos dos componentes. Code Studio[1,2] e Pencil Code são os únicos que têm somente um agente a ser programado. Snap!, Alice e StarLogo são os ambientes em que a programação segue a abordagem orientada a objetos, em que é possível criar métodos e atributos.

A Tabela 2.1 apresenta o resumo desta comparação.

Tabela 2.1 – Comparação entre APV.

	Web	Código aberto	Programação Textual	Depuração	Comandos ou Blocos Interativos	Desenvolver Interface com usuário	Multiagentes	Orientação a Objetos
Scratch	•	•			•		•	
Snap!	•	•			•		•	•
AIA	•	•				•	N/A	
Pencil Code	•	•	•		•			
Alice 3			•				•	•
StarLogo TNG							•	•
Code Studio[1]	•	•						
Code Studio[2]	•	•						
Code Studio[3]	•	•	•	•		•	N/A	

2.6 Estratégias de Ensino-Aprendizagem com PBB

Lee et al. (2011) propõem três elementos para colocar em prática a aprendizagem de PC:

1. Ambientes computacionais, que são os APV;
2. Processo de aprendizagem, sugerindo uma proposta de três estágios, com um padrão chamado Usar-Modificar-Criar, onde inicialmente o

aluno usa algo que já estava pronto (um programa que movimenta uma personagem), depois faz pequenas modificações (muda a aparência da personagem) e, por fim, acrescenta novas funcionalidades ou comportamentos, num ciclo de teste, análise e refinamento;

3. Oferecer domínios ou contextos concretos para que possam fazer associações com o seu quotidiano e assim melhorar as suas capacidades de abstração.

Brennan & Resnick (2012) recomendam quatro práticas (reconhecidas na engenharia de software) nas atividades para a aprendizagem de PC:

1. Ser incremental e iterativo, implementar um pouco e testar;
2. Testar e depurar;
3. Reusar e remisturar, o que permite aprender com o trabalho dos outros e contribuir com eles;
4. Abstrair e modularizar, separar em códigos menores os diferentes comportamentos dos objetos.

Os APV têm vindo a ser usados nas primeiras semanas nas disciplinas introdutórias de programação para diminuir a sobrecarga cognitiva com a sintaxe de uma nova linguagem e superar as barreiras iniciais com as capacidades e competências exigidas na programação (Rizvi et al., 2011; Dann et al., 2012; Mishra et al., 2014). O objetivo é ajudar a desenvolver intuitivamente um senso de como funcionam as estruturas de controlo (Malan & Leitner, 2007), a aperfeiçoar práticas responsáveis de codificação ao ter que produzir para outros (Wolz et al., 2009) e a praticar princípios de criação de programas, como a divisão do problema em pequenas partes (DiSalvo, 2014), para depois se aprofundarem com uma linguagem de programação textual.

Existem várias experiências relatadas com o uso de APV, porém a maioria em turmas de níveis anteriores ao ensino superior. Como o foco desta investigação é justamente para o uso desses ambientes no ensino superior, buscamos experiências limitadas a esse nível de ensino. Encontramos 14 trabalhos, onde o Alice foi usado com exclusividade em sete (Moskal et al., 2004; Parsons & Haden, 2007; Powers et al., 2007; Cliburn, 2008; Mullins et al., 2009; Garlick & Cankaya, 2010; Dann et al., 2012), Scratch exclusivo em quatro (Malan & Leitner, 2007; Wolz et al., 2009; Chetty & Barlow-Jones, 2012; Mishra et al., 2014), Scratch e Alice em um deles (Chang,

2014), Snap! em outro (Mladenović et al., 2016), e um trabalho com ambiente desenvolvido na própria investigação (Matsuzawa et al., 2015).

Em algumas destas propostas, as disciplinas foram divididas, iniciando com o APV, seguindo-se o restante do semestre com uma linguagem de programação convencional (Parsons & Haden, 2007; Powers et al., 2007; Cliburn, 2008; Garlick & Cankaya, 2010; Chetty & Barlow-Jones, 2012; Mishra et al., 2014), algumas outras propostas fizeram essa divisão por tópicos (Powers et al., 2007; Dann et al., 2012), outras criaram disciplinas adicionais ministradas concorrentemente ou antes da de introdução à programação (Moskal et al., 2004; Malan & Leitner, 2007; Mullins et al., 2009; Wolz et al., 2009; Mladenović et al., 2016), um trabalho usou o APV ao meio do semestre como forma de remediação (Chang, 2014) e outro permitiu que os alunos alternassem entre blocos e código nos exercícios (Matsuzawa et al., 2015).

Na maioria dos trabalhos em que foi relatado o processo de transferência da programação com blocos para código o professor apresentou a equivalência entre as estruturas e ações dos blocos com código. Parsons & Haden (2007) solicitaram aos alunos o desenvolvimento de um projeto com o Alice e à medida que a turma conheceu e aprendeu uma estrutura de controlo na linguagem real, solicitou que eles usassem essa estrutura no projeto do Alice. Dann et al. (2012) desenvolveram um *plugin* no NetBeans para transformar um projeto do Alice para Java e solicitaram mudanças nesse projeto em Java. O ambiente proposto em Matsuzawa et al. (2015) permitia que o aluno pudesse alternar livremente entre blocos e Java.

Quanto aos resultados, foram reportadas pequenas melhorias, apesar de não significativas, no desempenho dos estudantes (Mullins et al., 2009; Dann et al., 2012; Mladenović et al., 2016), na retenção dos alunos que antes desistiam a meio da disciplina (Moskal et al., 2004; Mullins et al., 2009; Matsuzawa et al., 2015), no entusiasmo e motivação em aprender (Moskal et al., 2004; Malan & Leitner, 2007; Chang, 2014; Mishra et al., 2014). Em algumas experiências houve pior desempenho ou dificuldade em relacionar as tarefas nos APVs com as linguagens de programação reais, dando a falsa sensação de que programar é trivial (Parsons & Haden, 2007; Powers et al., 2007; Cliburn, 2008; Garlick & Cankaya, 2010; Chetty & Barlow-Jones, 2012).

2.7 Considerações Finais

As três competências essenciais para resolver problemas de programação são a compreensão do código, a escrita ou geração de código e a depuração. O PC é um conjunto de conceitos inspirados na computação para resolver quaisquer tipos de problemas, e por isso se tem popularizado na produção de ferramentas, ambientes e metodologias de educação. Esses conceitos são basicamente a decomposição de problemas em tarefas menores, reconhecimento de padrões para reutilização de soluções anteriores, a abstração do problema para que a solução, ou parte dela, possa ser reutilizada noutras situações e projetar o algoritmo para concretizar as ideias. Existe uma série de dificuldades e equívocos que os alunos cometem durante o início da sua aprendizagem. Usar o conhecimento desses erros já identificados na literatura, permite-nos antecipar os problemas que os nossos alunos enfrentarão, e, assim, propor alternativas desde o início.

A PBB, através dos APV, permite que o aluno se concentre em resolver os problemas, e não na sua representação. Outra vantagem é acompanhar concretamente, através das animações, a execução da sua solução. Para utilização em cursos superiores têm vindo a ser usados com mais frequência os APV Alice e Scratch, com propostas para iniciar a disciplina com blocos e depois migrar para linguagem de programação, ou tratar essa migração por tópicos, e ainda a existência de uma disciplina inteira, prévia ou concomitantemente, com blocos. Quanto à transição entre blocos e código, a maioria das experiências apresentou aos alunos a equivalência entre ambos. Contudo, os resultados dessa abordagem no ensino superior não mostraram ser promissores. Os trabalhos relatam um distanciamento nas tarefas que alunos realizam no APV e nas linguagens de programação reais. Apenas dois trabalhos relataram a integração significativa entre as duas formas de resolver problemas, uma através do uso de um *plugin* para transformar aplicações do Alice em Java e outro que permitia livremente alternar entre blocos e código.

O próximo capítulo aborda o uso de jogos como uma estratégia de ensino-aprendizagem em vez de desenvolvê-los.

Capítulo 3

APRENDIZAGEM BASEADA EM JOGOS

3.1 Considerações Iniciais

A força dos jogos reside na capacidade de cativarem a aprendizagem através da imersão, onde o jogador vive a personagem do jogo, transmitindo-lhe as suas esperanças, valores e medos (Gee, 2003). Essa projeção do jogador para dentro do jogo torna mais significativo o processo de aprendizagem porque o aluno é um sujeito ativo nesse processo. A aprendizagem deve estar subordinada ao enredo, porém a diversão deve vir primeiro, e a instrução em seguida (Zyda, 2005). Em consequência dessas características motivacionais dos jogos, os alunos podem ser encorajados a vencerem desafios com programação num ambiente de entretenimento (Kazimoglu et al., 2012). Num trabalho prévio, identificámos 40 jogos (entre disponíveis para uso e descritos em artigos) que podem ser usados no ensino introdutório de programação (Vahldick et al., 2014), procurando mostrar a sua utilidade para ajudar a vencer a difícil barreira inicial nesse campo do conhecimento.

Este capítulo está organizado da seguinte forma: na próxima secção são apresentados alguns conceitos básicos sobre jogos sérios; na terceira secção são descritos os conceitos de motores de jogos, concluindo com a necessidade do seu desenvolvimento para jogos sérios; na quarta secção apresentamos as características dos jogos casuais e as possibilidades da sua aplicação em jogos sérios; na quinta secção apontamos a abordagem construcionista usando jogos e como pode ser

relevante para a aprendizagem da programação; na sexta secção são descritas algumas características sobre o uso de jogos na aprendizagem da programação e ilustrados vários exemplos sobre jogos desenvolvidos para esse fim, dando-nos assim uma base para a investigação.

3.2 Jogos Sérios

Os jogos sérios são uma tipologia de jogos projetados com a pedagogia como objetivo primário (Arnab et al., 2012). Os jogos atraem os alunos, aumentam a relevância e a imersão no conteúdo e auxiliam na transferência da aprendizagem. Para que o aluno goste de um jogo, necessita de se distanciar da ideia de sala de aula ou dos livros para viabilizar a aprendizagem (Aldrich, 2009).

Os jogos são meios mais naturais de aprender do que as escolas (Powell, 1982; Prensky, 2001). O ato de jogar consiste numa importante função biológica evolucionária relacionada com a aprendizagem. No mundo selvagem os animais *passam* conhecimentos aos seus descendentes através das brincadeiras. Os jovens envolvem-se mais nas atividades em que aprendem fazendo, e a sua motivação pode vir das competições entre quem faz melhor e mais rápido.

Os jogos sérios são projetados para que os jogadores desenvolvam novas competências e/ou conhecimentos, ou reforcem capacidades existentes (Boller & Kapp, 2017). Os jogos favorecem a aprendizagem, normalmente não exigindo um treino ou ajuda externa, a não ser pelos comentários das pessoas que se divertiram jogando (Gee, 2003). O sentimento de diversão permite que o aluno realize as suas tarefas de forma mais fácil, mesmo quando exige mais esforço (Prensky, 2001), porque mantém-no atento e envolvido na tarefa. A diversão é essencial nos jogos digitais (O'Neil et al., 2005; Sweetser & Wyeth, 2005) e útil para a aprendizagem, pois a experiência afetiva positiva pode melhorar os resultados da aprendizagem (Pavlas, 2010).

Além da diversão, existem outras características dos jogos sérios que contribuem para a melhoria dos resultados na aprendizagem, como os elementos de fantasia, curiosidade e tipos de tarefas inovadoras (Ronimus et al., 2014), controles intuitivos e dificuldade progressiva (Mckernan et al., 2015), e conteúdo adaptativo de

acordo com o estilo de aprendizagem do aluno (Soflano et al., 2015). Os objetivos nas tarefas criam o elo entre a mecânica do jogo, o desafio a ser vencido, e as decisões do aluno que lhe conferem a aquisição das competências (Lameras et al., 2017). Esses fatores nos jogos objetivam alcançar o estado de fluxo que é uma experiência tão gratificante que as pessoas estão dispostas a fazê-la por sua própria causa, mesmo que seja difícil ou perigosa (Csikszentmihalyi, 1990).

As regras e as mecânicas dos jogos são um conjunto de instruções que especificam como um jogo pode ser jogado, o que pode e o que não pode ser feito num jogo, as possíveis interações com as personagens, os ambientes e os objetos, e quais são os desafios e tarefas dum jogo (Salen & Zimmerman, 2004). Ao mesmo tempo que essas regras e mecânicas adicionam diversão a um jogo, elas são partes essenciais para a experiência de aprendizagem (Kapp, 2012). Quanto mais simples são as regras e as mecânicas melhor são direcionados os esforços do jogador em aprender as habilidades e conteúdos instrucionais. Os jogadores perdem o interesse e a atenção quando não sentem o controlo no jogo (Gee, 2004). Contudo, falhar no jogo oferece a oportunidade imediata do aluno refletir sobre a sua compreensão dos conceitos envolvidos na tarefa concluída com falhas. Refazer as tarefas num jogo ajuda o aluno a rever e a melhorar o seu conhecimento (Aldrich, 2009).

A forma como um estudante usa um jogo pode fazer parte formal da sua avaliação. Este aspeto pode influenciar muito a forma do aluno se empenhar no jogo e na sua aprendizagem a partir dele (Whitton, 2010). Avaliar a aprendizagem exige uma abordagem sistemática para determinar o sucesso e as dificuldades dos alunos (Bellotti et al., 2013). O grande desafio para criar essas avaliações é que necessitam naturalmente de serem tarefas do jogo, onde não se sacrifique a confiabilidade e validade do processo (Shute & Ke, 2012) e não se interrompa o fluxo na experiência de jogo correndo o risco do aluno perder o interesse (Chen, 2007). As interações com o jogo, ou seja, a sequência das ações, é que demonstram, com base nas escolhas do aluno, o que sabe ou não do assunto. O progresso do aluno ao avançar num jogo e acumular pontos de experiência pode ser uma evidência da sua aprendizagem. Não conseguir superar os desafios do jogo, representa que o aluno tem deficiências quanto às competências necessárias para fazê-lo, e rastrear as suas tentativas pode apontar quais são esses problemas e o apoio que necessita. Essas interações geram usualmente uma grande quantidade de dados a serem analisados. Learning Analytics (LA) é uma abordagem para identificação, análise e apresentação de dados sobre as

interações dos alunos nos ambientes de aprendizagem, com o propósito de identificar o seu progresso e apoiar professores, alunos e o próprio ambiente na tomada de decisões (Greller & Drachsler, 2012; Alonso-Fernandez et al., 2017).

A satisfação num jogo aumenta o nível de interesse no assunto e no conhecimento tratado pelo jogo (Iten & Petko, 2016). A compreensão sobre a diversão com os jogos ainda é pouco desenvolvida (Connolly et al., 2012). Existem alguns trabalhos que avaliam a influência da diversão em jogos sérios (Cohen, 2014; Brom et al., 2015) e outros que propõem instrumentos para medi-lo (Fu et al., 2009; Savi et al., 2011).

EGameFlow (Fu et al., 2009) é um instrumento adaptado do GameFlow (Sweetser & Wyeth, 2005) para avaliar jogos sérios. O jogo é mensurado em oito dimensões baseadas no fluxo de Csikszentmihalyi (1990):

- Concentração: a quantidade de informação ou tarefas extras e desnecessárias não comprometem nas atividades de aprendizagem?
- Clareza nos objetivos: os objetivos são compreensíveis e explícitos durante todo o jogo?
- Suporte: o aluno recebe suporte suficiente e no momento correto?
- Desafios: os desafios são coerentes com as habilidades e o conhecimento do aluno?
- Autonomia: o aluno tem o sentimento de que comanda o jogo ao conhecer os controlos necessários para realizar todas as ações?
- Imersão: o aluno sente-se envolvido profundamente nas tarefas do jogo?
- Interação social: existem recursos no jogo para promover a competição ou cooperação entre os alunos?
- Melhoria no conhecimento: o aluno consegue aumentar o seu nível de conhecimento e as suas habilidades enquanto cumpre os objetivos do jogo?

O EGameFlow contém 42 questões com níveis de concordância numa escala Likert de sete níveis. O valor final classifica o senso geral de diversão no jogo em uma escala de 0 a 100.

MEEGA (Savi et al., 2011) é um instrumento para avaliação de jogos sérios na área de Engenharia de Software, formado por três componentes e catorze dimensões:

- Motivação: baseado noutro modelo (ARCS) para compreender as maiores influências que motivam a aprendizagem; avalia quatro dimensões: atenção, relevância, confiança e satisfação;
- Experiência do utilizador: avalia a interação do utilizador com o jogo nas dimensões: imersão, desafio, competência, divertimento, controle e interação social;
- Aprendizagem: é medido com base nos três primeiros níveis da taxonomia de Bloom (conhecimento, compreensão e aplicação) e mais dois para aprendizagem de curto e longo prazo.

O MEEGA contém 27 itens fixos com níveis de concordância na escala Likert com cinco níveis. Além disso, sugere criar itens para avaliar os objetivos educacionais com base nos três níveis da taxonomia de Bloom.

3.3 Motores de Jogos Sérios

Os motores de jogos são programas de computador extensíveis que podem ser usados como base para vários jogos diferentes sem muitas modificações (Gregory, 2009), pois fornecem componentes reutilizáveis que aceleram o desenvolvimento de jogos. Esses componentes podem-se referir à animação e renderização, podem tratar da manipulação das entradas (teclado, rato, ou dispositivos fisiológicos), multimédia, representação e manipulação de caracteres, inteligência artificial, leis da física, redes e multijogadores e lógica do jogo (fases, desafios, pontuação e conquistas). Na prática, a maioria dos motores de jogos são ajustados para um género específico, como aventura (Moreno-Ger et al., 2008), estratégia em tempo real (Ponsen et al., 2005), tiro na primeira pessoa e jogos baseados em cenários (Westera et al., 2008).

Quase todos os criadores de jogos sérios usam motores de jogos comerciais, sugerindo este facto que existe uma lacuna de recursos para motores específicos para jogos sérios (Cowan & Kapralos, 2014). Os motores de jogos sérios necessitam de possuir recursos para fornecer conteúdo instrucional, fazer avaliações e oferecer suporte. Alguns motores de jogos sérios foram desenvolvidos para contextos específicos, como jornalismo (Hatfield & Shaffer, 2006), sustentabilidade e conservação (Lee, Xu, et al., 2014), gestão de segurança (Park & Kim, 2013) e

programação de computadores (Shabalina et al., 2008; Torrente et al., 2014). Motores com recursos embutidos para criar tipos específicos de jogos normalmente têm um valor pedagógico maior e facilitam o uso pelos professores (Moreno-Ger et al., 2008). O motor de Shabalina et al. (2008) estendeu outro motor 3D de código aberto que interpreta alguns arquivos para apresentar o material instrucional, e integrou um compilador para as questões de avaliação. O motor de Torrente et al. (2014) apresenta elementos de aprendizagem (conjunto de ações e estruturas, ou seja, os comandos para criar o programa) e uma camada de interpretação para converter o programa nesse conjunto de ações e estruturas. O maior desafio no desenvolvimento dos jogos para a aprendizagem de programação é a existência de recursos para que os alunos possam desenvolver e testar as suas soluções (Eagle & Barnes, 2009).

Ahmad et al. (2014) reviram seis *frameworks* e modelos para desenvolver jogos sérios e encontraram deficiências em como usá-los para desenvolver e testar, concluindo que as investigações necessitam de fornecer ferramentas mais apropriadas para fomentar o desenvolvimento e a adoção de motores de jogos sérios. Os modelos referidos pelos autores são especificações de alto-nível, que identificam requisitos e recursos que deveriam estar presentes em jogos sérios. Os motores de jogos deveriam tentar fornecer soluções padrão para a programação de jogos sem a necessidade de prescrever regras ou configurações particulares do jogo (Smith et al., 2010), mas construindo essas soluções através das APIs.

3.4 Jogos Sérios Casuais

Jogos Casuais (JC) são jogos fáceis de serem jogados e exigem poucas instruções para começar a usá-los (Kadle, 2009). São jogos que seguem princípios e heurísticas que amenizam a curva de aprendizagem para jogá-los (Juul, 2010; Trefry, 2010). Por outro lado, jogos *hardcore* (JH) são aqueles em que o jogador é inserido num ambiente virtual – tipicamente em 3D – em que são despendidas várias horas para aprender as regras, mecânicas, ferramentas e estratégias para vencer (Wang & Sun, 2011). Outra importante diferença é o tempo na sessão de jogo: enquanto nos JC o jogador termina uma tarefa em poucos minutos, as missões dos JH são desenvolvidas para envolverem o jogador em conquistar pequenos objetivos e

procurar cumpri-los por vários caminhos, o que acaba consumindo bastante tempo (Landers & Callan, 2011; Chiapello, 2013). Ambos os tipos de jogos podem ser valiosos como jogos sérios: JH para aprender conhecimento procedimental, ou como um jogo em grande escala para cobrir uma porção substancial de um curso ou para treinar um departamento de uma empresa; JC como ferramenta complementar para aprender conhecimento conceitual ou factual (Kadle, 2009; Smith & Sanchez, 2010).

Jogos casuais costumam misturar algumas mecânicas, como combinação, ordenação, procura, bater ou atirar, encadear, gerir, construir e socializar (Trefry, 2010). A combinação refere-se a alinhar um conjunto de objetos com características similares. Jogos de ordenação requerem habilidades naturais dos jogadores: colocar objetos numa sequência conforme algumas restrições. Mecânicas de procura referem-se ao jogador encontrar objetos escondidos. Jogos físicos possuem as mecânicas de atirar ou bater: alcançar um item exposto. Encadear é uma mecânica que permite ao jogador criar uma sequência de ações para cumprir um objetivo no jogo, que pode ser intercalada com outra sequência de ações no intuito de aproveitar alguma situação como, por exemplo, estar perto de um objeto fonte ou consumidor de um produto, para cumprir simultaneamente outros objetivos ganhando tempo e consequentemente mais pontos. As mecânicas de gestão na maioria das vezes trabalham em conjunto com as mecânicas de encadeamento. Jogos de gestão emulam processos de trabalho bem conhecidos em que o jogador está comprometido em realizar tarefas repetitivas para cumprir vários pequenos objetivos. Jogos com construção oferecem oportunidades para os jogadores exercitarem a sua imaginação ao criarem novas coisas a partir do zero. Jogos de cartas e de tabuleiro promovem naturalmente a interação social. Os recursos sociais dão a oportunidade para que os jogadores se divirtam em conjunto ou combatam uns contra os outros em jogos digitais. Os jogados em redes sociais permitem a interação assíncrona: os jogadores visitam os mundos dos seus amigos, ajudam-nos a finalizarem tarefas e acabam por ganhar pontos ou itens, ou formam grupos (gangues e clãs) de jogadores.

As características dos JC fazem deles uma interessante e prática opção para o uso educacional. Além de fáceis de serem jogados, estes jogos não requerem grandes requisitos de *software* e *hardware* (normalmente são executados num navegador), os objetivos são conquistados em poucos minutos, e permitem o foco na tarefa por não envolverem muitas ações e considerações amplas do seu mundo (Leonardou & Rigou, 2016). Os alunos necessitam de gerir o seu tempo e isto não é

possível se precisarem de gastar muito tempo para aprenderem a dominar um jogo, conhecendo as suas regras e as *nuances* das mecânicas, restando pouco tempo para se focarem no conteúdo e aprendizagem (Landers & Callan, 2011). Os JC vêm sendo usados com certo sucesso noutras áreas, como nos museus (Birchall et al., 2012) e na saúde (Gao & Mandryk, 2011). São jogados para os jogadores se distraírem e relaxarem, e como toda atenção e foco são exigidos na aprendizagem, os jogos sérios casuais podem partilhar algumas características dos JH no sentido em que os alunos se tornam intimamente imersos, profundamente comprometidos e retornam frequentemente (Radoff, 2011).

3.5 Abordagem Construcionista no Uso de Jogos

Os computadores podem melhorar a aprendizagem quando os alunos conseguem ver os resultados concretos dos seus esforços (Papert, 1980). Na abordagem construcionista de educação, os alunos coordenam a sua aprendizagem pela construção, manipulação e teste de conceitos num *micromundo* (Laurillard et al., 2013). Um *micromundo* é um espaço com premissas e restrições que fornecem um contexto para que o aprendiz construa o seu conhecimento através da experimentação (Papert, 1980). Os alunos aprendem pela exploração e construção nesse mundo, onde os efeitos das suas ações se refletem no que é correto ou incorreto nas suas crenças. Os resultados da aprendizagem ocorrem pela prática ativa.

Os jogos construcionistas trazem essas ideias (aprendizagem dirigida pelo aluno, construções pessoalmente significativas, ênfase em ideias significativas) em seus projetos (Weintrop & Wilensky, 2014). Na última década, os ambientes e os jogos de PC materializam a abordagem da aprendizagem construcionista. Para aprender conceitos abstratos de programação, os alunos necessitam de construir esses conceitos por experiência prática (Hidalgo-Céspedes et al., 2014). O PC e o construcionismo compartilham o desenvolvimento de duas habilidades básicas: raciocínio procedimental e depuração. Pensar procedimentalmente envolve dividir um problema em partes menores e reconhecer padrões que podem efetivamente se repetir (Papert, 1980). Depurar envolve sistematicamente tentar ajustar um pedaço de

código para identificar e corrigir erros para manter o sistema executando apropriadamente (Holbert & Wilensky, 2011).

Os jogos construcionistas contemplam dois princípios de projeto: as suas ferramentas e recursos devem ser expressivos e os objetivos precisam encorajar a exploração (Weintrop et al., 2012). O tamanho dos blocos de construção deve permitir que o aluno expresse ideias e estratégias que são significativas no seu contexto de aprendizagem: nem tão grandes para que o jogo seja muito fácil, e nem tão pequenas para evitar o tédio ou tarefas muito difíceis. Os jogos podem recompensar por uma variedade de descobertas, não sendo limitados a uma única ou a um pequeno conjunto de estratégias vencedoras. As atividades de criação podem fazer-se de muitas formas, mas é importante que os artefactos resultantes sejam identificáveis e úteis. Além disso, o ciclo típico de interação e resposta para desenvolver o PC (Kazimoglu et al., 2013) é adequado para qualquer jogo construcionista: (1) os alunos desenvolvem, executam ou depuram a solução, (2) o jogo realiza as ações baseados na solução submetida, e (3) o jogo fornece os resultados, respostas e suporte ao aluno. Este ciclo iterativo e interativo proporciona poderosas possibilidades para o aluno tentar e repetir as suas tentativas naquilo que acredita, e melhorar o seu conhecimento.

As Figuras 3.1 e 3.2 ilustram diferentes abordagens para construir a solução do problema. O Code.org já foi mencionado no capítulo anterior e é um exemplo de um ambiente construcionista onde as soluções são criadas usando somente o rato e encaixando blocos como no Lego. No Formula T Racing, os jogadores dirigem pintando (Holbert & Wilensky, 2011): cada cor corresponde a uma velocidade particular, e os jogadores pintam a pista para ajustar a velocidade do carro.

Figura 3.1 – Code.org

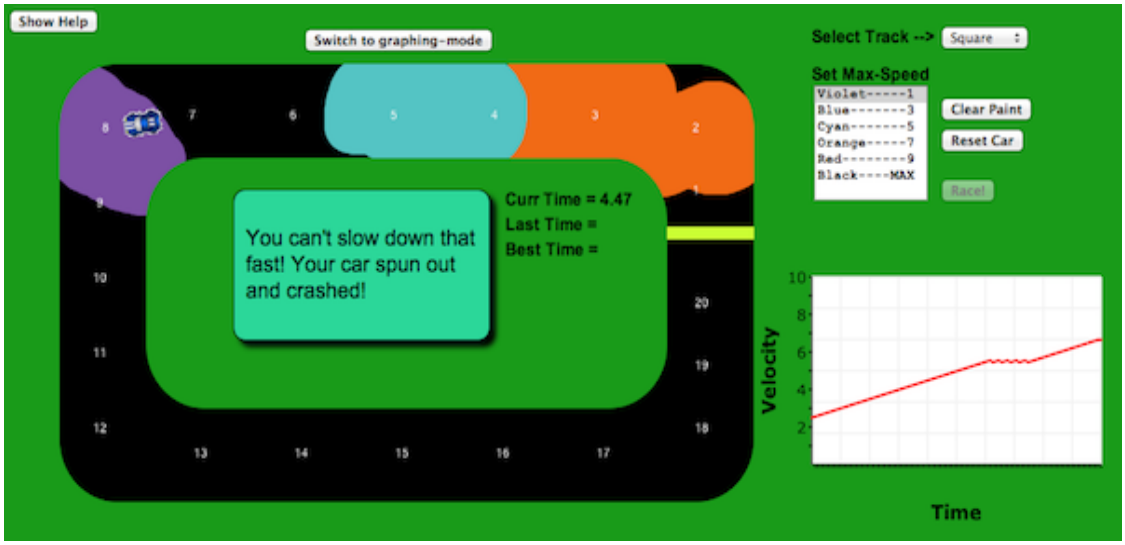


Figura 3.2 – Formula T Racing (Holbert & Wilensky, 2011)

3.6 Jogos para o Ensino da Programação e do Pensamento Computacional

O principal fator para promover a aprendizagem da programação é que o aluno pratique disciplinadamente e intensivamente (Robins et al., 2003). Mas, muitas vezes falta-lhes motivação para se envolverem nessas tarefas. A prática em resolver os problemas nos jogos, que podem ser mais motivadores do que os exercícios

tradicionais, promove a confiança e a experiência, construindo um conjunto estruturado de padrões de soluções, que lhes será muito útil quando se depararem com problemas usando linguagens de programação reais (Barnes et al., 2007).

Os jogos para programar misturam programação e jogabilidade fornecendo uma autêntica e rica experiência de programação (Weintrop & Wilensky, 2016b). Contudo, o uso de um jogo não tem sempre um impacto positivo, e é necessário considerar (Sung, 2009): mudanças no currículo, nos instrumentos e na forma de avaliação das atividades dos alunos; que o jogo seja simples, fácil e barato de ser implantado; que o jogo seja neutro quanto ao género e à perícia dos alunos, ou seja, que evite desencorajar tanto aqueles com pouco conhecimento prévio como os que já apresentam um certo domínio.

Existem essencialmente duas abordagens quanto à forma de integrar jogos na aprendizagem de programação: criar jogos ou jogá-los. Na primeira abordagem, é solicitado aos alunos que desenvolvam pequenos jogos ou pequenos programas em ambientes direcionados e restritos para aplicar os conceitos de programação (Bayliss & Strout, 2006; Leutenegger & Edgington, 2007). Na segunda abordagem experimentam os jogos para reforçar e praticar os conceitos de programação (Barnes et al., 2007; Muratet et al., 2010; Lee, Bahmani, et al., 2014). No capítulo anterior apresentámos alguns ambientes de autoria que são usados para a primeira abordagem. Nesta subsecção apresentamos uma lista de jogos direcionados para a segunda abordagem, que é o foco da nossa investigação.

Essa subsecção é baseada numa publicação prévia (Vahldick et al., 2014) e complementamos com novos trabalhos publicados posteriormente. Para orientar este levantamento formulámos três questões de pesquisa:

1) Que jogos estão disponíveis para ajudar na aprendizagem introdutória de programação publicados na literatura ou disponíveis na web nos últimos 10 anos?
2) Quais são as competências, introdutórias ou mais avançadas, desenvolvidas com esses jogos?
3) Quais são as lacunas que podem ser exploradas e não cobertas por esses jogos?

Para responder a estas perguntas é necessária uma amostra de jogos para programar. Procurámos em bibliotecas digitais como a ACM Digital Library, Science Direct e IEEE Xplore usando os termos *game*, *introductory programming*, *computer*

programming e *novice programming*. Em seguida, examinámos os resultados e considerámos somente os artigos que descrevem jogos usados, propostos ou desenvolvidos para apoiar a aprendizagem de programação introdutória. O mesmo processo foi feito com os artigos que estes referenciavam, ou eram referenciados, possivelmente não indexados nessas bibliotecas. Também foram consideradas algumas conferências de informática.

Em seguida, foram pesquisados os termos jogos e programação em bases de dados comerciais, como a AppStore e Google Play, em *websites* com jogos em Flash e na Web. Apenas foram recolhidos os jogos explicitamente relacionados com programação de computadores. Diversos jogos de puzzle exercitam o raciocínio, porém não são o foco deste trabalho e por isso não foram adicionados à lista de jogos (Tabela 3.1).

Para cada jogo foram executados os seguintes passos: 1) ler o artigo, no caso de o jogo ser descrito numa revista ou atas de uma conferência; 2) ler a página web, quando o jogo estava disponível para descarregar e existia informação suficiente para o identificar; ou 3) jogar o jogo quando não existia informação suficiente.

Foram considerados jogos apenas aqueles que contém desafios, ou missões, embutidos no jogo e que o próprio jogo consegue verificar as respostas. Isso significa que o aluno consegue autonomamente jogar (e aprender) sem a intervenção de um professor. Existem alguns jogos publicados que necessitam da mediação de professores para analisar as respostas dos alunos. Esses jogos não foram considerados neste estudo por não suportarem o trabalho autónomo dos alunos.

Encontrámos 54 jogos e organizámo-los em três géneros:

- LOGO-Like: (23 jogos) é um tipo de jogo de ação em que se programa o movimento de um robô, uma tartaruga, ou qualquer outro tipo de personagem, em que existem missões como, por exemplo, alcançar uma posição no mundo, ou recolher uma quantidade de objetos. Esses movimentos são comandados através de uma linguagem de programação simplificada, tal como a linguagem LOGO de Seymour Papert (Papert, 1988);
- Aventura: (11 jogos) o jogador controla uma personagem responsável por explorar um mundo, recolher objetos e conversar com outras personagens controladas pelo jogo (Wassila & Tahar, 2012);

- Geral: (20 jogos) foi encontrada uma pequena quantidade de vários géneros, como simulações, de estratégia em tempo real e jogos de desafio. Por essa razão, resolvemos agrupá-los num único género.

Para cada jogo, a Tabela 3.1 indica se é disponível para descarregar e usar, ou se foi obtido de um artigo, a plataforma onde o jogo é executado (não identificamos essa informação nos jogos indicados com "?"), e a forma como o aluno resolve os problemas: alguns através de linguagem de programação, e outros por arrastamento de blocos textuais ou iconográficos; os indicados como linguagem proprietária implicam que foram inventadas pelos autores do jogo.

Tabela 3.1 – Lista de Jogos

	Nome do Jogo	Tipo	Plataf.	Linguagem
1.	AlgoGame (Debabi & Bensebaa, 2016)	G	Web	-
2.	APIN (Dim & Edson, 2011)	L	Win	Blocos Textuais
3.	BootLogic http://botlogic.us/play	L	Web	Blocos Gráficos
4.	Baralho das Variáveis (Kahwage et al., 2013)	G	Web	-
5.	Bomberman (Chang & Chou, 2008)	G	Win, Linux	C
6.	Cargo-Bot http://twolivesleft.com/CargoBot	L	iOS	Blocos Gráficos
7.	Catos Hike http://hwahba.com/catoshike	L	iOS	Blocos Gráficos
8.	(Chao, 2016)	L	?	Blocos Textuais
9.	CheckIO http://js.checkio.org	G	Web	JavaScript
10.	CMX (Malliarakis et al., 2013)	A	Win	Java
11.	Coddy Luck http://appike.com/coddy	L	iOS	Blocos Gráficos
12.	Code.org http://learn.code.org	L	Web	Blocos Textuais
13.	Code Combat http://codecombat.com/	L	Web	JavaScript
14.	Code Factory (Gomes et al., 2017)	G	?	Blocos Textuais
15.	Code Hunt http://www.codehunt.com/	G	Web	Java ou C#
16.	Code Monkey http://www.playcodemonkey.com	L	Web	Linguagem proprietária
17.	Code Spells (Esper et al., 2014) https://sites.google.com/a/eng.ucsd.edu/codespells	A	Win	Java
18.	Codin Game http://www.codingame.com	G	Web	26 linguagens comerciais
19.	Colobot http://www.ceebot.com/colobot/index-e.php	A	Win	Proprietário inspirado em C++, C# and Java
20.	Cube Game (Piteira & Haddad, 2011)	L	Web	Blocos Gráficos
21.	Daisy the Dino http://www.daisythedinosaur.com	L	iOS	Blocos Textuais
22.	Dragon Architect (Bauer et al., 2015)	G	Web	Blocos Textuais
23.	Dream Coders (Chang et al., 2012)	A	Java	Java
24.	EleMental: The Recurrence (Chaffin et al., 2009)	G	Win	C#
25.	Elevator Saga http://play.elevatorsaga.com	G	Web	JavaScript
26.	Empire of Code http://empireofcode.com	G	Web	Python ou JavaScript
27.	Entrando pelo Cano (Scaico et al., 2012)	G	Web	-
28.	Gidget (Lee & Ko, 2011)	L	Web	Blocos Textuais

29.	Glitchspace http://www.glitchspace.com	L	Win, Mac	Blocos Textuais
30.	IA Game (Adamo-Villani et al., 2012)	A	iOS, Win	-
31.	Kodable http://www.surfscore.com	L	iOS	Blocos Gráficos
32.	Light-Bot 2 http://armorgames.com/play/6061/light-bot-20	L	Web	Blocos Gráficos
33.	Machinist-Fabrique http://learntocode.biz/	G	Win	Blocos Textuais
34.	Move the Turtle http://www.geekkids.me	L	iOS	Blocos Gráficos
35.	Prog & Play (Muratet et al., 2010)	G	Win, Linux	C, C++, Java, Ocaml, Ada e Compalgo
36.	ProGame (Dantas et al., 2011)	A	Win	-
37.	Program your robot (Kazimoglu et al., 2012)	L	Web	Blocos Gráficos
38.	Project Orion (Coelho et al., 2011)	A	Win	Configurado pelo professor
39.	RoboBUG (Miljanovic, 2015)	G	Win	C++
40.	RoboCom http://www.cubesteam.com/Game-RoboComBasic.html	L	Win, iOS, Android, Linux	Blocos Gráficos
41.	RoboLogic http://www.digitalsirup.com/apps/app_robologic.html	L	iOS	Blocos Gráficos
42.	Robotimov (Dantas et al., 2013)	A	Win	-
43.	Robot ON! (Miljanovic & Bradbury, 2016)	G	Win	C++
44.	Robozzle http://robozzle.com	L	Web, iOS, Android	Blocos Gráficos
45.	Ruby Warrior https://www.bloc.io/ruby-warrior	G	Web	Ruby
46.	Saving Sera (Barnes et al., 2007)	A	Win	Linguagem proprietária
47.	Space Goats (Wahner et al., 2012)	G	iOS, Android	Visual através de blocos
48.	Takkou (Barbosa et al., 2011)	L	?	Blocos Textuais
49.	TALENT (Maragos & Grigoriadou, 2011)	L	Web	Blocos Textuais
50.	The Catacombs (Barnes et al., 2007)	A	Win	-
51.	Train B&P (Liu et al., 2011)	G	Win	Linguagem proprietária
52.	Tynker Lost in Space http://www.brainpop.com/games/tynkerlostinspace	L	Web	Blocos Gráficos
53.	World of Variables (Zapušek & Rugelj, 2013) http://hrast.pef.uni-lj.si/~svet_spremenljivk/	G	Win	-
54.	Wu's Castle (Eagle & Barnes, 2009)	A	Win	-

Pode-se observar que apenas sete dos jogos do tipo LOGO-Like provêm de publicações. Esses trabalhos podem contribuir com lições aprendidas em relação às suas experiências.

A experiência de Lee & Ko (2011), com o jogo *Gidget*, envolveu a participação de 116 aprendizes de programação utilizando um sistema de monetização chamado *Amazon´s Mechanical Turk* em que o jogador recebia valores monetários de acordo com a quantidade de fases jogadas. Diferente dos demais jogos desta categoria, no jogo *Gidget* a personagem não anda um passo de cada vez, mas vai diretamente para onde estão os objetos ou outras personagens. A Figura 3.3 ilustra o ambiente de desenvolvimento, execução e depuração do jogo. O objetivo do estudo foi avaliar se representar emoções na personagem, por exemplo, os estilos dos diálogos e as

expressões faciais, contribuíam para a resolução dos exercícios. Foi utilizado um grupo de controlo onde a representação da personagem era desprovida de emoções. O estudo demonstrou que aspetos emocionais representados na personagem aumentaram a motivação para resolver os problemas.

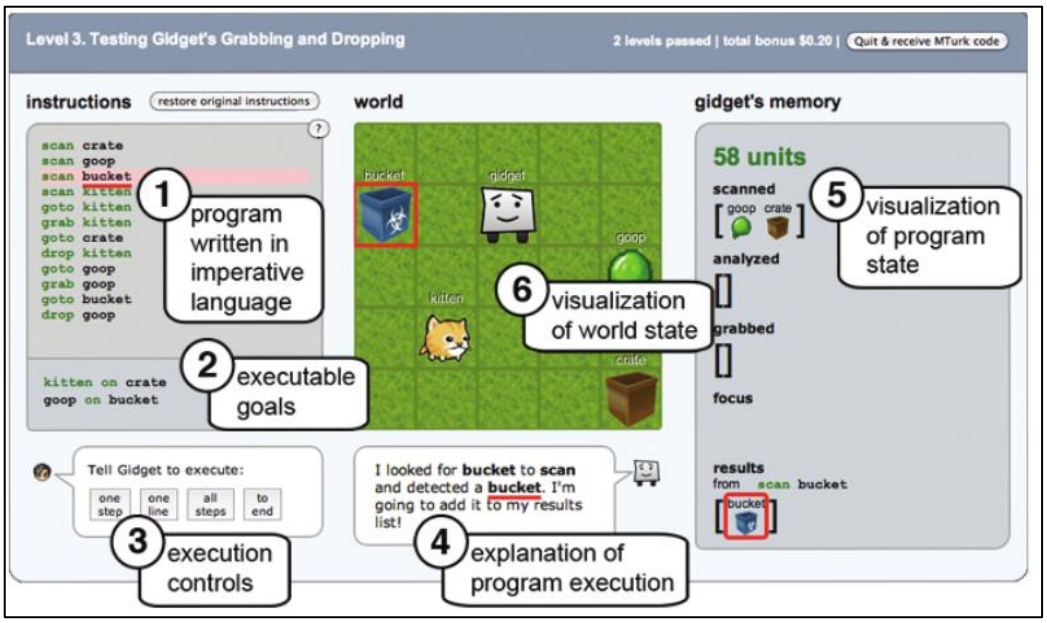


Figura 3.3 – Ambiente do Jogo (Lee & Ko, 2011)**, p. 3**

A avaliação na experiência com a utilização do jogo *Program your robot* (Figura 3.4) (Kazimoglu et al., 2012) teve como objetivo verificar qual a impressão que os 25 alunos tiveram, através de comentários e sugestões sobre a mecânica e usabilidade do jogo. Os alunos acharam o jogo adequado para ajudá-los a entender as estruturas na programação e o desenvolvimento de suas capacidades de resolução de problemas.

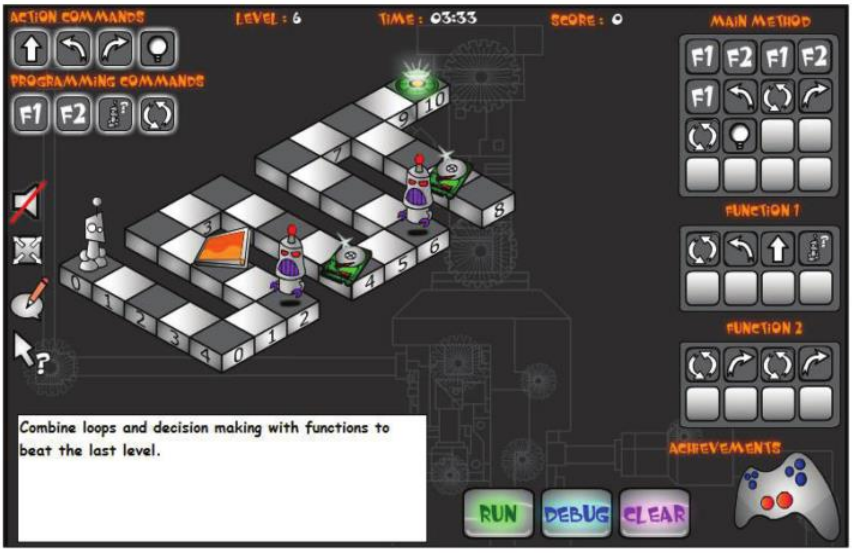


Figura 3.4 – Ambiente do Program your Robot (Kazimoglu et al., 2012)**, p. 5**

Em Maragos & Grigoriadou (2011), são descritas duas experiências. Na primeira, com uma população de 122 alunos de licenciatura do quarto ano, os autores estavam interessados em analisar a usabilidade, levantar sugestões e comentários sobre o jogo como ferramenta de ensino. A segunda, com 65 estudantes do ensino médio, usou pré e pós testes para verificar as melhorias na aprendizagem. O grupo de controlo permaneceu com o método tradicional e o experimental utilizou o jogo. Como resultado da análise das médias do pós-teste, os autores concluíram que existem evidências de que o jogo contribuiu positivamente para a aprendizagem.

Chao (2016) tentou identificar o comportamento na resolução de problemas de 158 alunos de um curso introdutório de C++. As aulas decorreram durante sete semanas. Na oitava semana o professor aplicou o jogo (Figura 3.5) com a turma. O estudo identificou quatro grupos comportamentais de alunos que podem ser usados futuramente para adaptar o jogo e oferecer suporte respeitando essas diferenças individuais. O grupo dos alunos que usam uma abordagem sequencial tende a não usar estruturas avançadas e aninhadas. O grupo da abordagem seletiva tende a usar maior complexidade (desnecessária) dentro das estruturas de repetição. O terceiro grupo é composto por alunos que seguem a abordagem repetitiva que percebem padrões de repetição no código e conseguem criar estruturas simples para atender esses padrões. Finalmente, o grupo dos alunos com abordagem de consecutivas tentativas que costumam experimentar várias combinações de estruturas demonstrando dificuldades em implementar as estruturas de controlo.

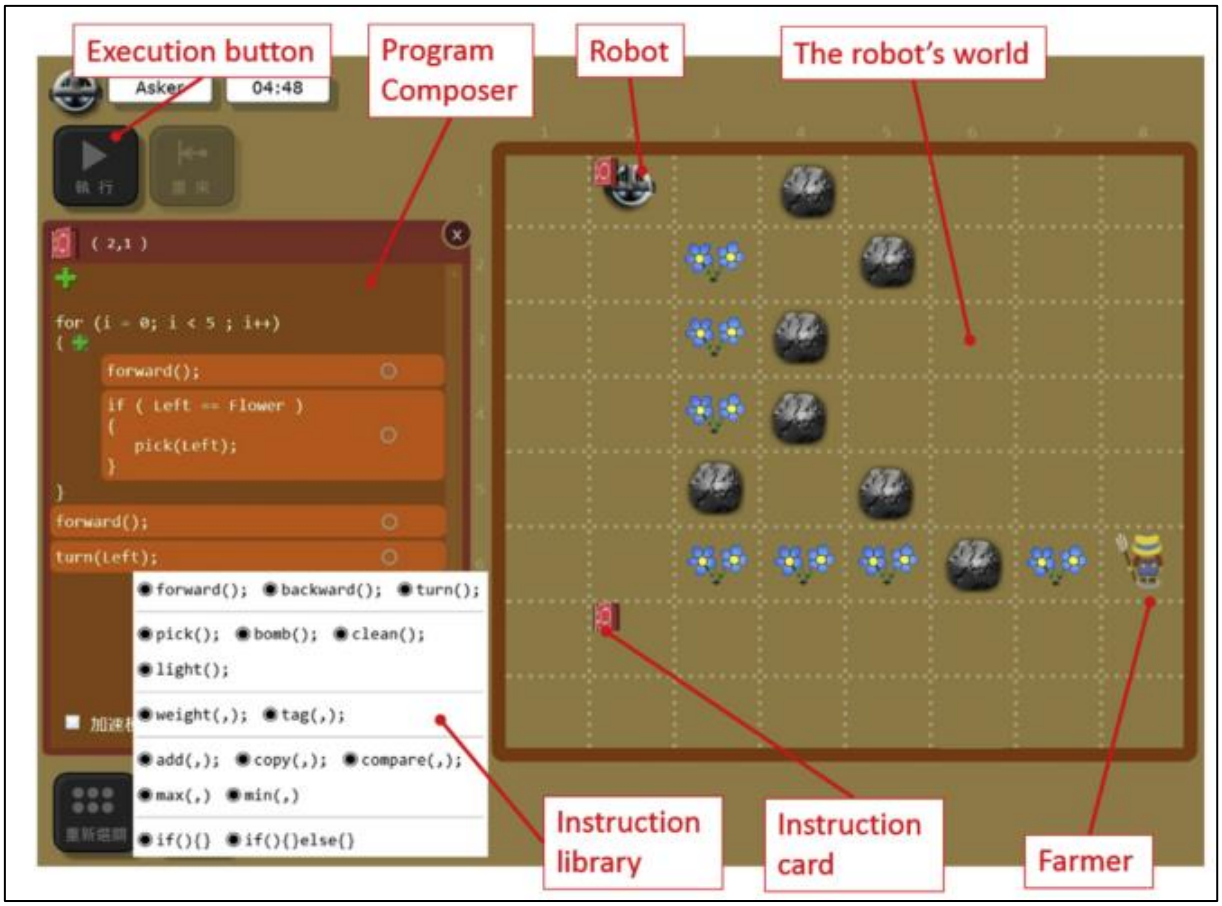


Figura 3.5 – Ambiente do Jogo (Chao, 2016)**, p. 5**

Os trabalhos de Piteira & Haddad (2011), Dim & Edson (2011) e Barbosa et al. (2011) não apresentaram resultados de experimentação.

Em relação aos jogos disponíveis para usar, sete deles têm versões exclusivas para a plataforma iOS. Quando uma turma apresenta restrições quanto à disponibilidade de equipamentos (*tablets* e *smartphones*), os jogos acessíveis pela Web são a melhor alternativa. Destacam-se três na Web: *LightBot 2* (Figura 3.6)*, Robozzle* (Figura 3.7)*, Code.org* e *Code Combat* (Figura 3.8). Os dois primeiros contam com funcionalidades muito semelhantes na forma como o aluno resolve os problemas, pois não possuem um comando específico para repetições, o que é resolvido através de chamadas recursivas às funções (Figuras 3.9 e 3.10). Além disso, ambos possuem editores de níveis, permitindo que o professor possa introduzir os seus próprios problemas. *Robozzle* tem a vantagem de ter versões também para iOS e Android. *Code.org* já foi discutido no capítulo anterior por ser um ambiente com várias opções de atividades, incluindo o tipo de missões discutidas neste capítulo.

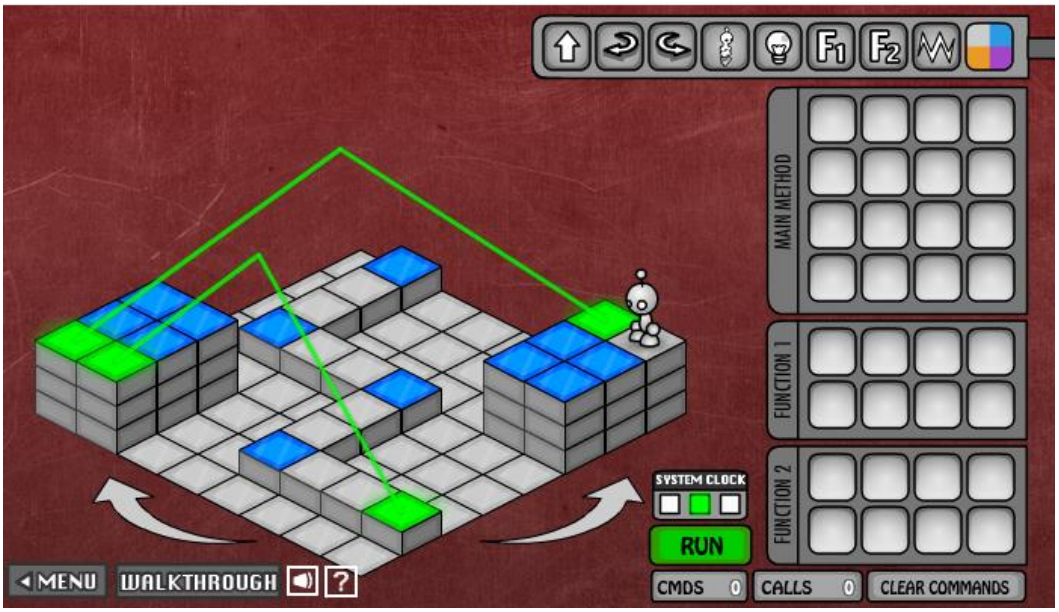


Figura 3.6 – LightBot 2

Figura 3.7 – Robozzle

Figura 3.8 – Code Combat

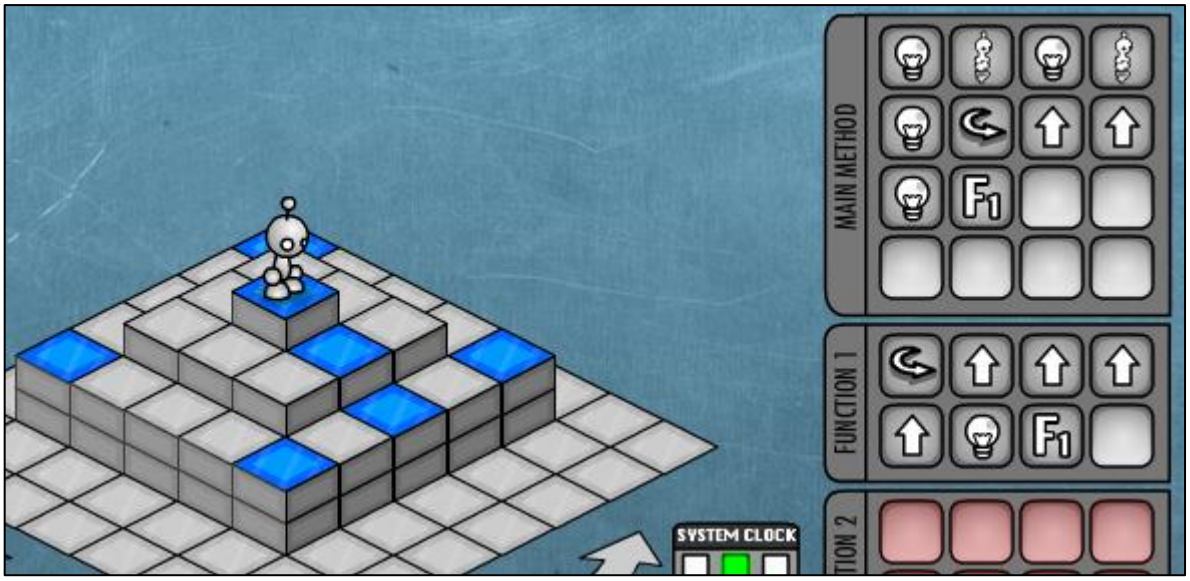


Figura 3.9 – Exemplo de recursão no LightBot 2

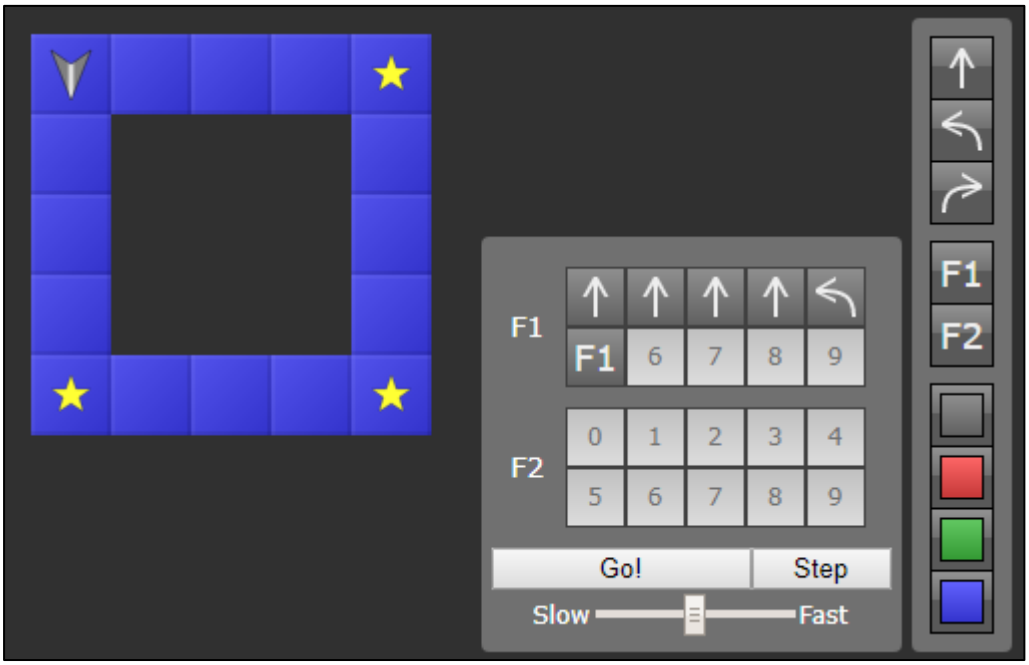


Figura 3.10 – Exemplo de recursão no Robozzle

Code Combat evoluiu desde 2014 para se tornar num jogo para um ambiente de aprendizagem, tanto que mudámos a sua classificação de um jogo de aventura para um jogo LOGO-Like. O jogo descreve um mundo medieval em que o herói luta contra ogros. As personagens interagem com o ambiente à medida que o jogador programa os seus comportamentos. Essa interação e codificação é efetuada no mesmo ecrã. As ações das personagens são baseadas em movimentação, ataque, conversa, construção de cercas e invocação de magias. O jogo suporta programação em JavaScript, Python, CoffeeScript e Lua. Existe um modo para jogar independentemente das aulas, um modo para professor e outro para aluno. No primeiro modo existem 8 campanhas para aprender assuntos básicos de programação, desenvolvimento Web, desenvolvimento de jogos, e práticas avançadas de programação. No modo professor, criam-se salas de aula e acompanha-se o progresso dos alunos (mostra apenas que missões cada aluno terminou e o tempo gasto para concluir contra a média da sala para concluir). No modo aluno, cada um é associado a uma sala de aula criada pelo professor.

Em relação aos restantes 31 jogos (aventura e gerais), pode-se observar uma participação maior de trabalhos publicados, e maior disponibilidade para serem usados: 10 jogos são de uso livre para o público.

Como já mencionado, para promover a aprendizagem, o aluno necessita de praticar muito, com problemas de natureza diferente. Os jogos encontrados nas

publicações contêm poucas fases (e por consequência exercícios para praticar). Por sua vez, os jogos acessíveis ao público contam com várias fases. Por exemplo, *Machinist-Fabrique* tem 64 fases; as missões em *Code Spells* são baseadas em realizar tarefas para gnomos espalhados por uma vila, e podem-se contar mais de 20 visíveis.

Em relação aos objetivos da investigação, devemos destacar os jogos *Machinist-Fabrique* (Figura 3.11), e *Ruby Warrior* (Figura 3.12), pois são organizados para promover a aprendizagem de programação de principiantes. Os primeiros desafios são fáceis para o jogador entender a linguagem de programação e os recursos do jogo.

Figura 3.11 – Machinist-Fabrique

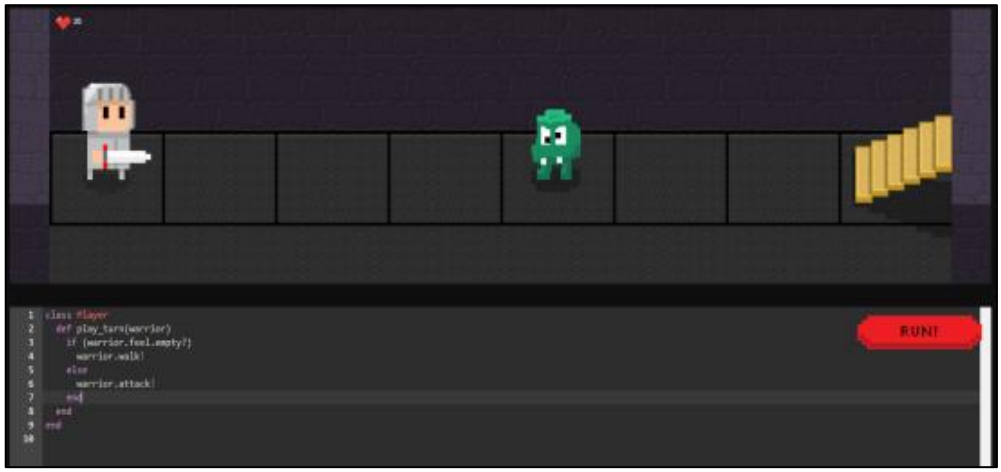


Figura 3.12 – Ruby Warrior

O jogo *Machinist-Fabrique* é baseado em desafios para o aluno programar o comportamento de máquinas. A programação é realizada com uma linguagem de blocos gráficos como no Scratch. As máquinas possuem eventos, guardam variáveis e a linguagem aceita condições e ciclos. Também se destaca por ser um jogo não violento.

Em *Ruby Warrior* o jogador programa o comportamento de um guerreiro inserindo código no método de uma classe em Ruby. O jogo serve-se de uma mecânica diferente de execução do comportamento, quando comparado com os outros jogos. Possui um ciclo principal que invoca de cada vez o método principal da classe. Logo, o aluno tem que programar o guerreiro tal como numa máquina de estados. A personagem possui ações para caminhar, olhar e atacar. Em cada iteração o guerreiro pode atacar ou caminhar uma única vez. Cada ação dessas gasta energia que desafia o jogador a cumprir a missão limitando a quantidade de código.

Quanto às publicações, 23 descrevem os recursos do jogo e somente seis descrevem a metodologia aplicada na experimentação, avaliação e validação da experiência de jogo. Connolly et al. (2012) analisaram as publicações de jogos sérios e perceberam a tendência em usar métodos quase experimentais e praticamente a ausência de estudos randomizados controlados. Esses estudos poderiam fornecer evidências mais fortes sobre o impacto dos jogos na educação, pois somente após um longo período de utilização se pode concluir concretamente sobre a melhoria da aprendizagem. A Tabela 3.2 apresenta a descrição das experiências dos seis artigos.

Tabela 3.2 – Análise de artigos que descreveram a experimentação

Autores	**Objetivos**	**Métodos**	**Resultados e conclusões**
(Chaffin et al., 2009)	Desenvolver um jogo para aprender a recursividade.	43 participantes. Aplicado um pré-teste somente a 16 participantes. Depois, para todos, 40 minutos de jogo em laboratório e, em seguida, um pós-teste e uma pesquisa para descrever a experiência. Ambos os testes tinham 5 questões e cada questão estava relacionada com o mesmo conceito/conteúdo entre os testes.	A comparação foi entre as questões dos testes, onde concluíram que com exceção de uma das questões, nas demais questões os alunos tiveram desempenho melhor. Concluíram que existiu uma significativa melhoria no desempenho. A diferença da pontuação do pós-teste entre ambos os grupos foi muito próxima indicando que não houve interação entre ambos os grupos. Analisando o *log* do jogo concluíram que não existiu correlação significativa entre o tempo gasto e a pontuação no pós-teste.
(Eagle & Barnes, 2009)	Desenvolver um jogo para ciclos e navegação em *arrays*.	92 participantes. Usaram a estratégia de amostra estratificada. Os grupos foram divididos em dois de acordo com um questionário demográfico e um pré-teste. Um dos grupos usou primeiro	Os pontos no pré-teste não foram significantemente diferentes para os dois grupos. Já os dois pós-testes demonstraram diferenças significativas em favor do grupo que teve o jogo como primeira tarefa, concluindo que o jogo contribuiu para a aprendizagem. A partir

		o jogo e depois o exercício tradicional, e o outro grupo realizou as mesmas atividades em ordem inversa. Após cada atividade foi aplicado um pós-teste. Os alunos tinham uma semana para fazer cada atividade e pós-teste. Toda a experiência foi conduzida à distância. Os logs do jogo foram enviados por e-mail aos pesquisadores.	dos *logs* puderam avaliar o tempo gasto e a quantidade de tentativas no jogo. Uma das questões demográficas pedia para o jogador se identificar como casual ou hardcore. Os jogadores hardcore alcançaram desempenho melhor no jogo.
(Kahwage et al., 2013)	Desenvolver um jogo para tipos de variáveis e operações.	28 participantes, 14 usaram o jogo. Houve uma sessão de 50 minutos de jogo em laboratório. Noutro instante foi aplicado um questionário com 7 questões sobre o conteúdo e para os 14 que jogaram, ainda mais 7 questões específicas sobre a usabilidade e funcionalidade do jogo.	80% dos alunos avaliaram a funcionalidade do jogo como boa ou muito boa. Não realizaram análise estatística para verificar a significância da diferença das médias. Mas os autores concluíram que como a média dos que jogaram foi superior o jogo melhorou a aprendizagem.
(Liu et al., 2011)	Entender a experiência de aprendizagem e exploração em atividades de resolução de problemas em programação.	117 participantes. Todos jogaram durante duas semanas em laboratório. Foi aplicado um formulário para avaliar o fluxo de interesse entre as atividades tradicionais em sala e as tarefas do jogo. Foi aplicado um questionário (MSLQ) para medir a motivação em aprender. Foi utilizado o *log* do jogo.	Através do questionário do fluxo de interesse puderam constatar que os alunos se sentem mais desafiados no jogo do que nos exercícios tradicionais. A motivação intrínseca é significantemente maior e a extrínseca menor usando o jogo. A análise do *log* serviu para investigar o comportamento, ou seja a sequência de ações que os alunos realizaram durante o jogo. Os autores apresentaram grafos direcionados onde os nós representam as ações (ler tutorial, reutilizar solução, desenvolver a solução, experimentar e rever a solução) e as arestas a sequência em que os nós são executados. Estes grafos representam os padrões de comportamento dos alunos ao resolver problemas. O artigo ilustrou três comportamentos de acordo com o sentimento do aluno em relação ao jogo: ansiedade, tédio e dentro do fluxo de interesse.
(Muratet et al., 2010)	Desenvolver um jogo que complemente as aulas sem modificar as escolhas da instituição.	15 participantes. A experiência de jogo aconteceu em dois momentos: no primeiro os participantes jogaram sem programar e no segundo aprenderam e usaram a API do jogo. Aplicaram um questionário para avaliar a motivação dos participantes quanto a jogar e quanto a aprender a programar. Filmaram e avaliaram o tempo despendido pelo professor durante as explicações. Aplicaram um pré-teste antes da experimentação e um questionário sobre a	O pré-teste foi comparado contra um exame da disciplina para posteriormente identificar melhorias nas habilidades quando comparado com aqueles que não usaram o jogo. Os autores não realizaram essa comparação, pois tinham somente 15 participantes, o que consideraram estatisticamente insuficiente. As observações durante a experiência serviram para identificar o nível de dificuldade das missões, no que concluíram que não estavam na ordem correta (da mais fácil para a mais difícil). O jogo pode ser usado com ou sem programação, porém 100% dos participantes concordaram que usar o modo programação do jogo aumentou a diversão. O jogo não teve boa avaliação quanto à estética e ao apoio

		usabilidade e entretenimento do jogo após. Durante a experimentação observaram e registaram a quantidade de missões completas e a quantidade de compilações e execuções em cada missão	assistência/suporte sobre os recursos do jogo.
(Debabi & Bensebaa, 2016)	Desenvolver um jogo para aprender algoritmos de ordenação.	33 participantes. Um grupo experimental usou o jogo por 30 minutos. Outro grupo teve o mesmo tempo com explicações sobre o assunto. Em seguida todos fizeram um exercício e o professor corrigiu.	A nota média do exercício foi muito superior no grupo experimental. Não realizaram análise estatística para verificar a significância da diferença das médias. Mas os autores concluíram que como a média dos que jogaram foi superior o jogo melhorou a aprendizagem.

De posse desta lista de jogos, podemos retornar às questões que nortearam este levantamento:

1) Que jogos estão disponíveis para ajudar na aprendizagem introdutória de programação publicados na literatura ou disponíveis na web nos últimos 10 anos?

Nesta lista existem 54 jogos classificados em três géneros: 23 jogos do tipo LOGO-Like, 11 de aventura e 20 do tipo geral. Podemos verificar que estão sendo investigadas alternativas ao tradicional comandar um robô de uma posição à outra. Ao mesmo tempo, observa-se que os jogos de aventura não são o tipo mais apropriado para suportar a fase inicial de aprendizagem de programação.

2) Quais são as competências, introdutórias ou mais avançadas, desenvolvidas com esses jogos?

Existem jogos que procuram desenvolver competências de compreensão e leitura de código, enquanto outros se destinam a melhorar a capacidade de identificar e resolver erros em programas. Mas a maioria dos jogos analisados está relacionada com construir uma solução para um problema. Os jogos com blocos estão mais relacionados com o apoio à iniciação à programação. Os jogos em que é usada alguma linguagem de programação são mais aconselhados para aqueles que já passaram pelo momento introdutório ao ultrapassarem as primeiras barreiras. Como já mencionado, a lista de jogos restringe-se àqueles em que a apresentação do problema e a correção da solução são realizados pelo próprio jogo. Não foram incluídos nesta lista as publicações e ambientes de autoria livre.

3) Quais são as lacunas que podem ser exploradas e não cobertas por esses jogos?

Qualquer ferramenta ou instrumento que venha a ser adotado por um professor para apoiar o seu ensino deve ter um ambiente de suporte para que possa acompanhar o desenvolvimento do aluno. Apenas dois jogos apresentam esse recurso: *Code.org* e *Code Combat*. Contudo, nenhum deles apresenta em detalhe a situação de cada solução submetida pelo aluno para permitir que o professor intervenha na aprendizagem individualizada. A maioria dos jogos não contém um sistema de pontuação ou conquistas, se pautando apenas por o aluno cumprir as tarefas das missões. Conquistar pontos e colecionar medalhas faz com que as pessoas se comprometam mais no jogo (Koster, 2014). Outra observação é que nenhum dos jogos considerou incorporar recursos para personalizar o seu herói. *Code Combat* usa um sistema para comprar equipamentos que fornecem novos métodos ao herói, porém não muda a aparência da personagem. O uso de avatares aumenta o sentido de presença, a satisfação e a motivação em vencer os desafios no jogo (Mazlan & Burd, 2011). Finalmente, como já mencionado no Capítulo 2, para se apropriar das competências necessárias para resolver problemas em programação, é necessário praticar muito. Pouquíssimos jogos oferecem (ou pelo menos descrevem nos artigos) mais do que uma dezena de desafios. É necessário fornecer vários problemas em diferentes formas de usar e aplicar as estruturas de programação no intuito de aumentar o âmbito das competências e habilidades na resolução de problemas.

3.7 Considerações Finais

Neste capítulo identificámos alguns princípios que direcionaram esta investigação. No que diz respeito aos jogos sérios, devemos incluir os objetivos e a avaliação da aprendizagem como elementos do próprio jogo, de tal forma que não ocorra uma interrupção na experiência de jogo com tarefas que teriam de ser realizadas num ambiente de desenvolvimento (IDE). Para aumentar a adoção do jogo por mais professores, deve dar flexibilidade, que pode ser conquistada através do desenvolvimento de um jogo seguindo os preceitos dos motores de jogos, em que eles possam criar os seus próprios desafios, ou modificar aqueles existentes. Como a

tarefa de aprender a programar é considerada difícil e exige muita concentração, um jogo sofisticado que exija mais esforço para aprendê-lo, pode causar um distanciamento do objetivo principal que é a aprendizagem, e por essa razão, focar-nos-emos no desenvolvimento de um jogo casual. Programar é construir artefactos que sejam exequíveis por um computador. Logo, as tarefas que um jogo para ensinar programação deveria oferecer, necessitam envolver atividades para criar algo, interagir e explorar com essa criação. Após a análise de 54 jogos, identificámos algumas lacunas que justificam esta investigação: a incorporação de um sistema de gestão do progresso do aluno, um sistema de pontuação (apesar de ser estranho, mas a grande maioria destes jogos não possui), personalizar a personagem, e oferecer muitas horas de prática diversificada no jogo.

Capítulo 4

PROCESSO DE INVESTIGAÇÃO

4.1 Considerações Iniciais

Este capítulo aborda o desenvolvimento e descreve os quatro ciclos de experimentação com o jogo. A segunda secção define os princípios da metodologia de investigação que foi adotada no desenvolvimento dessa tese. A terceira secção descreve a metodologia de projeto instrucional adotada para definir o projeto do jogo. A quarta secção apresenta a combinação de ambas as metodologias para orientar o desenvolvimento desta investigação. Na secção seguinte será apresentado o primeiro ciclo de desenvolvimento do jogo, chamado NoBug's SnackBar, onde primeiro é apontada a conceção inicial do jogo partindo de conceitos e comparações realizadas com ambientes e jogos descritos nos dois capítulos anteriores para fundamentar as decisões em cada uma das etapas metodológicas. Depois detalha cada uma dessas etapas, começando com a definição dos objetivos pedagógicos do jogo, seguida da escolha do género e mecânica do jogo integrado num modelo de jogo sério para a aprendizagem de pensamento computacional. Depois é apresentado o enredo do jogo, com o protótipo em papel e o seu teste com uma turma de licenciatura. A secção encerra descrevendo a avaliação do protótipo do jogo que foi a experiência piloto que aconteceu durante dois momentos de uma hora cada, em março de 2015, primeiro no curso de Mestrado em Educação na UC e depois no Bacharelado em Engenharia de Software na Udesc. A sexta secção apresenta o ciclo de desenvolvimento experimentado na UC durante três meses no curso de Licenciatura em Design e Multimédia entre setembro e novembro de 2015. A sétima secção descreve o terceiro

ciclo na Udesc durante dois meses no curso de Bacharelado em Engenharia de Software entre março e abril de 2016. A oitava secção explana o último ciclo na Udesc durante dois meses no curso de Bacharelado em Engenharia de Software entre agosto e setembro de 2016. Em cada um desses últimos três ciclos foi usada uma abordagem diferente no uso do jogo.

4.2 Metodologia de Investigação

Nas últimas décadas muito se tem investigado na utilização e desenvolvimento de tecnologias para as práticas de ensino-aprendizagem. Entretanto, uma numerosa quantidade dos resultados publicados apresentam uma pobre relação entre a teoria e a prática, em que os mesmos são aplicáveis em contextos muito específicos, ou ainda carecem de informações para serem reproduzíveis (Liu et al., 2011). As investigações geralmente são realizadas com um rigoroso processo de amostragem, em ambientes demasiado controlados, ou seja, as observações são realizadas sobre um conjunto de alunos num laboratório, fora do contexto quotidiano de aprendizagem desses alunos. Quando se trata de educação existem muitas variáveis envolvidas, que não podem ser estudadas isoladamente, pois se entrelaçam, e o seu conjunto é melhor que a soma das partes (Collins, 1992; DBRC, 2003; Squire, 2005b; Akker et al., 2006; Walker, 2006). As pesquisas em tecnologias na educação não têm somente o objetivo de observar e entender o processo cognitivo, mas também de gerar transformações nos processos e práticas de ensino, assim como criar produtos compreensíveis e aplicáveis em novos contextos (Brown, 1992; Akker et al., 2006).

A Design-Based Research (DBR) é uma metodologia de pesquisa preocupada com a aplicação pragmática das teorias de aprendizagem, visando estreitar a relação entre teoria e prática, através de exemplos tangíveis de aprendizagem, que podem ser usados e reusados no mundo real (Cocciolo, 2005; Squire, 2005b). Os pesquisadores estão interessados em demonstrar a utilidade dos meios tecnológicos, realizando a conexão entre a aprendizagem efetiva, o projeto curricular e a forma mais sensata de empregá-los.

A sala de aula é o ambiente ideal para realizar a pesquisa devido à naturalidade em que ocorrem as relações entre os alunos entre si, assim como com os professores.

Os métodos tradicionais de investigação em laboratório com intervenções individuais permitem verificar um padrão emergente que pode se confirmar na sala de aula (Collins, 1992; Cobb et al., 2003; DBRC, 2003; Squire, 2005b). Além do contexto da sala, o sucesso da investigação também depende da participação ativa dos professores envolvidos, considerando-os como coautores da pesquisa. Juntos descobrem as mudanças mais importantes com as práticas experimentadas (Brown, 1992; Squire, 2005a). O envolvimento dos professores em todas as fases otimiza o processo fazendo com que os ensaios sejam melhor projetados e evitam intervenções desnecessárias. A sua presença reforça o contexto da pesquisa em sala promovendo os testes com os alunos. Outra possibilidade é a participação ativa dos alunos, mesmo nas fases iniciais de planeamento, para proverem ideias quanto à natureza do produto que se pretende desenvolver (Collins, 1992; Cobb et al., 2003; DBRC, 2003; Wang & Hannafin, 2005).

As experiências devem ter uma duração de várias semanas para que se possa compreender as transformações cognitivas nos alunos e, assim, deduzir pequenas generalizações reproduzíveis com certa profundidade em situações similares (Majgaard et al., 2011). Não é possível observar mudanças significativas na aprendizagem se a intervenção é conduzida por um período muito curto (Brown, 1992; Squire, 2005b).

A DBR caracteriza-se como um processo iterativo e intervencionista (Brown, 1992; Gravemeijer & Cobb, 2006). A Figura 4.1 ilustra este processo cíclico de projeto, experimentação e análise dos resultados. As primeiras iterações apresentam poucos resultados e frágeis usualmente. Contudo, o aspeto iterativo da metodologia permite que as teorias evoluam, sejam ajustadas e otimizadas durante as intervenções (DBRC, 2003; Akker et al., 2006). Os pesquisadores vão experimentando as teorias através de protótipos até amadurecerem as suas ideias com uma teoria mais robusta. A evolução do protótipo, e por consequência das teorias, contribui para a compreensão das ações que levam ou não à aprendizagem (Walker, 2006; Majgaard et al., 2011). Esta compreensão acontece exclusivamente através das experiências. Estas produzem e são produzidas por novas teorias (DBRC, 2003) e acontecem sobretudo num contexto do mundo real (Squire, 2005b).

Com base no problema que se pretende resolver, são formuladas conjeturas teóricas que serão testadas através das intervenções. A intenção do investigador é criar condições para que essas teorias possam ser derrubadas (Brown, 1992),

determinar os limites na aplicação dessas teorias (Cobb et al., 2003), e, assim, fazer emergir os caminhos de aprendizagem (Squire, 2005b). Com base nos ganhos e perdas advindos dessas teorias, os pesquisadores constroem os seus produtos finais (Collins, 1992). Estes produtos podem ser a compreensão da influência dos ambientes no ensino e na aprendizagem, as práticas e recursos que devem constar em materiais instrucionais e a relação das didáticas com a metacognição (Walker, 2006).

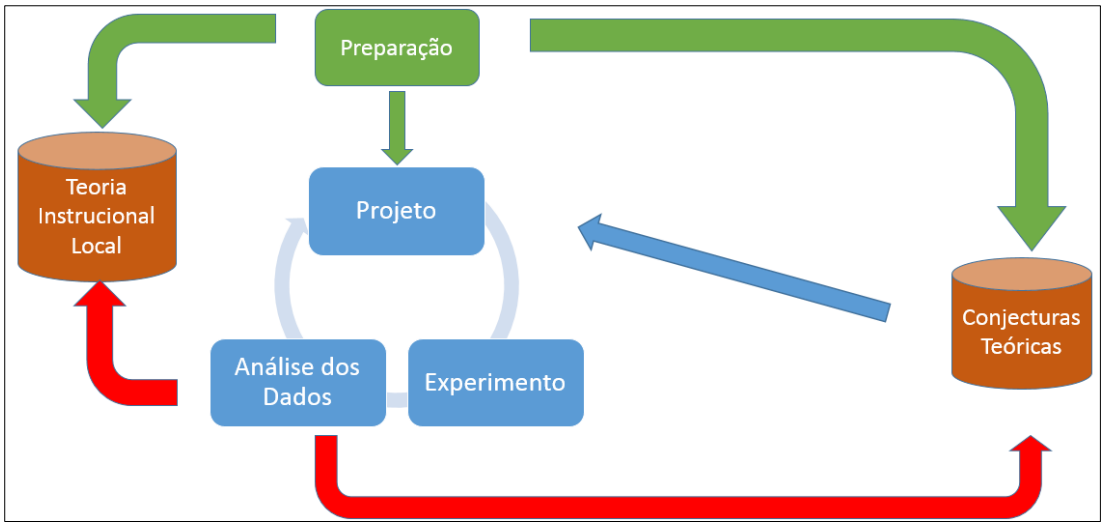


Figura 4.1 – Processo da DBR

Antes de se iniciar a pesquisa, existe uma fase de preparação. É o momento de definir uma teoria instrucional local que será a base de toda a investigação (DBRC, 2003; Cocciolo, 2005; Gravemeijer & Cobb, 2006). Esta teoria é a composição dos objetivos de aprendizagem de um domínio de conhecimento e as teorias de aprendizagem e meios (tecnologia) que darão suporte ao processo. Para elaborar os objetivos de aprendizagem, os pesquisadores utilizam o conhecimento do perfil dos alunos envolvidos na experiência e revisões da literatura, assim como avaliações de sala de aula de turmas anteriores para identificar lacunas que podem ser exploradas na pesquisa (Squire, 2005a; Majgaard et al., 2011). Desta forma, é possível determinar que mudanças se pretende alcançar nos alunos e, posteriormente, mensurar essas mudanças. Essas medidas são criadas e não descobertas, e a sua criação está entre os elementos mais importantes a serem definidos na pesquisa (Cobb et al., 2003; Wang & Hannafin, 2005; Gravemeijer & Cobb, 2006). As teorias de aprendizagem e os meios para darem suporte são criados a partir de estudos sobre como os estudantes e professores interagem para solucionar o problema que se pretende investigar (Cobb et al., 2003).

Como resultado desta etapa são elaboradas conjeturas teóricas que serão testadas durante as experiências (Walker, 2006). Elas não são estáticas, mas refinadas durante as próximas etapas. Cada experiência é composta de um conjunto bem limitado de conjeturas e objetivos, permitindo que o processo seja dinâmico e os resultados dessas etapas sirvam às etapas subsequentes (Cobb et al., 2003; Gravemeijer & Cobb, 2006).

A partir das conjeturas a serem testadas, determina-se o plano inicial de implementação, que é constituído pelas intervenções, que são ciclos repetitivos de projeto, experimentação e análise, com o objetivo da revisão, refinamento e generalização da teoria instrucional local, a definição dos meios necessários para aplicar essa teoria e compreender como funciona a aprendizagem (Wang & Hannafin, 2005).

Na etapa de projeto determinam-se (Cobb et al., 2003; DBRC, 2003; Akker et al., 2006; Gravemeijer & Cobb, 2006): a) as conjeturas a serem testadas (uma do conjunto original ou uma nova criada a partir da intervenção anterior); b) seleciona-se a amostra (alunos experimentados anteriormente, de um novo grupo, ou uma mescla de ambos); c) as regras para a aplicação da experiência (como será integrada no currículo, definir se realizada em laboratório ou sala de aula); d) o período e a duração da experimentação; e) o desenvolvimento ou modificações nos meios tecnológicos que serão usados; f) a definição das formas de avaliação, pré e pós testes quantitativos e/ou qualitativos. Sugere-se aplicar pré e pós testes a toda a amostra (experimental e de controle) e combinar uma análise aprofundada com alguns estudantes (Collins, 1992; Akker et al., 2006; Gravemeijer & Cobb, 2006; Majgaard et al., 2011). Os métodos qualitativos podem ser usados para observar tendências e consequências não previstas (Brown, 1992). As generalizações das teorias instrucionais são criadas a partir dessas tendências. Para confirmá-las, pode-se tentar reproduzir essas tendências com um grupo experimental em laboratório. Através dessa reprodução também podem ser encontradas novas conjeturas a serem testadas em sala de aula (Squire, 2005a). A análise quantitativa normalmente é usada em detrimento da análise qualitativa quando se pretende analisar amostras de grande dimensão (Brown, 1992).

A etapa de experimentação é iniciada com o esclarecimento dos alunos das intenções e objetivos dos testes. Os professores desempenham o seu papel naturalmente em sala de aula, auxiliando os alunos nas dificuldades (Brown, 1992).

Os pesquisadores observam como ocorrem as relações e comunicações entre os alunos e com o professor (Brown, 1992), pois dessas observações podem resultar explicações de como os alunos procederam para resolver os problemas. Também para entender o processo de resolução de problemas, quando da interação com os meios tecnológicos, os alunos devem pensar alto (Cobb et al., 2003). Todas essas interações devem ser monitorizadas e registadas para fornecerem dados para a próxima etapa.

A última etapa do ciclo, a análise dos dados, é o momento em que o conhecimento da pesquisa é gerado (Squire, 2005a). Todas as fases anteriores devem ser documentadas: tanto a preparação, como o projeto e a experimentação. Esses dados podem incluir vídeo e áudio das aulas, das reuniões e das entrevistas, antes e depois da experiência, registos das tarefas e avaliações realizadas pelos alunos, *log* das interações com os meios tecnológicos, anotações, e pré e pós testes (Cobb et al., 2003).

A maior dificuldade em administrar a DBR está nesta etapa. Durante a pesquisa são produzidas grandes quantidades de dados e somente uma pequena parte é aproveitada, havendo muito desaproveitamento e podendo, assim, influenciar negativamente a pesquisa (Brown, 1992; Gravemeijer & Cobb, 2006; Walker, 2006). Sugere-se que no momento de obter os dados já se façam anotações e destaques sobre a parte mais interessante ou significante observada e, dessa forma, ir catalogando e pré-selecionando a parte mais relevante dos resultados (Cocciolo, 2005; Wang & Hannafin, 2005).

No intuito de averiguar como a DBR pode ser conduzida com o desenvolvimento de jogos sérios, foram examinados os trabalhos de Squire (2005b), Magnussen (2008), Eseryel & Ge (2010), Shelton & Scoresby (2010), Majgaard et al. (2011), Wong et al. (2011) e Eseryel et al. (2012). Na sua fase de preparação utilizaram algumas das práticas descritas anteriormente, adicionalmente consultaram especialistas e literatura a respeito de desenvolvimento de jogos. Alguns relataram usar o mesmo grupo de alunos e outros tiveram disponibilidade de grupos diferentes durante as intervenções. Destes trabalhos, cabe destacar:

- Na fase de preparação usou-se intensivamente a prototipagem (papel ou digital), ou apresentação de *software* similares, envolvendo professores e por vezes também os alunos;

- Antes da versão ser experimentada em sala, passou-se por uma fase de versão alfa, testada por professores e/ou alunos;
- Os jogos foram ainda utilizados mais uma vez (versão beta), antes de projetar a sua integração no currículo;
- Os jogos sofreram grandes mudanças durante as intervenções, tanto a nível de jogabilidade e mecânica, como da forma como o conteúdo era abordado;
- Somente um dos trabalhos relatou a não integração do professor no processo e justifica isso como uma das principais razões do fracasso do projeto até aquele momento.

4.3 Metodologia de Projeto de Jogo Sério

As metodologias de projeto instrucional são utilizadas para guiar o processo de projeto e o desenvolvimento de vários tipos de médias para a aprendizagem (McMahon, 2009). O projeto instrucional é "a ação intencional e sistemática de ensino, que envolve o planejamento, o desenvolvimento e a utilização de métodos, técnicas, atividades, materiais, eventos e produtos educacionais em situações didáticas específicas, a fim de facilitar a aprendizagem humana a partir dos princípios de aprendizagem e instrução conhecidos" (Filatro, 2010). O projeto de jogos sérios é um processo de exploração das teorias pedagógicas aplicadas aos jogos para criar as melhores condições de ensino e aprendizagem (Perron & Wolf, 2009). Através desse processo identificam-se o que é necessário ensinar e as tarefas que o jogador necessita de completar para que a aprendizagem ocorra. Os resultados do projeto instrucional informam a equipa de desenvolvimento como apresentar a informação de uma forma que ajudará o aluno a compreendê-la (Iuppa & Borst, 2010).

