INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA
DE MINAS GERAIS – *CAMPUS* BAMBUÍ
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

Julia Gabriella Corrêa Silva

DESENVOLVIMENTO DE UM OBJETO VIRTUAL DE APRENDIZAGEM PARA O ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS: uma abordagem lúdica e interativa

Bambuí - MG
2024

JULIA GABRIELLA CORRÊA SILVA

DESENVOLVIMENTO DE UM OBJETO VIRTUAL DE APRENDIZAGEM PARA O ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS: uma abordagem lúdica e interativa

Trabalho de conclusão de curso apresentado ao Curso de Bacharelado em Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Cláudio Ribeiro de Sousa.

Bambuí - MG

2024

Catalogação na Fonte Biblioteca IFMG - Campus Bambuí

S586d Silva, Julia Gabriella Corrêa.
Desenvolvimento de um objeto virtual de aprendizagem para o ensino de programação orientada a objetos: uma abordagem lúdica e interativa - Campus Bambuí. / Julia Gabriella Corrêa Silva. – 2024.
78 f. : il. ; color.

Orientador: Cláudio Ribeiro de Sousa.
Trabalho de Conclusão de Curso (graduação) - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí, MG, Curso Bacharelado em Engenharia de Computação, 2024.

1. Programação orientada a objetos. 2. Jogo virtual educativo. 3. Unity. I. Sousa, Cláudio Ribeiro de. II. Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí, MG. III. Título.

CDD 005.636

Elaborada por Douglas Bernardes de Castro- CRB-6/2802

Julia Gabriella Corrêa Silva

DESENVOLVIMENTO DE UM OBJETO VIRTUAL DE APRENDIZAGEM PARA O ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS: uma abordagem lúdica e interativa

Trabalho de conclusão de curso apresentado ao Curso de Bacharelado em Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Cláudio Ribeiro de Sousa.

Aprovado em 02/04/2024 pela banca examinadora:

Documento assinado eletronicamente por **Claudio Ribeiro de Sousa**, **Professor EBTT**, em 02/04/2024, às 17:03, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Gabriel da Silva**, **Professor**, em 02/04/2024, às 17:04, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Marcos Roberto Ribeiro**, **Professor**, em 02/04/2024, às 17:05, conforme Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade do documento pode ser conferida no site https://sei.ifmg.edu.br/consultadocs informando o código verificador **1869271** e o código CRC **A50FC27E**.

Bambuí - MG

2024

AGRADECIMENTOS

Agradeço a Deus por estar presente em cada etapa desta jornada acadêmica.

À minha família, especialmente aos meus pais, Keila e Erivelto, e à minha irmãzinha Alice, cujo amor incondicional, suporte e dedicação foram o alicerce sobre o qual construí todo meu caminho de crescimento e aprendizado.

Aos meus amigos, cuja amizade e apoio foram fundamentais para manter minha determinação ao longo desta jornada.

Ao meu orientador, pela paciência e ensino de qualidade, que foram fundamentais para a realização deste trabalho. Suas orientações contribuíram para o meu crescimento acadêmico e profissional.

Aos demais professores que contribuíram para a minha formação.

A todos que, de alguma forma, contribuíram para esta jornada, direta ou indiretamente, o meu sincero agradecimento. Este trabalho não teria sido possível sem o apoio e incentivo de cada um de vocês. Obrigada por fazerem parte desta jornada.

RESUMO

A relevância da programação e a crescente necessidade de habilidades nesse domínio têm sido cada vez mais evidentes. No entanto, é comum encontrar estudantes enfrentando dificuldades e desmotivação ao longo do processo de aprendizagem da programação. Na programação orientada a objetos, é crucial internalizar não apenas a sintaxe da linguagem, mas também os conceitos e a lógica por trás da organização modular e da abstração de dados. Além disso, a falta de prática e *feedback* adequado pode tornar o processo desmotivador para muitos estudantes. O estudo sobre o uso de jogos como forma de aprendizado tem se tornado cada vez mais relevante e difundido nas áreas da educação e tecnologia. Assim, este trabalho teve como propósito a criação de um jogo virtual como ferramenta de suporte ao ensino-aprendizagem dos conceitos principais da programação orientada a objetos. Durante o progresso, foi elaborada uma história simples, ambientada em um contexto medieval, e aplicadas técnicas de gamificação. O jogo foi desenvolvido na plataforma Unity e estruturado em três fases de dificuldade progressiva, cada uma introduzindo e explorando conceitos específicos que são ensinados através da narrativa. Ao finalizar o desenvolvimento, foram conduzidos testes de caixa-preta e correções das falhas, para garantir a qualidade e a integridade do sistema. Os resultados mostraram uma possibilidade futura de conduzir testes em uma disciplina, o que poderia ser realizado por meio da implementação de um estudo de caso com um grupo de estudantes para fortalecer a validação da ferramenta.

Palavras-chave: Programação orientada a objetos. Jogo virtual educativo. Unity. Desenvolvimento de jogos.

ABSTRACT

The relevance of programming and the growing need for skills in this domain have become increasingly evident; however, it is common to find students facing difficulties and lack of motivation throughout the programming learning process. In object-oriented programming, it is crucial to internalize not only the language syntax but also the concepts and logic behind modular organization and data abstraction. Furthermore, the lack of practice and adequate feedback can make the process demotivating for many students. The study of using games as a learning tool has become increasingly relevant and widespread in the fields of education and technology. Therefore, this work aimed to create a digital game as a tool to support the teaching and learning of key concepts in object-oriented programming. During the development, a simple story set in a medieval context was developed, and gamification techniques were applied. The game was developed on the Unity platform and structured into three phases of progressive difficulty, each introducing and exploring specific concepts taught through the narrative. Upon completing the development, black-box tests were conducted, and flaws were corrected to ensure the quality and integrity of the system. The results show a future possibility of conducting tests in a course, which could be achieved through the implementation of a case study with a group of students to strengthen the validation of the tool.

Keywords: Object-oriented programming. Educational virtual game. Unity. Game development.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

DSR – *Design Science Research*

NPC – Personagem Não Jogável

OA – Objeto de Aprendizagem

POO – Programação Orientada a Objetos

RPG – *Role Playing Game*

TIC – Tecnologias de Informação e Comunicação

UI – Interface de Usuário

UML – Linguagem de Modelagem Unificada

SUMÁRIO

1 INTRODUÇÃO

Segundo Albuquerque (2021), aprender programação pode ser uma tarefa árdua para estudantes iniciantes em cursos de Engenharia e Tecnologia. Os professores estão cientes das dificuldades enfrentadas pelos alunos ao tentarem compreender os aspectos fundamentais para a construção de programas. Algumas das principais dificuldades incluem a falta de compreensão dos conceitos básicos da lógica de programação e a falta de prática e de um ambiente de aprendizagem adequado.