A metodologia de projeto adotado nesta investigação segue a proposta por Marfisi-Schottman et al. (2010) que constitui um modelo com sete passos, como ilustrado na Figura 4.2. Além dos passos, os autores também incluem os atores responsáveis por cada um deles. Como esta investigação envolveu essencialmente uma pessoa (o próprio investigador), foi suprimida a representação e descrição dos atores. Os modelos consideram uma equipa de desenvolvimento do jogo. Serão

descritos mais simplificadamente os passos, considerando que este é um projeto de investigação envolvendo uma equipa minimalista, constituída pelo investigador e pelos seus orientadores.

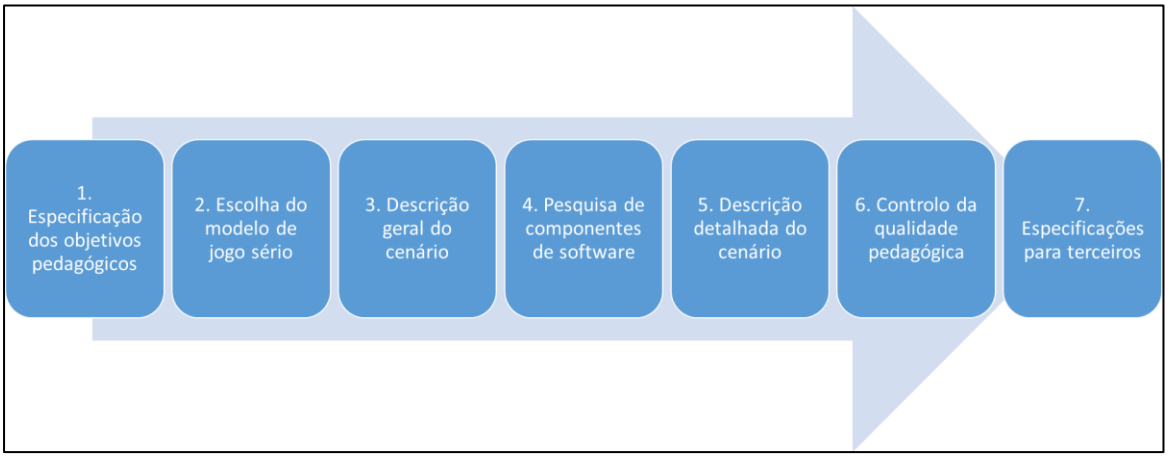


Figura 4.2 – Passos do projeto de jogos sérios. Adaptado de Marfisi-Schottman et al. (2010)

O primeiro passo é a especificação dos objetivos pedagógicos que consiste em definir o domínio de conhecimento, as capacidades e competências a serem aprendidas pelos alunos. O projetista identifica os resultados esperados na aprendizagem (memorizar, entender, aplicar, etc.) e que não devem ser confundidos com o objetivo do jogo, que é ganhar, uma vez que ganhar no jogo não está necessariamente relacionado com a aprendizagem (Van Staalduinen & Freitas, 2011; Lameras et al., 2017). Nessa fase também são definidas as formas de avaliar o progresso e a aprendizagem do jogador.

Em relação ao passo dois, a escolha do modelo de jogo sério está relacionada com o género, elementos e mecânicas que melhor se adequam aos objetivos instrucionais. Inclui as atividades de aprendizagem que se referem às ações que os alunos realizarão para chegar aos objetivos de aprendizagem (Filatro, 2008). Os tipos de conteúdos a serem aprendidos e as competências a serem adquiridas influenciam o tipo de jogo a ser usado (Prensky, 2005). Além disso, o projetista necessita considerar a componente diversão quando idealiza as atividades do jogo, pois o envolvimento consegue motivar os alunos a aprenderem sobre um assunto (Iten & Petko, 2016).

O enredo é tão importante para jogos sérios como não-sérios, pois permite que o jogador se projete na personagem do jogo (McDaniel et al., 2010). A história e os

cenários são projetados para promoverem a imersão e potencializar o interesse no jogo e no assunto.

No terceiro passo, os projetistas desenvolvem a ideia geral do jogo. Com os assuntos a serem abordados e o género de jogo especificados, já é possível definir com um pouco mais de detalhe as mecânicas do jogo. McDaniel et al. (2010) consideram três elementos que devem ser definidos para a narrativa do jogo: o contexto ou ambiente do jogo (onde acontece, quando acontece, o nível de fantasia e realidade, os tipos de conflitos, como são restringidas as ações da personagem, e como o progresso afeta a aparência do ambiente), a personagem (quem, qual o ponto de vista, a história é revelada no início ou desenrola-se durante o jogo, etc.) e a trama (qual é a trama principal, quais são as subtramas, qual o incidente que motivou a personagem a iniciar a jornada pela trama, etc.). Nesta fase, os roteiros e os fluxogramas em papel permitem testar e refazer rapidamente os conceitos chave do jogo (Schell, 2008; Iuppa & Borst, 2010).

A partir do protótipo é possível identificar os tipos de interações que existirão no jogo, e assim cumprir o quarto passo que é a procura por bibliotecas que possam satisfazer essas exigências. É mais produtivo desenvolver um jogo reutilizando componentes e partes já feitas, do que iniciar do zero. Também é possível realizar estudos para escolha de ferramentas de desenvolvimento e motores de jogos.

O quinto passo é a revisão e detalhe dos cenários considerando a tecnologia e os componentes adotados. É definido um fluxo como uma série de eventos para apoiar a aprendizagem (Filatro, 2008). É o momento de aplicar as teorias instrucionais e focar na forma de facilitar a transmissão do conhecimento (Van Staalduinen & Freitas, 2011). Os conceitos definidos no terceiro passo são detalhados e refinados para que se possa idealizar um protótipo funcional e testar as ideias em movimento.

No controlo de qualidade pedagógico (sexto passo) é feita a avaliação se todos os objetivos instrucionais foram considerados (Filatro, 2008) e se a mensuração do progresso e desempenho cobrem esses objetivos (Iuppa & Borst, 2010). Deve ser feito antes do teste com utilizadores reais, e depois, como um processo de evolução constante do jogo.

Um ponto a ser considerado é encontrar um *designer* para a equipa (Schell, 2008). Apesar de Squire (2005) ter verificado que os alunos consideraram mais importante os elementos da mecânica, a jogabilidade e os desafios do jogo do que a parte gráfica, essas afirmações foram feitas há mais de uma década, antes da

massificação dos dispositivos móveis e a popularização dos jogos nas redes sociais e dos vídeo jogos em 3D. Além da estética, outro recurso importante para imersão nos jogos é a música como elemento de motivação (Linek et al., 2012). Pode-se observar a necessidade de uma equipa multidisciplinar, e muitas vezes com membros externos para auxiliar a equipa de desenvolvimento. O último passo representa a comunicação com essa equipa multidisciplinar.

4.4 Metodologia de Desenvolvimento da Tese

No modelo de Marfisi-Schottman et al. (2010), o último passo do processo é o trabalho cooperativo com terceiros, como *designers* gráficos e músicos. Como não temos esses atores, inspirado nesse modelo, na prática realizámos um processo cíclico de desenvolvimento e definimos esse passo com terceiros como uma tarefa de um novo passo denominado **Desenvolvimento**, conforme pode ser observado na Figura 4.3.

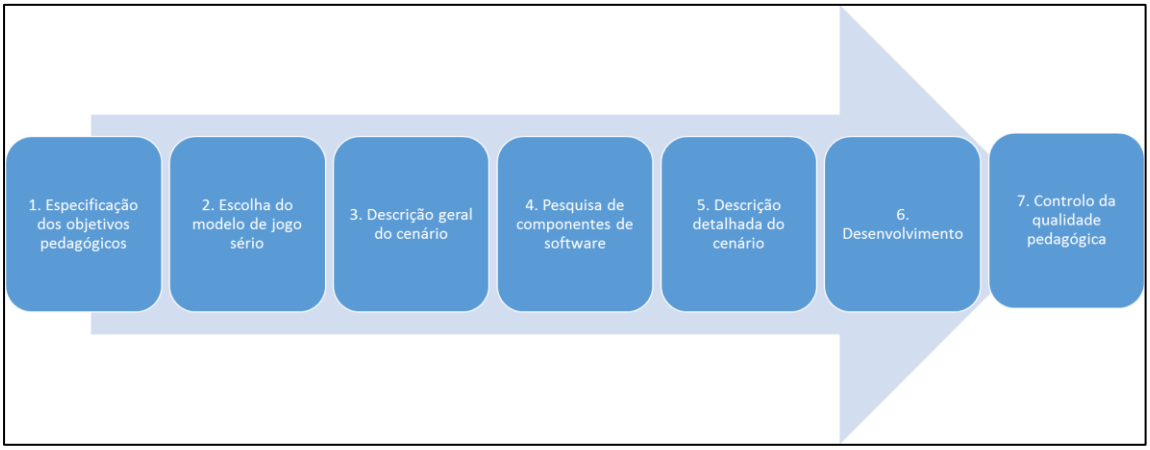


Figura 4.3 - Adaptação do modelo de Marfisi-Schottman et al. (2010)

Como essa investigação visa criar um artefacto tecnológico para ser inserido em um processo de ensino-aprendizagem introdutório de programação, e como relatado nos parágrafos anteriores sobre a validade da metodologia DBR para esse tipo de produto, a Figura 4.4 apresenta uma combinação de ambas as metodologias (Figuras 4.1 e 4.3) para orientar o desenvolvimento dessa investigação. A etapa de preparação da DBR englobou as duas primeiras etapas da metodologia de jogos

sérios. A partir de ambas, conseguimos definir os assuntos a serem abordados e a mecânica do jogo, formando as bases da teoria local e conjeturas teóricas. As demais etapas da investigação refinam e revisam essas bases. A etapa que se chamava “7. Controlo da qualidade pedagógica” foi desmembrada em duas etapas: a primeira é a etapa de experimentação que é quando ocorrem os testes, e a segunda etapa é análise de dados, que alimentam as bases de teorias instrucionais.

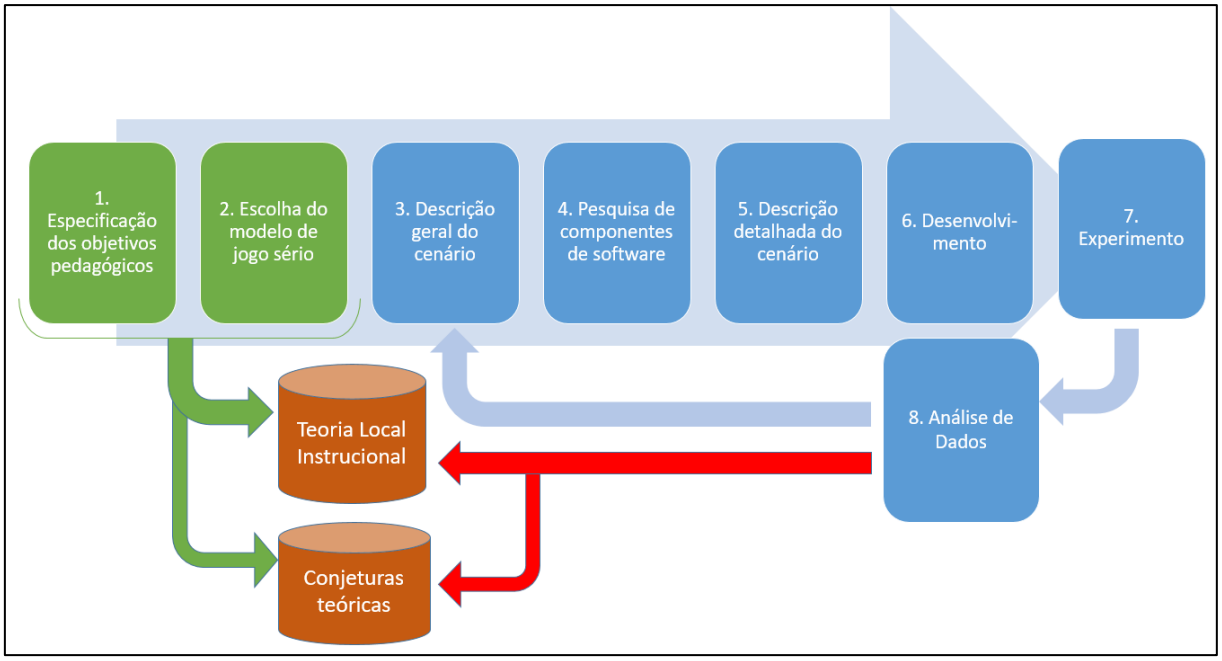


Figura 4.4 – Processo de desenvolvimento e investigação da tese

4.5 Primeiro Ciclo de Desenvolvimento

4.5.1 Conceção do Jogo

Com base na análise dos jogos mencionados no Capítulo 3, onde foram identificadas algumas lacunas e oportunidades para a investigação de novos jogos para a aprendizagem inicial da programação, verificámos a carência de um jogo abrangente, que envolva a maioria dos principais conceitos de uma disciplina introdutória de programação, em que as tarefas sejam voltadas para o nível de dificuldade e exigência do público do ensino superior. Assim, idealizámos um jogo que adotará a abordagem de Programação Baseada em Blocos, devido às vantagens citadas no Capítulo 2, em que a construção da solução é focada nos processos de

Pensamento Computacional e não nas especificidades das linguagens de programação. Outra vantagem na adoção desta abordagem é permitir que o jogo seja adotado independente da linguagem de programação lecionada pelo professor. Além disso, o jogo será uma aplicação *web* que será executada no navegador. Além de garantir facilidade no acesso por não exigir instalação de nenhum *software* adicional, ainda permitirá um controlo maior sobre as atualizações do jogo, que deverá ter uma grande periodicidade nas primeiras versões, e o uso pelos alunos, para que possamos monitorar o seu trabalho e intervir diretamente, ou indiretamente com mudanças no jogo.

Além disso, serão inseridos elementos que adicionam competitividade e imersão, como ganhar pontos e personalizar a aparência da personagem. Também pretendemos usar um nível diferente de contexto do que se costuma encontrar em produtos com movimento de robôs e tartarugas, oferecendo um contexto mais significativo e próximo do quotidiano dos estudantes desta faixa etária, como uma esplanada. Consequentemente, os comandos representarão ações de mais alto nível: em vez da personagem mover um passo ou virar à esquerda, ela poderá dirigir-se a um cliente ou ir preparar um cachorro-quente. Seguindo os atributos dos jogos casuais, as tarefas para serem concluídas exigirão alguns minutos, não sendo preciso explorar mundos, encontrar objetos ou aprender a usar alguma ferramenta para estar pronto para a tarefa. O jogador não precisará decorar comandos e nem usar uma interface complexa para resolver os problemas.

O jogo pretenderá promover a autonomia na aprendizagem do aluno sem a intervenção direta do professor. Para isso conduzirá o desenvolvimento do aluno através de uma sequência de aprendizagem e fornecerá 'dicas' e instruções sobre os seus erros e equívocos. Tanto as tarefas como a sequência de aprendizagem serão configuráveis pelo professor, assumindo assim este o papel de autor no jogo.

Como estratégia pedagógica, o jogo fornecerá diferentes tipos de tarefas conforme o avanço do aluno no planeamento instrucional. Esses tipos de tarefas podem ser usados para introduzir novos conceitos, apresentar boas práticas de programação até culminarem numa tarefa de construção da solução, que é o objetivo essencial da aprendizagem da programação.

Nas subsecções seguintes exploraremos a aplicação dos três primeiros passos do modelo de desenvolvimento de jogos sérios proposto nesta tese (subsecção 4.4).

4.5.1.1 Especificação dos Objetivos Pedagógicos

O objetivo do jogo é apoiar os alunos na introdução à programação. O foco é na resolução de problemas (aplicando os princípios do PC), e não na aprendizagem de uma linguagem de programação. Os estudantes devem aprender a manipular alguns constructos e estruturas de controlo essenciais para resolver esses problemas. A Figura 4.5 apresenta um grafo com os conceitos-chave de algoritmos conforme Mead et al. (2006). Os conceitos fundamentais estão em retângulos, e as elipses representam os conceitos transformativos. Conceitos fundamentais devem ser ensinados a partir do zero, aplicando modelos concretos ou situações reais. Conceitos transformativos agregam os conceitos previamente conhecidos integrando-os e reestruturando-os, e adicionando novos significados.

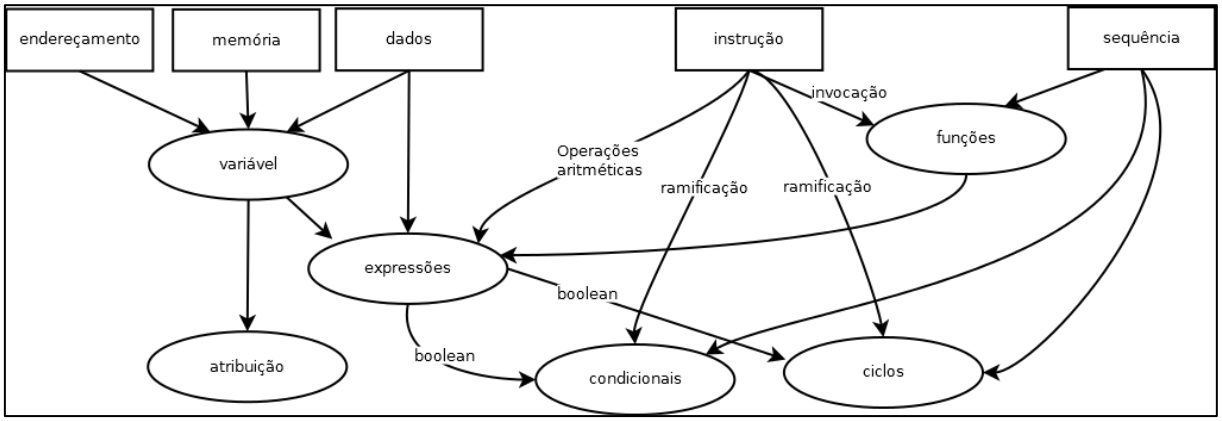


Figura 4.5 – Grafo de âncoras para os temas na aprendizagem de algoritmos. Adaptado de Mead et al. (2006)

Como mencionado anteriormente, a sequência e as atividades no jogo poderão ser criadas e definidas pelo professor. Entretanto, para que possamos experimentar e investigar, e assim avaliar as implicações do uso do jogo na aprendizagem dos alunos, inspirado no grafo da Figura 4.5, definimos os seguintes conteúdos instrucionais a serem disponibilizados nesta ordem: sequência, manipulação de variáveis, condicionais, ciclos, funções e *arrays*. Este último tópico não está no grafo, porém faz parte do currículo base para as disciplinas introdutórias de programação (ACM & IEEE, 2013). O jogo pretende formar um pensamento crítico e algorítmico no estudante, usando a abordagem do PC, e utilizando a PBB. Os constructos e estruturas da linguagem de blocos serão os instrumentos no jogo para o aluno resolver os problemas.

O tipo das atividades instrucionais deverá ser diferenciado dependendo do estado em que o aluno se encontra dentro de um dos conteúdos. Quando um assunto é introduzido, a atividade conterá vários blocos como sugestão. À medida que o aluno progride no jogo, adquirindo experiência e aprendendo boas práticas, desenvolverá as suas soluções do zero.

4.5.1.2 Modelo de Jogo Sério Escolhido

Jogos com tarefas para solucionar *puzzles*, jogos de simulação, estratégia, aventura, com vida artificial e de gestão de recursos ou tempo, são aconselhados para o desenvolvimento do raciocínio estratégico e tático (Prensky, 2005; Wassila & Tahar, 2012), pois usam técnicas de exploração, de aprender fazendo, baseadas em problemas e exemplos. A habilidade de identificar e conectar componentes para auxiliar o jogador a avançar, de formular e criar planos para vencer o próximo desafio, leva ao desenvolvimento das habilidades de resolução de problemas (Sherry & Pacheco, 2006).

Os jogos com a mecânica de gestão de tempo combinam com atividades em que um processo pode ser dividido em passos sequenciais dentro de um limite de tempo (Wassila & Tahar, 2012). O contexto do jogo é baseado em ambientes quotidianos, como um restaurante, uma loja de roupas ou um salão de barbeiro. O jogador assume o papel do empregado desses ambientes. Os clientes controlados pelo jogo fazem pedidos. O sucesso do jogador depende da sua agilidade em decidir qual a sequência de passos que consome menos tempo para atender esses pedidos. Esta sequência de passos é definida pela ordem em que o jogador *clica* nos elementos do jogo que produzem e combinam produtos intermediários para no final entregar o produto pronto ao cliente. O progresso no jogo é baseado em finalizar a maior quantidade de pedidos dentro de um limite de tempo. Ao finalizar um nível, o jogador recebe recompensas que podem ser trocadas por melhorias na sua loja. A cada novo nível, os clientes fazem pedidos mais complicados necessitando de novas e diferentes sequências de passos e combinação de itens (Trefry, 2010). A combinação de um ou mais elementos para criar um terceiro ou quarto elemento requer habilidades de sequenciação e resolução de problemas (Kapp, 2012). Como o objetivo do jogo é o aperfeiçoamento nas competências de resolução de problemas, decidimos adotar o género de jogo com mecânicas de gestão de tempo. O aluno controlará uma personagem para atender os pedidos dos clientes e para isso ele montará a solução

com blocos para definir a sequência de tarefas da personagem. A Figura 4.6 demonstra um modelo proposto para o desenvolvimento do PC com jogos digitais (Kazimoglu et al., 2013) em que se evidencia a prática da aprendizagem pela experimentação. Este modelo é composto por quatro componentes: o aluno, que é o agente que interage e recebe respostas do jogo; a interação, que são os elementos de aprendizagem no jogo, e, neste caso, as ferramentas para construir, executar e depurar a solução; o jogo em si, com as suas mecânicas, estética e dinâmicas, que materializam a solução do aluno e desempenham a animação e interação com outros elementos do jogo; finalmente, o *feedback*, que apresenta o sucesso ou aponta as deficiências e suas possíveis soluções, concretizando a aprendizagem através desses resultados.

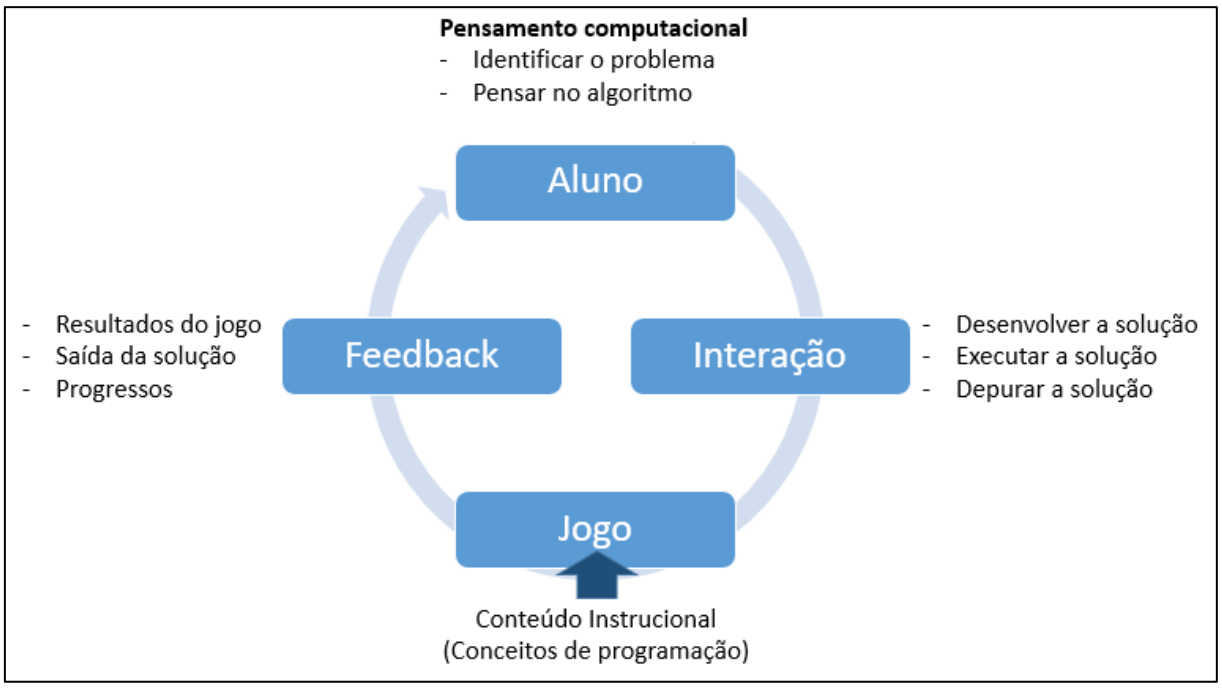


Figura 4.6 – Modelo cíclico de interação-feedback para desenvolvimento do PC em jogos digitais. Adaptado de Kazimoglu et al. (2013)

Em relação ao componente de interação, seguimos a sugestão, como já citado no Capítulo 2, de que um ambiente ideal de programação introdutória, entre outras características, deve ter um editor baseado em blocos e capacidade de executar passo-a-passo um programa (Xinogalos et al., 2017). O componente jogo seguiu as mecânicas de jogos de gestão de tempo no contexto de uma esplanada, pois é um cenário comum na vivência dos alunos, e assim não é necessário fornecer explicações adicionais para inseri-los no jogo. Em relação ao *feedback*, cada missão tem um conjunto de objetivos a serem cumpridos. Eles vão se destacando à medida que a

solução do aluno os vai cumprindo. Assim, o aluno pode acompanhar quanto ainda falta para terminar a solução. Além disso, no jogo haverá uma personagem, fazendo o papel de tutor do aluno, que fornecerá 'dicas' quando perceber que o aluno não está progredindo.

4.5.1.3 Descrição Geral do Cenário do Jogo

O jogo começa à mesa com uma conversa em família, com pai, mãe, filho e avó, onde o aluno assume o papel do filho. Nessa conversa o pai fala da sua insatisfação por o filho ainda estar em casa sem nenhuma ocupação. A conversa se desenrola com o filho dizendo que quer trabalhar numa esplanada, e o pai desafia-o, oferecendo-lhe o próprio negócio se ele se tornar um cozinheiro de sucesso. O ambiente do jogo replica uma esplanada, onde os clientes vêm com vontade de comer e/ou beber, e a personagem, comandada pelo jogador, prepara os lanches e as bebidas, usando equipamentos apropriados (frigorífico, espremedor de fruta, mostrador quente, etc.). A Figura 4.7a apresenta o protótipo em papel da esplanada, onde se pode observar o cliente sentado junto ao balcão, com o pedido dentro do balão. Existem dois equipamentos instalados na esplanada – o frigorífico e o mostrador quente – onde o encarregado (na cor verde) pode ir buscar o produto pretendido e depois ir até ao cliente entregá-lo. A solução em blocos é ilustrada em papel na Figura 4.7b.

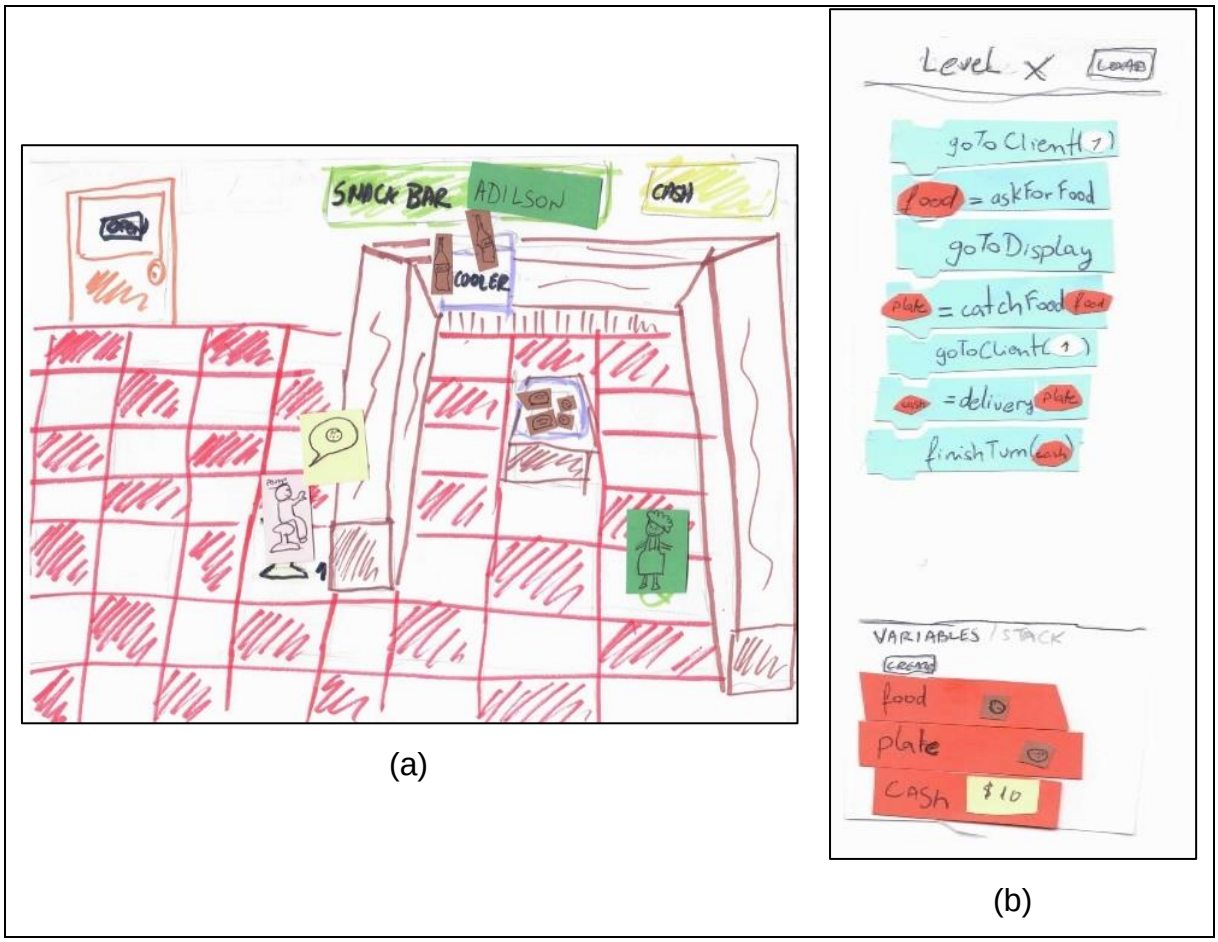


Figura 4.7 – Protótipo em papel (a) da esplanada e (b) da solução

As missões serão divididas em fases, onde cada fase está relacionada com um conjunto de tópicos, como, por exemplo, sequenciação e variáveis, ou tópicos num nível de profundidade, por exemplo, condicionais não aninhados e ciclos iterativos (*for*). A missão é selecionada pelo aluno, e, se for a primeira vez que entra no jogo, é apresentada uma janela com a descrição do problema a resolver.

No intuito de avaliar o protótipo em papel, foi apresentado a uma turma de programação do 2º semestre da Licenciatura em Engenharia Informática da UC (n=19) (Figura 4.8). Destes estudantes apenas 48% tinham obtido aprovação na disciplina introdutória de programação que tinha ocorrido no semestre anterior. O protótipo de papel foi organizado sobre uma mesa e interagimos com os alunos. Após uma breve explicação, foi colocado a eles um problema que tinham que resolver construindo uma solução com blocos. Em seguida foi adicionado mais um cliente ao problema, e os alunos resolveram-no e ainda procuraram uma forma de otimizar a solução.

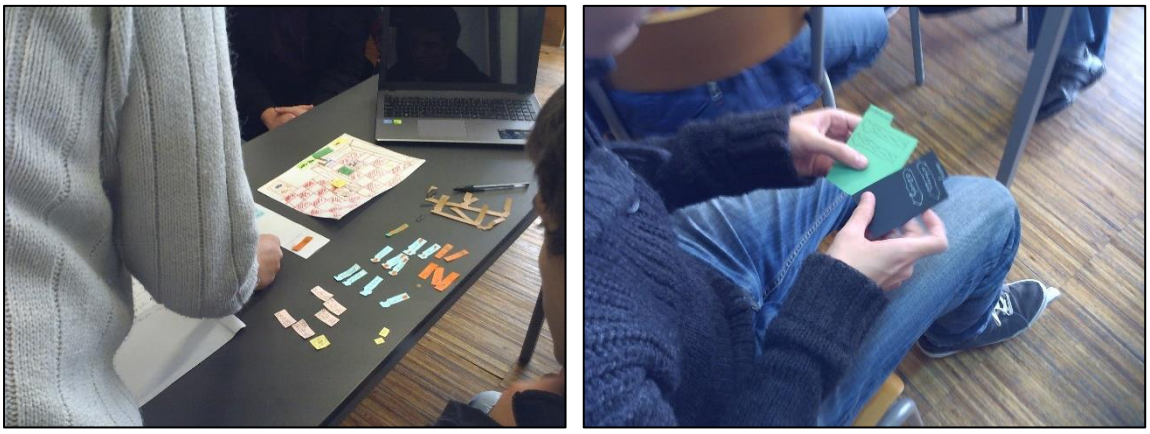


Figura 4.8 – Alunos manipulando o protótipo em papel

Foi entregue um questionário (Apêndice A) para identificar hábitos dos alunos e sugestões para o jogo e sua aplicação. A motivação para usar um jogo para apoiar a sua aprendizagem em programação é confirmada quando 79% dos estudantes afirmaram que gostariam que um jogo com estas características fosse considerado como um instrumento que o professor usasse em conjunto com outras atividades e somente 21% dos alunos sugeriu o jogo apenas como uma opção. Todos os alunos que ainda não tinham tido aprovação na disciplina introdutória de programação escolheram usar o jogo ou adicionalmente ou na mesma proporção que os exercícios convencionais. O sistema de blocos permitiu que os alunos pudessem resolver o problema, independente de dominarem alguma linguagem de programação. Os nomes das ações dos blocos foram compreendidos, tanto que inclusive otimizaram um código construído por eles próprios.

Tanto este inquérito, como outros estudos (Bothun et al., 2012; Entertainment Software Association, 2013; Carvalho et al., 2014), apontaram para o facto de que não é hábito da maioria da faixa etária do nosso público-alvo jogar em conjunto com outros. Logo, foi decidido desenvolver um jogo que possa ser jogado individualmente. Ao mesmo tempo, houve várias sugestões quanto à pontuação, em especial a criação de uma classificação conforme os pontos de cada jogador.

4.5.1.4 Escolha dos Componentes de Software

A necessidade de usar *plugins* para reproduzir conteúdos multimédia restringe o bom funcionamento das aplicações, pois os utilizadores podem estar com os seus navegadores com a autorização de descarregar e reproduzir esses *plugins* desativada, além da necessidade de ter o programa que os executa na versão correta

instalado no computador. Uma alternativa é o uso de HTML5, que vem com vários novos elementos da linguagem de marcação HTML, novas classes em JavaScript, e que podem ser executados nos navegadores independentemente de programas e autorizações adicionais.

O componente mais complexo no desenvolvimento do jogo proposto é o editor de blocos. No Capítulo 2 foram apresentadas três alternativas de componentes disponíveis: OpenBlocks em Java, Blockly e Waterbear em JavaScript. Escolhemos usar o Blockly, pois o JavaScript permite que se desenvolva o jogo com HTML5, os blocos têm melhor aparência e mais próxima do Scratch. O código é atualizado com frequência, porque essa biblioteca também é usada por outros projetos como o AIA e o Code.org.

Desenvolvemos um protótipo para experimentar o Blockly, e, com algumas modificações no código fonte de aplicações exemplos, conseguimos criar um ambiente com blocos para movimentar a personagem, usando inclusive variáveis (Figura 4.9). O componente permite transformar os blocos em código fonte, e acompanha com ele a tradução para JavaScript. Desta forma, o programa criado com blocos pode ser executado no navegador, sem necessidade de compilação e execução do lado do servidor, reduzindo a latência por dispensar a transmissão entre cliente e servidor.

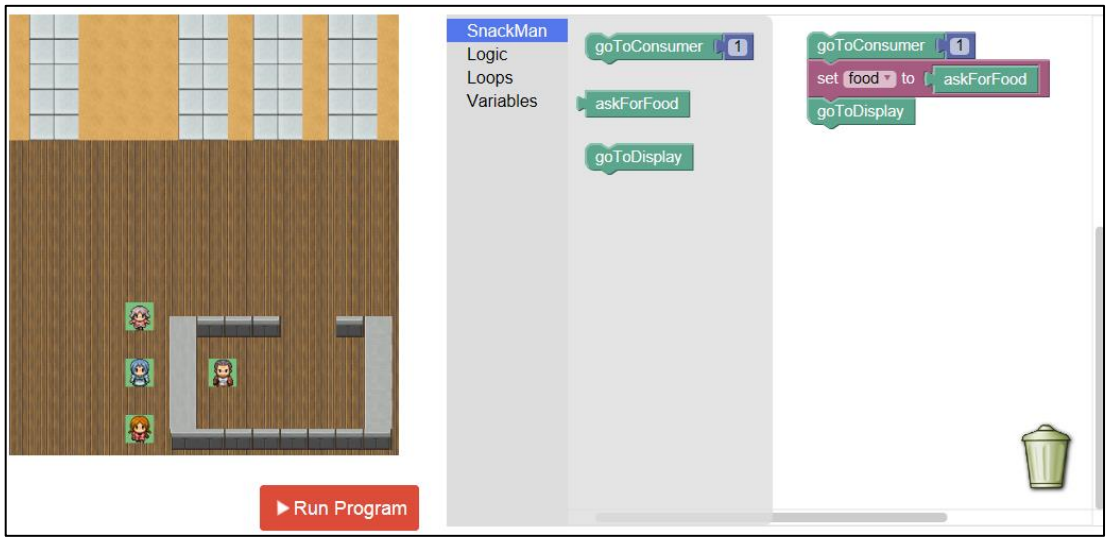


Figura 4.9 – Protótipo criado usando o Blockly

4.5.1.5 Descrição Detalhada do Cenário do Jogo

Nas subsecções anteriores foram definidos os objetivos educacionais (estipulando o conteúdo e a abordagem para resolver os problemas), o género, a mecânica e o enredo do jogo, e a tecnologia a ser usada para desenvolver o jogo. Nesta subsecção iremos apresentar o fluxo de navegação, detalhar alguns elementos do jogo, e apresentar a primeira versão.

A Figura 4.10 descreve o fluxo de navegação do jogo. O jogador inicialmente regista-se ou autentica-se. Em seguida é mostrado um mapa com as missões cumpridas, não cumpridas e prontas para jogar divididas em fases. Cada fase corresponderá a um conjunto de missões sobre um tema. A partir desse mapa o jogador tem noção da quantidade de tarefas que tem que realizar para finalizar uma fase e/ou o jogo. A partir dessa janela pode consultar as recompensas, distintivos e histórico do desempenho nas missões. Ao entrar a primeira vez numa missão, é exibida a descrição do desafio. Em seguida pode ir à loja comprar novos equipamentos para a esplanada. Depois desenvolve e testa a solução. O teste envolve executar ou depurar. No final do teste, o jogo avalia se a solução alcançou os objetivos esperados. Em caso afirmativo, a janela de seleção de missões é novamente aberta; senão, o aluno permanece no ciclo interação-feedback (Figura 4.6).

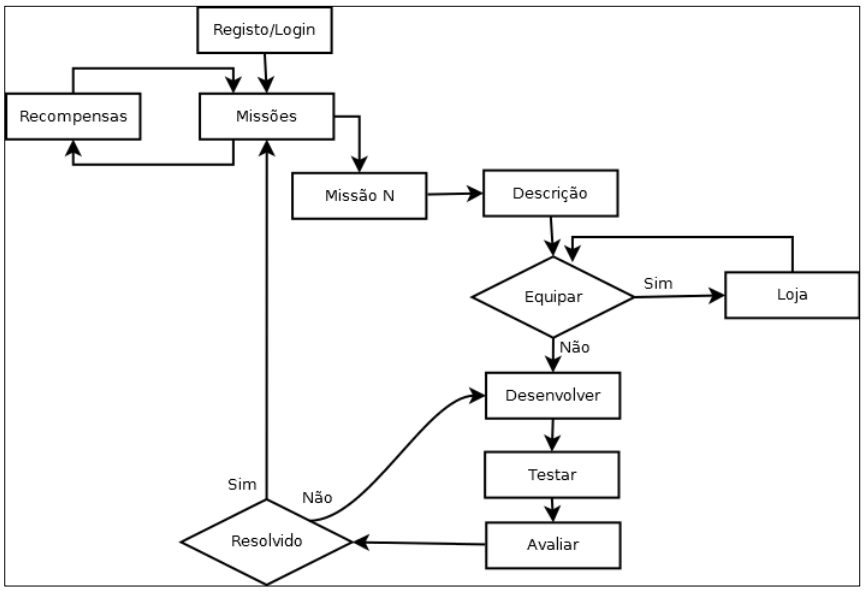


Figura 4.10 – Fluxo de Navegação Proposto do NoBug's SnackBar

O ambiente do jogo onde acontece este ciclo, na sua primeira versão, está ilustrado na Figura 4.11. No centro o aluno cria a sua solução com os blocos. À esquerda em cima está a área de animação, e em baixo estão os botões de controlo:

depurar, executar, interromper, comprar equipamento e consultar a descrição da missão. À direita está a lista das variáveis com seus conteúdos, que somente é apresentada quando o aluno depura. Na área de animação pode-se observar os três clientes, dois deles com pedidos, e a personagem junto do frigorífico.

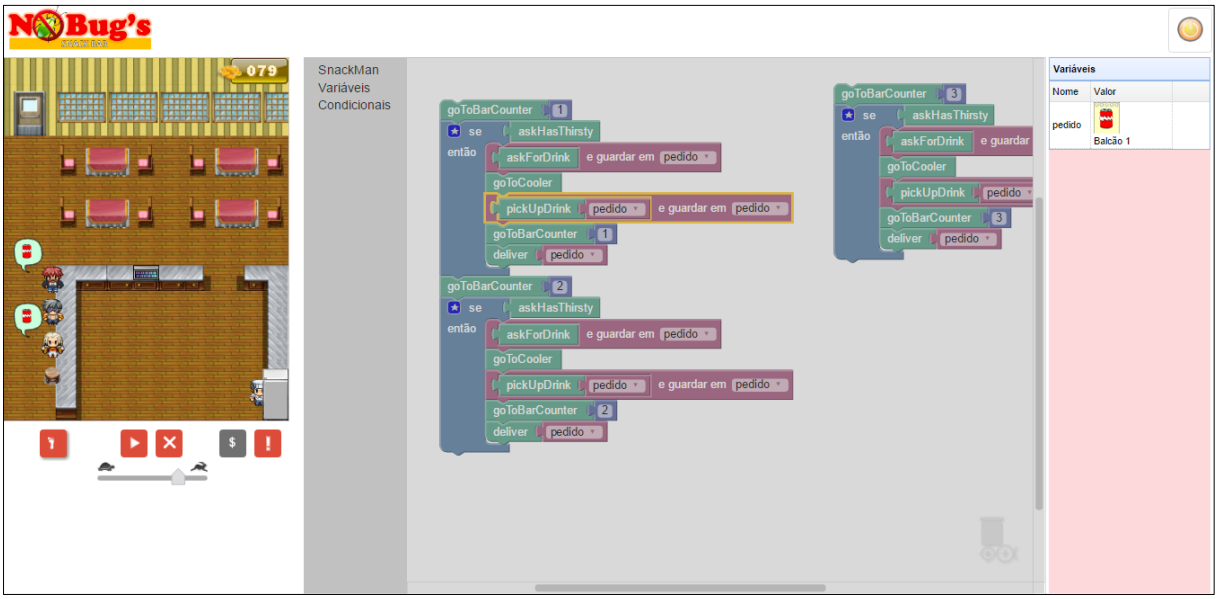


Figura 4.11 – Primeira Versão do Ambiente do Jogo NoBug's SnackBar

Os eventos Missões e Recompensas no fluxo de navegação foram implementados numa única janela, como ilustrado na Figura 4.12. As missões cumpridas estão a verde, a próxima a ser jogada a azul, e a vermelho àquelas que ainda não foram disponibilizadas. Observam-se as abas das fases: Aprendiz de Atendedor e Atendedor. Na lateral direita estão os jogadores classificados em três formas diferentes: a primeira pela pontuação acumulada, a segunda pelo tempo gasto para concluir as missões, e a terceira pela quantidade de tentativas para concluir as missões.

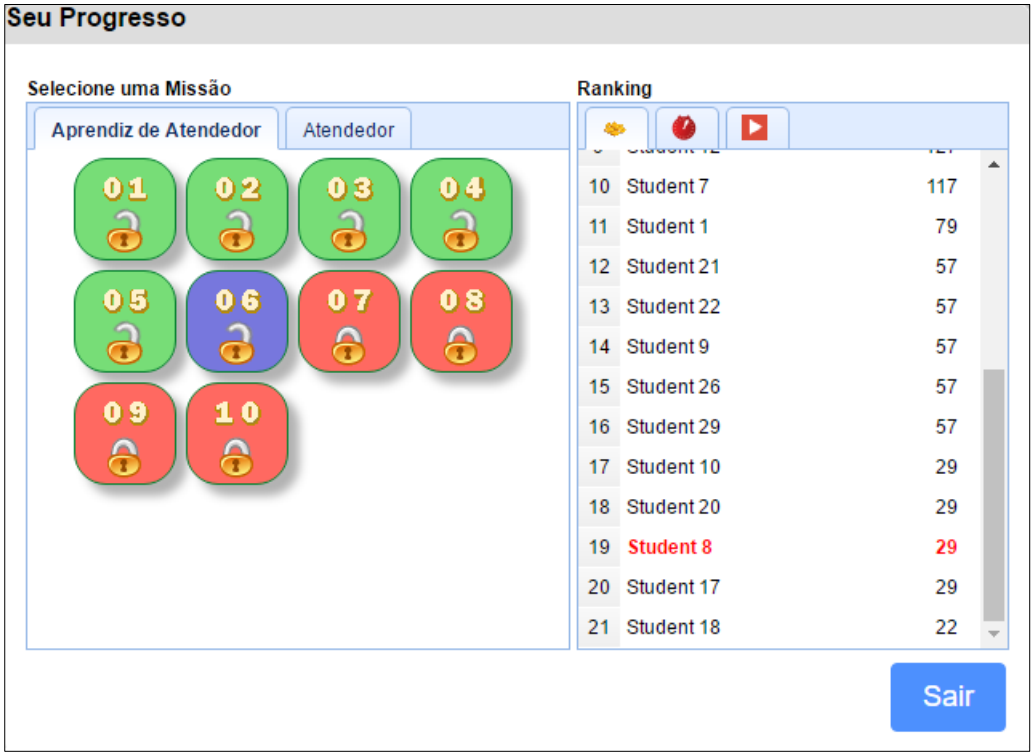


Figura 4.12 – Janela de seleção de missões e recompensas do NoBug's SnackBar

A janela de descrição das missões baseia-se em três estágios: primeiro apresenta algum texto para o enredo, depois explica um assunto novo, ou retoma algum já antigo, e, por fim, explana os objetivos da missão. A Figura 4.13 apresenta um exemplo dessa janela no último estágio. Os primeiros parágrafos explicam o desafio. Logo abaixo está uma lista de objetivos cumpridos ou por cumprir. Essa lista é útil para o aluno compreender quanto já cumpriu da solução, e quanto ainda lhe falta. No quadro amarelo estão expostos a pontuação que o aluno ganhará quando cumprir os objetivos. Mais abaixo pode-se ver um botão [Anterior] que implica que existam outras páginas antes dessa referentes aos estágios primeiro e segundo da descrição.

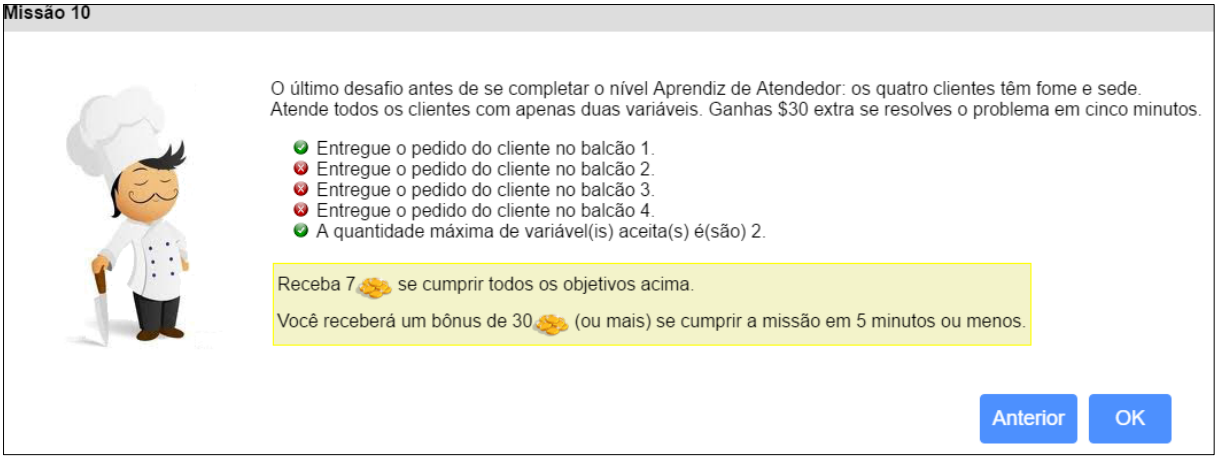
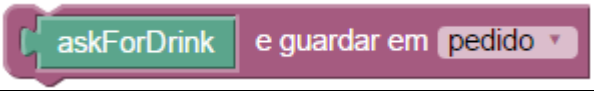
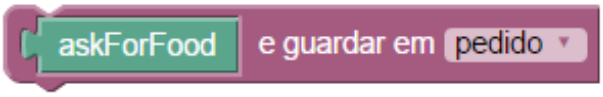
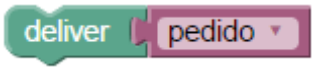
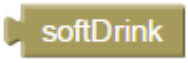


Figura 4.13 - Último estágio na janela de descrição de uma missão

Quando a solução termina de executar sem erros, o jogo verifica se todos os objetivos foram cumpridos. Os objetivos são baseados nas ações que a personagem pode fazer (Tabela 4.1). Logo, o jogo não compara a solução criada, mas se durante o decorrer de sua execução, a ação esperada foi executada, e se os parâmetros da ação contêm o valor previsto.

A Tabela 4.1 lista todos os blocos que foram criados especificamente no jogo para resolver as missões. Esses blocos dizem respeito a comandos e ações da personagem.

Tabela 4.1 – Lista de blocos de ação da personagem

Bloco	Significado
	Movimentar até ao cliente
	Perguntar o que deseja beber
	Movimentar até ao refrigerador
	Pegar uma bebida
	Perguntar o que deseja comer
	Movimentar até ao mostrador quente
	Pegar um cachorro quente
	Entregar
	Perguntar se está com sede
	Perguntar se está com fome
	Constante bebida
	Movimentar até à caixa de laranjas
	Pegar uma laranja
	Movimentar até à máquina de sumos
	Preparar e pegar um sumo

As missões da primeira versão estão ilustradas na Tabela 4.2. Na tabela está descrita para cada missão a tarefa do jogador, aquilo que se deseja que o aluno aprenda, e os blocos com que vai ter o primeiro contacto. Estas missões cobrem os assuntos de sequenciação, variáveis e condicionais. O projeto instrucional considerou um ritmo paulatino e incremental. Algumas missões obrigatoriamente usam o recurso de depuração para que o aluno valorize e entenda a sua importância. As duas primeiras missões são para o aluno se ambientar ao jogo. As oito missões seguintes focam-se em entender a sequência de instruções e a manipulação de variáveis. As últimas seis são sobre condicionais. Algumas missões têm restrições quanto à

quantidade de variáveis ou blocos, ou até se pode ganhar um bônus se resolver um problema com um limite de blocos definido na missão. Esse tipo de restrição ou incentivo servem para que os alunos pensem em resolver missões de forma otimizada. Nalgumas missões o aluno pode ganhar um bônus por resolver a missão até um determinado limite de tempo. Este tipo de incentivo serve para estimular a rapidez de raciocínio.

Tabela 4.2 – Missões da primeira versão

#	Tarefa	Objetivos Instrucionais	Novo bloco
1	Adicionar na área de trabalho dois blocos de movimentação até ao cliente.	Aprender sobre o ambiente do jogo: adicionar blocos, conectar um ao outro, a sequência de conexão, e executar a solução.	Movimentar até ao cliente
2	Os blocos já estão na área. Somente depurar.	Aprender sobre o ambiente do jogo: usar a depuração.	
3	Todos os blocos na área. Perguntar ao cliente 1 o que deseja comer e ir ao cliente 4. Somente depurar.	Armazenar valores nas variáveis.	Variáveis e perguntar o que deseja comer.
4	Perguntar a dois clientes o que desejam comer. Cada pedido numa variável diferente.	Armazenar valores em variáveis diferentes. Cada variável guarda o seu próprio conteúdo.	
5	Todos os blocos na área. Perguntar o que cliente deseja comer, ir buscar o cachorro e entregar ao cliente.	Sequência de instruções. Usar o valor da variável.	Movimentar até ao mostrador quente, pegar um cachorro quente e entregar.
6	Dois clientes a serem atendidos. Perguntar o que cliente deseja comer, ir buscar o cachorro e entregar ao cliente.	Reforçar a aprendizagem da missão anterior.	
7	Reduzir a quantidade de comandos da solução da missão anterior. Limite de quantidade de blocos.	Reconhecer que um mesmo problema pode ser resolvido de diferentes formas.	
8	Perguntar o que o cliente deseja beber e comer. Ir buscar o cachorro e a bebida e entregá-los. Ganha bónus em pontos se criar uma solução com uma certa quantidade de blocos.	Reforçar a aprendizagem de sequenciação e de manipulação de variáveis.	Perguntar o que deseja beber, movimentar até ao refrigerador e pegar a bebida.
9	Quatro clientes a serem atendidos, alguns com fome e outros com sede. Ganha bónus em pontos se criar uma solução até um limite de tempo.	Reforçar a aprendizagem de sequenciação e de manipulação de variáveis.	
10	Quatro clientes a serem atendidos, todos com fome e com sede. Ganha bónus em pontos se criar uma solução até um limite de tempo.	Reforçar a aprendizagem de sequenciação e de manipulação de variáveis.	

11	Três clientes e alguns com sede. Entregar a bebida para quem tem sede.	Usar condicionais.	Se-então e perguntar se está com sede.
12	Dois clientes que podem ter fome e/ou sede. Entregar o pedido de ambos.	Usar condicionais.	Perguntar se está com fome.
13	Dois clientes que podem ter fome ou sede. Entregar o pedido de ambos. Limite de quantidade de blocos.	Usar senão nos condicionais. Entender as redundâncias no código e como otimizar.	Se-então-senão
14	Dois clientes com sede de refrigerante ou sumo de laranja. Entregar o pedido de ambos. Ganha bónus em pontos se criar uma solução até um limite de tempo.	Reforçar a aprendizagem de condicionais.	Operador igual, constante refrigerante, movimentar até à caixa de laranjas, pegar uma laranja, movimentar até à máquina de sumos, preparar um sumo.
15	Dois clientes com fome ou sede de refrigerante ou sumo de laranja. Entregar o pedido de ambos. Ganha bónus em pontos se criar uma solução com uma certa quantidade de blocos.	Condicionais aninhados.	
16	Dois clientes com fome e/ou sede de refrigerante ou sumo de laranja. Entregar o pedido de ambos. Ganha bónus em pontos se criar uma solução até um limite de tempo.	Reforçar condicionais aninhados.	

A Figura 4.14a apresenta um trecho da solução da missão 15 e a Figura 4.14b apresenta a área da esplanada. Essa missão tem como descrição: *Agora os nossos clientes podem ter sede OU fome. E, se têm sede, podem pedir refrigerante ou sumo. Para resolver essa missão precisas usar um bloco de condicional dentro de outro*. O primeiro bloco serve para o atendente andar até a posição 1 do balcão do bar (representado por A na Figura 4.14b). Então pergunta ao cliente se tem sede. Em caso afirmativo, pergunta o que deseja beber e guarda o pedido na variável *pedido*. Em seguida o programa verifica se o pedido é um refrigerante. Em caso afirmativo, o atendente vai até o refrigerador (B na Figura 4.14b), pega a bebida de acordo com o pedido e armazena na variável *pedido*. Caso contrário, vai até a caixa de frutas (C na Figura 4.14b), pega a fruta de acordo com o pedido e armazena na variável *frutas*. Em seguida, vai até a máquina de frutas (D na Figura 4.14b), prepara o sumo e armazena na variável *pedido*. Caso o cliente não tivesse sede, então entramos num bloco para atender a fome. O atendente pergunta ao cliente o que ele quer comer e armazena em *pedido*. Depois vai ao mostrador-quente, pega o cachorro-quente de acordo com o pedido e armazena na variável *pedido*. Após finalizar esse bloco condicional mais

externo, o atendente volta até o cliente e entrega o que ele pediu. Nesse momento, se a entrega combina com o pedido, o jogador recebe dinheiro, e provavelmente cumpre uma das metas da missão. Caso contrário, o cliente fica com expressão de raiva, o jogador recebe uma mensagem de erro com uma explicação do problema e a execução termina.

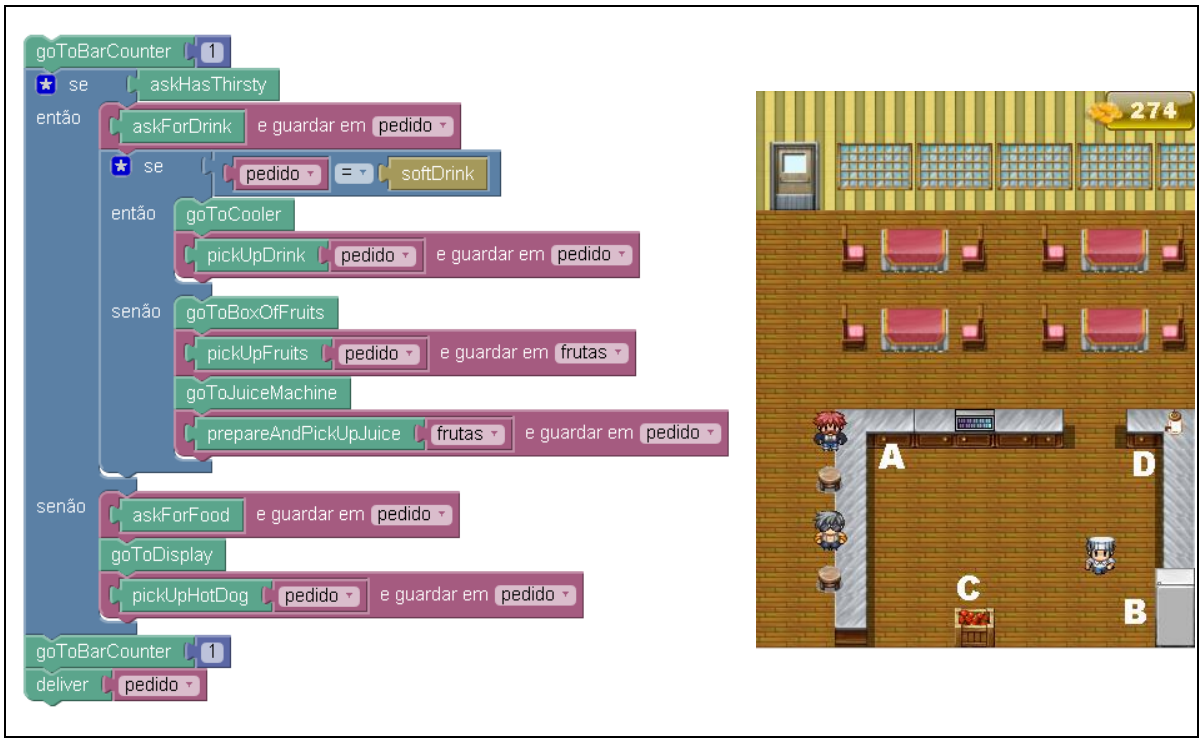


Figura 4.14 – (a) Blocos de código para missão 15; (b) a área da lanchonete

4.5.2 Avaliação do Primeiro Protótipo Funcional

Os objetivos desta avaliação foram **(1) mensurar o prazer dos alunos com o jogo** e **(2) identificar lacunas para melhorar a sua diversão**. Pretendeu-se também **(3) verificar se os desafios na sequência das missões ofereciam o sentimento de fluxo** descrito por Schell (2008). Não era objetivo aferir a aprendizagem, por isso, essa experiência teve a duração de apenas uma sessão para cada grupo de alunos.

Essa secção está relacionada aos seguintes trabalhos publicados:

- Vahldick, A., Mendes, A. J., Marcelino, M. J., Hogenn, M., & Schoeffel, P. (2015). Testando a Diversão em um Jogo Sério para o Aprendizado Introdutório de Programação. In *XXIII Workshop sobre Educação em Computação*. Recife, Brasil.
- Vahldick, A., Mendes, A. J., & Marcelino, M. J. (2015). Analysing the

Enjoyment of a Serious Game for Programming Learning with two Unrelated Higher Education Audiences. In *9th European Conference of Game-Based Learning*. Steinkjer, Norway.

4.5.2.1 População

A experiência aconteceu em dois momentos envolvendo duas populações com características completamente diferentes uma da outra entre áreas de conhecimento, países, géneros e faixa etária. Envolver dois grupos diferentes nos permitiu atingir pontos de vista mais amplos sobre o jogo e, provavelmente, mais diversos tipos de opiniões.

A experiência aconteceu no mês de março de 2015. Na primeira sessão participaram 19 alunos do Mestrado em Ciências da Educação (ME) da Universidade de Coimbra em Portugal, e na segunda 16 alunos do Bacharelado em Engenharia de Software (ES) da Universidade de Santa Catarina no Brasil.

A Tabela 4.3 apresenta os dados demográficos e hábitos de jogo dos dois grupos. Trinta e cinco (n=35) alunos participaram desse estudo. Podemos observar que o grupo ME essencialmente era formado por mulheres com média de 23 anos de idade sem nenhuma experiência prévia de programação. Entretanto, os alunos de ES eram homens jovens com alguma experiência prévia de programação. A maioria dos alunos jogava pelo menos semanalmente (62,86%) mas preferiam jogar sozinhos (65,72%). Essas são boas características da amostra para a experiência: primeiro, porque eram estudantes que gostavam de jogar, e segundo, porque nossa proposta de jogo não é multiusuário.

Tabela 4.3 – Dados demográficos e hábitos no primeiro ciclo

Variável	ME	ES	Total
Q1. Género			
Masculino	1	15	16 (45,71%)
Feminino	18	1	19 (54,29%)
Q2. Idade			
Média (Anos)	23.26	18.19	20.94
	±3.42	±1.94	±3.80
Q3. Qual seu conhecimento prévio em programação ou algoritmos?			
Nenhum	14 (73,68%)	8 (50,00%)	22 (62,86%)
Fiz algumas cadeiras	5 (26,32%)	6 (37,50%)	11 (31,44%)
Fiz um curso	0 (0.00%)	1 (6,25%)	1 (2,85%)
Trabalha profissionalmente	0 (0.00%)	1 (6,25%)	1 (2,85%)
Q4. Frequência que joga			
Diariamente	3 (15,79%)	4 (25,00%)	7 (20,00%)
Semanalmente	8 (42,11%)	7 (43,75%)	15 (42,86%)
Mensalmente	4 (21,35%)	3 (18,75%)	7 (20,00%)
Não tem hábito de jogar	4 (21,35%)	2 (12,50%)	6 (17,14%)
Q5. Frequência que joga com outros			
Essencialmente	3 (15,79%)	3 (18,75%)	6 (17,14%)
Metade das vezes	3 (15,79%)	3 (18,75%)	6 (17,14%)
Pouco	5 (26,32%)	6 (37,50%)	11 (32,86%)
Raramente	7 (42,10%)	4 (25,00%)	11 (32,86%)

ME – Mestrado em Educação; ES – Bacharelado em Engenharia de Software

4.5.2.2 Metodologia e Instrumentos

Cada aluno respondeu dois inquéritos: o primeiro (Apêndice B) para informar alguns dados demográficos e hábitos de jogos, e o segundo (Apêndice C) para identificarmos o tipo de jogador conforme a classificação de Bartle (2003). Em seguida, através de uma identificação única distribuída para cada aluno, eles puderam aceder e jogar individualmente *online* por uma hora a versão apresentada na secção 4.5. Na turma de ME a experiência foi acompanhada presencialmente pelo investigador, porém na turma de ES o professor da disciplina conduziu a experiência. No final da sessão, os estudantes responderam o inquérito (Apêndice D), adaptado do EGameFlow, para identificarmos quão divertido foi o jogo para eles. O inquérito incluiu ainda duas perguntas de resposta aberta para que os alunos descrevessem o que mais e o que menos gostaram no jogo.

Bartle (2003) classifica o comportamento dos jogadores em 4 tipos: socializadores (gostam de interagir com outros jogadores), exploradores (gostam de descobrir novos itens, cenários e personagens), empreendedores (gostam de interagir no mundo, conquistando objetivos, pontos e reputação) e assassinos (gostam de

interagir nos outros jogadores, ou seja, mudando o estado deles, por exemplo eliminando-os ou promovendo-os). Esta classificação auxilia os projetistas de jogos a determinarem elementos que divertem os tipos de jogadores que desejam atingir. Entretanto, um jogador não é totalmente de um tipo ou de outro, mas diverte-se com mais ou menos intensidade em cada um dos estilos. Por exemplo, pode-se classificar um jogador como Assassino-Explorador-Empreendedor-Socializador, indicando essa a ordem de preferência no comportamento dele.

Os alunos do ME não tiveram quaisquer aulas prévias de introdução à programação. Os alunos de ES experimentaram o jogo na sua segunda semana de aula com os assuntos dados de variáveis, expressões lógicas e condicionais.

4.5.2.3 Análise de Resultados

4.5.2.3.1 Fluxo de Jogo

A Figura 4.15 apresenta o tempo gasto (em minutos) por missão e a Figura 4.16 apresenta o percentual de estudantes que concluíram cada missão. Nesse gráfico, à medida que o tempo passa o número de estudantes diminui. Isto não significa que os alunos abandonaram o jogo, mas considera somente as missões que cada aluno completou e não aquelas em que estavam resolvendo quando encerrou o tempo da experiência. Os alunos de ES terminaram as missões mais rapidamente que o outro grupo, exceto nas missões 6 e 11. Uma diferença interessante foi observada na missão 7. Enquanto o grupo ES resolveu 60% mais rapidamente do que na sua missão 6 o grupo ME gastou 15% mais tempo do que na sua missão 6. A quantidade de tentativas na missão 7 também foi bem diferente: em média de 2,36 tentativas do grupo ES contra 5,94 do grupo ME. Essa comparação foi interessante porque o objetivo da missão 7 era otimizar o programa da missão 6 com um comando a menos. Os alunos de ES perceberam logo e com menos tentativas como resolver o problema. Não temos dados suficientes para fazer análise a partir da missão 11 porque somente 20% dos estudantes terminaram essas missões.

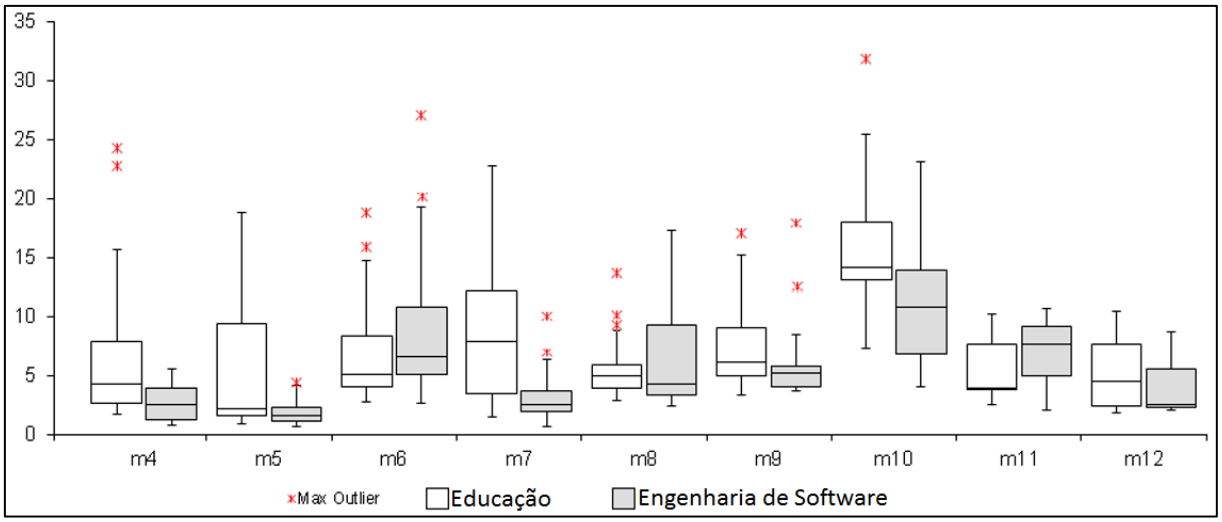


Figura 4.15 – Tempo gasto por missão

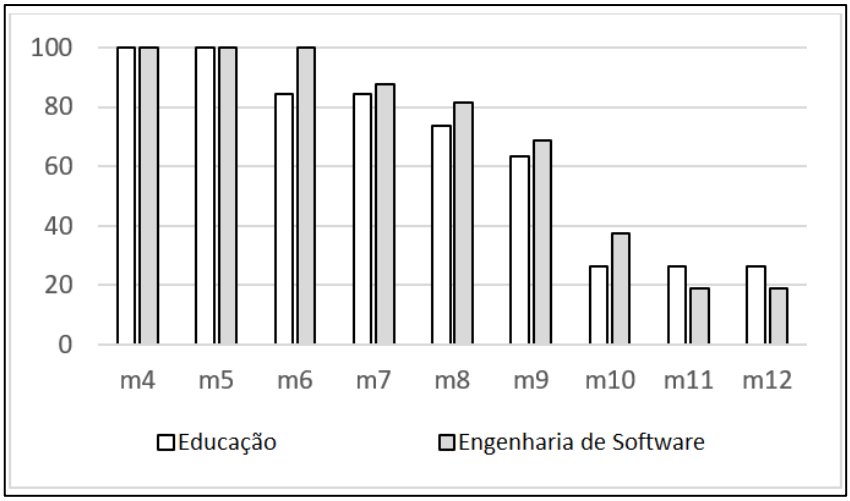


Figura 4.16 – Total de missões concluídas

Dois testes Mann-Whitney foram conduzidos para comparar o desempenho de ambos os grupos: comparando o tempo gasto em cada missão, e a quantidade de missões concluídas. Foram encontradas diferenças estatisticamente significativas no tempo gasto (p=0,001) e pelas médias (ES=91,20 e ME=118,01) podemos concluir que o grupo ES foi mais rápido do que o grupo ME (como esperado se comparando o perfil e a experiência). Entretanto, quando comparamos a quantidade de missões finalizadas entre os grupos, não foram encontradas diferenças estatisticamente significativas (p=0,756), permitindo-nos reter a hipótese nula.

Analisando a Figura 4.15 como uma curva de interesse (Schell, 2008), baseado nas medianas de cada caixa, nós podemos observar alguns picos e descidas antecipando a esperada subida na missão 10. Esta missão é a última da primeira fase, sendo mais extensa e complexa que as anteriores. O novo nível inicia com a missão

11 quando a curva inicia novamente. Não pudemos testar essa fase por completo por que são necessários mais do que uma hora para resolver as 16 missões criadas até agora. Observando a Figura 4.16, podemos detetar que todos os estudantes do grupo ES resolveram a missão 6, mas alguns deles não terminaram a missão 7. Analisando os dados, encontramos que dois estudantes que não finalizaram a missão 7 tiveram o segundo e terceiro piores tempos na missão anterior. Todos os alunos do grupo ME que finalizaram a missão 6 também finalizaram a missão 7.

4.5.2.3.2 Diversão Percecionada

Como gostaríamos de identificar alguma relação entre a preferência de jogo e a diversão percecionada pelos alunos, a Tabela 4.4 apresenta a distribuição de frequências dos alunos conforme o coeficiente de Bartle[14]. Dado o pequeno tamanho da amostra, neste estudo utilizamos apenas os dois primeiros estilos de preferência de cada jogador para ordenar a turma. Pode-se observar que a maioria (74,8%) dos alunos do grupo ES tem o tipo Assassino (K) como principal comportamento de jogador. Como não é um jogo violento, isso pode vir a estar relacionado com uma possível avaliação não positiva na satisfação do jogo. Novamente podemos observar a diferença entre os dois grupos: os alunos do grupo ME têm na sua maioria (47,3%) o tipo Explorador como preferência indicando a necessidade de constantemente apresentar novos itens para os jogadores. Como a amostra é menor que cinco em alguns casos, não foi possível executarmos testes de hipóteses para verificar a diferença na distribuição entre os grupos.

Tabela 4.4 – Coeficiente de Bartle dos participantes

	AE	EA	EK	ES	KA	KE	KS	SA
ME (%)	5 (26,3)	4 (21,0)	1 (5,3)	4 (21,0)	-	1 (5,3)	1 (5,3)	3 (15,8)
ES (%)	1 (6,3)	1 (6,3)	1 (6,3)	1 (6,3)	5 (31,3)	7 (43,5)	-	-

Os alfas de Cronbach dos inquéritos do EGameFlow nos dois grupos são considerados altamente confiáveis (α_{ME}=0.916 e α_{ES}=0.946). Na maioria das dimensões a média do grupo ES era maior do que o grupo ME (Tabela 4.5). A exceção foi na dimensão imersão. Foi conduzido um teste Mann-Whitney de amostras independentes para comparar os resultados das dimensões do inquérito entre os grupos. Não foi encontrada diferença significativa na média geral (p=0,1613) para rejeitar a hipótese que ambos os grupos apreciaram o jogo da mesma forma.

[14] O original em inglês é *Socializer*, *Explorer*, *Achiever* e *Killer*. Utilizaremos as iniciais em inglês para classificação dos jogadores.

Transformando a média para uma escala de 100, nós temos 76 para o grupo ME e 82 para o grupo ES. Isso sugere que o jogo deixou satisfeito a maioria dos jogadores em ambos os grupos.

Tabela 4.5 – Escala EGameFlow

Grupo	Concentração	Clareza dos objetivos	Suporte	Desafio	Autonomia	Imersão	Total
ME	3,8 (±0,50)	4,0 (±0,46)	4,1 (±0,54)	3,9 (±0,70)	3,4 (±0,95)	3,8 (±0,77)	3,8 (±0,59)
ES	4,1 (±0,46)	4,4 (±0,58)	4,3 (±0,57)	4,3 (±0,61)	4,2 (±0,63)	3,6 (±0,56)	4,1 (±0,48)

Os menores resultados foram encontrados nas dimensões de autonomia (grupo ME) e imersão (grupo ES). Autonomia está relacionada com o senso de controlo sobre as ações e escolhas (Sweetser & Wyeth, 2005; Fu et al., 2009). As questões abertas ajudaram-nos a entender esses resultados. Algumas respostas do grupo ME na questão "O que MENOS gostou no jogo?" foram:

"*Por vezes sentia-me confusa com o que tinha de fazer por ser muita coisa ao mesmo tempo*."

"*Demorei muito tempo para passar de nível porque por vezes não percebia as indicações dadas.*"

"*Quando as sequências são grandes torna-se difícil* "

Os alunos do grupo ME tiveram que aprender vários assuntos durante a experiência, como os conceitos de sequência de comandos e manipulação de variáveis. Isso leva a cometer alguns erros e algum sentimento de perda de controlo. Porém, o jogo tenta prevenir e recuperar-se de alguns erros através de mensagens de aviso (Federoff, 2002). Alguns comentários do grupo ME remetem-se a esse problema:

"*A falta de instruções quando não conseguia executar a tarefa corretamente*"

"*Não compreender completamente as indicações que aparecem quando faço algo errado*"

"*Quando não obtemos sucesso no desafio a informação não está claramente explicitada o que causa alguns obstáculos*"

"*Demorei muito tempo para passar de nível porque por vezes não percebia as indicações dadas*"

Isso sugere que deveríamos rever as mensagens de erros adicionando exemplos, contraexemplos e conteúdo instrucional. Entretanto, notamos que na maioria das vezes as mensagens foram ignoradas: o jogo mostrava e o aluno fechava a mensagens sem lê-la. Alguma dúvida podia ter sido sanada por essas mensagens.

Quando o aluno nos chamava para pedir uma ajuda, pedíamos que ele repetisse a ação com atenção a leitura da mensagem que ia ser mostrada pelo jogo. Se a dúvida ainda persistisse, podiam nos chamar novamente. Em todos os casos, o aluno não voltou a nos chamar.

Imersão se refere ao grau de envolvimento, engajamento e entrosamento dos jogadores com o jogo (Brown & Cairns, 2004). Apesar de Squire (2005) mencionar que os estudantes estão mais preocupados com as mecânicas do jogo do que a sua qualidade gráfica, houve respostas do que menos gostou referente à estética do jogo, uma no grupo ME e três no grupo ES. De acordo com Sweetser & Wyeth (2005), dois elementos de jogo que podem ser usados para promover a imersão são o áudio e a narrativa. Nessa versão o jogo não tinha som. Uma possível melhoria é a introdução de som ambiente e vozes das personagens. Adicionalmente, objetivos e as missões podem ser melhores integradas com o enredo do jogo.

Em relação às outras cinco dimensões, também encontramos algumas respostas com base nas questões abertas. A concentração é mantida através da provisão de estímulos que atraem e mantém a atenção do jogador, respeitando a carga de trabalho, reduzindo as tarefas e elementos não relacionados para cumprir a atividade principal. Sobre o que mais gostaram, 71,42% das respostas se referiam à dimensão concentração. Algumas das respostas foram:

"*O jogo é muito envolvente, você acaba entrando no jogo e dificilmente perde a concentração em que está fazendo*"

"*Ele é viciante e aprendo jogando com ele*"

"*O processo didático de como organizar minhas ideias, assim como na construção de um programa*"

"*A forma como o desafio me prendeu no sentido de completar a tarefa*"

"*No geral é um jogo bastante interessante e capta de imediato a nossa atenção*"

"*O facto de nos obrigar a pensar, ser lógico. Obrigou-nos a perceber qual a lógica do jogo para o podermos jogar*"

"*É divertido e viciante, a competitividade, as recompensas e as metas são uma motivação*"

Duas das três respostas negativas do grupo ME referiam-se quando a solução exigia muito código e a outra que o ambiente de sala de aula não é o ideal para concentrar-se.

A análise da curva de interesse (Figura 4.15) pode ser reusada para avaliar elementos de desafios. Apesar de termos concluído que essa curva satisfaz os princípios sugeridos por Schell (2008), a média de tempo na missão 10 é aproximadamente o dobro da missão exatamente anterior. Nós precisávamos reavaliar os desafios da missão 10 para reduzir essa média de tempo.

A clareza dos objetivos e o suporte foram as dimensões mais bem avaliadas nos dois grupos. Para a clareza não encontramos quaisquer respostas nas questões abertas. Entretanto, para o suporte encontramos quatro respostas positivas e oito negativas entre ambos os grupos. Possivelmente esses alunos associaram algum fracasso com os textos que explicavam os seus erros durante o jogo. De qualquer forma, nós precisamos melhorar o suporte quando os alunos falham em cumprir os objetivos.

4.5.2.3.3 Resumo dos Resultados

Havia três objetivos principais nessa experiência piloto:

(1) Mensurar o prazer dos alunos com o jogo

Usando as respostas do EGameFlow, ao observar os coeficientes (ME=3,8 e ES= 4,1) de maneira geral pudemos considerar que o jogo agradou. Mesmo os alunos de ES terem o perfil mais voltado a Assassinos, segundo a classificação de Bartle, eles acabaram por gostar um pouco mais que os alunos de ME. As dimensões com as menores médias foram a autonomia na turma do ME e imersão na ES. O sentimento de falta de autonomia na ME provavelmente deveu-se ao fato de que os estudantes não tinham quaisquer conhecimentos prévios sobre programação, e ainda como um aluno mencionou, o ambiente de sala não é o mais propício para se concentrar num jogo desse tipo. Os alunos jogaram um protótipo que precisava de muitas melhorias para torná-lo mais imersivo. Os alunos apenas jogaram uma missão após a outra, sem considerar pontuação ou quaisquer outros elementos que poderiam tornar um jogo mais divertido.

(2) Identificar lacunas para melhorar a sua diversão

Para o jogo ainda faltavam elementos básicos, como um sistema de pontuação, música e sons, e uma estória em que integre e dê significado ao jogo. Além disso, também faltava criar uma identidade própria tanto do jogo (usar figuras e imagens criadas por nós mesmos) quanto do personagem (permitir personalizar o

personagem). Quanto a parte educativa era preciso adicionar material instrucional para que os alunos pudessem ter um suporte adicional para compreender os assuntos sendo tratados no jogo e tomar cuidado para que boa parte das soluções das missões sejam pequenas que possam ser visualizadas no ambiente sem necessidade de *zoom*.

(3) Verificar se os desafios na sequência das missões oferecem o sentimento de fluxo

Avaliando os tempos gastos por missão, pudemos verificar que existem momentos em que os alunos finalizam rapidamente dando a sensação de controlo e outros que exigem um pouco mais de trabalho e concentração, inclusive para desafiá-los e que ocorra a aprendizagem significativo. Essa alternância torna o jogo nem aborrecido e nem frustrante.

4.6 Segundo Ciclo de Desenvolvimento

4.6.1 Modificações no Jogo

As modificações apresentadas nessa subsecção foram inspiradas nas observações e conclusões da experiência piloto.

Na versão alfa reutilizamos várias imagens e figuras disponíveis na web que são de outros jogos. Para estabelecer nossa própria identidade visual, criamos nossas próprias personagens, e redefinimos o aspeto das páginas. A Figura 4.17 apresenta a janela de abertura e autenticação no jogo, destacando no centro o logotipo para identificar o jogo.

Figura 4.17 – Janela de autenticação no Ciclo 2

Para melhorar a imersão no jogo, inventamos uma estória que é contada ao aluno na primeira vez que acede o jogo. Contamos a estória descrita na subsecção 4.5.1.3. A Figura 4.18 é apresentada para contar a estória

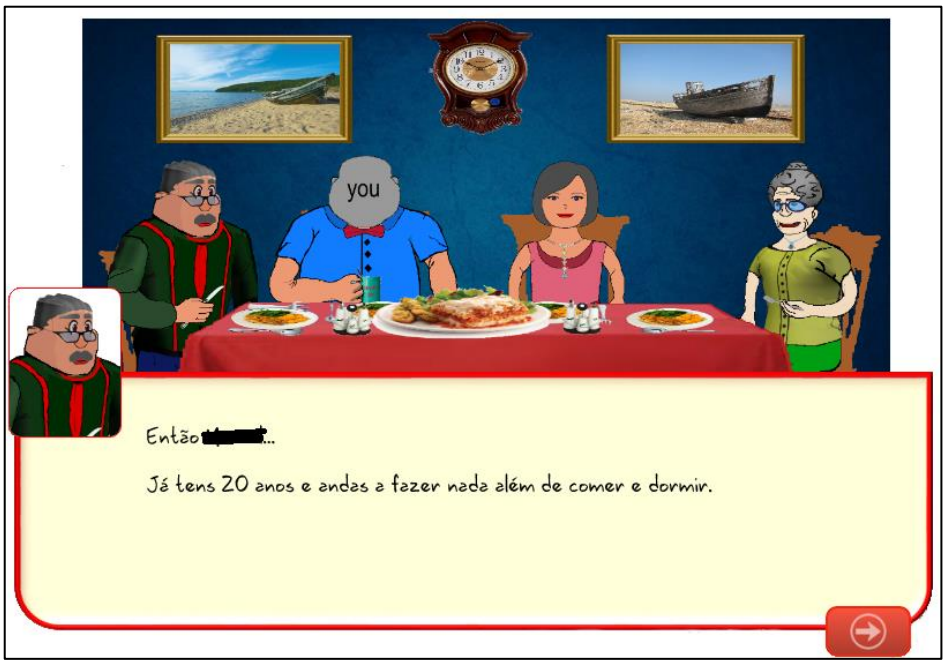


Figura 4.18 – O enredo do jogo no Ciclo 2

Passada a conversa, o ambiente é introduzido pelo Professor Burguer (Figura 4.19), que é a personagem de referência para as explicações, dicas e mensagens no jogo. O Professor Burguer desempenha o papel de um tutor e amigo que mantém o aluno informado e o auxilia nas dificuldades.

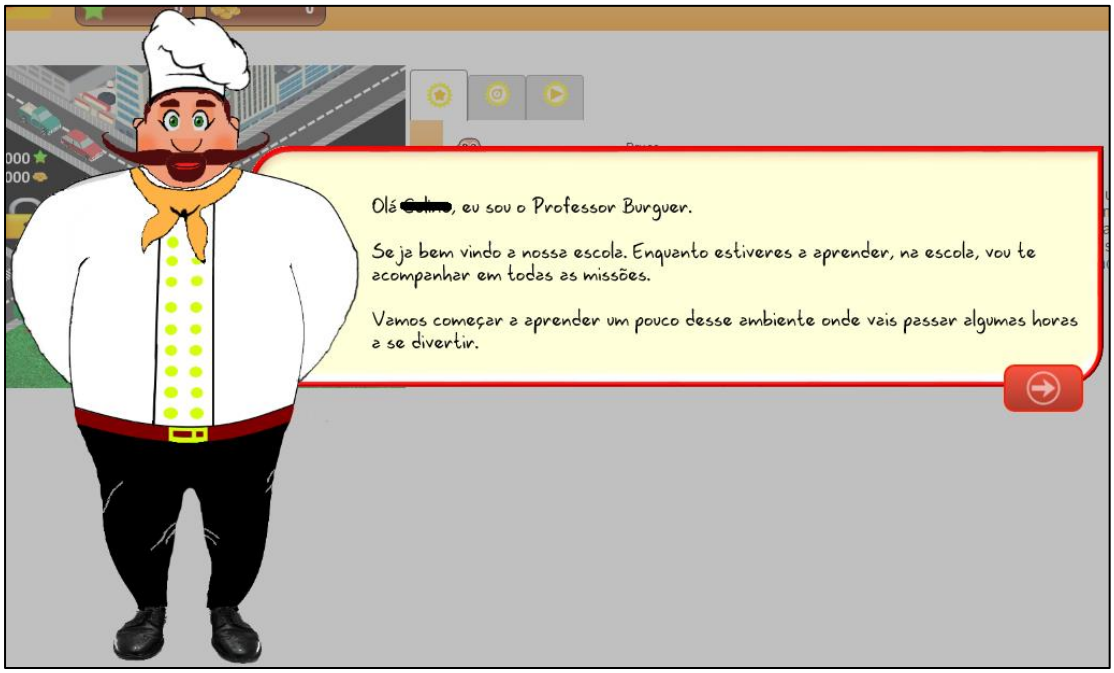


Figura 4.19 – Apresentando o ambiente do jogo no Ciclo 2

Redefinimos a janela de missões e recompensas (Figura 4.12) para se tornar um painel de controlo do aluno (Figura 4.20). No centro está a cidade com a escola de restauração e as áreas futuras que são abertas conforme o aluno vai acumulando pontos e dinheiro. Como a estória contou, a ideia é que sejam áreas onde o aluno vai poder abrir seu próprio negócio. Na verdade, não desenvolvemos essas áreas e a pontuação é impossível de ser conquistada para liberar essas áreas. Queremos mais tarde verificar se esse estímulo influenciou na experiência de jogo do aluno. Para entrar no seletor de missões (Figura 4.23), o aluno precisa clicar no edifício da escola. Foi adicionado outro item para auxiliar na imersão e justificar a acumulação de pontos e dinheiro: a personalização do seu avatar. A imagem do avatar do aluno é apresentada à esquerda, acima do botão para aceder a janela de configuração do avatar (Figura 4.21). O último botão serve para abrir uma janela (Figura 4.22) onde o usuário pode modificar sua palavra passe e seu correio eletrónico. Ainda no painel de controlo (Figura 4.20), à direita está a classificação dos alunos em três formas diferentes: a primeira pela pontuação acumulada, a segunda pelo tempo gasto para concluir as missões, e a terceira pela quantidade de tentativas para concluir as missões. Além do nome e do valor (pontos, tempo ou tentativas) também é apresentado o avatar de cada um, no intuito de criar uma relação de diversão ao apreciarem os avatares entre eles.