Com o intuito de auxiliar os alunos, educadores buscam empregar métodos de ensino eficazes que utilizem recursos interativos e práticos para facilitar a compreensão e melhorar o desenvolvimento de suas habilidades de programação. O uso de jogos educativos tem se mostrado uma opção frequente para despertar a curiosidade e o interesse dos estudantes, incentivando-os a aprender, questionar e aprimorar novas habilidades e áreas de conteúdo (SILVA; QUEIROZ, 2014). Os jogos educativos podem ser projetados para ensinar conceitos fundamentais de programação, como lógica, estruturas de dados, algoritmos e paradigmas, de maneira mais atraente.

Por meio de jogos educativos, os estudantes podem aprender de forma interativa, colocando em prática os conceitos ensinados. Essa abordagem cria um ambiente estimulante onde os alunos podem explorar, experimentar e aplicar conceitos de programação de maneira lúdica, levando a um entendimento mais profundo e a uma maior motivação para aprender (SILVA; QUEIROZ, 2014).

Quando se trata de desenvolver um jogo na área da computação, é crucial considerar a sua adaptabilidade. Afinal, quanto mais o jogo puder ser portado para diferentes plataformas, maior será o alcance do público e, consequentemente, maior será a influência que ele pode ter em seu objetivo. No caso de jogos educacionais, a acessibilidade se torna ainda mais importante, pois isso pode resultar em uma ferramenta didática envolvente, permitindo que um maior número de alunos e professores aproveitem os benefícios (PIMENTA, 2022).

O presente trabalho teve como propósito o desenvolvimento de um objeto de aprendizagem (OA) em formato de jogo virtual educativo, visando promover a compreensão dos conceitos fundamentais da programação orientada a objetos (POO) de maneira interativa e divertida. Para isso, foi utilizada uma *game engine* como ferramenta principal, juntamente com a incorporação de elementos da gamificação durante o processo de desenvolvimento do jogo.

1.1 Objetivos

Nesta seção, são apresentados o objetivo geral e os objetivos específicos que direcionaram a realização do trabalho.

1.1.1 Objetivo Geral

O objetivo geral do presente trabalho foi desenvolver um jogo virtual educativo com o propósito de tornar o processo de aprendizagem mais eficiente e interessante para os alunos da área de computação, que já possuem conhecimentos básicos em lógica de programação, ao ensinar os conceitos da disciplina de programação orientada a objetos.

1.1.2 Objetivos Específicos

Para alcançar o objetivo geral, foi necessário cumprir os seguintes objetivos específicos:

- Identificar os principais conceitos de programação orientada a objetos;
- Pesquisar e selecionar os elementos de gamificação utilizados no objeto virtual de aprendizagem;
- Escolher a plataforma e implementar o *software* do jogo educativo;
- Realizar testes e validação do *software* implementado.

1.1.3 Resultados Esperados

Espera-se que, com a finalização deste trabalho, o jogo educativo proposto possa ser utilizado como uma ferramenta complementar ao ensino tradicional de programação orientada a objetos, oferecendo uma abordagem inovadora e eficiente para auxiliar no aprendizado dos alunos. Além disso, espera-se que o jogo estimule o interesse e a motivação dos estudantes pelo aprendizado da disciplina, aumentando a compreensão e a fixação dos conhecimentos adquiridos.

1.2 Justificativa

A programação orientada a objetos é um dos principais conceitos da computação. No entanto, muitos alunos enfrentam dificuldades em compreender seus conceitos teóricos, o que pode prejudicar seu processo de aprendizado e desmotivá-los. Para Kölling (1999), um

problema comum no ensino de programação orientada a objetos são as ferramentas disponíveis, que, muitas vezes, são confusas e complexas. Assim, profissionais da área da educação estão cada vez mais empenhados em desenvolver abordagens de ensino que estimulem o envolvimento dos alunos no processo de aprendizagem de POO de maneira didática e intuitiva.

Considerando os problemas citados, o presente trabalho desenvolve um jogo educativo com elementos de gamificação para tornar o processo de aprendizado mais atraente e envolvente para os alunos de computação que já possuem conhecimento básico em lógica de programação, contribuindo, assim, para a compreensão dos conceitos de programação orientada a objetos de forma lúdica e interativa.

1.3 Estrutura do documento

A estrutura deste trabalho foi dividida em seis capítulos. O Capítulo 2 aborda o referencial bibliográfico; no Capítulo 3, apresenta-se a metodologia do trabalho; o Capítulo 4 expõe o desenvolvimento do *software*; no Capítulo 5, são mostrados os resultados; e o Capítulo 6 contém as conclusões obtidas.

2 REFERENCIAL TEÓRICO

Neste capítulo, são abordados os conceitos relacionados aos temas e referências principais do presente trabalho, incluindo os conceitos de linguagens de programação, programação orientada a objetos, objetos virtuais de aprendizagem, gamificação, jogos educacionais, ferramentas e métodos utilizados no desenvolvimento do jogo e o estado da arte.

2.1 Linguagem de Programação

As linguagens de programação e os algoritmos são elementos essenciais no campo da Ciência da Computação e no Desenvolvimento de *Software*. Os algoritmos são sequências de passos lógicos que permitem a resolução de algum problema. Segundo Farrer (1999), para que um computador possa executar operações seguindo comandos, é necessário programar um algoritmo e armazená-lo na memória, ou seja, realizar a transcrição para uma linguagem compreensível pelo computador. Os computadores executam diretamente somente os algoritmos escritos em linguagem de máquina.

O uso de linguagens de programação é uma forma estruturada de criar um código de computador, seguindo um conjunto específico de regras de sintaxe e semântica. Esse código, que pode ser compilado em um programa, contém instruções para direcionar o processamento de um computador (SANTOS, 2003).

Um paradigma de programação é um conjunto de conceitos e técnicas que definem uma metodologia e um estilo de desenvolvimento de *software*, apresentando uma maneira de estruturar e solucionar problemas usando uma linguagem de programação. Existem vários paradigmas, como procedural, orientado a objetos, funcional e lógico. Cada paradigma tem suas próprias características, *design* e modelos de execução que afetam a maneira como os programas são estruturados, organizados e executados. A escolha do paradigma adequado depende das necessidades do projeto, do problema a ser resolvido e das preferências do desenvolvedor (SEBESTA, 2011).

Para este projeto, foi escolhido abordar os conceitos do paradigma de programação orientada a objetos devido à sua popularidade e aos benefícios proporcionados pela criação de programas mais estruturados e reutilizáveis. De acordo com a pesquisa realizada pelo IEEE Spectrum (2023), algumas das principais linguagens de programação de 2023 foram Python, Java e C++, as quais incorporam a programação orientada a objetos, demonstrando a

relevância desse paradigma no mercado de desenvolvimento de *software*.