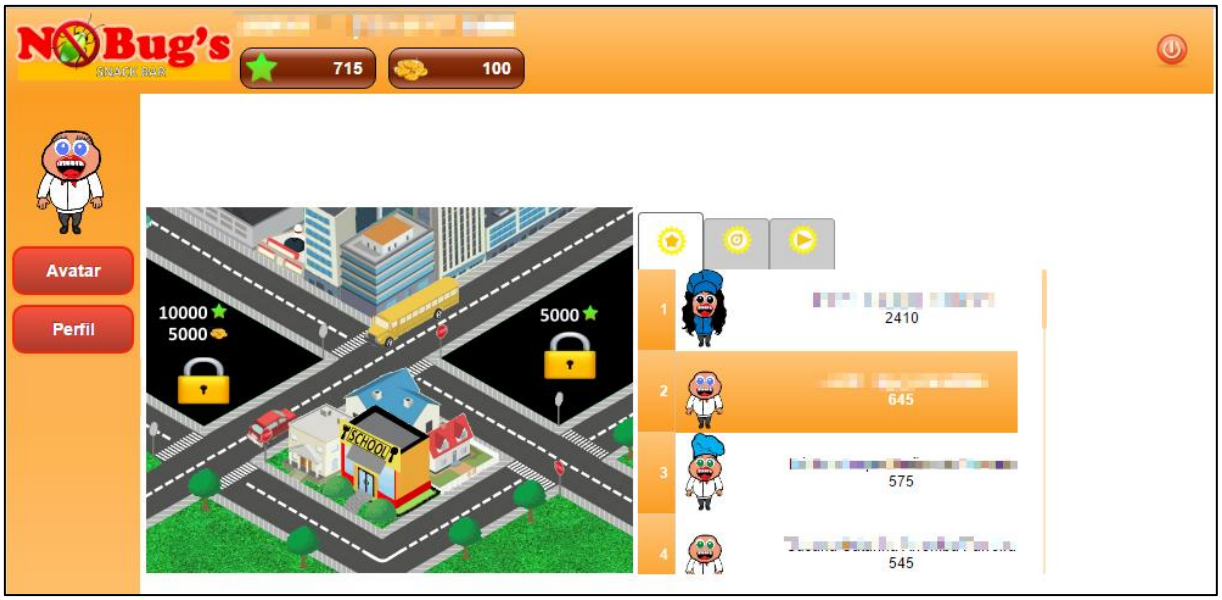


Figura 4.20 – Painel de controlo no Ciclo 2

Para o avatar podem ser configuradas características do corpo, chapéu e roupa (Figura 4.21). O avatar do aluno segue o seu género: se for feminino, os lábios e os cílios são maiores, além de personalizar o corte de cabelo; se for masculino, em vez do cabelo, o aluno configura tipos de bigode ou barba. Inicialmente cada avatar começa careca, e o aluno consegue somente configurar as cores dos olhos e da pele. À medida que ele vai conquistando pontos e dinheiro, novas opções vão sendo disponibilizadas.

Figura 4.21 – Configuração do avatar no Ciclo 2

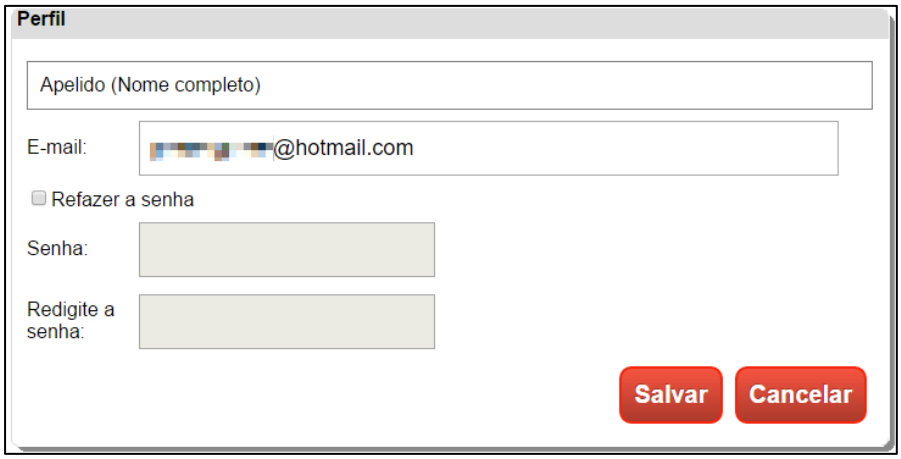


Figura 4.22 – Configuração do perfil do aluno no Ciclo 2

O seletor de missões (Figura 4.23) apresenta cada uma das fases do jogo. Em cada fase existem as missões concluídas (aquelas marcadas com um tique), as missões que podem ser jogadas naquele instante (coloridas com fundo em azul) e aquelas não concluídas (com a caixa vazia). O jogador pode aceder as missões já concluídas a título de consulta. Entretanto, ele ainda consegue experimentar novamente essas missões, alterando as soluções, executando ou depurando, porém, sem receber novas bonificações sobre essa experiência.

Figura 4.23 – Seletor de missões no Ciclo 2

Foram definidas 57 missões (algumas aproveitadas da versão piloto) organizadas em cinco fases com nomes referentes às funções dentro de uma esplanada: Aprendiz de atendente, Contador, Atendente, Aprendiz de Chef e Chef. Na fase 1 as missões foram dedicadas ao sequenciamento de ações, a fase 2 sobre

manipulação de variáveis, a fase 3 sobre condicionais, a fase 4 sobre ciclos e a fase 5 sobre funções e arrays. A Tabela 4.6 relaciona todas as missões das quatro primeiras fases, com suas tarefas, objetivos instrucionais e blocos novos introduzidos. As missões onde o identificador está marcado com asterisco indicam que foram reaproveitadas da versão piloto. Existem dois tipos de missões: todos os blocos já ficam disponíveis e o aluno precisa organizá-los em uma ordem para alcançar os objetivos, e aquelas em que o aluno constrói a solução. As missões na Tabela 4.6 onde o identificador está marcado com sinal de adição (+) indicam que é o tipo de missão para organizar blocos. As fases e missões são apresentadas sequencialmente: o jogador precisa concluir uma missão por vez, e somente vai para a próxima fase após concluir todas da fase corrente. Na fase 4, de ciclos, praticamente todas as missões tem o limite de quantidade de blocos. Isso serve para garantir que os alunos usem os blocos de repetição. Do contrário, repetiriam o código para atender cada cliente. A Tabela 4.7 lista todos os blocos que foram criados adicionalmente nesta segunda versão do jogo para resolver as missões. Esses blocos dizem respeito a comandos e ações da personagem.

Tabela 4.6 – Missões das quatro primeiras fases no Ciclo 2

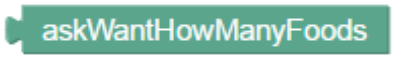
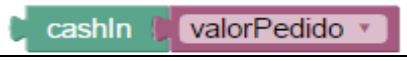
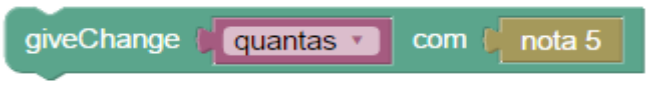
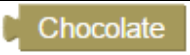
#	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 1 – Aprendiz de atendente: sequenciamento e introdução às variáveis			
As mesmas 10 missões da versão alfa. Consultar a Tabela 4.2.			
Fase 2 – Contador: manipulação de variáveis			
11	Perguntar a três clientes quanta comida desejam. Falar no final a quantidade total.	Usar acumuladores, ou seja, variáveis que vão somando valores no decorrer da execução.	Perguntar quanta comida deseja, falar e somar
12	Perguntar a três clientes quanta comida e quantas bebidas desejam. Falar no final a quantidade de comida e a quantidade de bebidas.	Usar dois acumuladores.	Perguntar quantas bebidas deseja
13	Perguntar a três clientes quanto desejam comer e beber. Cada produto tem seu preço unitário. Falar no final o valor total do pedido.	Usar acumuladores com multiplicação.	Multiplicar.
14	Dois clientes. Perguntar ao cliente quanto deseja comer e beber. Cobrar dele o valor que sempre será pago acima do valor do pedido. O saldo é guardado como gorjeta. Falar no final o total do dinheiro recebido e o total das gorjetas.	Aumento de complexidade usando variáveis e acumuladores.	Subtração e cobrar do cliente
15	Três clientes. Existe um estoque inicial. Perguntar quanto desejam comer. Falar no final quanto sobrou do estoque. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Um exercício mais simples que o anterior. Uso de acumuladores e no final usá-lo com uma subtração.	

16	Dois clientes. Perguntar ao cliente quanto deseja comer. Cobrar dele o valor que sempre será pago acima do valor do pedido. O saldo é a gorjeta. Dessa gorjeta, calcular quantas bebidas poderiam ser entregues ao cliente. Falar a quantidade dessas bebidas. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Usar a divisão de inteiros.	Divisão
17	Perguntar ao cliente quanto deseja comer e beber. Cobrar dele o valor que poderá ser pago acima do valor do pedido. Calcular o troco considerando que somente existem moedas de 2. A sobra fica como gorjeta. Devolver o troco em moedas de 2.	Prática com os quatro operadores matemáticos.	Dar o troco e constante moeda de 2.
18	Perguntar ao cliente quanto deseja comer e beber. Cobrar dele o valor que poderá ser pago acima do valor do pedido. Devolver o troco na íntegra e na menor quantidade possível.	Prática com os quatro operadores matemáticos.	Constantes moeda de 1 e nota de 5.
Fase 3 – Atendente: condicionais			
19*	Três clientes e alguns com sede. Entregar a bebida para quem tem sede.	Usar condicionais.	Se-então e perguntar se está com sede.
20*	Dois clientes que podem ter fome e/ou sede. Entregar o pedido de ambos.	Usar condicionais.	Perguntar se está com fome.
21	Dois clientes que podem ter fome ou sede. Entregar o pedido de ambos. Falar a quantidade de comida e bebida entregue.	Usar condicionais e acumuladores.	
22*	Dois clientes que podem ter fome ou sede. Entregar o pedido de ambos. Limite de quantidade de blocos.	Usar senão nos condicionais. Entender as redundâncias no código e como otimizar.	Se-então-senão
23*	Dois clientes com sede de refrigerante ou sumo de laranja. Entregar o pedido de ambos. Ganha bónus em dinheiro se criar uma solução até um limite de tempo.	Reforçar a aprendizagem de condicionais.	Operador igual, constante refrigerante, movimentar até à caixa de laranjas, pegar uma laranja, movimentar até à máquina de sumos, preparar um sumo.
24*	Dois clientes com fome ou sede de refrigerante ou sumo de laranja. Entregar o pedido de ambos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Condicionais aninhados.	
25	Dois clientes com fome ou sede de refrigerante ou sumo de laranja. Entregar o pedido de ambos. Falar no final o total de vendas. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Condicionais aninhados com acumuladores.	
26*	Dois clientes com fome e/ou sede de refrigerante ou sumo de laranja. Entregar o pedido de ambos. Ganha bónus em pontos se criar uma solução até um limite de tempo. Limite de quantidade de blocos.	Reforçar condicionais aninhados.	

27	Dois clientes com fome e/ou sede. Calcular o preço do pedido, com desconto se ele tiver fome e sede. Falar no final o total do pedido.	Usar operadores lógicos nos condicionais. Usar acumuladores.	Operador lógico E
28	Dois clientes com fome e/ou sede. Se o cliente deseja dois cachorros e uma bebida ele ganha um gelado. Preparar e entregar somente o gelado. Falar no final a quantidade de gelados entregues.	Reforçar o uso do operador lógico E nos condicionais.	Movimentar até à máquina de gelados, preparar e pegar um gelado, Constante sabor chocolate
29	Dois clientes com fome e/ou sede. De acordo com a quantidade de cachorros e bebidas é entregue um dos três sabores de gelado. Falar no final a quantidade de cada um dos sabores entregues.	Usar operador lógico OU nos condicionais.	Operador Lógico OU, Constantes sabor morango e baunilha
30	Perguntar ao cliente quanta comida, bebidas e gelados deseja. O preço do pedido é diferenciado de acordo com essas quantidades. Falar no final o valor do pedido.	Operadores lógicos OU e E concatenados no mesmo condicional.	Perguntar quantos gelados deseja
31	Perguntar ao cliente quanta comida e gelados deseja. Entregar somente os gelados. Entrega um gelado extra se pedir uma quantidade de cachorros.	Comparações com variáveis do tipo booleano	Perguntar o sabor do gelado que deseja, constante verdadeiro ou falso
Fase 4 – Aprendiz de Chef: ciclos			
32+	Três clientes com ou sem fome. Entregar os pedidos.	Uso do ciclo iterativo (for) aninhado com um condicional.	Para
33	Um cliente só com fome, outro só com sede, e o terceiro com ambos. Entregar os pedidos.	Uso de ciclo iterativo com dois condicionais.	
34	Três clientes com fome de 0 a 3 cachorros. Entregar os pedidos. Limite de quantidade de blocos.	Uso de ciclos iterativos aninhados.	
35	Três clientes com fome (sorvete ou cachorros). Entregar os pedidos. Limite de quantidade de blocos.	Uso de ciclos iterativos aninhados com um condicional.	Constante cachorro quente
36	Três clientes. Falar a posição do cliente que mais pediu alimentos e sua quantidade. Limite de quantidade de blocos.	Uso de ciclos para encontrar o maior valor. Imita os exercícios que pedem um número determinado de vezes para o usuário digitar um número (peso, nota do aluno, quantidade, etc). Enquanto digita vai comparando se é o maior da série já digitada.	

37	Três clientes. Falar qual foi a maior quantidade de comida pedida, e quantos clientes que pediram essa quantidade.	Aperfeiçoamento da missão anterior.	
38+	Três clientes com fome ou não. Entregar os pedidos.	Uso dos ciclos condicionais.	Repita enquanto
39	Três clientes com fome, sede, ambos, ou nada. Calcular o preço total de vendas na esplanada. Falar esse preço no final. Limite de quantidade de blocos.	Uso dos ciclos condicionais.	
40	Três clientes com fome, sede, ambos, ou nada. Calcular o preço total de vendas na esplanada. Falar esse preço no final. Para de perguntar aos clientes quanto o total das vendas atingir $7.	Usar dados do ciclo para comparar na condição de fim de ciclo.	
41	Três clientes com fome, sede, ambos, ou nada. Entregar somente cachorros. Parar de atender quando tiver entregado cinco cachorros.	Uso dos ciclos condicionais aninhados.	
42	Os clientes entram e saem da esplanada. Entregar pelo menos 8 cachorros no total. O cliente é atendido por completo. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Uso dos ciclos condicionais aninhados.	Existe um cliente nessa posição?
43	Mesmo que a missão anterior, porém agora o cliente pode ser atendido parcialmente, ou seja, interrompe-se o atendimento quando atingir o limite de cachorros.	Uso dos ciclos condicionais aninhados.	
44	Mesmo que a missão anterior, mas agora considera-se entrega limite de cachorros e bebidas.	Uso dos ciclos condicionais aninhados.	
45	O mesmo que os anteriores. Agora o limite é atender exatamente um valor total de vendas. Por isso, precisa-se tomar cuidado para não vender um produto que ultrapasse esse limite.	Uso dos ciclos condicionais aninhados.	

Tabela 4.7 – Lista de blocos de ação da personagem no Ciclo 2

Bloco	Significado
	Perguntar quanta comida deseja
	Perguntar quantas bebidas deseja
	Falar
	Cobrar do cliente
	Dar o troco
	Constante moeda de 2
	Constante moeda de 1.
	Constante nota de 5
	Movimentar até à máquina de gelados
	Preparar e pegar um gelado
	Perguntar quantos gelados deseja
	Perguntar o sabor do gelado que deseja
	Constante sabor chocolate
	Constante sabor morango
	Constante sabor baunilha
	Constante cachorro quente
	Existe um cliente nessa posição?
	Movimentar até ao cliente na mesa.

A Tabela 4.8 apresenta as missões da última fase. Aqui seguimos uma abordagem diferente. As duas primeiras missões tinham disponíveis todos os blocos e estruturas do jogo. Essas duas missões podiam ser jogadas e repetidas. Não acumulavam pontos, mas serviam como missões para teste e experimentação. As demais 10 missões foram classificadas conforme o nível de dificuldade (F-fácil, M-Médio, D-Difícil), e podiam ser jogadas em qualquer sequência. Resolvendo as 10 missões na sequência, o jogo ensinava os assuntos de arrays e funções.

Tabela 4.8 – Missões da quinta fase no Ciclo 2

#	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 5 – Chef: funções e arrays			
46	Clientes entram e saem da esplanada. Cobrar o valor do pedido, devolver o troco, entrega-lo. Parar quando tiver recebido $50	Uso de procedimentos e funções	Função e Procedimento
47	Cinco clientes: três no balcão e dois na mesa. Perguntar a cada cliente quantos cachorros deseja e guardar em um array. Iterando pelo array, falar a posição que mais pedi cachorros.	Uso de arrays.	Movimentar até ao cliente na mesa, criar array, obter o tamanho do array, aceder um elemento do array e alterar um valor no array.
48	(F) Três clientes. Alguns clientes têm sede. Guardar as quantidades de bebidas que cada cliente deseja, desde que tenha sede. No final falar o array. Limite de quantidade de blocos.	Criar array e guardar no array.	
49	(F) Três clientes. Alguns clientes têm sede. Guardar os pedidos no array. Depois iterar pelo array para preparar e entregar os pedidos. No final falar o array.	Obter elemento do array.	
50	(F) Três clientes com ou sem sede. Calcular o valor do pedido de cada um e guardar no array. No final falar o array. Limite de quantidade de blocos no programa principal.	Criar funções, invocar funções e passagem de parâmetros.	
51	(F) Cinco clientes, com fome e/ou sede. Calcular o valor do pedido de cada um e guardar no array. No final falar o array. Limite de quantidade de blocos no programa principal.	Reforços dos tópicos anteriores.	
52	(M) Cinco clientes, com ou sem sede. Guardar em um array os pedidos (refrigerantes ou sumos) e no outro array na posição ZERO a quantidade de refrigerantes na UM a quantidade de sumos. No final falar os arrays.	Manipulação de dois arrays.	
53	(M) Três clientes com sede. Guardar a quantidade em um array e ao mesmo tempo comparar para saber a maior quantidade. Uma função vai retornar o valor do pedido conforme a quantidade: se casa com a maior quantidade tem desconto. Um segundo array vai guardar os valores dos pedidos. No final falar os arrays.	Reforços dos tópicos anteriores.	
54	(M) Cinco clientes com ou sem fome. Somente três clientes podem ser atendidos. Guardar em um array o pedido e no outro a posição do cliente que pediu. Depois iterar sobre esse array e atender os pedidos. Limite de quantidade de blocos no programa principal.	Reforços dos tópicos anteriores.	
55	(M) Três clientes com ou sem fome. Guardar em um array o valor do pedido. Entregar o pedido ao cliente. Limite de quantidade de blocos no programa principal.	O limite de blocos no programa principal é bem reduzido, sugerindo uma boa modularização.	
56	(D) Três clientes com fome e/ou sede. Cobrar o valor do pedido e devolver o troco com moedas de 2 e 1. Guardar em um array quantas moedas de 2 foram entregues para cada cliente. Guardar em outro array	Reforços dos tópicos anteriores.	

	quantas moedas de 1. No final falar ambos os arrays. Limite de quantidade de blocos no programa principal.		
57	(D) Cinco clientes, com fome e/ou sem sede. Calcular o valor do pedido e guardar em um array. Depois ordenar esse array de forma crescente. No final falar o array.	Método de ordenação bolha	

A estratégia definida para o aluno ganhar pontos era baseada no tempo em que ele gastava para finalizar uma missão (Figura 4.24). Quanto mais rápido o aluno completava a missão, mais pontos ele ganhava. No exemplo, cada estrela vale 30 pontos, e uma delas já foi perdida. O cronômetro ao lado representa o tempo consumido para a estrela corrente.

Figura 4.24 – Forma de ganhar pontos pela pressão do tempo

Foi adicionada ao jogo uma trilha sonora que varia conforme a estrela já consumida no jogo. Cada estrela está associada a uma música que toca enquanto o aluno estiver jogando naquela estrela. Isso ajuda a mantê-lo informado sem precisar visualizar constantemente a caixa de estrelas. Utilizamos seis arquivos MP3 de licença Creative Commons e Royalty Free. Após perdidas as três estrelas, o jogo toca aleatoriamente outro conjunto de três músicas. As músicas são as seguintes, de acordo com a quantidade de estrelas disponíveis:

- Três estrelas: Buddy de Benjamin Tissot;
- Duas estrelas: Energy de Benjamin Tissot;
- Uma estrela: Savage Law de Di Evantile;
- Nenhuma estrela: Carefree e Wallpaper de Kevin MacLeod, e Cute de Benjamin Tissot.

O novo ambiente do jogo está ilustrado na Figura 4.25. Foi mantida a posição da área de edição da solução (no centro), a área de animação e os botões de controlo (à esquerda) e a lista de variáveis quando o aluno depura (à direita). A personagem que é mostrada na área de animação segue aquilo que o aluno configurou no seu avatar. Os botões para controlar a animação são (1) depuração, (2) execução e (3) interromper a execução/depuração. Abaixo dos botões está o controlo de velocidade da execução da animação (a tartaruga para o mais lento até a lebre para o mais veloz). O botão (7) permite ligar e desligar a música de fundo e o botão (8) fecha essa página e volta ao Painel de Controlo. Na área (9) é apresentado o controlo para receber

pontos (no exemplo cada estrela vale 40 pontos) e na área (10) estão a quantidade de pontos e dinheiro já ganho pelo aluno.

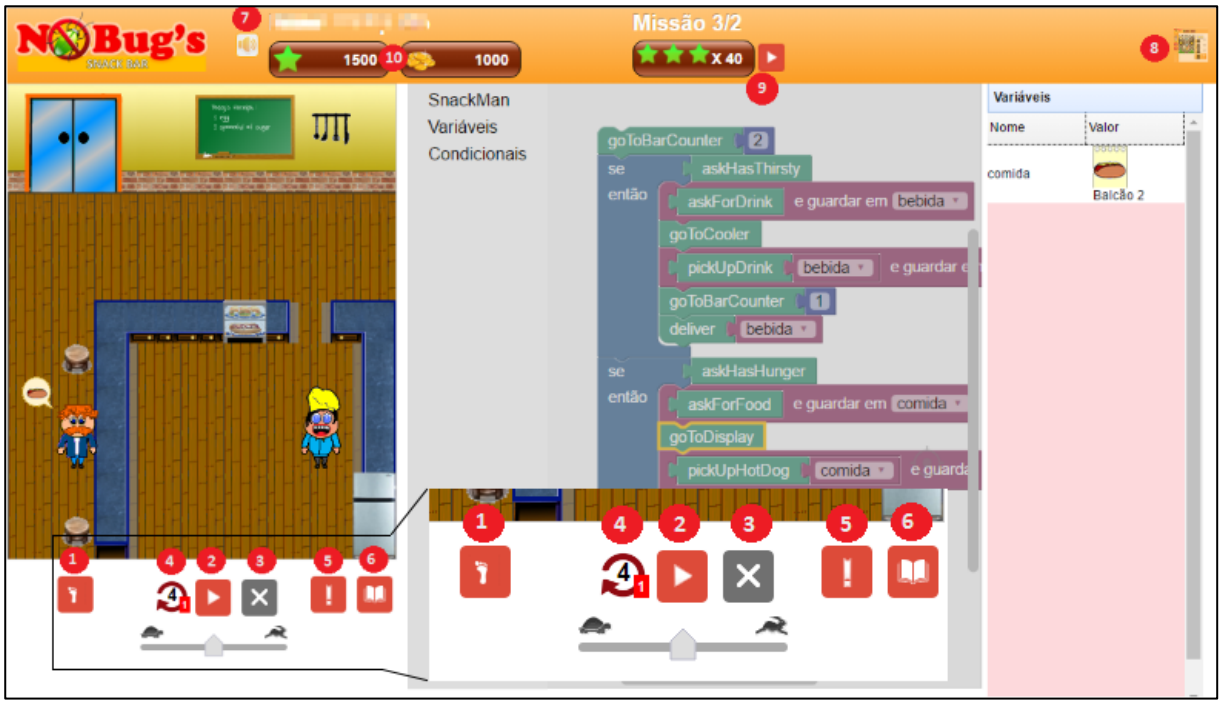


Figura 4.25 – Ambiente do Jogo NoBug's SnackBar no Ciclo 2

No ciclo de desenvolvimento anterior notamos que os alunos desenvolviam a solução para resolver os pedidos conforme desenhados na área de animação, não procurando soluções gerais. Por exemplo, se deviam fazer algo quando um dos três clientes tinha fome e na animação era o cliente na posição 2 que tinha fome, então muitos alunos faziam a solução para atender somente esse cliente e não para qualquer um que pudesse ter fome. Para resolver essa situação, seguimos os princípios dos testes automatizados. A mesma solução passou a poder ser submetida mais de uma vez com diferentes configurações de entrada. O ciclo (4) com um número dentro representa quantas vezes essa solução será submetida, e o número no retângulo vermelho indica quantas vezes a solução já foi executada com sucesso. No exemplo, a mesma solução será executada 4 vezes, com configurações diferentes de cliente, e já foi executada 1 vez com sucesso. Se ocorrer um erro, ou não atingir os objetivos, as execuções são interrompidas.

O aluno clica no botão (5) para visualizar uma nova janela com a descrição e os objetivos da missão (Figura 4.26). Nessa janela está a lista de objetivos, contendo os que foram cumpridos e os não cumpridos. A caixa amarela fornece a quantidade

de pontos que o aluno recebe para cada estrela, e também a quantidade em dinheiro de bônus para, por exemplo, resolver a missão dentro de uma quantidade de blocos.

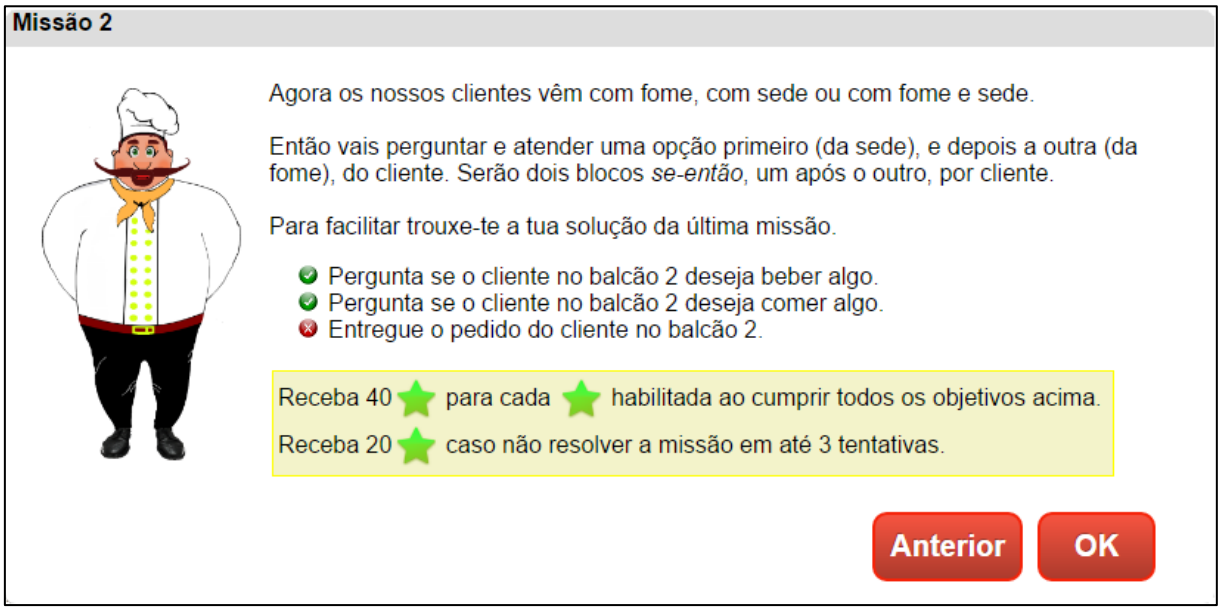


Figura 4.26 – Janela da Descrição da Missão

No ciclo de desenvolvimento anterior foi detetada a necessidade de o jogo fornecer material instrucional para que o aluno possa complementar e se instruir referente aos conceitos que estão sendo praticados na missão. Para atender essa observação, o aluno clica no botão (6) e o jogo apresenta uma janela (Figura 4.27) contendo a explicação do conteúdo relacionado à missão, com ilustrações, e até o relacionamento entre os blocos e as linhas de código em Java. A decisão por essa linguagem foi baseada em seguir a mesma usada nas aulas das turmas em que o jogo será experimentado.

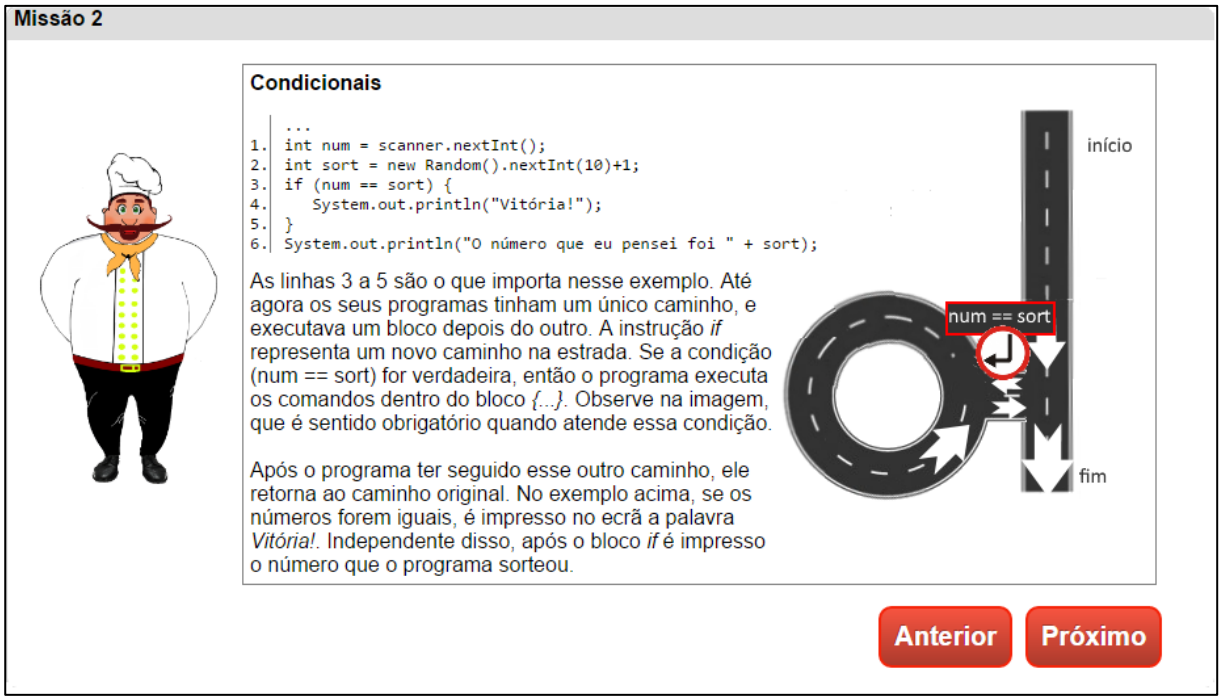


Figura 4.27 – Janela com Conteúdo de Aprendizagem

Foi criada uma área administrativa para que possamos acompanhar o progresso dos alunos e intervir quando eles precisassem de algum auxílio. A Figura 4.28 mostra a página que permite ao professor acompanhar o progresso. Cada caixa azul representa uma missão. Na caixa estão inscritos o número da missão, seu nome, e quantos alunos estão na missão sem concluí-la. Por exemplo, a primeira caixa apresenta quantos alunos nunca entraram no jogo, e a segunda tem um aluno que entrou na missão e não a venceu ainda. Entre a terceira e quarta caixa estão desenhadas as reticências, representando que não existem alunos parados entre essas missões. Ao clicar na parte inferior de uma caixa são listados os alunos que estão na missão. No exemplo, existem dois alunos: o primeiro com 9 tentativas e quase 8 minutos de jogo, e o segundo com 23 tentativas e 39 minutos de jogo. Ao clicar no botão com a lupa de uma das linhas dos alunos, o professor pode ver em detalhe as tentativas do aluno (Figura 4.29).

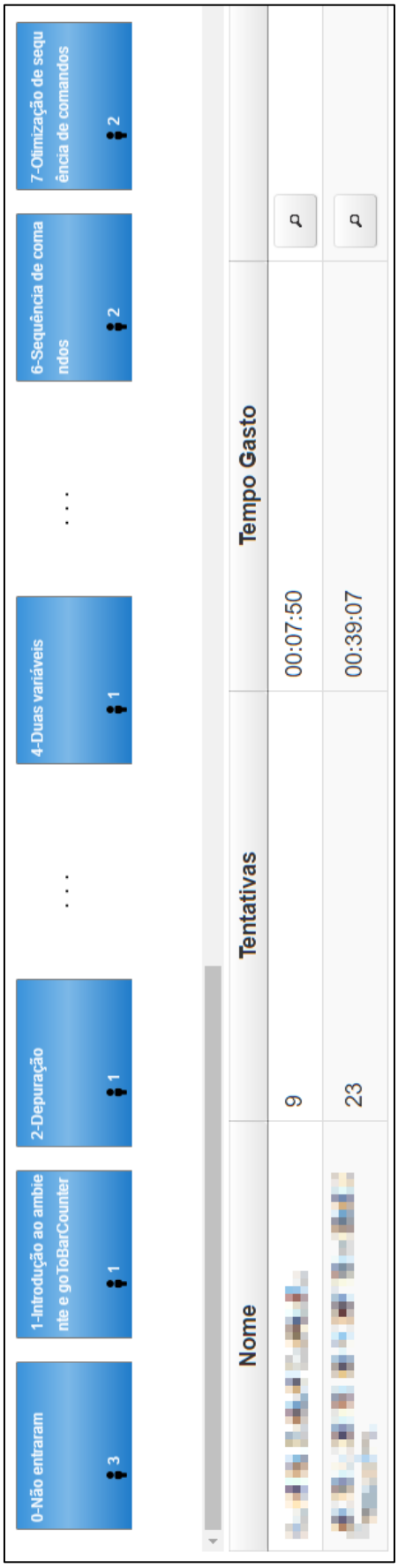
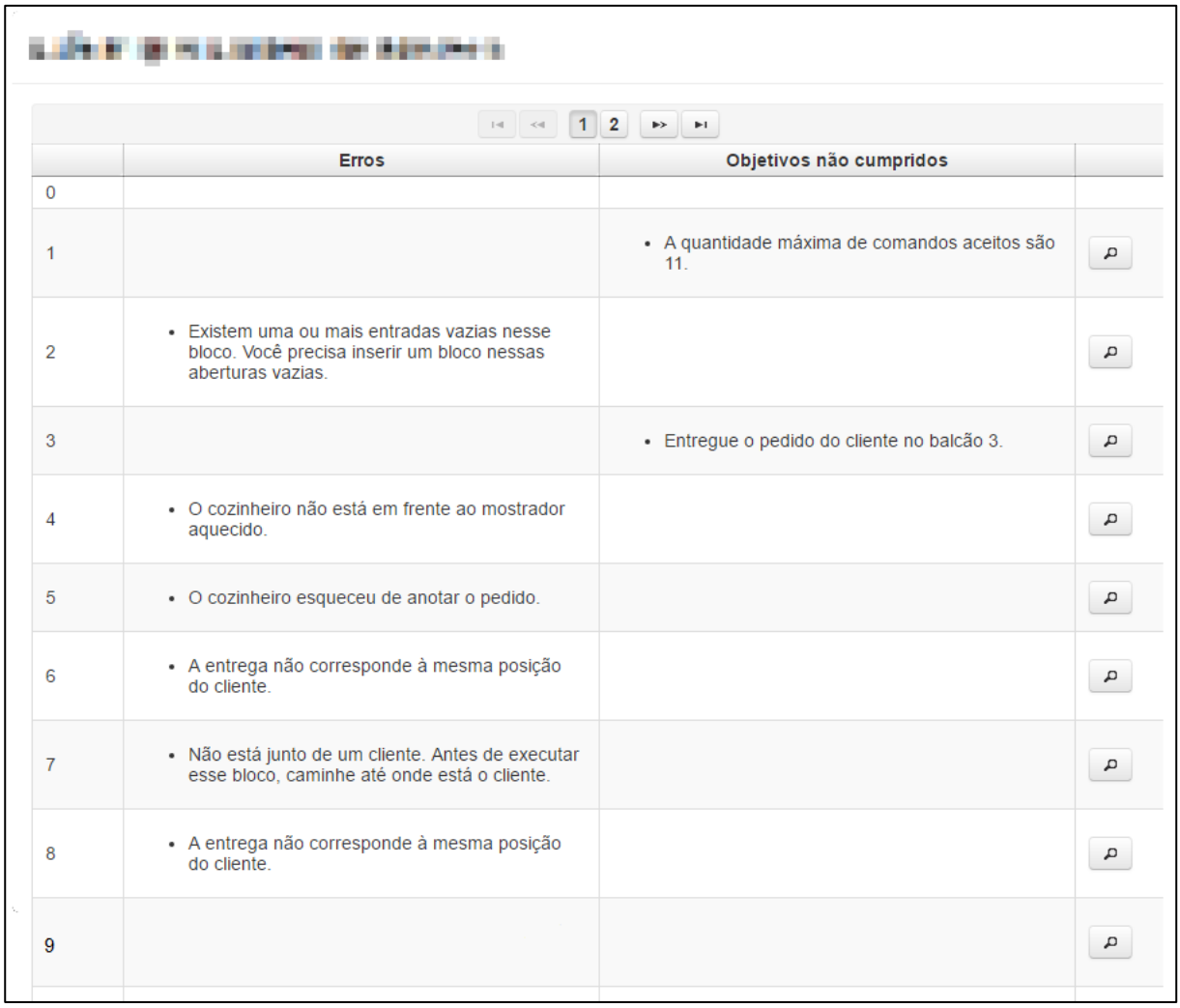


Figura 4.28 – Área Administrativa



1 2

	Erros	Objetivos não cumpridos	
0			
1		• A quantidade máxima de comandos aceitos são 11.	
2	• Existem uma ou mais entradas vazias nesse bloco. Você precisa inserir um bloco nessas aberturas vazias.		
3		• Entregue o pedido do cliente no balcão 3.	
4	• O cozinheiro não está em frente ao mostrador aquecido.		
5	• O cozinheiro esqueceu de anotar o pedido.		
6	• A entrega não corresponde à mesma posição do cliente.		
7	• Não está junto de um cliente. Antes de executar esse bloco, caminhe até onde está o cliente.		
8	• A entrega não corresponde à mesma posição do cliente.		
9			

Figura 4.29 – Detalhes das tentativas do aluno

Na lista de tentativas pode-se examinar o progresso do aluno, consultando os erros que ele cometeu, os objetivos não cumpridos e ao clicar no botão com a lupa pode visualizar a solução tentada. Os erros correspondem àqueles que interromperam a execução da solução. O texto que aparece na coluna é o mesmo que o aluno visualizou. Os objetivos não cumpridos são aqueles não atendidos quando a solução terminou sua execução. Por essa razão, cada tentativa contém um erro ou objetivos não cumpridos. Caso o aluno tenha interrompido a execução por sua decisão, aparece uma linha em branco como por exemplo na linha 9.

4.6.2 Avaliação do Segundo Protótipo

No ciclo anterior os alunos usaram o jogo por uma hora para avaliarmos a diversão e o interesse em jogar. Neste ciclo, os alunos utilizaram o jogo como uma ferramenta auxiliar para a sua aprendizagem nos meses iniciais da cadeira de

introdução à programação. Logo, eles estavam livres para usar o jogo quando e quanto quisessem. Na situação de os alunos usarem o jogo como uma ferramenta de seu quotidiano, dentro do tempo expandido de prática, o objetivo dessa experiência foi novamente **(1) mensurar o prazer dos alunos com o jogo**, **(2) identificar lacunas para melhorar a sua diversão, (3) verificar se os desafios na sequência das missões oferecem o sentimento de fluxo**, e como novo objetivo, **(4) identificar se o jogo contribuiu para a aprendizagem do aluno.** Apesar de três objetivos se repetirem, esse ciclo difere do anterior na forma de sua execução: enquanto o anterior foi num tempo limitado, sendo observados dentro do laboratório, esse ciclo avaliou a utilidade e o interesse do jogo quando combinado junto de outras tarefas do quotidiano dos alunos.

Essa secção está relacionada ao seguinte capítulo de livro já publicado:

- Vahldick, A., Marcelino, M. J., & Mendes, A. J. (2017). Principles of a Casual Serious Game to Support Introductory Programming Learning in Higher Education. In R. A. P. Queirós & M. T. Pinto (Eds.), *Gamification-based e-learning Platform for Computer Programming Education*. IGI Global.

4.6.2.1 População

O ciclo aconteceu entre Setembro e Novembro de 2015 envolvendo 3 turmas no total 60 alunos (36,7% homens e 63,3% mulheres) da Licenciatura em Design e Multimédia da Universidade de Coimbra. As turmas tinham mais alunos, porém nem todos responderam ao último inquérito aplicado, pelo que nos limitámos às conclusões de 60 alunos. A idade média deles era 19,3 anos (± 2,24). Em relação aos hábitos de jogar, 31,7% declarou jogar diariamente. No outro extremo, 37,7% declarou jogar raramente. Quanto ao conhecimento prévio em programação, 60% deles declarou não ter qualquer experiência ou contato prévio.

4.6.2.2 Metodologia e Instrumentos

Todos os alunos foram registados no jogo. Em cada turma, fizemos uma apresentação inicial sobre os objetivos do jogo e da investigação. No primeiro acesso de cada aluno, o jogo apresentava um inquérito (Apêndice B) para identificarmos alguns dados demográficos e hábitos de jogo. Os professores aconselharam os alunos a jogar para praticar e melhorar a compreensão dos conceitos que eram

apresentados em sala. Jogar foi opcional e não havia qualquer relação direta entre o jogo e as aulas, ou seja, os professores não lecionavam baseando-se no jogo, e o jogo também não tinha relação com os exercícios fornecidos pelos professores. A intenção foi auxiliar os alunos a melhorar a compreensão e a aplicabilidade das estruturas de programação. Era esperado que os alunos que tivessem maior dificuldade no entendimento do assunto em sala fossem os mais motivados em jogar.

Após as duas semanas de experiência, com base na quantidade de missões cumpridas, alguns alunos foram convidados para uma entrevista com o objetivo de identificar a necessidade de melhorias imediatas. A cada duas semanas os estudantes tinham que realizar sua autoavaliação sobre a experiência com o jogo. Além disso, dentro do jogo, os alunos eram inquiridos após a 12ª, 18ª, 28ª, 38ª e 44ª missão com a mesma adaptação do EGameFlow aplicado no ciclo de desenvolvimento anterior (Apêndice D). Durante todo o ciclo, a evolução dos alunos foi monitorada e a eles foram enviadas mensagens pelo correio eletrónico com sugestões e conselhos para superar as suas dificuldades. Esses auxílios se transformaram em dicas dentro do jogo para ajudar aqueles alunos que ainda não haviam jogado as respetivas missões. O ciclo foi finalizado após dois meses, com a aplicação de questionário final de avaliação (Apêndice E). Os alunos foram agrupados em quatro grupos de acordo com a quantidade de tempo despendido no jogo. O questionário foi diferente para cada grupo, com questões direcionadas conforme a experiência de jogo, para identificar características que lhes agradaram e aquelas que precisam ser melhoradas. Foram analisados o *log* de interações para identificar a frequência de entrada e acompanhar a evolução dos alunos no jogo.

4.6.2.3 Análise de Resultados

4.6.2.3.1 Primeira Entrevista

Após duas semanas do início da experiência, seis alunos foram convidados para uma entrevista semiestruturada (Apêndice F) com o objetivo de verificar qual o sentimento deles sobre o jogo. Nós esperávamos identificar fraquezas que pudessem ser corrigidas imediatamente e verificar a utilidade com o jogo integrado à disciplina. Três alunos vieram à entrevista: dois que jogaram bastante e outro que não jogou tanto. Consideramos os seguintes pontos que foram referenciados por pelo menos dois deles como os mais importantes para a nossa reflexão:

1. **Pressão do Tempo:** até aquela altura a forma de conseguir pontos era pela pressão do tempo, ou seja, o tempo corria enquanto os alunos resolviam a missão, alternando a música de fundo a cada estrela perdida. Com isso, eles não tinham tempo para prestar atenção em alguns recursos no ambiente do jogo, como a depuração ou aceder ao material instrucional, perturbando o desenvolvimento do seu raciocínio lógico. O sentimento que os alunos exprimiram era que estavam lutando contra o tempo no jogo;

2. **Conquistar Pontos:** como os alunos estavam a jogar, naturalmente eles gostariam de ganhar pontos. Com o desejo de ganhar mais pontos, dois estudantes procuraram encontrar uma estratégia para fugir à pressão do tempo: dedicar mais atenção à leitura da descrição da missão e rascunhar a solução com caneta-e-papel antes de entrar no jogo. Pudemos observar que mesmo que o jogo tenha algumas fraquezas no seu projeto, conquistar pontos faz os estudantes procurar estratégias para obter um melhor desempenho no jogo;

3. **Personalização do Avatar:** os alunos solicitaram mais opções para configurar seu avatar. O avatar era mais importante para identificá-los entre outros alunos na lista de classificação de jogadores do que usá-lo na própria animação, e isso foi um componente motivador para conquistar pontos;

4. **Estética:** os gráficos e a interface em geral não foram considerados muito agradáveis. Entretanto, todos os estudantes declararam não se importarem muito com a qualidade gráfica porque eles estavam jogando para aprender;

5. **Trilha Sonora:** à medida que os estudantes perdiam estrelas o ritmo da música ficava mais acelerado aumentando a pressão. Os alunos declararam que, apesar de a música os ajudar a controlar o tempo sem precisarem olhar para o cronômetro, eles acabavam por a desligar para não terem mais um elemento perturbador na concentração;

6. **Suporte do jogo:** todos expuseram que as dicas e sugestões apresentadas pelo jogo foram suficientes, inclusive, que lhes ajudou a mudar a compreensão de algumas coisas nas missões seguintes por causa dessas ajudas;

7. **Momento de jogar e melhorias na aprendizagem:** eles principalmente acediam ao jogo nos momentos ociosos das aulas, não exclusivamente nas aulas de programação. Porém, eles admitiram que não tinham jogado em casa. Os professores convidaram-nos para jogar, mas não faziam qualquer relação do jogo com as aulas. Naquela altura os alunos estavam a estudar o conceito de variável nas aulas. Os

entrevistados confirmaram que as práticas com variáveis ajudaram eles a entender melhor as explicações dos professores e no desempenho com os exercícios.

Essas conclusões forneceram-nos algumas direções para pequenas melhorias. Analisámos no *log* o tempo consumido pelos alunos para concluir as missões, e duplicámos os limites de tempos, diminuído assim a pressão. Também introduzimos conjuntos de teclas para inserir novos blocos, de modo a melhorar a celeridade na montagem da solução. Além disso, foram adicionadas novas configurações para o avatar: novas cores de peles, diferenciação entre os homens e mulheres, adicionadas opções de bigodes e barbas aos homens, e cortes de cabelos às mulheres.

4.6.2.3.2 Autoavaliação da Experiência no Jogo

A autoavaliação a cada duas semanas já era prática nessas disciplinas em semestres anteriores. Os alunos descreviam seu desempenho e as dificuldades sentidas e os professores usavam essa informação para prover suporte individualizado aos alunos. Nós adicionámos mais uma pergunta à autoavaliação para que os alunos comentassem sobre a sua experiência com o jogo.

Ainda que durante o semestre os alunos tivessem submetido cinco autoavaliações, nós usamos somente as primeiras duas devido à aderência dos alunos ao jogo (Figura 4.30). As datas limites para submeter as avaliações estão representadas por linhas vermelhas verticais. Após a segunda linha, os acessos ao jogo decresceram radicalmente. A primeira avaliação foi submetida por 68 alunos e a segunda por 63, sendo que 60 deles submeteram ambas as avaliações.

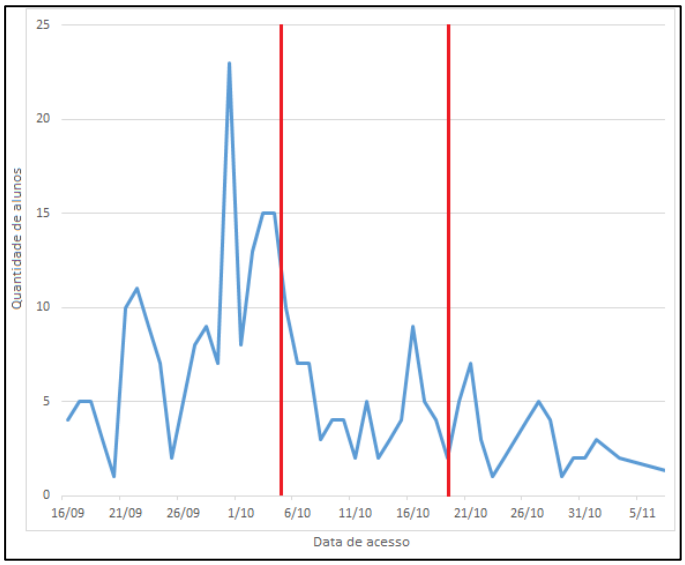


Figura 4.30 – Acessos de diferentes alunos no jogo no Ciclo 2

Nós analisámos o desempenho destes estudantes conforme descreveram na autoavaliação e dividimos em dois grupos: (1) **indo bem**, que foram aqueles que declararam não terem problemas com as aulas e acreditarem que estavam compreendendo o assunto; (2) **com dificuldades**, que foram aqueles que declararam estarem a ter problemas com a disciplina. Em seguida, agrupamos a experiência de jogo também conforme descrita na autoavaliação, em quatro categorias: (1) **avaliação positiva**, que foram aqueles que declararam que o jogo era interessante e poderia ser útil para a sua aprendizagem; (2) **dificuldades no jogo**, que foram aqueles que listaram dificuldades em finalizar as missões; (3) **não jogaram**, os que não preencheram o campo de comentário ou que afirmaram não terem jogado; (4) **avaliação negativa**, que foram aqueles que explanaram não acreditar que o jogo pudesse auxiliá-los na aprendizagem de programação. A Tabela 4.9 apresenta o cruzamento dessas categorias. Apesar das quantidades totais entre as avaliações terem o mesmo valor, acontece que alguns alunos que avaliaram que estavam indo bem na primeira avaliação mudaram de opinião na segunda, e vice-versa. Na segunda avaliação menos alunos tiveram avaliações positivas ou negativas, porém mais alunos não jogaram, o que pode ser constatado na Figura 4.30.

Tabela 4.9 – Autoavaliação

		Avaliação Positiva	Dificuldades no Jogo	Não Jogaram	Avaliação Negativa	Total
Avaliação 1	Indo bem	7 (11,7)	4 (6,7)	10 (16,7)	2 (3,3)	**23 (38,3)**
	Com dificuldades	9 (15,0)	4 (6,7)	20 (33,3)	4 (6,7)	**37 (61,7)**
	Total	**16 (26,7)**	**8 (13,3)**	**30 (50,0)**	**6 (10,0)**	**60 (100,0)**
Avaliação 2	Indo bem	7 (11,7)	2 (3,3)	14 (23,3)	0 (0,0)	**23 (38,3)**
	Com dificuldade	3 (5,0)	6 (10,0)	26 (43,3)	2 (3,3)	**37 (61,7)**
	Total	**10 (16,7)**	**8 (13,3)**	**40 (66,6)**	**2 (3,3)**	**60 (100,0)**

Nos comentários da avaliação positiva, alguns estudantes responderam que o jogo era viciante devido à conquista de pontos e porque podiam comparar o avatar deles com o dos outros nas listas de classificação. Outro estudante comentou "*Agora consigo pensar melhor sobre os exercícios e fazer diagramas no caderno que me ajudam a resolver os problemas mais facilmente*".

Categorizamos em três grupos as respostas de quem teve dificuldades para jogar: (1) **pressão do tempo**, agrupando alguns alunos que afirmaram ter tido dificuldades para se concentrarem ao longo do tempo enquanto raciocinavam para resolver os problemas; (2) **compreensão da descrição das missões**, inclui os comentários sobre a dificuldade em extrair a informação da explicação; (3) **suporte**

insuficiente, inclui situações em que os alunos dizem ter tido situações no jogo em que não receberam ajuda suficiente para vencer as dificuldades.

Os alunos que responderam que não jogaram (ou pararam de jogar) declararam que não perceberam relação entre o jogo e as aulas, gostariam de resolver por código em vez de blocos, a qualidade gráfica precisa melhorar e *"(…) eu achei que era um pouco repetitivo após algumas fases e me senti cansado*". Adicionalmente, os alunos sugeriram que (1) os objetivos das missões fossem sempre visíveis, (2) receber recompensas quando realizasse objetivos extras, (3) mais itens desbloqueáveis e (4) opções de avatar.

4.6.2.3.3 Diversão Percecionada

Nós planeámos para que o jogo inquirisse os alunos quanto à diversão, usando o questionário da escala EGameFlow (Apêndice E), em cinco momentos, após a 12ª, 18ª, 28ª, 38ª e 44ª missão. A Figura 4.31 apresenta a quantidade total de missões finalizadas por cada aluno. Somente cinco estudantes concluíram 18 ou mais missões. Por essa razão, resolvemos considerar avaliar somente o primeiro inquérito (n=22).

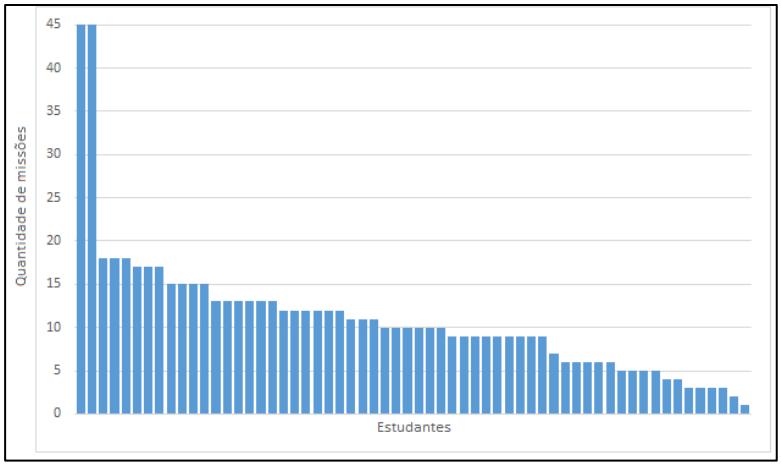


Figura 4.31 – Estudantes e missões cumpridas no Ciclo 2

O questionário contém duas partes: (1) 27 questões com cinco níveis de concordância na escala Likert (1-não concordo totalmente e 5-concordo totalmente) e (2) duas questões abertas permitindo que os estudantes comentassem o que mais e o que menos tinham gostado no jogo. Analisamos e codificamos o conteúdo dessas questões usando a técnica da *Grounded Theory* (Corbin & Strauss, 2015). A Figura 4.32 apresenta o resultado da codificação dessas respostas. Quanto aos aspetos positivos:

- Instrução: 11 alunos gostaram de poder comandar uma personagem através da programação. Eles consideraram que usar um jogo para aprender pode ser uma boa prática para ultrapassar as barreiras iniciais da aprendizagem de programação;
- Desafios: 6 consideraram interessante a sequência das missões, a dificuldade crescente e os problemas que o jogo instigava a serem resolvidos;
- Usabilidade: 4 alunos acharam o jogo simples de ser usado;
- Suporte: um aluno evidenciou o suporte que demos durante o jogo fornecendo dicas e motivando a continuar.

Quanto aos aspetos negativos:

- Pressão do Tempo: 4 alunos queixaram-se sobre o tempo ser muito curto, ou pediram para acrescentar recursos que aumentassem a produtividade na criação da solução;
- Usabilidade: 4 alunos reclamaram da falta de simplicidade de uso no jogo. Na altura não era possível consultar e interagir com missões já concluídas, sugestão feita naquele momento;
- Descrição da Missão: 3 alunos destacaram a necessidade de melhorar a explicação das missões;
- Audiovisual: 3 alunos sugeriram melhorar a interface gráfica e a música;
- Desafios: 2 alunos não viram utilidade em algumas missões e afirmaram que outras eram muito difíceis;
- Codificação: 2 alunos gostariam de ver relação do jogo com codificação, seja simplesmente mostrando a conversão dos blocos em código, ou produzindo com codificação;
- Feedback: 2 alunos sentiram dificuldades e o jogo não conseguiu dar o suporte necessário para que eles vencessem os problemas;
- Nada: 2 alunos não conseguiram apontar pontos fracos do jogo.

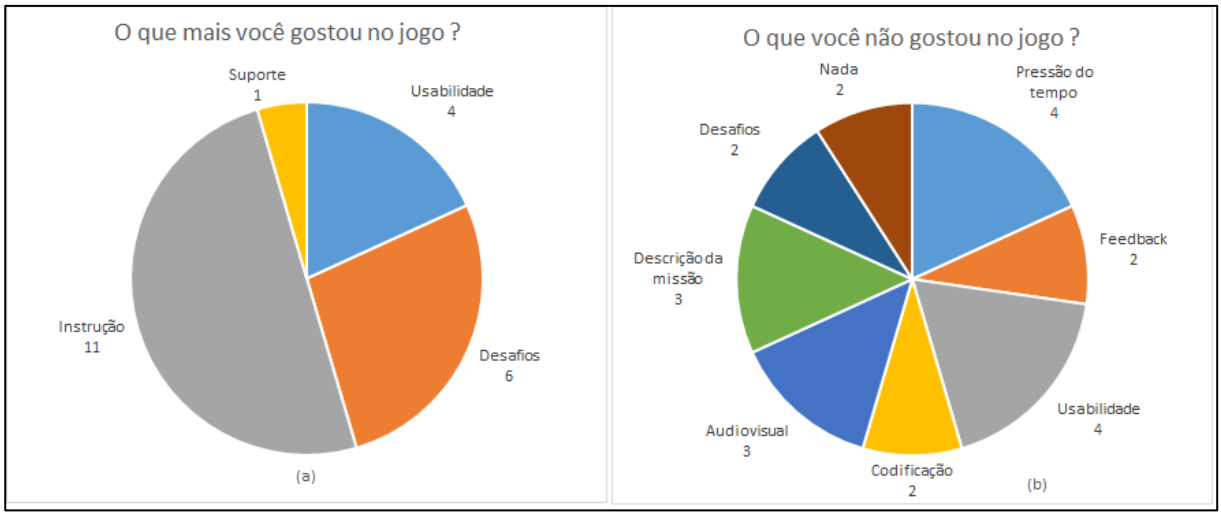


Figura 4.32 – Opiniões sobre o jogo (a) positivas e (b) negativas

A Tabela 4.10 apresenta o resultado da escala EGameFlow, comparando com os resultados do primeiro ciclo. Nesse ciclo, os dois piores itens avaliados foram a imersão e autonomia, inclusive ambos com maior desvio padrão, demonstrando que existiu uma grande discrepância entre a opinião dos alunos. Quanto ao item autonomia, ao analisarmos as autoavaliações e as respostas das questões abertas, foi possível concluir que a pressão do tempo frustrou os alunos ao ponto de sentirem a falta de controlo no jogo. Outros pontos que interferiram nesse fator dizem respeito a possibilidade de codificar em vez de usar blocos, e à usabilidade, especialmente no que respeita à falta de algumas opções como desfazer e refazer as últimas ações. Em relação à imersão, os alunos pediram mais opções para configuração do avatar e melhorias na qualidade gráfica da interface e sons no jogo. Jogos sérios analisados (Fu et al., 2009) também apresentaram o item imersão como aquele com menor valor entre as categorias. Isso pode-nos conduzir a acreditar que quando o aluno não consegue resolver os problemas ou responder a uma questão pode romper com a imersão no jogo. Apesar de que não é possível comparar os resultados dos dois ciclos, pois o período, a disponibilidade e a necessidade de jogar foram distintas, pode-se notar que o valor total é ligeiramente maior que a turma de ME e menor que a turma de ES.

Tabela 4.10 – Primeira medição da escala EGameFlow no Ciclo 2

Ciclo	Concentração	Clareza dos objetivos	Suporte	Desafio	Autonomia	Imersão	Total
2-UC	3.9 (±0.62)	4.1 (±0.62)	4.0 (±0.69)	4.1 (±0.67)	3.7 (±0.82)	3.5 (±0.90)	3.9 (±0.62)
1-ME	3,8 (±0,50)	4,0 (±0,46)	4,1 (±0,54)	3,9 (±0,70)	3,4 (±0,95)	3,8 (±0,77)	3,8 (±0,59)
1-ES	4,1 (±0,46)	4,4 (±0,58)	4,3 (±0,57)	4,3 (±0,61)	4,2 (±0,63)	3,6 (±0,56)	4,1 (±0,48)

4.6.2.3.4 Inquérito Final

Após finalizar o ciclo, foi aplicado um inquérito, para mensurar o sentimento de diversão e aprendizagem (Apêndice E), contendo 26 questões com cinco níveis de concordância na escala Likert (1-não concordo totalmente e 5-concordo totalmente) e 3 questões de resposta aberta. O questionário não foi o mesmo para todos os estudantes, pois eles tiveram quantidades diferente de missões concluídas. Por essa razão precisamos encontrar grupos discretos de alunos, com isso foi aplicada a técnica de Análise Hierárquica de Agrupamentos (Johnson, 1967) com o método de ligação de Ward usando a quantidade de tempo que o aluno jogou. Foi usado o ambiente estatístico R[15] para os testes de clusterização. Essa técnica foi escolhida por ser a mais comumente usada, e se baseia em encontrar pares similares em função de um coeficiente, nesse caso, a quantidade de tempo jogado. Esse coeficiente foi definido em função da persistência em jogar, pois alguns alunos gastaram muito tempo em tentativas para terminarem as missões. O método Ward apresentou-se mais adequado quanto a distribuição dos grupos, resultando em quatro grupos: o primeiro foi aquele que consumiu mais tempo e o último que praticamente não jogou ou jamais entrou no jogo. A Tabela 4.11 apresenta a população de cada grupo, quantos destes responderam o inquérito, o que desejávamos identificar no grupo e a lista de questões. Apresentaremos uma análise dos resultados de cada grupo, e ao final faremos uma comparação para responder as questões de investigação.

Tabela 4.11 – Grupos e questões ao inquérito final no Ciclo 2

Grupo	Amostra	Respondeu	Questão de Investigação	Lista de Questões
1	2	2	Porque continuou a jogar?	Q1-Q18
2	12	10	Quais foram as dificuldades no jogo?	Q1-Q6, Q10-Q24, Q27-Q29
3	36	32	Quais foram as razões para parar de jogar?	Q12-Q25, Q28, Q29
4	23	16	Porque jogou tão pouco ou nunca entrou?	Q24-Q26, Q28, Q29
Totais	**73**	**60**		

Grupo 1: Porque continuou a jogar?

Este grupo continha somente duas estudantes que finalizaram todas as missões. A Figura 4.31 mostra as respostas desse grupo. Uma das estudantes teve dúvida (nível 3 da escala Likert) se aprendeu o processo de depuração (Q6) e se melhorou o seu raciocínio lógico (Q10). Ambas concordam que o jogo ajudou na aprendizagem de condicionais e ciclos (Q7-Q9) e observaram relação com o que aprenderam no jogo e as aulas (Q12-Q13). Entre os elementos de jogo avaliados, em

[15] https://www.r-project.org/

ordem decrescente ficaram classificados: conquistar pontos (Q14) empatado com a classificação com os demais jogadores (Q17), a personalização do avatar (Q16) empatado com abrir seu próprio negócio (Q18) e o pior classificado foi a música ambiente (Q15).

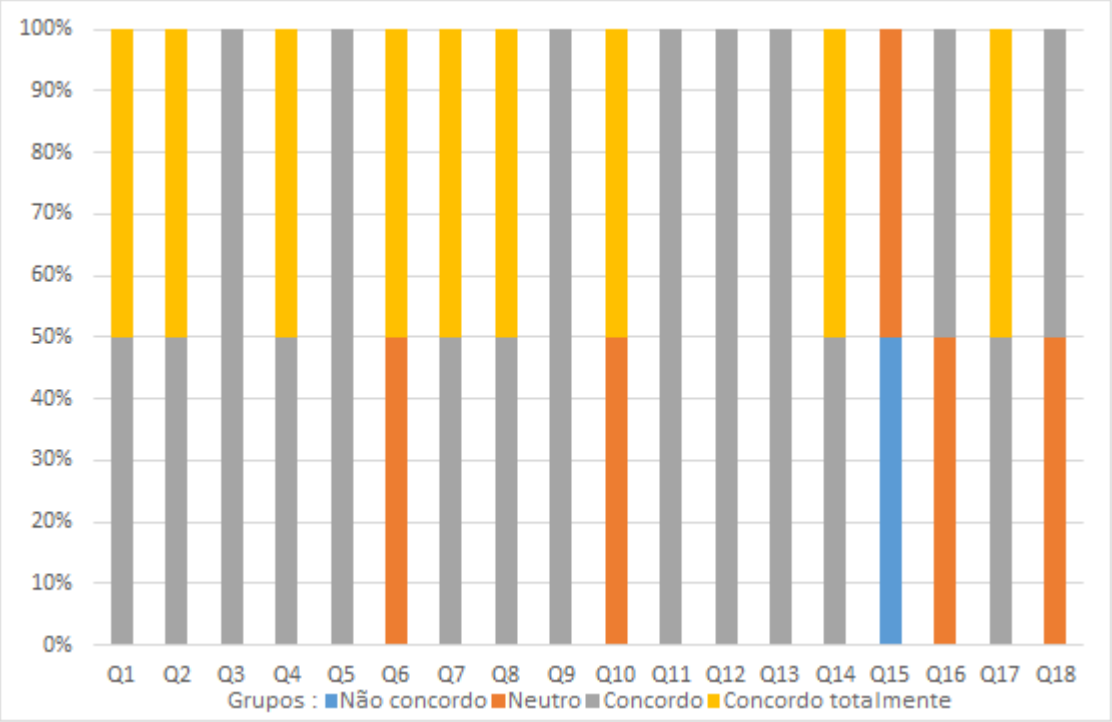


Figura 4.33 – Respostas do Grupo 1

Grupo 2: Quais foram as dificuldades no jogo?

O grupo 2 incluía os estudantes que terminaram várias missões (entre 12 a 18), mas interromperam de jogar, e queríamos tentar identificar os seus motivos. A Figura 4.34 apresenta as respostas desse grupo. Os alunos não concordaram (40%) que aprenderam coisas úteis para as aulas (Q4) e nem para o processo de depuração (Q6), eles admitiram que não tiveram conhecimento suficiente para avançar no jogo (Q11) e não perceberam fortes relações do jogo com as aulas (Q12-Q13). A maioria dos alunos (60%) declarou que o jogo não forneceu suporte suficiente (Q20) e metade deles (50%) não concordou que a dificuldade aumentou apropriadamente entre as missões (Q21). Metade deles concordou totalmente que resolver por codificação pode ser mais divertido do que pelos blocos (Q24). Entre os elementos de jogo avaliados, em ordem decrescente ficaram classificados: conquistar pontos (Q14), em segundo empatados a personalização do avatar (Q16) e abrir seu próprio negócio (Q18), em quarto a classificação com os demais jogadores (Q17), e o pior classificado foi a música ambiente (Q15).

Nas questões abertas sobre a sugestão de recursos em programação (Q27): 30% respondeu escrever código e os restantes não deram sugestões. Sobre as sugestões de recursos no jogo (Q28), 30% novamente respondeu escrever código, 30% não deram sugestões, houve uma sugestão para melhorar a explicação das missões, outra para usar mais o teclado para criar a solução, outra para mudar a aparência do avatar para ter um ar mais profissional, ou sofisticado, conforme se vai progredindo no jogo, e finalmente uma sugestão para jogar no modo multiusuário. A última questão aberta pedia a indicação das razões de terem parado de jogar (Q29): três responderam porque devido aos compromissos com as aulas, dois porque não passaram em uma missão, um não observou benefícios com a aula, outro não gostou do jogo, mais outro porque a dificuldade entre as missões não era apropriada, um preferiu investir o tempo dele praticando através dos exercícios fornecidos pelo professor e outro não respondeu.

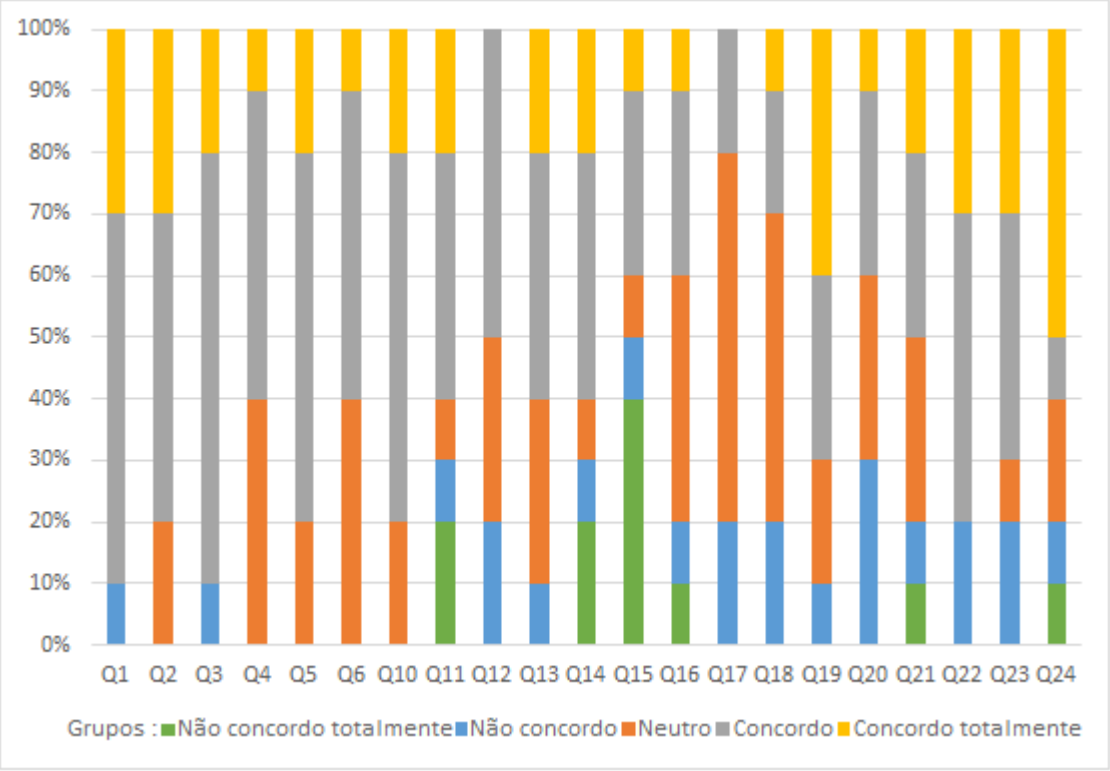


Figura 4.34 – Respostas do Grupo 2

Grupo 3: Quais foram as razões para parar de jogar?

Esse grupo incluía os alunos que jogaram entre 6 e 11 missões. Queríamos identificar o que os impediu de avançar no jogo. A Figura 4.35 mostra essas respostas. Aproximadamente 40% não concordou que exista a relação entre o jogo e as aulas (Q12-Q13). Este grupo teve grandes dificuldades de entender as mensagens de erro

(Q22) e afirmaram que o suporte não foi suficiente (Q20). Metade deles concordaram que resolver por codificação pode ser mais divertido do que por blocos (Q24). Esse grupo não achou tão atraente o contexto da esplanada (Q25) e 60% deles não gostou da qualidade gráfica do jogo (Q26). Entre os elementos de jogo avaliados, em ordem decrescente foram classificados: classificação com os demais jogadores (Q17), conquistar pontos (Q14), personalizar o avatar (Q16), abrir seu próprio negócio (Q18) e música ambiente (Q15). Foi uma classificação interessante: eles não estavam nas primeiras posições, mas responderam que conquistar pontos era o recurso do jogo que mais os motivava.

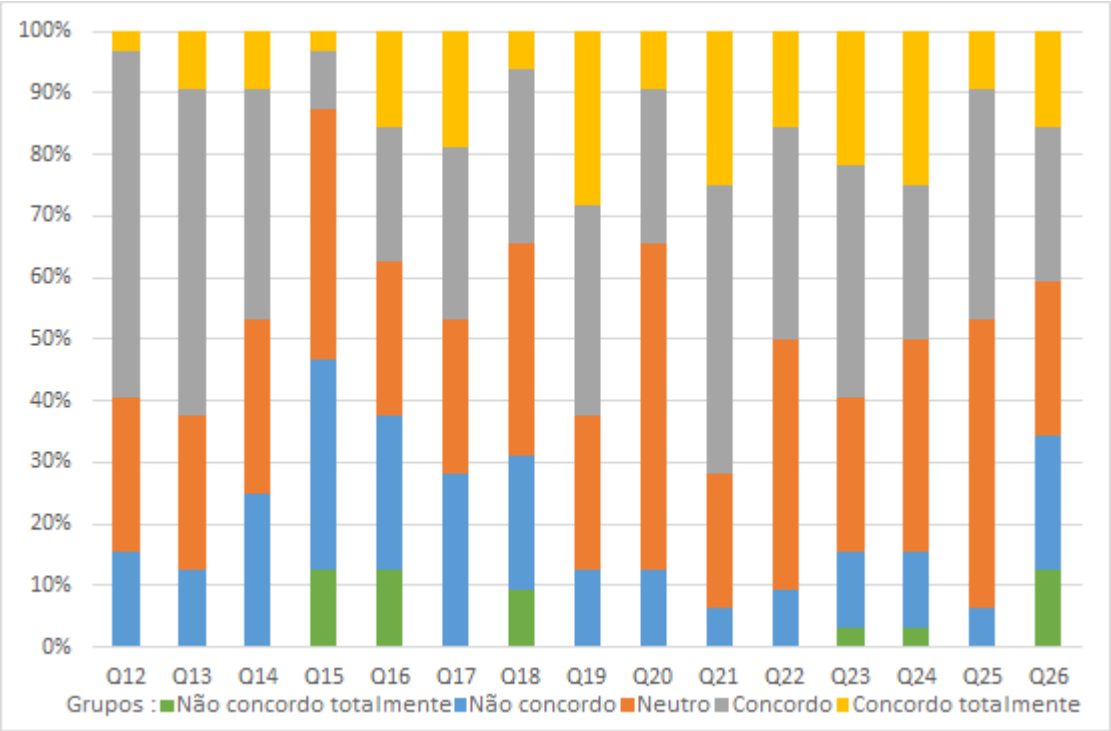


Figura 4.35 – Respostas do Grupo 3

Nas questões abertas sobre a sugestão de recursos no jogo (Q28) (alguns deram mais de uma sugestão): oito para melhorar a qualidade gráfica, cinco sobre escrever código, três para oferecer mais opções de ajuda e suporte, três sobre material instrucional, três para oferecer mais opções de configuração para o avatar, dois para o jogo ser multiusuário, um para usar mais o teclado para criar a solução, outro referiu a música e outro pediu um contexto diferente da esplanada. A última questão aberta pedia para explicarem as razões de terem parado de jogar (Q29): devido aos compromissos com as aulas (37,5%), não passaram em uma missão (28,1%), preferiram investir o seu tempo praticando através dos exercícios fornecidos

pelo professor (15,6%), não gostaram do jogo (9,4%), dificuldade entre as missões não era apropriada (3,1%), e não deram sugestões (6,3%).

Grupo 4: Porque jogou tão pouco ou nunca entrou?

Este grupo incluía os alunos que jogaram até 5 missões. A Figura 4.36 apresenta as respostas desse grupo. Metade deles concordaram totalmente que resolver por código seja mais divertido do que por blocos (Q24). Esses alunos não gostaram (85%) do contexto da esplanada (Q25) e nem (80%) da qualidade gráfica das imagens (Q26).

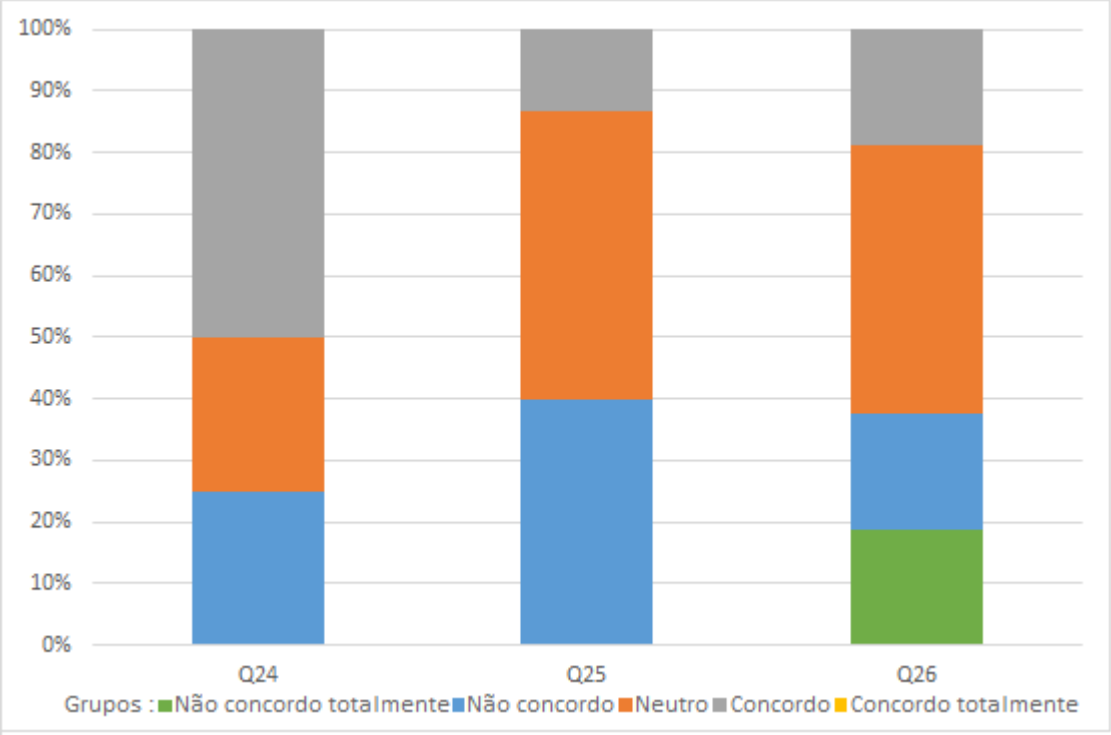


Figura 4.36 – Respostas do Grupo 4

Nas questões abertas sobre a sugestão de recursos no jogo (Q28): melhorar a estética (37,5%), material instrucional (18,75%), escrever código (12,5%), não gostaram do jogo (6,25%) e não deram sugestões (25%). A última questão aberta pedia para explicarem as razões de terem parado de jogar (Q29): não gostam de jogos digitais, do gênero ou da mecânica do jogo (43,75%), devido aos compromissos com as aulas (31,25%), qualidade gráfica (12,5%), tinham domínio sobre a programação e acharam o jogo desnecessário (12,5%).

Comparação entre os grupos:

Os grupos 1 e 2 concordaram que o jogo contribui na evolução do raciocínio lógico (Q1), na aprendizagem de manipulação com variáveis (Q2) e decomposição

(Q3), aprenderam coisas úteis para as aulas (Q4) e reforçaram assuntos que tinham tido em sala (Q5). Entretanto, ambos tiveram dificuldades em reconhecer a aprendizagem de técnicas de depuração (Q6). Como o segundo grupo completou de 12 a 18 missões, e o terceiro de 6 a 11, esses grupos sentiram muito menos o efeito do jogo sobre a aprendizagem (Q10-Q13), do que o grupo 1.

Em relação ao elemento de jogo mais motivador (Q14-Q18), os grupos 1 e 2 selecionaram conquistar pontos. Surpreendentemente, estar entre os melhores na classificação foi o elemento mais motivador do grupo 3. Esse elemento também foi bem votado pelo grupo 1, mas não tão bem pelo grupo 2. Todos os grupos selecionaram a música como o elemento menos motivador. A trilha sonora foi adicionada no jogo, pois a música permite que as pessoas fiquem tocadas, imersas e incorporadas ao jogo (Schell, 2008). De qualquer forma, Linek et al. (2012) apontaram não haver correlação da música ambiente no impacto com desempenho e aprendizagem. Logo, a música foi inclusa para manter a característica de ser um jogo.

As questões Q19 a Q23 estavam relacionadas ao sentimento de controlo do jogador e suporte oferecido ao jogo. A questão Q24 vinha tentar confirmar a preferência dos alunos em usar codificação em vez de blocos. Essas perguntas foram direcionadas aos grupos 2 e 3, que jogaram várias missões, porém pararam, no intuito de percebermos o que resultou essa interrupção. Ambos os grupos tiveram a mesma opinião nessas questões. Eles concordaram que compreenderam o que cada bloco fazia (Q19), a dificuldade crescente entre as missões foi apropriada (Q21), entenderam os objetivos que não conseguiam cumprir (Q23) e que usar codificação pode ser mais interessante que construir com blocos (Q24). Por outro lado, o suporte não foi suficiente para a evolução deles (Q20).

Sobre as sugestões para melhorar o jogo (Q28), as três mais comentadas a melhoria da aparência do jogo, escrever código e material instrucional. As três razões mais citadas para terem parado de jogar (ou nem entrarem no jogo) (Q29) foram os compromissos com as aulas, não conseguirem vencer uma missão, e não gostarem do jogo, do gênero ou da mecânica.

4.6.2.3.5 Resumo dos Resultados

Para esse ciclo havia quatro objetivos:

(1) Mensurar o prazer dos alunos com o jogo

Apenas duas estudantes concluíram as 57 missões. O *log* apresentou uma descida significativa nos acessos ao jogo por parte dos alunos. Para apoiar o acréscimo no desinteresse dos alunos, a autoavaliação quinzenal comprovou a quantidade de alunos que não jogaram. Já na primeira entrevista identificamos a insatisfação dos alunos com a gestão do tempo dentro da missão, o que foi reiterado na autoavaliação. Os alunos se sentiram motivados em ganhar pontos para personalizar seus avatares para serem comparados nas classificações da turma. O teste de EGameFlow foi aplicado com 36,7% dos alunos e ficou muito próximo do ciclo de desenvolvimento anterior, mesmo com os alunos a poderem decidir quando jogavam. A dimensão mais mal avaliada foi a imersão, apontando a necessidade de melhorarmos a integração dos recursos de diversão com as tarefas no jogo. Vários estudantes declararam que poderia ser mais divertido jogar por codificação em vez dos blocos. Mas acreditamos que isso vem da necessidade de ser mais produtivo no momento de fazer a solução, ou seja, se fosse digitado seria mais rápido do que arrastar blocos. Muitos alunos afirmaram que interromperam de jogar devido às obrigações nas cadeiras.

(2) Identificar lacunas para melhorar a sua diversão

O que pudemos observar nessa experiência é que os alunos estavam muito interessados em ganhar pontos no jogo. Enquanto conseguiam ganhar pontos, os alunos mantinham-se jogando. Quando o jogo apresentava uma grande dificuldade para obter pontos, eles paravam de jogar. O tipo de problema que eles precisavam resolver no jogo requeria tempo para raciocinar e pensar. O nosso principal erro nessa versão foi usar a pressão do tempo para medir a forma de ganhar pontos. Eles admitiram que lhes diverte gastar os pontos ganhos para personalizar seu avatar e competir com seus colegas pelas melhores posições na classificação. Precisamos melhorar a qualidade gráfica e a aparência do jogo, que pode contribuir em melhorar a imersão.

(3) Verificar se os desafios na sequência das missões oferecem o sentimento de fluxo

Não foi analisado o nível de dificuldade das missões pois apenas 5 alunos avançaram até a 18ª missão.

(4) Identificar quanto o jogo contribuiu para a aprendizagem do aluno

Na entrevista inicial, os alunos que jogaram já declararam que conseguiam fazer analogias entre o que era praticado no jogo e o que estava sendo dado pelos professores nas aulas. As duas estudantes que finalizaram as 57 missões concordaram que o jogo exigia esforço e disciplina, mas elas se dedicaram a terminar as missões de um assunto antes de ele ser dado em aula, e conseguiam entender mais fácil e rapidamente que seus colegas. Uma delas nunca tinha tido contato prévio com o assunto e outra já tinha tido algum contato com a programação. Ao verificar nos números de autoavaliação, pode-se observar que 61,7% dos alunos afirmaram terem dificuldades na disciplina. Essas autoavaliações são originárias do primeiro mês de aula em que ainda são abordados assuntos essenciais como variáveis e condicionais. Por essa razão, vislumbramos que o jogo possa ser ferramenta útil para ajudá-los a melhor entenderem os assuntos. Uma forma dos alunos valorizarem mais o jogo talvez seja aumentar a relação com as aulas, onde o professor cite o jogo ou demonstre algum conceito usando o jogo.

Devido à baixa adesão dos alunos no jogo, os dois últimos objetivos não foram cobertos por esse ciclo.

4.7 Terceiro Ciclo de Desenvolvimento

4.7.1 Modificações no Jogo

As modificações apresentadas nessa subsecção foram inspiradas nas observações e conclusões do ciclo de desenvolvimento anterior, e representam aquilo que será testado neste ciclo. Diferente do que aconteceu no ciclo anterior, em que houve muitas modificações durante o próprio ciclo, o mesmo não aconteceu neste ciclo: todas as modificações feitas antes do ciclo permaneceram assim até ao seu encerramento. O projeto gráfico do jogo foi todo refeito. A Figura 4.37 – pode ser comparada com a Figura 4.17 – apresenta a janela de abertura e autenticação no jogo, o logotipo refeito, e as baratas ficam movendo-se pela página.

Figura 4.37 – Janela de autenticação a partir do Ciclo 3

A estória do jogo continua a mesma, e é contada na primeira vez que o aluno acede o jogo. As imagens foram refeitas (pode se comparar a Figura 4.18 com a Figura 4.38) e agora é atribuído um rosto inicial do protagonista de acordo com o género (masculino ou feminino) do aluno. O desenho do rosto das demais personagens segue o padrão visual do avatar.