2.2 Programação Orientada a Objetos

De acordo com Santos (2003), a programação orientada a objetos (POO) é um paradigma que utiliza classes e objetos para representar e manipular dados em programas computacionais. Esses dados podem ser representações de pessoas, itens, tarefas, processos, conceitos, ideias, entre outros. A POO apresenta recursos como a herança, que permite o compartilhamento de características entre classes; o polimorfismo, que possibilita o tratamento uniforme de objetos de diferentes classes; e o encapsulamento, que oculta as informações internas. Além disso, é possível estruturar o código de forma modular, proporcionando maior flexibilidade, segurança, reutilização e a manutenção do sistema.

Segundo Varejão (2004, p. 20), "com o avanço da computação, os sistemas de *software* têm se tornado cada vez maiores e mais complexos. O paradigma orientado a objetos oferece conceitos que visam tornar mais rápido e confiável o desenvolvimento desses sistemas".

Os conceitos fundamentais do paradigma de programação orientada a objetos foram introduzidos pelos noruegueses Ole-Johan Dahl e Kristen Nygaard em 1967. Os pesquisadores desenvolveram uma linguagem de programação chamada Simula 67, com o objetivo de permitir a modelagem e a simulação de sistemas complexos, contendo conceitos como classes, objetos e herança (DAHL, MYHRHAUG e NYGAARD, 1968).

O Simula 67 influenciou outras linguagens de programação, resultando no surgimento de linguagens modernas e populares, como Java, Delphi, C++, C#, Ruby e Python. Embora o paradigma de POO seja considerado antigo, sua aplicação no mercado de trabalho se tornou mais relevante devido à crescente variedade de linguagens utilizadas pelas grandes empresas (MONQUEIRO, 2007).

Apesar de a POO ser um paradigma de programação amplamente utilizado nos cursos de computação e nas empresas, seu estudo ainda é desafiador para muitos alunos, especialmente aqueles que já possuem experiência com programação procedural. Isso ocorre devido à necessidade de aprender novas formas de abstração e de pensar em termos de objetos e suas interações (KÖLLING, 1999).

2.3 Objeto de Aprendizagem

Segundo Braga *et al.* (2012), as dinâmicas de ensino e aprendizagem estão sendo afetadas diretamente pelo aumento da utilização das Tecnologias de Informação e Comunicação (TIC), resultando em novos modelos pedagógicos. As TICs são um conjunto de recursos tecnológicos utilizados para obter, armazenar, transmitir e manipular informações. São aplicadas em diversos setores, incluindo indústria, comércio e educação. O avanço das TICs na educação vem tornando os ambientes educacionais e os recursos digitais essenciais para a aprendizagem, seja ela presencial ou a distância.

Nesse contexto, segundo Wiley (2000, p.23), os objetos de aprendizagem são definidos como "qualquer recurso digital que pode ser reutilizado para apoiar a aprendizagem". Esses objetos podem ser utilizados como ferramentas digitais para auxiliar na criação de novas estratégias, proporcionando vantagens tanto para professores quanto para alunos em diferentes ambientes e contextos de ensino.

A ideia dos objetos de aprendizagem surgiu em 1990, impulsionada por uma experiência pessoal de Wayner Hodgins ao observar seus filhos brincando com blocos de *Lego*. Hodgins percebeu as diferentes preferências de cada um dos seus filhos em relação ao aprendizado, enquanto um preferia seguir instruções para alcançar um resultado, o outro valorizava a liberdade e a criatividade na construção. Essa observação levou o autor a idealizar um mundo onde as pessoas poderiam criar ou usar objetos de aprendizagem pré-montados, seguindo instruções ou determinando seus próprios resultados em contextos educacionais (HODGINS, 2002).

Existem diversas formas de criar OAs e, para garantir a eficiência desses recursos no processo de ensino e permitir sua reutilização, parcial ou total, em atividades futuras, é preciso produzi-los com base em processos tecnológicos e pedagógicos apropriados (BRAGA *et al.*, 2012). Portanto, é importante ressaltar que os OAs podem ser áudios, imagens digitais, textos, mapas, vídeos, simuladores, hipertextos, programas, entre outros, desde que tenham a capacidade de auxiliar na assimilação de conceitos educacionais (REBOUÇAS; MAIA; SCAICO, 2021).

Os processos técnicos dos objetos de aprendizagem abrangem diversas características fundamentais que garantem a sua efetividade, tais como: a acessibilidade, que se refere à possibilidade de utilização remota em diversos locais; a agregação, que permite agrupamento com outros recursos; a autonomia, que verifica se o objeto pode ser utilizado de forma

independente; a classificação, que possibilita catalogar e facilitar a identificação; a durabilidade, que avalia se os recursos podem ser usados mesmo após mudanças; a interoperabilidade, que verifica se podem ser utilizados em diferentes ambientes; a reusabilidade, que verifica a capacidade de incorporá-los em diversas aplicações (GALAFASSI *et al.*, 2014).

Os processos pedagógicos apropriados dos objetos de aprendizagem abrangem diversos aspectos fundamentais, tais como: a interatividade, que verifica a interação do aluno com o conteúdo; a autonomia, que promove a iniciativa e a tomada de decisão do aluno, eliminando a dependência dos professores; a cooperação, que permite aos alunos a troca de opiniões e o trabalho coletivo; a cognição, que se refere à carga cognitiva imposta para o aluno; a afetividade, que é responsável pelos sentimentos e motivações do aluno em relação à sua aprendizagem durante a interação (GALAFASSI *et al.*, 2014).

2.4 Gamificação

A gamificação é um termo utilizado para descrever a agregação de elementos de jogos, como mecânicas e características de *design* em contextos que não são tipicamente relacionados a jogos. Esse conceito tem sido amplamente aplicado em áreas como educação, negócios e entretenimento, com o propósito de aumentar a motivação e o engajamento dos usuários em atividades que, de outra forma, poderiam ser menos interessantes (DETERDING *et al.*, 2011).

A gamificação, para Albertazzi, Ferreira e Forcellini (2018), não está relacionada exclusivamente à concepção de um jogo, mas consiste em adicionar atributos de *design* para proporcionar experiências lúdicas, usando elementos ou experiências específicas de jogos para atingir seus objetivos.

O termo “gamificação” foi utilizado pela primeira vez em 2008 em uma postagem de blog na *internet* e se popularizou somente em 2010 (DETERDING *et al.*, 2011). No entanto, o uso de elementos de jogos em contextos não relacionados ao entretenimento não é algo novo. Um exemplo histórico ocorreu na União Soviética durante o período de Lenin, em 1917, em que competições socialistas eram empregadas para incentivar os trabalhadores a aumentar a produção (NELSON, 2012).

De acordo com Kapp (2012), a gamificação pode ser classificada em duas categorias: estrutural e de conteúdo. Na gamificação estrutural, elementos de jogos são utilizados para motivar os alunos através do conteúdo sem alterações significativas. Na gamificação de

conteúdo, a aplicação de elementos de jogos tem como objetivo alterar o conteúdo para que se torne mais parecido com um jogo. Alguns elementos que podem ser empregados no *design* da gamificação incluem jogadores, interação, desafios, *feedback*, recompensas, pontuações, níveis, regras, entre outros.

A gamificação na educação, quando devidamente implementada, pode proporcionar uma experiência de aprendizagem mais imersiva e envolvente, permitindo que os estudantes experimentem novos desafios e a oportunidade de aumentar a compreensão do conteúdo (HOLM; HUKE, 2020).

2.5 Jogos Educacionais

Segundo Falkembach (2006), as atividades lúdicas possuem a capacidade de entreter, prender a atenção e ensinar um conhecimento de maneira mais eficiente que os métodos convencionais de ensino. Nos jogos, por exemplo, isso ocorre porque eles transmitem informações de várias maneiras, ao mesmo tempo que estimulam diferentes sentidos, mantendo, assim, a atenção e o interesse dos usuários. Portanto, qualquer atividade que inclua um aspecto lúdico pode ser um recurso para facilitar o processo de ensino-aprendizagem.

Com o avanço tecnológico, muitos dos jogos são considerados uma nova forma de contar histórias e transmitir informações com roteiros e narrativas complexos e elaborados. Enquanto os jogos comerciais possuem diversos especialistas para narrativa, roteiro e animação, os jogos educativos, em sua maioria, não contam com essa diversidade de especialistas. No entanto, as características emocionais e lúdicas das narrativas podem gerar empatia, auxiliar na memorização de conteúdos educacionais e melhorar o consumo de conhecimento (ALVES; BATTAIOLA; CEZAROTTO, 2016).

Klopfer *et al*. (2009) identificaram cinco características fundamentais que devem estar em um jogo educativo, sendo elas: a liberdade para falhar, onde é permitido que o jogador possa realizar coisas que parecem falhas, gerando aprendizado tanto com o fracasso quanto com o sucesso; a liberdade para experimentar, essa característica se correlaciona com a liberdade para falhar, mas sugere também que o jogador tenha espaço para inventar novas abordagens e explorar diferentes atividades; a liberdade para criar identidades, onde é permitido que o jogador explore diferentes identidades no mundo ficcional, não se limitando à natureza do mundo real; a liberdade de esforço, ou seja, o jogo deve envolver momentos de intensa atividade seguidos de momentos relaxados, gerando variações no esforço do jogador; e a liberdade de interpretação, em que cada jogador tem uma forma única de vivenciar o

mesmo jogo, influenciado por motivações individuais, sociais e culturais, desafiando a padronização do contexto de aprendizagem.

2.6 *Game Engine*

O desenvolvimento de jogos é uma tarefa complexa que requer habilidades e conhecimentos multidisciplinares. As *game engines* são ferramentas desenvolvidas para simplificar esse processo, fornecendo uma estrutura que pode ser personalizada de acordo com as necessidades do projeto. Essas ferramentas fornecem módulos com objetivos específicos que facilitam a construção do jogo e integram recursos avançados de gráficos, som e interatividade, possibilitando a criação de um produto de qualidade (ALVES, 2020). Atualmente, dentre as diversas *engines* disponíveis no mercado, a Unity e a Godot são duas opções principais para este projeto.

A plataforma Unity[1] é uma *engine* disponível para Windows, Linux e Mac, que oferece suporte para jogos 2D e 3D. As linguagens de programação aceitas são C# e Javascript. Embora seja amplamente utilizada no desenvolvimento de jogos, também pode ser aplicada em outras áreas, aproveitando recursos como gráficos 3D, simulações físicas, suporte multimídia e capacidade multiplataforma (SILVA FILHO *et al.*, 2018). Segundo Cavalcante e Pereira (2018), a Unity possui uma interface completa e intuitiva, além de uma documentação extensa, com tutoriais disponíveis e uma comunidade ativa. No entanto, há algumas desvantagens, como limitações na licença gratuita e a necessidade de uma conexão estável.

A Godot é uma *engine* multiplataforma que possui uma linguagem de programação própria chamada GDScript, mas também suporta outras linguagens, como C# e C++. Oferece muitos recursos para o desenvolvimento de jogos, incluindo suporte a gráficos 2D e 3D, animações, áudio e efeitos visuais. Além disso, a Godot possui um editor de cena poderoso que permite a criação, organização e renderização de ambientes e personagens de forma simples (GODOT, 2023). Segundo Cavalcante e Pereira (2018), a Godot é uma opção gratuita de código aberto que dispensa a necessidade de instalação e oferece facilidade de uso. Contudo, apresenta menos documentação disponível, limitações em projetos complexos e possui uma comunidade menor em comparação com a Unity.

Segundo Alves (2020), a escolha de uma *game engine* é um processo complexo e cuidadoso que exige uma análise minuciosa de vários pontos relevantes do projeto. Portanto, durante o trabalho, foi realizada uma pesquisa para avaliar qual ferramenta seria mais

1 Plataforma de desenvolvimento de jogos, disponível em: Unity.

adequada para o projeto, considerando aspectos como facilidade de uso, suporte da comunidade, quantidade de recursos, desempenho da plataforma, entre outros, garantindo, dessa forma, um desenvolvimento eficiente e de alta qualidade para o jogo.

2.7 SCRUM

As metodologias ágeis são abordagens incrementais para a especificação, colaboração, desenvolvimento e entrega do *software*. Essas metodologias foram desenvolvidas como uma alternativa aos métodos tradicionais de gerenciamento de projetos, que, muitas vezes, eram rígidos e burocráticos. Possuem o objetivo de realizar alterações de forma rápida e eficiente, permitindo flexibilidade e adaptabilidade ao desenvolvimento de *software* (SOMMERVILLE, 2011). Segundo Sommerville (2011, p. 50), a "abordagem Scrum é um método ágil geral, mas seu foco está no gerenciamento do desenvolvimento iterativo, ao invés das abordagens técnicas específicas da engenharia de *software* ágil".

O Scrum "combina quatro eventos formais para inspeção e adaptação, contidos dentro de um evento, a *Sprint*. Esses eventos funcionam porque implementam os pilares empíricos do Scrum: transparência, inspeção e adaptação" (SCHWABER; SUTHERLAND, 2020, p. 4).

As *Sprints* são eventos de período de três a quatro semanas, nos quais é realizado o desenvolvimento para entregar um conjunto de itens do *Product Backlog*. A equipe se concentra exclusivamente no trabalho dos itens selecionados, realizando diariamente reuniões curtas para alinhamento. No decorrer da *Sprint*, existem alguns pontos que devem ser seguidos: não são permitidas alterações que possam afetar a meta principal; a qualidade do trabalho deve ser mantida do início ao final, entregando as atividades de forma correta, nos prazos estipulados; e o *Product Backlog* pode ser refinado, permitindo que a equipe tenha a responsabilidade de revisar e ajustar itens que forem necessários (SCHWABER; SUTHERLAND, 2020).