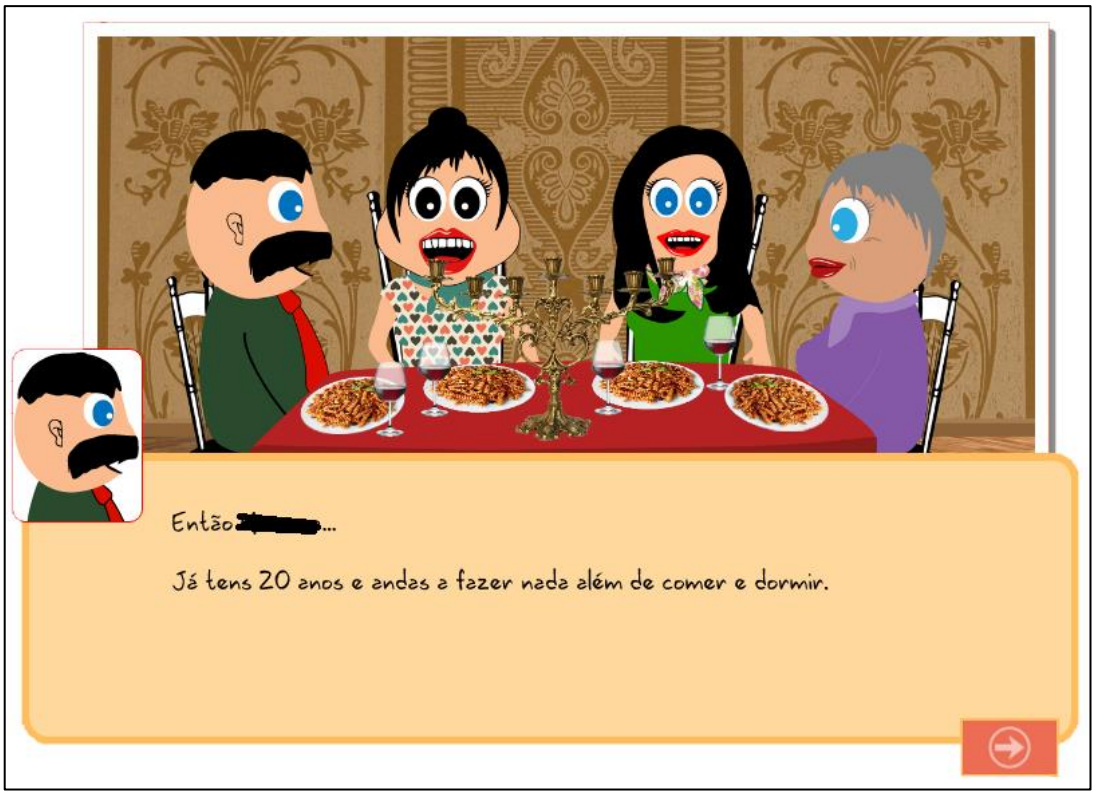


Figura 4.38 – O enredo do jogo a partir do Ciclo 3

Esse padrão também foi aplicado ao Professor Burguer, como podem ser comparadas a Figura 4.19 com Figura 4.39. Esse é o momento em que o Professor introduz o aluno com o ambiente do jogo.

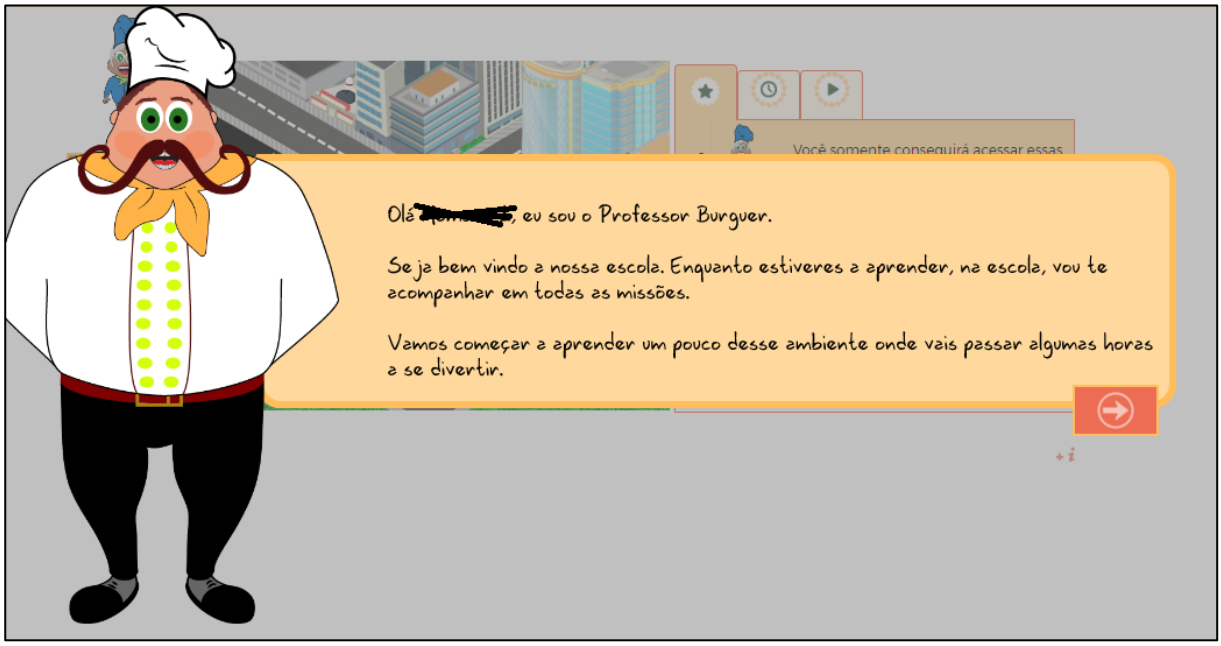


Figura 4.39 – Apresentando o ambiente do jogo a partir do Ciclo 3

O painel de controlo continuou com as mesmas funções (Figura 4.20 com Figura 4.40). As áreas futuras continuam não implementadas, porém verificamos no ciclo anterior que elas constituem como elementos motivadores por estimularem a curiosidade dos alunos.

Figura 4.40 – Painel de controlo no Ciclo 3

O botão [Conquistas] abre uma janela com as recompensas adicionais. Essas recompensas são uma nova forma dos jogadores receberem pontos ao cumprirem

objetivos marginais e não só por terminarem as missões. Os alunos podem ganhar pontos, moedas, ou qualquer outra bonificação que o professor desejar estipular, quando terminam todas as missões de uma fase até uma data. Por exemplo, neste ciclo os alunos receberam ponto extra nas avaliações ao cumprirem as fases dentro das datas estipuladas pelo professor, que normalmente antecedem ao início da explicação de um novo assunto nas aulas. Esse botão permite aceder as recompensas conquistadas (Figura 4.41).

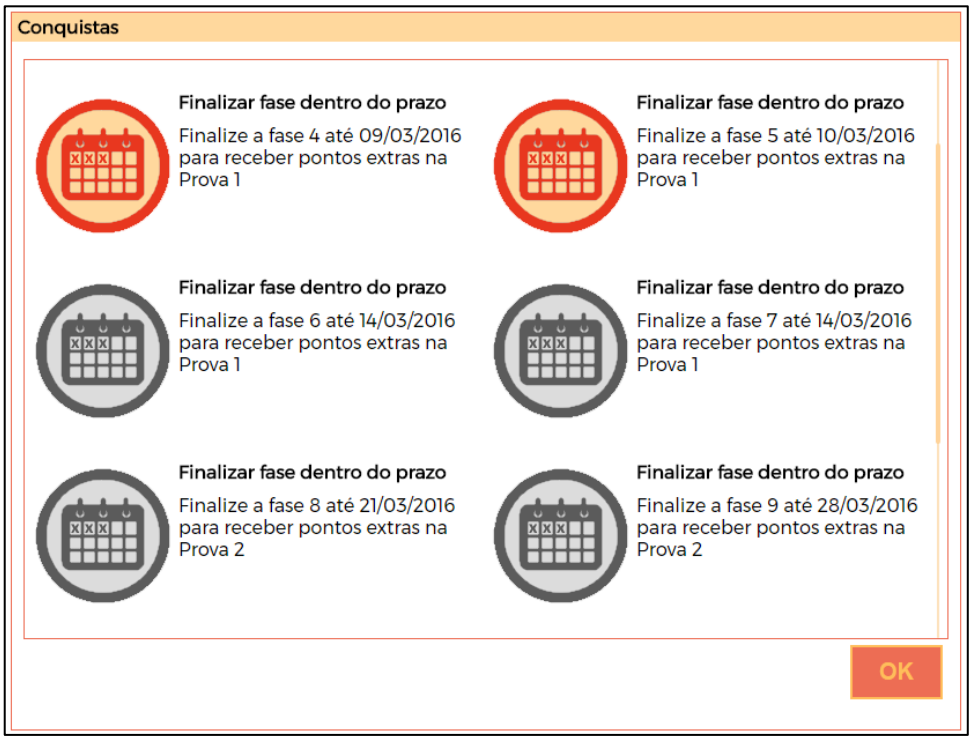


Figura 4.41 – Sistema de Recompensas

O seletor de missões (comparar Figura 4.23 com Figura 4.42) continua sendo acedido ao clicar no edifício da escola do painel de controlo. Pode-se observar no seletor que houve mudança em como as fases são divididas. O projeto instrucional foi revisado para que os assuntos sejam introduzidos mais gradualmente entre as missões. No segundo ciclo existiam 57 missões distribuídas em 5 fases. O aluno precisava jogar todas as missões de uma fase em sequência para poder aceder a próxima. Nessa versão distribuímos 73 missões em uma nova estrutura de fases (Figura 4.43), com menor quantidade de missões em cada uma. Como os três primeiros grupos envolvidos na avaliação anterior responderam que não concordavam que tinham aprendido a importância da depuração (Q6), além de alguns alunos reclamarem por não ter primeiramente entendido o uso do botão para executar passo-a-passo, e outros que não sabiam da seleção múltipla com Copiar e Colar, criamos

uma primeira fase do jogo, com sete missões, para que os alunos aprendam os recursos básicos para edição e se acostumem com o ambiente do jogo. As fases 2 a 4 abordam a manipulação de variáveis, as fases 5 a 7 as estruturas condicionais, as fases 8 a 10 os ciclos e as fases 11 e 12 os arrays. Reduzimos a quantidade de assuntos se comparado com a versão anterior, ao remover o tópico de funções e procedimentos. Consideramos esse tópico fora do escopo do jogo que é vencer as barreiras iniciais da programação no exercício do seu raciocínio lógico.

As setas representam as fases de pré-requisito. Por exemplo, quando o aluno conclui a fase 2, o jogo lhe permite jogar as fases 3, 4 e 5. Isso lhe dá a oportunidade de continuar jogando outras fases mesmo se estiver tendo dificuldade com alguma missão. No seletor, as fases que podem ser jogadas ficam com seus números piscando, as fases concluídas têm seus números em cor branca e as bloqueadas em cinza. Na Figura 4.42 as fases concluídas são a 1, 2 e 5, as liberadas são 3, 5, 6, 7 e 8 e as bloqueadas são 9, 10, 11 e 12.

Figura 4.42 – Seletor de missões no Ciclo 3

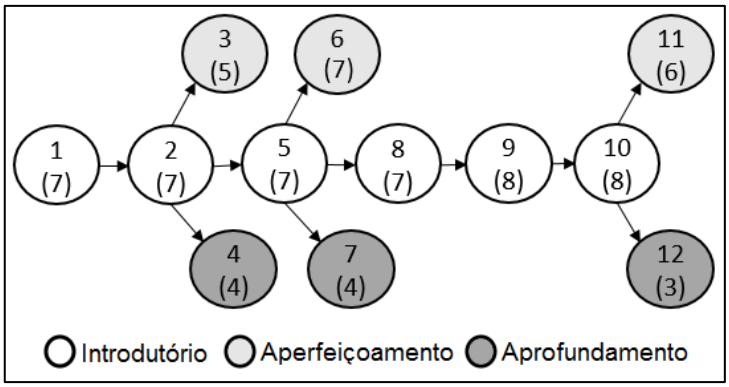


Figura 4.43 – Sequência das Fases no Ciclo 3

Na Figura 4.43 os números entre parêntesis representam a quantidade de missões na fase, e pode-se observar que as fases foram divididas entre três níveis: introdutório, aperfeiçoamento e domínio. As fases introdutórias referem-se ao conhecimento essencial que o aluno precisa dominar em cada assunto. As fases de aperfeiçoamento apresentam mais exemplos do conhecimento aplicado de outras formas. As fases de domínio desafiam o aluno com missões de maior complexidade e menos suporte. Apesar do gráfico representar que existe uma relação de pré-requisito entre as fases 10 com 11 e 12, na verdade isso é apresentado aos alunos como uma sugestão, pois ao concluir a fase 9, essas três já ficam disponíveis aos alunos. A fase 8 trata dos ciclos iterativos (for) e a fase 9 dos ciclos condicionais (while). A fase 10 foi rotulada como introdutória por conveniência para que o aluno tivesse a impressão de que era preciso fazê-la. Porém, nesta fase foram incluídos problemas em que é preciso usar os dois tipos de ciclos aninhados, ou seja, os alunos acabavam por se aperfeiçoarem no assunto de ciclos.

Na versão anterior existiam dois tipos de tarefas nas missões: (1) todos os blocos da solução estavam disponíveis e desordenados na área de trabalho e a tarefa do aluno era organizar esses blocos, e (2) construir a solução. Esses tipos de tarefas poderiam servir para introduzir um assunto (através do tipo organizar) e para praticar um assunto (pelo tipo construir). Porém, constatou-se que a sequência de tarefas não era a melhor e por isso foi feita uma revisão de quando aplicar cada um dos tipos. Além disso, definimos novos tipos de tarefas para alinhar com o momento e o nível de domínio que o aluno está a aprender um novo conteúdo. Uma vez que as fases estão mais curtas e mais focadas num assunto, organizamos para que as primeiras missões apresentem exemplos e boas práticas e nas finais o aluno possa demonstrar o conhecimento aprendido assim como avançar construindo seus próprios significados.

Para nos orientar quanto a essa sequência de tarefas, nos inspiramos na Taxonomia de Bloom Revisada (Anderson et al., 2001) que classifica os objetivos e habilidades segundo o nível de domínio sobre um assunto, iniciando das mais simples, como reconhecimento e lembrança de assuntos para as mais complexas, como a utilização de um conhecimento em novas situações. É possível mapear a Taxonomia de Bloom com as características e modelo dos jogos permitindo que eles sejam mais divertidos, e ao mesmo tempo, incluindo um propósito de aprendizagem como meio de entender melhor como o conhecimento é adquirido (Lameras et al., 2017).

A Tabela 4.12 apresenta essa relação entre os tipos de tarefas e os seis níveis de domínio cognitivo da Taxonomia de Bloom Revisada. Essa sequência é essencialmente respeitada nas missões das fases introdutórias e de aperfeiçoamento. As missões das fases de aprofundamento incluem sempre tarefas categorizáveis a partir da categoria de aplicação. Foi criado um novo tipo de tarefa que é “Corrigir erros”: a missão mostra uma solução que não funciona (contém erros); o aluno pode executar a primeira vez sem custo, e a partir daí tirar suas conclusões e resolver os erros.

Tabela 4.12 – Relação entre as categorias dos domínios cognitivos da Taxonomia de Bloom Revisada e os tipos de tarefas das missões no Ciclo 3

Categorias	Tipos de Tarefas
1. Conhecer **2. Compreender**	**Corrigir erros:** é fornecida a solução com erros. O aluno precisa alterar a variável que está sendo referenciada, trocar operadores de comparações ou lógicos, trocar blocos de posição ou inserir novos blocos. **Organizar blocos:** todos os blocos já estão disponíveis espalhados na área de trabalho e o aluno precisa organizá-los na sequência correta.
3. Aplicar	**Construir iniciando com sugestões:** são fornecidos alguns blocos que podem ser o início, o fim ou uma parte central da solução. O aluno cria a solução, inserindo novos blocos, para completar a parte que falta.
4. Analisar	**Construir:** o aluno cria sua solução a partir do zero.
5. Avaliar **6. Criar**	**Construir com restrições:** as tarefas têm restrições quanto a quantidade de blocos usados, quantidade de vezes que pode ser usado um determinado bloco, e/ou a quantidade de variáveis que podem ser usadas.

As Tabelas 4.13 a 4.24 relacionam todas as missões de cada fase, com seu tipo (CE - Corrigir erros, OB - Organizar blocos, COn – Construir 1-aplicar, 2-analisar e 3-avaliar/criar), as tarefas, os objetivos instrucionais e os novos blocos introduzidos

na missão. Os blocos são baseados nos criados nas duas versões anteriores (Tabelas 4.1 e 4.7) e novos blocos para essa versão (Tabela 4.25).

Tabela 4.13 – Missões da Fase 1 no Ciclo 3

#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 1 – Afiando as facas				
1	OB	Juntar dois blocos na área de trabalho para movimentar até dois clientes	Conectar os blocos, a sequência dos blocos, e executar a solução.	Movimentar até ao cliente
2	CO	Adicionar dois blocos de movimentação	Adicionar um novo bloco na área de trabalho e conectá-lo aos existentes.	
3	-	Usar os botões do ambiente	Ligar/desligar a música, abrir a janela de objetivos e explicações e retornar uma missão.	
4	CO	Copiar, colar e alterar parâmetros dos blocos. Ir até o cliente e depois ao mostrador. Fazer isso para cada cliente	Usar comandos de copiar e colar.	Movimentar até ao mostrador quente
5	-	Clicar no botão de executar passo-a-passo	Conhecer e entender a tarefa de executar passo-a-passo.	
6	OB	Juntar os blocos para movimentar-se aos três clientes, mostrador e refrigerador	Usar recursos de *zoom* e velocidade de animação	Movimentar até ao refrigerador
7	CE	O erro é remover o bloco para movimentar até ao mostrador depois do cliente 1, e adicionar um bloco para movimentar até o refrigerador	Conhecer e entender as tarefas associadas ao tipo de missão Consertar Erros.	

Tabela 4.14 – Missões da Fase 2 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 2 – Anotar pedidos					
1	8	CE	Perguntar aos dois clientes quanto querem comer e guardar nas variáveis. Falar no final a soma. Corrigir o erro das duas vezes que perguntou está sendo salvo na mesma variável	Salvar e aceder variáveis e manipulação dos valores através de soma.	Perguntar quanta comida deseja e Falar
2	9	CE	Perguntar a três clientes quanto querem comer e falar no total no final. Faltam os blocos para fazer a soma	Criar a soma com três variáveis.	
3	10	OB	Perguntar o que cliente deseja comer, ir buscar o cachorro e entregar ao cliente.	Armazenar, aceder e substituir valores de variáveis.	Perguntar o que deseja comer, Pegar um cachorro quente e Entregar
4	11	CO1	Perguntar a dois clientes o que deseja comer e quantas bebidas. Entregar os lanches e falar no final o total das bebidas	Junção das missões 2 e 3.	Perguntar quantas bebidas deseja
5	12	CO2	Perguntar ao cliente o que ele deseja comer e beber, e fazer a entrega de ambos	Salvar e referenciar a variável correta de acordo com a situação.	Movimentar até ao refrigerador e Pegar uma bebida
6	13	CO2	Perguntar a três clientes quanta comida e quantas bebidas desejam. Falar no final a quantidade de comida e a quantidade de bebidas.	Práticas de acesso a variáveis.	
7	14	CO3	O mesmo que da missão anterior, mas com a restrição de usar duas variáveis no máximo.	Uso de acumuladores	

Tabela 4.15 – Missões da Fase 3 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 3 – Aprendiz de caixa					
1	15	CE	Perguntar a três clientes quanto desejam comer e beber. Cada produto tem seu preço unitário. Falar no final o valor total do pedido. Corrigir o erro da variável incorreta referenciada na hora de falar	Usar acumuladores com multiplicação.	Multiplicar
2	16	OB	Dois clientes. Perguntar ao cliente quanto deseja comer e beber. Cobrar dele o valor que sempre será pago acima do valor do pedido. O saldo é guardado como gorjeta. Falar no final o total do dinheiro recebido e o total das gorjetas.	Operações matemáticas com as variáveis	Subtrair e Cobrar do cliente
3	17	CO2	Três clientes. Existe um estoque inicial. Perguntar quanto desejam comer. Falar no final quanto sobrou do estoque.	Operações matemáticas envolvendo constantes	
4	18	CO2	Dois clientes. Perguntar ao cliente quanto deseja comer. Cobrar dele o valor que sempre será pago acima do valor do pedido. O saldo é a gorjeta. Dessa gorjeta, calcular quantas bebidas poderiam ser entregues ao cliente. Falar a quantidade dessas bebidas.	Usar a divisão de inteiros.	Divisão
5	19	CO3	Dois clientes. Perguntar ao cliente quanto deseja comer e beber. Existe um valor de venda e outro de custo. Falar o total de vendas e total de lucro. Restrição de usar no máximo 4 variáveis.	Operações matemáticas com as variáveis	

Tabela 4.16 – Missões da Fase 4 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 4 – Caixa					
1	20	CO3	Dois clientes a serem atendidos. Perguntar o que cliente deseja comer, ir buscar o cachorro e entregar ao cliente. Restrição na quantidade de blocos.	Reconhecer que um mesmo problema pode ser resolvido de diferentes formas.	
2	21	CO2	Perguntar ao cliente quanto deseja comer e beber. Cobrar dele o valor que poderá ser pago acima do valor do pedido. Calcular o troco considerando que somente existem moedas de 1. Devolver o troco em moedas de 1.	Prática com os quatro operadores matemáticos.	Dar o troco e constante moeda de 1.
3	22	CO3	Perguntar ao cliente quanto deseja comer e beber. Cobrar dele o valor que poderá ser pago acima do valor do pedido. Calcular o troco considerando que somente existem moedas de 2. A sobra fica como gorjeta. Devolver o troco em moedas de 2. Falar a gorjeta. Restrição na quantidade de blocos.	Prática com os quatro operadores matemáticos.	Constante moeda de 2.
4	23	CO2	Perguntar ao cliente quanto deseja comer e beber. Cobrar dele o valor que poderá ser pago acima do valor do pedido. Devolver o troco na íntegra e na menor quantidade possível.	Prática com os quatro operadores matemáticos.	Constante nota de 5.

Tabela 4.17 – Missões da Fase 5 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 5 – Aprendiz de Atendente					
1	24	CE	Dois clientes e alguns com sede. Entregar a bebida para quem tem sede. Os dois erros estão nos operadores de comparação	Usar condicionais.	Se-então, operadores de comparação e Perguntar se está com sede.
2	25	CE	Um cliente que pode estar com fome, sede ou ambos. Entregar o pedido conforme a vontade. Todos os blocos de ações já estão no ambiente. Faltam os condicionais e colocar os blocos dentro deles.	Usar condicionais.	Perguntar se está com fome.
3	26	OB	Perguntar ao cliente quantas bebidas deseja. Conforme a quantidade, o preço unitário pode variar. Calcular o preço e cobrar do cliente.	Usar condicionais.	Se-então-senão.
4	27	CO2	Perguntar ao cliente o tipo de bebida que deseja, prepará-la e entrega-la.	Usar condicionais com senão	Movimentar até à caixa de laranjas, Pegar uma laranja, Movimentar até à máquina de sumos, Preparar e pegar um sumo, Constantes bebida e sumo de laranja
5	28	CO2	Dois clientes com fome e/ou sede. Calcular o preço do pedido, com desconto se ele tiver fome e sede. Falar no final o total do pedido.	Usar operadores lógicos nos condicionais.	Operador lógico E
6	29	CO2	Um cliente com fome e/ou sede. Se ele tiver vontade de certa quantidade de lanches ou bebidas, ele recebe de presente um sumo de laranja.	Usar operadores lógicos nos condicionais.	Operador lógico OU
7	30	CO3	Definir o preço unitário da comida e bebida de acordo com a vontade do cliente. Falar o preço unitário. Restrições na quantidade máxima de variáveis e blocos.	Condições aninhadas.	

Tabela 4.18 – Missões da Fase 6 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 6 – Atendente novato					
1	31	CE	Dois clientes com fome e sede. Conforme a quantidade de fome e sede, o cliente ganhar um gelado de chocolate. Final falar a quantidade de gelados entregues. Os erros estavam no operador lógico e na comparação com constantes.	Acumuladores dentro dos condicionais.	Movimentar até à máquina de gelados, Preparar e pegar um gelado e constante Chocolate
2	32	CE	Um cliente. O preço do sorvete e gelado mudava de acordo com a quantidade de comida e sabor do gelado. Os erros estavam no operador lógico, nos valores atribuídos às variáveis e na posição do condicional.	Interpretar e transpor a descrição das condições para a solução.	Perguntar quantos gelados deseja
3	33	CE	Entregar de presente um sabor de gelado de acordo com a quantidade de lanches e bebidas que o cliente deseja. Faltaram os condicionais.	Condicionais aninhados com senão.	Constantes morango, chocolate e baunilha
4	34	OB	Perguntar ao cliente quanta comida, bebidas e gelados deseja. O preço do pedido é diferenciado de acordo com essas quantidades. Falar no final o valor do pedido.	Operadores lógicos OU e E concatenados no mesmo condicional.	Perguntar quantos gelados deseja
5	35	CO2	Entregar ao cliente somente a primeira comida que ele pediu.	Pequeno programa que exige um pouco de atenção nos detalhes.	
6	36	CO2	Dois clientes. Falar quantos clientes tem somente fome e somente sede.	Práticas com operadores lógicos.	
7	37	CO3	Três clientes com fome e sede. Falar no final qual é o maior valor financeiro pedido.	Usar condicionais para descobrir o maior valor.	

Tabela 4.19 – Missões da Fase 7 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 7 – Atendente					
1	38	CO2	Falar a quantidade de clientes na esplanada.	Usar condicionais com acumuladores.	Existe um cliente nessa posição?
2	39	CO2	Falar a média de cachorros pedidos. Podem existir de Zero a Três clientes na esplanada.	Cálculo de médias e divisão por zero. É uma missão que usa tempo para ganhar estrelas. Serve para analisarmos como o tempo interferiu na concentração.	
3	40	CO2	Zero a três clientes. Falar o cliente que teve a maior quantidade de cachorros pedidos.	Descobrir o maior valor.	
4	41	CO3	Entregar um sorvete de brinde ao cliente que mais pediu gelados. Restrição na quantidade de blocos.	Descobrir o maior valor e manter quem foi.	

Tabela 4.20 – Missões da Fase 8 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 8 – Aprendiz de Chef					
1	42	CE	Três clientes que podem ter fome e/ou sede. Falar no final quanta comida foi pedida. Corrigir o erro no passo inicial e final do ciclo, e na variável acumuladora.	Uso do ciclo iterativo (for)	Para
2	43	CE	Um cliente. Entregar todos os lanches pedidos pelo cliente. Corrigir o erro na variável referenciada para final do ciclo	Valor final variável do ciclo.	
3	44	OB	Três clientes que podem ter sede. Entregar a bebida para eles.	Condicionais aninhados dentro dos ciclos.	
4	45	CO1	Três clientes. Cobrar o valor do pedido. A sobra fica como gorjeta. No final falar o total da gorjeta. Restrição na quantidade de blocos.	Operações matemáticas e acumuladores dentro do ciclo.	
5	46	CO2	Três clientes. Perguntar a quantidade de lanches e no final falar a média de lanches pedidos. Restrição na quantidade de blocos.	Cálculo de média usando ciclos. Considerar que nenhum cliente pode ter pedido lanches (divisão por zero).	
6	47	CO2	Entregar gelados para todos os clientes conforme a quantidade de lanches pelo primeiro cliente. Restrição na quantidade de blocos.	Condicionais aninhados dentro de outros condicionais.	
7	48	CO3	Perguntar aos clientes quantos lanches desejam. No final falar a maior quantidade e quem pediu. Restrição na quantidade de blocos e variáveis.	Descobrir o maior valor e manter quem foi.	

Tabela 4.21 – Missões da Fase 9 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 9 – Estagiário de Chef					
1	49	CE	Entregar os cafés pedidos pelo cliente. Corrigir o erro de sinal no contador de cafés.	Valor final variável do ciclo.	Movimentar até à máquina de café, Preparar dois cafés e Pega um café
2	50	CE	Três clientes que podem ter fome e/ou sede. Falar no final quanta comida foi pedida. Corrigir o erro no condicional do ciclo, na inicialização da variável usada no controlo do ciclo e na atribuição da variável acumuladora.	Uso dos ciclos condicionais.	Repita enquanto.
3	51	OB	Três clientes com fome, sede, ambos, ou nada. Perguntar pela comida e somar a quantidade de lanches desejados. Parar de ir aos clientes quando tiver alcançado 4 cachorros.	Uso dos ciclos condicionais.	
4	52	CO1	Três clientes com fome, sede, ambos, ou nada. Perguntar e calcular o total de pedido. Parar de ir aos clientes quando tiver atingido $7. Restrição na quantidade de blocos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Uso dos ciclos condicionais.	
5	53	CO2	Enquanto o cliente tiver sede, entregar todas as bebidas. Restrição na quantidade de blocos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Condicionais aninhados dentro dos ciclos condicionais.	
6	54	CO2	Três clientes com sede de sumo. Entregar todo o sumo pedido. Restrito a usar somente um bloco de ciclo na solução. Restrição na quantidade de blocos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Exige raciocínio para o controlo no movimento do atendente.	
7	55	CO2	Três clientes com fome e sede. Entregar os pedidos para os clientes até atingir exatamente $12. Restrição na quantidade de blocos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Como na missão anterior com um pouco mais de complexidade.	
8	56	CO3	Três clientes. Entregar somente para aquele que pediu a maior quantidade de lanches. Restrição na quantidade de blocos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Dois ciclos na sequência.	

Tabela 4.22 – Missões da Fase 10 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 10 – Chef novato					
1	57	CE	Três clientes e entregar todo o lanche. Corrigir a variável que controla a quantidade de iterações no ciclo interno e variável referenciada para movimentar o atendente no ciclo interno.	Ciclos iterativos aninhados	
2	58	CE	Entregar os lanches aos clientes até atingir 5 cachorros entregues. Não interromper o atendimento do cliente. Faltaram o contador de posição do cliente e a soma de cachorros atendidos.	Ciclo iterativo aninhado em um ciclo condicional	
3	59	CO1	Entregar todos os pedidos de lanches e gelados dos três clientes. Restrição na quantidade de blocos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Ciclo condicional aninhado em um ciclo iterativo	
4	60	OB	Entram e saem clientes que sentam nos bancos desocupados. Eles vêm com fome. Atender por completo dois clientes	Ciclo condicional aninhando condicionais que tem aninhado ciclos iterativos.	
5	61	CO1	Entram e saem clientes que sentam nos bancos desocupados. Eles vêm com fome. Atender os clientes até alcançar 8 lanches entregues. Não deixar nenhum cliente atendido parcialmente. Restrição na quantidade de blocos.	Ciclo condicional aninhando condicionais que tem aninhado ciclos iterativos.	
6	62	CO1	Entram e saem clientes que sentam nos bancos desocupados. Eles vêm com fome. Atender os clientes até alcançar 7 lanches entregues. Interromper o atendimento quando alcançada essa quantidade. Restrição na quantidade de blocos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Ciclos condicionais com operadores lógico E.	
7	63	CO1	Entram e saem clientes que se sentam nos bancos desocupados. Eles vêm com fome e sede. Atender os clientes até alcançar 5 lanches ou 3 bebidas entregues. Interromper o atendimento quando alcançada essa quantidade. Restrição na quantidade de blocos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Ciclos condicionais com operadores lógico OU.	

8	64	CO1	Entram e saem clientes que sentam nos bancos desocupados. Eles vêm com fome e sede. Entregar os pedidos para os clientes até atingir exatamente $26. Interromper o atendimento quando alcançada esse valor. Restrição na quantidade de blocos. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Ciclos condicionais com vários operadores lógicos E encadeados.	

Tabela 4.23 – Missões da Fase 11 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 11 – Chef					
1	65	CE	Salvar no array o pedido de bebida de cada um dos três clientes. No final falar o array. Corrigir o índice usado para armazenar no array.	Criação e armazenamento de arrays.	Criar array e alterar um valor no array.
2	66	OB	Salvar no array o pedido de bebida de cada um dos três clientes. A partir da iteração no array, obter quais foram as bebidas pedidas e pegá-las no refrigerador. No final falar o array.	Aceder os valores do array.	Obter valores do array.
3	67	CO1	Criar os arrays com tamanhos conforme a quantidade de bebidas desejadas. Armazenar esses desejos no array e em seguida falar o array. Restrição na quantidade de variáveis. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Criar arrays com tamanho conforme a necessidade e navegar por eles respeitando esse tamanho.	Tamanho do array e Qual a n-ésima bebida que tem vontade ?
4	68	CO2	Anotar os pedidos de bebida em um array e noutro array as bebidas prontas. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Manipular dois arrays.	
5	69	CO2	Armazenar num array os pedidos de comida e bebida (todos os clientes tem fome e sede). Falar esse array. Depois no mesmo array as bebidas e lanches preparados. Falar esse array. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Usar o mesmo array para armazenar mais de um tipo de informação.	
6	70	CO2	Mesmo que da missão anterior. Mas os clientes podem ter fome e/ou sede. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Criar e manipular um array em que é incerta a quantidade de valores que serão armazenados	

Tabela 4.24 – Missões da Fase 12 no Ciclo 3

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 12 – Chef master					
1	71	CO2	Perguntar a cada cliente o que deseja comer e beber. De acordo com o pedido, armazene o preço do pedido no array. Falar esse array. Calcular a média, o menor e o maior valor e salvar esses valores num outro array. Falar esse array. Restrição na quantidade de variáveis. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Usar arrays em vez de variáveis simples. Calcular média, menor e maior valores de um array.	
2	72	CO2	Anotar em um array os preços unitários (de $1 a $4) dos produtos pedidos pelos clientes. Falar esse array. Calcular a frequência dos valores nesse array e armazenar em outro array. Falar esse array. Restrição na quantidade de variáveis. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Calcular a frequência dos valores de um array.	
3	73	CO2	Cinco clientes, com fome e/ou sem sede. Calcular o valor do pedido e guardar em um array. Depois ordenar esse array de forma crescente. No final falar o array. Ganha bónus em dinheiro se criar uma solução com uma certa quantidade de blocos.	Ordenação com método bolha.	

Tabela 4.25 – Lista de novos blocos de ação da personagem no Ciclo 3

Bloco	Significado
	Movimentar até à máquina de café
	Preparar dois cafés
	Pegar um café
	Qual a n-ésima bebida que tem vontade?

Em relação à configuração do avatar, além da melhoria visual (comparar Figura 4.21 com Figura 4.44), foi adicionado um novo tipo de roupa, e os acessórios que são habilitados quando é alcançado um montante em moedas.

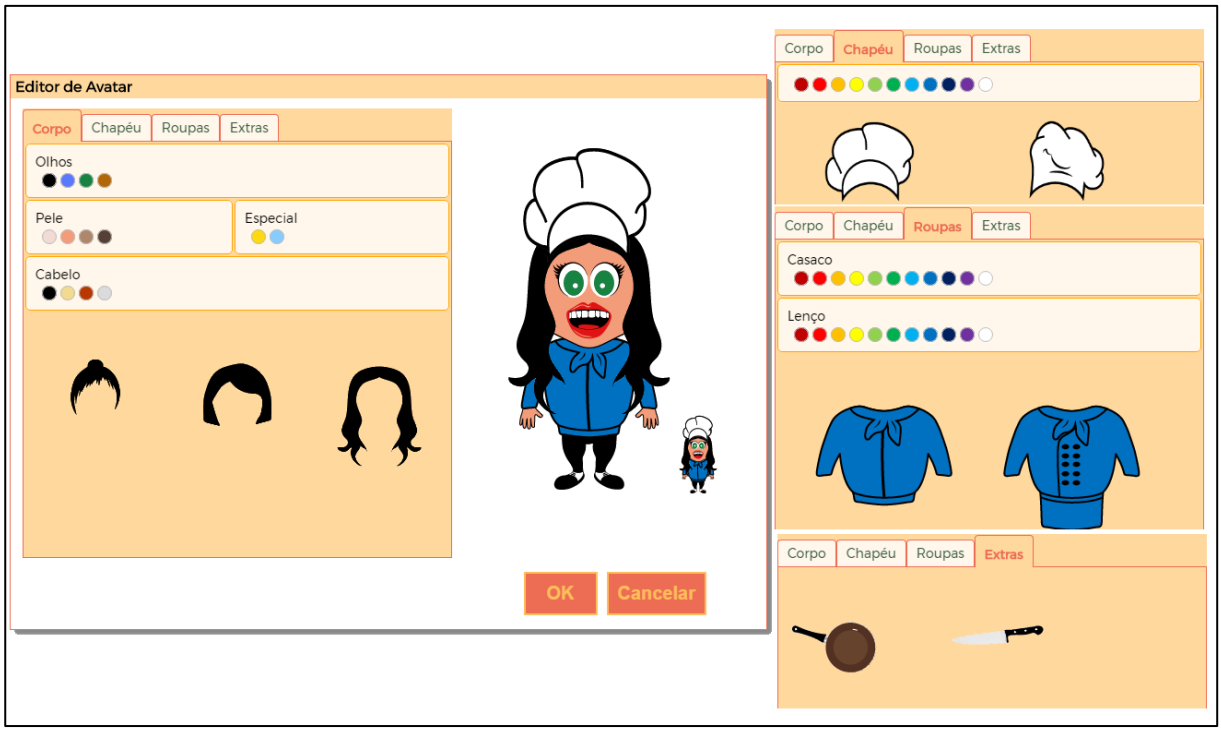


Figura 4.44 – Configuração do avatar no Ciclo 3

Como mencionado anteriormente, os alunos ganhavam pontos em função do tempo que demoravam a resolver cada missão. Verificou-se que essa não era uma boa estratégia, pois vários alunos mostraram dificuldades em se concentrar para resolver o problema, focando-se antes no tempo disponível. Assim, neste terceiro ciclo modificámos a forma de ganhar pontos, passando a considerar o número de tentativas. A tentativa representa a execução ou início da depuração de uma solução. A Figura 4.45 ilustra o projeto visual do contador de tentativas. No exemplo, cada estrela está associada com três tentativas. O aluno já perdeu uma estrela e mais duas tentativas da segunda estrela. Cada estrela ganha lhe concederá ao final 70 pontos de experiência.

Figura 4.45 – Ganhar pontos pela quantidade de tentativas

Para auxiliar os alunos que não conseguem avançar na missão, o jogo fornece a solução em pseudocódigo com o custo de consumir todas as estrelas da missão. O jogo mostra na lateral direita a solução (Figura 4.46) quando o aluno clicar nesse botão. O aluno pode ler, interpretar identificando os blocos a serem usados para cada linha e montar a sua solução.

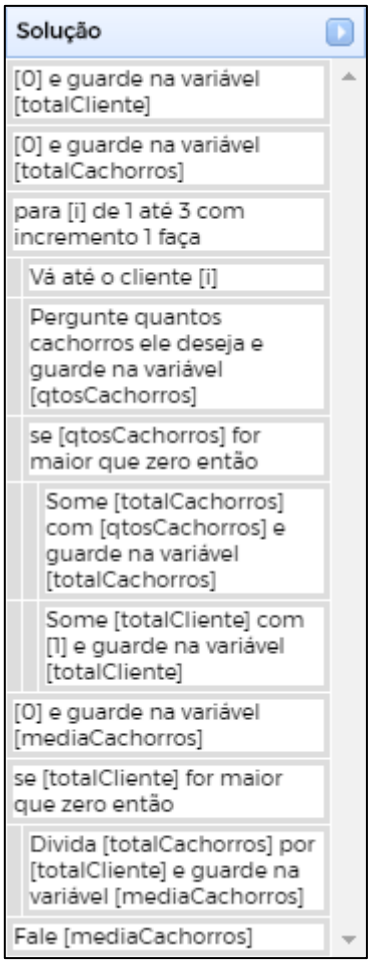


Figura 4.46 – Solução em Pseudocódigo

O novo ambiente do jogo está ilustrado na Figura 4.47 (comparar com a Figura 4.25). As modificações seguiram tanto as sugestões de alunos como a nossa perceção da necessidade de melhorar a usabilidade no ambiente:

1- Nessa área ficaram somente os botões de controlo da animação (executar, depurar e interromper a execução/depuração), e a apresentação da quantidade de testes a serem executados na solução com a quantidade de testes bem-sucedida;

2- A janela com os objetivos está sempre visível, sendo restaurada ou minimizada (ao clicar no ícone *+i*) e arrastada pelo ambiente, o aluno consegue acompanhar o cumprimento dos objetivos enquanto o programa está a executar;

3- Os preços dos produtos pertinentes à missão ficam o tempo todo visíveis em um quadro na esplanada. Assim o aluno não precisa abrir a janela de explicação;

4- Na versão anterior, sempre que terminava uma missão, o jogo retornava à tela de seleção de missões, exigindo cliques adicionais para ir para a próxima missão. Isso era necessário devido à pressão de tempo para resolver a missão. Como modificámos a forma de ganhar pontos, então quando o aluno finaliza uma missão, a próxima já é aberta. Além disso, ele podia alternar entre as missões anteriores e a corrente. Pode-se observar as elipses preenchidas e marcadas (são aquelas já concluídas), a elipse preenchida sem marcação (é a missão corrente ainda não concluída) e as demais elipses não preenchidas são aquelas ainda não acessíveis;
5- Foi incluído um botão para ligar e desligar a música ambiente;
6- Nas missões onde se pede para corrigir erros, é permitido ao aluno restaurar para a solução inicial, com erro, da missão. Esse recurso é necessário quando o aluno trabalhou na solução, mas seguiu um caminho que não leva à solução;
7- Foi adicionado um botão para abrir o pseudocódigo nas missões do tipo construir das fases introdutórias e aperfeiçoamento.
8- Abre a janela de recompensas (Figura 4.41).

Todos os botões não relacionados com a animação foram transportados para a parte superior. O botão 9 abre a janela de explicação da missão, o botão 10 retorna ao painel de controlo e o botão 11 sai do jogo.

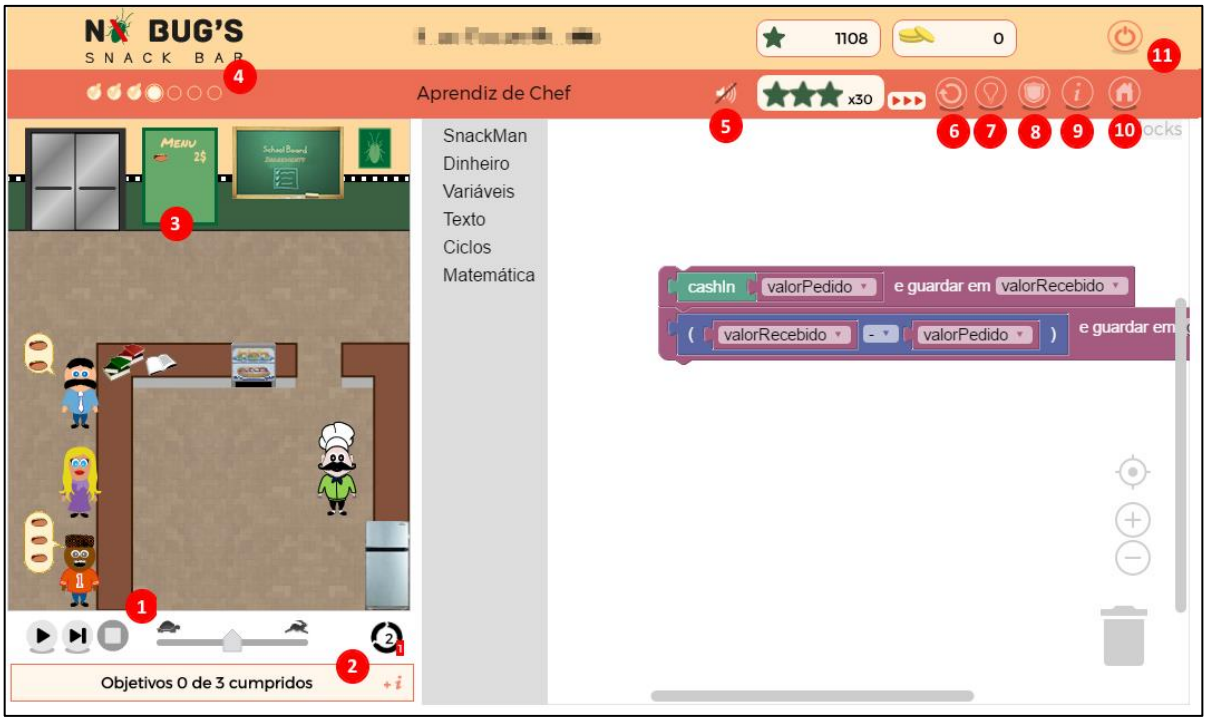


Figura 4.47 – Ambiente do Jogo NoBug's SnackBar no Ciclo 3

O material instrucional foi incorporado na explicação das missões, eliminando a consulta separada a esse material. Como acreditávamos serem importantes as explanações sobre o significado e o funcionamento dos blocos de programação (variáveis, condicionais, ciclos e arrays), nesta versão inserimos essa informação no fluxo normal de leitura do aluno. A Figura 4.48 apresenta o exemplo da explicação de uma missão. As primeiras duas páginas dizem respeito à explicação o que é uma variável, sua representação e manuseio com blocos. A terceira explica os blocos novos de ações da personagem e a última é sobre o objetivo da missão.

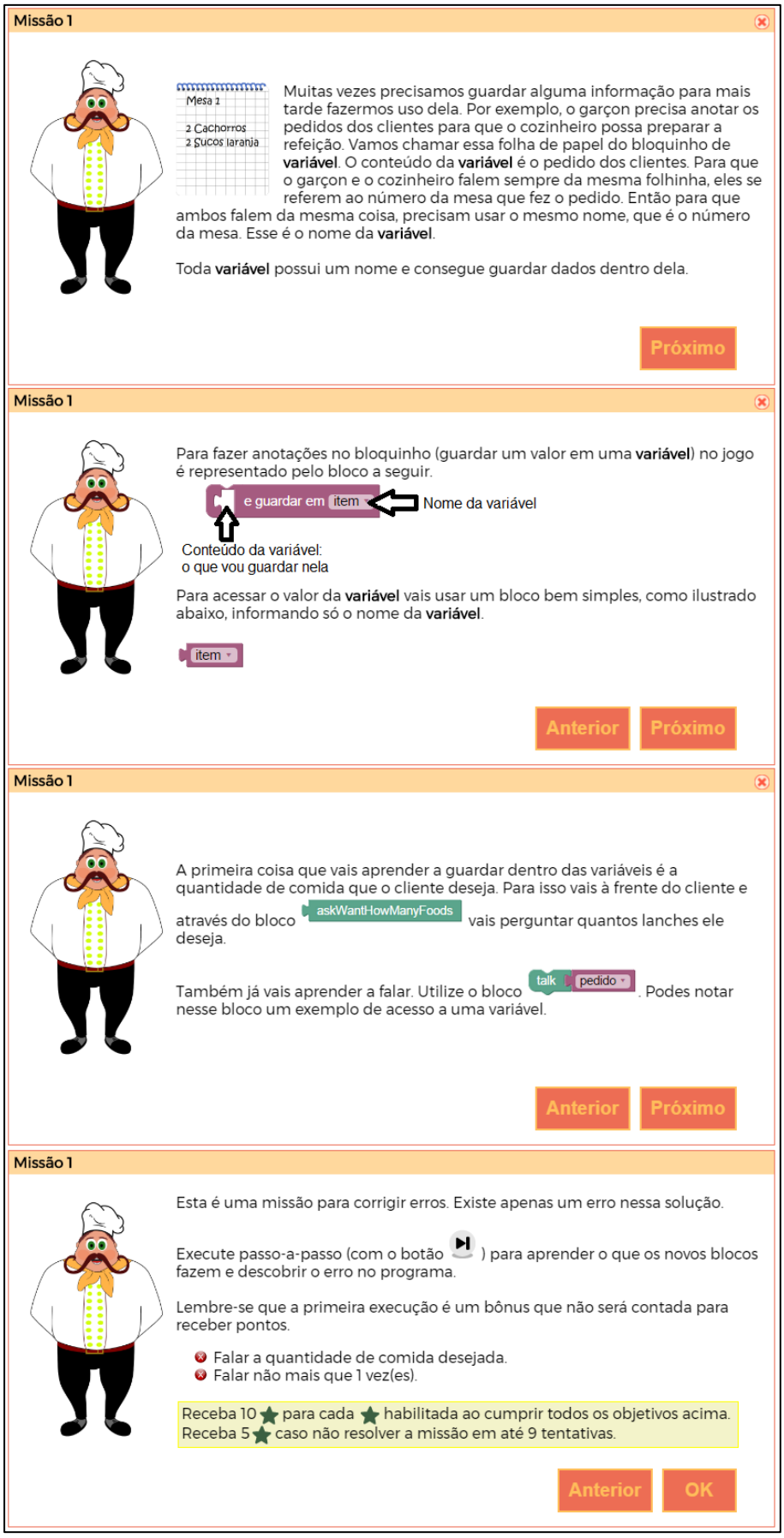
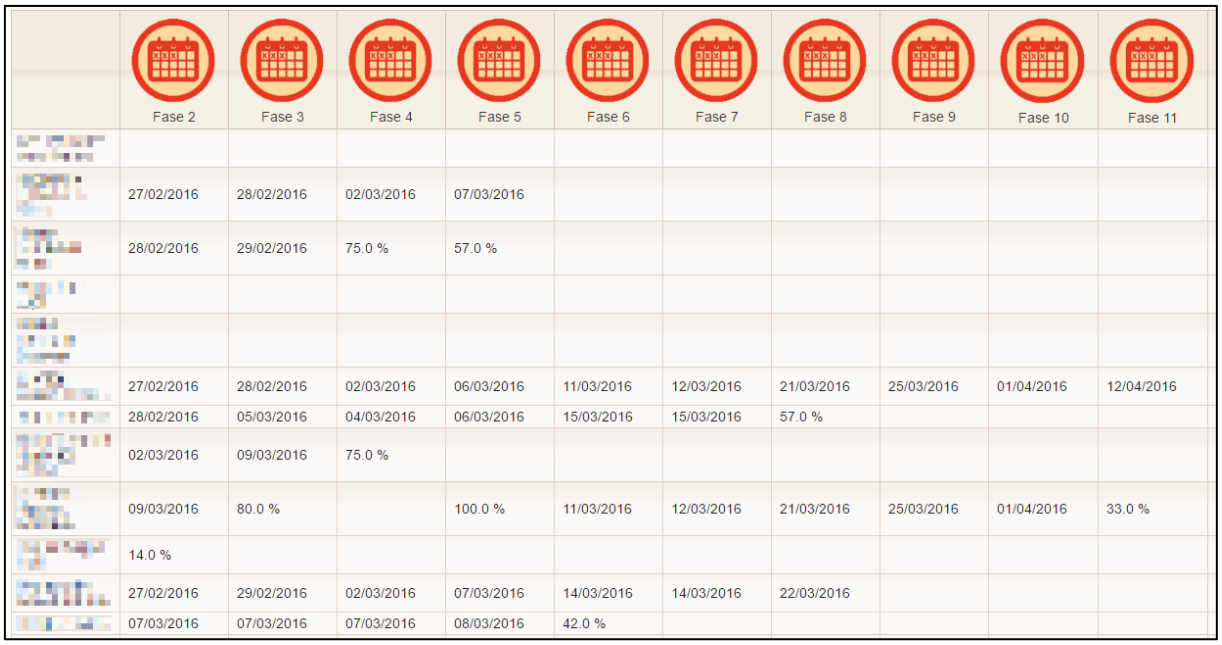


Figura 4.48 – Janelas da explicação da missão

Na área administrativa do professor foi criada uma página que permite acompanhar o estado das recompensas (Figura 4.49). O registro da data implica em conclusão da fase dentro do prazo e por consequência, merecedor da conquista. Está ilustrado um aluno com 100% da Fase 5, indicando que ele finalizou a fase, mas fora do prazo estabelecido. Outros percentuais indicam que o aluno está a meio da fase.



	Fase 2	Fase 3	Fase 4	Fase 5	Fase 6	Fase 7	Fase 8	Fase 9	Fase 10	Fase 11
	27/02/2016	28/02/2016	02/03/2016	07/03/2016						
	28/02/2016	29/02/2016	75.0 %	57.0 %						
	27/02/2016	28/02/2016	02/03/2016	06/03/2016	11/03/2016	12/03/2016	21/03/2016	25/03/2016	01/04/2016	12/04/2016
	28/02/2016	05/03/2016	04/03/2016	06/03/2016	15/03/2016	15/03/2016	57.0 %			
	02/03/2016	09/03/2016	75.0 %							
	09/03/2016	80.0 %		100.0 %	11/03/2016	12/03/2016	21/03/2016	25/03/2016	01/04/2016	33.0 %
	14.0 %									
	27/02/2016	29/02/2016	02/03/2016	07/03/2016	14/03/2016	14/03/2016	22/03/2016			
	07/03/2016	07/03/2016	07/03/2016	08/03/2016	42.0 %					

Figura 4.49 – Página do professor para acompanhar as recompensas

4.7.2 Avaliação do Terceiro Protótipo

No segundo ciclo tivemos uma baixa adesão dos alunos ao jogo. Além de todas as melhorias nesta versão, também modificámos como o jogo foi integrado na disciplina. Para o professor da disciplina foi desenvolvido um material para apresentar em sala, introduzindo todos os temas através do uso de blocos. Além disso, através do sistema de recompensas, o professor bonificava nas provas dos alunos que concluíssem a fase até a data limite. Como o assunto da primeira prova envolvia variáveis e condicionais, então as conquistas até a fase 7 atribuíam bonificações para essa prova. A segunda prova estava relacionada com os restantes assuntos do jogo.

Os objetivos neste ciclo foram os mesmos do anterior: **(1) mensurar o prazer dos alunos com o jogo**, **(2) identificar lacunas para melhorar a sua diversão, (3) verificar se os desafios na sequência das missões oferecem o sentimento de fluxo**, e **(4) identificar se o jogo contribuiu para a aprendizagem do aluno**. A

intenção foi verificar que efeitos poderiam surtir as mudanças efetuadas no jogo e na forma de integrá-lo na disciplina.

Os dados parciais dessa secção foram publicados no seguinte trabalho:

- Vahldick, A., Mendes, A. J., Marcelino, M. J., & Farah, P. R. (2016). Pensamento Computacional Praticado com um Jogo Casual Sério no Ensino Superior. In *XXIV Workshop sobre Educação em Computação*. Porto Alegre, Brasil.

4.7.2.1 População

O ciclo aconteceu entre 26 de Fevereiro e 22 de Abril de 2016 envolvendo uma turma com um total de 36 alunos (86,1% homens e 13,9% mulheres) do Bacharelado em Engenharia de Software da Universidade do Estado de Santa Catarina, no Brasil. A idade média deles era 19,9 anos (± 3,54). Em relação aos hábitos de jogar, 33,3% declarou que jogava diariamente. No outro extremo, 33,3% declarou que jogava raramente. Quanto ao conhecimento prévio em programação, 77,8% deles declarou não ter qualquer experiência ou contato prévio.

4.7.2.2 Metodologia e Instrumentos

Todos os alunos foram registados no jogo. O professor da disciplina apresentou o jogo em laboratório na sua segunda aula no semestre. No primeiro acesso de cada aluno, o jogo apresentava um inquérito (Apêndice B) para identificarmos alguns dados demográficos e os seus hábitos de jogo. Os alunos puderam nessa aula responder o questionário e fazer as missões da primeira fase (ambientação ao jogo). O professor apresentou o sistema de recompensas em que os alunos concluindo algumas fases até certa data receberiam pontos extras nas provas. Essas datas estavam programadas para as fases introdutórias serem jogadas antes de o assunto correspondente ser apresentado na aula, e as fases de aperfeiçoamento e aprofundamento durante o tratamento do assunto nas aulas. Nas aulas seguintes, o professor utilizava o material que havíamos disponibilizado para introduzir qualquer novo assunto, primeiro apresentando na representação de blocos e depois relacionando com a linguagem de programação adotada (Java).

Aplicámos dois inquéritos durante a investigação. O primeiro inquérito (Apêndices G e H) foi aplicado após a 15ª missão avaliando alguns aspetos da

diversão e a aprendizagem percecionados pelos alunos. O segundo (Apêndice I), após a 20ª missão, era uma versão reduzida do EGameFlow. A intenção de medir no meio do ciclo foi para termos uma noção do sentimento dos alunos. Além disso, na conclusão do ciclo foi aplicado outro inquérito (Apêndice J) para avaliar o sentimento final dos alunos. O questionário foi diferente para cada grupo, com questões direcionadas conforme a experiência de jogo, para identificar características que lhes agradaram e aquelas que precisavam ser melhoradas.

Durante todo o jogo os alunos tinham sido monitorizados, tal como no ciclo anterior. Isso permitiu o envio de dicas e ajudas por mensagens de correio eletrónico, sempre que necessário. Essas mesmas dicas foram sendo adicionadas nas missões para auxiliar os alunos que ainda as não tinham jogado.

Foram analisados o *log* de interações para identificar a frequência de entrada e acompanhar a evolução dos alunos no jogo, a sequência de tentativas, o uso do assistente com pseudocódigo e avaliado o nível de dificuldade de cada missão conforme a quantidade de tentativas dos alunos.

4.7.2.3 Análise de Resultados

4.7.2.3.1 Adesão ao Jogo

Para verificar a adesão dos alunos a esta nova versão do jogo, foi contabilizada a quantidade de alunos que concluíram cada uma das 73 missões (Figura 4.50). Metade da turma concluiu até a 35ª missão (47,8%), que era praticamente a metade do total de missões e que correspondia à primeira missão da sétima fase. Isso demonstrou um acréscimo considerável na adesão do jogo se comparado com o ciclo anterior. A missão 23 teve menos concluintes que as missões posteriores. Esta missão correspondia à última missão da quarta fase, de domínio, que era opcional para o percurso dos alunos. Tanto esse tipo de fase, quanto de aperfeiçoamento, eram fases opcionais, e por essa figura podemos verificar que ainda assim os alunos tentaram seguir a sequência de fases. A motivação pela bonificação nas provas pode ter sido o impulsionador para que eles tentassem concluí-las.

Figura 4.50 – Relação entre a quantidade de jogadores que concluiu cada missão no ciclo 3

4.7.2.3.2 Sequência de Conclusão das Missões

A estrutura das fases, menos rígida do que no ciclo anterior, permitiu que os alunos não ficassem “presos” no jogo por não conseguirem concluir uma missão. Isso pode ser verificado na Figura 4.51 que ilustra a sequência que foi adotada pelos alunos para concluírem as missões. Pela espessura das linhas pode-se constatar o número de alunos que decidiu seguir um determinado caminho. As elipses indicam as missões, e algumas delas contém dois números (por exemplo 16_17, 26_28 e 33_36), indicando que **todos os alunos** sempre concluíram essas missões nessa sequência. As fases 1 e 2 são obrigatoriamente sequenciais. Após a conclusão destas fases todos os alunos passaram para a primeira missão da fase 3 (a missão 15). Apesar de possível, não houve alunos a passar diretamente da fase 2 para a fase 5, mesmo sendo ambas introdutórias. Ou seja, todos tentaram evoluir um pouco mais no assunto de variáveis ao jogar as fases 3 (aperfeiçoamento) e 4 (aprofundamento) antes de fazer tentativas na fase 5.

As Figuras 4.52 a 4.60 apresentam o número de tentativas em cada missão das fases 3 a 11 considerando apenas daqueles alunos que concluíram as missões. A fase 12 não foi considerada porque 2 das suas 3 missões alcançaram somente 2 conclusões. As figuras estão limitadas com seus valores até 60, pois o limite superior maior encontrado foi de 57 na fase 4. Todas as discrepâncias acima de 60 são anotadas nos gráficos com uma seta seguida dos valores discrepantes. O intuito de

apresentar a variabilidade das missões, é propormos uma comparação entre essa variabilidade e o número de tentativas das missões que foram concluídas em sequência (elipses com mais de uma missão na Figura 4.51). Por exemplo, as missões 16 e 17 (Figura 4.52) tiveram uma variabilidade muito baixa, assim como a mediana da quantidade de tentativas. O mesmo pôde ser observado entre as missões 33 a 36 (Figura 4.55). A partir dessa comparação, podemos sugerir que enquanto os alunos estão vencendo, eles mantêm a sequência das missões oferecidas pelo jogo. Nenhum aluno iniciou as fases do assunto de ciclos (fase 8) sem ao menos ter encerrado as fases de aperfeiçoamento (fase 6) ou domínio (fase 7) sobre condicionais. As fases 8 e 9 (missões 42 a 56) são obrigatoriamente sequenciais. Os alunos terminaram a fase 9 e concluíram ao menos a primeira missão (de número 57) da fase 10. De maneira geral, é possível observar que apesar da liberdade na sequência das fases a maioria preferiu seguir a sugestão de sequência da imagem apresentada no seletor de missões (Figura 4.42).

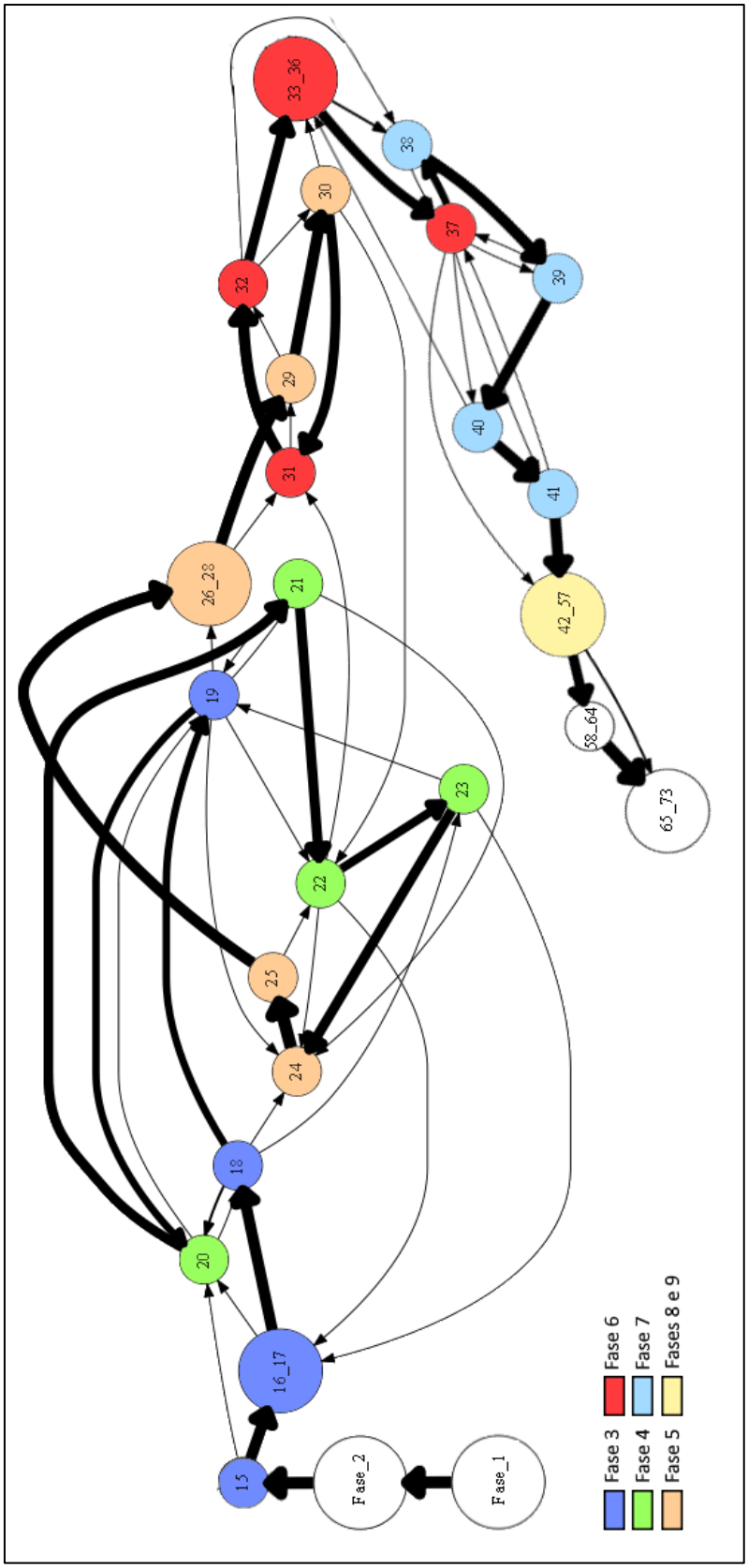


Figura 4.51 – Sequência de Finalização das Missões

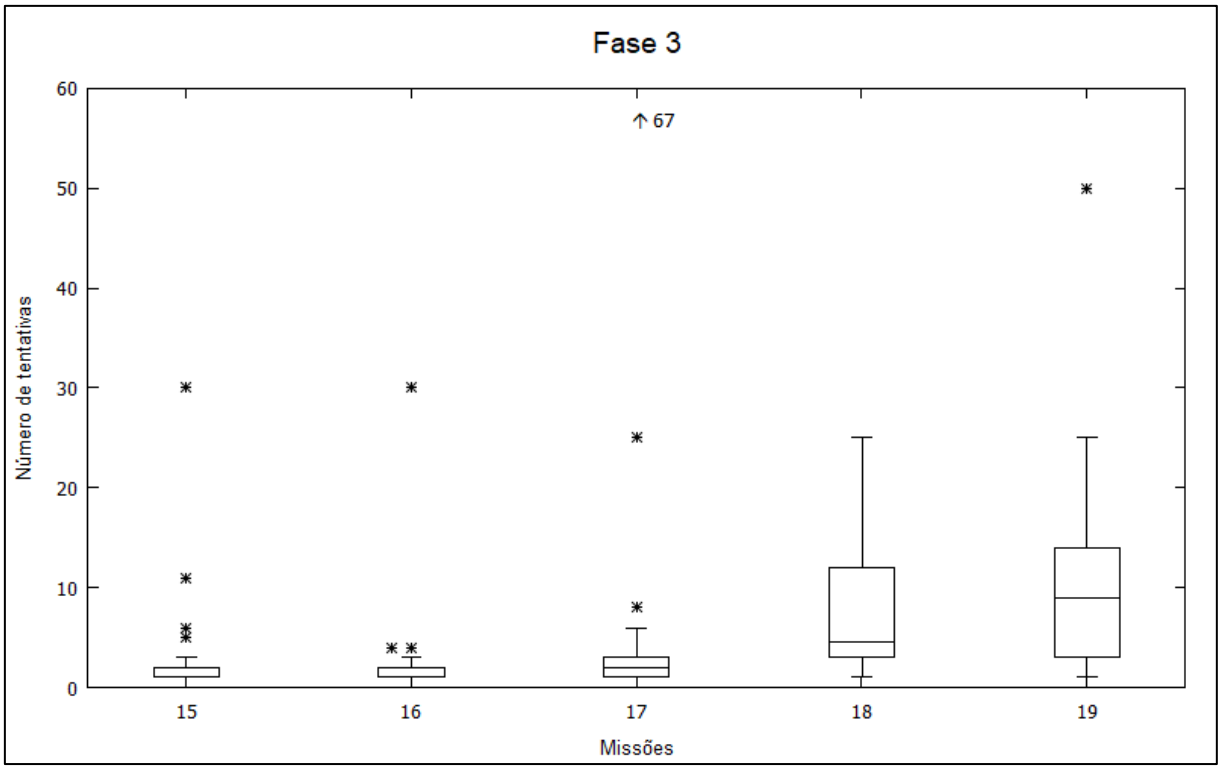


Figura 4.52 – Tentativas na Fase 3 no Ciclo3

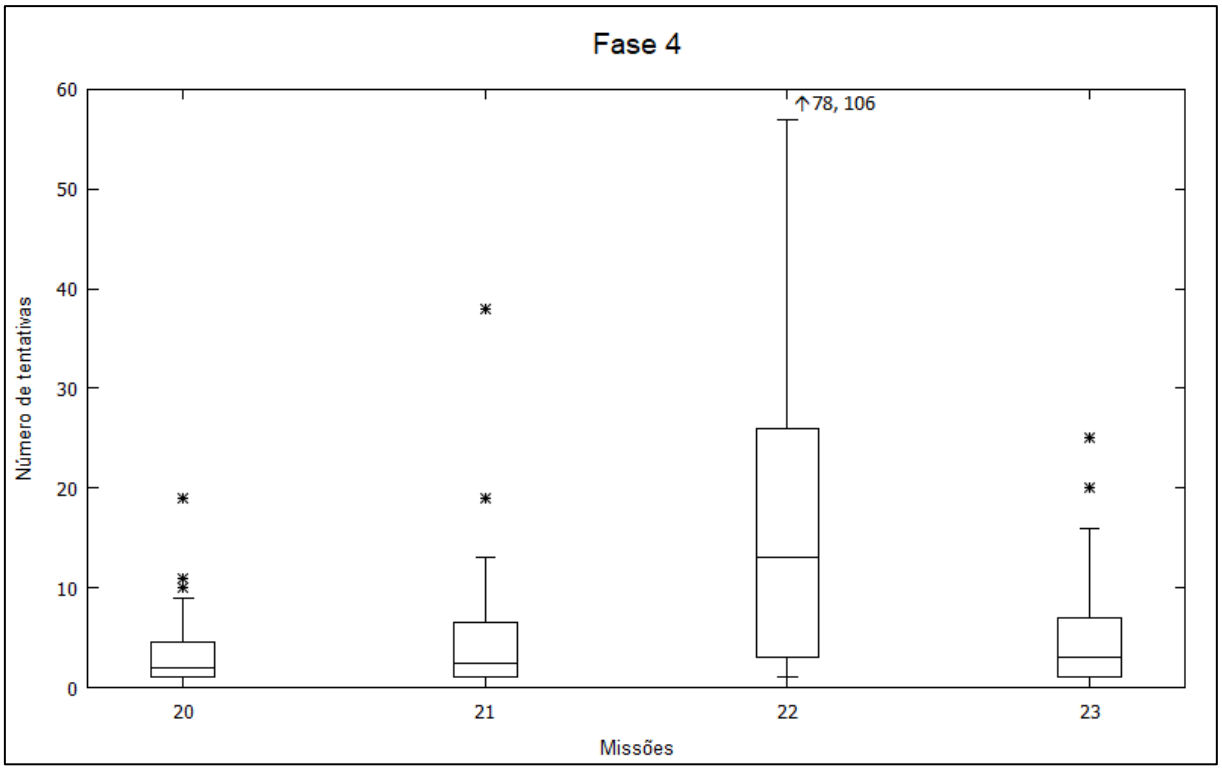


Figura 4.53 – Tentativas na Fase 4 no Ciclo 3

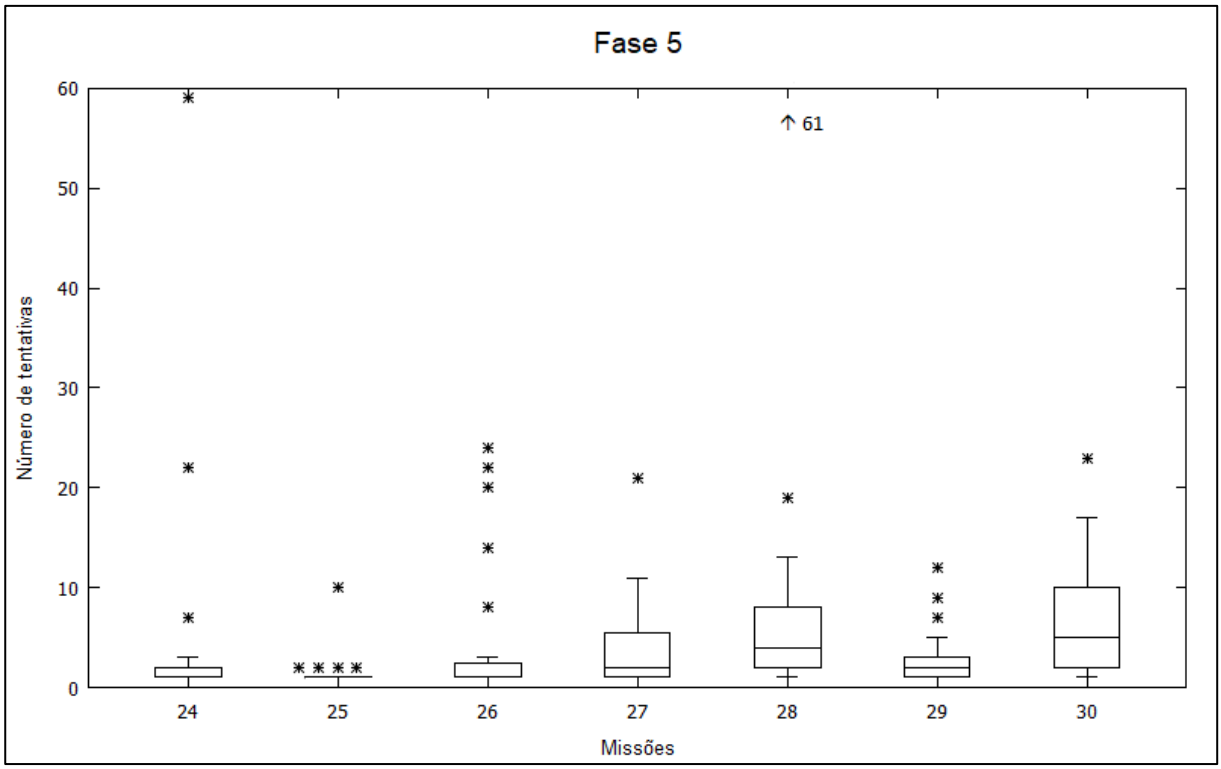


Figura 4.54 – Tentativas na Fase 5 no Ciclo 3

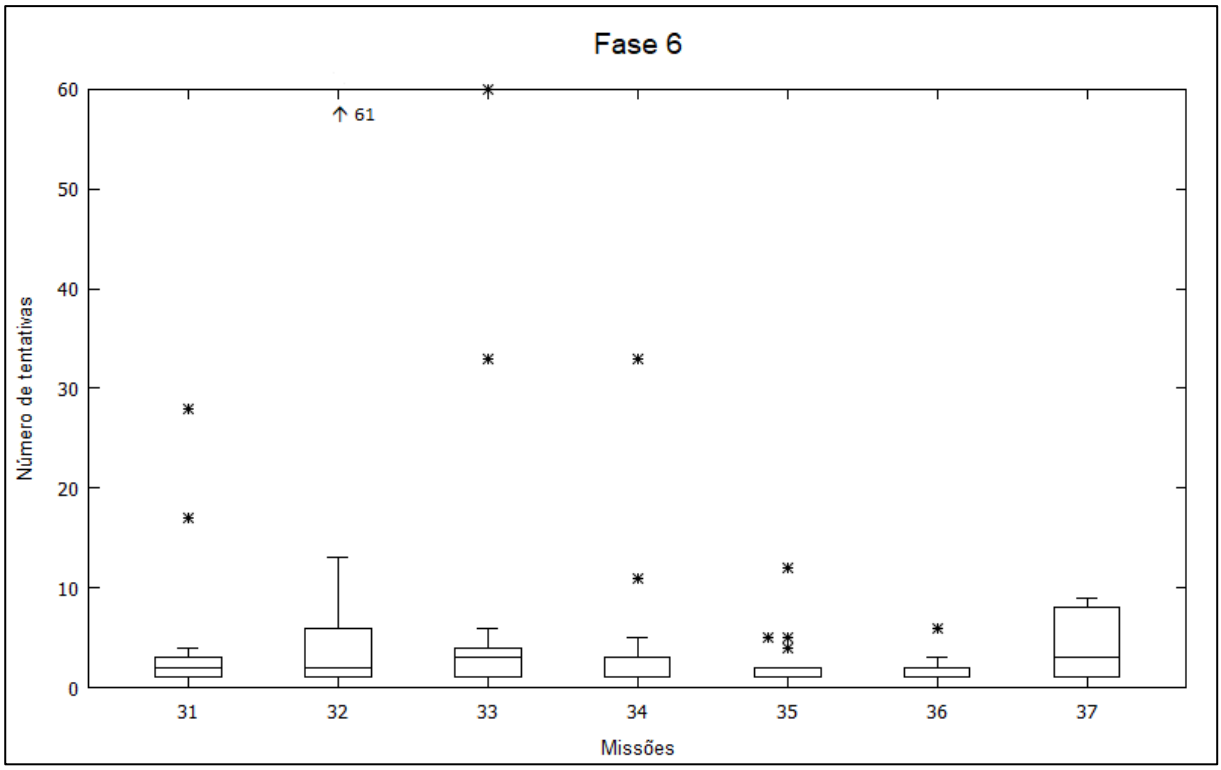


Figura 4.55 – Tentativas na Fase 6 no Ciclo 3

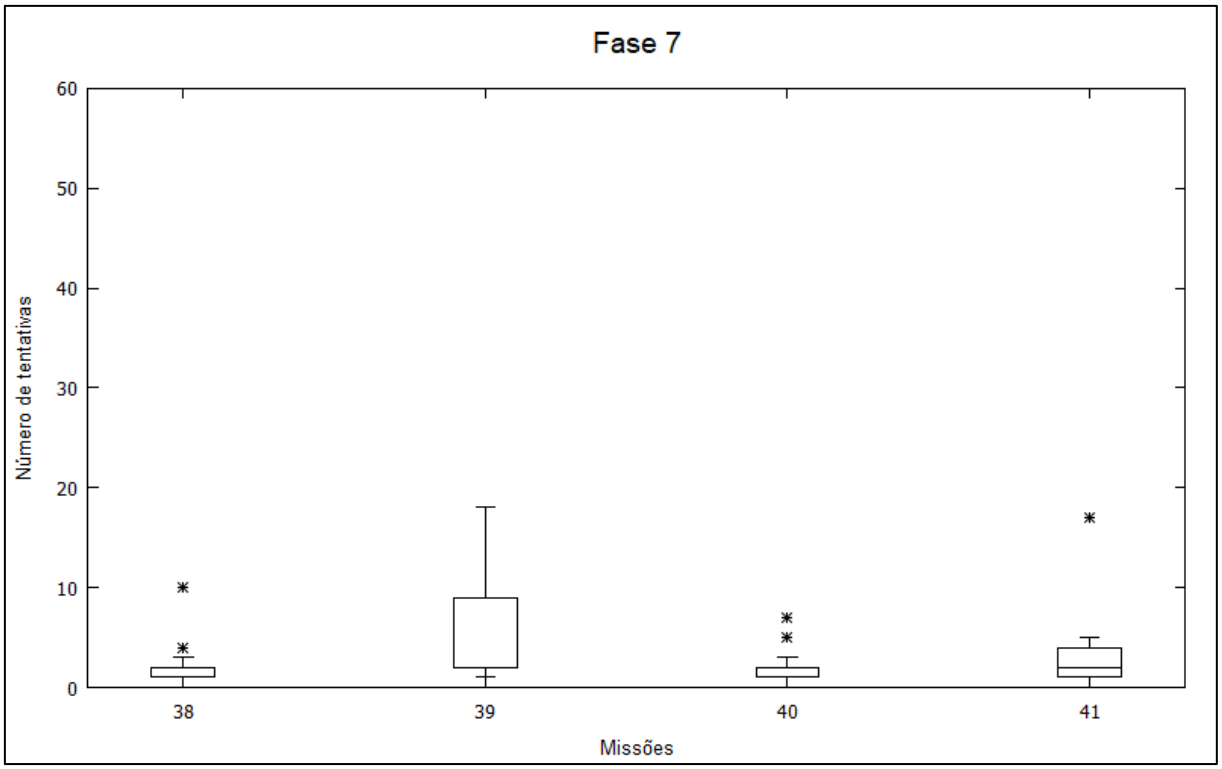


Figura 4.56 – Tentativas na Fase 7 no Ciclo 3

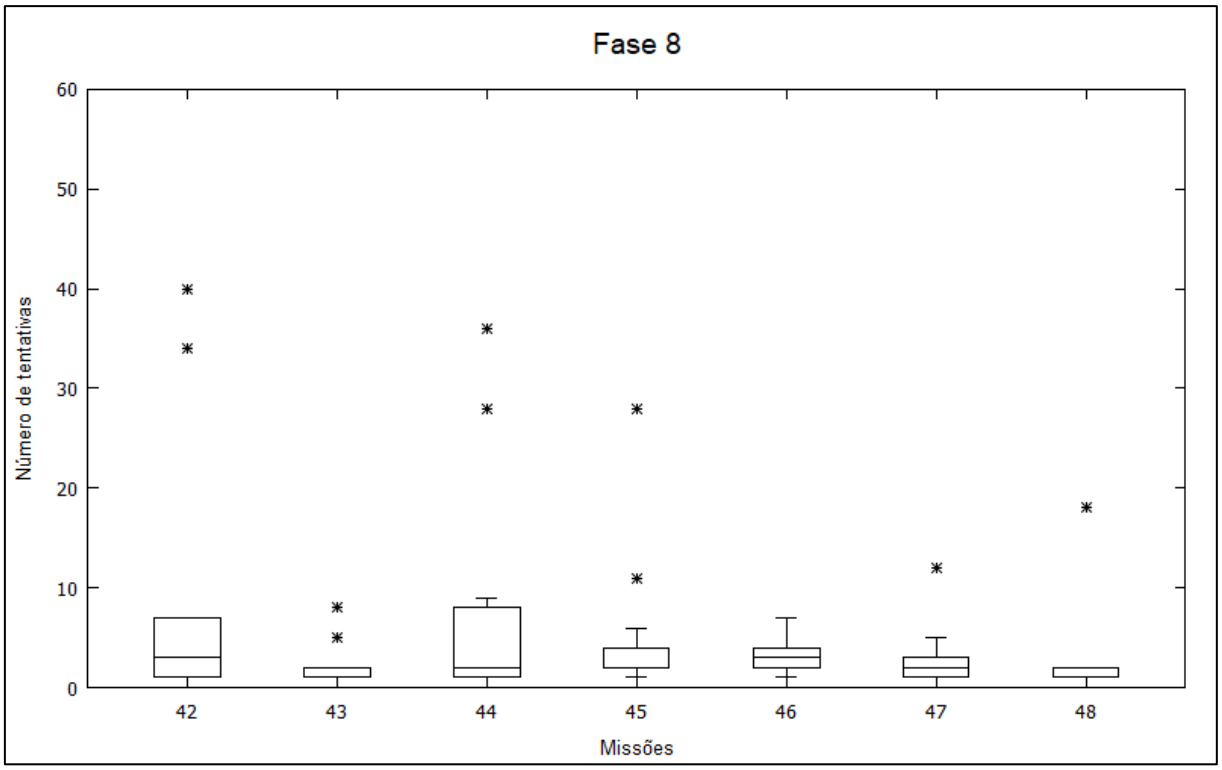


Figura 4.57 – Tentativas na Fase 8 no Ciclo 3

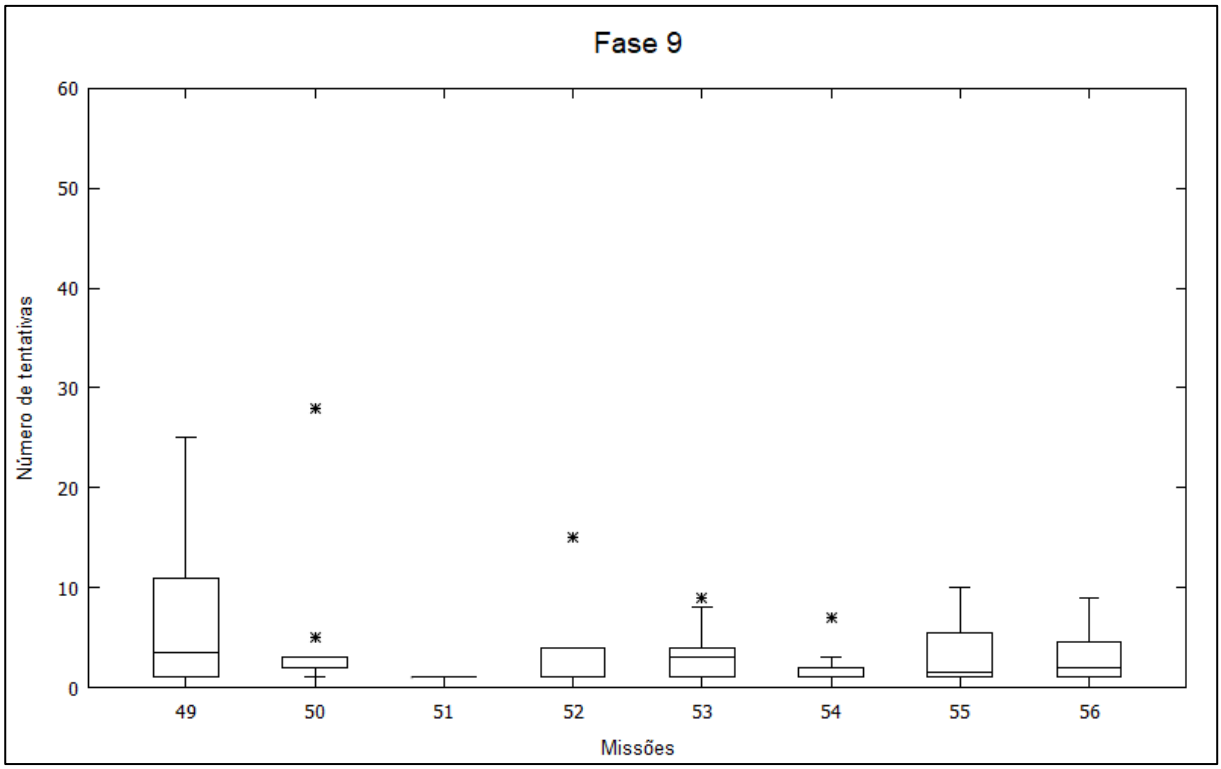


Figura 4.58 – Tentativas na Fase 9 no Ciclo 3

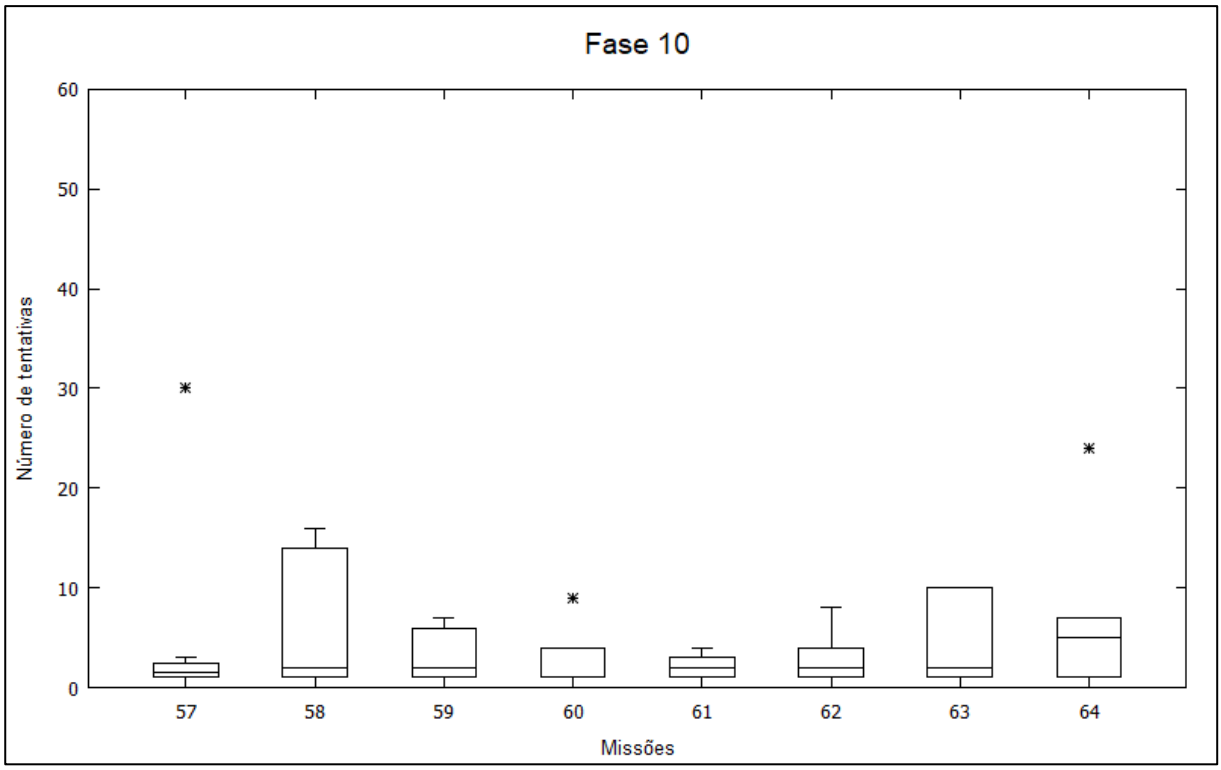


Figura 4.59 – Tentativas na Fase 10 no Ciclo 3

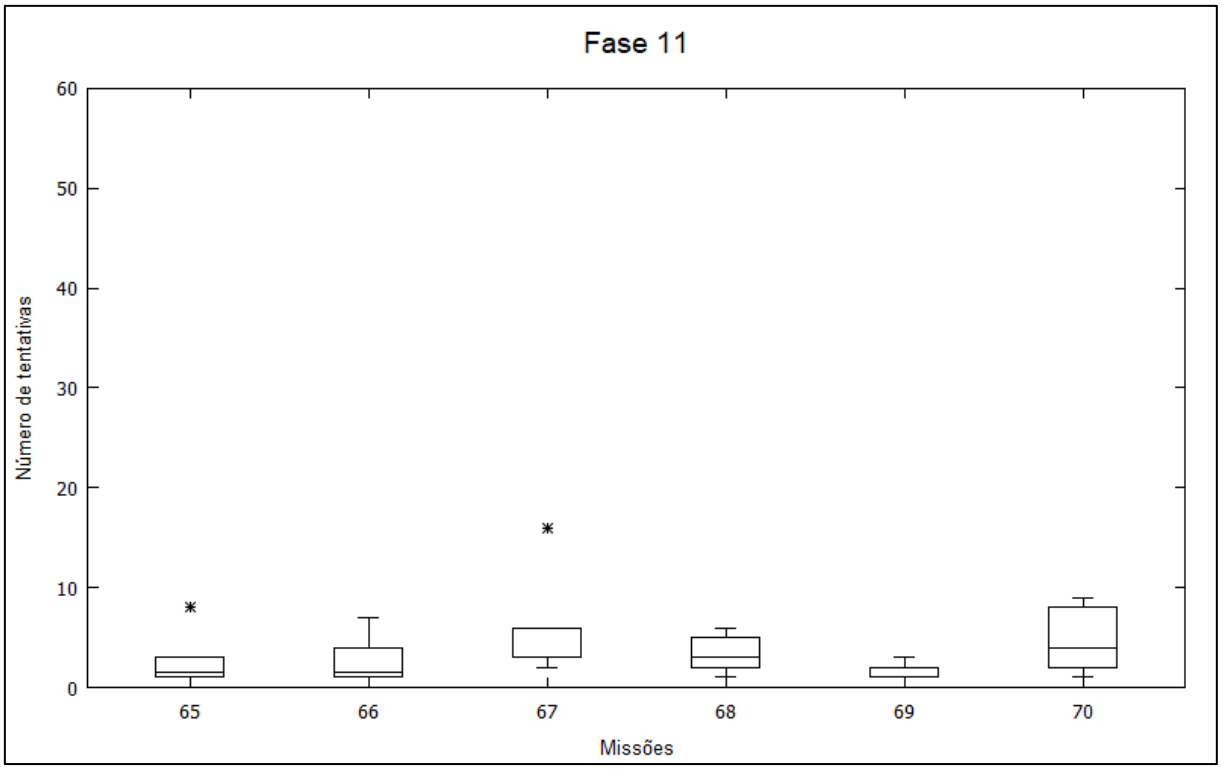


Figura 4.60 – Tentativas na Fases 11 no Ciclo 3

4.7.2.3.3 Dificuldade das Missões

Analisar a dificuldade das missões é importante para verificarmos a necessidade de revisar o projeto instrucional. Para identificar a dificuldade das missões foi verificada a quantidade de tentativas que os alunos necessitaram para concluir cada uma delas. É interessante verificar nas tentativas as discrepâncias, mostrando a insistência de alguns alunos em concluir a missão, ainda que experimentando grandes dificuldades. Por exemplo, na missão 22 um aluno fez mais de 100 tentativas. É importante frisar, que nos gráficos das Figuras 4.52 a 4.60 somente constam as tentativas das missões concluídas. Então ainda existem as tentativas daqueles alunos que não concluíram as missões.