O *Product Backlog* é uma lista de todas as funcionalidades, requisitos e melhorias que precisam ser implementadas no produto. Esta lista pode ser atualizada e melhorada à medida que novas informações forem disponibilizadas. A ordem dos itens é baseada na prioridade, ou seja, os itens mais importantes são colocados no topo da lista (SCHWABER; SUTHERLAND, 2020).

2.8 Linguagem de Modelagem Unificada (UML)

A linguagem de modelagem unificada é uma representação visual para modelagem de *software* baseada no paradigma orientado a objetos. Consiste em vários diagramas, cada um com um propósito, incluindo diagramas de casos de uso, diagramas de classes, diagramas de sequência, diagramas de atividades, entre outros. Estes diagramas permitem que os desenvolvedores colaborem entre si, comunicando-se visualmente e compartilhando um entendimento comum do sistema (GUEDES, 2011).

"A UML é uma linguagem de modelagem cujo objetivo é auxiliar os engenheiros de *software* a definirem as características do sistema, tais como seus requisitos, seu comportamento, sua estrutura lógica e a dinâmica dos processos implantados" (GUEDES, 2011, p. 19).

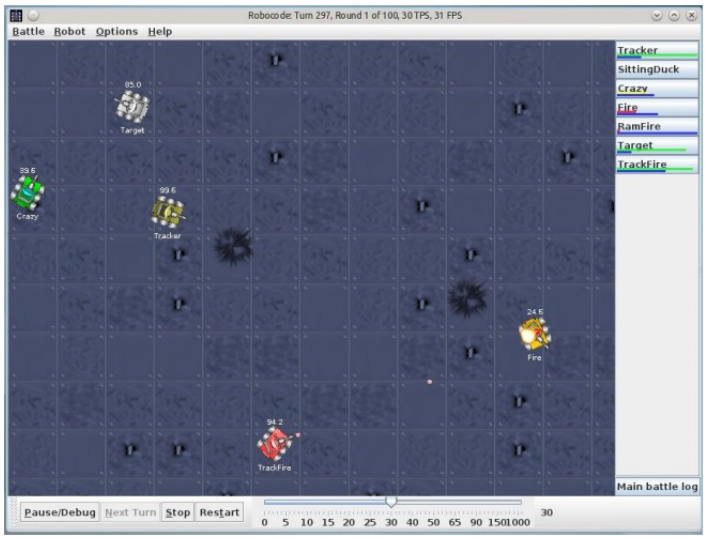
Nos últimos anos, a UML tornou-se uma das linguagens padrão de modelagem mais adotadas, reduzindo ambiguidades e erros de interpretação durante o desenvolvimento de *software*, facilitando a comunicação entre os membros da equipe, a documentação eficiente para o sistema e a possibilidade de realizar análises mais detalhadas (GUEDES, 2011).

2.9 Estado-da-arte

Rodrigues, Nogueira e Queiroga (2017) apresentam quatro ferramentas para facilitar o aprendizado de POO, sendo elas RoboCode, Greenfoot, Jogo de Tabuleiro e o aplicativo Aprendiz Digital. Essas ferramentas foram desenvolvidas com o objetivo de proporcionar aos estudantes uma experiência interativa na compreensão dos conceitos.

O Robocode é um sistema de simulação de batalha de robôs programáveis em Java, onde os alunos devem aplicar conceitos de POO para criar estratégias. A programação do robô é baseada na herança e no uso de atributos e métodos para definir seus movimentos. Na batalha, vence o robô que executar a melhor estratégia e derrotar o oponente. Os alunos enfatizaram que a ferramenta forneceu uma maneira agradável de aprender vários conceitos de programação orientada a objetos. No entanto, alguns mencionaram a dificuldade e confusão de implementar comandos, além da sugestão de não permitir o uso de robôs prontos da *internet*, devido à quantidade de robôs idênticos (RODRIGUES; NOGUEIRA; QUEIROGA, 2017). A Figura 1 apresenta uma imagem da ferramenta Robocode.

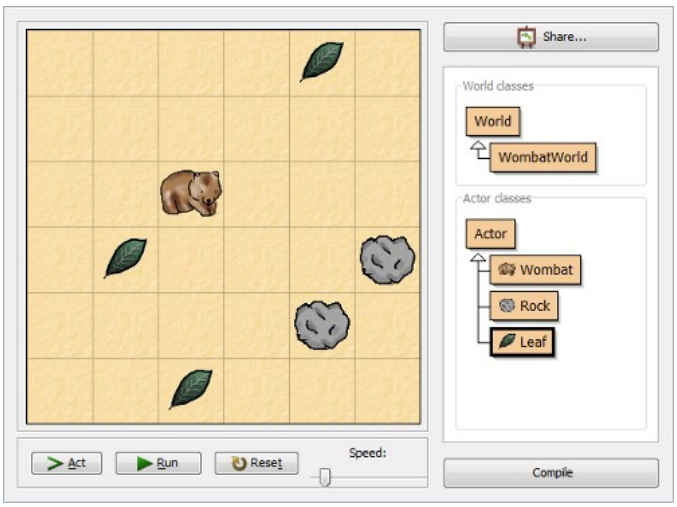
Figura 1 - Interface Robocode



Fonte: AMARAL; SILVA; PANTALEÃO, 2015.

Greenfoot é uma ferramenta que utiliza comandos predefinidos e permite a geração automática de código em Java ao arrastar e soltar itens em um ambiente virtual. Oferece suporte ao desenvolvimento de aplicações gráficas, incluindo animação, movimento, rolagem, caixas de texto, relógios e cronômetros. Durante as aulas, foram utilizados exemplos de jogos no Greenfoot para explicar conceitos básicos de POO, com foco na criação de classes, atributos e métodos (RODRIGUES; NOGUEIRA; QUEIROGA, 2017). A Figura 2 apresenta uma imagem da ferramenta Greenfoot.

Figura 2 - Interface Greenfoot



Fonte: GREENFOOT, 2023.