Através dos diagramas de caixa pôde-se determinar quais as missões que colocaram mais dificuldades aos estudantes na sequência de aprendizagem, levando a um maior número de tentativas para a sua resolução. Nessa análise desconsideramos as fases 11 e 12 pela baixa quantidade de alunos (2 <= n <= 5) que concluíram essas fases. A Figura 4.61 apresenta o diagrama para a fase 2, que não foi contemplada na análise da sequência de aprendizagem da secção anterior por ser uma fase obrigatória na sequência.

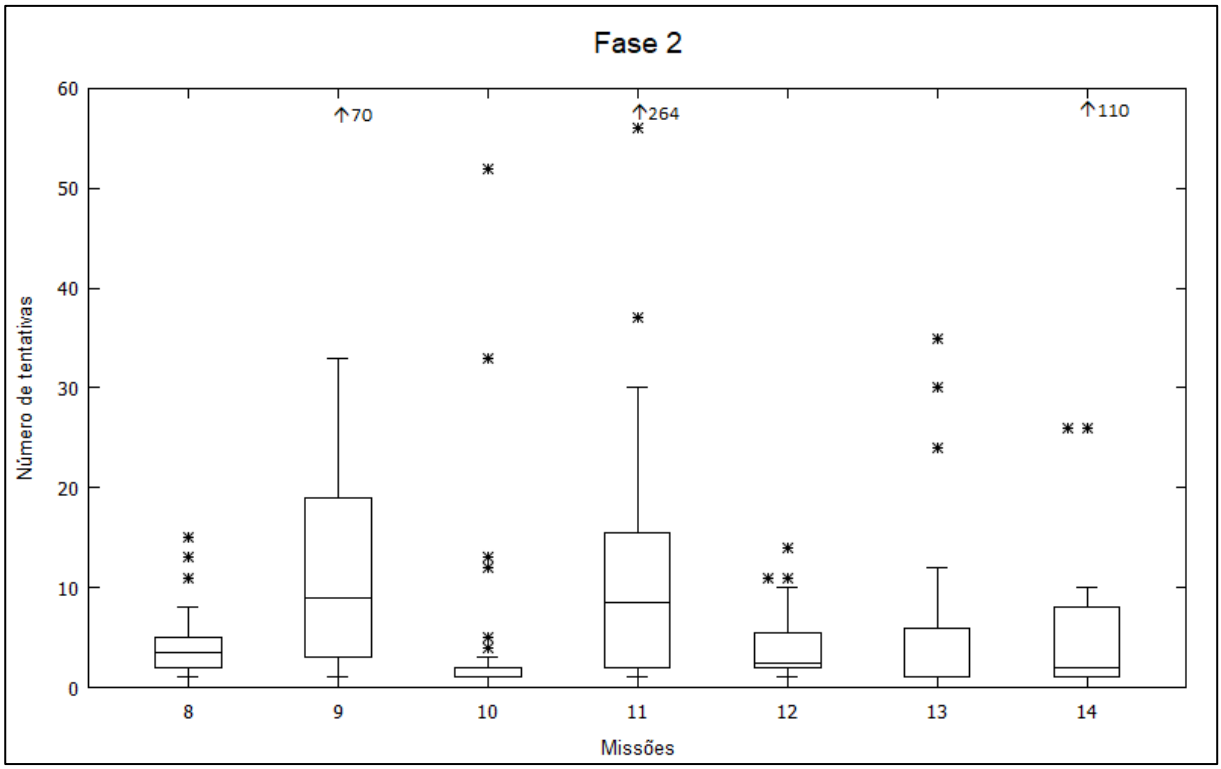


Figura 4.61 – Tentativas na Fase 2

A Tabela 4.26 apresenta somente as missões com mais de 5 tentativas na mediana entre todas as missões das fases 2 a 10. É notável que estas missões pertencem todas às fases iniciais, 2 a 4. O número de tentativas está identificado nas colunas “mediana” e “quartil maior”. A coluna “alunos” representa quantos alunos cumpriram a missão. A coluna “erros” inclui o número de tentativas cuja execução foi interrompida pelo jogo, por exemplo, por ser solicitado algo a um cliente, sem estar junto dele, ou somar um valor a uma variável sem a ter inicializado. Nesses casos a execução era interrompida. Os “objetivos não cumpridos” são as tentativas em que a execução foi encerrada sem erro, mas em que um ou mais objetivos da missão ficaram pendentes. Por exemplo, um dos objetivos poderia ser “falar[16] a comida total desejada pelos clientes”, e na solução pode não ter havido um comando de fala, ou ter falado a quantidade incorreta. A coluna “total” é a soma das duas colunas anteriores. A coluna “total das execuções” representa a soma das execuções com erro, objetivos não cumpridos, interrompidos pelo próprio aluno ao clicar no botão parar e a execução de sucesso que encerra a missão.

[16] Devido um dos comandos da personagem chamar-se “Talk”, será usado o termo “falar” e não “indicar” ou “escrever” quando comentarmos sobre os exemplos de soluções.

Tabela 4.26 – Missões com maiores medianas na quantidade de tentativas no Ciclo 3

Fase	Missão	Mediana	Quartil maior	Alunos	Erros	Objetivos não cumpridos	Total	Total das Execuções
2	9	9	19	33	29	247	276	474
2	11	8,5	15,25	32	347	185	532	615
3	19	9	13	28	11	229	240	290
4	22	13	24,5	27	194	303	497	576

Na missão 9, 22 estavam relacionados com o uso de uma expressão sintaticamente incorreta. Isso acontece quando blocos exigem parâmetros e estes não foram indicados. Especificamente nessa missão, os alunos tentaram criar expressões de soma sem adicionarem as variáveis a serem consideradas no cálculo. Existiam dois objetivos na missão: falar a quantidade de itens pedidos pelo cliente e não o fazer mais do que 2 vezes. O primeiro objetivo não foi cumprido 236 vezes. Notadamente o problema dos alunos era não saber como se fazia uma soma de três variáveis.

Na missão 11, dos 347 erros na execução, 147 foram de um mesmo aluno. Ainda assim a quantidade de erros foi muito alta. Analisando os erros, constatou-se que o mais ocorrido (149 vezes) foi causado por uma noção errada quanto ao funcionamento das variáveis, pois os alunos acreditavam que quando se coloca um valor numa variável, o valor anterior ficaria na mesma disponível de alguma forma e poderia ser acedido. Quanto aos objetivos não cumpridos, o que mais ocorreu (119 vezes) foi não falar a quantidade de bebida desejada justamente pelo erro de noção das variáveis.

Na missão 19 encontrámos uma inconsistência no enunciado durante o ciclo, pelo que decidimos não considerar os erros encontrados. A missão 22 envolvia conceitos básicos de matemática, onde a personagem tinha que devolver o troco ao cliente em moedas de 2. O erro mais comum foi o cálculo errado do troco, o qual foi encontrado 102 vezes. Como os clientes são honestos, a execução da solução é interrompida, e por isso é considerada “erro” e não “objetivo não cumprido”. Quando conseguiam entregar o troco corretamente, 295 vezes não falaram o valor correto da gorjeta, que é a diferença entre o troco a ser devolvido e aquele entregue com moedas de 2, ou seja, a gorjeta só podia ter o valor 1 ou 0. Pode-se notar falha na aplicação de conhecimentos de matemática em problemas quotidianos.

Com base nesses dados, pudemos concluir a necessidade de reforçar com alguma missão anterior à 9 para exemplificar a soma de três variáveis, outra anterior

à missão 11 para deixar claro que a colocação de um valor numa variável implica a perda do anterior, e para a missão 22 mais missões envolvendo cálculos básicos.

4.7.2.3.4 Uso do Assistente com Pseudocódigo

Analisar o quanto o assistente foi usado permite-nos identificar a quantidade de alunos que considerou que poderia ser de utilidade. Além disso, a diferença entre o momento em que o assistente foi usado e o momento da conclusão da missão podia fornecer indícios sobre o conhecimento do aluno naquela altura, pois, se mesmo com essa ajuda ele demorava a concluir a missão provavelmente ele ainda não tinha assimilado bem o conhecimento sobre o assunto em causa.

Esse assistente mostra a solução da missão em pseudocódigo (Figura 4.46). É um recurso que consome todas as estrelas, porém permite ao aluno avançar ao montar a solução interpretando o pseudocódigo fornecido. A Figura 4.62 ilustra a relação entre a quantidade de missões concluídas em que foi usado o assistente e daquelas em que ele não foi usado. As maiores taxas de utilização aconteceram nas missões 19 e 30. Como já mencionado, foi identificada uma inconsistência no enunciado da missão 19 que provavelmente causou essa quantidade de tentativas. Analisando o uso do recurso na missão 30, verifica-se que foi usado por seis alunos, sendo que quatro deles usaram o recurso antes da primeira execução, ou seja, antes de tentarem resolver a missão. Isto pode representar uma falta total de compreensão do enunciado, ou a necessidade do aluno cumprir a missão para seguir adiante e ganhar os pontos extra na prova. Mesmo com essa ajuda, estes quatro alunos ainda gastaram algumas tentativas para concluir a missão: dois fizeram 7 tentativas e os outros dois alunos necessitaram de 11 tentativas.

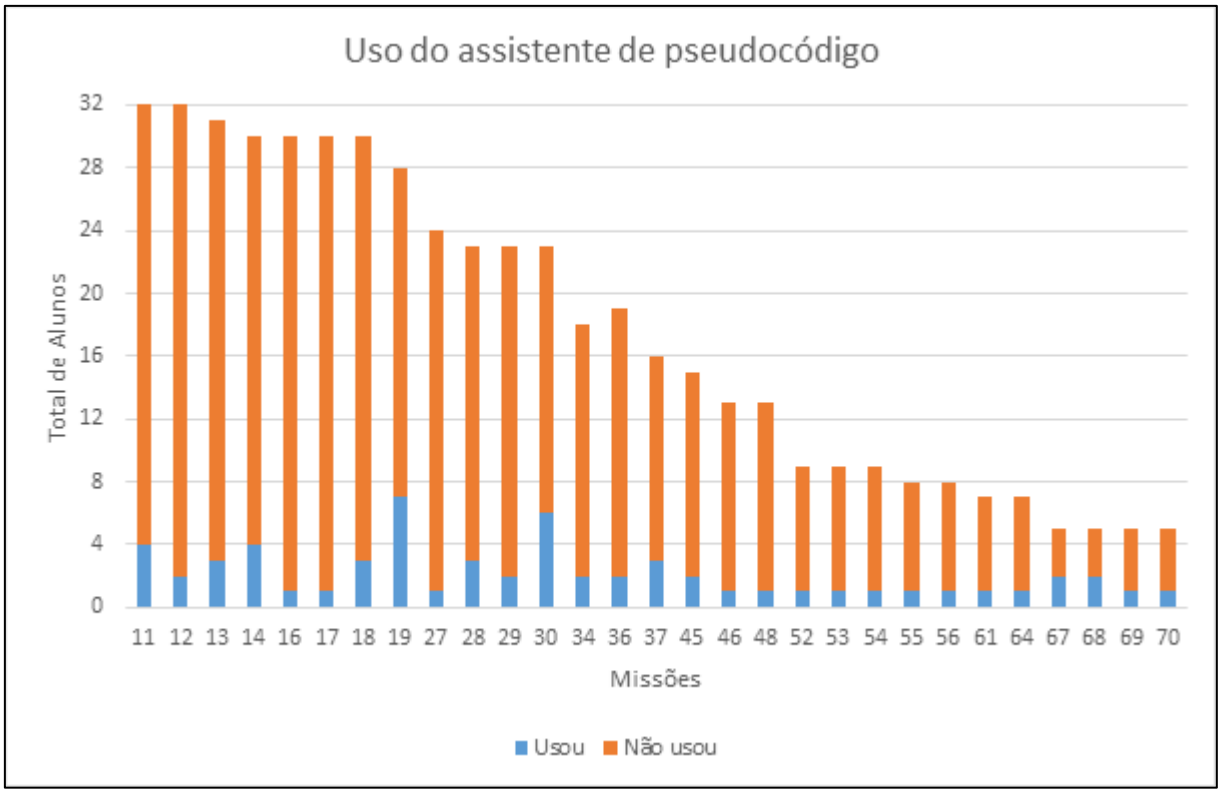


Figura 4.62 – Uso do assistente de pseudocódigo no Ciclo 3

O assistente foi usado 63 vezes nas missões concluídas por todos os alunos. A média entre a tentativa que o aluno disparou o recurso do assistente e a tentativa em que finalizou a missão foi de 6,7 (± 5,8) tentativas demonstrando que existe uma disparidade muito grande nessa amostra. A mediana foi de 3 tentativas. Isso não nos permite afirmar que o assistente tenha auxiliado o aluno na solução imediatamente. Porém, acreditamos que sem essa ajuda, esses alunos teriam desistido de jogar.

4.7.2.3.5 Aprendizagem e Diversão Percecionados

A aprendizagem percecionada é um conjunto de crenças e sentimentos em relação a aprendizagem ocorrida (Caspi & Blau, 2008) e reflete o senso do estudante que algum novo conhecimento foi adquirido e alguma nova compreensão foi alcançada, mesmo que esse conhecimento e compreensão subjetivos estiverem em contraste com o desempenho acadêmico (Caspi & Blau, 2011). A aprendizagem percecionada representa o grau de confiança que o aluno tem em relação ao seu domínio de um dado conhecimento.

Para verificar o sentimento dos alunos quanto à sua aprendizagem foi usado um inquérito (Apêndice G), aplicado após a 15ª missão, tendo sido recolhidas 28 respostas. O inquérito tinha 10 questões com cinco níveis de concordância na escala

Likert (1-não concordo totalmente e 5-concordo totalmente). A Figura 4.63 apresenta os resultados. Vamos considerar as escalas menores ou igual a 3 como aquelas em que os alunos estão insatisfeitos e acima disso estão satisfeitos. Os itens Q1-Q4 questionavam sobre o conteúdo aprendido: raciocínio lógico (Q1), variáveis (Q2), condicionais (Q3), e decomposição (Q4), tendo sido obtidas respostas que indicam satisfação de mais de 80% dos alunos. As questões Q5, Q6 e Q8 relacionavam a aprendizagem do jogo com os assuntos abordados nas aulas, a questão Q7 dizia respeito à depuração e a Q9 era sobre a sensação de ter evoluído de forma geral com o jogo. Todos esses itens também tiveram satisfação próxima ou acima de 80%. A dificuldade das missões não foi considerada compatível com o conhecimento (Q10) por cerca de 40% dos alunos. Isso significa que, apesar de terem um alto grau de confiança na sua aprendizagem, vários dos alunos acharam o jogo difícil.

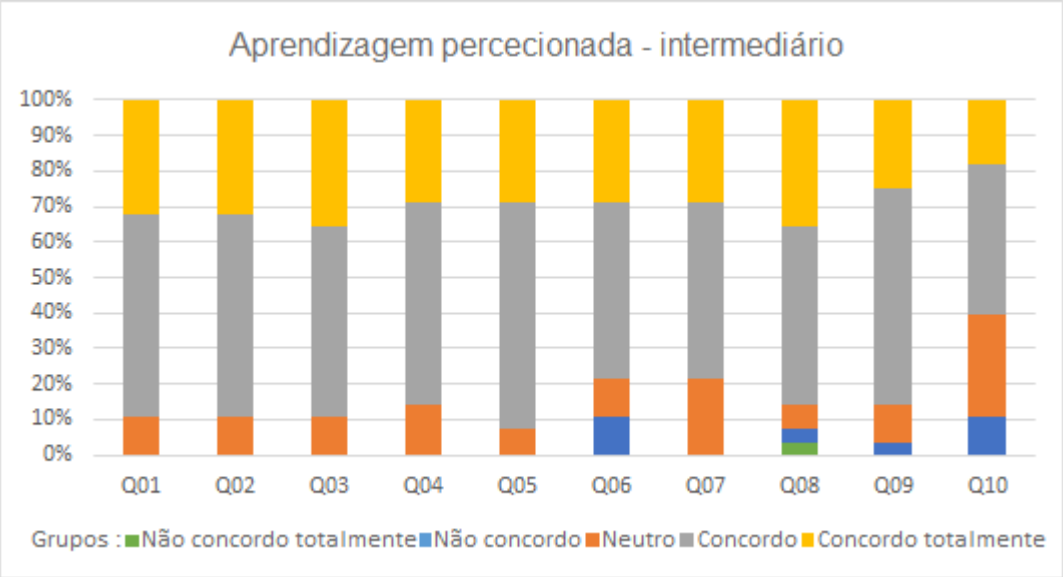


Figura 4.63 – Avaliação da aprendizagem percecionada no meio do ciclo 3

Junto do inquérito para avaliar a aprendizagem percecionada também foi aplicado outro (Apêndice H) para verificar o sentimento de diversão até aquele momento. O inquérito tinha 12 questões com cinco níveis de concordância na escala Likert e uma questão de resposta aberta. A Figura 4.64 apresenta os resultados obtidos. Quase 30% dos alunos não concordaram com o contexto do jogo ocorrer numa esplanada (Q11). O item Q12 pergunta se a qualidade gráfica os desmotivou a jogar. Mais de 80% dos alunos concordou com essa afirmação, o que foi uma surpresa, pois tínhamos melhorado a parte gráfica do jogo. Entretanto, como em torno de 67% dos alunos afirmou jogar pelo menos uma vez por semana, acreditamos que

eles tenham feito essa comparação com base em jogos comerciais com gráficos sofisticados. Outra hipótese seria a própria interpretação errônea da pergunta, uma vez que foi feita pela negativa (“Eu fiquei desmotivado com a qualidade gráfica do jogo”) enquanto as restantes foram colocadas pela positiva. Algumas questões obtiveram mais de 30% de respostas indicando algum nível de insatisfação: a influência da música sobre seu ritmo de trabalho (Q13), o suporte do jogo foi suficiente para resolver as missões (Q15), e a dificuldade entre as missões era adequada (Q16). A partir dessas respostas podemos observar que muitos alunos tiveram um sentimento de dificuldade para resolver as missões. As questões 17 a 19 visavam identificar o que mais lhes motivava no jogo. O item mais bem avaliado foi estar nas melhores posições na tabela de classificação (Q18), seguido de abrir seu próprio negócio (Q19) e por último personalizar o seu avatar (Q17).

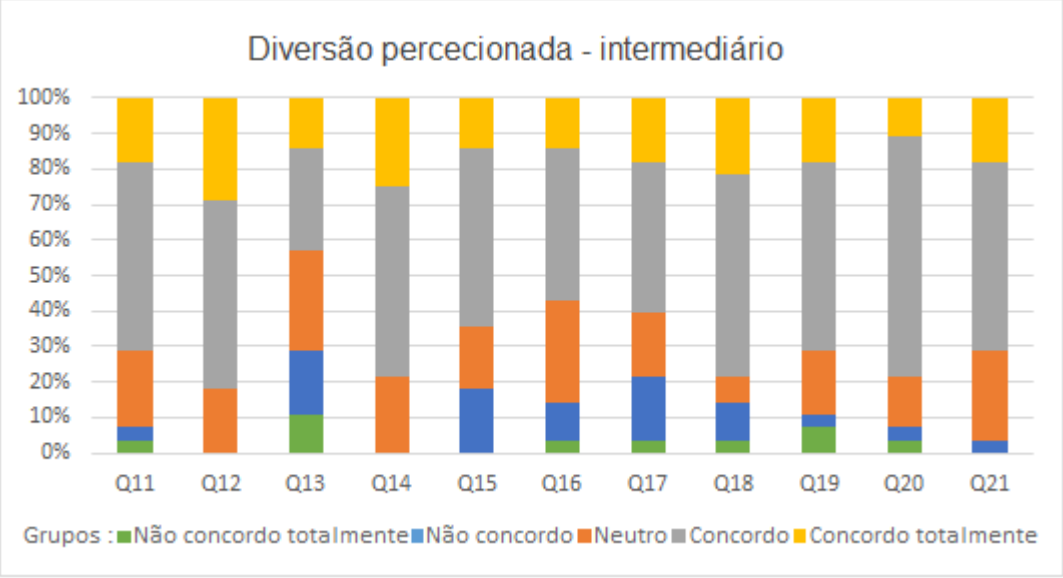


Figura 4.64 – Avaliação da diversão percecionada no meio do ciclo 3

A questão 21 pedia para os alunos darem sugestões para melhorar o jogo. Foram dadas treze sugestões: sete delas para melhorar o suporte dado pelo jogo com mais dicas e explicações e apresentar uma solução possível válida após a conclusão da missão; três sugestões para mais opções de configuração para o avatar; duas para rever o nível de dificuldade entre as missões; e uma sugestão para aumentar a quantidade de tentativas por estrela. A partir dessas sugestões podemos determinar que os alunos estavam a sentir dificuldades no jogo, pois 10 das 13 sugestões foram no sentido de facilitar ou ajudar na solução das missões, o que corrobora a Q10 do

inquérito da aprendizagem percecionada onde 40% dos alunos afirmou que o seu conhecimento de programação não era compatível com a dificuldade no jogo.

No intuito de averiguarmos em que dimensão a evolução da versão atual do jogo representava uma melhoria quando comparada com a versão anterior, executamos testes de amostras independentes Mann-Whitney, entre as respostas ao inquérito neste ciclo, com as coletadas no final do ciclo anterior. Enquanto no 3º ciclo todos os alunos tinham a mesma quantidade de missões concluídas (n=15) quando responderam ao inquérito, as respostas do ciclo anterior foram obtidas no final, com diferenças de experiência de jogo entre os alunos. O item Q3 foi eliminado da comparação, pois ele verificava a aprendizagem de condicionais, que no ciclo anterior aconteceu somente com dois alunos. A Tabela 4.27 apresenta os resultados do teste. As linhas em fundo acinzentado representam as questões que obtiveram um nível de significância menor que 0,05, rejeitando a hipótese nula de igualdade entre as amostras, e nesses casos a última coluna apresenta a amostra com maior média. Novamente a questão 12 nos surpreendeu, demonstrando que no ciclo anterior houve maior satisfação com a qualidade gráfica do jogo do que nesse ciclo. Nas demais questões, sempre o ciclo atual teve os melhores resultados. Os alunos encontraram mais evidências de relação entre o jogo e a disciplina (08), acharam o contexto da esplanada mais atraente (11), que a música ditou mais o ritmo de trabalho deles (13), e se sentiam mais motivados em estar mais bem posicionados na tabela de classificação (18) e abrir seu próprio negócio (19) do que os alunos do ciclo anterior.

Tabela 4.27 – Comparação da diversão e aprendizagem entre os ciclos 2 e 3

Questão	*p*	Ciclo 2 (n)	Amostra que excedeu
01	0,942	12	-
02	0,896	12	-
04	0,738	12	-
05	0,146	12	-
06	0,850	12	-
07	0,247	12	-
08	0,001	44	3
09	0,569	10	-
10	0,738	44	-
11	0,004	48	3
12	0,000	48	2
13	0,032	44	3
14	0,522	42	-
15	0,222	42	-
16	0,287	42	-
17	0,117	44	-
18	0,034	44	3
19	0,005	44	3
20	0,375	42	-
21	0,598	42	-

Aplicamos o instrumento EGameFlow após a 20ª missão. Criamos uma versão reduzida apenas com as 10 questões mais relevantes para aquele momento e mais duas questões abertas sobre o que MAIS e o que MENOS agradaram no jogo (Apêndice I). Calculamos a média dessas questões e executamos o teste de Mann-Whitney em relação ao ciclo anterior. Naquele ciclo foram 22 respostas (n=22) e neste foram 29 respostas (n=29). A Tabela 4.28 apresenta as médias de ambos os ciclos nas 6 dimensões limitadas entre as 10 questões. Foram encontradas diferenças estatisticamente significativas ($p<0,05$) na dimensão suporte e na média total, podendo indicar a necessidade de melhorias para o próximo ciclo. Para um melhor entendimento, realizamos uma análise individual das questões (Tabela 4.29). A única questão com diferença significativa é "F2-Eu recebi suporte imediato sobre minhas ações". Como já foi constatado no inquérito anterior (na 15ª missão), os alunos sentiram a necessidade de maior suporte durante o jogo. Ainda, em relação ao suporte nesse momento da avaliação, na questão "F1-Eu recebi suporte sobre o meu progresso durante o jogo", apesar de não haver uma diferença estatisticamente significativa, a sua média foi inferior se comparada ao ciclo anterior. Somando a isso, o que mais foi mencionado na questão aberta sobre o que menos agradou diz respeito ao suporte. Todas essas respostas podem nos indicar a necessidade de investimento para melhorar o suporte.

Tabela 4.28 – Comparação do EGameFlow entre os ciclos 2 e 3

Ciclo	Concentração	Clareza dos objetivos	Suporte	Desafio	Autonomia	Imersão	Total
2	4,05 (±0,849)	4,23 (±0,685)	3,84 (±0,834)	3,68 (±1,129)	3,55 (±1,057)	3,45 (±1,066)	3,82 (±0,957)
3	3,75 (±1,070)	3,83 (±0,759)	3,10 (±1,307)	3,48 (±1,090)	3,62 (±1,049)	3,62 (±1,226)	3,56 (±1,146)
p	0,143	0,063	0,005*	0,504	0,718	0,344	0,031*

Tabela 4.29 – Comparação dos itens do EGameFlow entre os ciclos 2 e 3

Questão	Ciclo 2	Ciclo 3	*p*
C3	4,41 (±0,666)	4,00 (±1,069)	0,176
C7	3,73 (±0,883)	3,41 (±1,119)	0,443
C8	4,00 (±0,873)	3,83 (±0,966)	0,608
G1	4,23 (±0,685)	3,83 (±0,759)	0,063
F1	3,82 (±0,907)	3,17 (±1,284)	0,078
F2	3,86 (±0,774)	3,03 (±1,349)	0,028*
H3	3,68 (±1,129)	3,48 (±1,090)	0,504
A7	3,55 (±1,057)	3,62 (±1,049)	0,718
I1	3,64 (±0,953)	3,41 (±1,162)	0,754
I7	3,27 (±1,162)	3,83 (±0,966)	0,081

Em relação ao que mais agradou os alunos, a maioria mencionou sobre o desenvolvimento do raciocínio lógico e a aprendizagem de programação usando jogos. Novamente, podemos verificar que a abordagem de jogos para o aprendizado é uma novidade bem-vinda para os alunos, como uma forma de praticarem o desenvolvimento de resolução problemas, que é uma das habilidades elementares que se espera adquirir uma disciplina introdutória de programação.

4.7.2.3.6 Inquérito Final

Após o final da experiência foi aplicado o último inquérito. O inquérito completo contém 39 questões (Apêndice J) originárias dos inquéritos de aprendizagem e diversão percecionados, EGameFlow reduzido e algumas questões novas. Os alunos foram agrupados utilizando os mesmos procedimentos do ciclo anterior, existindo um conjunto de questões para cada grupo. A diferença entre as questões está basicamente nas perguntas sobre o conteúdo aprendido, por exemplo, “Eu aprendi com o jogo a usar ciclos” ou “Eu aprendi com o jogo a usar condicionais”. Além de questões com cinco níveis de concordância na escala Likert, também existiam questões de resposta aberta. A Tabela 4.30 lista as características de cada grupo: o tamanho da amostra e quantos desses responderam, os assuntos que praticaram no jogo conforme as missões jogadas, e a quantidade de questões para cada grupo. O Apêndice J apresenta a relação dessas questões para cada grupo.

Tabela 4.30 – Grupos no inquérito final do Ciclo 3

Grupo	Amostra	Respondeu	Assuntos jogados	Quantidade de questões
1	9	8	Fase 9 completa	37
2	6	6	Condicionais completo	38
3	8	5	Introdução dos condicionais completo	36
4	7	2	Iniciando nos condicionais	34
5	6	2	Jogavam variáveis	29
6	8	4	Não entrou no jogo	2
Totais	**44**	**27**		

Para comparar a avaliação entre os grupos, executámos o teste não-paramétrico de amostras independentes de Kruskal Wallis. Ainda, calculámos a média dos itens para identificar aqueles que exigirão mais atenção para o próximo ciclo. A Tabela 4.31 apresenta o resultado desses cálculos. Três itens tiveram diferenças estatisticamente significativas ($p<0,05$) entre os grupos. O G3 apresentou menor sensação de controlo (Q15) entre os cinco grupos, e G1 a maior sensação. O G3 foi aquele que sentiu maior sensação de passagem do tempo durante o jogo (Q19) e G5 foi o que menos teve essa sensação de imersão. G1 estava mais preocupado em ganhar pontos para personalizar o avatar (Q23) e o G3 era o menos preocupado com isso. Os demais itens não tiveram diferenças estatisticamente significativas entre os grupos. Ao avaliarmos os itens com médias menores que três, pudemos concluir que os alunos demonstraram não estarem entusiasmados para jogarem (Q18), que a música não ditava o ritmo de trabalho (Q22) e que não tinham tanto interesse em relação à personalização do avatar (Q23). A questão 21 referia-se à desmotivação que a qualidade gráfica poderia interferir. Ao contrário do inquérito anterior, onde a maioria havia respondido que concordavam que estavam desmotivados, nesse inquérito final a média de 2,65 indica que estão satisfeitos com a qualidade gráfica do jogo. Quatro itens estavam acima da média 4,00. Os alunos concordaram que a maior parte do tempo gasto no jogo era concentrado com tarefas de aprendizagem (Q01), que entenderam o que os blocos faziam (Q16), que aperfeiçoaram seu conhecimento quanto ao raciocínio lógico (Q25), aprenderam a usar condicionais (Q27) e perceberam grande relação do jogo com os assuntos na disciplina (Q33). Vale destacar que a questão 39 perguntava aos alunos se eles recomendavam que o jogo fosse usado no semestre seguinte, o que resultou na média 3,87.

Tabela 4.31 – Análise do inquérito final no Ciclo 3

Questão	Média	*p*	Menor	Maior
01	4,04 (±0,825)	0,052		
02	3,04 (±1,186)	0,665		
03	3,43 (±1,037)	0,620		
04	3,61 (±1,033)	0,428		
05	3,65 (±0,935)	0,330		
06	3,74 (±1,096)	0,662		
07	3,43 (±0,896)	0,500		
08	3,09 (±1,125)	0,300		
09	3,09 (±1,164)	0,149		
10	3,35 (±0,982)	0,720		
11	3,13 (±1,014)	0,456		
12	3,70 (±0,876)	0,092		
13	3,09 (±1,203)	0,269		
14	3,35 (±1,027)	0,846		
15	3,13 (±0,968)	0,029*	G3: 2,00 (±0,707)	G1: 3,63 (±0,518)
16	4,09 (±0,733)	0,410		
17	3,17 (±1,154)	0,072		
18	2,87 (±1,254)	0,312		
19	3,57 (±1,343)	0,038*	G5: 1,50 (±0,707)	G3: 4,60 (±0,894)
20	3,30 (±1,428)	0,496		
21	2,65 (±1,229)	0,511		
22	2,61 (±1,340)	0,234		
23	2,87 (±1,456)	0,024*	G3: 1,40 (±0,548)	G1:3,88 (±1,126)
24	3,93 (±1,141)	0,135		
25	4,10 (±0,768)	0,435		
26	3,95 (±0,740)	0,224		
27	4,05 (±0,621)	0,911		
28	3,86 (±1,167)	0,070		
29	3,67 (±0,966)	0,490		
30	3,71 (±0,902)	0,528		
31	3,86 (±0,793)	0,163		
32	3,70 (±0,822)	0,706		
33	4,17 (±0,650)	0,620		
34	3,68 (±0,946)	0,109		
38	3,87 (±1,325)	0,529		

A questão 35 perguntava aos cinco grupos “Qual foi a tua maior dificuldade enquanto jogavas?”. Houve 23 respostas. A resposta mais citada foi “compreender os erros e criar a solução” com 12 ocorrências. Outras respostas referiam-se à compreensão dos enunciados, prazo para cumprir as conquistas e a incoerência com a dificuldade entre as missões.

A questão 36 perguntava a todos os grupos “Sugira algum recurso que mais te atrairia se tivesse no jogo.”. Houve 27 respostas, onde 9 delas pediram mais suporte, como apontar com mais clareza os erros, ter mais dicas no jogo e mostrar a solução das missões após concluí-las. Outras sugestões foram “chat entre jogadores e um sistema offline”, “Poder escolher o nome do personagem (…)”, “Copiar e colar blocos das fases anteriores”, e “(…) e digitarmos os códigos”.

A questão 37 perguntava aos grupos 2, 3, 4 e 5 “Qual foi o principal motivo que te fez parar de jogar?”. Houve 15 respostas, onde a resposta mais evidente (n=8) foi conciliar o tempo gasto com o jogo e com as outras obrigações das aulas.

A questão 39 era exclusiva para o grupo 6: “Qual foi o principal motivo de não jogares?”. Com 4 respostas, onde 3 responderam por falta de interesse e outro por falta de tempo.

Avaliámos as diferenças entre a aprendizagem percecionada quando estavam a meio do jogo (pelo inquérito anterior) e ao final da experiência. Ambos os questionários foram respondidos por 21 alunos. Foi executado um teste não-paramétrico Wilcoxon de amostras emparelhadas para verificar se existem diferenças entre os dois momentos. Existem duas questões com diferenças estatisticamente significativas ($p<0,05$): “Q5-Eu aprendi coisas novas com o jogo que foram úteis na disciplina” ($p=0,033$) negativa, ou seja, na primeira avaliação houve mais concordância com esse item do que na avaliação final; e “Q10-A dificuldade dos desafios estava compatível com o meu conhecimento em programação” ($p=0,038$) também negativa, demonstrando que os alunos concordaram que o jogo era demasiado difícil para o seu conhecimento.

Também avaliámos a diversão percecionada entre o meio e ao final da experiência usando o teste Wilcoxon. Existiram três questões com diferenças estatisticamente significativas ($p<0,05$) negativas: “Q12-Eu fiquei desmotivado com a qualidade gráfica do jogo” ($p=0,001$; “Q20- Eu compreendi as mensagens de erro que o jogo me mostrava” ($p=0,007$) e “Q21- Eu entendi quais eram os objetivos que não consegui cumprir que o jogo me mostrava” ($p=0,021$). Em relação a Q12 a diferença significativa demonstrava que eles estavam mais desmotivados a meio do que ao final, e podemos considerar que as mudanças na qualidade gráfica satisfizeram os alunos. Em relação as outras duas questões vão de encontro ao que já definimos anteriormente, sobre a dificuldade do jogo.

4.7.2.3.7 Resumo dos Resultados

Os objetivos desse ciclo foram:

(1) Mensurar o prazer dos alunos com o jogo

Mais alunos jogaram proporcionalmente mais missões se comparado com o ciclo anterior. Na avaliação intermediária a dimensão do suporte foi significativamente inferior à avaliação no ciclo anterior. Como os alunos jogaram mais, sentiram maior dificuldade e precisaram de suporte adicional. Isso foi confirmado na avaliação final, onde os alunos enfatizaram ainda mais que o jogo era difícil. Em termos de elementos de diversão, eles estavam muito motivados em relação a abrirem o seu próprio

negócio, ou seja, acumular pontos para desbloquear novas áreas no cenário. Essa turma não demonstrou tanto interesse em personalizar o avatar como a turma anterior. Houve apenas uma sugestão para usar codificação, o que demonstra que os alunos agora com mais tempo para raciocinarem não sentem tanto a necessidade de produtividade na montagem da solução.

(2) Identificar lacunas para melhorar a sua diversão

É evidente que os alunos acharam o jogo difícil. Várias sugestões de melhorar o suporte, as dicas e a ajuda vão de encontro a um sistema inteligente que perceba as dificuldades dos jogadores e possa assisti-los dentro do contexto em que estejam. As dicas são apresentadas quando o aluno fica inerte no jogo, e a partir de verificação simples (por exemplo, a existência ou não de certos blocos na solução) é apresentada uma mensagem sugerindo que o aluno examine seu código para a possibilidade de fazer a alteração, ou inserção, de blocos específicos conforme o resultado da verificação simples. Como o enfoque dessa investigação não é aplicação de sistemas inteligentes, devemos fazer para o próximo ciclo a revisão no projeto instrucional para introduzir missões que possam responder melhor às dificuldades dos alunos. Em relação aos aspetos de elementos de jogo para motivarem a diversão, os resultados dos inquéritos demonstraram que não precisamos mais investir nessa direção.

(3) Verificar se os desafios na sequência das missões oferecem o sentimento de fluxo

Existe uma alternância entre missões fáceis e difíceis criando essa variação entre os momentos de sentimento de controlo e tensão. Além disso, com a proposta de sequenciamento das fases, permitiu-se que os alunos continuassem a jogar mesmo não conseguindo concluir uma determinada missão. Inclusive aconteceram situações em que os alunos concluíram fases mais difíceis antes das mais fáceis, por exemplo, as fases 6 e 7. As quatro missões mais difíceis estavam nas primeiras quatro fases, principalmente na segunda fase que era pré-requisito para abrir outras duas. Aproximadamente 16% dos alunos desistiram já na segunda fase. O desafio está muito alto logo no começo. O assistente de pseudocódigo manteve alguns alunos jogando, uma vez que a média entre a primeira missão que ele foi usado e a última missão concluída pelos 19 alunos que o usaram foi em torno de 30 missões, ou seja, ele não foi usado por poucos alunos várias vezes, mas em situações de necessidade.

(4) Identificar se o jogo contribuiu para a aprendizagem do aluno

Baseando-se na aprendizagem percecionada, os alunos tinham um alto grau de confiança (em torno de 80%) de que estavam aprendendo com o jogo ao meio da experiência. Essa confiança diminuiu um pouco quando avaliado ao final da experiência. Os alunos jogaram durante muitas horas se comparado com o ciclo anterior, até porque eram motivados em conquistar ponto extra para as provas. Entretanto é preciso usar um instrumento de avaliação da disciplina para comparar o desempenho do aluno no jogo e na disciplina.

A forma de integração com as aulas ainda precisa ser melhorada, visto que os alunos responderam no inquérito final que o jogo representa uma sobrecarga de trabalho, pois eles ainda precisavam resolver concomitantemente exercícios de programação. Isso pode ser comprovado pela adesão de alunos no jogo: nas primeiras semanas, onde existem menos trabalhos e exercícios em sala, a maioria dos alunos resolveu as missões.

4.8 Quarto Ciclo de Desenvolvimento

4.8.1 Modificações neste Ciclo

Este ciclo envolveu a manutenção do jogo, como nos ciclos anteriores, e o desenvolvimento de um novo artefacto (NoBugsJ) com a intenção de auxiliar o professor na transferência do conhecimento aprendido com o jogo para a linguagem de programação da disciplina, que no caso foi o Java.

4.8.1.1 Modificações no NoBug's SnackBar

Uma das conclusões do ciclo anterior foi a necessidade de rever o projeto instrucional. Por essa razão, resolvemos adicionar dois novos tipos de tarefas para preparar melhor os alunos para as tarefas mais importantes que são as de criação. A Tabela 4.32 apresenta os tipos de tarefas e a sua sequência inspirada na Taxonomia de Bloom revisada. Os dois novos tipos de tarefas são "Múltipla escolha" e "Completar

nos espaços". As missões de "Múltipla Escolha" têm o objetivo de apresentar um exemplo correto do uso de um novo assunto, e treinar o aluno na leitura e interpretação de programas. As missões de "Corrigir erros" tinham três tipos de tarefas: (1) alterar as referências e uso incorreto de nome de variáveis ou operadores, e (2) inserir novos blocos ou (3) trocá-los de posição. Agora limitamos para que corrigir erro tenha somente o primeiro tipo de tarefa. Os dois outros tipos de tarefa foram convertidos para missões de "Completar nos espaços". Porém, para direcionar o raciocínio do aluno nós criamos um novo bloco onde ele deve inserir os blocos que faltam (Figura 4.65). Também foi atribuída uma cor de fundo da área de edição para cada tipo de tarefa, que também é usada nas elipses de seleção de missões (Figura 4.66).

Tabela 4.32 – Relação entre as categorias dos domínios cognitivos da Taxonomia de Bloom Revisada e os tipos de tarefas das missões no Ciclo 4

Categorias	Cor	Tipos de Tarefas
1. Conhecer **2. Compreender**	**Azul**	**Múltipla escolha:** é fornecida uma solução e uma questão sobre ela com quatro opções de resposta. O aluno seleciona uma das opções. O jogo executa a solução e verifica a resposta do aluno.
3. Aplicar	**Vermelho**	**Corrigir erros:** é fornecida a solução com erros. O aluno precisa alterar a variável que está sendo referenciada ou trocar operadores de comparações ou lógicos.
4. Analisar	**Verde**	**Organizar blocos:** todos os blocos já estão disponíveis espalhados na área de trabalho e o aluno precisa organizá-los na sequência correta.
5. Avaliar	**Amarelo**	**Completar nos espaços:** é fornecida uma solução parcial faltando alguns blocos. O aluno completa a solução adicionando os blocos faltantes.
6. Criar	**Branco**	**Construir iniciando com sugestões:** são fornecidos alguns blocos que podem ser o início, o fim ou uma parte central da solução. O aluno cria a solução, inserindo novos blocos, para completar a parte que falta. **Construir:** o aluno cria sua solução a partir do zero. **Construir com restrições:** as tarefas têm restrições quanto a quantidade de blocos usados, quantidade de vezes que pode ser usado um determinado bloco, e/ou a quantidade de variáveis que podem ser usadas.

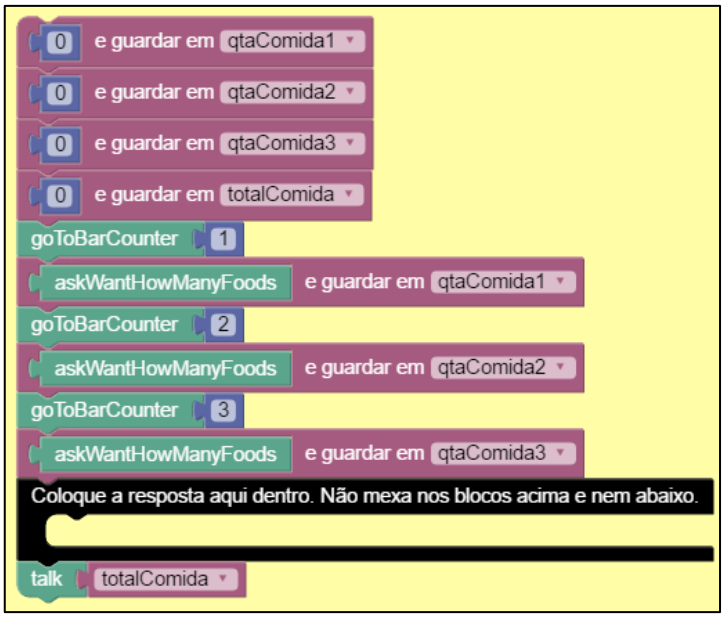


Figura 4.65 – Tipo de tarefa "Completar nos espaços"

Figura 4.66 – Seletor de missões

O assunto de *arrays* foi retirado do jogo por dois motivos:

1) O principal objetivo do jogo é preparar os alunos para resolverem problemas com programação, fazendo-os compreender o funcionamento e uso das estruturas básicas de qualquer programa. O assunto de arrays extrapola este escopo;
2) A representação dos blocos para manipulação de arrays é complexa e inautêntica, tornando o assunto mais complicado de aprender com o uso de blocos.

Com essas alterações, a estrutura da sequência das missões foi alterada (Figura 4.67). As fases continuam com seus assuntos originais, entretanto houve um acréscimo na quantidade de missões em algumas delas: duas novas na fase 1, duas na fase 2, 1 na fase 3, 4 na fase 5, e 1 na fase 8, totalizando 74 missões. A Tabela 4.33 apresenta as alterações nas fases e missões para o próximo ciclo. O tipo ME se refere à múltipla escolha e ES a completar os espaços. Durante o desenvolvimento do jogo dependíamos dos recursos de edição no ambiente, disponibilizados pelo *framework* Blockly. A partir desse ciclo existe um novo recurso no editor, que é desfazer a última ação no editor (Ctrl+Z) e refazer (Ctrl+Y).

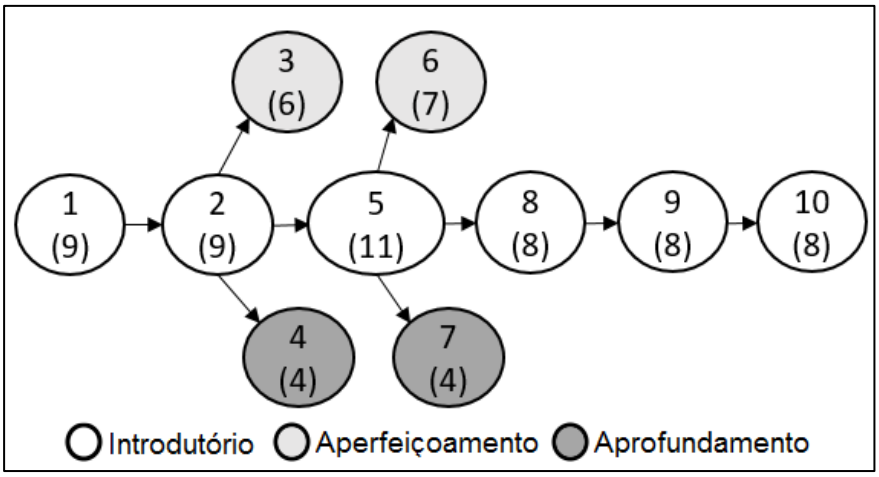


Figura 4.67 – Sequência das Fases no Ciclo 4

Tabela 4.33 – Alterações nas missões e fases no Ciclo 4

#	#	Tipo	Tarefa	Objetivos Instrucionais	Novos Blocos
Fase 1 – Afiando as facas					
1 a 3		Sem modificações			
4		Adicionadas tarefas para usar Ctrl+Z e Ctrl+Y			
5 e 6		Sem modificações			
7		ME	Responder “Quantas vezes o cozinheiro vai para a geladeira, a partir do cliente?”	Conhecer e entender as tarefas associadas ao tipo de missão Múltipla Escolha.	
8		Era a missão 7			
9		ES	Inserir dois blocos para completar o programa.	Conhecer e entender as tarefas associadas ao tipo de missão Completar Espaços.	
Fase 2 – Anotar pedidos					
1	10	ME	Responder “Que número o cozinheiro vai falar durante a execução do programa?”	Leitura de blocos matemáticos	Perguntar quanta comida deseja e Falar
2	11	Era a missão 1			
3	12	Era a missão 2, mas foi trocado o tipo de tarefa para “Completar nos espaços”			
4 a 7	13 a 16	Eram as missões 3 a 6.			
8	17	CE	Perguntar aos dois clientes quanto querem beber e usa uma só variável: na primeira pergunta guarda na variável e na segunda pergunta e soma na variável. Falar no final a soma. Corrigir o erro que na segunda pergunta vem como subtração.	Exemplo de como pode reusar a mesma variável. O aluno aprende que pode usar operadores matemáticos em conjunto com variáveis e quaisquer outros blocos de ação. A próxima missão limita a quantidade de variáveis, então	

				essa serve como exemplo.	
9	18	Era a missão 7			
Fase 5 – Aprendiz de Atendente					
1	29	ME	Responder “Que número o cozinheiro vai falar durante a execução do programa?”	Leitura de blocos condicionais	Se-então, operadores de comparação e Perguntar quanta comida deseja e Falar
2 e 3	30 e 31	Eram as missões 1 e 2			
4	32	CO1	Perguntar a três clientes se tem fome. Conte quantos clientes tem fome e no final fale essa quantidade.	Não haviam missões para o aluno construir com um condicional sem o senão.	
5	33	ME	Responder “Que número o cozinheiro vai falar durante a execução do programa?”	Leitura de blocos condicionais com senão	Se-então-senão.
6	34	Era a missão 3			
7	35	OB	Perguntar ao cliente o que deseja beber (sempre será sumo), prepará-la e entrega-la.	Uso dos blocos de manipulação da máquina de sumos sem envolver condicionais.	Movimentar até à caixa de laranjas, Pegar uma laranja, Movimentar até à máquina de sumos, Preparar e pegar um sumo, Constantes bebida e sumo de laranja
8 a 11	36 a 39	Eram as missões 4 a 7			
Fase 6 – Atendente novato					
3 e 5	42 e 44	Continuam sendo as mesmas missões, mas foram trocados os tipos de tarefa para “Completar nos espaços”			
Fase 8 – Aprendiz de Chef					
1	51	ME	Responder “Que número o cozinheiro vai falar durante a execução do programa?”	Leitura de blocos de ciclo iterativo.	Para
2 a 8	52 a 58	Eram as missões 1 a 7.			

O sistema de conquistas foi trocado para fornecer moedas como recompensa quando o aluno concluir a fase dentro do prazo. Essas moedas podiam ser gastas em melhorias no seu avatar ou para custear o uso do assistente. O aluno agora tem duas opções de custo para usar o assistente de pseudocódigo: gastar todas as estrelas da missão, como no ciclo anterior, ou pagar uma moeda.

Na parte administrativa até a versão anterior o jogo apresentava os alunos que ainda não tinham concluído uma determinada missão. Mas o professor não tinha

condições de ter a visualização nem do histórico do aluno e nem do histórico da missão. Nesta versão substituímos aquela página por um mapa que mostra todos os alunos com a conclusão de suas missões (Figura 4.68). Cada linha representa um aluno, e as colunas são as missões. As células em verde são as missões concluídas com base na quantidade de tentativas dentro da amplitude interquartil. Em vermelho são as missões concluídas com a quantidade de tentativas além do limite superior da amplitude interquartil (discrepância máxima). Em branco representam-se missões não concluídas, ainda que o aluno tenha já feito algumas tentativas. Em amarelo são indicadas as missões não concluídas, mas que apresentam já uma quantidade discrepante de tentativas. Os números representam a quantidade de tentativas. Ao clicar nesses números é apresentada a mesma janela da Figura 4.29 para que o professor possa conferir a evolução do aluno em cada uma das tentativas. Esse mapa pode ser ordenado pelo nome dos alunos, pela quantidade de tentativas discrepantes (os alunos com maior discrepância ficam em primeiro), quantidade de missões feitas (os alunos com menos missões concluídas ficam em primeiro) e tempo consumido para concluir a missão (os alunos com maior tempo ficam em primeiro). Conforme a ordenação também vai mudando os valores apresentados na célula. A ordenação privilegia sempre deixar por primeiro os alunos que supostamente precisam de mais atenção do professor.

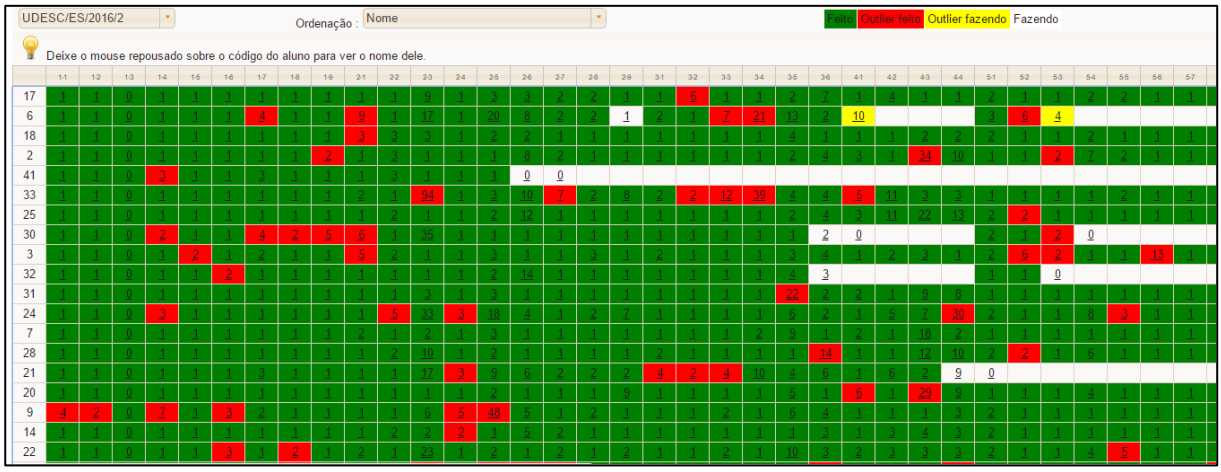


Figura 4.68 – Estado dos alunos entre as missões

4.8.1.2 NoBugsJ

NoBugsJ é um *framework* que suporta a resolução em Java das missões do jogo NoBug's SnackBar. Tal como no jogo, os alunos programam para que o atendente cumpra os pedidos dos clientes. Entretanto, nesse modo eles podem usar

qualquer Ambiente de Desenvolvimento Integrado para Java, como o Eclipse ou NetBeans, para resolver as missões usando programação em código. Em vez de jogar online, como acontece com o jogo, cada missão é armazenada em um arquivo criptografado fornecido pelo professor. Primeiro, os alunos descarregavam o *framework* (como um arquivo Jar), e depois descarregavam os arquivos com as missões. Eles resolveram cada missão através da criação de classes Java que carregavam esses arquivos e interagiam com os elementos das missões para cumprir os objetivos. Existia somente um tipo de tarefa: criação da solução. O código a seguir apresenta a estrutura inicial da classe para resolver uma missão. A classe precisa estender a classe *SnackMan* fornecida no *framework*. O método *serve* é abstrato na classe ancestral e precisa ser implementada nas classes descendentes. Os alunos programam suas soluções nesse método *serve*. No método principal (*main*) existe somente uma linha que inicia a execução da missão através do arquivo de definição (no exemplo mission5_8.dat).

```
import pt.uc.dei.nobugssnackbar.VisualWorld;
import pt.uc.dei.nobugssnackbar.SnackMan;

public class Level5Mission8 extends SnackMan {
  @Override
  public void serve() throws Exception {
     //Aqui o aluno resolve a missão
  }
  public static void main(String[] args) {
    VisualWorld.start("mission5_8.dat", Level5Mission8.class);
  }

}
```

Quando a classe é executada, o ambiente do NoBugsJ é apresentado (Figura 4.69). Na área 1 o aluno controla a interação: executa ou interrompe o atendente, reinicia (botão [Novo]) o estado dos clientes e do atendente, ou lê a descrição da missão (botão [Enunciado]. Na área 2 o aluno observa a animação do seu código. A área 3 inclui uma lista com os objetivos da missão e a área 4 apresenta a saída da consola onde todas as mensagens produzidas pelo atendente aparecem.

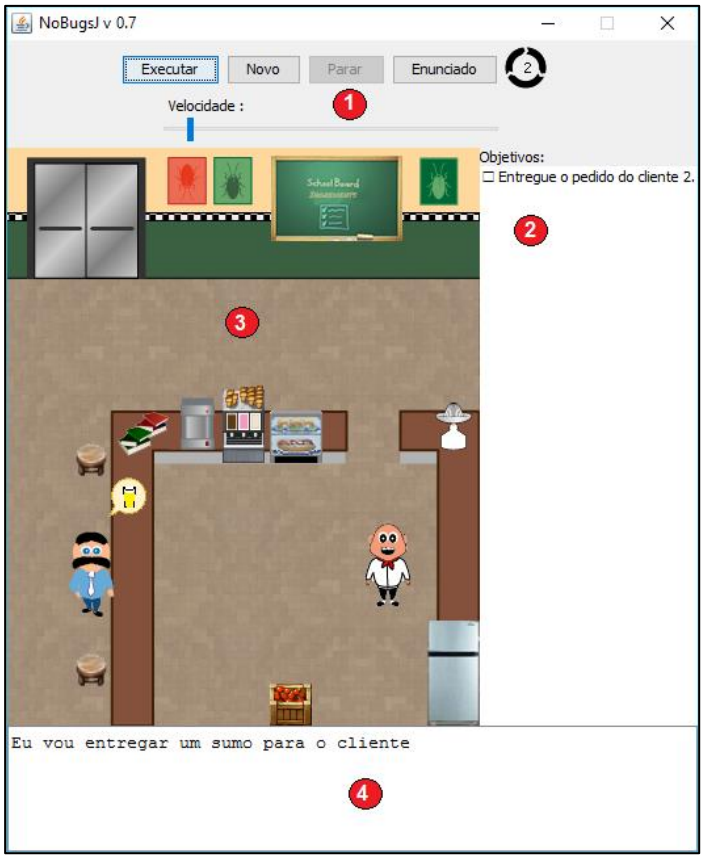


Figura 4.69 – Ambiente do NoBugsJ

A Figura 4.70 exemplifica a solução de uma missão com blocos e o seu equivalente em código Java. Todos os blocos do jogo têm um método similar no NoBugsJ.

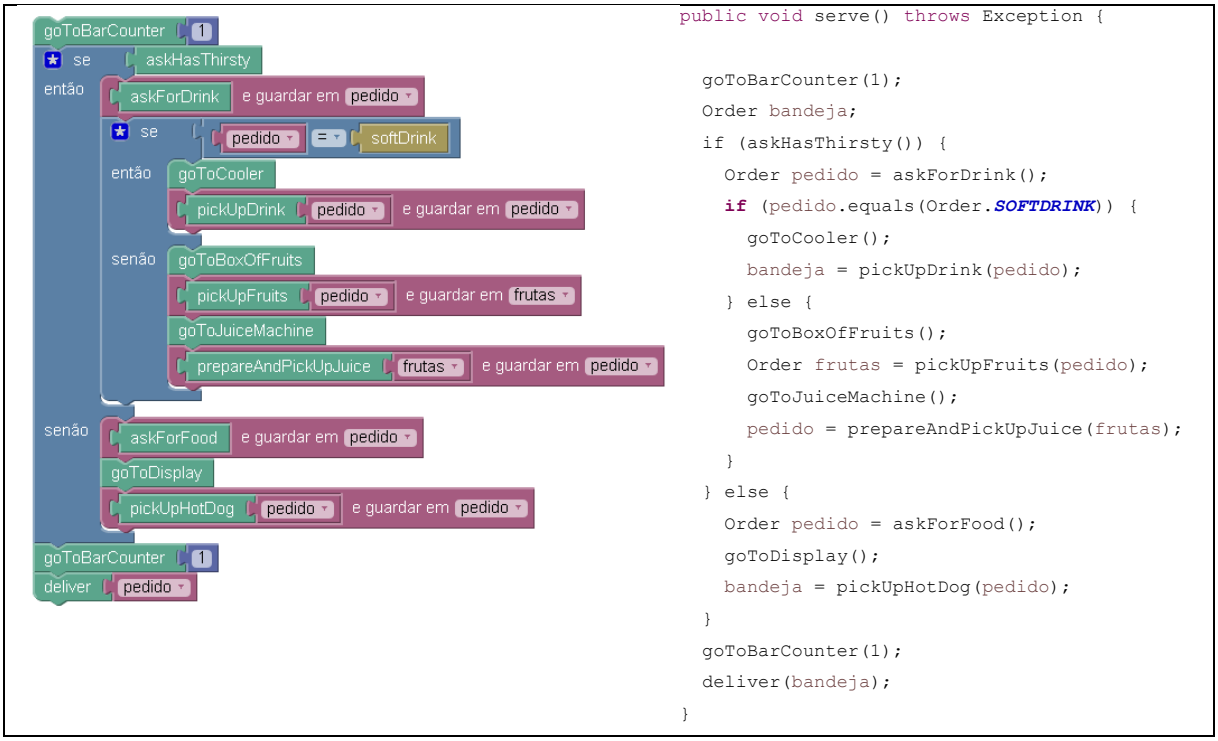


Figura 4.70 – Exemplo de solução em blocos e código

4.8.2 Avaliação do Quarto Protótipo

No ciclo anterior houve uma adesão significativamente maior se comparado ao segundo ciclo. Apesar de melhorada a integração do jogo na disciplina, muitos acabaram por desistir de jogar a partir da metade da experiência, e uma das justificativas mais citadas foi em relação à administração do tempo necessário para jogar, tendo em conta as outras obrigações da disciplina. Por essa razão, resolvemos mais uma vez modificar como o jogo é usado na disciplina. Não estamos mais interessados em conferir ou melhorar a diversão no jogo, mas em criar artefactos e estratégias que promovam a aprendizagem de programação introdutória de modo lúdico. Por essa razão, além da forma como o jogo é administrado, desenvolvemos um *framework* em Java (chamada NoBugsJ) para oferecer um suporte para a transição entre os blocos e uma linguagem de programação.

Nesse ciclo fizemos mais uma mudança, procurando verificar se a ordem pela qual os alunos jogam as missões de uma fase tem influência no seu desempenho. Obrigatoriamente todos os alunos jogam a fase 1. Uma vez concluída essa fase, todas as seguintes estarão disponíveis para jogar, com exceção da fase 10 que exige a conclusão da fase 9. Dentro de cada fase, cerca de metade dos alunos pode jogar as missões por qualquer ordem, enquanto a outra metade teve que seguir a sequência

proposta dentro da fase. A distribuição dos alunos por cada grupo levou em conta o seu estilo de aprendizagem, como se refere em seguida.

Os objetivos dessa experiência foram **(1) identificar a melhoria no desempenho na disciplina com o uso do jogo e o *framework* em Java** e **(2) identificar o comportamento dos alunos quanto ao uso dos recursos do jogo.** No primeiro objetivo pretendemos averiguar se existe alguma relação entre o desempenho dos alunos no jogo e o desempenho nas avaliações da disciplina. No segundo objetivo desejamos identificar a diferença de uso dos elementos de diversão e se essa diferença se reflete na forma de resolver os problemas.

4.8.2.1 População

A avaliação aconteceu entre 05 de Agosto e 16 de Setembro de 2016 envolvendo uma turma com um total 33 alunos (87,9% homens e 12,1% mulheres) do Bacharelado em Engenharia de Software da Universidade do Estado de Santa Catarina, no Brasil. A idade média deles era 22,5 anos (± 4,66). Em relação aos hábitos de jogar, 33,3% declarou jogar diariamente. No outro extremo, 24,2% declarou jogar raramente. Quanto ao conhecimento prévio em programação, 81,8% deles declarou não ter qualquer experiência ou contato prévio.

4.8.2.2 Metodologia e Instrumentos

Em relação a conciliar o tempo com jogo e o tempo com a aula, a primeira modificação foi na forma como o jogo foi integrado na disciplina (Figura 4.71). O professor apresentou o jogo em laboratório, onde os alunos puderam preencher o formulário demográfico (Apêndice B) e o Index of Learning Styles (Soloman & Felder, 1999), um instrumento que permite determinar o estilo de aprendizagem de um aluno, considerando o modelo de Felder & Silverman (1998) (Apendice K).

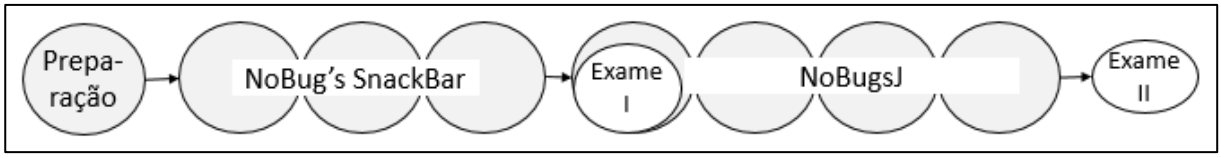


Figura 4.71 – Configuração da Experiência

Khenissi et al. (2016) investigaram a relação entre o género de jogo e a preferência de aprendizagem segundo o modelo de Felder & Silverman. Desse estudo duas hipóteses foram aceites: que os alunos do estilo sequencial preferem os jogos

de puzzle e os sensoriais de jogos casuais. Por essa razão, decidimos distribuir igualmente os alunos sequenciais e os alunos globais entre aqueles que podem escolher qualquer tipo de missão para jogar e os que devem seguir a sequência de aprendizagem proposta no jogo.

Em seguida, o jogo foi usado com exclusividade durante três semanas, sem aulas presenciais. O professor da disciplina esteve ausente, então a turma contou com o suporte do investigador por correio eletrónico e presencialmente por um monitor. Após as três semanas, foi aplicado um exame (Apêndice L) em sala, realizado em papel, com a presença do professor. O exame incluiu 5 questões: (Q1) indicar a saída de um programa dado, (Q2) responder uma questão de múltipla escolha sobre a saída de um programa dado, (Q3) corrigir um programa com dois erros, (Q4) completar o código de um programa e (Q5) criar a solução de um problema. As questões não seguiram o contexto do jogo, porém era esperado que os alunos respondessem usando a representação dos blocos. As questões foram comuns a qualquer exame de introdução à programação, como cálculo de fatorial (Q3), Índice de Massa Corporal (Q4) e divisão de números usando sucessivas subtrações (Q5). Antes da aplicação do exame os alunos foram inquiridos para identificarmos a sua perceção sobre a aprendizagem (Apêndice N) e sobre a diversão (Apêndice O).

Após o primeiro exame os alunos usaram Java e NoBugsJ no Eclipse. A didática aplicada para a transferência do conhecimento de blocos para linguagem de programação se baseou nas técnicas de "*bridging and hugging*" (Perkins & Salomon, 1988). Em *bridging*, o professor auxilia os alunos a criar generalizações de um conceito mais abstrato para uma situação mais concreta, enriquecendo o significado do conceito. Em *hugging*, o professor cria novas situações concretas para reforçar o conhecimento aprendido. Um exemplo da aplicação dessas técnicas foi usando Alice e Java (Dann et al., 2012), onde a técnica de *bridging* foi aplicada independente dos programas, com discussão em sala a partir de um exemplo dado pelo professor e os alunos sugerirem exemplos similares. A técnica de *hugging* foi resolvida através de um plugin no NetBeans que converte uma aplicação em Alice para Java.

Nesta investigação, para a parte do *bridging*, foram convertidas 39 missões do jogo para serem usadas com NoBugsJ. Cada missão foi armazenada em um arquivo criptografado e disponibilizado para os alunos descarregarem. Após o professor explicar o assunto (variáveis, condicionais, etc.) e demonstrar a relação entre os blocos e a linguagem Java, ele fornecia essas missões convertidas para que os

estudantes pudessem resolvê-las e praticar o conceito em Java. Em seguida, o professor ia para o momento *hugging* e fornecia exercícios tradicionais (por exemplo, a média de um conjunto de números dados pelo usuário, cálculo de idade, etc.) para usar o mesmo conhecimento em novas situações. Por fim, o segundo exame (Apêndice M) foi aplicado com cinco questões com a tarefa de desenvolver a solução em Java. Novamente, não foi usado qualquer comando específico do NoBugs. Antes dos alunos iniciarem o exame, eles responderam a um inquérito (Apêndice P) sobre a experiência com o jogo e o *framework* em Java.

Foi analisado o *log* de interações para identificar a frequência de entrada dos alunos no jogo e acompanhar a sua evolução, a sequência de tentativas e o uso do recurso de solução em pseudocódigo. Procurou-se também avaliar o nível de dificuldade de cada missão conforme a quantidade de tentativas dos alunos e o seu comportamento em relação aos elementos de diversão e de resolução de problemas.

4.8.2.3 Análise de Resultados

4.8.2.3.1 Estilos de Aprendizagem

Na primeira aula os alunos responderam ao inquérito ILS (Apêndice K) antes de entrar no jogo. A Figura 4.72 mostra o resultado das respostas dos 33 alunos que participaram nesse ciclo. Constata-se que a maioria dos alunos é ativa, sensorial, visual e sequencial.

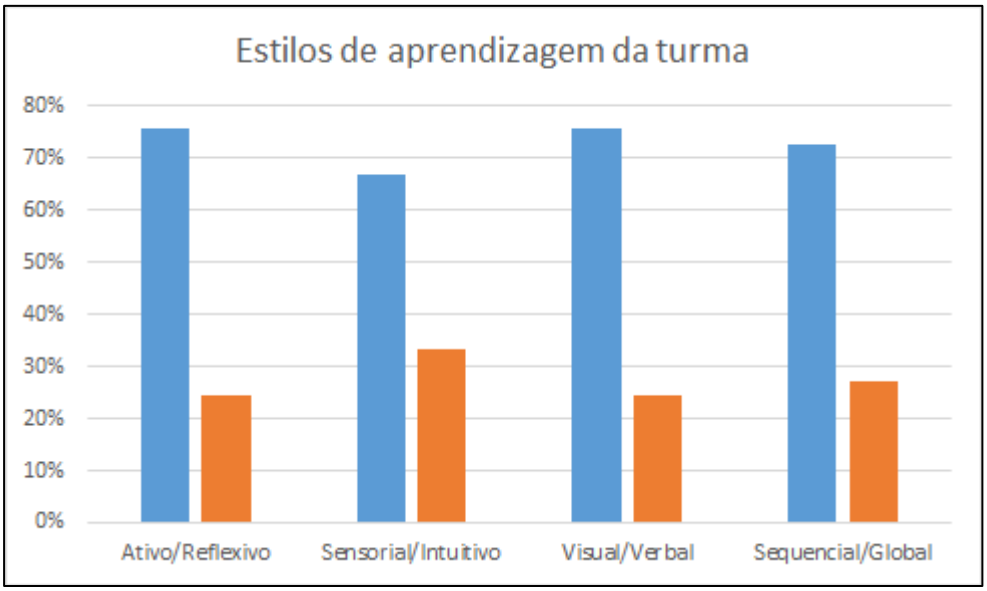


Figura 4.72 – Estilos de Aprendizagem

Para a dimensão sequencial foram classificados 24 alunos, onde 11 destes alunos tiveram a possibilidade de decidir que missões queriam jogar dentro de cada fase. Consequentemente, 9 alunos foram classificados como global e desses 5 tiveram a mesma possibilidade. A Figura 4.73 adiciona mais uma dimensão a essa distribuição: o conhecimento prévio sobre algoritmos e programação. Havia 4 respostas possíveis: sem conhecimento prévio, frequentou algumas cadeiras, fez um curso ou trabalha profissionalmente. Pode-se observar a equiparação entre a distribuição de alunos com e sem a possibilidade de selecionarem a missão que desejam jogar para os alunos sem conhecimento prévio, que são a principal preocupação dos professores.

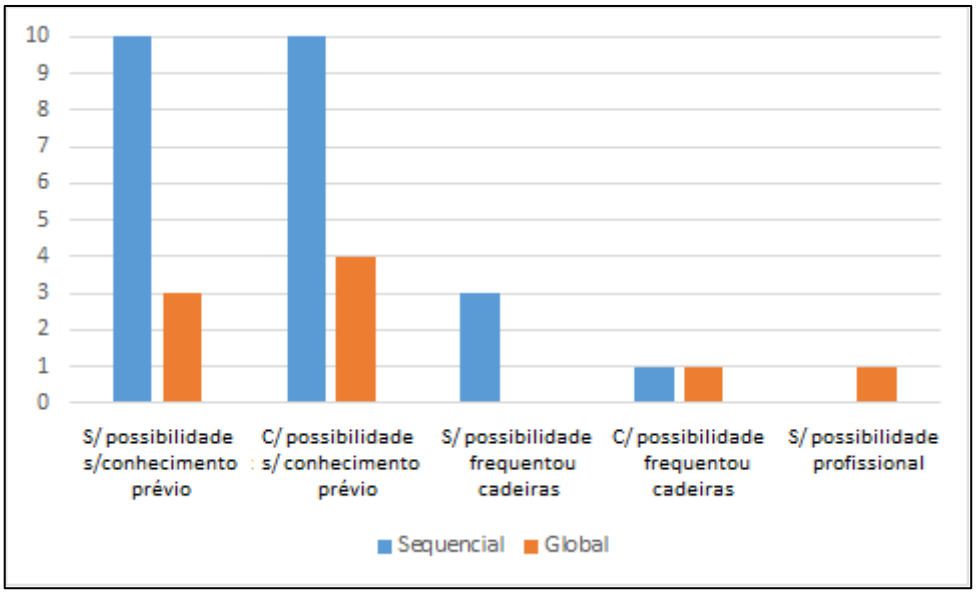


Figura 4.73 – Distribuição dos alunos sobre a dimensão sequencial/global, com ou sem possibilidade de jogar qualquer missão e seu conhecimento prévio de algoritmos e programação

4.8.2.3.2 Adesão ao Jogo

A Figura 4.74 apresenta a quantidade de alunos que concluíram cada uma das 74 missões do jogo. No ciclo anterior, no total de 73 missões, metade da turma concluiu 47,8% das missões e apenas 2 alunos (5,6%) tinham concluído todo o jogo. Nesse ciclo pelo menos metade da turma concluiu 48 missões (64,9%) e 10 alunos (31,3%) concluíram todo o jogo. Esses números demonstram uma adesão acrescida se comparada com o ciclo anterior. Também pode-se observar na figura alguns vales, como a 18ª (última missão da fase 2), entre a 25ª e 28ª (as quatro missões da fase 4), e entre 47ª e 50ª missão (as quatro missões da fase 7). Isso demonstra que alguns

alunos continuaram a jogar sem obedecer restritamente a sequência de aprendizagem proposta pelo jogo.

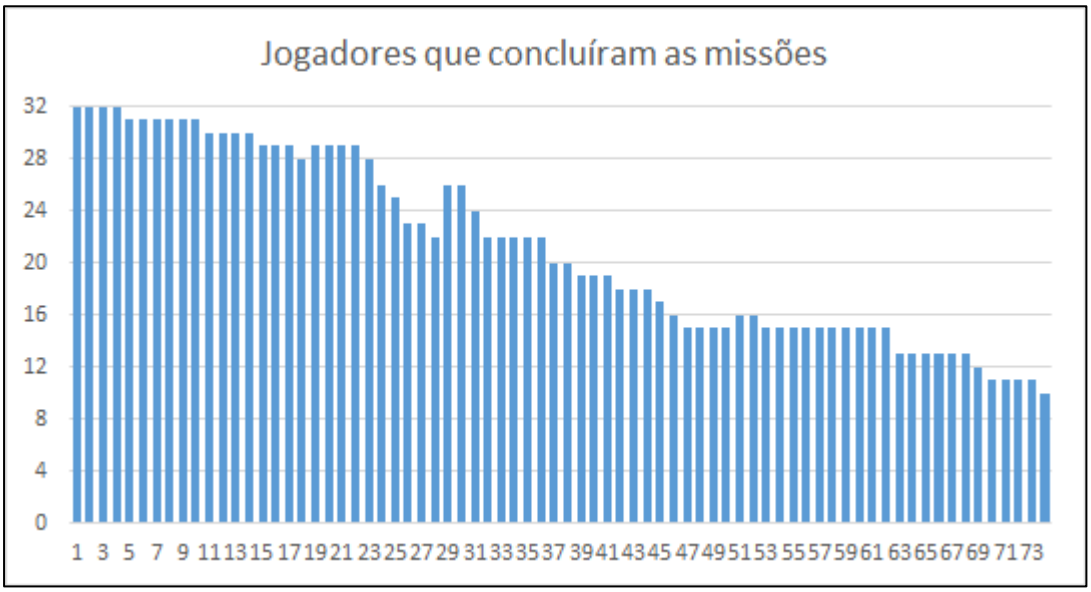


Figura 4.74 – Relação entre a quantidade de jogadores que concluiu cada missão no Ciclo 4

4.8.2.3.3 Sequência de Conclusão das Missões

A Figura 4.75 e Figura 4.76 apresentam os grafos dirigidos com a sequência de conclusão das missões dos alunos com possibilidade de escolher as missões dentro da fase e aqueles sem possibilidade, respetivamente. As missões que foram concluídas sempre em sequência foram agrupadas em um único vértice. A diferença visível entre os dois grupos é que aqueles com possibilidade seguiram o projeto instrucional, fase a fase, enquanto os restantes tenderam a resolver as missões com menos preocupação em relação a concluir fase por fase. O motivo para isso pode vir da opinião de um dos alunos que teve possibilidade de escolha: “se eu não conseguia terminar uma missão porque eu não tinha entendido, eu ia para a próxima missão da mesma fase, e muitas vezes resolvendo aquela me ajudou a entender a resolver a anterior”. Os alunos ao terem dificuldades em concluir uma missão não ficavam parados no jogo. Aqueles sem a possibilidade de escolher outra missão dentro da mesma fase, tentavam continuar jogando em outra fase. Porém, aqueles que podiam escolher a missão dentro da fase tentavam jogar logo a missão seguinte, o que era benéfico, pois a missão podia trazer alguma nova informação que fosse necessária na missão anterior, e assim mantinham a evolução dentro do mesmo tema.

Podemos constatar que os alunos com mais liberdade, acabaram por seguir mais de perto o projeto instrucional. Estes alunos tenderam a permanecer tentando dentro da fase em que se encontram, em vez de procurar progredir para outras fases.

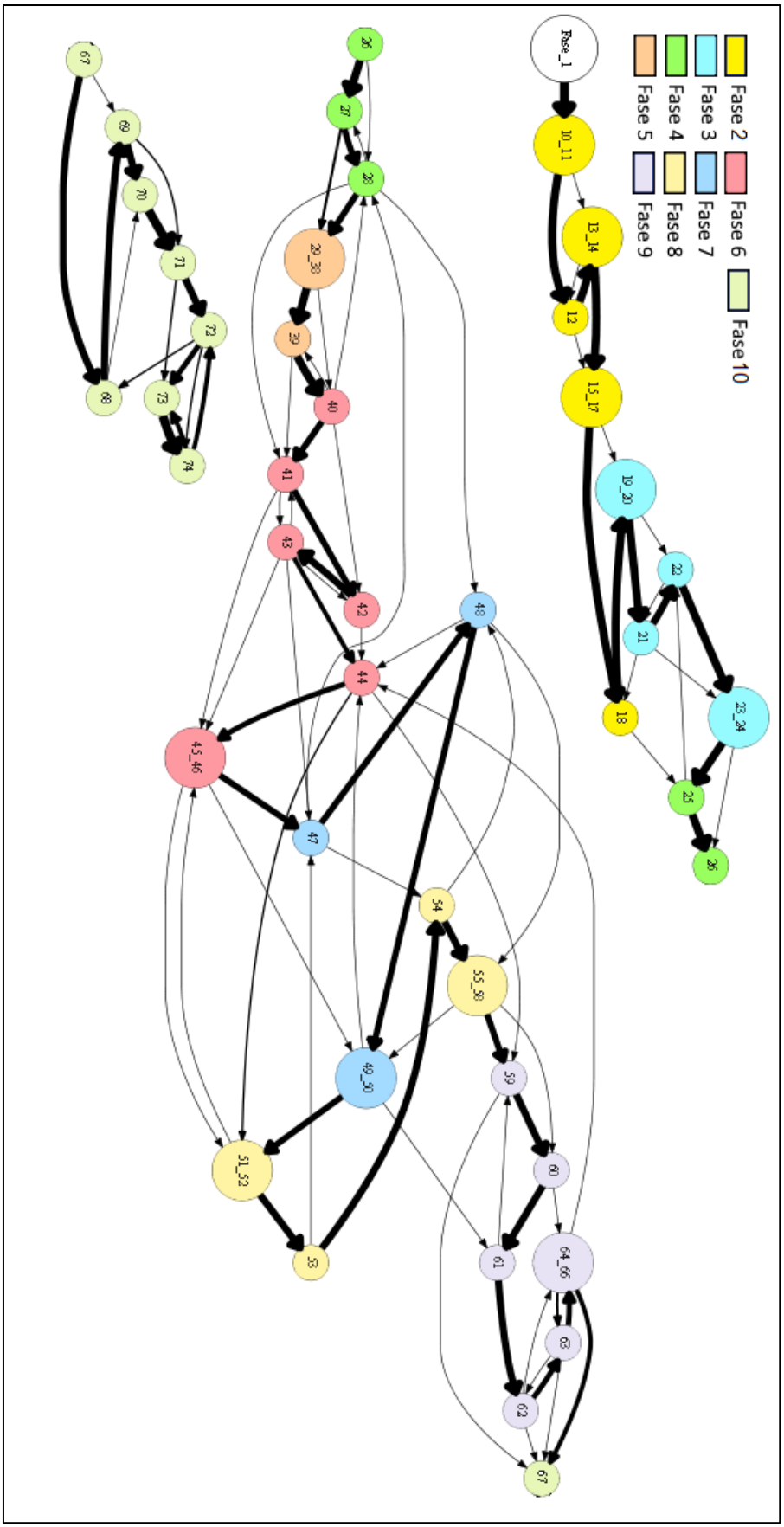


Figura 4.75 – Sequência de finalização das missões para os alunos com possibilidade de escolherem dentro da fase

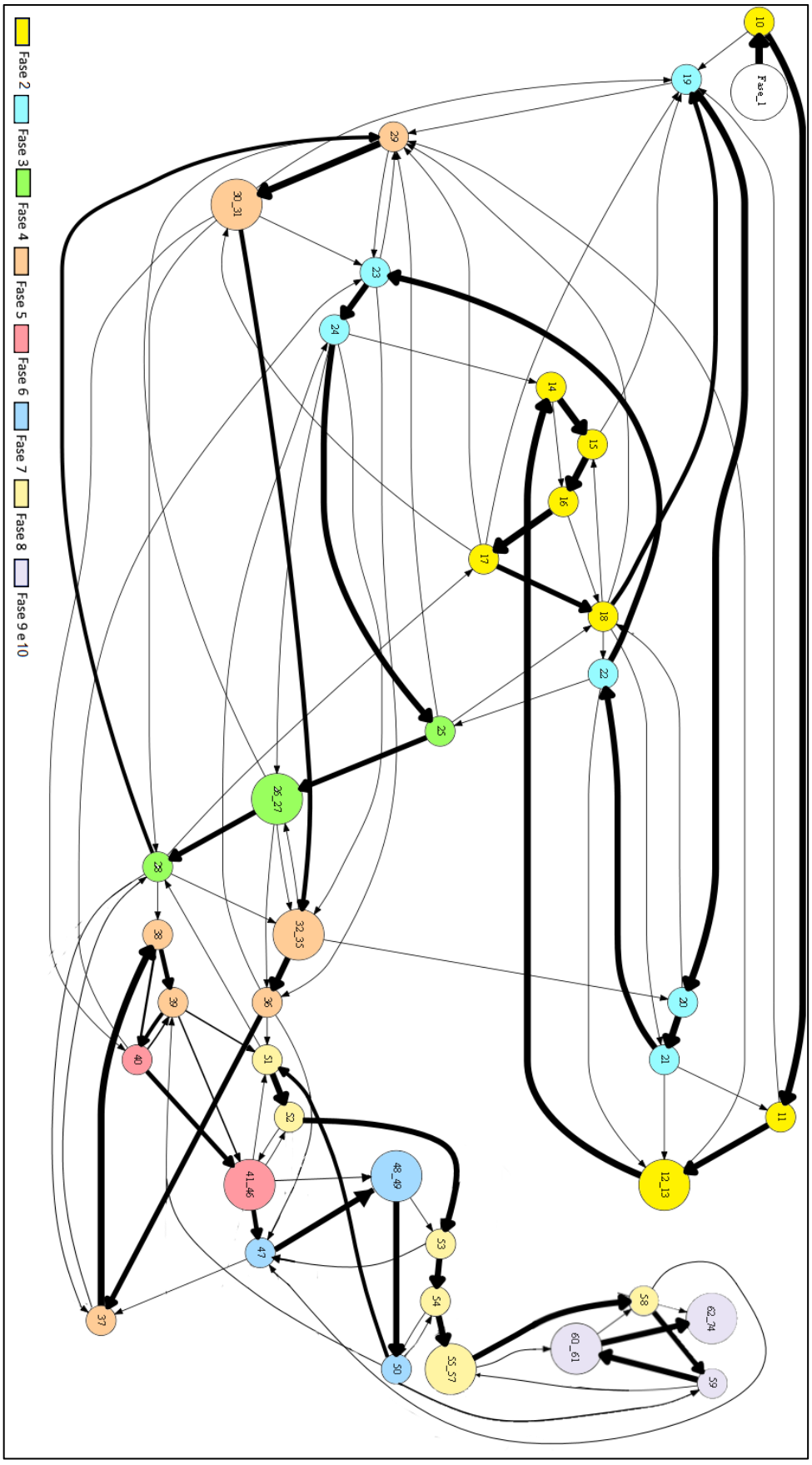


Figura 4.76 – Sequência de finalização das missões para os alunos sem possibilidade de escolherem dentro da fase

4.8.2.3.4 Relação do Jogo com Desempenho Acadêmico

O desempenho acadêmico neste contexto foi aferido através das notas de dois exames. O primeiro exame foi realizado tendo os alunos estudado somente com o jogo. A Tabela 4.34 mostra o resultado de cada questão nesse exame. Algumas questões aceitavam respostas parciais, e por essa razão, o aluno podia receber uma fração dos pontos da questão. A questão 1 foi a leitura de um código e o aluno tinha que responder escrevendo um número em relação à execução. Como pode ser observado no exame (Apêndice L), essa questão exigia muita atenção na leitura e anotação dos itens coletados na passagem do robô, e não havia resposta parcial, o que levou somente 30% dos alunos acertarem. A questão 2 foi de múltipla escolha também com a leitura de um código. Na questão 3 o aluno precisava corrigir dois erros. 40% dos alunos resolveram essa questão, onde 16,7% corrigiram ambos os erros e 23,3% apenas um deles. Na questão 4 o aluno precisava completar o código. Na questão 5 o aluno precisava criar a solução. A média final da disciplina para ser aprovado é 7,0 (sete) na escala de 10,0 (dez). O primeiro exame foi feito por 30 alunos e a média da turma foi 5,4 (±2,33) onde 33% deles (n=10) obteve nota maior ou igual a 7,0.

Tabela 4.34 – Respostas do Primeiro Exame

	Q1	Q2	Q3	Q4	Q5
100%	30,0%	86,7%	16,7%	56,7%	10,0%
50%< 75%			23,3%		20,0%
<50%					13,3%

O segundo exame envolveu a programação em Java de cinco questões. Esse exame foi feito por 25 alunos e a média da turma foi 6,0 (±2,48) onde 48% deles (n=12) obteve nota maior ou igual a 7,0.

Para analisar o impacto da experiência no jogo com o desempenho, nós examinamos a correlação bivariada de Pearson entre as notas dos dois exames, a quantidade de missões concluídas e o tempo despendido para resolver as missões. A Tabela 4.35 mostra o resultado do teste. Nós encontramos correlações moderadas positivas entre a quantidade de missões concluídas e as notas dos exames, assim como correlações fracas entre o tempo e as notas dos exames. Com isso, podemos concluir quanto mais os alunos jogam, maior é a probabilidade de terem um desempenho melhor nos exames. O tempo gasto não tem tanta relação a isso, pois

ele é suscetível a eventos externos e no comportamento do aluno em como resolve as missões.

Tabela 4.35 – Correlação entre os exames e a experiência de jogo

	Exame 1	Exame 2	Total de missões	Tempo despendido
Exame 1	-	0,770**	0,666**	0,452*
Exame 2		-	0,641**	0,410*
Total de missões			-	0,767*
Tempo despendido				-

** p<0,01 * p<0,05

Para conduzir as próximas análises, dividiremos a turma em dois grupos usando a nota do primeiro exame com o valor 7,0 como corte. Além disso, para dar maior significância à análise, consideramos somente os alunos que tenham concluído pelo menos 50% das missões do tipo de tarefa “Criar” (n=33), pois referem-se objetivo principal no ensino de programação que é o aluno desenvolver a sua solução com base no enunciado de um problema. A Tabela 4.36 mostra a divisão entre os dois grupos, e entre todas as missões e somente àquelas de criação, a quantidade de tentativas, tempo despendido e quantidade de missões concluídas. Também é apresentado o tipo de conhecimento prévio em programação (N=nenhum e F=frequentou cadeiras), se teve a possibilidade de jogar qualquer missão dentro da fase e a nota do primeiro exame. A criação desses grupos vem da necessidade de tentar compreender a diferença de comportamento entre um grupo e outro.

Tabela 4.36 – Grupos a serem adotados nas próximas análises

	Todas as missões (n=74)			**Missões de criação (n=33)**					
Grupo	**Tentat.**	**Tempo**	**Qtdade**	**Tentat.**	**Tempo**	**Qtdade**	**CP**	**Possib.**	**Nota**
1	657	105932	67	224	60378	30	N	S	3,0
	182	38052	41	99	27633	16	N	S	4,0
	106	29390	40	50	20876	16	N	N	5,0
	207	85859	67	132	66147	30	N	S	5,0
	148	63020	72	75	45941	31	N	N	5,5
	351	64930	70	156	44766	31	N	S	5,5
	227	71905	62	147	50195	26	N	S	6,0
	176	25792	42	95	15950	16	F	S	6,5
2	332	105619	74	190	74914	33	N	N	7,0
	106	25783	61	63	19720	26	N	N	7,5
	432	108575	74	207	67249	33	N	N	8,0
	152	28675	74	100	20754	33	F	N	8,0
	172	48131	72	107	32513	31	N	S	8,0
	111	56275	65	64	39171	29	F	N	8,5
	219	57096	62	131	40444	26	N	N	8,5
	494	177507	74	304	133113	33	N	S	9,0
	88	35859	54	47	24523	22	N	S	10,0

Foram executados testes não-paramétricos de Mann-Whitney para comparar as médias entre as colunas da Tabela 4.36, e a única diferença estatisticamente significativa foi na nota do exame (p=0,000), o que é esperado pois foi a informação

usada para dividir os grupos. Esses testes demonstram que para essa amostra de 17 alunos não houve diferenças entre a quantidade de missões que foram jogadas, nem o conhecimento prévio ou possibilidade de terem jogado as missões que quisessem dentro das fases. Vamos tentar identificar outras diferenças nas próximas secções.

4.8.2.3.5 Dificuldade das Missões

A revisão no projeto instrucional objetivou diminuir a dificuldade em algumas missões. Nessa subsecção vamos verificar o reflexo desses ajustes.

As figuras contêm três grupos: os grupos 1 e 2, conforme a divisão na secção anterior, e o grupo 0, que representa todos os demais alunos.

A Figura 4.77 apresenta as missões da fase 2. Apesar de ter inserido uma nova missão para tentar ajudar os alunos a entenderem melhor o uso de operadores matemáticos, a missão 12 (terceira dentro da fase) ainda continua com alto índice de tentativas. Na maioria das vezes, a mediana do grupo 2 é menor ou igual que do grupo 1. As Figuras 4.78 a 4.85 apresentam as tentativas das fases 3 a 10.

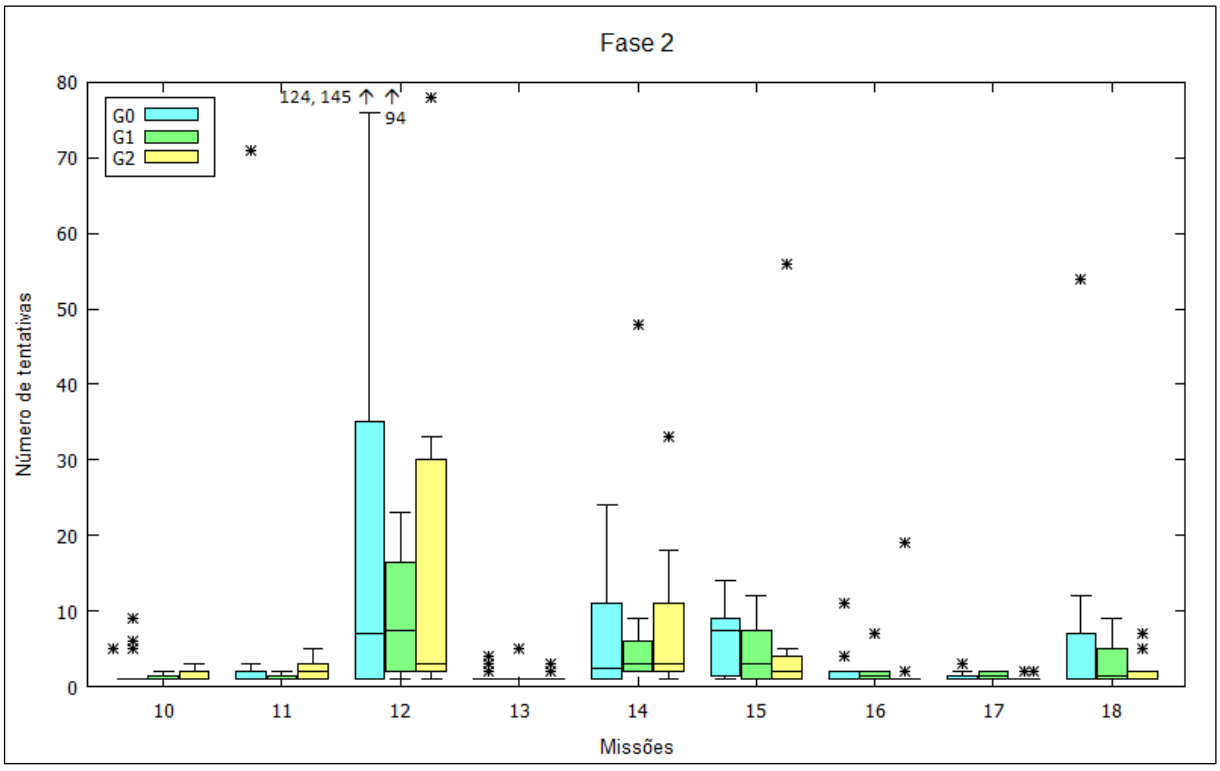


Figura 4.77 – Tentativas na Fase 2 do Ciclo 4

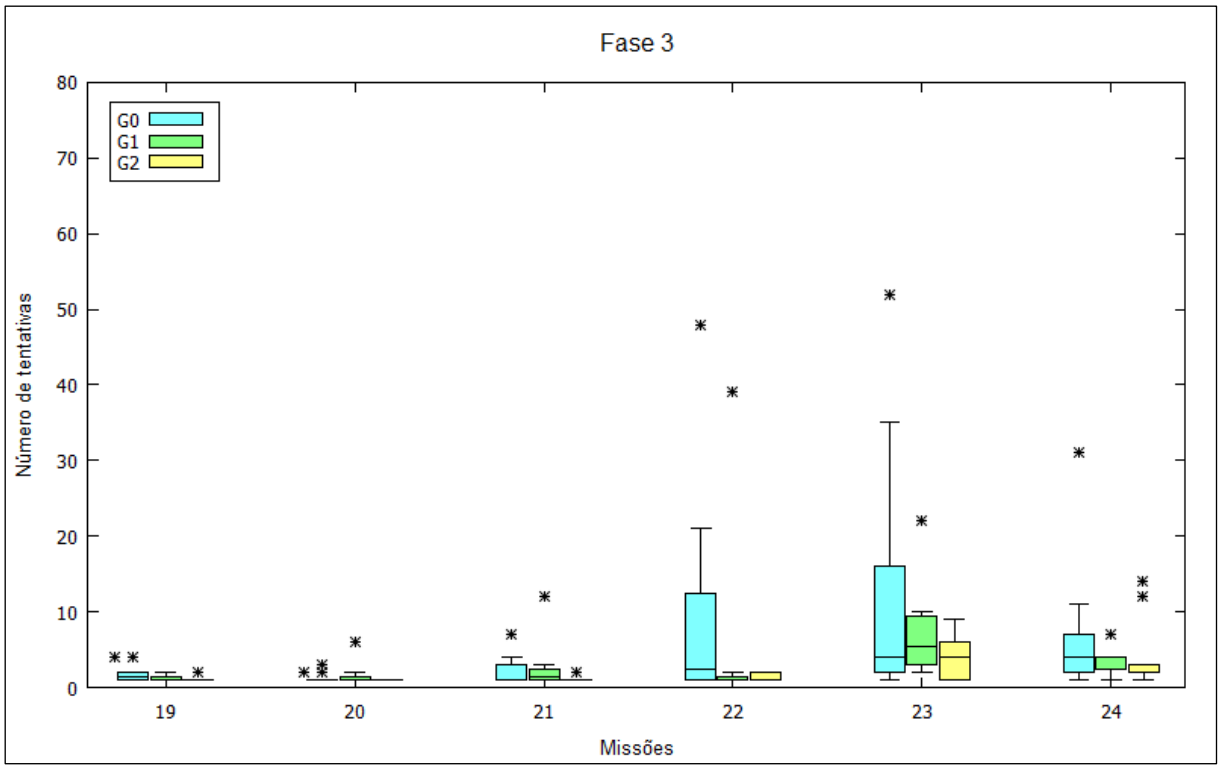


Figura 4.78 – Tentativas na Fase 3 do Ciclo 4

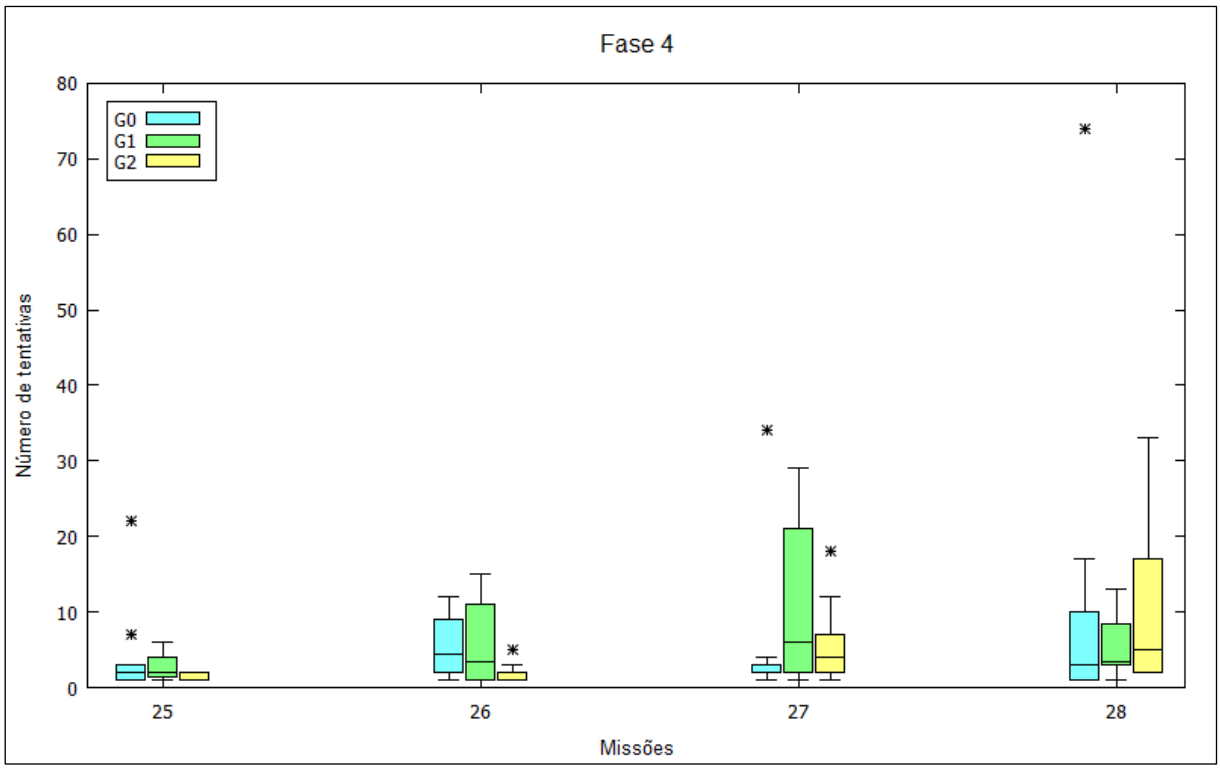


Figura 4.79 – Tentativas na Fase 4 do Ciclo 4

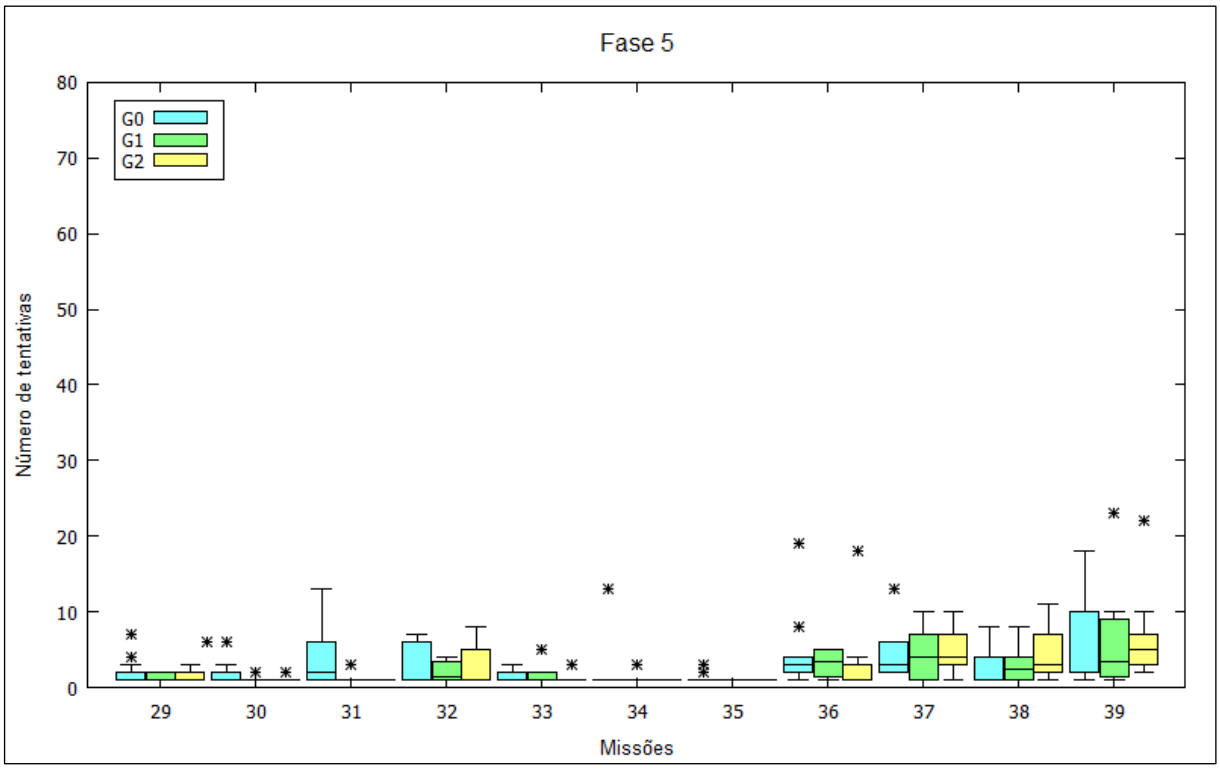


Figura 4.80 – Tentativas na Fase 5 do Ciclo 4

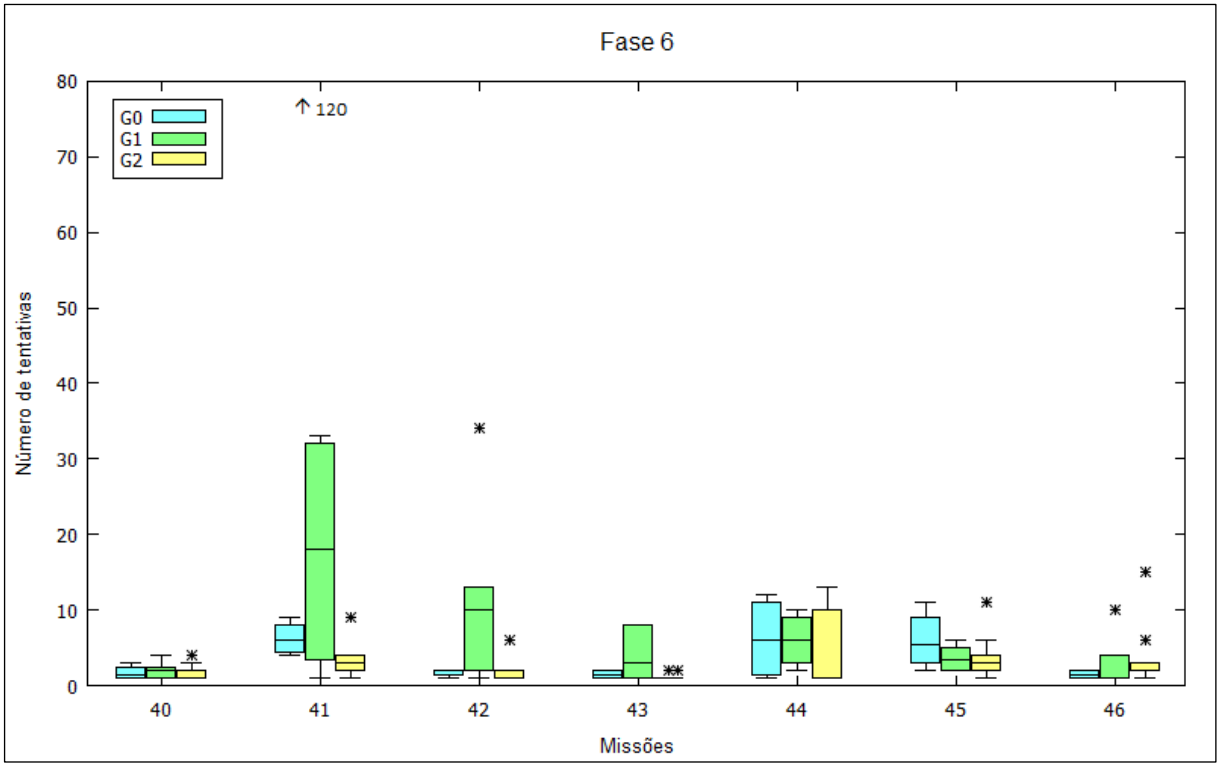


Figura 4.81 – Tentativas na Fase 6 do Ciclo 4

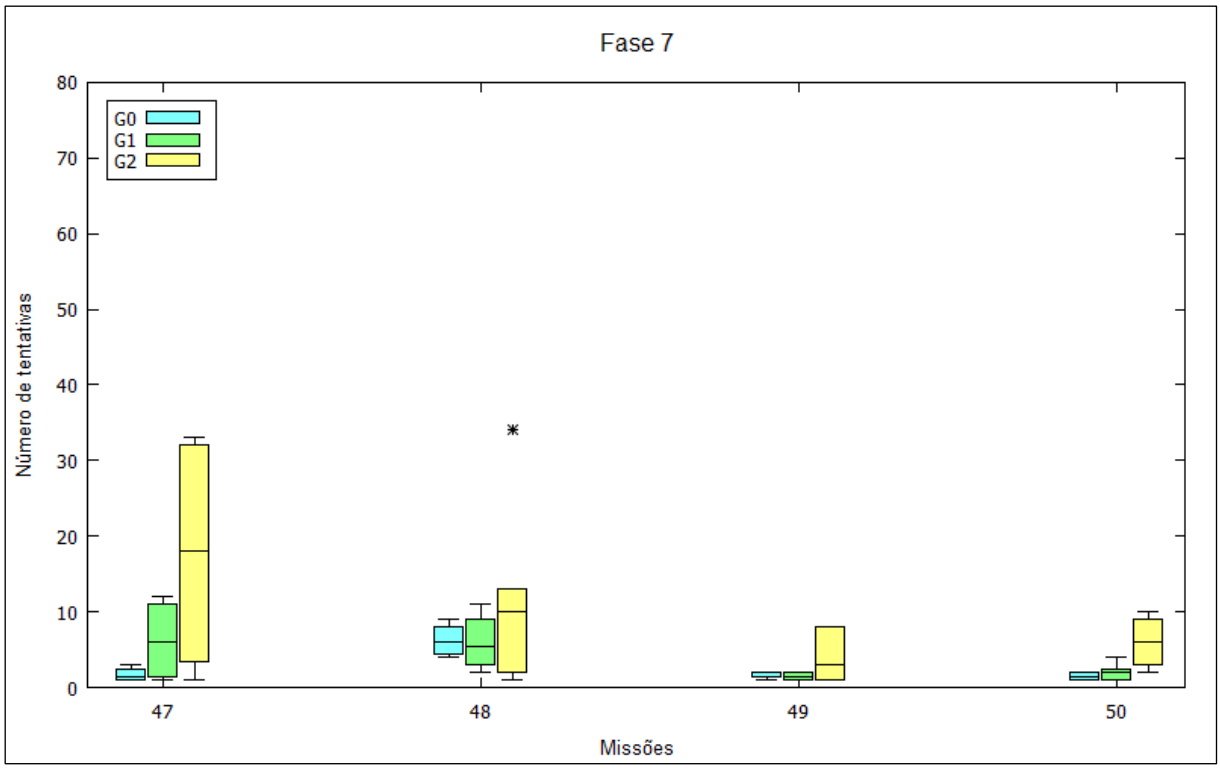


Figura 4.82 – Tentativas na Fase 7 do Ciclo 4

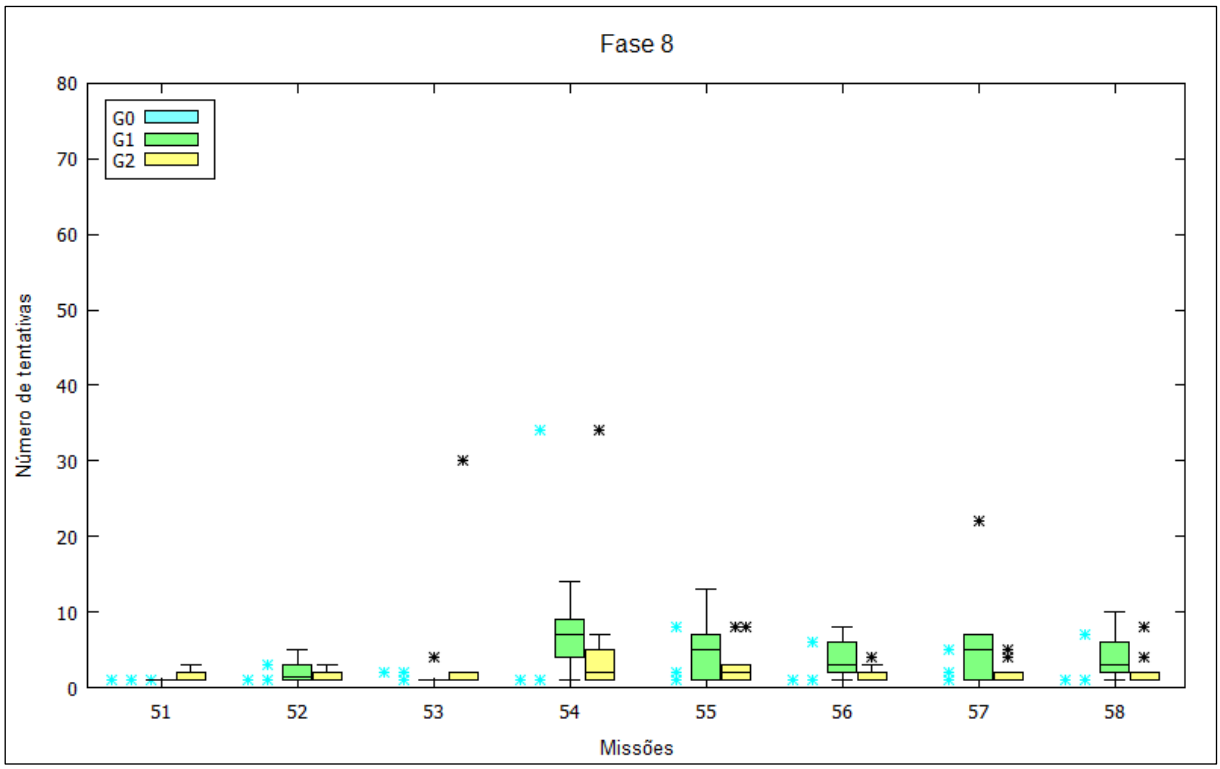


Figura 4.83 – Tentativas na Fase 8 do Ciclo 4

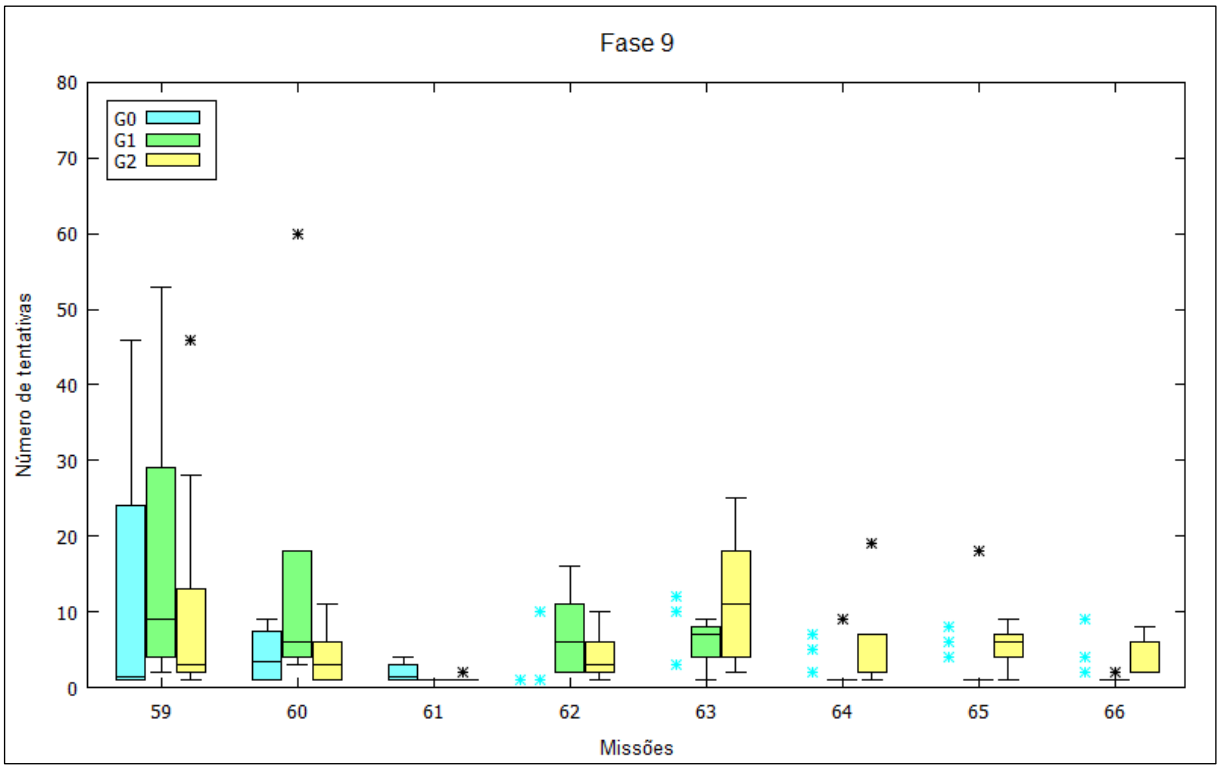


Figura 4.84 – Tentativas na Fase 9 do Ciclo 4

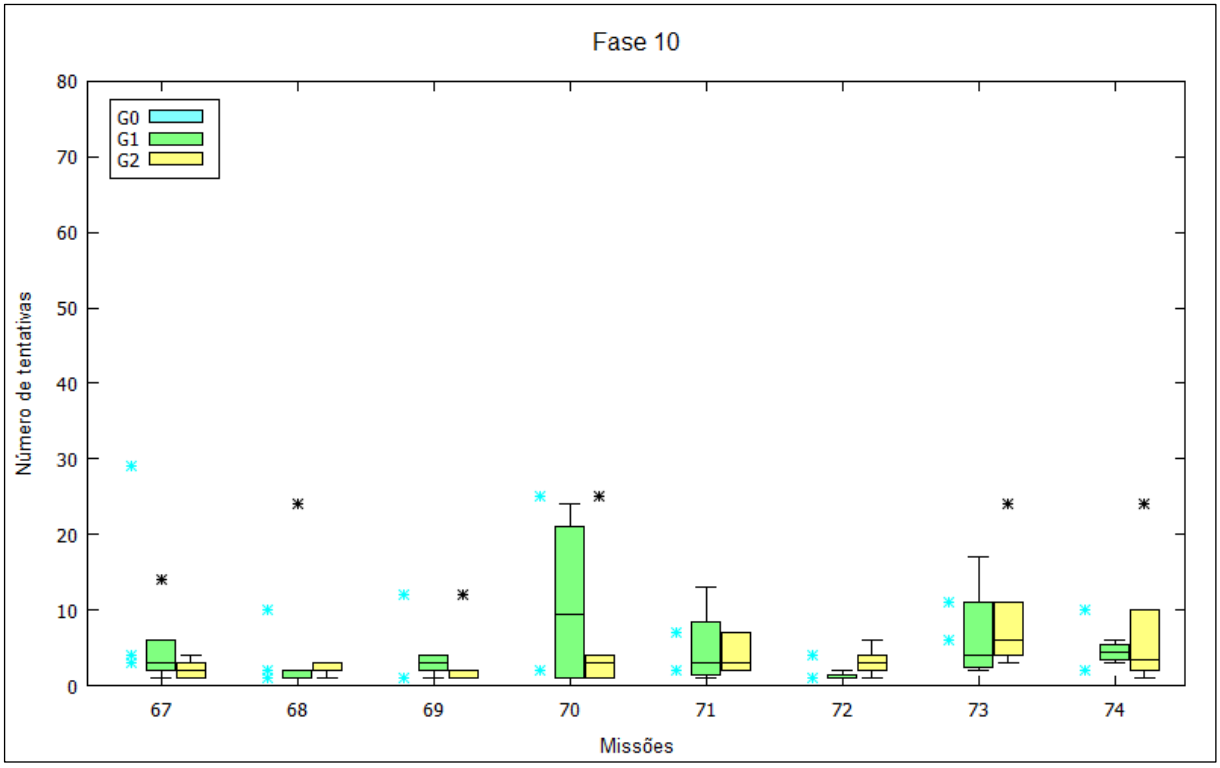


Figura 4.85 – Tentativas na Fase 10 do Ciclo 4

Examinando os gráficos é possível ter uma noção do nível de dificuldade das missões, ao mesmo tempo, da manutenção do fluxo de diversão (Schell, 2008). No ciclo anterior listamos todas as missões que tiveram mais de 5 tentativas na sua

mediana, classificando assim as mais difíceis. Seguindo o mesmo procedimento, a Tabela 4.37 apresenta essa mediana de tentativas das missões de todas as fases (desconsiderando a primeira fase). Tanto nesse ciclo como no anterior, todas as missões com mais de 5 tentativas de medianas foram incluídas na análise, e podemos observar que no ciclo anterior as missões mais difíceis estavam concentradas nas quatro primeiras fases e neste ciclo, com exceção da missão mais difícil, as demais estão distribuídas entre as quatro últimas fases.

Tabela 4.37 – Missões com maiores medianas na quantidade de tentativas no ciclo 4

Fase	Missão	Mediana	Quartil maior	Quantidade	Erros	Objetivos não cumpridos	Total	Total das Execuções
2	12	8	32,25	30	53	496	549	794
7	48	8	16,50	15	26	107	133	172
9	63	8	11,00	13	77	10	87	117
10	73	6	9,50	11	29	26	55	89

Apesar de ter inserido uma nova missão e trocado o tipo de tarefa, a missão 12 (que era a missão 9 no ciclo 3) continua sendo a mais difícil. No ciclo anterior, 33 alunos jogaram a missão e finalizaram com 474 execuções, a mediana foi 9 e o quartil maior 19. Enquanto no ciclo anterior houve vários tipos de erros, nesta todas as 53 vezes em que houve interrupção da execução foi causada por não adicionar as variáveis na expressão matemática. As 496 tentativas sem sucesso foram causadas por não terem conseguido indicar o resultado esperado. Podemos considerar que os alunos mostraram não saber como resolver a soma de três números, e a missão adicional não auxiliou os alunos a entenderem como resolver essa missão.

Na missão 48, a solução foi interrompida 21 vezes porque tentava-se aceder o valor de uma variável não inicializada e 104 vezes não foi cumprido o objetivo de falar o valor esperado. Nessa missão o aluno tinha que usar três condicionais, e dentro delas manipulava algumas variáveis, acumulando valores à medida que a solução avançava. Os alunos tentaram manipular variáveis não inicializadas, além de terem dificuldades em calcular a média aritmética de uma soma.

A missão 63 envolve um ciclo repita-enquanto com três condicionais aninhados entre si. Essa é uma missão com alguma complexidade que é mais bem resolvida incrementalmente. Nessa altura, consideramos que os alunos já não se importam mais com a pontuação como nas primeiras missões, mas com a resolução do problema, nem que exija mais tentativas. Isso sugere que o jogo seja revisto para que

esse tipo de problemas mais complexos possa ser resolvido incrementalmente com algum outro tipo de contagem de pontos que não a quantidade de tentativas.

A missão 73 carece da mesma possibilidade de desenvolvimento incremental. O aluno precisa resolver com dois ciclos repita-enquanto aninhados, e dentro do ciclo mais interno estão mais dois condicionais aninhados. Foram vários os tipos de erros, mas, assim como na missão anterior, podemos considerar como erros naturais no desenvolvimento de uma solução.

Com base na análise dessas missões, podemos conjeturar que os alunos se tenham desenvolvido mais naturalmente nas fases iniciais, visto que as maiores dificuldades se localizam nas últimas missões. Ainda é preciso rever alguma missão anterior à 12 que demonstre como somar três variáveis.

4.8.2.3.6 Uso do Assistente com Pseudocódigo

A Figura 4.86 apresenta a distribuição percentual entre as missões que foram concluídas usando ou não o assistente. Pode-se constatar que as duas missões em que mais alunos usaram este recurso foram a 65ª com quase 50% dos alunos e 66ª com 30% dos alunos. As missões 21 a 23 fazem parte da fase de aperfeiçoamento das variáveis, tendo o assistente sido usado por 30% dos alunos nessas missões. A Figura 4.87 apresenta a comparação do uso do assistente entre os dois grupos, segundo o desempenho acadêmico. Ainda foi adicionado outro grupo referente aos alunos que não foram classificados nos dois grupos anteriores (aqueles que fizeram menos que 50% das missões do tipo Criar). Em algumas missões o assistente foi usado com exclusividade por alunos do G1 (por exemplo, 37 e 38) ou por alunos do G2 (16, 32 e 55). Um teste não-paramétrico Mann-Whitney foi executado para comparar a quantidade de vezes entre os dois grupos resultando em p=0,001 e, uma vez que o Mean Rank do G1 (alunos com notas mais baixas) foi superior, podemos concluir estatisticamente que esse grupo usou mais o assistente.

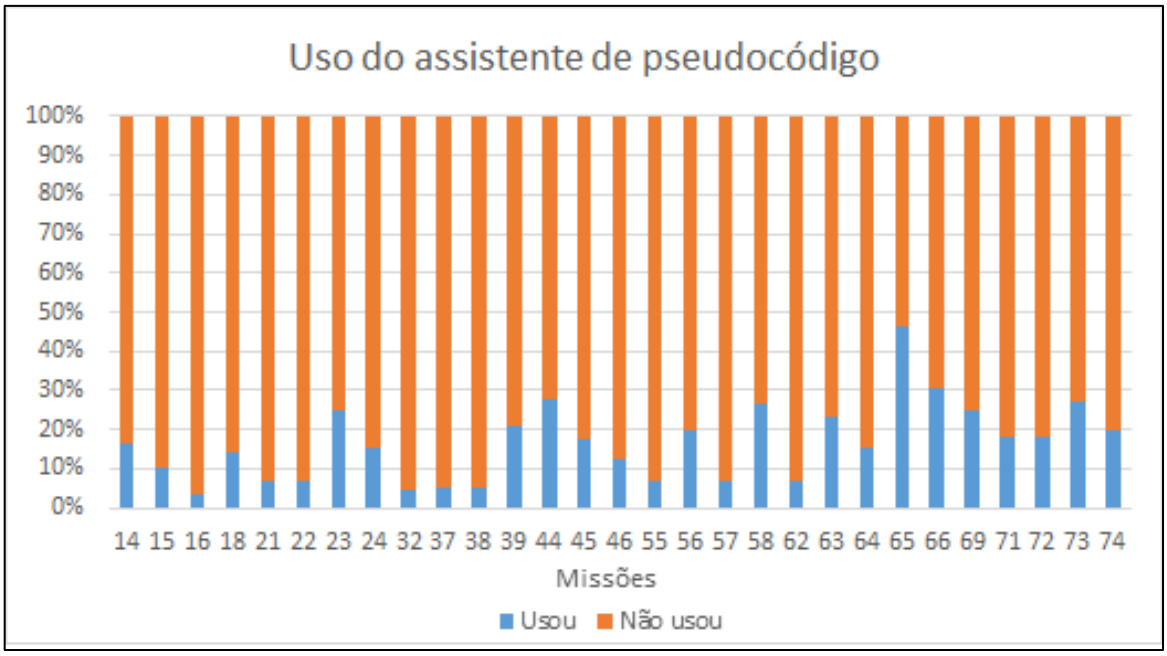


Figura 4.86 – Uso do assistente de pseudocódigo no Ciclo 4

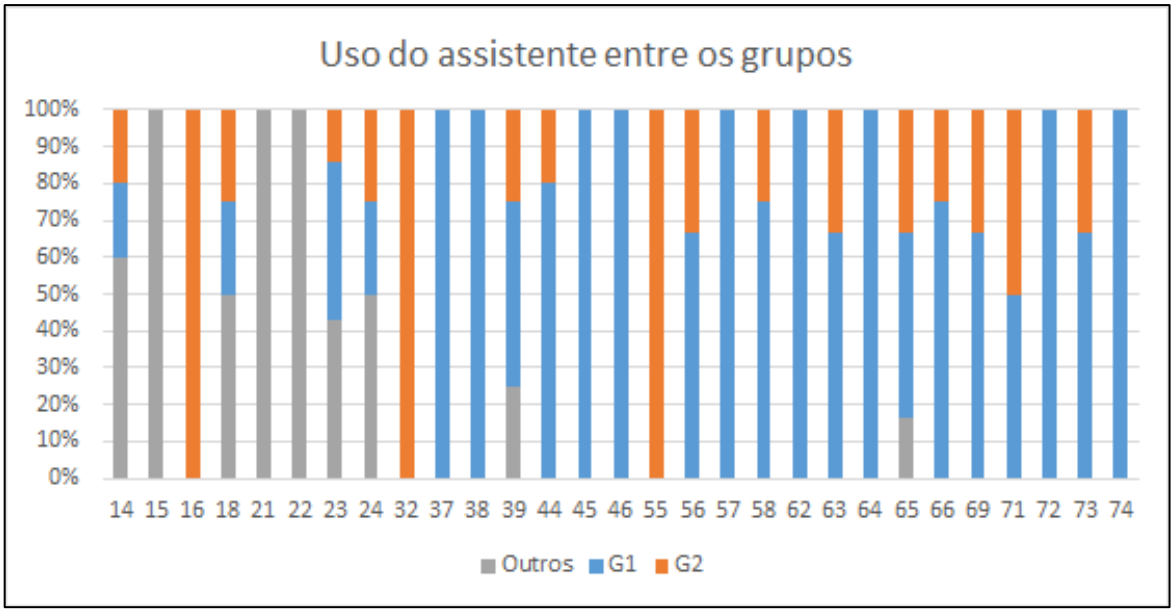


Figura 4.87 – Comparação do uso do assistente entre os grupos

Para verificar o quão eficaz foi o uso do assistente, calculamos a diferença entre a tentativa em que o assistente foi disparado e a tentativa de conclusão da missão. O assistente foi usado 82 vezes por 18 alunos. Em média foram necessárias 4,9 (± 7,0) tentativas entre o momento de usar o assistente e de concluir a missão, com uma mediana de 2 tentativas. Para averiguar a eficiência dos alunos com o uso do assistente entre os ciclos 3 e 4, executamos o teste não-paramétrico Mann-Whitney que resultou em p=0,086 não havendo diferenças estatísticas significantes entre os ciclos.

4.8.2.3.7 Aprendizagem Percecionada

Nessa seção vamos comparar a relação entre a aprendizagem medida através do primeiro exame e o sentimento dos alunos sobre a sua aprendizagem. Para isso usamos um inquérito (Apêndice N) de 10 questões com base no inquérito já aplicado no ciclo anterior e agruparemos as respostas conforme a divisão sugerida na secção 4.8.2.3.4 pelo desempenho acadêmico. Os alunos responderam o inquérito antes de realizarem o primeiro exame. A Figura 4.88 apresenta os resultados desse inquérito, onde a primeira coluna são as respostas do G1 e a segunda do G2. Visualmente as médias do G2 são maiores que do G1. Executamos testes não-paramétricos Mann-Whitney para verificar em quais itens existem diferenças significativas entre os dois grupos. A Tabela 4.38 mostra o resultado dos testes. Em três itens foram encontradas diferenças significativas. Observando esses itens quanto aos seus *Mean Ranks*, podemos concluir que o G2 sentiu maior confiança quanto a aprendizagem de ciclos (Q4), observou uma maior relação das missões no jogo com o conteúdo das aulas (Q9) e tem mais certeza na recomendação do uso jogo para o próximo semestre (Q10). Nos demais itens não existem diferenças significativas, porém, ao totalizar os itens, podemos concluir que os alunos do G2 tiveram mais confiança na sua aprendizagem do que os do G1 (p=0,000).

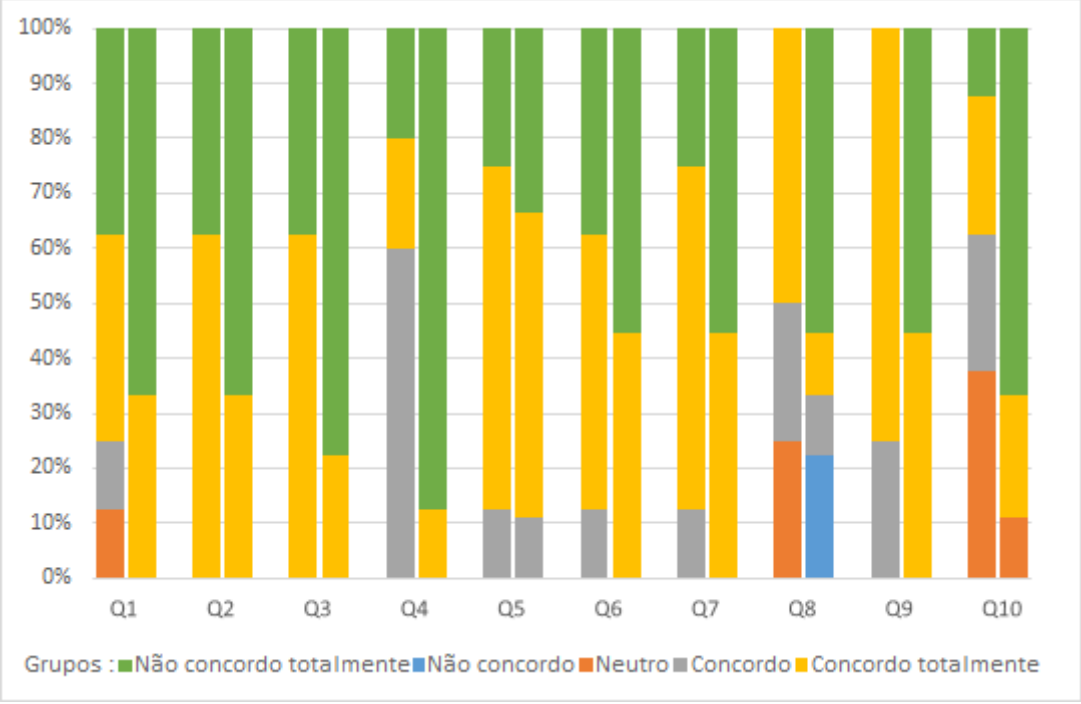


Figura 4.88 – Aprendizagem Percecionada no Ciclo 4

Tabela 4.38 – Análise da Aprendizagem Percecionada entre os Grupos

Item	Mean Rank G1	Mean Rank G2	p-value
1	7,31	10,50	0,200
2	7,69	10,17	0,321
3	7,19	10,61	0,167
4	4,00	8,88	0,030*
5	8,62	9,33	0,815
6	7,94	9,94	0,423
7	7,38	10,44	0,236
8	7,38	10,44	0,236
9	6,00	11,67	0,021*
10	6,19	11,50	0,027*
Total	**64,38**	**100,04**	**0,000***

*p-value<=α=0.05

Finalmente, vamos analisar as respostas no inquérito final do ciclo 3 com essas respostas do ciclo 4. Executámos testes não-paramétricos Mann-Whitney para verificar em quais itens existem diferenças significativas entre os dois grupos. A Tabela 4.39 mostra o resultado dos testes. Três itens tiveram diferenças estatisticamente significativas: “2-Eu aprendi com o jogo como manipular variáveis”, “3-Eu aprendi com o jogo a usar condicionais” e “6-Eu aprendi com o jogo como é importante depurar para resolver os erros”. Além disso, somando todos os itens, os alunos do ciclo 4 apresentaram um sentimento maior de confiança naquilo que aprenderam do que os do ciclo 3 (p=0,000).

Tabela 4.39 – Análise da Aprendizagem Percecionada entre os ciclos

Item	Mean Rank Exp3	Mean Rank Exp4	p-value
1	17,60	21,85	0,243
2	15,88	23,97	0,024*
3	14,74	22,71	0,023*
4	12,18	15,96	0,220
5	17,00	22,59	0,128
6	15,79	24,09	0,021*
7	16,57	23,12	0,073
8	20,54	20,44	0,977
9	20,50	20,50	1,000
10	20,78	20,12	0,852
Total	**167,98**	**210,72**	**0,000***

*p-value<=α=0.05

4.8.2.3.8 Diversão Percecionada

Com base nos ciclos anteriores e inspirado no EGameFlow, desenvolvemos um inquérito (Apêndice O) com 26 questões divididas com as mesmas 6 dimensões. A Figura 4.89 apresenta o resultado das questões, a primeira coluna as respostas do G1 e a outra do G2. Foram executados testes não-paramétricos Mann-Whitney para avaliar se existe diferença significativa. Apenas um item apresentou diferenças com

p=0,011: “O1-As metas globais do jogo foram apresentadas com clareza” e através do *Mean Rank* é possível concluir que o G2 concorda mais que o G1 com essa questão. Também foi executado o teste com o total dos itens, e não houve diferenças significativas. Com isso, podemos concluir que independente do desempenho acadêmico, os alunos tiveram a mesma perceção quanto a sua diversão.

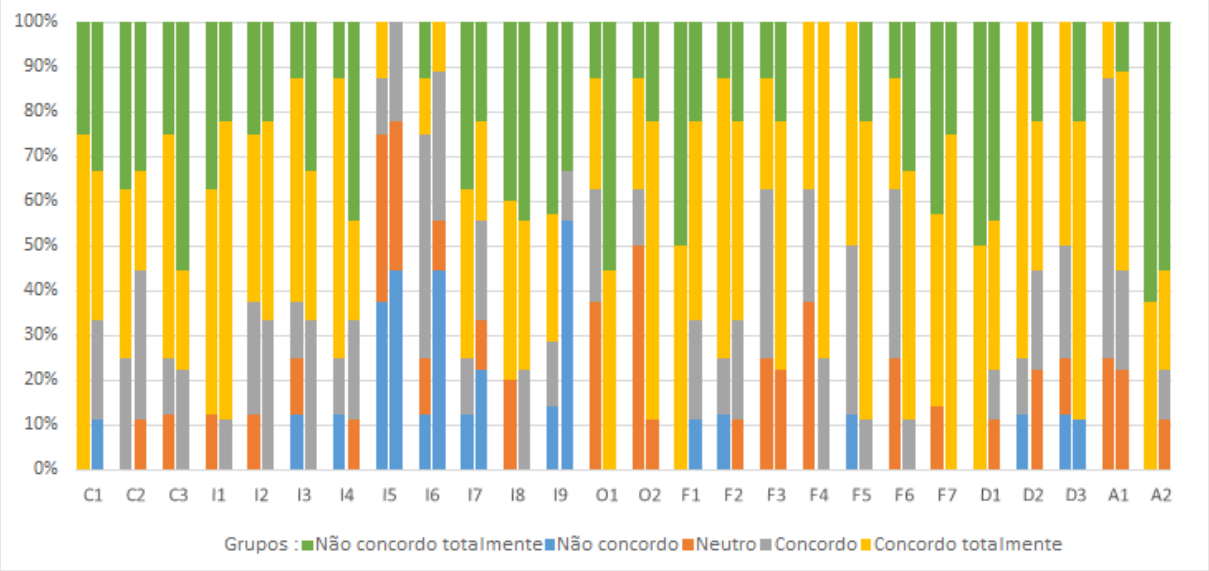


Figura 4.89 – Diversão Percecionada no Ciclo 4

4.8.2.3.9 Diversão Medida

Nesse quarto ciclo conseguimos registrar muitos dados sobre a interação do aluno no jogo (*logs*). Por essa razão, conseguimos determinar quais foram os recursos mais acedidos no jogo.

Chamamos de diversão medida a assiduidade em aceder os elementos não relacionados com a aprendizagem, mas com recursos para manter a motivação em continuar jogando, como a conquista de pontos. Nessa investigação, são três os itens para a diversão considerados mensuráveis: a personalização do avatar, as tabelas de classificações dos jogadores e a conquista de recompensas. A medição ocorre quando o aluno clica para abrir a janela de personalização do avatar ou conquista de recompensas, ou quando manipula as tabelas de classificação. Analisando o *log*, identificamos em quantas secções (a cada autenticação) que cada um desses recursos foi acedido pelos alunos. A Tabela 4.40 apresenta a quantidade de vezes que os alunos de cada grupo fizeram a autenticação, acederam as janelas do avatar, da tabela de classificação e das recompensas.

Tabela 4.40 – Análise da Diversão Medida no Ciclo 4

Grupos	Autenticação	Avatar	Classificação	Recompensas
1	383	52 (13,6%)	61 (15,9%)	26 (6,8%)
2	475	48 (10,1%)	130 (27,4%)	56 (11,8%)
p-value	-	0,115	0,000*	0,013*

*p-value<=α=0.05

Para comparar se existem diferenças significativas entre os grupos, executamos testes de quiquadrado (Chi-square) para verificar se as proporções são as mesmas entre os dois grupos. A proporção foi calculada em relação à quantidade de autenticações. Os recursos da tabela de classificação e recompensas foram usados de forma significativamente diferente pelos grupos. O recurso mais usado pelo G2 foi a visualização da tabela de classificação, seguido pela consulta das conquistas e personalização do avatar. Já no G1, a tabela de classificação também foi o mais usado, seguido pela personalização do avatar e recompensas. Ambos os grupos tiveram a consulta às tabelas de classificação como o mais usado, entretanto, o G2 teve uma maior incidência de uso que o G1.

4.8.2.3.10 Comportamento no Jogo

Nas subsecções anteriores conduzimos análises entre os grupos criados com base no desempenho acadêmico na esperança de nos fornecer pistas no que difere um do outro. Aproveitando os logs, tentamos identificar o comportamento que os alunos assumem quando vão resolver missões do tipo criação. Esse comportamento foi considerado com base na sequência em que realizam uma das cinco ações:

1. Explicação (EXP): é o momento em que o usuário acede a explicação da missão. Quando uma missão é acedida uma primeira vez, automaticamente o jogo mostra a explicação da missão. Entretanto, depois que o aluno fecha essa janela, ele pode retornar a reler a explicação. Não contabilizamos essa primeira vez que mostra automaticamente a explicação;

2. Depuração (DEB): é o momento em que o aluno decide executar passo-a-passo a sua resolução. O programa executa uma linha da solução, composta por um ou mais blocos, a cada vez que o usuário clica no botão de depuração. Uma sessão de depuração consiste no primeiro momento que o usuário clica nesse botão, e termina em uma das seguintes formas: 1-clicando no botão de reiniciar; 2-cumpriu todos os objetivos da missão, terminou com sucesso, e com isso vence a missão; 3-não cumpriu os objetivos da missão, e terminou com fracasso; 4-aconteceu algum erro

durante a execução, por exemplo, perguntar algo a um cliente sem estar ao pé dele, ou tentar obter algum produto sem estar ao pé da máquina que fornece esse produto. Foram contabilizadas a sessões completas independente da forma de finalizar;

3. Execução (RUN): é o momento em que o aluno decide executar a sua resolução. Ele clica no botão de execução, e a solução é iniciada. A execução pode ser encerrada nas mesmas quatro formas apresentadas no recurso de depuração. Uma secção consiste do momento que a execução é iniciada até o seu encerramento. Contabilizamos uma secção completa;

4. Desenvolvimento (DEV): é o momento que o aluno modifica a sua solução, adicionando, removendo ou alterando blocos. O aluno pode modificar várias vezes a sua solução. Definimos uma sessão de desenvolvimento, aquela em que é intercalada pelos outros quatro momentos. Por exemplo, se o aluno leu a explicação, em seguida fez várias alterações, depois executou com falhas, voltou a alterar a solução e executou novamente, definimos aqui duas sessões de desenvolvimento. Também consideramos duas sessões, quando o aluno estava modificando, saiu do jogo, e retornou noutro momento para continuar no desenvolvimento da solução;

5. Revisão (REV): é o momento em que o aluno sai de uma missão que não concluiu, e voltou a experimentar outras missões já concluídas, executando ou depurando, e por fim retornando à missão original. Consideramos isso um momento de revisão dos conceitos, em que o aluno não compreende exatamente o seu erro, ou como resolver uma missão, e experimenta novamente missões finalizadas. Se nesse caminho, o aluno entrou em outras missões não concluídas diferentes daquela em que ele saiu originalmente, já se perde a relação de revisão. Contabilizamos como revisão quando isso acontece dentro da mesma sessão. As missões concluídas podem ser de qualquer tipo, pois esses outros tipos é que dão a introdução e boas práticas nas técnicas de resolução de problemas, e nas missões de criação é que o aluno demonstra o que aprendeu.

A Tabela 4.41 apresenta a quantidade de vezes que aconteceu a transição entre uma ação (De) e outra (Para) em cada um dos grupos. Executamos testes de quiquadrado (Chi-square) para verificar se as proporções são as mesmas entre os dois grupos. Para cada transição em que as proporções não forem iguais, é identificado como um comportamento particular daquele grupo. Por exemplo, a partir da ação EXP indo para a ação RUN, foi identificado a diferença na proporção

(p=0,032). Como o G2 tem o percentual maior que G1 (1,5% > 0,5%), podemos concluir que essa transição é mais característica do G2.

Tabela 4.41 – Transição entre duas ações

De	Para	G1	%	G2	%	Total	p-value
EXP	DEV	719	78,2	788	76,1	**1507**	0,253
	RUN[b]	5	0,5	16	1,5	**21**	0,032
	DEB	22	2,4	36	3,5	**58**	0,160
	REV[a]	3	0,3	3	0,3	**6**	-
	EXP	170	18,5	193	18,6	**363**	0,941
	Total	**919**	**47,0**	**1036**	**53,0**	**1955**	
DEB	DEV	432	72,2	501	72,8	**933**	0,816
	RUN[a]	1	0,2	4	0,6	**5**	-
	DEB	67	11,2	77	11,2	**144**	0,994
	REV[a]	1	0,2	5	0,7	**6**	-
	EXP	97	16,2	101	14,7	**198**	0,445
	Total	**598**	**46,5**	**688**	**53,5**	**1286**	
DEV	EXP	526	35,4	603	35,6	**1129**	0,919
	RUN[b]	311	21,0	406	24,0	**717**	0,042
	DEB[b]	644	43,4	663	39,2	**1307**	0,016
	REV[b]	3	0,2	21	1,2	**24**	0,001
	Total	**1484**	**46,7**	**1693**	**53,3**	**3177**	
REV	EXP[a]	2	6,9	5	2,8	**7**	-
	REV[b]	23	46,0	154	85,1	**177**	0,000
	DEB[a]	0	0,0	1	0,06	**1**	-
	RUN[a]	1	3,4	0	0,0	**1**	-
	DEV	3	10,3	21	11,6	**24**	-
	Total	**29**	**13,8**	**181**	**86,2**	**210**	
RUN	EXP	46	22,8	51	20.3	**97**	0,527
	RUN[a]	1	0,05	4	1.6	**5**	-
	DEB[b]	8	4,0	36	14.3	**44**	0,000
	DEV[b]	147	72,8	160	63.7	**307**	0,041
	REV	0	0,0	0	0.0	**0**	-
	Total	**202**	**44,6**	**251**	**55.4**	**453**	

[a] Duas células com até 5 ocorrências, não sendo possível a execução do teste estatístico.
[b] p-value<=α=0.05

Com base nas transições que apresentaram diferenças significativas, desenhamos um grafo para cada grupo procurando identificar os comportamentos mais evidentes em cada um deles (Figura 4.90 e Figura 4.91). Ao observarmos o grafo do grupo 1, fica evidente que após uma execução fracassada, esses alunos logo em seguida tendem a modificar a solução e depois usam os recursos de depuração. Isso pode ser entendido que a partir da execução eles já confiam saber o que é necessário alterar na solução. Depois da alteração, então preferem usar os recursos de depuração. Em relação ao grupo 2, após reler a explicação eles costumam executá-la provavelmente para tentar identificar o erro não percebido mesmo após a leitura. Após a execução fracassada, esse grupo prefere usar mais a depuração, indicando um cuidado maior em identificar o erro, em vez de ir logo alterar a solução. Enquanto estão desenvolvendo a solução, esses alunos ainda utilizam soluções anteriores.

Podemos observar na Tabela 4.41 que os alunos do grupo 1 utilizam raramente essa estratégia. É interessante verificar a diferença entre o grupo 1 e 2 após modificar a solução: o grupo 1 prefere mais utilizar a depuração, e o grupo 2 prefere mais a execução.

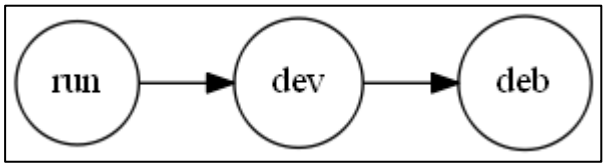


Figura 4.90 – Comportamento evidente do G1

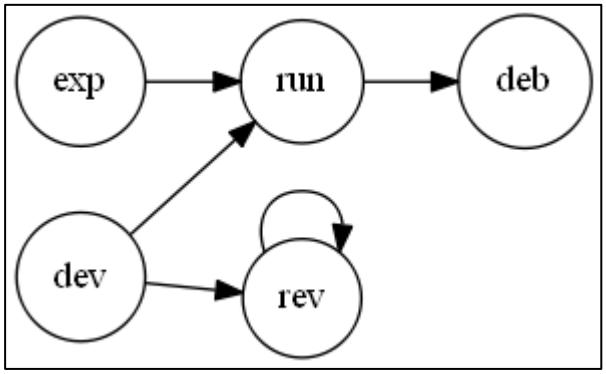


Figura 4.91 – Comportamento evidente do G2

4.8.2.3.11 Avaliação do uso do NoBugsJ

Após o primeiro exame os alunos aprenderam Java iniciando por exercícios com NoBugsJ e depois o professor usou exercícios tradicionais. Antes de aplicar o segundo exame, os alunos responderam um inquérito final (Apêndice P) respondido no formulário em papel. Foram recolhidas 29 respostas anônimas apresentadas na Figura 4.92. Os itens Q1 e Q2 demonstram que a maioria dos alunos concorda que o jogo junto com NoBugsJ foram úteis para a aprendizagem deles. As respostas dos itens Q3, Q4 e Q5 foram menos expressivos, principalmente no Q5 onde a maioria dos alunos não concordam que as práticas com NoBugsJ tenham preparado eles para outros tipos de exercícios de programação. Entretanto, a opinião geral deles é que essa abordagem continue a ser usada no próximo semestre (Q6).

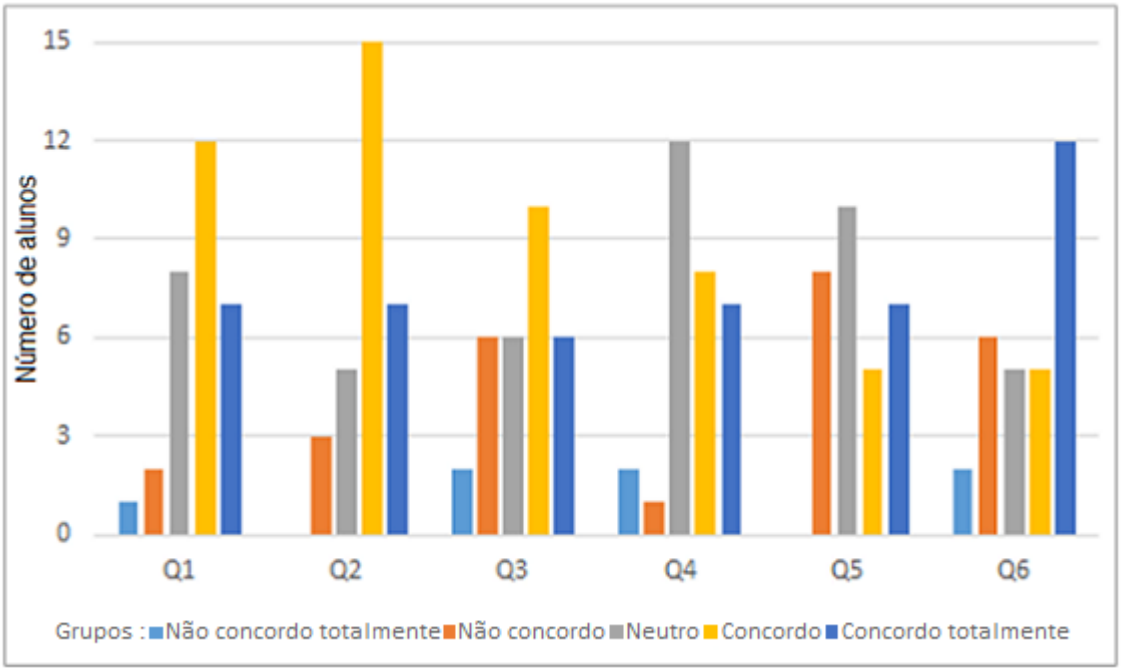


Figura 4.92 – Opiniões dos alunos sobre o uso do NoBugsJ

O questionário também incluía duas questões abertas para ajudar-nos a entender melhor essas respostas: uma para citar aspetos positivos na experiência e outro para apontar falhas e sugerir melhorias. A resposta mais evidente em relação aos aspetos positivos foi que estavam aprendendo programação enquanto se divertiam, mostrando coerência com as questões Q1 e Q2. Outros comentários foram "*nós conseguimos visualizar as ações de nossos comandos*" e "*me ensinou a usar variáveis e condicionais*".

A resposta que apontou aspetos negativos explicou-nos porque os alunos não foram muito positivos nas respostas Q4 e Q5. Essencialmente foram três motivos: primeiro, nos exercícios tradicionais, o usuário é perguntado para entrar com valores que traz um novo conjunto de preocupações, como a validação e transformação, que não são praticados no NoBugsJ. Segundo, a estrutura básica da classe exigida no NoBugsJ, e abordagem Orientada a Objetos associada, trouxe alguma complexidade para o primeiro contato com Java. Finalmente, os alunos sentiram a falta de aspetos do jogo, como conquistar pontos que não estavam presentes no NoBugsJ. Essas limitações parece que afetaram a motivação dos alunos (o que é coerente com as respostas do item Q3).

Esses resultados foram apresentados e discutidos com o professor. Apesar de termos providenciado material instrucional para ajudar na transição entre os blocos e Java, as respostas de todas as missões convertidas, o professor não se sentia

confiante suficientemente para usar o NoBugsJ. A presença do professor em todos os estágios da investigação, do projeto ao desenvolvimento, fortalece a interação na sala de aula, promovem otimização nos testes e intervenções mais significativas (Brown, 1992). No nosso caso o professor executou a experiência, porém ele não contribuiu no projeto e desenvolvimento dos artefactos. Provavelmente isso limitou a sua habilidade em relacionar as missões do NoBugsJ com os exercícios tradicionais. Entretanto, essa nova abordagem não aumentou a quantidade de aulas para atingir os objetivos da disciplina.

4.8.2.3.12 Resumo dos resultados

Para esse ciclo havia dois objetivos:

(1) Identificar a melhoria no desempenho na disciplina com o uso do jogo e o *framework* em Java

Encontramos correlações significativas entre o desempenho dos alunos e a quantidade de missões jogadas. Em relação a aprendizagem percecionada, os alunos desse ciclo sentiram uma maior confiança no conhecimento do que no ciclo anterior. Usar o jogo na totalidade da parte introdutória da disciplina demonstrou ser a melhor alternativa de integração entre o jogo e as aulas.

(2) Identificar o comportamento dos alunos quanto ao uso dos recursos do jogo

Permitir que os alunos pudessem aceder a qualquer missão dentro da própria fase possibilitou que eles tivessem foco no assunto que estavam estudando. Se o aluno não estivesse progredindo na missão, ele tentava a seguinte que poderia dar um subsídio para compreensão da missão anterior. Apesar de termos alterado o projeto instrucional, uma missão da fase 2 continua sendo a mais difícil. Essa é a primeira missão em que os alunos precisam inserir novos blocos. Provavelmente precisamos prepará-los para experimentarem fazer a soma com duas variáveis, ou mesmo com três, mas utilizar o tipo de tarefa “Organizar os blocos”. As duas missões anteriores não forneceram subsídio suficiente para eles superarem os desafios dessa missão. As missões mais difíceis exigem naturalmente um processo de desenvolvimento incremental não suportado no jogo. Como os alunos têm receio de gastar tentativas para não perderem pontos, tentam resolver o problema sem a decomposição. Um aluno com nota 9,0 foi constantemente classificado com um valor

discrepante, pois ele não se importava com a diversão, mas com a aprendizagem, por isso ele fazia várias tentativas incrementando a sua solução.

Os alunos gostavam de aceder as tabelas de classificação no jogo. O grupo dos alunos com melhor desempenho acadêmico acederam mais a janela de conquista do que os outros alunos que, por sua vez, preferiram aceder mais a personalização do avatar do que os alunos com melhor desempenho acadêmico. Essa preferência no jogo explica porque os alunos gostariam que mantivesse algum sistema de pontuação com NoBugsJ: eles gostam de se compararem uns com os outros. Além disso, os alunos sentiram que não foi natural a migração de NoBugsJ para os exercícios tradicionais de programação, sentindo a falta de alguns objetivos comuns nesses exercícios, como a transformação e validação das entradas.

4.9 Considerações Finais

O jogo desenvolvido e investigado nesta tese, chamado de NoBug's SnackBar, abrange os conteúdos essenciais de uma primeira disciplina de programação, tais como sequenciação, variáveis, condicionais e ciclos. O jogo foi inspirado nas mecânicas de gestão de tempo, em que ele vai apresentando padrões diferentes através de objetos e de combinação de objetos ao jogador que deve criar um produto que combine com o padrão apresentado pelo jogo. O contexto do jogo é uma esplanada e o jogador desempenha o papel do encarregado que atende os pedidos de comer e de beber dos clientes. O jogador comandará o encarregado construindo um programa através de blocos. O jogo corre em navegadores usando HTML5 e dispensa, assim, a sua instalação e de programas adicionais.

A investigação foi conduzida através de quatro ciclos em semestres distintos. Após cada ciclo, a partir de análise qualitativas e quantitativas foram detetadas conjeturas e o jogo foi modificado para serem testadas. Com isso, todos os recursos do jogo, assim como o projeto instrucional, foram descritos nesse capítulo. Com base nessas experiências, como resultados da investigação, foram estabelecidas teorias sobre os elementos de jogo e a forma de integrar o jogo nas aulas, que serão apresentadas no próximo capítulo.

Capítulo 5

RESULTADOS DA INVESTIGAÇÃO

5.1 Considerações Iniciais

Os quatro ciclos descritos no capítulo anterior objetivaram desenvolver o jogo e definir a sua integração com as aulas através de um processo de retroalimentação e constante evolução. A partir dessas intervenções foram produzidas várias observações, e algumas delas fomentaram mudanças na versão do jogo para o ciclo seguinte, assim como na forma de o aplicar. Durante os ciclos foram avaliados aspetos de diversão e aprendizagem. O jogo seguiu o modelo dos jogos casuais em que, além de outras características, não seja exigido do aluno comprometimento de muitas horas seguidas para finalizar uma tarefa.

Essa investigação resultou em quatro artefactos: (1) o jogo NoBug's SnackBar com suas funcionalidades e estrutura exaustivamente descritos no capítulo anterior; (2) *framework* NoBugsJ para resolver as missões com programação de classes em Java conforme descrito na secção 4.8.1.2; (3) um conjunto de ideias que argumentamos serem boas práticas ou boas considerações para (a) a metodologia de investigação, (b) o projeto e (c) aplicação de um jogo para a aprendizagem introdutória de programação; (4) um modelo de arquitetura de motor de jogos sérios construcionistas que generalizamos a partir do jogo testado, visto que ele passou por várias evoluções, e o seu aperfeiçoamento foi rapidamente aplicado.

5.2 Contributos para a Utilização de Jogos Casuais para a Aprendizagem Introdutória de Programação

Com base em praticamente dois anos envolvidos no processo iterativo e intervencionista de experimentação, formulámos um conjunto de ideias e observações que podem ser distribuídos em três grupos: (1) investigação, onde estão descritos os procedimentos adotados durante o ciclo de projeto, experimentação, conclusões dos resultados e ajuste do projeto para início de um novo ciclo; (2) aplicação do jogo, pois foram testadas três formas diferentes de usar o jogo, e com isso se pode sugerir quando uma forma é melhor que a outra; (3) jogo em si, onde estão listados os recursos que trouxeram a perceção de diversão e aqueles que foram realmente acedidos, e questões quanto a sua utilidade instrucional.

Essa secção está relacionada ao seguinte capítulo de livro já publicado. No entanto, o capítulo foi baseado no segundo ciclo de desenvolvimento, e aqui na tese temos os dois ciclos seguintes para revisar e ampliar os princípios definidos no capítulo:

- Vahldick, A., Marcelino, M. J., & Mendes, A. J. (2017). Principles of a Casual Serious Game to Support Introductory Programming Learning in Higher Education. In R. A. P. Queirós & M. T. Pinto (Eds.), *Gamification-based e-learning Platform for Computer Programming Education*. IGI Global.

5.2.1 Investigação

O primeiro ciclo, com a experiência piloto, foi realizado em sala supervisionado por um investigador ou professor, com a intenção de perceber as oportunidades de aprendizagem e diversão com o jogo. Os três ciclos seguintes foram conduzidos em disciplinas de primeiro semestre por aproximadamente dois meses para que possamos deduzir pequenas generalizações e termos tempo para tentar, durante o próprio ciclo, fazer as mudanças para que a aprendizagem e diversão pudessem ser aperfeiçoadas. A natureza da investigação conduzida nessa tese tentou entregar um produto usável aos alunos, que exigiu a robustez de suportar acessos concorrentes, minimizar falhas e erros de programação do próprio jogo, e ainda ser divertido e ter um senso de utilidade para que os alunos pudessem retornar ao jogo. Essa secção

aborda algumas dificuldades que tivemos ao conduzir a investigação durante esses dois anos.

1.1-O envolvimento do professor da disciplina é fundamental para uma boa utilização do jogo. Durante os três últimos ciclos, os professores não estavam ligados diretamente no projeto do jogo e com isso não tinham completo domínio sobre ele. Eles tinham uma visão genérica e superficial sobre o jogo. Essa falta de contato interferiu na própria motivação dos alunos ao observarem que o professor desconhecia o objeto daquilo que sugeria que eles usassem para praticar e aprimorar os seus conhecimentos sobre programação. Em todos os ciclos tivemos que manter o monitoramento da evolução dos alunos e o contato por troca de mensagens de correio eletrónico para manutenção da motivação. Essa conjetura corrobora naquilo que Brown (1992) e Squire (2005a) já advertiram sobre a necessidade do envolvimento dos professores para reforçar o contexto da pesquisa em sala promovendo os testes com os alunos.

1.2-Monitoramento frequente para intervir rapidamente. Erros não são tolerados mesmo num projeto de investigação. Em dois momentos ficou evidente a necessidade de rápida manutenção no jogo: o primeiro momento foi durante o segundo ciclo em relação à forma de ganhar pontos, e o segundo momento foi no terceiro ciclo, devido ao erro de enunciado em uma missão. No segundo ciclo já vinha sendo advertido que a contagem do tempo incomodava os alunos durante a resolução da missão. A primeira forma que adotamos para resolver o problema foi duplicar os tempos limites para dar uma folga maior aos alunos. No fim demonstrou ser a própria contagem do tempo, independente da quantidade, o fator que influenciava na concentração dos alunos. Ao modificamos para os pontos por tentativa, já tínhamos perdido boa parte da turma, ou seja, a frustração experimentada impediu-os de retornarem ao jogo. No terceiro ciclo a última missão da fase 3 tinha um erro no enunciado que conduzia os alunos a resolverem a missão de uma forma que não casava com os objetivos. O erro foi corrigido e explicado aos alunos, onde muitos deles simplesmente tiveram que executar a solução sem necessidade de modificar nada, pois já estava correta. Isso causou um sentimento de incredulidade que acabou afastando alguns alunos. Nesse caso agimos com maior rapidez, porém, o alto índice de tentativas e tempo gasto na missão sem alcançar o sucesso, acabou por frustrar esses alunos.

1.3-Participação ativa dos alunos na investigação. Enquanto no segundo ciclo estávamos em contato presencial com os alunos, no terceiro e quarto o contacto foi unicamente por correio eletrónico, pois estávamos em continentes distantes. Essa intervenção teve que ser conduzida dessa forma pois em Portugal a entrada de alunos acontece uma vez por ano, e no segundo ciclo aproveitámos essa entrada. Com as mudanças no jogo, e para obtermos resultados mais breves, optamos pelos ciclos seguintes acontecerem no Brasil. No segundo ciclo tivemos contato pelo menos semanalmente, inclusive em “conversas de corredor”, com os alunos envolvidos, que puderam nos sugerir pequenas melhorias ou modificações no jogo. Apesar do contato por meio eletrónico ser rotineiro com os alunos dos ciclos no Brasil, a natureza da conversa escrita e a falta de proximidade, nos limitaram quanto ao relacionamento ativo com os alunos.

5.2.2 Aplicação do Jogo

Nos três últimos ciclos foram tentadas três formas distintas de uso do jogo, respetivamente, (1) jogar opcionalmente, (2) o assunto do jogo citado nas aulas com a conclusão das fases contribuindo para a nota do exame, e (3) jogar faz parte do currículo. Nos próximos parágrafos apresentamos, com base no trabalho realizado durante os ciclos de desenvolvimento do jogo, as nossas conclusões sobre o momento e a melhor forma de conduzir cada situação considerando que os professores tenham conhecimento do jogo e que este dê suporte à forma de aplicá-lo nesses contextos.

2.1-Jogar opcionalmente e desvinculado das aulas. Essa é uma forma de usar o jogo quando o professor não pode alterar a sua metodologia pedagógica por questões institucionais, ou acredita que já tem uma boa estratégia pedagógica, adotando o jogo como forma de apoiar os alunos com maior dificuldade. Mesmo sendo opcionais, as dez primeiras missões, equivalente à fase 2, foram concluídas por praticamente metade da população no segundo ciclo. As conversas com os alunos revelaram que havia algumas pequenas comunidades de alunos que discutiam, se ajudavam e até competiam entre si. Sabemos que o jogo nesse ciclo ajudou alguns alunos a melhor compreenderem o início de um novo assunto. Naturalmente, quando o professor disponibiliza uma lista de exercícios e sugere que os alunos os resolvam, ele também não tem como obrigá-los a cumprir, a não ser que essas listas sejam consideradas instrumentos de avaliação que contribuam na nota final da disciplina.

Uma forma de usar o jogo sem ligação direta às aulas é sugerir que os alunos pratiquem um conjunto de missões ou fases para se prepararem para um determinado assunto. Isso pode ser sugerido pontualmente aos alunos que o professor sente terem mais dificuldade em resolver os exercícios. O professor pode acompanhar no ambiente administrativo as falhas e a evolução dos alunos e assim identificar os que necessitam de intervenção.

2.2-Jogo ou a sua abordagem usada nas aulas. Esse modo serve para as situações em que o professor pode e está interessado em introduzir mudanças nas suas estratégias, como forma de complementar as aulas. O professor estimula a curiosidade dos alunos ao iniciar os assuntos usando a representação de blocos para explicar o significado das estruturas, em vez de apresentar fluxogramas ou usar diretamente uma linguagem de programação. Os blocos podem igualmente ser usados para praticar alguns exercícios adicionais. Tal como sugerido na situação anterior, o professor pode sugerir um conjunto de missões ou fases que antecede um dado assunto. O objetivo é entender o funcionamento das estruturas de programação independentemente da linguagem de programação. No momento seguinte, o professor pode apresentar os mesmos exemplos e exercícios com a linguagem de programação.

2.3-Conclusão das fases contribui para a avaliação da disciplina. Essa opção serve aos professores e instituições que têm a liberdade de adicionar novos tipos de instrumentos de avaliação. Nos modos anteriores, os alunos podiam ser motivados pelo professor ou por si próprios, ao detetarem a necessidade de superar suas dificuldades na aprendizagem. Infelizmente, isso nem sempre é o suficiente. Um meio adotado pelo professor no terceiro ciclo foi atribuir pontos adicionais nos exames aos alunos que finalizaram, até uma data limite, um conjunto de fases relacionadas com o assunto do exame. Isso ajudou a dar ritmo inicial aos alunos, uma vez que cerca de 55% dos que começaram o jogo ganharam as recompensas até à quarta fase, e 22% até à quinta fase. Entretanto, por ser uma atividade não acompanhada presencialmente pelo professor, pode promover dois comportamentos não desejáveis: (1) o aluno prover o seu acesso a outro para que esse conclua as missões; (2) receber a solução como imagem capturada e assim copiar a solução de um colega. Por isso, consideramos que a pontuação adicional deve ter um pequeno impacto no global da avaliação da disciplina. O jogo somente permite copiar e colar blocos dentro da própria missão. Isso exige que o aluno que queira utilizar a solução de outro aluno, tenha que

a construir, inserindo os blocos e mudando seus parâmetros. Uma alternativa seria a liberação de missões especiais durante a aula presencial, em que os alunos que as concluíssem receberiam essa pontuação extra. Reduziria as hipóteses de cópia, e ainda assim exigiria do aluno uma prática prévia no jogo para que ele tivesse um bom desempenho nessa missão especial.

2.4-Jogar faz parte do currículo. Uma das justificativas dos alunos para não adotarem ou abandonarem o jogo são as obrigações que a própria disciplina exige com as listas de exercícios que é necessário resolver. Esse modo foi usado no último ciclo e é sugerido para os professores que têm total liberdade na definição das estratégias pedagógicas e modelos de avaliação. O professor inclusive não precisou adicionar carga horária para que o programa fosse totalmente cumprido. Nesse ciclo os alunos tinham três semanas para jogar. Não havia aulas presenciais da disciplina, ou seja, podiam usar essa carga horária para jogar, e por isso nem tinham tarefas extras, o que permitia ainda se dedicarem extra classe. A verdade é que o professor também não esteve disponível. Logo, apesar de contarem com a ajuda de um monitor e da intervenção do investigador, os alunos precisavam aprender com autonomia. Após esse período, foi aplicada uma avaliação usando apenas os conceitos aprendidos no jogo. No nosso estudo, encontrámos uma correlação moderada do desempenho acadêmico com a quantidade de missões concluídas. Isso indicia que o jogo contribuiu para o desempenho dos alunos. Nas semanas seguintes, após o primeiro exame, o professor mostrou a equivalência das estruturas do jogo com uma linguagem de programação. Apesar de não termos realizado o mesmo formato da experiência com a presença do professor nas aulas, mesmo que essas sejam dedicadas ao suporte ao jogo, estamos convictos de que a sua participação contribua significativamente para a aprendizagem dos alunos.

5.2.3 Jogo Casual Sério

Nessa secção vamos apresentar uma série de observações sobre as características do jogo que promoveram, ou que podem promover, aprendizagem e diversão, conforme a análise dos resultados dos quatro ciclos realizados.

3.1-Usar a quantidade de tentativas em vez de restringir pelo tempo. Essa questão foi colocada em primeiro lugar, uma vez que acreditamos ter sido uma das maiores lições aprendidas na investigação. A resolução de problemas com

programação exige do aluno concentração em vez de raciocínio rápido. O cronômetro faz com que o aluno perca a calma e experimente o seu programa várias vezes, sem necessariamente ter feito uma análise minuciosa. Já quando se usa a atribuição de pontos por quantidade de tentativas, o aluno tem todo o tempo disponível. Para evitar perca de pontos, ele deve avaliar bem o seu programa antes de começar a executar. A mudança de um modo para outro inclusive fez com que os alunos deixassem de sugerir a utilização de codificação em vez de blocos, o que nos leva a considerar que essa necessidade de codificar identificada no segundo ciclo emergiu de procurarem ser mais rápidos a produzir a solução.

3.2-Ganhar pontos influencia o comportamento dos alunos. Todo jogo precisa de uma forma de recompensar o jogador. Isso acontece normalmente através da conquista de pontos. Durante os nossos estudos verificaram-se comportamentos que emergiram pela vontade dos alunos ganharem a maior quantidade de pontos. Por exemplo, no segundo ciclo, dado que estavam a jogar contra o tempo, alguns alunos resolveram primeiro a missão com papel e caneta, para depois entrarem rapidamente a solução no jogo. No terceiro ciclo, o jogo permitia que o aluno se autenticasse mais de uma vez simultaneamente, o que fez com que alguns alunos numa sessão tentassem resolver a missão, mas paravam pouco antes de a concluir, para então na outra sessão refazerem a solução e obterem a pontuação máxima. Esse problema foi resolvido na versão do quarto ciclo. De qualquer forma, isso ilustra como a existência de pontuação impulsionou os alunos a criarem estratégias para superar os desafios. Além disso, apesar dos alunos mencionarem que jogaram pela aprendizagem, estamos certos de que sem a conquista de pontuação seria provável a menor adesão dos alunos. Inclusive, no quarto ciclo, com o uso do NoBugsJ, os alunos sentiram a ausência de um sistema de pontuação. Mas, os alunos precisaram de um estímulo para usar e exibir esses pontos acumulados, como se verá de seguida.

3.3-Personalização do avatar. Inicialmente, a personagem que representava os alunos começava com a mesma aparência. À medida que iam conquistando pontos, novas opções iam sendo liberadas (cor dos olhos, cabelos, barbas, cor e estilo do dólmã e do chapéu, e acessórios). Isso ofereceu um sentimento de progresso ao aluno, uma vez que tinha a possibilidade de evoluir a aparência do seu avatar. Logo, uma meta inicial dos alunos foi acumular pontos para configurar a personagem, e por isso, solicitaram mais opções. Com o avanço no jogo, o interesse com a diversão na

personalização do seu avatar foi direcionado para a comparação com os avatares dos outros jogadores disponíveis na tabela de classificação.

3.4-Tabela de classificação. No último ciclo, quando pudemos rastrear todas as ações dos jogadores, concluímos que a tabela de classificação foi o recurso mais acedido pelos alunos, inclusivamente por aqueles com pior desempenho acadêmico. Mesmo nos ciclos anteriores, através das opiniões relativas à diversão percecionada, os alunos dos grupos que menos jogaram elegeram esse recurso como o mais motivador. Apesar do jogo não prover nenhum recurso de competição direta entre os jogadores, essa comparação estimula a sua busca por fazer mais e melhor nas missões seguintes. O jogo oferece três tipos de tabela de classificação: pela pontuação total, pela quantidade de tentativas, e pelo tempo gasto. Um aluno não necessariamente pode estar na mesma posição em todas as tabelas. Eles usam essas diferentes classificações para constatar em que aspeto precisam melhorar. Os alunos em pior posição solicitaram a revisão dessa tabela para que eles não aparecessem. Com isso, a tabela apresenta a todos os alunos os dez melhores classificados. Caso o aluno não esteja entre esses dez, a sua classificação aparece apenas no seu painel de controlo.