O jogo de tabuleiro contém 216 posições e pode ser preenchido com quatro cores diferentes, cada uma representando um tipo de atividade relacionada ao ensino de POO. Foi utilizado para organizar as atividades e notas dos alunos durante o semestre, estabelecendo equipes e regras (RODRIGUES; NOGUEIRA; QUEIROGA, 2017). A Figura 3 apresenta

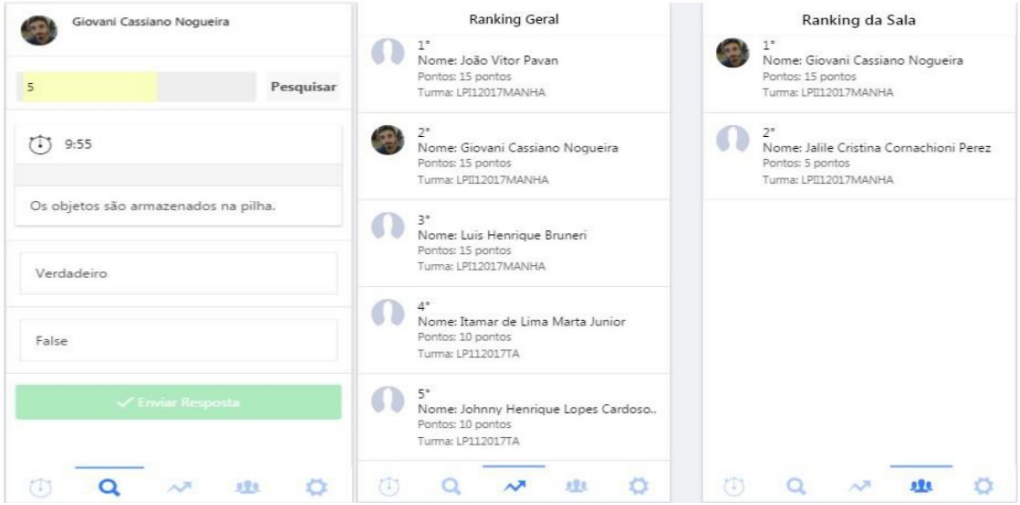
uma imagem do tabuleiro preenchido pelos alunos.

Figura 3 - Tabuleiro coloridos pelos alunos

Fonte: RODRIGUES; NOGUEIRA; QUEIROGA, 2017.

Por fim, o Aprendiz Digital é um aplicativo desenvolvido para atender às necessidades dos professores, oferecendo uma plataforma de aprendizagem para os alunos. No aplicativo, os alunos podem se cadastrar, fazer *login*, receber questões agendadas pelos professores e verificar as notas para compor a média final. Os alunos também participam de *rankings*, tanto da sua turma quanto geral, para avaliar seu desempenho (RODRIGUES; NOGUEIRA; QUEIROGA, 2017). A Figura 4 apresenta três telas do aplicativo Aprendiz Digital.

Figura 4 - Telas do aplicativo Aprendiz Digital



Fonte: RODRIGUES; NOGUEIRA; QUEIROGA, 2017.

O uso dessas ferramentas no ensino-aprendizagem da programação orientada a objetos mostrou-se eficiente entre os alunos. Essas abordagens permitiram que aprendessem de forma mais fácil e interessante, utilizando mídias digitais que já fazem parte do cotidiano. Contudo, existem alguns pontos que devem ser melhorados, como a falta de clareza nas regras e o fato de as funcionalidades não serem intuitivas o suficiente, podendo gerar questionamentos e dificuldades na compreensão.

Da Silva *et al.* (2016) apresentam o POOGame, um jogo educacional baseado em jogos de *Role Playing Game* (RPG) para o ensino de programação orientada a objetos, utilizando a linguagem de programação Java. O jogo permite ao jogador aprender os conceitos da disciplina ao controlar um personagem que enfrenta desafios e batalhas. Foi desenvolvido utilizando-se a *game engine* Unity, obtendo uma avaliação positiva por parte dos alunos, que o consideraram atraente, útil para reforçar conceitos. No entanto, observou-se a necessidade de incluir mais conteúdos e melhorar os desafios e o modo de aprendizagem. A Figura 5 exibe uma imagem do jogo POOGame.

Figura 5 - Fase 2: Campo de batalha

Fonte: DA SILVA *et al.,* 2016.

Toretti Júnior (2013) apresenta um estudo de caso visando explorar a possibilidade de utilizar a Unity para a criação e manipulação de conteúdo em jogos digitais 2D, além da interação com o ambiente. Para isso, foi desenvolvido o jogo Robo Force, que apresenta um policial que pilota uma armadura robótica e deve atravessar vários níveis, enfrentando inimigos e chegando ao confronto final com o vilão principal. O projeto adota uma estética futurista, com personagens e cenários de características mecânicas.

A plataforma Unity oferece suporte a diversos formatos de conteúdo, disponibiliza ferramentas para manipulação de imagens, integração com animações, componentes de física para objetos em 2D, entre outros. Além disso, conta com recursos adicionais através de *plugins* e *scripts* de terceiros, permitindo uma maior personalização e simplificação do desenvolvimento de jogos em 2D (TORETTI JÚNIOR, 2013).

No estudo de Toretti Júnior (2013), a Unity se mostrou uma ferramenta flexível, permitindo a aplicação e criação de jogos em 2D de forma rápida e eficiente. Além disso, são apresentadas diversas técnicas relacionadas à manipulação de imagens, conceitos de

localização espacial em jogos e as ferramentas disponíveis por meio da criação do jogo Robo Force.

Com base nos trabalhos mencionados, esta proposta teve como objetivo traçar uma linha de pesquisa para o desenvolvimento de um jogo educacional para o ensino dos conceitos do paradigma de programação orientada a objetos que incorpore as metodologias relevantes nesses estudos.

O trabalho atual se diferencia ao buscar uma abordagem centrada na criação de um jogo virtual na plataforma Unity estruturado em fases progressivas, com uma narrativa medieval e a aplicação de técnicas de gamificação. Destaca-se, também, a importância da prática de testes para garantir a qualidade do sistema desenvolvido.

3 METODOLOGIA

Este capítulo apresenta os aspectos metodológicos e abordagens utilizados no desenvolvimento deste trabalho, incluindo a classificação da pesquisa, o *framework* do Design Science Research (DSR) e as etapas que o compõem, com as ferramentas, estratégias de ensino dos conceitos, requisitos e métodos do sistema.

3.1 Classificação do trabalho

Este trabalho adota uma abordagem qualitativa, que busca avaliar o processo e a compreensão do jogo educativo na aprendizagem. Em relação à natureza, trata-se de uma pesquisa aplicada, já que visa solucionar um problema com aplicação prática no contexto educacional. Quanto aos objetivos, classifica-se como pesquisa exploratória, pois busca se aprofundar no conhecimento sobre o tema em questão (GERHARDT; SILVEIRA, 2009).

Quanto ao procedimento, classifica-se como artefato de instanciação do DSR, já que objetiva realizar a implementação do sistema educativo para avaliar o seu impacto na aprendizagem dos alunos (BAX, 2015). A pesquisa tem como intuito apresentar algo presumivelmente melhor, buscando demonstrar uma nova abordagem, que seja mais eficaz no ensino-aprendizagem do paradigma de programação orientada a objetos (WAZLAWICK, 2009).

3.2 *Framework* do DSR

Foi escolhido para este trabalho o *framework* proposto por Peffers *et al.* (2007) para a condução da DSR, contendo as etapas de identificação do problema, definição dos resultados esperados, projeto e desenvolvimento, demonstração, avaliação e comunicação. Ao longo desta seção, é explorada cada uma dessas etapas de forma aprofundada, fornecendo as abordagens utilizadas.