3.5-Suporte. Em todas as avaliações realizadas ao longo do desenvolvimento, o suporte às dificuldades dos alunos foi sempre apontado como a maior fraqueza do jogo. O jogo não tem um sistema de inteligência computacional para monitorar e apoiar o aluno conforme o contexto. Uma das ações que adotámos foi acompanhar manualmente a evolução dos alunos, identificar as suas dificuldades e enviar mensagens de correio eletrónico com a ajuda. Em seguida, toda essa ajuda foi incorporada no jogo. O sistema para tentar identificar o problema do aluno era muito elementar, baseado em verificar o estado atual com condicionais e apresentar uma mensagem de texto. Isso não mostrou ser o suficiente. Como o foco dessa investigação não foi desenvolver um sistema inteligente de suporte, tentamos minimizar o problema usando outras abordagens que pudessem evitar que o aluno ficasse paralisado em uma missão:

3.5.1-Acesso e interação com as missões já concluídas: foi uma solicitação recebida durante o segundo ciclo. Os alunos não só podem consultar as missões que já concluíram, como podem interagir com elas, seja executando, depurando ou até alterando a solução, mesmo que não interfira mais na pontuação. No quarto ciclo pudemos constatar que no grupo de alunos com melhor desempenho foi mais

evidente esse comportamento de aceder e interagir com essas missões já cumpridas. Voltar a jogar uma missão serve tanto para rever algum conceito, como para testar algo que poderia fazer o aluno perder pontos na missão em aberto;

3.5.2-Direcionamento na disponibilidade de blocos: essa foi outra sugestão já percebida logo no ciclo 2, onde os blocos estavam disponíveis para as missões, mesmo quando não eram necessários. Após revisão das missões para o terceiro ciclo, os blocos disponíveis nos menus passaram a ser somente os necessários para resolver a missão. Apesar de podermos estar restringindo o processo de criação dos alunos, mantivemos essa abordagem no intuito do jogo servir para instruir o que é melhor para o tipo de problema em específico tratado na missão. Depois, quando o aluno transferir esse conhecimento para a linguagem de programação, é que ele teria a necessidade de aprender a adaptar o tipo de solução conforme o problema;

3.5.3-Assistente de pseudocódigo: essa melhoria foi usada nos terceiro e quarto ciclos. Após pagar o custo da ajuda (uma moeda ou todas as estrelas da missão) o jogo mostra a solução da missão em pseudocódigo. Aos alunos cabe a tarefa de interpretar o pseudocódigo e construir a solução, ou seja, inserir e arrastar os blocos, e mudar seus parâmetros. Num primeiro momento pode-se pensar que esse tipo de ajuda é o mesmo que mostrar a própria solução, mas não foi o que os dados mostraram. As medianas do número de tentativas entre o momento em que o assistente foi usado e a conclusão da missão foram 3 e 2 tentativas, nos ciclos 3 e 4, respetivamente. Isso mostra que os alunos não alcançavam a solução diretamente com a leitura do pseudocódigo, e necessitavam ainda de algumas tentativas para concluir a missão;

3.5.4-Livre escolha das missões a serem jogadas: a partir do terceiro ciclo propusemos uma estrutura de fases menos rígida. A partir do quarto ciclo foi testada uma alternativa ainda com mais liberdade, pois permitia aos alunos escolher jogar quaisquer das fases, e dentro de cada uma delas podiam jogar quaisquer das missões. A intenção foi fornecer mais uma forma de evitar que o aluno ficasse bloqueado em alguma missão. Podia se supor que esse tipo de artifício levasse o aluno a não seguir a sequência de aprendizagem sugerida pelo jogo, e por fim, deturpar o projeto instrucional pensado para sua evolução. No entanto, comparando a sequência de aprendizagem dos alunos entre os ciclos 3 e 4, verificámos que aqueles com mais liberdade de escolha foram mais regulares na ordem de resolver as missões entre as fases. Os alunos com essa livre escolha primaram por resolver

primeiro o conjunto de missões dentro da fase, mesmo que a ordem não tenha sido obedecida. Os alunos procuravam avançar de acordo com o projeto instrucional, mas para evitar ficar parados no jogo, experimentavam jogar outras missões, e algumas delas puderam ensiná-los algo que faltava para que eles cumprissem as anteriores.

3.6-Áreas bloqueadas: o painel de controlo tinha algumas áreas que seriam desbloqueadas quando o aluno alcançasse uma quantidade de pontos. Essas áreas diziam respeito à montagem de uma esplanada pelo próprio aluno. Entretanto, elas não foram desenvolvidas, e inclusive era impossível alcançar os pontos sugeridos para abertura dessas áreas. Contudo, os alunos demonstraram motivação em conquistar os pontos e concluir as missões visando alcançar o desbloqueio dessas áreas. Podemos considerar que esse tipo de artifício pode ser recomendado para manter a adesão dos alunos ao jogo.

3.7-Sistema de Conquistas: a partir do terceiro ciclo foi implantando um sistema de conquistas, que diziam respeito às recompensas recebidas quando os alunos atingiam objetivos marginais, como por exemplo, finalizar uma fase até determinada data. Naquele ciclo os alunos foram recompensados com pontos extra nos instrumentos de avaliação da disciplina. No quarto ciclo os alunos ganhavam moedas que podiam ser usadas para contratar o assistente de pseudocódigo e habilitar o uso de objetos (como facas e panelas) na personalização do avatar. Naquele ciclo, o grupo com melhor desempenho acadêmico teve o sistema de conquistas mais acedido que a personalização do avatar. Isso pode demonstrar mais um recurso que estimula a motivação dos alunos.

3.8-Domínio cognitivo e tarefas: durante os três últimos ciclos a investigação aprimorou na definição de diferentes tipos de atividades conforme o estágio na aprendizagem de um novo conceito. Os tipos de atividades estavam relacionados com os domínios cognitivos da Taxonomia de Bloom. Os primeiros estágios envolvem tarefas mais simples, como determinar a saída produzida pela execução de um programa, para em seguida fazer pequenas correções numa solução com erros, e assim por diante, até chegar nas tarefas para desenvolver uma solução do zero. Essa sequência visa apresentar primeiro aos alunos programas funcionais, e boas práticas, para apenas depois dele ter visto esses exemplos, o jogo possa então solicitar ao aluno que crie sua própria solução.

3.9-Área do professor usando análises simples de aprendizagem: os recursos de administração foram subutilizados pelos professores, conforme já

justificado na secção 5.2.1. Porém, eles foram intensamente utilizados pelo investigador para monitorar a evolução dos alunos e fornecer-lhes suporte. A última versão permitiu que tivéssemos uma visão dos problemas de cada missão, inclusive de inspecionar cada tentativa para procurarmos padrões nos erros e identificar soluções possíveis no projeto instrucional. Através de análises simples, como quantidade de tentativas, tempo gasto, acesso à janela de explicação, foi possível desenvolver uma ferramenta em o professor pode rapidamente verificar os alunos que exigem atenção, e o tipo de cuidado que eles precisam receber. É primordial que os jogos sérios consigam fornecer essa informação para aumentar a sua utilidade como ferramenta educacional.

5.3 Arquitetura de Motor de Jogos Casuais para a Aprendizagem de Programação

O jogo foi concebido para que as missões e a sequência de aprendizagem fossem customizadas pelo professor. Na investigação não produzimos as ferramentas e editores para essas customizações, porém o jogo acede a um banco de dados onde estão as missões e sequências, interpreta essa informação e apresenta ao aluno. Essa capacidade de reutilizar o NoBug's SnackBar em vários contextos é o que caracteriza ele como um motor de jogos. Essa secção, relacionada com o artigo publicado a seguir, aborda alguns aspetos da implementação que podem vir a orientar e inspirar outros projetistas de jogos com enfoque em aprendizagem construcionista:

- Vahldick, A., Mendes, A., & Marcelino, M. (2016). Towards a Constructionist Serious Game Engine. In *17th International Conference on Computer Systems and Technologies*. Palermo, Italia.

5.3.1 Modelo de Motor Construcionista de Jogos Sérios

Generalizámos a arquitetura do jogo para a especificação de um modelo de motor de jogos sérios chamado de "*Construcionist Serious Game Engine"* (CSGE). A arquitetura desse motor está ilustrada na Figura 5.1 onde são apresentados os componentes essenciais e suas relações para criar jogos que promovam a

aprendizagem de um domínio através da abordagem construcionista. Existem três elementos externos integrados ao CSGE: (1) Aluno que interage com o jogo através da Interface com o Usuário (IU); (2) Modelo do Aluno que tem armazenado o perfil do estudante: as missões concluídas, os pontos ganhos, o tempo e as tentativas gastas para finalizar cada missão, estilo de aprendizagem, etc; e (3) Modelo de Domínio, que contém o conteúdo, sequência do curso e das tarefas de aprendizagem (missões). O diagrama adota quatro notações visuais para representar cada tipo de componente e suas relações. Linhas representam as conexões entre os componentes. Elipses simbolizam os dados gerados pelas atividades e acedidos ou alterados por outros. Retângulos duplos arredondados representam a IU. Por ser um modelo, a especificação é livre da tecnologia a ser adotada para IU (web, mobile, gráfico, etc.), para o armazenamento dos dados (relacional, XML, binário, etc.), e inclusive a representação das soluções (blocos, textos, fluxogramas, etc.). Os componentes em branco representam todos aqueles que não dependem da abordagem de aprendizagem ou das customizações do professor. Os componentes em verde representam as partes em que o instrutor pode especificar ou personalizar por exemplo, conforme o seu estilo de ensino ou o perfil da turma. Esses componentes são as missões e seu conteúdo, e a ordem de apresentação dessas missões. Eles também podem ter mecanismos de adaptação para personalizar o ensino, alterando as tarefas das missões ou como acontece a interação do jogo com o aluno, dosando a quantidade de textos explicativos, gráficos, esquemas e imagens baseado no estilo de aprendizagem, o conhecimento prévio e o progresso do aluno no jogo. Os componentes em vermelho relacionam-se com os elementos e comportamentos construcionistas do jogo.

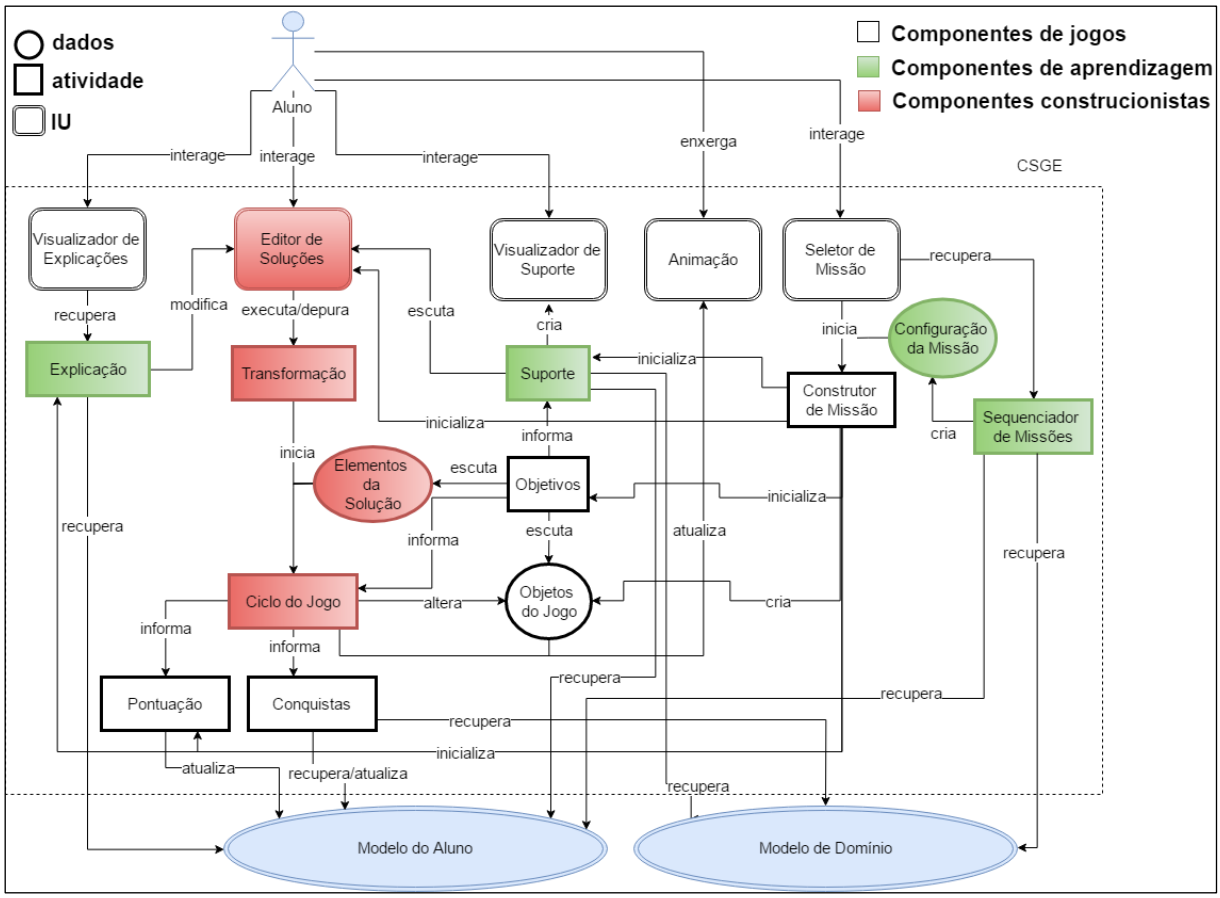


Figura 5.1 – Arquitetura do CSGE

O **Seletor de Missões** mostra as missões realizadas e disponíveis. Ele recupera essas informações do componente Sequenciador de Missões. De acordo com a escolha do usuário, ele inicia o Construtor de Missão passando a Configuração da Missão. A **Configuração da Missão** representa cada missão disponível no jogo contendo todas as informações sobre ela: explicação do conteúdo de aprendizagem e descrição da missão; regras, restrições e definição de objetivos; mundo e sua configuração de objetos; e assim por diante. O **Construtor de Missão** é responsável por inicializar o mundo do jogo. Ele cria os Objetos do Jogo e inicializa os componentes Suporte, Objetivos, Explicação, Pontuação e Editor de Soluções. O **Sequenciador de Missões** carrega a sequência de aprendizagem do Modelo de Domínio para identificar a ordem da missão. Ele também verifica as missões já realizadas pelo aluno, recuperando essa informação do Modelo do Aluno e calcula quais as missões que estão disponíveis para jogar.

O usuário desenvolve a solução dos seus problemas no **Editor de Soluções**. Quando o usuário decide testar (executar ou depurar) a solução, o componente **Transformação** converte a representação da solução para alguma estrutura interna

(**Elementos da Solução**) gerida pelo Ciclo do Jogo e inicia a tentativa. O **Ciclo do Jogo** é responsável por iterar pelos Elementos da Solução, modificar o estado dos Objetos do Jogo e solicitar a atualização da área de Animação. Os **Objetos do Jogo** referem-se aos elementos que compõem os desafios das missões. A **Animação** é uma área onde os estudantes podem observar a solução modificando o mundo do jogo. O componente **Objetivos** tem a responsabilidade de verificar se os objetivos foram atingidos. Os Elementos da Solução e os Objetos do Jogo têm registado o componente Objetivos como ouvinte. Isso implica que, quando alguma coisa modifica aqueles componentes, o componente Objetivos é informado das mudanças. A tarefa do componente Objetivos é comparar as restrições do Elementos de Solução e o estado dos Objetos do Jogo com a Configuração da Missão. Se a tentativa teve sucesso (cumpriu todos os objetivos), o Ciclo do Jogo informa os componentes de Pontuação e de Conquistas. O componente **Pontuação** controla os pontos, experiência ou moedas a serem recebidos e atualiza o Modelo do Aluno. O componente **Conquistas** verifica os prêmios que não são específicos da missão, mas do jogo como um todo. Por exemplo, ser o primeiro a finalizar um conjunto de missões, ser o mais rápido ou que teve menos tentativas, ou resolver uma fase até determinada data. Ele busca os dados dos Modelos do Aluno e de Domínio. Os prêmios conquistados pelos alunos são atualizados no Modelo do Aluno.

O componente **Suporte** é responsável por prover as dicas e ajuda aos estudantes em dois momentos: enquanto está a desenvolver a solução e quando a solução falha após ou durante a execução. O Editor tem esse componente registado como ouvinte, o que implica que o Suporte atua quando o aluno está ocioso ou com problemas. Nesses momentos o componente pode recomendar alguma ação. Quando uma solução submetida não resolve a missão, esse componente pode fazer alguma sugestão para resolver o problema. O **Visualizador de Suporte** apresenta o conteúdo desses comentários. Esse conteúdo leva em consideração as características específicas da missão fornecidas pelo Construtor de Missão, os Modelos do Aluno e de Domínio.

O componente **Explicação** disponibiliza o conteúdo instrucional, descreve as tarefas das missões e dá algum suporte para compreensão da missão e identificação de alguma informação inicial importante para o aluno iniciar sua solução. O **Visualizador de Explicações** apresenta essa informação.

5.3.2 Instanciação do Modelo

O modelo é uma generalização do jogo. Porém, apresentamos nessa secção alguns detalhes da implementação do jogo e mostramos como o jogo realizou o modelo. A Figura 5.2 ilustra a arquitetura cliente-servidor seguida pelo jogo. O lado cliente representa o jogo que o aluno vê. Ele é executado em navegadores web e desenvolvido com JavaScript e HTML5. O lado servidor é desenvolvido com tecnologias Java. O lado cliente interage com o servidor através de uma classe chamada UserControl usando tecnologia AJAX. Toda a informação usada pelo lado cliente é recuperada e armazenada solicitando serviços à classe UserControl. O Modelo do Aluno e o Modelo de Domínio estão guardados em um banco de dados. O Modelo de Aluno inclui todas as tentativas de solução que o aluno efetuou nas várias missões. O Modelo de Domínio é a especificação de cada missão e a sequência de exibição dessas missões. Os componentes Sequenciador de Missões e Conquistas foram implementados no lado servidor, pois eles não têm como função atualizar os componentes de IU.

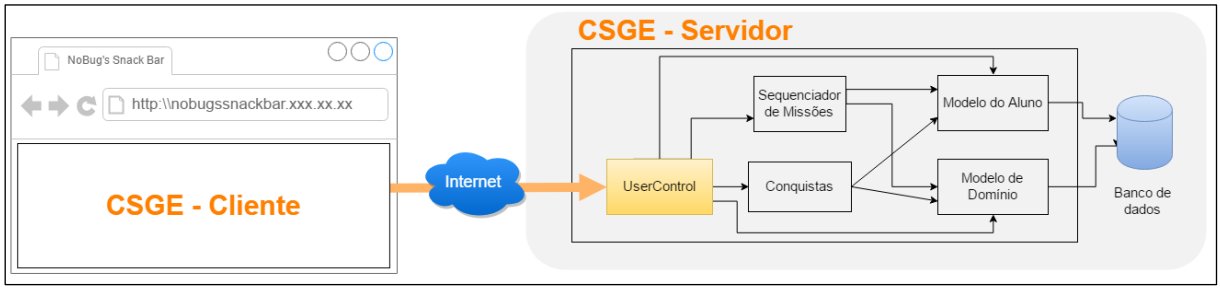


Figura 5.2 – Arquitetura do Jogo

A Figura 5.3 apresenta todos os artefactos – ficheiros e classes dentro deles – do componente CSGE-Cliente. As linhas contínuas com setas representam associações entre as classes, com triângulo indica uma relação de herança e tracejadas são as relações de uso. Referente à edição e execução da solução, a Figura 5.4 destaca os componentes do modelo e suas materializações, através de um retângulo no lado superior direito do componente é relacionado o seu artefacto. O **Editor de Soluções** se refere à área central do ambiente do jogo (Figura 4.47) e **Animação** está relacionado com o lado esquerdo do ambiente, onde é apresentada a esplanada e o desenho da personagem e dos clientes.

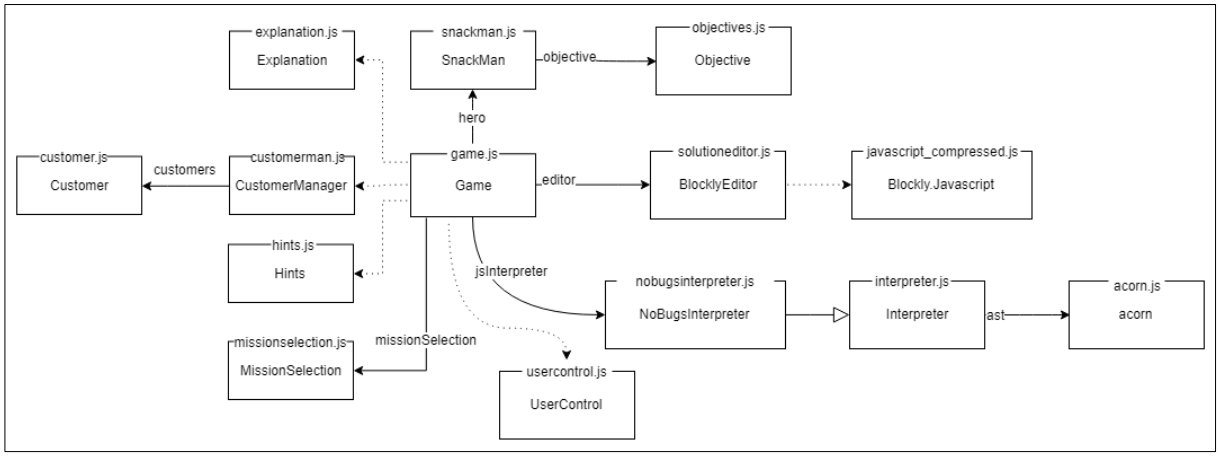


Figura 5.3 – Artefactos do Componente CSGE-Cliente

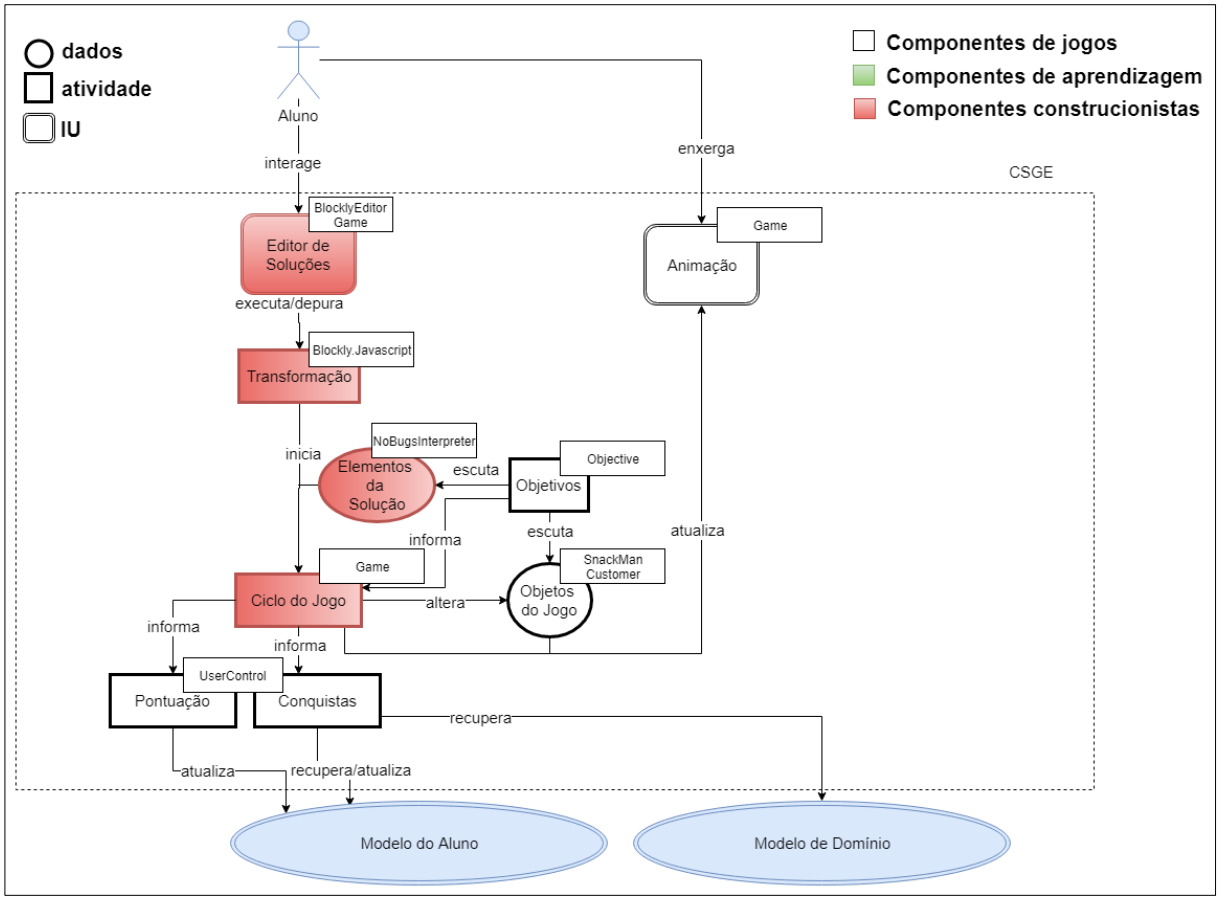


Figura 5.4 – Componentes de Edição e Execução da Solução

O fluxograma do ciclo do jogo é ilustrado na Figura 5.5. Assim que o aluno lança a execução, é feita uma verificação na solução (Analisar sintaxe) para verificar se existem entradas vazias nos blocos (exemplo da Figura 5.6), situação que gera uma exceção. No passo seguinte, a solução em blocos fornecida pelo aluno é transformada para código JavaScript (Gerar código JS) para em seguida dividi-lo em uma Árvore de Sintaxe Abstrata (Gerar AST). Juntos esses dois processos são a

materialização do componente **Transformação**. Cada folha dessa árvore representa uma parte de cada expressão de uma linha de código, por exemplo, uma operação matemática, uma chamada ou retorno de uma função, etc. Enquanto existirem folhas acontece um ciclo de execução do comando e sua animação. Quando se tratar de comandos da própria linguagem, por exemplo, repetição, condição, atribuição de variáveis, operações matemáticas, etc., então é tratada a expressão. Se for um comando da personagem, por exemplo *goToBarCounter(1)* ou *askForDrink()*, é gerado um conjunto de informações (**Elementos da Solução**) para ser criada uma lista de comandos para animação e atualizado o estado dos **Objetos do Jogo** (personagem e clientes). Cada comando de animação representa uma imagem da personagem e sua posição na área de animação. Enquanto houver esses comandos, a personagem e os clientes são animados. Quando não houver mais folhas na AST, implica que o último comando da solução foi executado, e o jogo verifica se todos os objetivos foram conquistados. Em caso positivo, são registados os pontos de experiência e moedas ganhas na missão. Em caso negativo, é apresentada uma mensagem de erro na tela. Existem vários tipos de objetivos, por exemplo, entregar o pedido desejado do cliente, falar um determinado valor, restringir a quantidade de variáveis ou a quantidade de vezes que um determinado comando pode ser executado, obrigatoriamente usar uma estrutura *for* ou *while*, etc. Por essa razão o componente **Objetivos** atua sobre os **Objetos do Jogo** para, por exemplo, conferir se o cliente está satisfeito, ou se o que foi falado corresponde ao esperado, e sobre os **Elementos da Solução**, para conferir se a quantidade de variáveis está de acordo, ou se a quantidade de vezes que um comando é disparado corresponde ao limite.

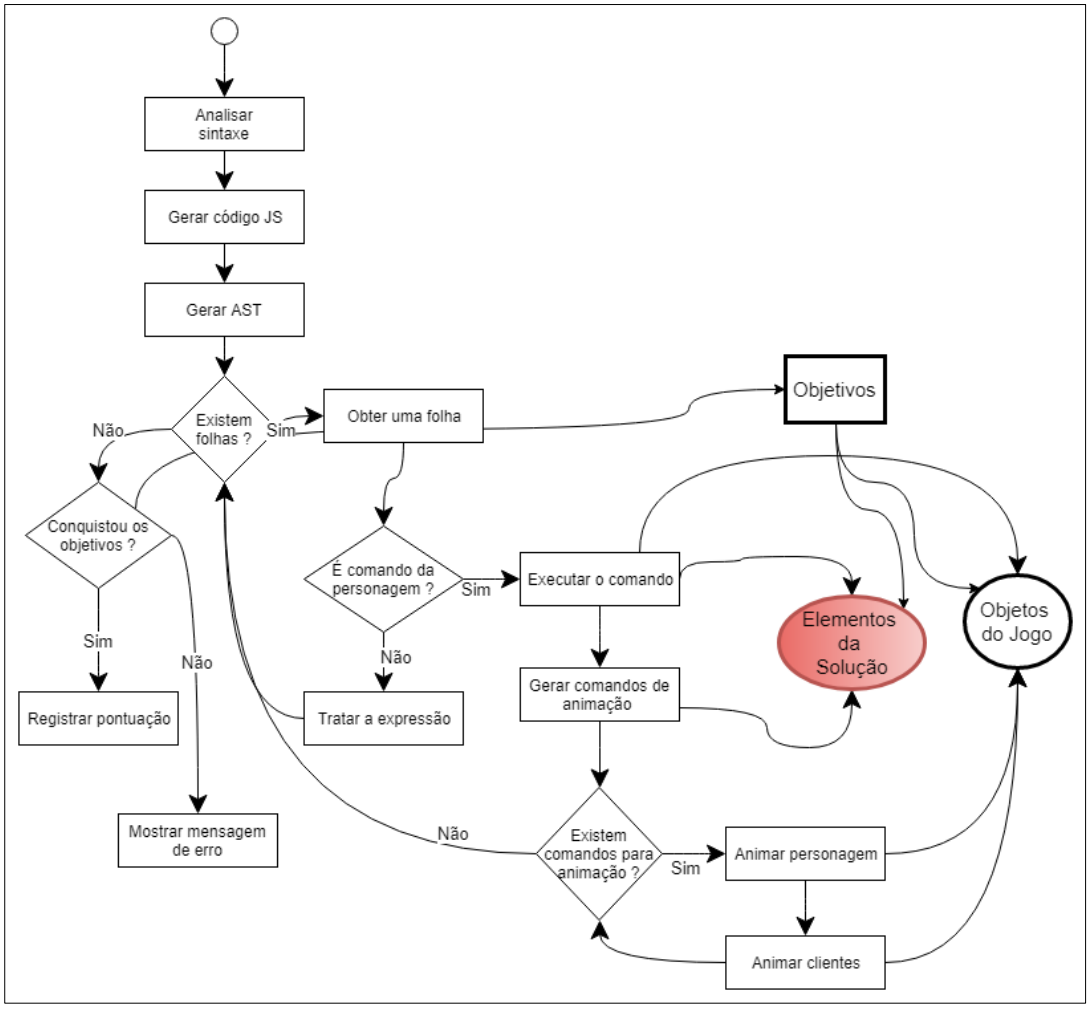


Figura 5.5 – Fluxograma do Ciclo de Jogo

Figura 5.6 – Exemplo de entradas vazias em blocos

A Figura 5.7 apresenta os componentes das funcionalidades de visualização das explicações, de suporte e de seleção de missões. O artefacto *MissionSelection* constrói a janela de seleção da missão (Figura 4.66) com base na **Configuração da Missão** fornecida pelo **Sequenciador de Missões**. Esse sequenciador acede os modelos de aluno e de domínio para a montagem da configuração. O sequenciador é uma classe que está no servidor e cria um objeto da configuração da missão. Após o aluno selecionar a missão, o **Construtor de Missão**, que é um conjunto de métodos da classe Game, inicializa o **Suporte**, **Editor de Soluções**, **Objetivos**, o verificador da **Pontuação** e a **Explicação**. O artefacto *Explanation* é responsável por construir a janela e gerir a sequência de exibição da explicação da missão (Figura 4.48). O

artefacto *Hints* verifica constantemente o que o aluno está a produzir e apresenta uma mensagem caso atenda alguma condição de disparo, por exemplo, quando o aluno fica inativo por algum tempo, quando ocorreu um erro durante a execução em determinado bloco, ou quando o aluno excedeu o limite da quantidade de blocos ou de variáveis esperadas para a missão.

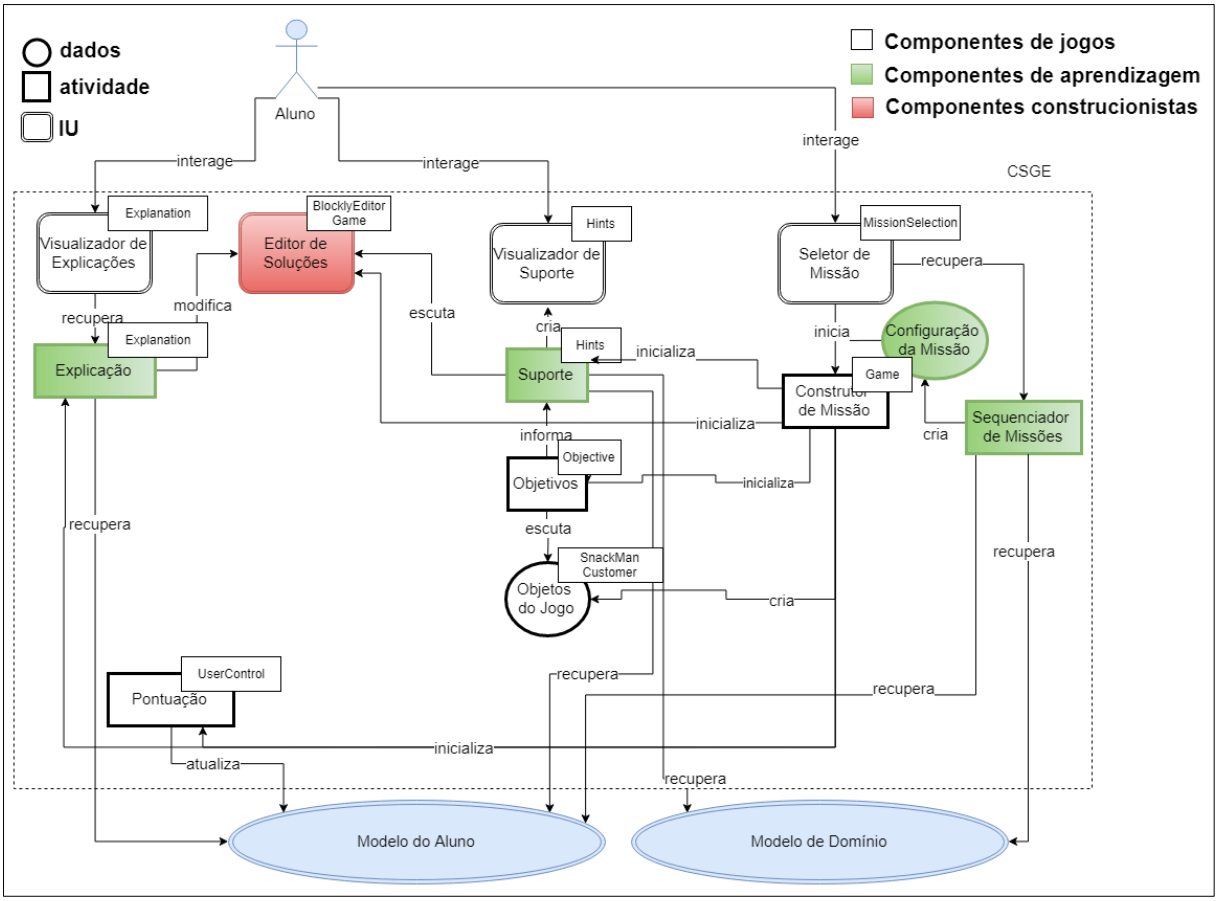


Figura 5.7 – Componentes de explicações, suporte e seleção de missões

As missões são mapeadas usando o formato eXtensible Markup Language (XML). A Tabela 5.1 relaciona o conteúdo da missão, com o componente e o elemento XML.

Tabela 5.1 – Mapeamento dos componentes do CSGE e XML

Descrição	CSGE	Elemento XML
Conteúdo de aprendizagem e a descrição da tarefa	Explicação	<explanation>
Regras que definem quando uma dica é mostrada	Suporte	<hints>
Assistente de pseudocódigo	Suporte	<help>
Comandos disponíveis para criar a solução	Editor de soluções	<commands>
Configuração dos clientes e seus desejos	Objetos do Jogo	<customers>
Objetivos, restrições do jogo e pontuação	Objetivos e Pontuação	<objectives>
Blocos iniciais sugeridos na solução	Editor de soluções	<xml>

O elemento <explanation> contém um conjunto de elementos <page> representando cada página de explicação.

```
<explanation>
  <page> Essa é a página 1 da explicação. </page>
  <page> Essa é a última página. </page>
</explanation>
```

O elemento <hints> contém dois elementos: <sequence> com o conjunto de dicas e sugestões a serem apresentados durante a produção da solução, e <errors> com o conjunto de dicas e sugestões a serem apresentados quando acontece algum erro na execução ou a execução finalizou sem conquistar todos os seus objetivos. Independente das situações, toda dica ou sugestão é definida pelo elemento <hint>. Esse elemento possui os seguintes atributos:

- **category:** especifica o tipo de comportamento da dica. As categorias implementadas são:
    - ChooseCategory(int): caso o aluno esteja fazendo nenhuma ação aparece uma sugestão apontando para o item do menu que ele deveria clicar;
    - SelectCommand(int, int): caso o aluno tenha clicado no item do menu informado no primeiro parâmetro vai aparecer uma sugestão apontando para o comando a ser inserido na solução;
    - StackTogetherTwo: caso existam apenas dois blocos separados na solução o jogo vai sugerir juntá-los;
    - RunProgram: se o aluno ainda não tentou nenhuma execução, o jogo sugere clicar no botão para iniciar a execução;
    - DebugProgram: se o aluno ainda não tentou nenhuma depuração, o jogo sugere clicar no botão para iniciar a depuração;
    - GoalButton: se foram feitas mais de duas tentativas, o jogo sugere ao aluno consultar os objetivos da missão;
    - SourceCode: abre a janela de sugestão exatamente ao lado do bloco em que foi interrompida a execução;
    - VariableWindow: aponta uma sugestão ao lado da janela de variáveis do lado direito do ambiente que é mostrada quando se está depurando;
    - TestBlock: adiciona um triângulo no bloco (como na Figura 5.6) que ao ser clicado mostra uma sugestão;
    - Iddle: mostra uma sugestão quando o aluno está um tempo sem ação;

- LastError: aponta uma sugestão ao lado do bloco que disparou o último erro;
- ShowCountInstructions: aponta uma sugestão ao lado do contador de blocos. O contador de blocos aparece automaticamente quando uma missão é restrita quanto à quantidade de blocos;
- Slider: uma sugestão que aponta para o controlo da velocidade de animação;
- Free: de uso geral, sem comportamento padrão. A janela de sugestão aparece centralizada no ambiente do jogo.

- **time:** é o tempo que deve aguardar antes de disparar a sugestão;
- **modal:** aceita os valores *true* ou *false*. Define se a janela de sugestão deve ou não ser explicitamente fechada pelo aluno. No caso de não ser modal, a janela já fecha quando o aluno clicar em qualquer região da página;
- **condition:** define as condições que devem ser atendidas para apresentar a sugestão. Algumas das categorias citadas anteriormente já possuem condições padrão (por exemplo, se a quantidade de blocos é igual a um, ou a quantidade de tentativas é maior que 5). Porém é possível adicionar ou alterar suas condições. Existe um conjunto de funções acessíveis que permitem aceder o estado da solução e da execução:
  - countInstructions():int : retorna a quantidade de linhas de código. Por exemplo, o retorno dessa função com o código da Figura 5.8 será igual a seis;
  - countTopInstructions():int : retorna a quantidade de conjuntos blocos separados de blocos. Com o exemplo da Figura 5.8 será igual a 2;

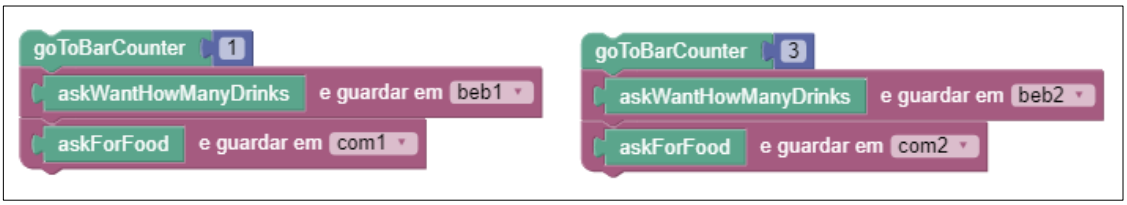


Figura 5.8 – Exemplo de dois conjuntos de blocos

  - showedHint():boolean : retorna *true* se a dica já foi exibida;
  - howManyRuns():int :retorna a quantidade de tentativas da missão;
  - countVariableName():int : retorna a quantidade de variáveis declaradas;

- blockTypeLastError():String : retorna o nome do bloco em que aconteceu o último erro. Essa função é útil quando aconteceu um erro na execução e a partir do bloco que partiu o erro, poder tentar explicar ao aluno o que ele precisa fazer;
- containsBlockTypeLastError(String):boolean : essa verifica se o bloco que disparou o erro contém o bloco passado como parâmetro. Sua aplicação é a mesma que no exemplo da função anterior;
- isGoalNotAchieved(int):boolean : verifica se o objetivo passado como parâmetro ainda não foi cumprido;
- lastTalkText():String : retorna a última mensagem que a personagem falou;
- typeActiveBlock():String : retorna o bloco selecionado na área de edição.

O conteúdo do elemento <hint> contém o texto a ser exibido na dica. Seguem alguns exemplos desse elemento:

```
<hint category="LastError" condition="Hints.blockTypeLastError() == 'do_deliver'" >
    Um erro comum é o uso da variável incorreta: guardou os produtos em uma variável
    e está fazer a entrega a partir de outra vari&#225;vel. Verifique se os blocos
    askForFood e pickupHotDog estão referenciando corretamente suas variáveis.
</hint>

<hint category="Free" condition="Hints.howManyRuns() >= 4 &&
 (Hints.isGoalNotAchieved(1) || Hints.isGoalNotAchieved(2))" >
    Consulte a sua solução da missão anterior. Nela você já tinha praticado a soma de
    variáveis.
</hint>
```

O elemento <help> serve para identificar o conteúdo do assistente de pseudocódigo. Cada linha é indicada pelo elemento <line>. O atributo *indent* é usado para endentar (colocar espaços à frente) das linhas.

```
<help>
   <line>Vá até o cliente 2</line>
   <line>Pergunte quantas bebidas ele deseja e guarde na variável [beb]</line>
   <line>Pergunte quantos lanches ele deseja e guarde na variável [com]</line>
   <line>Se [com == 3] OU [beb &gt; 1]</line>
   <line indent="1">Vá até a caixa de frutas</line>
   <line indent="1">Pegue uma fruta com [juiceOfOrange] e guarde na variável
   [frutas]</line>
   <line indent="1">Vá até a máquina de sumos</line>
   <line indent="1">Prepare e pegue o sumo com [frutas] e guarde na variável
   [bebida]</line>
   <line indent="1">Vá até o cliente 2</line>
```

```
   <line indent="1">Entregue [bebida]</line>
</help>
```

O elemento <command> serve para indicar quais são as categorias e comandos disponíveis no menu.

```
<commands>
   <category name="snackMan" show="true" />
   <category name="goToBarCounter" show="true" />
   <category name="askWantHowManyFoods" show="true" />
   <category name="askWantHowManyDrinks" show="true" />
</commands>
```

O elemento <customers> é usado para identificar as vontades dos clientes. Cada cliente é representado pelo elemento <customer>, que contém os seguintes elementos:

- <id>: é um número representando um dos três tipos de desenhos de cliente (1-homem branco de bigode, 2-mulher, 3-homem negro);
- <init>: a posição em que o cliente é iniciado na animação. Pode ser *counter*__*n*__, onde __*n*__ é 1 a 3, indicando que começa no balcão, *table*__*n*__ com __*n*__ entre 1 e 2, indicando que começa em uma mesa, ou pode ser *door* que inicia na porta da esplanada;
- <dest>: a posição final do cliente, em que ele está pronto para ser atendido. Pode ser *counter*__*n*__ ou *table*__*n*__. Se o atributo init for *door*, é executada a animação do cliente entrando na porta e caminhando até o seu destino;
- <orders>: agrupa um conjunto de um ou mais pedidos. Mais adiante será explicado como funciona com mais de um pedido;
    - <order>: representa um pedido que é o conjunto de vontades atuais. O cliente pode ter fome e/ou sede. Essas vontades são representadas por imagens dentro de um balão;
        - <foods>: conjunto de vontades de comer;
            - <food>: vontade de comer. Cada elemento possui o preço do item (atributo *price*). Como conteúdo pode ser *hotdog*, *icecreamofchocolate, icecreamofstrawberry* ou *icecreamofvanilla*;
        - <drinks>: conjunto de vontades de beber;

- <drink>: vontade de beber. Também possui o preço. Como conteúdo pode ser *coke*, *coffee* ou juiceoforange.

Segue um exemplo do elemento <customers>:

```
<customers>
   <customer>
      <id>02</id>
      <init>counter2</init>
      <dest>counter2</dest>
      <orders>
         <order>
          <foods>
             <food price="2">hotdog</food>
             <food price="2"> icecreamofchocolate</food>
          </foods>
          <drinks>
             <drink price="3">coke</drink>
          </drinks>
         </order>
      </orders>
   </customer>
</customers>
```

Para suportar ciclos de execução, ou seja, a mesma solução ser executada mais vezes com diferentes configurações de clientes, é preciso que seja indicado quantas vezes a solução será executada usando o elemento <tests>, o que implica na existência dessa quantidade de elementos <order>. O exemplo abaixo implica que serão feitas 5 execuções, logo se espera 5 elementos <order> em cada cliente.

```
<tests>5</tests>
```

Os objetivos são definidos através do elemento <objectives>. Cada objetivo é definido por um elemento <objective> dentro dele. Além disso, aquele elemento possui alguns atributos:

- **missionType**: relacionado com os tipos de tarefas (Tabela 4.32). Os valores possíveis são *multipleChoice*, *fixBugs*, *fillInGap* e *sort*. O tipo de tarefa padrão é *createFromScratch* caso não seja informado o atributo;
- **xpIndividual**: indica quantos pontos o aluno ganha por estrela conquistada;
- **xpFinal**: indica quantos pontos o aluno ganha após não haver mais estrelas;

- **xpTotalRun**: identifica quantas execuções é preciso até não ter mais estrelas. Esse valor dividido por 3 é a quantidade de execuções por estrela;
- **commQtd**: quantidade máxima de blocos permitidos;
- **maxCommands**: quantidade máxima de blocos para receber pontuação extra;
- **maxCommandsReward**: pontos extra recebidos quando a solução atende o atributo *maxCommands*;
- **maxCommandsRewardCoins**: moedas recebidas quando a solução atende o atributo *maxCommands*;
- **varQtd**: quantidade máxima de variáveis permitidas;
- **ordered**: os objetivos precisam ser cumpridos na mesma ordem que constam os elementos <objective>;
- **buttonRun**: habilita ou não o botão de execução. Por padrão é sempre habilitado;
- **buttonDebug**: habilita ou não o botão de depuração. Por padrão é sempre habilitado;
- **buttonRunQtdAttempts**: por quantas tentativas o botão de execução fica habilitado. Após essa quantidade, o aluno somente consegue testar a solução com o botão de depuração.

O conteúdo do elemento <objective> determina os tipos de objetivos da missão. De acordo o tipo de objetivo é preciso usar alguns atributos e existe um texto padrão apresentado na janela com os objetivos (item 2 na Figura 4.47). Os tipos de objetivo são:

- **goesTo**: a personagem precisa ir até a posição do cliente. O cliente destino é indicado pela junção dos atributos *pos* (1 a 3) e *place* (*counter* e *table*);
- **askForFood, askForDrink, askHasHunger, askHasThirsty, cashIn, giveTheWholeChange, askWantHowManyFoods, askForIceCream, askWantHowManyDrinks, askWantHowManyIceCream** e **deliver**: a personagem faz as devidas perguntas e interações ao cliente também identificado pelos atributos *pos* e *place*;

- **goesToDisplay** e **goesToCooler**: a personagem foi até o mostrador quente e o refrigerador, respetivamente;
- **pickUpFood** e **pickUpDrink**: a personagem pega o pedido do cliente identificado pelos atributos *pos* e *place*;
- **deliverGifts**: a personagem entrega um presente ao cliente identificado pelos atributos *pos* e *place*. Além disso possui um atributo *value* para identificar o que deve ser entregue ao cliente e um atributo *gift* usado para complementar o texto padrão do objetivo. Quando o cliente recebe um presente é executada uma animação de um coração sobre ele;
- **customDeliver**: o princípio é o mesmo do objetivo **deliverGifts**. A diferença é que não existe a animação do coração, ou seja, era uma vontade aguardada pelo cliente, mas que exige um tratamento diferenciado para indicar o seu atendimento. Por exemplo, uma missão pode ter como objetivo que caso o cliente peça dois cachorros, em vez de receber ambos, a personagem deve entregar um cachorro e um gelado. Não é um presente, mas uma condição especial de entrega;
- **giveSomeChange**: utilizado para a personagem entregar o troco parcialmente ao cliente. Existe um atributo para indicar o tipo de dinheiro a ser devolvido (por exemplo, *$$$money2* significa moedas de 2) e o valor pelo atributo *value*;
- **conditional**: objetivo que é atendido quando uma condição é satisfeita. Usa-se o atributo *condition* para determinar a regra e *text* para o texto que deve aparecer na janela de objetivos;
- **talk**: a personagem fala algo. O atributo *value* identifica o que precisa ser dito, e *text* o que deve aparecer na janela de objetivos;
- **callTimes**: a solução precisa usar o bloco identificado no atributo *type* a quantidade de vezes exatamente indicado no atributo *times*. O atributo *block* é usado para criar o texto na janela de objetivos;
- **useMath**: a solução precisa usar um bloco matemático com o operador indicado no atributo *operator*, e com os argumentos definidos nos atributos *arg0* e *arg1*. Ainda contém o atributo *text* para identificar o texto a ser exibido na janela de objetivos.

Existe um conjunto de funções utilizadas nesses objetivos. Como a comparação do cumprimento dos objetivos é baseada em atender um cliente, todas as funções estão na classe CustomerManager do ficheiro customerman.js (ver Figura 5.3):

- totalOfFood():int : retorna a quantidade de comida pedida por todos os clientes;
- totalOfDrink():int : retorna a quantidade de bebida pedida por todos os clientes;
- totalOfMoneyIfSell():int : retorna em valor total monetário se atender todos os pedidos dos clientes;
- customerMoneyIfSell(int):int : retorna o valor total monetário se atender todas as vontades do cliente sentado na posição do balcão indicado no parâmetro;
- totalOfMoneyGave():int : retorna o valor total pago pelos clientes;
- totalOfMoneyDelivered():int : retorna o valor total monetário somente do que foi entregue aos clientes;
- customerMoneyGave(int):int : retorna o valor total pago pelo cliente sentado na posição do balcão indicado no parâmetro;
- totalOfFoodDelivered():int : retorna o total da quantidade de comida entregue a todos os clientes;
- totalOfFoodDeliveredCustomer(int):int : retorna a quantidade de comida entregue para o cliente sentado no balcão indicado no parâmetro;
- totalOfDrinksDelivered():int : retorna o total da quantidade de bebidas entregues a todos os clientes;
- fullDeliveredCustomers():int : retorna a quantidade de clientes atendidos totalmente;
- fullDeliveredCustomersHowManyPlaces():int : retorna a quantidade de sítios diferentes de clientes atendidos.

Segue um exemplo do elemento <objectives>:

```
<objectives xpIndividual="50" xpFinal="10" xpTotalRun="6" commQtd="15" maxCommands="11" maxCommandsRewardCoins="2" missionType="fillInGap">
    <objective pos="2" place="counter">deliver</objective>
    <objective pos="2" place="counter">askWantHowManyFoods</objective>
    <objective block="para" type="controls_for" times="1">callTimes</objective>
</objectives>
```

O elemento <testsvars>, usado em conjunto com <tests>, define um conjunto de variáveis e seus valores em cada execução, que são usadas no teste dos objetivos. Cada iteração informada em <tests> deve corresponder a um elemento <test>. Dentro desse elemento são aceites uma ou mais variáveis. Cada variável é identificada pelo elemento <var> contendo o atributo *name* e o valor do elemento corresponde ao valor da variável. Para obter o valor dessas variáveis, o objetivo pode usar a função Game.readVariableTest(nomeVariavel). O seguinte exemplo demonstra que serão executados dois testes, usando uma variável de nome *foodPrice*. Em cada teste é verificado se a personagem falou (objetivo **talk**) o valor do cachorro. O valor é obtido através da execução da função Game.readVariableTest('foodPrice').

```
<tests>2</tests>
<testsvars>
   <test>
       <var name="foodPrice">3</var>
   </test>
   <test>
       <var name="foodPrice">1</var>
   </test>
</testsvars>
...
<objective text="o preço do cachorro"
              value="parseInt(Game.readVariableTest('foodPrice'))">talk</objective>
```

O Apêndice Q apresenta dois exemplos de missões onde podem ser consultados exemplos práticos de todos os elementos acima.

Para o controlo de versão do código fonte foi usado o GitHub, que permitiu inclusive trabalho colaborativo em alguns momentos do projeto em que tínhamos bolseiros contribuindo com a investigação. O projeto está hospedado em https://github.com/adilsonv77/nobugssnackbar.

5.4 Considerações Finais

Esse capítulo encerra a descrição da investigação ao apresentar duas das contribuições como resultado do trabalho realizado. A primeira é uma lista apresentando as lições aprendidas sobre os principais pontos do processo de

investigação, as três formas testadas de aplicar o jogo integrado em sala de aula, e uma série de princípios que podem auxiliar projetistas futuros na conceção de novos jogos para a aprendizagem de programação. A segunda contribuição foi a generalização de um modelo de arquitetura para motores de jogos casuais sérios, com base no próprio jogo desenvolvido durante os dois anos. Esse modelo apresenta componentes de jogos, de aprendizagem e aqueles referentes à abordagem construcionista de conduzir as tarefas de aprendizagem, e suas relações. O modelo pode vir a ser usado como uma inspiração e cuidados a serem tomados por projetistas de novos jogos.

Capítulo 6

CONCLUSÕES

6.1 Considerações Iniciais

O principal objetivo dessa tese foi identificar elementos de jogo e estratégias para sua integração com as aulas que pudessem contribuir para a aprendizagem de competências de resolução de problemas através da programação. Para poder atingir este objetivo, um novo jogo sério foi desenvolvido, chamado NoBug's SnackBar, e experimentado quatro vezes, em semestres distintos, usando uma metodologia iterativa e interativa, que permitiu fazer melhorias após cada ciclo. A experiência não limitou os alunos nem em tempo e nem em espaço, pois foi conduzida para livre uso em quaisquer horários e locais, permitindo-nos possivelmente avaliar resultados mais próximos da realidade em termos de fluxo e aprendizagem. No Capítulo 4 foi descrito o jogo e suas funcionalidades. No Capítulo 5 foram apresentados outros resultados da tese: um conjunto de contributos que descrevem princípios de projeto e utilização de jogos casuais para a aprendizagem de programação, e um modelo de arquitetura para motores de jogos casuais para a aprendizagem de programação. Nesse capítulo retornamos às questões que conduziram a investigação e tentamos respondê-las. Por último, serão apresentadas sugestões para a continuidade no desenvolvimento dessa investigação.

6.2 Questões de Investigação

A questão de investigação que serviu de enquadramento desse trabalho foi "Como se pode melhorar a aprendizagem das habilidades de resolução de problemas, em programação de computadores introdutória, usando um jogo digital?". Para responder à essa pergunta, decompomos em três novas perguntas que conduziram o desenvolvimento evolutivo do jogo.

A primeira pergunta abordava a validade de usar jogos na aprendizagem de programação: **Q1 – Quais são as razões para usar jogos sérios no ensino e aprendizagem das habilidades de resolução de problemas em disciplinas de introdução à programação de computadores?**

A aprendizagem de programação se materializa com a prática, ou seja, não é suficiente ler e compreender as sintaxes ou funcionamento das estruturas de controlo, mas é preciso resolver problemas construindo suas soluções: o aprender fazendo. Entretanto, não basta concretizar alguns poucos problemas. O aluno precisa construir vários programas com certa diversidade de situações. Nesse caminho ele passa por inúmeras dificuldades e frustrações, e com persistência e dedicação pode superá-las. O aluno tem de se sentir motivado para continuar essa trajetória. A manutenção dessa motivação é o principal apelo no uso de jogos sérios, pois eles são projetados para usar a diversão como um fator para continuar jogando ao resolver os problemas no jogo.

Desconsiderando o primeiro ciclo, onde foi feito um teste de conceitos, os três ciclos seguintes envolveram 129 alunos (63,6% homens e 36,4% mulheres) com 70,5% (n=91) afirmando não terem qualquer conhecimento prévio de programação. Em relação aos hábitos de jogar, praticamente a mesma população afirmou jogar diariamente 32,6% (n=42), ocasionalmente 34,1% (n=44) e raramente 33,3% (n=43). Essa distribuição é coerente com outras pesquisas que demonstraram que os alunos de cursos superiores afirmam jogar menos do que de escolaridades anteriores (Carvalho et al., 2014). Mesmo assim, quase 70% dos alunos afirmaram ter algum contato com jogos, e mesmo aqueles que declararam não ter o hábito, é bem possível que se sintam confortáveis com uso de jogos digitais.

No Capítulo 3 identificamos 54 jogos para a aprendizagem de programação publicados na literatura e disponíveis para serem acedidos. Existem jogos para

dispositivos móveis e outros executáveis em navegadores, permitindo um fácil acesso e disponibilidade para jogar. Existem jogos em que é necessário usar alguma linguagem de programação, mas na sua maioria usam blocos ou alguma outra simbologia gráfica para montagem das soluções, demonstrando o maior interesse em produzir nas pessoas as habilidades de resolverem problemas com pensamento computacional. Os trabalhos publicados sobre experiências com jogos demonstraram melhorias na aprendizagem e os alunos se sentem motivados em cumprir os desafios nos jogos.

Neste trabalho desenvolvemos um novo jogo por não termos encontrado outro jogo que permitisse a customização do projeto instrucional, tanto na definição das tarefas de aprendizagem, quanto na organização dessas tarefas. Além disso, existe uma lacuna para que os professores possam monitorar o desenvolvimento dos alunos durante o jogo, e assim auxilia-los no suporte em sala de aula. Ao oferecê-lo como um ambiente de aprendizagem, conseguimos extrapolar o limite de ser apenas um jogo, para ser uma ferramenta de apoio em que o professor possa auxiliar os alunos a vencerem as barreiras iniciais na aprendizagem de programação

Através dessas estatísticas, e trabalhos já publicados, podemos justificar a importância em usar essa abordagem para apoiar a aprendizagem de programação. A questão seguinte que tentamos responder envolve identificar o que o jogo precisa oferecer: **Q2 – Quais os recursos que devem fazer parte de um jogo sério para que promova a aprendizagem de resolução de problemas em disciplina de introdução à programação de computadores?**

Usamos algumas formas de chamar atenção inicial aos alunos para o jogo, como conseguir pontos para personalizar o avatar, três tabelas de classificação e um sistema de conquistas. Porém, um elemento de jogo que testamos que não é frequentemente considerado nas publicações, é a própria conquista de pontos. Apesar de termos criado estratégias para justificar a conquista dos pontos, observamos que com o passar do jogo, o aluno se sente atraído por ganhá-los simplesmente por satisfação pessoal, não para comparar com os demais. Os alunos criam estratégias e tentam melhorar a sua própria maneira de resolver os problemas objetivando maximizar a quantidade de pontos ganhos na missão. O aluno não se sente atraído ao jogo pelo simples fato de ser um jogo, mas pelo sentimento visível de que ele venceu um desafio. Ganhar mais pontos criou um comportamento emergente no segundo ciclo ao fazer alguns alunos rascunharem a solução primeiro no papel, e no

terceiro ciclo tentaram (e conseguiram) burlar a missão ao abrir duas instâncias do jogo, onde numa os alunos testavam e experimentavam, e com a solução na outra instância submetiam para ganhar os pontos como se fosse a primeira tentativa. Podemos concluir, que mesmo não sendo a pontuação um objetivo primário de um jogo sério, a sua utilização aumenta a atração pelo jogo.

A alternativa entre resolver problemas com codificação ou por blocos, recaiu pela segunda opção para que não houvesse necessidade prévia para ensinar a sintaxe de uma linguagem, e para que ficássemos com foco em promover no aluno a resolução de problemas e não a aprendizagem de uma linguagem de programação. No último ciclo apresentámos uma solução para o processo de transferência de conhecimento entre a construção por blocos ou por código, em que foi preciso desconectar o jogo e criar outro artefacto (o *framework* NoBugsJ). Com essa troca de tecnologias e produtos usados, perdeu-se alguma interação lúdica, como o sistema de pontuação, descrito no parágrafo anterior como primordial para manter a motivação. Uma possibilidade poderia seguir a ideia de Matsuzawa et al. (2015), em que os alunos tinham a livre escolha entre produzir numa representação ou noutra, e ainda assim alternar entre uma e outra durante um mesmo exercício, conforme a confiança e a maturidade no processo de programação.

Os erros são comuns já quando se trabalha com programação, e são mais comuns ainda enquanto está se aprendendo. Porém, as falhas, mesmo no jogo, deixam os alunos descontentes e fazem eles perderem a concentração e a motivação em continuar jogando. As falhas são necessárias para a aprendizagem. Se um aluno passa as missões sem cometer erros ou falhas, pode representar que (1) ele tenha as soluções copiadas de outros colegas, (2) o conhecimento prévio dele torna desnecessário o uso do jogo, ou (3) existe uma falha no projeto do jogo que diminui a validade das tarefas para a aprendizagem. A maior carência no jogo foi a disponibilidade de um sistema de suporte, que apresente sugestões e dicas para os alunos quando alcançarem um estado de tentativas sucessivas sem sucesso. O assistente de pseudocódigo disponibilizado nos ciclos 3 e 4 demonstrou ser útil para manter os alunos no jogo. Apesar disso, é notável nos ciclos a queda da adesão no jogo. Entre os dois últimos ciclos aumentámos a quantidade de alunos que finalizaram o jogo de 5,5% para 31,3%, e que fizeram pelo menos metade das missões de 47,8% para 64,9%, porém apenas 33,3% deles obteve nota igual ou superior a 7,0 (essa é a nota para ser aprovado na disciplina). Nosso foco na investigação não foi o estudo de

sistemas inteligentes, porém um sistema de aconselhamento acoplado ao jogo poderia fornecer o suporte personalizado. Algo parecido foi feito quando monitorámos o desenvolvimento dos alunos e intervimos enviando individualmente mensagens por correio eletrónico sugerindo e dando dicas para que vencessem a missão. Para operacionalizar esse monitoramento e intervenção, foi desenvolvido na última versão um sistema administrativo que destaca os alunos que necessitam de auxílio, e apresenta em detalhes toda as tentativas da missão atual e das anteriores.

Outra alternativa para aumentar a adesão do jogo, foi experimentar que os alunos pudessem escolher livremente as missões que desejavam jogar. O jogo continuava sugerindo as missões conforme o seu caminho de aprendizagem, mas o aluno tinha a opção de se adiantar em alguma missão. Segundo os alunos, esse recurso evitou que ficassem sem evoluir em determinada fase, e assim podiam tentar a missão seguinte. A utilidade disso foi que eles podiam aprender algo na missão seguinte, que pudesse ser usado na missão anterior, e assim manter o ritmo de evolução. Apesar de não termos evidências estatísticas de que isso manteve mais alunos, foi possível verificar que os alunos que tinham essa possibilidade de escolher qualquer missão seguiam mais próximos o projeto instrucional.

Definimos um conjunto de tarefas de acordo com a ordem da missão dentro de uma fase (que representa um assunto, como variáveis, condicionais e ciclos). Classificamos as tarefas baseando-nos na Taxonomia de Bloom, iniciando por missões com múltiplas escolhas, e terminando a sequência com tarefas de criação da solução, que é a competência meta do jogo. Apesar de não ter conseguido fazer qualquer estudo estatístico sobre essa sequência de tipos de tarefas, é bem provável que ele tenha servido ao seu propósito de introduzir o tema e prover o aluno com ferramentas para construir coisas novas.

Finalmente, o professor é um recurso fundamental no jogo. Os alunos sentiram alguma dificuldade em ver a utilidade do jogo quando o professor também não tinha jogado ou não conseguia ajudá-los a sanarem as dúvidas. Numa parte do ciclo 4 o professor esteve ausente, pelo que aquele período de estudo dos alunos foi basicamente como num curso a distância, contando com o suporte do investigador por correio eletrónico e de um monitor presencialmente. Ainda assim, os alunos apresentaram o sentimento da falta de um professor para apoiá-los.

Uma vez identificada a motivação em usar jogos e os recursos que um jogo poderia ter para alcançar algum sucesso no aperfeiçoamento de habilidades de

resolução de problemas, o nosso último desafio era identificar como integrar o jogo com as aulas, e aí surge a terceira questão: **Q3 – Como pode ser integrado um jogo sério com a resolução de problemas em disciplinas de introdução à programação de computadores?**

Em cada um dos três ciclos que duraram um semestre, o jogo foi integrado de uma maneira distinta. Nos dois primeiros ciclos os alunos afirmaram que as atividades da própria disciplina dificultavam a organização de seu tempo para conciliar a realização das tarefas no jogo. Além disso, os alunos têm um sentimento de utilidade para definir suas prioridades: se eles não visualizam uma relação direta da tarefa com a disciplina, eles tendem a negligenciar essa tarefa. Por essa razão, no último ciclo as tarefas no jogo foram conciliadas com a própria disciplina, inclusive finalizando o período de jogo com uma avaliação (uma prova ou exame), e a continuação da disciplina para uso de uma linguagem de programação partia do conhecimento adquirido no jogo. Essa integração trouxe mais conforto aos alunos poderem aliar seu tempo de estudo com a disciplina. Ainda assim, como sugestão para experiências futuras, uma quarta forma de aplicar o jogo, que diferencia do último ciclo, seria adicionar a presença do professor, e usando as aulas para dar o suporte aos alunos, exigindo assim um professor que tenha jogado.

Para que o jogo pudesse ser experimentado dessas três maneiras, ele precisou ser flexível o suficiente para que as adaptações pudessem ser implantadas com o mínimo de esforço. Isso foi possível pois o jogo foi implementado seguindo os princípios dos motores de jogos. As missões e a organização de aprendizagem não foram definidas dentro do jogo, mas em uma base de dados, e a tarefa do jogo foi interpretar a sequência e a definição da missão. Assim podemos conseguir aumentar a adoção do jogo por mais instituições de ensino e professores. Porém, é necessário o desenvolvimento de uma ferramenta para edição dessas missões e fazer o planeamento instrucional, que não foi realizado durante essa investigação.

6.3 Trabalhos Futuros

Durante a investigação foram identificadas algumas questões não resolvidas que servem de motivação para avançar no desenvolvimento de jogos para

aprimoramento de habilidades de resolução de problemas em programação. O jogo precisa ser avaliado com outras abordagens de integração com as aulas, mas para isso precisa prover um conjunto de ferramentas que facilite a sua adoção e cobrir uma maior gama de assuntos da disciplina. Igualmente será interessante efetuar testes mais abrangentes que possam verificar mais claramente o impacto da utilização do jogo na aprendizagem inicial de programação.

Em relação à trilha sonora, em todos os ciclos a música de fundo apresentou-se sempre como um recurso de jogo que não teve muita influência na diversão nem na aprendizagem. Inclusive muitos alunos relataram que preferiam ouvir sua própria lista de reprodução. A música ambiente fazia parte da evolução dentro da missão pois ela variava conforme o aluno perdia estrelas permitindo-lhe uma associação entre a música e a situação em que ele se encontrava com a pontuação daquela missão. A tensão aumentava à medida que ele perdia pontos. Outra investigação (Linek et al., 2012) já havia concluído que, apesar da trilha sonora ser um agente potencial para estimular a motivação, ela não apresentou qualquer tipo de efeito sobre a aprendizagem. A música como facilitadora da aprendizagem em jogos sérios ainda precisa ser explorada (Starks, 2014).

Os objetivos de uma missão podem servir como guia para o aluno desenvolver a sua solução. Enquanto algumas missões iniciais apresentam em detalhe os objetivos, por exemplo, “ir até ao cliente 1”, “pedir quantos lanches deseja”, “ir até ao mostrador quente”, “pegar o lanche pedido”, e assim por diante, outras missões têm como objetivo apenas “entregue o pedido do cliente 2”. Era esperado que os alunos nas fases mais avançadas já tivessem experiência e discernimento para decompor um objetivo maior em partes menores, e por isso as missões sempre apresentavam algum aspeto novo, além do que as fases mais avançadas exigiam que os alunos produzissem soluções mais complexas. Assim, poderá fazer sentido rever os objetivos de cada missão para que as das fases introdutórias sejam mais bem detalhados (com maior granularidade), e reduzindo o detalhe nas fases de domínio.

Além da revisão da granularidade dos objetivos, também precisamos repensar o funcionamento do sistema de tentativas. Atualmente, os alunos produzem a solução completa logo na primeira tentativa, pois eles ganham os pontos conforme a quantidade de tentativas. Entretanto, o ideal seria que eles resolvessem de forma iterativa e incremental, especialmente no caso das missões mais complexas. O jogo precisa dar o suporte a essa boa prática, e pode fazê-lo apresentando

incrementalmente os objetivos da missão, onde a quantidade de tentativas seja baseada nesse conjunto de objetivos, e não mais no total de tentativas da missão. Inclusive é importante variar o número de tentativas por estrela para cada novo conjunto de objetivos que é apresentado.

Neste trabalho desenvolvemos um *framework* (NoBugsJ) que permitiu ao professor auxiliar os alunos a transferir o conhecimento aprendido com os blocos para a programação Java. Se o próprio jogo já pudesse conter um editor de código, os próprios alunos já podiam fazer essa transferência com o jogo, pelo que este desenvolvimento poderá fazer sentido.

O suporte às dificuldades dos alunos poderia ser melhorado através da implementação de um Sistema de Recomendação (SR) para auxilia-los com sugestões e dicas para avançarem nas missões, ao mesmo tempo que ele poderia adaptar e personalizar o caminho de aprendizagem. Muitos alunos também apresentaram dificuldades em extrair informações do enunciado para construir a solução. O SR também pode promover esse exercício de leitura e interpretação dos enunciados.

Por fim, para permitir que o jogo possa ser adotado em diferentes contextos, atendendo mais professores e turmas, é preciso que se desenvolva um editor de missões e do projeto instrucional. Esse editor poderia promover uma comunidade, ou uma rede social, para criação de missões, assim como é feito no Robozzle[17]. Assim os usuários poderiam criar e compartilhar novas missões, e os professores usar e alterar algumas dessas missões desenvolvidas para personalizar o seu projeto instrucional.

[17] http://robozzle.com

REFERÊNCIAS BIBLIOGRÁFICAS

ACM, & IEEE. (2013). *Computer Science Curricula 2013*.

Adamo-Villani, N., Cooper, S., & Whittinghill, D. (2012). Building a serious game for teaching secure coding in introductory programming courses. In *33rd Annual Conference of the European Association for Computer Graphics* (p. A45). Cagliari, Italy.

Ahmad, M., Rahim, L. A., & Arshad, N. I. (2014). A review of educational games design frameworks: An analysis from software engineering. In *International Conference on Computer and Information Sciences* (pp. 1–6). Kuala Lumpur, Malaysia.

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005). An Analysis of Patterns of Debugging Among Novice. In *10th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 84–88). Monte de Caparica, Portugal.

Akker, J. van den, Gravemeijer, K., McKenney, S., & Nieveen, N. (2006). Introducing Educational Design Research. In J. van den Akker, K. Gravemeijer, S. McKenney, & N. Nieveen (Eds.), *Educational Design Research* (pp. 3–7). Routledge.

Aldrich, C. (2009). *Learning online with games, simulations, and virtual worlds: Strategies for online instruction*. San Francisco: John Wiley & Sons.

Alonso-Fernandez, C., Calvo, A., Freire, M., Martinez-Ortiz, I., & Fernandez-Manjon, B. (2017). Systematizing game learning analytics for serious games. In *IEEE Global Engineering Education*. Athens, Greece.

Anderson, L. W., Krathwohl, D. R., & Bloom, B. S. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Allyn & Bacon.

Armoni, M., Meerbaum-Salant, O., & Ben-ari, M. (2015). From Scratch to "Real" Programming. *ACM Transactions on Computing Education*, *14*(4), 25:1-15.

Arnab, S., Freitas, S. de, Bellotti, F., Lim, T., Louchart, S., Suttie, N., … Gloria, A. De. (2012). *Pedagogy-driven design of Serious Games: An overall view on learning and game mechanics mapping, and cognition-based models*. *Research Report*.

Barbosa, L. S., Fernandes, T. C. B., & Campos, A. M. C. (2011). Takkou : Uma Ferramenta Proposta ao Ensino de Algoritmos. In *XIX Workshop sobre Educação em Computação. XXXI Congresso da Sociedade Brasileira de Computação* (pp.

1–10). Natal, Brazil.

Barnes, T., Powell, E., Chaffin, A., Godwin, A., & Richter, H. (2007). Game2Learn: Building CS1 learning games for retention. In *12th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 121–125). Dundee, Scotland.

Barr, D., Harrison, J., & Conery, L. (2011). Computational Thinking: A Digital Age Skill for Everyone. *Learning and Leading with Technology*, *38*(6), 20–23.

Bartle, R. A. (2003). *Designing Virtual Worlds*. New Riders Publishing.

Bau, D., Bau, D. A., Dawson, M., & Pickens, C. S. (2015). Pencil Code : Block Code for a Text World. In *14th International Conference on Interaction Design and Children* (pp. 445–448). Medford, USA.

Bauer, A., Butler, E., & Popovic, Z. (2015). Approaches for Teaching Computational Thinking Strategies in an Educational Game : A Position Paper. In *IEEE Blocks and Beyond Workshop* (pp. 121–123). Atlanta, USA.

Bayliss, J. D., & Strout, S. (2006). Games as a “flavor” of CS1. In *37th ACM Technical Symposium on Computer Science Education* (pp. 500–504). Houston, USA.

Bellotti, F., Kapralos, B., Lee, K., Moreno-Ger, P., & Berta, R. (2013). Assessment in and of serious games: An overview. *Advances in Human-Computer Interaction*, 1–11.

Ben-Ari, M. (2013). Visualization of programming. In *Improving computer science education* (pp. 52–65).

Berry, M., & Kölling, M. (2014). The State Of Play : A Notional Machine for Learning Programming. In *19th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 21–26). Uppsala, Sweden.

Birchall, D., Henson, M., Burch, A., Evans, D., & Goldman, K. H. (2012). Levelling up: Towards best practice in evaluating museum games. In *Museums and the Web*. San Diego, USA.

Black, M. (2016). Seven Semesters of Android Game Programming in CS2. In *21th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 5–10). Arequipa, Peru.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing Computational Thinking in Compulsory Education*.

Boller, S., & Kapp, K. (2017). *Play to Learn: Everything You Need to Know About Designing Effective Learning Games*. ATD Press.

Bothun, D., Driscoll, L., Lieberman, M., & Yatsko, M. (2012). *The evolution of video gaming and content consumption*. PricewaterhouseCoopers LLP.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *2012 Annual Meeting of the American Educational Research Association* (pp. 1–25). Vancouver, Canada.

Brom, C., Levcik, D., Buchtová, M., & Klement, D. (2015). Playing educational micro-games at high schools : Individually or collectively ? *Computers in Human Behavior*, *48*, 682–694.

Brown, A. L. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The Journal of the Learning Sciences*, *2*(2), 141–178.

Brown, E., & Cairns, P. (2004). A grounded investigation of game immersion. In *Human actors in Computing systems* (pp. 1297–1300). Vienna, Austria.

Burning Glass Technologies. (2016). *Beyond Point and Click the Expanding Demand for Coding Skills*.

Carter, J., Bouvier, D., Cardell-Oliver, R., Hamilton, M., Kurkovsky, S., Markham, S., … White, S. (2011). ITiCSE 2011 Working Group Report Motivating All our Students ? In *16th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 1–18). Darmstadt, Germany.

Carvalho, A. A., Araújo, I. C., Zagalo, N., Gomes, T., Barros, C., Moura, A., & Cruz, S. (2014). Os jogos mais jogados pelos alunos do Ensino Básico ao Ensino Superior. In *2º Encontro sobre Jogos e Mobile Learning* (pp. 23–37). Coimbra, Portugal.

Caspersen, M. E., & Bennedsen, J. (2007). Instructional design of a programming course: A Learning theoretic approach. In *International Workshop on Computing Education Research* (pp. 111–122). Atlanta, GA.

Caspi, A., & Blau, I. (2008). Social presence in online discussion groups: Testing three conceptions and their relations to perceived learning. *Social Psychology of Education*, *11*(3), 323–346.

Caspi, A., & Blau, I. (2011). Collaboration and psychological ownership: How does the tension between the two influence perceived learning? *Social Psychology of Education*, *14*(2), 283–298.

Cesar, E., Cortés, A., Espinosa, A., Margalef, T., Moure, J. C., Sikora, A., & Suppi, R. (2017). Introducing computational thinking, parallel programming and performance engineering in interdisciplinary studies. *Journal of Parallel and*

Distributed Computing, (In press).

Chaffin, A., Doran, K., Hicks, D., & Barnes, T. (2009). Experimental Evaluation of Teaching Recursion in a Video Game. In *2009 ACM SIGGRAPH Symposium on Video Games* (Vol. 1, pp. 79–86). New Orleans, Lousiana.

Chande, S. V. (2015). A Conceptual Framework for Computational Thinking as a Pedagogical Device. *International Journal of Innovative Research in Computer and Communication Engineering*, *3*(11), 11682–11688.

Chang, C.-K. (2014). Effects of Using Alice and Scratch in an Introductory Programming Course for Corrective Instruction. *Journal of Educational Computing Research*, *51*(2), 185–204.

Chang, J. K.-W., Dang, L. H., Pavleas, J., McCarthy, J. F., Sung, K., & Bay, J. (2012). Experience with Dream Coders: developing a 2D RPG for teaching introductory programming concepts. *Journal of Computing Sciences in Colleges*, *28*(1), 227–236.

Chang, W., & Chou, Y. (2008). Introductory c programming language learning with game-based digital learning. In *Advances in Web Based Learning-ICWL 2008* (pp. 221–231). Springer, Berlin, Heidelberg.

Chao, P.-Y. (2016). Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers and Education*, *95*, 202–215.

Chen, J. (2007). Flow in games (and everything else). *Communications of the ACM*, *50*(4), 31.

Chetty, J., & Barlow-Jones, G. (2012). Bridging the Gap : the Role of Mediated Transfer for Computer Programming. In *4th International Conference on Education Technology and Computer*. Singapore, China.

Chiapello, L. (2013). Formalizing casual games: A study based on game designers' professional knowledge. In *Digital Games Research Association Conference* (pp. 1–16). Atlanta, USA.

Cliburn, D. C. (2008). Student opinions of Alice in CS1. In *38th Annual Frontiers in Education Conference* (pp. 1–6). Saratoga Springs, USA.

Cobb, P., Confrey, J., DiSessa, A., Lehrer, R., & Schauble, L. (2003). Design experiments in educational research. *Educational Researcher*, *32*(1), 9–13.

Cocciolo, A. (2005). Reviewing design-based research. Retrieved February 14, 2014, from http://www.thinkingprojects.org/wp-content/dbr.doc

Coelho, A., Kato, E., Xavier, J., & Gonçalves, R. (2011). Serious game for introductory programming. In *2nd International Conference on Serious Games Development and Applications* (pp. 61–71). Lisbon, Portugal.

Cohen, E. L. (2014). What makes good games go viral? The role of technology use, efficacy, emotion and enjoyment in players' decision to share a prosocial digital game. *Computers in Human Behavior*, *33*, 321–329.

Collins, A. (1992). Toward a design science of education. In E. Scanlon & T. O'Shea (Eds.), *New Directions in Educational Technology* (pp. 15–22). Springer Berlin Heidelberg.

Connolly, T. M., Boyle, E. a., MacArthur, E., Hainey, T., & Boyle, J. M. (2012). A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education*, *59*(2), 661–686.

Corbin, J., & Strauss, A. (2015). *Basics of qualitative research: Techniques and Procedures for Developing Grounded Theory* (4th ed.). Los Angeles, USA: Sage Publications.

Cowan, B., & Kapralos, B. (2014). A survey of frameworks and game engines for serious game development. In *IEEE 14th International Conference on Advanced Learning Technologies* (pp. 662–664). Athens, Greece.

Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience*. New York: Harper Perennial.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C. C., & Woollard, J. (2015). *Computational thinking: A guide for teachers*. Computing at Schools.

Curzon, P., Dorling, M., Ng, T., Selby, C. C., & Woollard, J. (2014). Developing computational thinking in the classroom: a framework, (June).

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer : Alice 3 to Java. In *43rd ACM Technical Symposium on Computer Science Education* (pp. 141–146). Raleigh, USA.

Dantas, V. F., Alencar, L. A., & Freitas, P. J. B. (2011). ProGame : Um jogo para apoiar o ensino-aprendizagem de programação. In *XXII Simpósio Brasileiro de Informática na Educação* (pp. 1878–1882). Aracaju, Brazil.

Dantas, V. F., Macedo, E. R. De, Andrade, J. R. B., Coutinho, D. R. A., Cavalcante, A. F., Vasconcelos, T. G., & Pereira, M. E. D. S. (2013). Combinando desafios e aventura em um jogo para apoiar a aprendizagem de programação em vários

níveis cognitivos. In *II Congresso Brasileiro de Informática na Educação*. Campinas, Brazil.

DBRC. (2003). Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher*, *32*(1), 5–8.

Debabi, W., & Bensebaa, T. (2016). Using serious game to enhance algorithmic learning and teaching. *Journal of E-Learning and Knowledge Society*, *12*(2), 127–140.

Dim, C. A., & Edson, F. (2011). APIN: Uma Ferramenta Para Aprendizagem de Lógicas e Estímulo do Raciocínio e da Habilidade de Resolução de Problemas em um Contexto Computacional no Ensino Médio. In *XIX Workshop sobre Educação em Computação. XXXI Congresso da Sociedade Brasileira de Computação* (pp. 1–9). Natal, Brazil.

DiSalvo, B. (2014). Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science. *IEEE Computer Graphics and Applications*, *34*(6), 12–15.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57–73.

Eagle, M., & Barnes, T. (2009). Experimental evaluation of an educational game for improved learning in introductory computing. In *40th ACM Technical Symposium on Computer Science Education* (pp. 321–325). Chattanooga, USA.

Eckerdal, A., McCartney, R., Moström, J. E., Sanders, K., Thomas, L., & Zander, C. (2007). From Limen to Lumen: Computing students in liminal spaces. In *3th International Workshop on Computing Education Research* (pp. 123–132). Atlanta, USA.

Entertainment Software Association. (2013). *Essential facts about the computer and video game industry: 2013 sales, demographic and usage data*. Entertainment Software Association.

Equipa de Recursos e Tecnologias Educativas. (2015). Iniciação à Programação no 1.º Ciclo do Ensino Básico. *Ministério da Educação. República Portuguesa.* Retrieved January 16, 2017, from http://www.erte.dge.mec.pt/iniciacao-programacao-no-1o-ciclo-do-ensino-basico

Eseryel, D., & Ge, X. (2010). Designing Effective Game-Based Learning Environments : Implications for Design Research. Paper presented at “Educational Design Research: Local Change, Global Impact: A Special

Conference to Honor Professor Thomas C. Reeves Upon his Retirement from the University of Georgia."

Eseryel, D., Guo, Y., & Law, V. (2012). Interactivity3 Design and Assessment Framework for Educational Games to Promote Motivation and Complex Problem-Solving Skills. In D. Ifenthaler, D. Eseryel, & X. Ge (Eds.), *Assessment in Game-Based Learning: Foundations, Innovations, and Perspectives* (pp. 257–285). New York, NY: Springer New York.

Esper, S., Wood, S. R., Foster, S. R., Lerner, S., & Griswold, W. G. (2014). Codespells: how to design quests to teach java concepts. *Journal of Computing Sciences in Colleges*, *29*(4), 114–122.

European Union Digital Single Market. (2016). Coding - the 21st century skill. Retrieved from https://ec.europa.eu/digital-single-market/en/coding-21st-century-skill

Federoff, M. A. (2002). *Heuristics and usability guidelines for the creation and evaluation of fun in video games*. Indiana University.

Felder, R. M., & Silverman, L. K. (1998). Learning and Teaching Styles in Engineering Education. *Journal of Engineering Education*, *78*(7), 674–681.

Figueiredo, L. (2011, June). *O Papel da Motivação na Construção da Aprendizagem*. Tese de doutoramento em Tese de Mestrado, Universidade de Lisboa.

Filatro, A. (2008). *Design instrucional na prática*. São Paulo: Pearson Education do Brasil.

Filatro, A. (2010). *Design instrucional contextualizado* (3a ed.). São Paulo: Senac São Paulo.

Fu, F. L., Su, R. C., & Yu, S. C. (2009). EGameFlow: A scale to measure learners' enjoyment of e-learning games. *Computers and Education*, *52*(1), 101–112.

Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernán-Losada, I., Jackova, J., … Thompson, E. (2007). Developing a computer science-specific learning taxonomy. In *12th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 152–170). Dundee, Scotland.

Gao, Y., & Mandryk, R. L. (2011). GrabApple: The design of a casual exergame. In *10th International Conference of Entertainment Computing* (pp. 35–46). Vancouver, Canada.

Garlick, R., & Cankaya, E. C. (2010). Using Alice in CS1 – A Quantitative Experiment. In *15th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 165–168). Ankara, Turkey.

Gee, J. P. (2003). *What video games have to teach us about learning and literacy*. New York: Palgrave Macmillan.

Gee, J. P. (2004). Learning by design: Games as learning machines. *Interactive Educational Multimedia*, *8*(April 2004), 15–23.

Gomes, A. de J., Santos, Á. N. F. S., Páris, C. P. das D., & Martins, N. C. (2017). Playing with Programming: A Serious Game to Start Programming. In *Gamification-based e-learning Platform for Computer Programming Education* (pp. 261–277).

Gomes, A., & Mendes, A. J. (2007). Learning to program-difficulties and solutions. In *International Conference on Engineering Education* (pp. 1–5). Coimbra, Portugal.

Gomes, A., & Mendes, A. J. (2015). À procura de um contexto para apoiar a aprendizagem inicial de programação. *Educação, Formação & Tecnologias*, *8*(1), 13–27.

Google. (n.d.). Computational Thinking Concepts Guide. Retrieved January 17, 2017, from https://static.googleusercontent.com/media/www.wenca.cn/en/us/edu/pdf/women-who-choose-what-really.pdf

Gravemeijer, K., & Cobb, P. (2006). Design research from a learning design perspective. *Educational Design Research*, 17–51.

Green, T. R. G., & Petre, M. (1996). Usability Analysis of Visual Programming Environments: A “Cognitive Dimensions” Framework. *Journal of Visual Languages and Computing*, *7*(2), 131–174.

Gregory, J. (2009). *Game Engine Architecture*. A K Peters.

Greller, W., & Drachsler, H. (2012). Translating Learning into Numbers: A Generic Framework for Learning Analytics. *Educational Technology and Society*, *15*(3), 42–57.

Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43.

Guzdial, M. (2004). Programming Environments for Novices Specializing Environments for Novices Logo and its Descendants: The Goal of Computational Literacy. *Computer Science Education Research*, 127–154.

Guzdial, M., & Soloway, E. (2002). Teaching the Nintendo generation to program. *Communications of the ACM*, *45*(4), 17.

Harvey, B., & Mönig, J. (2010). Bringing “No Ceiling” to Scratch: Can One Language

Serve Kids and Computer Scientists? *Constructionism*, 1–10.

Hatfield, D., & Shaffer, D. W. (2006). Press play: Designing an epistemic game engine for journalism. In *7th International Conference on Learning Sciences* (pp. 236–242). Bloomington, USA.

Hidalgo-Céspedes, J., Marín-Raventós, G., & Lara-Villagrán, V. (2014). Playing with Metaphors: A Methodology to Design Video Games for Learning Abstract Programming Concepts. In *19th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 348–348). Uppsala, Sweden.

Hijon-Neira, R. B., Velázquez-Iturbide, Á., Pizarro-Romero, C., & Carriço, L. (2014). Game programming for improving learning experience. In *19th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 225–230). Uppsala, Sweden.

Holbert, N. R., & Wilensky, U. (2011). FormulaT racing: Designing a game for kinematic exploration and computational thinking. In *7th International Conference on Games + Learning + Society*. Madison, USA.

Ibanez, L. (2015). Blockly makes it easier to learn to code. Retrieved February 2, 2017, from https://opensource.com/education/15/2/blockly-makes-easier-every-one-learn-code

Iten, N., & Petko, D. (2016). Learning with serious games: is fun playing the game a predictor of learning success? *British Journal of Educational Technology*, *47*(1), 151–163.

Iuppa, N., & Borst, T. (2010). *End-to-End Game Development: Creating Independent Serious Games and Simulations from Start to Finish*. Oxford: Elsevier Inc.

Jenkins, T. (2002). On the Difficulty of Learning to Program. In *3rd Annual LTSN-ICS Conference* (pp. 53–58). Loughborough, UK.

Johnson, L., Becker, S. A., Estrada, V., & Freeman, A. (2015). *Horizon Report: 2015 Higher Education Edition*. *Reading*. Austin, Texas: The New Media Consortium.

Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, *32*(3), 241–254.

Juul, J. (2010). *A casual revolution: Reinventing video games and their players*. Cambridge, MA: MIT Press.

Kadle, A. (2009). Casual and Serious Digital Games for Learning – Some Considerations. Retrieved December 27, 2015, from https://www.upsidelearning.com/blog/index.php/2009/04/18/casual-and-serious-

digital-games-for-learning-some-considerations/

Kahwage, C., França, E. L. de, Nunes, R. C., Carvalho, R., & Souza, D. T. (2013). Jogo Baralho das Variáveis. In *XXXIII Congresso da Sociedade Brasileira de Computação* (pp. 450–459). Maceio, Brazil.

Kapp, K. M. (2012). *The gamification of learning and instruction: Game-based methods and strategies for training and education*. San Francisco, CA: Pfeiffer.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012). A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. *Procedia - Social and Behavioral Sciences*, *47*, 1991–1999.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2013). Understanding computational thinking before programming: Developed guidelines for the design of games to learn introductory programming through game-play. In P. Felicia (Ed.), *Developments in Current Game-Based Learning Design and Development*. Hershey, PA: IGI Global.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming. *ACM Computing Surveys*, *37*(2), 83–137.

Khenissi, M. A., Essalmi, F., Jemni, M., Kinshuk, Graf, S., & Chen, N. S. (2016). Relationship between learning styles and genres of games. *Computers and Education*, *101*, 1–14.

Kinnunen, P., & Malmi, L. (2006). Why students drop out CS1 course? In *Second International Workshop on Computing Education Research* (pp. 97–108).

Klopfer, E., & Begel, A. (2007). StarLogo TNG: An Introduction to Game Development. *Journal of E-Learning*, 1–15.

Koster, R. (2014). *A Theory of Fun for Game Design* (2nd ed.). O' Reilly Media, Inc.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. In *10th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 14–18). Monte de Caparica, Portugal.

Lameras, P., Arnab, S., Dunwell, I., Stewart, C., Clarke, S., & Petridis, P. (2017). Essential features of serious games design in higher education: Linking learning attributes to game mechanics. *British Journal of Educational Technology*, *48*(4), 972–994.

Landers, R. N., & Callan, R. C. (2011). Casual social games as serious games: The psychology of gamification in undergraduate education and employee training. In M. Ma, A. Oikonomou, & L. C. Jain (Eds.), *Serious Games and Edutainment*

Applications (pp. 399–423). Springer London.

Laurillard, D., Charlton, P., Craft, B., Dimakopoulos, D., Ljubojevic, D., Magoulas, G., … Whittlestone, K. (2013). A constructionist learning environment for teachers to model learning designs. *Journal of Computer Assisted Learning*, *29*(1), 15–30.

Lee, G. E., Xu, Y., Brewer, R. S., & Johnson, P. M. (2014). Makahiki: An open source game engine for sustainability education and conservation. In *International Conferences on Education Technologies and Sustainability* (pp. 7–11). New Tapei City, Taiwan.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., … Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, *2*(1), 32–37.

Lee, M. J., Bahmani, F., Kwan, I., Laferte, J., Charters, P., Horvath, A., … Ko, A. J. (2014). Principles of a debugging-first puzzle game for computing education. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 57–64). Melbourne, Australia.

Lee, M. J., & Ko, A. J. (2011). Personifying Programming Tool Feedback Improves Novice Programmers' Learning. In *7th International Workshop on Computing Education Research* (pp. 109–116). Providence, Rhode Island.

Leonardou, A., & Rigou, M. (2016). An adaptive mobile casual game for practicing multiplication. In *20th Pan-Hellenic Conference on Informatics*. Patras, Greece.

Leutenegger, S., & Edgington, J. (2007). A games first approach to teaching introductory programming. In *38th ACM Technical Symposium on Computer Science Education* (Vol. 39, pp. 115–118). Covington, USA.

Levy, R. B.-B., & Ben-Ari, M. (2008). Perceived behavior control and its influence on the adoption of software tools. In *13th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 169–173). Madrid, Spain.

Linek, S. B., Marte, B., & Albert, D. (2012). Background Music in Educational Game: Motivational Appeal and Cognitive Impact. In *Developments in Current Game-Based Learning Design and Deployment* (1st ed., pp. 219–230). IGI Global.

Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. In *14th Annual Conference on Innovation and Technology in Computer Science Education* (Vol. 41, pp. 161–165). Paris, France.

Liu, C.-C., Cheng, Y.-B., & Huang, C.-W. (2011). The effect of simulation games on the learning of computational problem solving. *Computers & Education*, *57*(3),

1907–1918.

López, M. A., Duarte, E. V., Gutierrez, E. C., & Valderrama, A. P. (2016). Teaching Abstraction , Function and Reuse in the First Class of CS1 – A Lightbot Experience. In *21th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 256–257). Arequipa, Peru.

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In *4th International Workshop on Computing Education Research* (pp. 101–112). Sydney, Australia: ACM Press.

Lu, J. J., & Fletcher, G. H. L. (2009). Thinking About Computational Thinking Categories and Subject Descriptors. In *40th ACM Technical Symposium on Computer Science Education* (pp. 260–264). Chattanooga, USA.

Magnussen, R. (2008, November). *Representational inquiry in science learning games*. University of Aarhus, Danish School of Education.

Majgaard, G., Misfeldt, M., & Nielsen, J. (2011). How design-based research and action research contribute to the development of a new design for learning. *Designs for Learning*, *4*(2), 8–27.

Malan, D. J. (n.d.). CS50 at Harvard. Retrieved January 18, 2017, from https://cs50.harvard.edu/

Malan, D. J., & Leitner, H. H. (2007). Scratch for Budding Computer Scientists. In *38th ACM Technical Symposium on Computer Science Education* (pp. 223–227). Covington, USA.

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2013). Towards a new massive multiplayer online role playing game for introductory programming. In *6th Balkan Conference in Informatics* (pp. 156–163). Thessaloniki, Greece: ACM Press.

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2017). CMX : The Effects of an Educational MMORPG on Learning and Teaching Computer Programming. *IEEE Transactions on Learning Technologies*, *10*(2), 219–235.

Malone, T. W. (1980). What makes things fun to learn? Heuristics for designing instructional computer games. In *3rd ACM SIGSMALL Symposium* (Vol. 162, pp. 162–169). Palo Alto, California, USA.

Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education*, *16*(3), 211–227.

Maragos, K., & Grigoriadou, M. (2011). Exploiting TALENT as a Tool for Teaching and Learning. *The International Journal of Learning*, *18*(1), 431–440.

Marfisi-Schottman, I., George, S., & Tarpin-Bernard, F. (2010). Tools and Methods for Efficiently Designing Serious Games. In *4th European Conference on Game-Based Learning* (pp. 226–234). Copenhagen, Denmark.

Matsuzawa, Y., Ohata, T., Sugiura, M., & Sakai, S. (2015). Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment Basic Function and Interface. In *46th ACM Technical Symposium on Computer Science Education* (pp. 185–190). Kansas City, USA.

Mazlan, M. N. A., & Burd, L. (2011). Does an avatar motivate? In *41th Annual Frontiers in Education Conference*. Rapid City, USA.

McDaniel, R., Fiore, S. M., & Nicholson, D. (2010). Serious Storytelling: Narrative Considerations for Serious Games Researchers and Developers. In *Serious game design and development: Technologies for training and learning*. Hershey, PA: IGI Global.

Mckernan, B., Mikeal, R., Stromer-galley, J., Kenski, K., Clegg, B. a, Folkestad, J. E., … Strzalkowski, T. (2015). We don't need no stinkin' badges : The impact of reward features and feeling rewarded in educational games. *Computers in Human Behavior*, *45*, 299–306.

McMahon, M. (2009). The DODDEL Model: A Flexible Document-Oriented Model for the Design of Serious Games. In *Games-Based Learning Advancements for Multi-Sensory Human Computer Interfaces: Techniques and Effective Practices*. Londres: IGI Global.

Mead, J., Gray, S., Hamer, J., James, R., Sorva, J., Clair, C. St., & Thomas, L. (2006). A cognitive approach to identifying measurable milestones for programming skill acquisition. In *11th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 182–194). Bologna, Italy.

Miljanovic, M. A. (2015). *RoboBUG : A Game-Based Approach to Learning Debugging Techniques*. University of Ontario Institute of Technology.

Miljanovic, M. A., & Bradbury, J. S. (2016). Robot ON!: A Serious Game for Improving Programming Comprehension. In *Games and Software Engineering* (pp. 33–36). Austin, USA.

Mishra, S., Balan, S., Iyer, S., & Murthy, S. (2014). Effect of a 2-week Scratch Intervention in CS1 on Learners with Varying Prior Knowledge. In *19th Annual*

Conference on Innovation and Technology in Computer Science Education (pp. 45–50). Uppsala, Sweden.

Mladenović, S., Krpan, D., & Mladenović, M. (2016). Using Games to Help Novices Embrace Programming: From Elementary to Higher Education. *International Journal of Engineering Education*, *32*(1), 521–531.

Moreno-Ger, P., Burgos, D., Martínez-Ortiz, I., Sierra, J. L., & Fernández-Manjón, B. (2008). Educational game design for online education. *Computers in Human Behavior*, *24*(6), 2530–2540.

Moser, P. (2017). Programa Qualificação em Tecnologia para Jovens através do Ensino Lúdico e da Robótica (UDESC). Retrieved from http://ceavi.udesc.br/?id=678

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. In *35th ACM Technical Symposium on Computer Science Education* (pp. 75–79). Norfolk, USA.

Mullins, P., Whitfield, D., & Conlon, M. (2009). Using alice 2.0 as a first language. *Journal of Computing Sciences in Colleges*, *24*(3), 136–143.

Muratet, M., Torguet, P., Viallet, F., & Jessel, J. (2010). Experimental feedback on Prog & Play, a serious game for programming practice. In *31st Annual Conference of the European Association for Computer Graphics* (pp. 1–8). Norrköping, Sweden.

Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: the good, the bad, and the quirky – a qualitative analysis of novices' strategies. In *39th ACM Technical Symposium on Computer Science Education* (pp. 163–167). Portland, USA.

O'Neil, H. F., Wainess, R., & Baker, E. L. (2005). Classification of learning outcomes: evidence from the computer games literature. *Curriculum Journal*, *16*(4), 455–474.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York: Basic Books, Inc.

Papert, S. (1988). *Logo : computadores e educação* (3rd ed.). São Paulo: Editora Brasiliense.

Park, C. S., & Kim, H. J. (2013). A framework for construction safety management and visualization system. *Automation in Construction*, *33*(April), 95–103.

Parsons, D., & Haden, P. (2007). Programming osmosis: Knowledge transfer from imperative to visual programming environments. In *Conference of the National Advisory Committee on Computing Qualifications* (pp. 209–215). Nelson, New

Zealand.

Pavlas, D. (2010). *A Model of Flow and Play in Game-based Learning: The Impact of Game Characteristics, Player Traits, and Player States*. of Central Florida.

Pea, R. D., Soloway, E., & Spohrer, J. C. (1987). The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics*, *9*(1), 5–30.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., … Paterson, J. (2007). A survey of literature on the teaching of introductory programming. In *12th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 204–223). Dundee, Scotland.

Perkins, D. N., & Salomon, G. (1988). Teaching for Transfer. *Educational Leadership*, *46*(1), 22–32.

Perron, B., & Wolf, M. J. P. (2009). *The Video Game Theory Reader 2*. New York: Taylor & Francis.

Piech, C., Sahami, M., Koller, D., Cooper, S., & Blikstein, P. (2012). Modeling how students learn to program. In *43rd ACM Technical Symposium on Computer Science Education* (pp. 153–158). Raleigh, USA.

Piteira, M., & Costa, C. (2013). Learning Computer Programming: Study of difficulties in learning programming. In *International Conference on Information Systems and Design of Communication* (pp. 75–80). Lisbon, Portugal.

Piteira, M., & Haddad, S. (2011). Innovate in your program computer class: an approach based on a serious game. In *Workshop on Open Source and Design of Communication* (pp. 49–54). Lisbon, Portugal.

Ponsen, M. J. V., Lee-Urban, S., Muñoz-Avila, H., Aha, D. W., & Molineaux, M. (2005). Stratagus: An open-source game engine for research in real-time strategy games. In *Reasoning, Representation, and Learning in Computer Games* (pp. 78–83). Edinburgh, Scotland.

Powell, B. (1982). *Guia do chefe escoteiro*. *Editora Escoteira*. União dos Escoteiros do Brasil.

Powers, K., Ecott, S., & Hirshfield, L. M. (2007). Through the looking glass: Teaching CS0 with Alice. In *38th ACM Technical Symposium on Computer Science Education* (pp. 213–217). Covington, USA.

Prensky, M. (2001). *Digital game-based learning*. New York: McGraw-Hill.

Prensky, M. (2005). Computer games and learning: Digital game-based learning. In

Handbook of Computer Game Studies (pp. 97–122). Cambridge: The MIT Press.

Price, T. W., & Barnes, T. (2015). Comparing Textual and Block Interfaces in a Novice Programming Environment. In *11th International Conference on International Computing Education Research* (pp. 91–99). Omaha, USA.

Radoff, J. (2011). *Game on: Energize your business with social media games*. Indianapolis, IN: Wiley.

Rizvi, M., Humphries, T., Major, D., Jones, M., & Lauzun, H. (2011). A CS0 Course using Scratch. *The Journal of Computing Sciences in Colleges*, *26*(3), 19–27.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172.

Ronimus, M., Kujala, J., Tolvanen, A., & Lyytinen, H. (2014). Children's engagement during digital game-based learning of reading: The effects of time, rewards, and challenge. *Computers and Education*, *71*, 237–246.

Roque, R. V. (2007). *OpenBlocks: an extendable framework for graphical block programming systems*. Massachusetts Institute of Technology.

Rosbach, A. H., & Bagge, A. H. (2013). Classifying and Measuring Student Problems and Misconceptions. In *Norsk Informatikkonferanse* (pp. 1–12). Stavanger, Norway.

Salen, K., & Zimmerman, E. (2004). *Rules of play: Game design fundamentals*. Cambridge, MA: MIT Press.

Savi, R., Wangenheim, C. G. Von, & Borgatto, A. F. (2011). A Model for the Evaluation of Educational Games for Teaching Software Engineering. In *25th Brazilian Symposium on Software Engineering* (pp. 194–203). São Paulo, Brazil.

Scaico, P., Marques, D. L., Melo, L. D. A., André, M., Neto, S. V. M., Oliveira, A., … Scaico, A. (2012). Um jogo para o ensino de programação em Python baseado na taxonomia de Bloom. In *XX Workshop sobre Educação em Computação. XXXII Congresso da Sociedade Brasileira de Computação* (pp. 1–10). Curitiba, Brazil.

Schanzer, E., Krishnamurthi, S., & Fisler, K. (2015). Blocks Versus Text : Ongoing Lessons from Bootstrap. In *IEEE Blocks and Beyond Workshop* (pp. 125–126). Atlanta, USA.

Schell, J. (2008). *The art of game design: A book of lenses*. Burlington, MA: Elsevier Inc.

Selby, C. C. (2014). *How Can the Teaching of Programming Be Used to Enhance Computational Thinking Skills?* University of Southampton.

Selby, C. C. (2015). Relationships: Computational Thinking, Pedagogy of Programming, and Bloom's Taxonomy. In *10th Workshop in Primary and Secondary Computing Education*. London, UK.

Settle, A., Vihavainen, A., & Sorva, J. (2014). Three views on motivation and programming. In *19th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 321–322). Uppsala, Sweden.

Shabalina, O., Vorobkalov, P., Kataev, A., & Tarasenko, A. (2008). Educational games for learning programming languages. In *Methodologies and Tools of the Modern (e-)Learning* (pp. 79–83). Sofia, Bulgaria: Institute of Information Theories and Applications.

Shelton, B. E., & Scoresby, J. (2010). Aligning game activity with educational goals: following a constrained design approach to instructional computer games. *Educational Technology Research and Development*, *59*(1), 113–138.

Sherry, J., & Pacheco, A. (2006). Matching Computer Game Genres to Educational Outcomes. *The Electronic Journal of Communication*, *16*(1&2), 1–9.

Shodiev, H. (2015). Computational Thinking and Simulation in Teaching Science and Mathematics. In *Interdisciplinary Topics in Applied Mathematics, Modeling and Computational Science* (pp. 405–410). Springer International Publishing.

Shute, V. J., & Ke, F. (2012). Assessment in Game-Based Learning. In D. Ifenthaler, D. Eseryel, & X. Ge (Eds.), *Assessment in game-based learning: Foundations, innovations and perspectives* (pp. 43–58). New York: Springer.

Sirkiä, T., & Sorva, J. (2012). Exploring programming misconceptions: An analysis of student mistakes in visual program simulation exercises. In *12th Koli Calling International Conference on Computing Education Research* (pp. 19–28). Tahko, Finland.

Smith, A. M., Nelson, M. J., & Mateas, M. (2010). LUDOCORE: A logical game engine for modeling videogames. In *IEEE Conference on Computational Intelligence and Games* (pp. 91–98). Copenhagen, Denmark.

Smith, P. A., & Sanchez, A. (2010). Mini-games with major impacts. In J. Cannon-Bowers & C. Bowers (Eds.), *Serious game design and development: Technologies for training and learning* (pp. 1–12). Hershey, PA: IGI Global.

Soares, A., Fonseca, F., & Martin, N. L. (2015). Teaching introductory programming with game design and problem-based learning. *Issues in Information Systems*, *16*(Iii), 128–137.

Soflano, M., Connolly, T. M., & Hainey, T. (2015). An Application of Adaptive Games-Based Learning based on Learning Style to Teach SQL. *Computers & Education*, *86*, 192–211.

Soloman, B. A., & Felder, R. M. (1999). Index of Learning Styles Questionnaire. Retrieved June 18, 2014, from http://www.engr.ncsu.edu/learningstyles/ilsweb.html

Sorva, J., Karavirta, V., & Malmi, L. (2013). A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Transactions on Computing Education*, *13*(4), 15.1-15.64.

Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: are the folk wisdoms correct? *Communications of the ACM*, *29*(7), 624–632.

Squire, K. D. (2005a). Design research for game-based learning. Retrieved February 19, 2014, from http://www.academiccolab.org/resources/documents/DBR-ED-Tech4.pdf

Squire, K. D. (2005b). Resuscitating research in educational technology: Using game-based learning research as a lens for looking at design-based research. *Educational Technology*, *45*(1), 8–14.

Stamouli, I., & Huggard, M. (2006). Object oriented programming and program correctness: the students' perspective. In *Second International Workshop on Computing Education Research* (pp. 109–118). Canterbury, UK.

Starks, K. (2014). Cognitive behavioral game design : a unified model for designing serious games. *Frontiers in Psychology*, *5*(February), 1–10.

Striewe, M., & Goedicke, M. (2014). Code reading exercises using run time traces. In *19th Annual Conference on Innovation and Technology in Computer Science Education* (p. 346). Uppsala, Sweden.

Sung, K. (2009). Computer games and traditional CS courses. *Communications of the ACM*, *52*(12), 74–78.

Sweetser, P., & Wyeth, P. (2005). GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment*, *3*(3), 1–24.

Torrente, J., Manero, B., & Fernández-Manjón, B. (2014). A game engine to learn computer science languages. In *44th Annual Frontiers in Education Conference* (pp. 765–771). Madrid, Spain.

Trefry, G. (2010). *Casual game design: Designing play for the gamer in all of Us*. Burlington, MA: Elsevier.

Vahldick, A., Mendes, A. J., & Marcelino, M. J. (2014). A review of games designed to improve introductory computer programming competencies. In *44th Annual Frontiers in Education Conference* (pp. 781–787). Madrid, Spain.

Van Staalduinen, J.-P., & Freitas, S. de. (2011). A game-based learning framework: Linking game design and learning. In *Learning to play: exploring the future of education with video games* (pp. 29–45). New York: Peter Lang.

Wahner, T., Kartheuser, M., Sigl, S., Nolte, J., & Hoppe, A. (2012). Logical Thinking by Play Using the Example of the Game “Space Goats.” In *3rd International Conference on Serious Games Development and Applications* (pp. 174–182). Bremen, Germany.

Walker, D. (2006). Toward productive design studies. In J. van den Akker, K. Gravemeijer, S. McKenney, & N. Nieveen (Eds.), *Educational Design Research* (pp. 9–19). Routledge.

Wang, F., & Hannafin, M. J. (2005). Design-based research and technology-enhanced learning environments. *Educational Technology Research and Development*, *53*(4), 5–23.

Wang, H., & Sun, C.-T. (2011). Game reward systems : Gaming experiences and social meanings. In *Digital Games Research Association Conference* (pp. 1–15). Utrecht, The Netherlands.

Wangenheim, C. A. G. Von. (2017). Computação na Escola (UFSC). Retrieved January 16, 2017, from http://www.sbc.org.br/institucional-3/chancela-sbc/computacao-na-escola

Wassila, D., & Tahar, B. (2012). Using serious game to simplify algorithm learning. In *International Conference on Education and e-Learning Innovations* (pp. 1–5). Sousse, Tunisia.

Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. In *19th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 39–44). Uppsala, Sweden.

Watson, C., Li, F. W. B., & Lau, R. W. H. (2011). Learning Programming Languages through Corrective Feedback and Game-Based Concept Visualisation. In *Advances in Web-Based Learning - ICWL 2011* (pp. 11–20).

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147.

Weintrop, D., Holbert, N. R., Wilensky, U., & Horn, M. (2012). Redefining Constructionist Video Games: Marrying Constructionism and Video Game Design. In *Constructionism 2012* (pp. 645–649). Athens, Greece.

Weintrop, D., & Wilensky, U. (2014). Situating programming abstractions in a constructionist video game. *Informatics in Education*, *13*(2), 307–321.

Weintrop, D., & Wilensky, U. (2015). To block or not to block , that is the question : Students' perceptions of blocks-based programming. In *ACM SIGCHI Interaction Design and Children* (pp. 199–208). Medford, USA.

Weintrop, D., & Wilensky, U. (2016a). Bringing Blocks-based Programming into High School Computer Science Classrooms. In *Annual Meeting of the American Educational Research Association*. Washington, USA.

Weintrop, D., & Wilensky, U. (2016b). Playing by Programming: Making Gameplay a Programming Activity. *Educational Technology*, *56*(3), 36–41.

Westera, W., Nadolski, R. J., Hummel, H. G. K., & Wopereis, I. G. J. H. (2008). Serious games for higher education: A framework for reducing design complexity. *Journal of Computer Assisted Learning*, *24*(5), 420–432.

White House. (2016). President Obama Announces Computer Science For All Initiative. Retrieved January 18, 2017, from https://www.whitehouse.gov/the-press-office/2016/01/30/fact-sheet-president-obama-announces-computer-science-all-initiative-0

Whitton, N. (2010). *Learning with digital games: A practical guide to engaging students in higher education*. New York: Taylor & Francis.

Wilson, C. (2015). Hour of code---a record year for computer science. *ACM Inroads*, *6*(1), 22–22.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of The Royal Society A*, *366*(July), 3717–3725.

Wing, J. M. (2010). Computational Thinking: What and Why? *Unpublished work*. Retrieved January 17, 2017, from http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

Wing, J. M., & Stanzione, D. (2016). Progress in computational thinking, and expanding the HPC community. *Communications of the ACM*, *59*(7), 10–11.

Winslow, L. E. (1996). Programming pedagogy: A psychological overview. *ACM*

SIGCSE Bulletin, *28*(3), 17–22.

Wolz, U., Leitner, H. H., Malan, D. J., & Maloney, J. (2009). Starting with scratch in CS 1. In *40th ACM Technical Symposium on Computer Science Education* (pp. 2–3). Chattanooga, USA.

Wong, L.-H., Boticki, I., Sun, J., & Looi, C.-K. (2011). Improving the scaffolds of a mobile-assisted Chinese character forming game via a design-based research cycle. *Computers in Human Behavior*, *27*(5), 1783–1793.

Xinogalos, S., Satratzemi, M., & Malliarakis, C. (2017). Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment? *Education and Information Technologies*, (22), 145–176.

Zapušek, M., & Rugelj, J. (2013). Learning programming with serious games. *EAI Endorsed Transactions on Game-Based Learning*, *13*(3), 1–8.

Zyda, M. (2005). From visual simulation to virtual reality to games. *Computer*, *38*(9), 25–32.

Apêndice A

LEVANTAMENTO DE SUGESTÕES E HÁBITOS COM JOGOS

Instrumento aplicado a uma turma de alunos após apresentar o protótipo em papel.

Caro estudante, Primeiramente agradeço sua atenção e cooperação durante a apresentação e discussão do protótipo em papel do jogo que pretendo desenvolver durante o meu doutoramento. Agora, preciso de uma última contribuição sua respondendo o presente questionário. Ele tem dois objetivos: (i). mensurar os hábitos quanto ao uso de jogos na vida quotidiana dos alunos da sua turma; (ii). identificar ideias que podem contribuir para que o jogo seja mais divertido e desafiador. Agradeço novamente, Adilson Vahldick – Doutorando em Ciência e Tecnologia da Informação/FCTUC

0. Identificação:	Idade:	Quantidade de semestres que frequentou a disciplina introdutória de programação:

1. Em média, quantas horas por semana você gasta com jogos digitais (uma semana tem 168 horas)?
( ) < 8 ( ) 8 a 16 ( ) 16 a 24 ( ) > 24

2. Quanto desse tempo você costuma gastar jogando em *multiplayer* ?

( ) Não costumo jogar multiplayer ( ) Jogo pouco multiplayer	( ) Em média metade do meu tempo ( ) Essencialmente, jogo multiplayer

3. Escreva o nome dos últimos dois jogos digitais que você jogou ou que está jogando.

4. Escreva o nome de dois jogos do Facebook que mais gostastes de jogar.

5. Você costuma jogar (ou já jogou) outros tipos de jogos que não são os digitais (por exemplo, cartas e de tabuleiro) ? Em caso positivo, cite os nomes daqueles que você considera importante mencionar porque lhe influenciaram/influenciam de alguma forma.

6. Naturalmente, é muito provável que se gaste mais tempo resolvendo os problemas dentro do jogo do que para resolver os mesmos problemas através dos exercícios tradicionais.

Remeta-se ao semestre passado, lembrando do seu desempenho na disciplina. Assinale a alternativa que corresponde à sua necessidade naquela altura.

[ ] Gostaria que o jogo fosse usado **opcionalmente** em conjunto com os exercícios tradicionais, ou seja, como uma prática adicional para quem se interessar.
[ ] Gostaria que o jogo fosse usado **adicionalmente** em conjunto com os exercícios tradicionais, ou seja, para reforçar os conceitos dos exercícios.
[ ] Gostaria que o jogo tenha mesma importância do que os exercícios tradicionais, ou seja, haja uma **distribuição igualitária** na quantidade de práticas entre ambos.
[ ] Gostaria que o jogo fosse **evidentemente** usado em conjunto com os exercícios tradicionais, ou seja, os exercícios tradicionais viriam para complementar os conceitos praticados no jogo.
[ ] Gostaria que o jogo fosse usado **antes** de começarmos a praticar com os exercícios tradicionais.
[ ] Gostaria que os exercícios tradicionais fossem opcionais.

7. Escreva duas sugestões para o jogo, e dois pontos que não te interessariam ver no jogo.

Seria fixe se o jogo...	**Não me interessa ver no jogo...**

8. Que sugestões você daria em relação à pontuação e bonificação de pontos no jogo ?

9. O jogo vai aplicar vários conceitos da disciplina introdutória de programação. Escreva uma sugestão para cada um dos conceitos abaixo.

a. Variáveis	b. Estruturas de condição
c. Estruturas de iteração/repetição	d. Funções e passagens de parâmetros
e. Recursividade	f. Listas

10. O cenário do protótipo do jogo apresentado é uma esplanada (*snack bar*). Escreva duas alternativas de cenários que poderíamos usar na contextualização do jogo.

Apêndice B

DADOS DEMOGRÁFICOS PARA O CICLO 1

#	Pergunta
1	Género
2	Idade
3	Com que frequência você costuma jogar jogos digitais?
4	Quanto desse tempo você costuma gastar jogando em multiplayer?
5	Qual é o seu conhecimento prévio em programação/algoritmos?
6	Escreva o nome dos últimos dois jogos digitais que você jogou ou que está jogando.
7	Escreva o nome de dois jogos do Facebook que mais gostou de jogar.

Apêndice C

CLASSIFICAÇÃO DE BARTLE

#	Pergunta	Opção 1	Opção 2
1	Você acha mais interessante num jogo se:	Estiver conversando com os amigos num bar	Caçando Orcs sozinho para aumentar a sua experiência
2	Ao jogar, que é mais divertido:	Ter a maior pontuação na lista	Derrotar o seu melhor amigo num combate um contra o outro
3	Uma nova área é aberta no jogo. O que lhe empolga mais para entrar nessa área?	Descobrir a sua história	Receber o novo equipamento da área
4	Você prefere ganhar:	Um concurso de trivia	Uma batalha de arena
5	Num jogo multiplayer, você prefere:	Ter uma espada duas vezes mais potente do que qualquer outro no jogo	Ser a pessoa mais temida no jogo
6	Lhe agrada mais:	Matar um grande monstro	Vangloriar-se como os amigos sobre esse seu feito
7	O que é mais importante num multiplayer para você?	O número de áreas para explorar	O número de pessoas jogando
8	Num jogo multiplayer, outro jogador matou você. Você quer:	Descobrir por que, e tentar convencê-lo a não fazer novamente.	Traçar a sua vingança
9	Num jogo multiplayer, você prefere ser conhecido como:	Alguém com o melhor e mais exclusivo equipamento no jogo.	Alguém que pode correr de um lado a outro do mundo, e realmente conhece os melhores caminhos.
10	Num jogo multiplayer, você prefere participar de um clã de:	Assassinos	Sábios
11	O que é mais excitante?	Uma batalha mortal	Um cenário bem interativo
12	Você prefere:	Saber mais segredos do que os seus amigos	Aumentar o seu nível mais rápido do que os seus amigos
13	Você prefere:	Ganhar dois níveis de experiência	Ter um amuleto que aumenta o dano que você faz contra outros jogadores em 10%
14	Num jogo multiplayer, você prefere ser:	Popular	Rico
15	O que faz você desfrutar mais num jogo multiplayer:	Fazer seus próprios mapas do mundo, em seguida, vendê-los	Administrar o seu próprio bar

16	Você prefere:	Convencer os seus inimigos a trabalhar para você, não contra você	Derrotar os seus inimigos
17	Você prefere:	Derrotar um inimigo	Explorar uma nova área
18	Você prefere receber como recompensa:	Uma varinha com três cargas de um feitiço que lhe permite controlar outros jogadores, contra a sua vontade	Pontos de experiência
19	O que faz você desfrutar mais num jogo multiplayer:	Conseguir um novo item	Saber das últimas fofocas
20	Você está sendo perseguido por um monstro. O que você faz?	Pede a um amigo para ajudar a matá-lo	Esconder-se num lugar que sabe que aí o monstro não vai segui-lo
21	O que você desfruta mais?	Ganhar um duelo contra outro jogador.	Ser aceite por um clã.
22	O que você prefere fazer:	Resolver um enigma que ninguém mais consegue	Obter um certo nível de experiência mais rapidamente do que qualquer outra pessoa
23	O que você prefere ter num jogo multiplayer:	Canal privado, sobre o qual você e seus amigos podem se comunicar	A sua própria casa, valendo milhões de moedas de ouro
24	Se você está sozinho numa área, você acha que:	É seguro explorar	Vai procurar outros lugares para caçar
25	Você é um guerreiro num jogo multiplayer, e você quer lutar contra um dragão muito difícil. Como você aborda esse problema?	Experimenta uma variedade de armas e magia contra ele, até que você encontre a sua fraqueza.	Obtém um grande grupo de jogadores para matá-lo.
26	Você é um guerreiro em um jogo multiplayer, e prestes a entrar em um calabouço desconhecido. Você pode convidar mais um personagem para essa aventura. Você traz:	Um bardo, que é um bom amigo e ótimo para entreter você	Um feiticeiro para identificar os itens que você lá encontrar
27	Você prefere:	Mostrar-lhe a lâmina afiada do seu machado	Ouvir o que alguém tem a dizer
28	Você tende a:	Saber coisas que ninguém mais faz	Ter itens que mais ninguém tem
29	Num jogo multiplayer, você encontra um novo jogador. Você acha que ele é :	Alguém que possa apreciar o seu conhecimento do jogo	Uma presa em potencial
30	Num jogo multiplayer, você estaria mais propenso a se gabar:	Pelos outros jogadores que você matou	Pelo seu equipamento

Apêndice D

EGAMEFLOW ADAPTADO

Fator	#	Conteúdo
Concentração	C3	A maior parte das atividades no jogo estão relacionadas com a tarefa de aprendizagem
	C4	As distrações não são destacadas no jogo
	C5	Geralmente eu conseguia permanecer concentrado no jogo
	C6	Eu não me distraio das tarefas que um jogador necessita de se concentrar
	C7	Eu não estava sobrecarregado com tarefas não relacionadas com os objetivos da missão
	C8	A quantidade de trabalho no jogo é adequada
Clareza nos Objetivos	G1	As metas globais do jogo foram apresentadas no começo do jogo
	G2	As metas globais do jogo foram apresentadas com clareza
	G3	As metas intermédias foram apresentadas no começo de cada missão
	G4	As metas intermédias foram apresentadas com clareza
Suporte	F1	Eu recebi suporte sobre o meu progresso durante o jogo
	F2	Eu recebi suporte imediato sobre as minhas ações
	F3	Eu fui notificado imediatamente sobre novas tarefas
	F4	Eu fui notificado imediatamente sobre novos eventos
	F5	Eu recebi imediatamente informação sobre o meu sucesso (ou falha) das metas das missões
Desafios	H3	O jogo forneceu *dicas* suficientes que me ajudaram a superar os desafios
	H4	O jogo forneceu suporte suficiente que me ajudou a superar os desafios
	H8	As dificuldades nos desafios aumentam à medida que melhoram as minhas capacidades
	H9	O jogo fornece novos desafios nos tempos apropriados
Autonomia	A7	Eu tenho a sensação de controlo sobre o jogo
	A8	Eu sabia qual devia ser o meu próximo passo no jogo
Imersão	I1	Eu não senti o tempo passar durante o jogo
	I2	Eu esqueci-me das coisas ao meu redor enquanto jogava
	I3	Temporariamente esqueci dos problemas diários enquanto jogava
	I5	Eu consegui ficar envolvido no jogo
	I6	Eu estou emocionalmente envolvido com o jogo
	I7	Eu sinto-me profundamente envolvido com o jogo
		O que MAIS te agradou no jogo?
		O que MENOS te agradou no jogo?

Apêndice E

INQUÉRITO FINAL NO CICLO 2

Questão	Conteúdo
Q1	O jogo ensinou-me a raciocinar logicamente
Q2	O jogo ensinou-me sobre o uso de variáveis
Q3	O jogo fez-me entender a divisão da solução de um problema em pequenas tarefas
Q4	Eu aprendi coisas novas com o jogo que foram úteis na disciplina
Q5	O jogo fez-me compreender melhor alguns assuntos que eu já tinha visto
Q6	O jogo ensinou-me a importância da depuração
Q7	O jogo ensinou-me como usar condicionais simples, compostos e encadeados
Q8	O jogo ensinou-me a aplicar ciclos for
Q9	O jogo ensinou-me a aplicar ciclos while
Q10	Com o avançar das missões ficou evidente a minha evolução em termos de raciocínio lógico
Q11	Eu tinha o conhecimento mínimo de programação exigido para resolver as missões
Q12	Existe grande relação entre as missões no jogo e os exercícios da aula
Q13	A dificuldade dos desafios estava compatível com o meu conhecimento em programação (com o que aprendi em sala).
Q14	A relação entre a quantidade de estrelas e a passagem do tempo fez-me ficar envolvido na missão
Q15	A música ambiente do jogo ditava o ritmo do meu trabalho
Q16	A personalização do avatar foi um fator que me motivou
Q17	Estar nas melhores posições das tabelas de classificação foi um fator que me motivou
Q18	Conquistar a pontuação para abrir o meu próprio negócio foi um fator que me motivou
Q19	Eu compreendi o tipo de ação que todos os blocos faziam
Q20	O suporte que o jogo me oferecia foi suficiente para resolver as missões
Q21	A dificuldade entre uma missão e a seguinte era adequada
Q22	Eu compreendi as mensagens de erros que o jogo me mostrava
Q23	Eu entendi quais eram os objetivos que não consegui cumprir que o jogo me mostrava
Q24	Usar programação codificada (p.e. Processing) em vez dos blocos poderia ser mais divertido
Q25	O contexto do jogo (esplanada) é atrativo
Q26	A qualidade gráfica do jogo foi um fator que NÃO me desmotivou
Q27	Em relação ao assunto de programação, escreve o que te atrairia se o jogo tivesse essa opção
Q28	Em relação ao jogo, escreve uma sugestão de algum recurso que te atrairia se o jogo tivesse essa opção
Q29	Escreve o principal motivo (seja do jogo ou externo) que te levou a parar de jogar (ou a nem jogar)

Apêndice F

ENTREVISTA SEMIESTRUTURADA

#	Descrição
1	Qual foi a tua estratégia no jogo? Quando entras na missão, como procedes?
2	Fala-me da tua experiência no jogo.
3	O que mais te agrada, ou te chama a atenção no jogo?
4	Quais as tuas dificuldades no jogo (o que menos te agrada, o que te frustra)?
5	Lembras-te de algum erro/bug no jogo?
6	O tempo das missões é suficiente?
7	O suporte oferecido pelo jogo é suficiente? Ajuda-te a resolver as missões?
8	O material instrucional tem sido útil ou é totalmente dispensável?
9	O que achas da trilha sonora do jogo? O comportamento do som e da situação/do momento em que estás jogando?
10	A qualidade dos gráficos tem alguma influência no teu desempenho?
11	Tens feito logo as missões quando o professor as disponibiliza? Em caso negativo, qual é o motivo de deixar para a última hora?
12	Costumas fazer todas as missões extras? Em caso negativo, qual é o motivo?
13	Quanto do teu tempo livre acreditas que estás despendendo no jogo? Isso traz-te satisfação ou frustração?
14	Acreditas que o jogo está contribuindo para entender o assunto da disciplina? De que forma está contribuindo: com exemplos, parte teórica, a prática, revisão do assunto?
15	Jogaste sozinho ou com colegas do lado? Como foi a interação, a conversa? Como os colegas resolveram o deles: copiaram, olhando para a tua resposta, ou executaram aquilo de que se lembraram?
16	A questão de ganhar XP interfere na tua concentração? Tentas resolver com pressa para alcançar o máximo possível de XP?
17	O que mudavas no jogo? Quais as tuas sugestões para o jogo?

Apêndice G

APRENDIZAGEM PERCECIONADA NO CICLO 3

Questão	Descrição
Q1	O jogo ajudou-me a desenvolver o raciocínio lógico
Q2	O jogo fez-me entender como manipular variáveis
Q3	O jogo fez-me entender a usar condicionais
Q4	Eu aprendi com o jogo que facilita o meu trabalho dividir o problema em partes menores
Q5	Eu aprendi coisas novas com o jogo que foram úteis na disciplina
Q6	O jogo fez-me compreender melhor alguns assuntos que eu já tinha visto
Q7	Eu aprendi com o jogo como é importante depurar para resolver os erros
Q8	Existe grande relação entre as missões no jogo e o conteúdo das aulas
Q9	Com o avançar das missões ficou evidente a minha evolução em termos de raciocínio lógico
Q10	A dificuldade dos desafios era compatível com o meu conhecimento em programação

Apêndice H

DIVERSÃO PERCECIONADA NO CICLO 3

Questão	Descrição
Q11	O contexto do jogo (lanchonete) é atrativo
Q12	Eu fiquei desmotivado com a qualidade gráfica do jogo
Q13	A música ambiente do jogo ditava o ritmo do meu trabalho
Q14	Eu compreendi o tipo de ação que todos os blocos faziam
Q15	O suporte que o jogo me oferecia foi suficiente para resolver as missões
Q16	A dificuldade entre uma missão e a seguinte era adequada
Q17	Eu fiquei motivado em jogar para obter pontos que me permitisse aceder a personalização do avatar
Q18	Eu estava motivado para ficar nas melhores posições das tabelas de classificação
Q19	Conquistar a pontuação para abrir o meu próprio negócio foi um fator que me motivou
Q20	Eu compreendi as mensagens de erros que o jogo me mostrava
Q21	Eu entendi quais eram os objetivos que não consegui cumprir que o jogo me mostrava
Q22	Sugere algum recurso no jogo que gostavas que tivesse

Apêndice I

EGAMEFLOW REDUZIDO NO CICLO 3

Fator	#	Conteúdo
Concentração	C3	A maior parte do meu tempo gasto no jogo foi em tarefas de aprendizagem de programação
	C7	Eu não estava sobrecarregado com tarefas não relacionadas com os objetivos da missão
	C8	A quantidade de trabalho no jogo é adequada
Clareza nos Objetivos	G1	As metas globais do jogo foram apresentadas no começo do jogo
Suporte	F1	Eu recebi suporte sobre o meu progresso durante o jogo
	F2	Eu recebi suporte imediato sobre as minhas ações
Desafio	H3	O jogo forneceu *dicas* suficientes que me ajudaram a superar os desafios
Autonomia	A7	Eu tenho a sensação de controlo sobre o jogo
Imersão	I1	Eu não senti o tempo passar durante o jogo
	I7	Eu sinto-me profundamente envolvido com o jogo
	Q23	O que MAIS te agradou no jogo?
	Q24	O que MENOS te agradou no jogo?

Apêndice J

INQUÉRITO FINAL NO CICLO 3

#	# ant.[18]	Descrição	G1	G2	G3	G4	G5	G6
Q01	C3	A maior parte do meu tempo gasto no jogo foi em tarefas de aprendizagem de programação	X	X	X	X	X	
Q02	C7	Eu não estava sobrecarregado com tarefas não relacionadas com os objetivos da missão	X	X	X	X	X	
Q03	C8	A quantidade de trabalho no jogo é adequada	X	X	X	X	X	
Q04	G2[19]	As metas globais do jogo foram apresentadas com clareza	X	X	X	X	X	
Q05	G4[2]	As metas intermediárias foram apresentadas com clareza	X	X	X	X	X	
Q06	F1	Eu recebi suporte sobre o meu progresso durante o jogo	X	X	X	X	X	
Q07	F2	Eu recebi suporte imediato sobre as minhas ações	X	X	X	X	X	
Q08	Q15	O suporte que o jogo me oferecia foi suficiente para resolver as missões	X	X	X	X	X	
Q09	Q20	Eu compreendi as mensagens de erros que o jogo me mostrava	X	X	X	X	X	
Q10	Q21	Eu entendi quais eram os objetivos que não consegui cumprir que o jogo me mostrava	X	X	X	X	X	
Q11	H3	O jogo forneceu *dicas* suficientes que me ajudaram a superar os desafios	X	X	X	X	X	
Q12		As dificuldades nos desafios aumentam à medida que melhoram as minhas capacidades	X	X	X	X	X	
Q13	Q10	A dificuldade dos desafios era compatível com o meu conhecimento em programação	X	X	X	X	X	
Q14	Q16	A dificuldade entre uma missão e a seguinte era adequada	X	X	X	X	X	
Q15	A7	Eu tenho a sensação de controlo sobre o jogo	X	X	X	X	X	
Q16	Q14	Eu compreendi o tipo de ação que todos os blocos faziam.	X	X	X	X	X	
Q17	I5[2]	Eu consegui ficar envolvido no jogo	X	X	X	X	X	
Q18	I6[2]	Enquanto eu jogava senti-me entusiasmado(a).	X	X	X	X	X	
Q19	I1	Eu não senti o tempo passar durante o jogo	X	X	X	X	X	
Q20	Q11	O contexto do jogo (lanchonete) é atrativo	X	X	X	X	X	
Q21	Q12	Eu fiquei desmotivado com a qualidade gráfica do jogo	X	X	X	X	X	
Q22	Q13	A música ambiente do jogo ditava o ritmo do meu trabalho	X	X	X	X	X	
Q23	Q17	Eu fiquei motivado em jogar para obter pontos que me liberassem a personalização do avatar	X	X	X	X	X	
Q24	Q18	Eu estava motivado em ficar nas melhores posições das tabelas de classificação	X	X				

[18] Questões dos inquéritos de aprendizagem (Apêndice G) e diversão (Apêndice H) percecionada, e EGameFlow reduzido (Apêndice I).

[19] Questões do EGameFlow Adaptado (Apêndice D)

Q25	Q1	Eu aprendi com o jogo a raciocinar logicamente (sequência de passos)	X	X	X	X		
Q26	Q2	Eu aprendi com o jogo como manipular variáveis	X	X	X	X		
Q27	Q3	Eu aprendi com o jogo a usar condicionais	X	X	X			
Q28		Eu aprendi com o jogo a usar ciclos	X	X				
Q29	Q4	Eu aprendi com o jogo que facilita o meu trabalho dividir o problema em partes menores	X	X	X	X		
Q30	Q7	Eu aprendi com o jogo como é importante depurar para resolver os erros	X	X	X	X		
Q31	Q5	Eu aprendi coisas novas com o jogo que foram úteis na disciplina	X	X	X	X		
Q32	Q6	O jogo fez-me compreender melhor alguns assuntos que eu já tinha visto	X	X	X	X	X	
Q33	Q9	Existe grande relação entre as missões no jogo e o conteúdo das aulas	X	X	X	X	X	
Q34		A minha motivação foi influenciada por obter pontos extras nas provas	X	X	X			
Q35		Qual foi a tua maior dificuldade enquanto jogavas?	X	X	X	X	X	
Q36	Q22	Sugere algum recurso que mais te atrairia se estivesse no jogo.	X	X	X	X	X	X
Q37		Qual foi o principal motivo que te fez parar de jogar?		X	X	X	X	
Q38		Eu recomendo usar o jogo para essa disciplina no próximo semestre.	X	X	X	X	X	
Q39		Qual foi o principal motivo de não jogares?						X

Apêndice K

ÍNDICE DE ESTILOS DE APRENDIZAGEM

#	Pergunta	Opção a	Opção n
1	Eu compreendo melhor alguma coisa depois de	experimentar.	refletir sobre ela.
2	Eu me considero	realista.	inovador(a) .
3	Quando eu penso sobre o que fiz ontem, é mais provável que aflorem	figuras.	palavras.
4	Eu tendo a	compreender os detalhes de um assunto, mas a estrutura geral pode ficar imprecisa.	compreender a estrutura geral de um assunto, mas os detalhes podem ficar imprecisos.
5	Quando estou aprendendo algum assunto novo, ajuda-me	falar sobre ele.	refletir sobre ele.
6	Se eu fosse um professor, eu preferiria ensinar uma disciplina	que trate com fatos e situações reais.	que trate com ideias e teorias.
7	Eu prefiro obter novas informações através de	figuras, diagramas, gráficos ou mapas.	instruções escritas ou informações verbais.
8	Quando eu compreendo	todas as partes, consigo entender o todo.	o todo, consigo ver como as partes se encaixam.
9	Num grupo de estudo, trabalhando um material difícil, eu provavelmente	tomo a iniciativa e contribuo com ideias.	assumo uma posição discreta e escuto.
10	Acho mais fácil	aprender fatos.	aprender conceitos.
11	Num livro com uma porção de figuras e desenhos, eu provavelmente	observo as figuras e desenhos cuidadosamente.	atento para o texto escrito.
12	Quando resolvo problemas de matemática, eu	usualmente trabalho de maneira a resolver uma etapa de cada vez.	frequentemente antevejo as soluções, mas tenho que me esforçar muito para conceber as etapas para chegar a elas.
13	Nas disciplinas que cursei eu	em geral fiz amizade com muitos dos colegas.	raramente fiz amizade com muitos dos colegas.
14	Em literatura de não-ficção, eu prefiro	algo que me ensine fatos novos ou me indique como fazer alguma coisa.	algo que me apresente novas ideias para pensar.

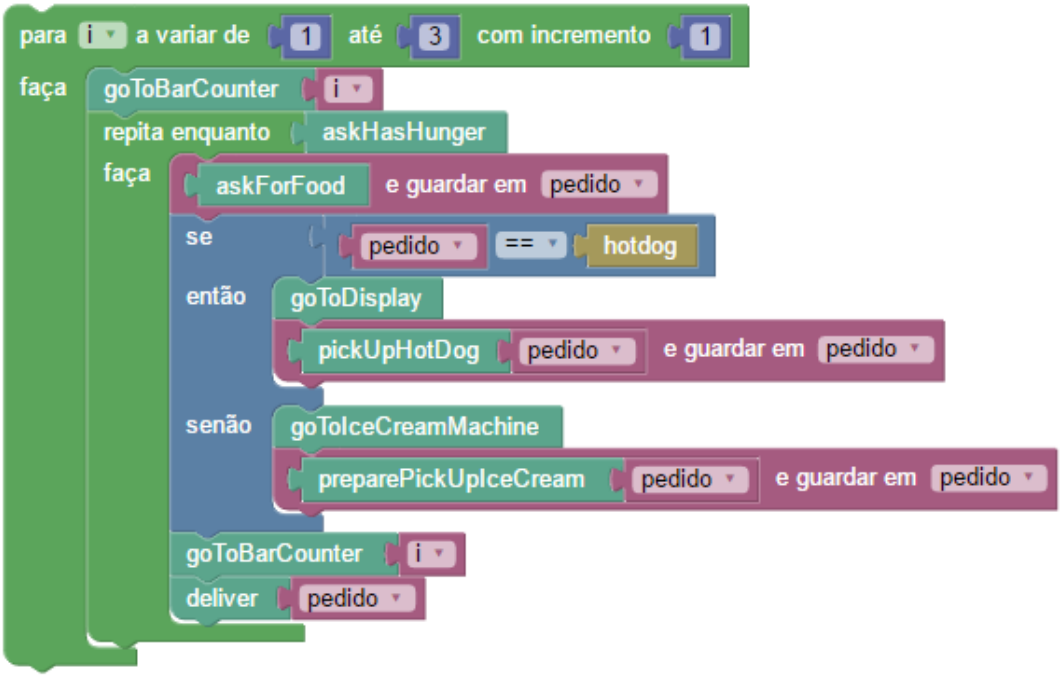
15	Eu gosto de professores	que colocam uma porção de diagramas no quadro.	que gastam bastante tempo explicando.
16	Quando estou analisando uma história ou novela eu	penso nos incidentes e tento colocá-los juntos para identificar os temas.	tenho consciência dos temas quando termino a leitura e então tenho que voltar atrás para encontrar os incidentes que os confirmem.
17	Quando inicio a resolução de um problema para casa, normalmente eu	começo a trabalhar imediatamente na solução.	primeiro tento compreender completamente o problema.
18	Prefiro a ideia do	certo.	teórico.
19	Relembro melhor	o que vejo.	o que ouço.
20	É mais importante para mim que o professor	apresente a matéria em etapas, sequências claras.	apresente um quadro geral e relacione a matéria com outros assuntos.
21	Eu prefiro estudar	em grupo.	sozinho(a).
22	Eu costumo ser considerado(a)	cuidadoso(a) com os detalhes do meu trabalho.	criativo(a) na maneira de realizar meu trabalho.
23	Quando procuro orientação para chegar a um lugar desconhecido, eu prefiro	um mapa.	instruções por escrito.
24	Eu aprendo	num ritmo bastante regular. Se estudar *pesado*, eu "chego lá".	em saltos. Fico totalmente confuso(a) por algum tempo, e então, repentinamente eu tenho um "click".
25	Eu prefiro primeiro	experimentar as coisas.	pensar sobre como é que eu vou fazer.
26	Quando estou lendo como lazer, eu prefiro escritores que	explicitem claramente o que querem dizer.	dizem as coisas de maneira criativa, interessante.
27	Quando vejo um diagrama ou esquema numa aula, relembro mais facilmente	a figura.	o que o(a) professor(a) disse a respeito dela.
28	Quando considero um conjunto de informações, provavelmente eu	presto mais atenção nos detalhes e não percebo o quadro geral.	procuro compreender o quadro geral antes de atentar para os detalhes.
29	Relembro mais facilmente.	algo que fiz.	algo sobre o qual pensei bastante.
30	Quando tenho uma tarefa para executar, eu prefiro	dominar uma maneira para a execução da tarefa.	encontrar novas maneiras para a execução da tarefa.
31	Quando alguém está me mostrando dados, eu prefiro	diagramas e gráficos.	texto sumarizando os resultados.
32	Quando escrevo um texto, eu prefiro trabalhar (pensar a respeito ou escrever)	a parte inicial do texto e avançar ordenadamente.	diferentes partes do texto e ordená-las depois.
33	Quando tenho que trabalhar num projeto em grupo, eu prefiro que se faça primeiro	um debate (brainstorming) em grupo, onde todos	um brainstorming individual, seguido de reunião do grupo para comparar ideias.

		contribuem com ideias.	
34	Considero um elogio chamar alguém de	sensível.	imaginativo.
35	Das pessoas que conheço numa festa, provavelmente recordo-me melhor	de sua aparência.	do que elas disseram de si mesmas.
36	Quando estou aprendendo um assunto novo, eu prefiro	concentrar-me no assunto, aprendendo o máximo possível.	tentar estabelecer conexões entre o assunto e outros com ele relacionados.
37	Mais provavelmente sou considerado(a)	expansivo(a).	reservado(a).
38	Prefiro disciplinas que enfatizam	material concreto (fatos, dados).	material abstrato (conceitos, teorias).
39	Para entretenimento, eu prefiro	assistir televisão.	ler um livro.
40	Alguns professores iniciam as suas aulas com um resumo do que irão cobrir. Tais resumos são	de alguma utilidade para mim.	muito úteis para mim.
41	A ideia de fazer o trabalho de casa em grupo com a mesma nota para todos do grupo	agrada-me.	não me agrada.
42	Quando estou fazendo cálculo longos	tendo a repetir todos os passos e a conferir o meu trabalho cuidadosamente.	acho cansativo conferir o meu trabalho e tenho que me esforçar para fazê-lo.
43	Tendo a descrever os lugares onde estive	com facilidade e com bastante detalhe.	com dificuldade e sem detalhe.
44	Quando estou resolvendo problemas em grupo, mais provavelmente eu	penso nas etapas do processo de solução.	penso nas possíveis consequências, ou sobre as aplicações da solução para uma ampla faixa de áreas.

Apêndice L

Primeiro Exame Aplicado no Ciclo 4

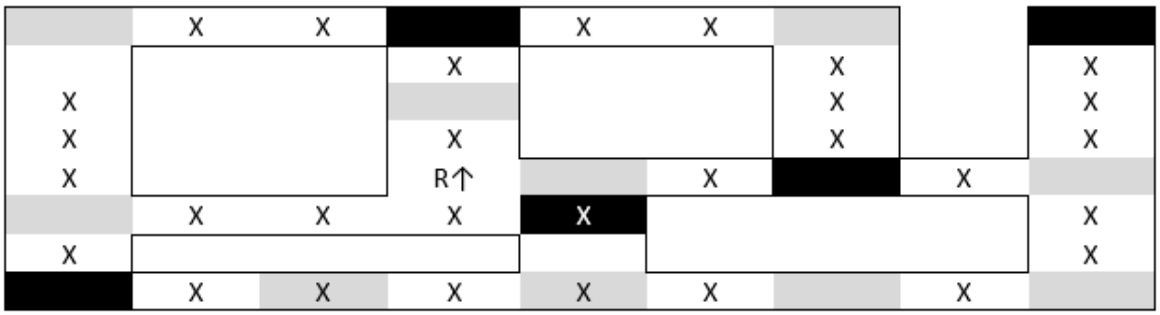
Para facilitar o desenvolvimento das respostas, em vez de desenhar os blocos, podes escrever em português, como no exemplo abaixo.



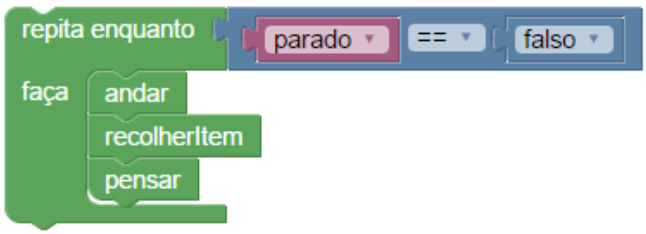

```
para i de 1 até 3 com 1
faça
  goToBarCounter(i)
  repita enquanto (askHasHunger)
  faça
    askForFood em pedido
    se pedido == hotdog
    então
      goToDisplay
      pickUpHotDog(pedido)em pedido
    senão
      goToIceCreamMachine
```

```
            preparePickUpIceCream(pedido) em pedido
        goToBarCounter(i)
        deliver(pedido)
```

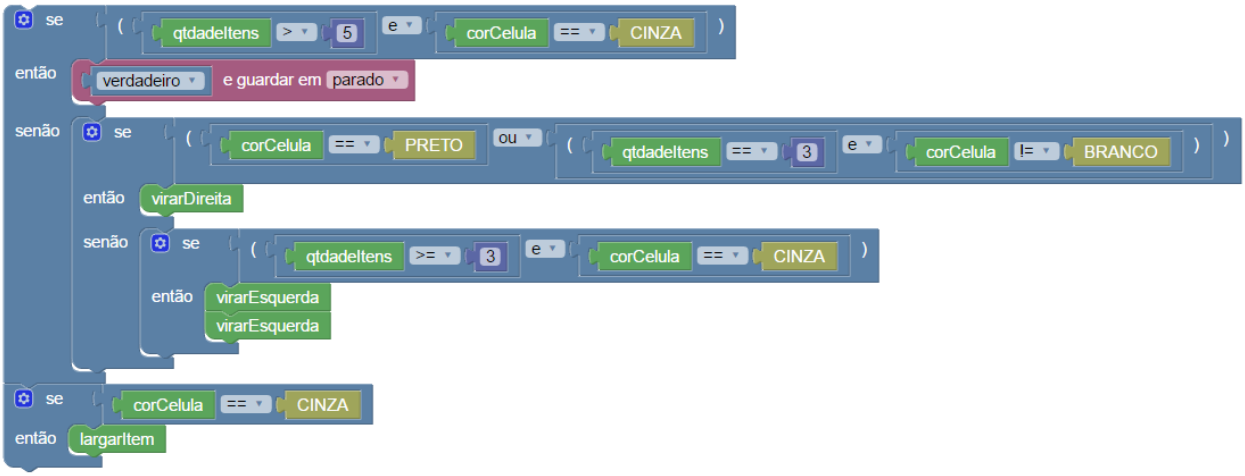
Questão 1



A figura acima representa um labirinto com itens (X) no caminho. Um robô (R) navega por esse labirinto. O símbolo (↑) indica a direção em que o robô inicia a navegação. O programa abaixo apresenta a lógica geral do robô, assim que ele é ligado. O bloco **andar** anda um passo para frente. O bloco **recolherItem** pega UM item que tem na célula (se existir algum item), retira do labirinto e coloca na bolsa. A variável **parado** inicia com o valor **falso.**

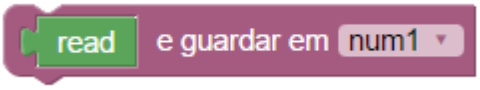
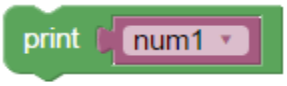


O programa abaixo representa a lógica do bloco **pensar.** O bloco **corCelula** retorna a cor da célula (branco, cinza ou preto). O bloco **qtdadeItens** retorna quantos itens tem na bolsa. O bloco **largarItem** devolve um item na célula em que o robô está (pode haver mais de um item na mesma célula). Os blocos **virarEsquerda** e **virarDireita** fazem o robô girar -90 ou 90 graus, respetivamente, em redor do seu eixo.

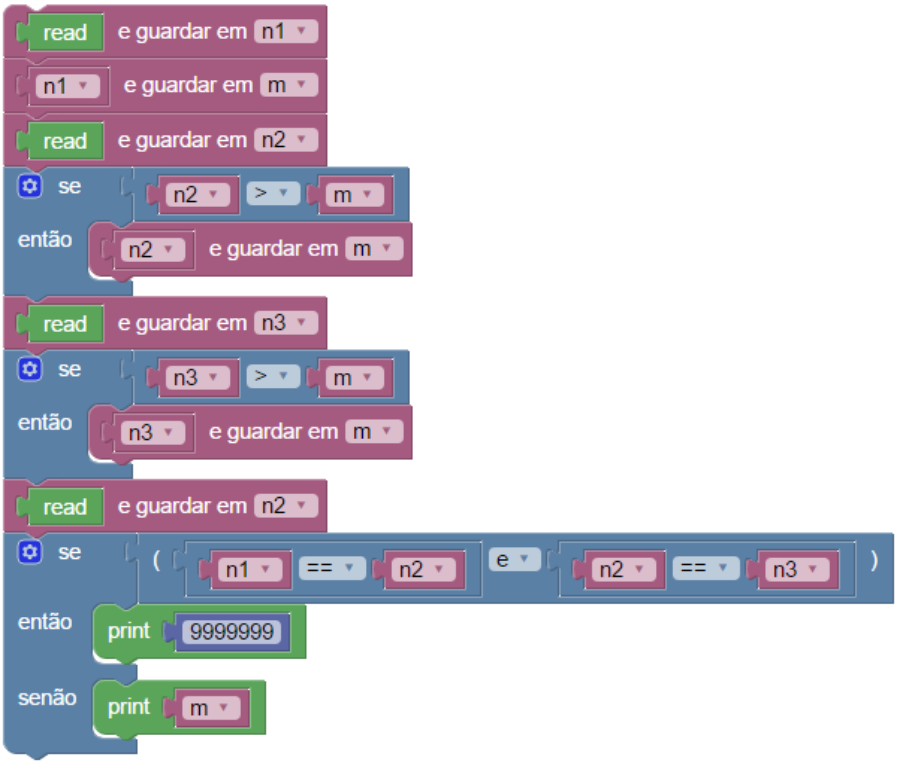


Após executar o programa acima, quantos itens existem na bolsa do robô ? _____

As questões a seguir usarão dois tipos novos de blocos:

Bloco	Explicação	Exemplo	Escrita simplificada
	Pedir para o usuário digitar um número.		`read em num1`
	Imprimir um número na tela.		`print(num1)`

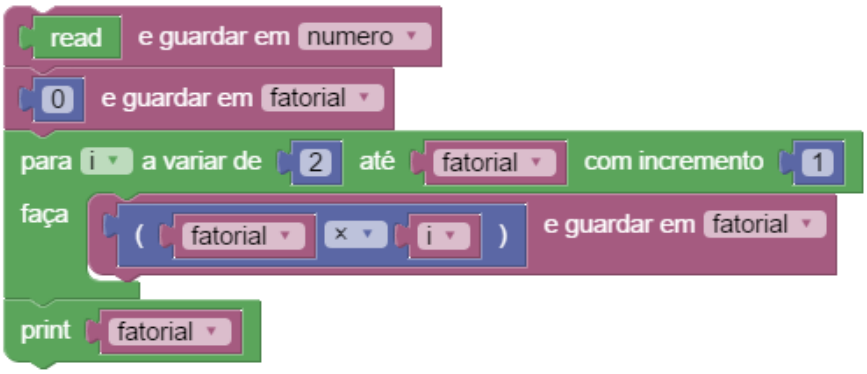
02. Analisando o programa abaixo:



Sobre ele é INCORRETO afirmar:

a) Exibirá o maior entre três números lidos, exceto se os três valores forem iguais.
b) Se forem lidos os valores 1, 3 e 6 nas variáveis n1, n2 e n3 respectivamente, a variável m receberá o valor 1, em seguida o valor 3 e, por último, o valor 6.
c) Se forem lidos os valores 7, 2 e 9 nas variáveis n1, n2 e n3 respectivamente, a variável m receberá o valor 7, em seguida o valor 2 e, por último, o valor 9.
d) Se forem lidos os valores 9, 7 e 2 nas variáveis n1, n2 e n3 respectivamente, a variável m receberá apenas o valor 9.
e) Se forem lidos os valores -1, -3 e -8 nas variáveis n1, n2 e n3 respectivamente, a variável m receberá apenas o valor -1.

03. A fórmula para calcular o fatorial de um número é :
N! = N * (n-1) * (n-2) * … * 1, Ex.: 5! = 5 * 4 *3 * 2 *1 = 120. O programa abaixo tenta resolver o fatorial de um número, porém contém 2 erros. **Reescreva o programa com os DOIS erros corrigidos**. (Não faltam blocos e nem tem a mais).



04. O índice de massa corpórea (IMC) de um indivíduo é obtido dividindo-se o seu peso (em *Kg*) pela sua altura (em *m*) ao quadrado. Assim, por exemplo, uma pessoa de 1,67m e pesando 55kg tem IMC igual a 19,72, já que:

$$IMC = \frac{peso}{altura^2} = \frac{55kg}{(167cm * 167cm)/10000} = 19{,}72$$

Como os nossos programas ficam somente com a parte inteira, o resultado é 19. Existe uma escala que classifica o indivíduo segundo o IMC: 1-Baixo peso (<=18); 2-(Normal; 19 a 25); 3-(Excesso de peso; >= 26).

O programa abaixo solicita ao usuário que forneça o seu peso em *kg* e a sua altura em c*m* e a partir deles calcula o seu índice de massa corpórea. Escreva da forma mais otimizada possível a parte final que falta para classificar o indivíduo: imprima 1, 2 ou 3, conforme a classificação explicada acima.

05. O resultado da divisão inteira de um número inteiro por outro número inteiro pode sempre ser obtido utilizando-se apenas o operador de subtração. Assim, se quisermos calcular (7/2), basta subtrair o dividendo (2) ao divisor (7), sucessivamente, até que o resultado seja menor do que o dividendo. O número de subtrações realizadas corresponde ao quociente inteiro, conforme o exemplo seguinte:

7 – 2 = 5
5 – 2 = 3
3 – 2 = 1

Escreva um programa que solicite ao usuário dois números inteiros positivos (primeiro o divisor depois o dividendo) e realize o cálculo da divisão entre eles, seguindo as etapas explicadas acima, e imprima na tela o resultado e a sobra. No exemplo acima, o resultado é 3 e o que sobra é 1. Imprima primeiro o resultado e depois imprima o que sobra. Se o usuário entrar com o número ZERO como dividendo, o programa não faz o cálculo e imprime 0.

Apêndice M

SEGUNDO EXAME APLICADO NO CICLO 4

1. Faça um programa que leia via teclado um número inteiro e exiba o dia correspondente da semana, conforme a sequência:1-Domingo, 2-Segunda, 3-Terça, 4-Quarta, 5-Quinta, 6-Sexta, 7-Sábado. Se for digitado outro valor diferente de 1 a 7, deve aparecer na tela a mensagem valor inválido.

2. Escreva um programa em Java que leia via teclado a idade de uma pessoa, determine a sua classificação segundo a seguinte tabela:

- Maior de idade;

- Menor de idade;

- Pessoa idosa (idade superior ou igual a 65 anos).

3. Escreva um programa que imprima todos os números de 1.000 a 2.000, inclusive, em ordem crescente.

4. Faça um programa em Java que leia via teclado um número *n*, inteiro e positivo, calcule e imprima a seguinte soma:

Soma = 1/1 + 1/2 + 1/3 + 1/4 + ... + 1/n

5. A Série de FETUCCINE é gerada a partir de 2 termos fornecidos pelo usuário. Considerando *n* como o enésimo termo da série, os termos seguintes são gerados com a soma ou subtração dos 2 termos anteriores, de acordo com a seguinte regra:

a) Para a posição impar do temo, deve obedecer à seguinte condição:

$A_n = A_{n-1} + A_{n-2}$, para posição impar;

b) Caso a posição do termo seja par, deve obedecer à seguinte condição:

$A_n = A_{n-1} - A_{n-2}$, para posição par.

Desenvolva um programa em Java que leia os dois primeiros termos via teclado e calcule os 10 primeiros termos da série a partir desses primeiros termos digitados pelo usuário.

Exemplo:

Termo 1 = **1**; Termo 2 = **2**

Resultado: **1-2-3-1-4-3-7-4-11-7**

Apêndice N

APRENDIZAGEM PERCECIONADA NO CICLO 4

#	Conteúdo
Q1	Eu aprendi com o jogo a raciocinar logicamente (sequência de passos)
Q2	Eu aprendi com o jogo como manipular variáveis
Q3	Eu aprendi com o jogo a usar condicionais
Q4	Eu aprendi com o jogo a usar ciclos
Q5	Eu aprendi com o jogo que facilita o meu trabalho dividir o problema em partes menores
Q6	Eu aprendi com o jogo como é importante depurar para resolver os erros
Q7	Eu aprendi coisas novas com o jogo que foram úteis na disciplina
Q8	O jogo fez-me compreender melhor alguns assuntos que eu já tinha visto
Q9	Existe grande relação entre as missões no jogo e o conteúdo das aulas
Q10	Eu recomendo usar o jogo para essa disciplina no próximo semestre.

Apêndice O

DIVERSÃO PERCECIONADA NO CICLO 4

Fator	#	Conteúdo
Controle	C1	A maior parte do meu tempo gasto no jogo foi em tarefas de aprendizagem de programação
	C2	Eu não estava sobrecarregado com tarefas não relacionadas com os objetivos da missão
	C3	A quantidade de trabalho no jogo é adequada
Clareza nos Objetivos	O1	As metas globais do jogo foram apresentadas com clareza
	O2	As metas intermédias foram apresentadas com clareza
Suporte	F1	Eu recebi suporte sobre o meu progresso durante o jogo
	F2	Eu recebi suporte imediato sobre as minhas ações
	F3	O suporte que o jogo me oferecia foi suficiente para resolver as missões
	F4	Eu compreendi as mensagens de erros que o jogo me mostrava
	F5	Eu entendi quais eram os objetivos que não consegui cumprir que o jogo me mostrava
	F6	O jogo forneceu *dicas* suficientes que me ajudaram a superar os desafios
	F7	Os e-mails enviados com *dicas* ajudaram-me a resolver os problemas nas missões.
Desafios	D1	As dificuldades nos desafios aumentam à medida que melhoram as minhas capacidades
	D2	A dificuldade dos desafios estava compatível com o meu conhecimento em programação
	D3	A dificuldade entre uma missão e a seguinte era adequada
Autonomia	A1	Eu tenho a sensação de controlo sobre o jogo
	A2	Eu compreendi o tipo de ação que todos os blocos faziam.
Imersão	I1	Eu consegui ficar envolvido no jogo
	I2	Enquanto eu jogava senti-me entusiasmado(a).
	I3	Enquanto eu jogava acabava por jogar mais tempo do que pretendia.
	I4	O contexto do jogo (lanchonete) é atrativo
	I5	Eu fiquei desmotivado com a qualidade gráfica do jogo
	I6	A música ambiente do jogo ditava o ritmo do meu trabalho

	I7	Eu fiquei motivado em jogar para obter pontos que me permitisse aceder a personalização do avatar
	I8	Eu estava motivado para ficar nas melhores posições das tabelas de classificação
	I9	Os e-mails enviados com *dicas* me motivaram a continuar jogando.

Apêndice P

INQUÉRITO FINAL NO CICLO 4

#	Conteúdo
1	As primeiras semanas de aulas com o jogo na web ajudaram-me a preparar para a aprendizagem de programação Java.
2	Os exercícios de Java integrados no NoBugs no Eclipse ajudaram-me a iniciar a programação Java.
3	Sentia-me motivado em fazer os exercícios de Java com o NoBugs no Eclipse.
4	O NoBugs em Java foi fácil de usar.
5	O NoBugs em Java preparou-me para os demais exercícios de Java. (fiz bem os exercícios que não eram em NoBugs Java)
6	Recomendo usar o NoBugs em Java no próximo semestre.
7	Descreve os pontos positivos em teres utilizado o NoBugs no Eclipse para aprender Java.
8	Cita os pontos negativos em teres utilizado o NoBugs no Eclipse e que melhorias podemos fazer tanto nas aulas, quanto na ferramenta.

Apêndice Q

EXEMPLOS DE MISSÃO EM XML

Exemplo 1

```
<?xml version='1.0'?>
<mission>
	<explanation hasInstruction="false">

		<page type="goal">
		<![CDATA[
J&#225; que consegues anotar tudo e ainda entregar comida ao cliente, ent&#227;o
vou te desafiar a
  entregar bebida. No menu existe um novo bloco para perguntar o que deseja beber
e outro bloco
  para pegar a bebida na geladeira. Esse
  bloco segue o mesmo princ&#237;pio para pegar o cachorro-quente no mostrador.
Vejamos se
  consegues desenrascar desse problema. <br/><br/>
Tens um &#250;nico cliente com fome e com sede. Pergunte o que ele deseja.
V&#225; aos locais para
  pegar o pedido. Volte ao cliente para fazer <b>as duas entregas</b>: da comida
e da bebida.
		]]>
		</page>
	</explanation>

	<hints>
		<sequence>
		</sequence>

		<errors>
			<hint category="LastError" time="0"
condition="Hints.blockTypeLastError() == 'do_deliver'" >
			<![CDATA[Um erro comum &#233; o uso da vari&#225;vel
incorreta: guardou os produtos em uma vari&#225;vel e est&#225; a fazer a entrega
			a partir de outra vari&#225;vel. Verifique em que
vari&#225;vel o bloco <imghex id="m5_i2">IMAGEM REPRESENTADA EM
HEXADECIMAL</imghex>
			guardou o valor e use a mesma vari&#225;vel como
par&#226;metro para o bloco <span style="color:white; background-color:
rgb(91,165,140); padding: 3px">deliver</span>.
			]]>
			</hint>

			<hint category="LastError" time="0"
condition="Hints.blockTypeLastError() == 'variables_set' &amp;&amp;
Hints.containsBlockTypeLastError('prepare_pickUpHotDog') == true">
```

```
                    <![CDATA[Um erro comum &#233; o uso da vari&#225;vel
incorreta: guardou o pedido em uma vari&#225;vel e est&#225; a pegar o produto
com
                    base em outra vari&#225;vel. Verifique em que vari&#225;vel o
bloco <imghex id="m5_i1"> IMAGEM REPRESENTADA EM HEXADECIMAL </imghex> guardou o
valor e use a mesma vari&#225;vel como
                    par&#226;metro deste <imghex id="m5_i2"> IMAGEM REPRESENTADA
EM HEXADECIMAL</imghex>.
                    ]]>
                    </hint>

                    <hint category="LastError" time="0"
condition="Hints.blockTypeLastError() == 'variables_set' &amp;&amp;
Hints.containsBlockTypeLastError('prepare_pickUpDrink') == true">
                    <![CDATA[Um erro comum &#233; o uso da vari&#225;vel
incorreta: guardou o pedido em uma vari&#225;vel e est&#225; a pegar o produto
com
                    base em outra vari&#225;vel. Verifique em que vari&#225;vel o
bloco <span style="color:white; background-color: rgb(91,165,140); padding:
3px">askForDrink</span> guardou o valor e use a mesma vari&#225;vel como
                    par&#226;metro deste <span style="color:white; background-
color: rgb(91,165,140); padding: 3px">pickUpDrink</span>.
                    ]]>
                    </hint>
             </errors>
      </hints>

      <help>
             <line>V&#225; at&#233; o cliente2</line>
             <line>Pergunte o que ele deseja comer e guarde na vari&#225;vel
[com]</line>
             <line>Pergunte o que ele deseja beber e guarde na vari&#225;vel
[beb]</line>

             <line>V&#225; at&#233; o mostrador</line>
             <line>Pegue a comida informada em [com] e guarde na vari&#225;vel
[com]</line>

             <line>V&#225; at&#233; a geladeira</line>
             <line>Pegue a bebida informada em [beb] e guarde na vari&#225;vel
[beb]</line>

             <line>V&#225; at&#233; o cliente 2</line>
             <line>Entregue [com]</line>
             <line>Entregue [beb]</line>
      </help>

      <commands>
             <category name="snackMan" show="true" />

             <category name="goToBarCounter" show="true" />
             <category name="goToDisplay" show="true" />
             <category name="askForFood" show="true" />
             <category name="goToCooler" show="true" />
             <category name="askForDrink" show="true" />
             <category name="pickUpHotDog" show="true" />
             <category name="pickUpDrink" show="true" />
             <category name="deliver" show="true" />

             <category name="vars" show="true" />
      </commands>

      <cooker>initial</cooker>
      <customers>
             <customer>
                    <id>01</id>
                    <init>counter2</init>
```

```
                    <dest>counter2</dest>
                    <orders>
                          <order>
                                <foods>
                                      <food qt="1" price="0">hotdog</food>
                                </foods>
                                <drinks>
                                      <drink qt="1" price="0">coke</drink>
                                </drinks>
                          </order>
                    </orders>
              </customer>

       </customers>

       <objectives xpIndividual="20" xpFinal="10" xpTotalRun="9"
buttonRunQtdAttempts="5" >
              <objective pos="2" place="counter">askForFood</objective>
              <objective pos="2" place="counter">askForDrink</objective>

              <objective pos="2" place="counter">pickUpFood</objective>
              <objective pos="2" place="counter">pickUpDrink</objective>

              <objective pos="2" place="counter">deliver</objective>
       </objectives>

       <xml xmlns="http://www.w3.org/1999/xhtml">
       </xml>
</mission>
```

Exemplo 2

```
<?xml version='1.0'?>
<mission useIfMutator="false">
       <explanation hasInstruction="false">
              <page type="goal">
              <![CDATA[
Vamos aprender o verdadeiro prop&#243;sito do bloco <i>repita-enquanto</i>.
  Podemos ler esse bloco como <b>repita enquanto a condi&#231;&#227;o for
verdadeira</b>.
  Essa &#233; a diferen&#231;a entre um tipo de ciclo e outro. Enquanto o bloco
<i>para</i>
  era usado para repetir uma quantidade de vezes, esse &#233; usado para repetir
enquanto
  uma condi&#231;&#227;o for verdadeira.
  ]]>
              </page>
              <page type="goal">
              <![CDATA[
Retornando a um exemplo j&#225; visto, na imagem abaixo podemos ler como "repita
enquanto i &lt;= 10".
  Quando tiver um valor maior que 10, e a execu&#231;&#227;o chegar a essa
compara&#231;&#227;o, ela
  retorna falso, e sai do ciclo.<br/><br/>
              <span style="text-align: center; width: 100%; display:block">
              <imghex id="while_for" style="height:180px"> IMAGEM REPRESENTADA EM
HEXADECIMAL </imghex>
              </span>
              ]]>
              </page>
              <page type="goal">
              <![CDATA[
 &#201; comum que os problemas com blocos <i>repita-enquanto</i> mudem o valor da
   vari&#225;vel dentro do ciclo conforme a necessidade, e n&#227;o com
incremento fixo de 1 em 1
   como vinha acontecendo com o bloco <i>para</i>. <br/><br/>
              <span style="text-align: center; width: 100%; display:block">
```

```
              <imghex id="while" style="height:120px"> IMAGEM REPRESENTADA EM
HEXADECIMAL</imghex>
             </span>
             <br/>
Por exemplo, no ciclo da imagem acima em vez da vari&#225;vel <i>i</i>
   ser incrementada em 1 ela pode ser incrementada com o valor
   de outra vari&#225;vel como na imagem abaixo.<br/><br/>
              <span style="text-align: center; width: 100%; display:block">
               <imghex id="soma_vars" style="height:62px"> IMAGEM REPRESENTADA EM
HEXADECIMAL</imghex>
              </span>
             ]]>
             </page>

             <page type="goal">
             <![CDATA[
Nessa miss&#227;o vais falar a quantidade total de cachorros desejada pelos
clientes.<br/><br/>
Por&#233;m, quando essa quantidade tiver pelo menos atingido 4 unidades, n&#227;o
  deves ir perguntar mais nada a cliente algum. Logo, quando chegar a um cliente,
  fazes todas as perguntas necess&#225;rias, e      calculas a quantidade, como
vinhas fazendo.
  Entretanto, s&#243; vais ao pr&#243;ximo cliente, se a quantidade total ainda
  n&#227;o tiver atingido 4.<br/><br/>
&#201; garantido que a quantidade total de cachorros entre os tr&#234;s clientes
   sempre atingir&#225; no m&#237;nimo 4.
             ]]>
             </page>


      </explanation>
      <hints>
             <sequence>
             </sequence>

             <errors>
             </errors>
      </hints>
      <commands>
      </commands>
      <cooker>initial</cooker>

      <tests>3</tests>
      <testsvars>
             <test>
                    <var name="total">5</var>
             </test>
             <test>
                    <var name="total">6</var>
             </test>
             <test>
                    <var name="total">4</var>
             </test>
      </testsvars>
      <customers>
             <customer>
                    <id>01</id>
                    <init>counter1</init>
                    <dest>counter1</dest>
                    <orders>
                           <order>
                                  <foods>
                                         <food qt="1" price="2">hotdog</food>
                                         <food qt="1"
price="1">icecreamofchocolate</food>
                                  </foods>
                                  <drinks>
```

```
                                </drinks>
                        </order>
                        <order>
                                <foods>
                                        <food qt="1" price="2">hotdog</food>
                                        <food qt="1" price="2">hotdog</food>
                                        <food qt="1" price="2">hotdog</food>
                                </foods>
                                <drinks>
                                </drinks>
                        </order>
                        <order>
                                <foods>
                                        <food qt="1"
price="1">icecreamofchocolate</food>
                                </foods>
                                <drinks>
                                        <drink qt="1" price="3">coke</drink>
                                        <drink qt="1" price="3">coke</drink>
                                </drinks>
                        </order>
                </orders>
        </customer>

        <customer>
                <id>02</id>
                <init>counter2</init>
                <dest>counter2</dest>
                <orders>
                        <order>
                                <foods>
                                        <food qt="1" price="2">hotdog</food>
                                        <food qt="1" price="2">hotdog</food>
                                        <food qt="1"
price="1">icecreamofchocolate</food>
                                </foods>
                                <drinks>
                                </drinks>
                        </order>
                        <order>
                                <foods>
                                        <food qt="1" price="2">hotdog</food>
                                        <food qt="1" price="2">hotdog</food>
                                        <food qt="1" price="2">hotdog</food>
                                </foods>
                                <drinks>
                                </drinks>
                        </order>
                        <order>
                                <foods>
                                        <food qt="1" price="2">hotdog</food>
                                        <food qt="1" price="2">hotdog</food>
                                        <food qt="1" price="2">hotdog</food>
                                </foods>
                                <drinks>
                                </drinks>
                        </order>
                </orders>
        </customer>

        <customer>
                <id>03</id>
                <init>counter3</init>
                <dest>counter3</dest>
                <orders>
                        <order>
                                <foods>
                                        <food qt="1" price="2">hotdog</food>
```

```
                                    <food qt="1" price="2">hotdog</food>
                              </foods>
                              <drinks>
                                    <drink qt="1" price="3">coke</drink>
                              </drinks>
                        </order>
                        <order>
                              <foods>
                                    <food qt="1" price="2">hotdog</food>
                              </foods>
                              <drinks>
                                    <drink qt="1" price="3">coke</drink>
                              </drinks>
                        </order>
                        <order>
                              <foods>
                                    <food qt="1" price="2">hotdog</food>
                              </foods>
                              <drinks>
                              </drinks>
                        </order>
                  </orders>
            </customer>
      </customers>
      <objectives missionType="sort" xpIndividual="10" xpFinal="5"
xpTotalRun="3">
            <objective text="o total de cachorros"
value="parseInt(Game.readVariableTest('total'))">talk</objective>
      </objectives>
      <xml>
            <block type="controls_whileUntil" deletable="false" x="16" y="2">
                  <value name="BOOL">
                        <block type="logic_compare" deletable="false"
movable="false">
                              <field name="OP">LT</field>
                              <value name="A">
                                    <block type="variables_get"
deletable="false" movable="false">
                                          <field
name="VAR">totalCachorros</field>
                                    </block>
                              </value>
                              <value name="B">
                                    <block type="math_number"
deletable="false" movable="false">
                                          <field name="NUM">4</field>
                                    </block>
                              </value>
                        </block>
                  </value>
            </block>
            <block type="variables_set" deletable="false" x="17" y="95">
                  <field name="VAR">totalCachorros</field>
                  <value name="VALUE">
                        <block type="math_number" deletable="false"
movable="false">
                              <field name="NUM">0</field>
                        </block>
                  </value>
            </block>
            <block type="variables_set" deletable="false" x="305" y="96">
                  <field name="VAR">sorvetes</field>
                  <value name="VALUE">
                        <block type="ask_askWantHowManyIceCream"
deletable="false" movable="false"></block>
                  </value>
            </block>
```

```
            <block type="variables_set" deletable="false" x="21" y="162">
                  <field name="VAR">totalCachorros</field>
                  <value name="VALUE">
                        <block type="math_arithmetic" deletable="false"
movable="false">
                              <field name="OP">ADD</field>
                              <value name="A">
                                    <block type="variables_get"
deletable="false" movable="false">
                                          <field
name="VAR">cachorrosDoCliente</field>
                                    </block>
                              </value>
                              <value name="B">
                                    <block type="variables_get"
deletable="false" movable="false">
                                          <field
name="VAR">totalCachorros</field>
                                    </block>
                              </value>
                        </block>
                  </value>
            </block>
            <block type="variables_set" deletable="false" x="22" y="241">
                  <field name="VAR">cachorrosDoCliente</field>
                  <value name="VALUE">
                        <block type="math_arithmetic" deletable="false"
movable="false">
                              <field name="OP">MINUS</field>
                              <value name="A">
                                    <block type="ask_askWantHowManyFoods"
deletable="false" movable="false"></block>
                              </value>
                              <value name="B">
                                    <block type="variables_get"
deletable="false" movable="false">
                                          <field name="VAR">sorvetes</field>
                                    </block>
                              </value>
                        </block>
                  </value>
            </block>
            <block type="move_goToBarCounter" deletable="false" x="17" y="321">
                  <value name="VALUE">
                        <block type="variables_get" deletable="false"
movable="false">
                              <field name="VAR">cliente</field>
                        </block>
                  </value>
            </block>
            <block type="do_talk" deletable="false" x="280" y="320">
                  <value name="VALUE">
                        <block type="variables_get" deletable="false"
movable="false">
                              <field name="VAR">totalCachorros</field>
                        </block>
                  </value>
            </block>
            <block type="variables_set" deletable="false" x="19" y="384">
                  <field name="VAR">cliente</field>
                  <value name="VALUE">
                        <block type="math_number" deletable="false"
movable="false">
                              <field name="NUM">1</field>
                        </block>
                  </value>
            </block>
            <block type="variables_set" deletable="false" x="268" y="377">
```

```
                    <field name="VAR">cliente</field>
                    <value name="VALUE">
                          <block type="math_arithmetic" deletable="false"
movable="false">
                                <field name="OP">ADD</field>
                                <value name="A">
                                      <block type="math_number"
deletable="false" movable="false">
                                            <field name="NUM">1</field>
                                      </block>
                                </value>
                                <value name="B">
                                      <block type="variables_get"
deletable="false" movable="false">
                                            <field name="VAR">cliente</field>
                                      </block>
                                </value>
                          </block>
                    </value>
              </block>
       </xml>

</mission>
```