3.2.1 Identificação do Problema

Segundo Peffers *et al.* (2007), é necessário definir claramente o problema da pesquisa e a importância de se encontrar uma solução. Como apresentado no Capítulo 1, o problema que motivou esta pesquisa consiste na dificuldade dos alunos em compreender os conceitos de POO, gerando diversos problemas no processo de aprendizagem, podendo desmotivá-los. A

relevância de se identificar uma solução para esse problema reside no fato de que este paradigma é um conceito fundamental na computação, e uma compreensão adequada é essencial para o desenvolvimento de habilidades e competências.

3.2.2 Definição dos Resultados Esperados

De acordo com Peffers *et al.* (2007), nesta etapa, é necessário estabelecer os objetivos de uma solução com base na especificação do problema e no conhecimento do que é possível e viável. Os objetivos, neste trabalho, são qualitativos, descrevendo como um novo componente deverá auxiliar na solução de problemas ainda não abordados. Conforme descrito na Seção 1.1, o objetivo é propor um artefato de instanciação para o desenvolvimento de um jogo virtual educativo com o propósito de tornar o processo de aprendizagem eficiente e interessante para os alunos de computação.

3.2.3 Projeto e Desenvolvimento

Para Peffers *et al.* (2007), esta etapa consiste na criação do artefato, incluindo determinar as funcionalidades, arquitetura e os recursos necessários para passar de objetivos para o *design* e desenvolvimento. Nas seguintes subseções desta etapa, também são mostrados os conceitos de POO, a linguagem de programação, os elementos de gamificação e o plano Scrum a serem abordados. A etapa de desenvolvimento do jogo feito na plataforma Unity é encontrada no Capítulo 4.

3.2.3.1 Unity

É essencial possuir ferramentas de qualidade para assegurar o êxito na implementação de projetos, permitindo a realização efetiva de qualquer tarefa. No processo de seleção da ferramenta Unity, foram consideradas várias características, como a facilidade de uso, ampla gama de recursos disponíveis, apoio da comunidade, desempenho da plataforma, entre outros. Conforme demonstrado no Capítulo 2, a escolha da ferramenta Unity foi fundamentada na análise de um trabalho que utilizou a plataforma para o desenvolvimento de jogos em 2D.

3.2.3.2 Conceitos de POO

O jogo tem como objetivo ensinar conceitos de POO por meio da narrativa da história. A seleção e a classificação dos conceitos basearam-se nos trabalhos de Santos (2003), Barnes

e Kölling (2004), autores cujos livros são amplamente adotados no ensino de POO. Com o intuito de proporcionar uma experiência de aprendizado progressiva, os conceitos foram categorizados em três níveis de dificuldade, que correspondem às fases do jogo: fácil, médio e difícil.

Os jogadores são guiados por desafios que envolvem a aplicação dos conceitos em situações específicas. À medida que a história avança, eles são gradualmente expostos a conceitos mais complexos, permitindo que absorvam e pratiquem os princípios fundamentais de maneira progressiva e interativa. Foram implementadas três fases, cada uma abordando o nível de dificuldade dos conceitos. Na primeira fase, os jogadores são introduzidos a conceitos simples, como classes, objetos, atributos e métodos. Na segunda fase, conceitos intermediários são apresentados, incluindo *getters, setters* e encapsulamento. Na terceira fase, é abordado o conceito mais desafiador, herança.

3.2.3.3 Elementos de gamificação

Para selecionar os elementos de gamificação a serem abordados, foi feita uma análise de quais objetivos de aprendizagem seriam alcançados e como adequar os elementos aos conceitos de POO. Baseando-se em Carvalho (2020), os elementos de gamificação utilizados foram: contação de história, onde o jogo tem uma narrativa que conecta os outros elementos e os conceitos de POO, fornecendo contexto e imersão para os jogadores; mecânica de níveis - foi implementado um sistema de fases no qual os jogadores podem avançar à medida que completam tarefas, sendo que cada nível traz novos desafios e conceitos; mecânica de *feedback* - ao longo das dinâmicas das fases, foram implementados retornos, visando proporcionar ao jogador uma melhor orientação durante a progressão do jogo.

3.2.3.4 Linguagem Java

A linguagem de programação Java foi escolhida para abordar os conceitos de POO, com base nos trabalhos dos autores Santos (2003), Barnes e Kölling (2004). A decisão de optar por essa linguagem decorreu de sua ampla utilização para introduzir o ensino desses conceitos, respaldada por sua popularidade e aplicabilidade. Esses aspectos foram destacados na pesquisa realizada pelo IEEE Spectrum (2023), que classificou a linguagem Java como a terceira mais popular em 2023 na categoria "*Jobs"*. Com sua abordagem orientada a objetos, permite que os desenvolvedores criem programas modulares e flexíveis, tornando a

linguagem adequada para explicar e explorar os princípios fundamentais da POO.

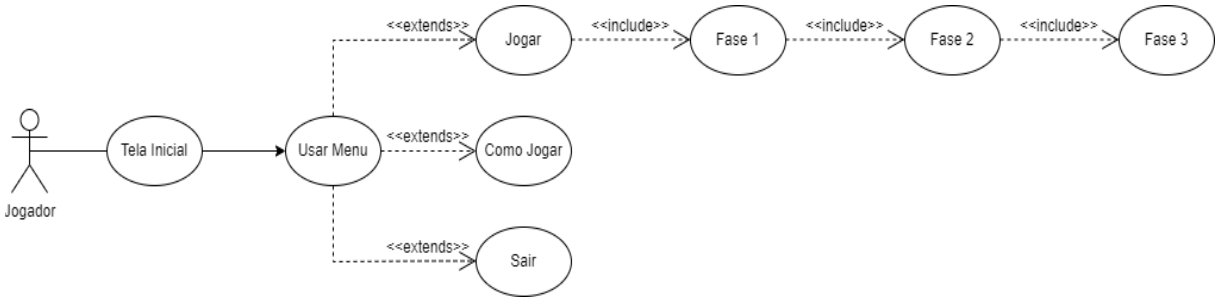
A história do jogo foi elaborada com o propósito de apresentar uma trama concisa e interativa. Trata-se de um jogo de plataforma, e os detalhes da trama estão descritos no Apêndice A. O cenário medieval proporciona um contexto envolvente no qual a linguagem Java é retratada como escrituras antigas. A cada fase, o jogador recebe orientações de como prosseguir, aprofundando-se ainda mais na trama para desenrolar o final da história.

3.2.3.5 Diagrama de Caso de Uso

Por meio da avaliação dos diferentes diagramas UML existentes, para o presente trabalho, escolheu-se utilizar o diagrama de caso de uso. O emprego do diagrama de casos de uso foi uma abordagem benéfica para o desenvolvimento do jogo educativo proposto, pois é uma ferramenta de representações gráficas que busca auxiliar na compreensão dos requisitos do sistema a partir da perspectiva dos usuários. Nesse contexto, os usuários são os alunos de computação que irão interagir com o jogo. Eles fornecem uma linguagem simples e compreensível para o entendimento de como o sistema funcionará, e o diagrama tem o objetivo de identificar os atores que irão interagir com o *software*, bem como as funcionalidades que o sistema disponibilizará (GUEDES, 2011).

Para a criação do diagrama de caso de uso do jogo, foi utilizado o *software* Draw.io *online*. A Figura 6 mostra o sistema principal, que contém os seguintes casos de usos: Usar Menu, Jogar, Como Jogar, Sair, Fases 1, 2 e 3.

Figura 6 - Caso de Uso Geral



Fonte: Elaborado pela autora, 2023.

O caso de uso “Usar Menu” acontece na tela inicial do jogo. Ao selecionar essa opção, o sistema oferece ao jogador as opções de jogar, como jogar ou sair. O caso de uso “Jogar” abrange todas as fases do jogo, oferecendo interfaces e comportamentos distintos para cada uma delas. Cada fase é sequencial, o que significa que é preciso concluir a fase anterior para progredir para a próxima. Dessa forma, é necessário ter um caso de uso individual para cada

fase.

O caso de uso “Como Jogar” fornece ao jogador as informações necessárias para entender a mecânica do jogo. Ao selecionar esta opção, o jogador vê um tutorial explicando os controles do jogo. O caso de uso “Sair” permite que o jogador finalize o jogo.

Os casos de uso “Fase 1”, “Fase 2” e “Fase 3” fornecem aos jogadores desafios que, gradualmente, se tornam mais difíceis à medida que o jogo avança. Cada fase possui uma narrativa que une os conceitos apresentados e orienta o jogador nas etapas necessárias para concluir com sucesso os exercícios propostos. A narrativa desempenha um papel crucial, pois não apenas fornece contexto e motivação, mas também ajuda os jogadores a internalizar conceitos de maneira envolvente e imersiva.

3.2.3.6 Levantamento de Requisitos

O levantamento de requisitos é o processo de identificar e coletar as necessidades e restrições de um sistema, visando garantir as expectativas dos usuários. Segundo Sommerville (2011), a especificação de requisitos consiste em definir os requisitos de usuário e de sistema, que devem ser claros, completos e consistentes.

Para este trabalho, foram definidos os requisitos funcionais e não funcionais do *software*. Os requisitos funcionais são as funcionalidades implementadas no sistema, enquanto os requisitos não funcionais são as restrições e características relacionadas à qualidade, desempenho e segurança. Para defini-los, foram utilizadas técnicas de levantamento de requisitos, nas quais os requisitos são classificados e documentados de maneira clara e mensurável (SOMMERVILLE, 2011).

A seguir, no Quadro 1, são descritos os requisitos funcionais do jogo.

Quadro 1 - Requisitos Funcionais

Identificador	Descrição
[RF-01]	O conhecimento da fase anterior deverá servir como base para a posterior.
[RF-02]	O personagem deve ser capaz de se mover para a esquerda, direita e pular.
[RF-03]	O personagem deve ter animações para diferentes ações, como andar e pular.
[RF-04]	A primeira fase deve ensinar os conceitos de classe, objeto, atributo e método.
[RF-05]	A segunda fase deve ensinar os conceitos de *getter, setter* e encapsulamento.
[RF-06]	A terceira fase deve ensinar o conceito de herança.
[RF-07]	O jogo deve ter trechos de códigos para exemplificar cada conceito em prática, utilizando a linguagem de programação Java.
[RF-08]	O jogador deve receber *feedback* de orientação dentro do jogo.

Fonte: Elaborado pela autora, 2023.

A seguir, no Quadro 2, são descritos os requisitos não funcionais do jogo.

Quadro 2 - Requisitos Não Funcionais

Identificador	Descrição
[RNF-01]	O jogador deve controlar o personagem usando o teclado.
[RNF-02]	O jogador deve interagir com os objetos da cena usando o teclado e o mouse.
[RNF-03]	O jogo deve conter três fases: fácil, média e difícil.
[RNF-04]	O jogo deve ter uma interface de usuário intuitiva e fácil de usar.
[RNF-05]	Os elementos da interface de usuário, como menu, botões e indicadores de status devem ser compreensíveis.
[RNF-06]	O jogo deve ser compatível com o sistema operacional Windows.
[RNF-07]	O jogo deve ser desenvolvido para ser executado em *desktop*.

Fonte: Elaborado pela autora, 2023.

3.2.3.7 SCRUM

Conforme mostrado no Capítulo 2, a metodologia Scrum foi escolhida no desenvolvimento do jogo educacional apresentado neste trabalho. O processo de desenvolvimento foi estruturado em cinco *sprints*, sendo as três primeiras focadas na primeira, segunda e terceira fases do jogo, enquanto a quarta e a quinta foram direcionadas para testes e elaboração do manual, respectivamente. A autora do trabalho assumiu o papel de *Product Owner*, encarregada de gerenciar o *product backlog* e interpretar as necessidades e requisitos do projeto.

No Quadro 3, é possível analisar o *product backlog* do desenvolvimento, onde a primeira coluna apresenta o número de identificação para cada artefato, a segunda coluna traz o nome de cada função, e a terceira coluna, a estimativa de tempo em semanas. Essa organização estratégica não apenas facilitou a execução eficaz das tarefas em cada *sprint*, mas também proporcionou uma visão clara e estruturada do progresso durante todo o desenvolvimento do jogo.

Quadro 3 - *Product Backlog*

Identificador	**Nome**	**Tempo (Semanas)**
01	Desenvolvimento da fase 1	4
02	Desenvolvimento da fase 2	4
03	Desenvolvimento da fase 3	4
04	Realização de testes	1
05	Elaboração do manual	1

Fonte: Elaborado pela autora, 2023.

Ao início de cada *sprint planning*, foi escolhido um item do *backlog* do produto para implementação. Após o término de cada dia de desenvolvimento, foram realizadas *daily scrum* para avaliar o progresso alcançado. As reuniões foram feitas tanto de forma colaborativa, com o orientador do trabalho, quanto individualmente, para permitir momentos de revisão das tarefas realizadas durante o dia. Ao concluir cada *sprint*, promovia-se uma *sprint review* para garantir que todas as metas estabelecidas no planejamento fossem atingidas.

Antes de iniciar a primeira *Sprint*, foi necessário um período de ambientação com a plataforma Unity, para compreender as funcionalidades do sistema e os comandos disponíveis. Após esta etapa, foi iniciada a busca pelos *assets* necessários, incluindo personagens e cenários, realizando-se as adaptações pertinentes para alinhá-los ao enredo da história do jogo.

3.2.4 Demonstração

A demonstração, segundo Peffers *et al.* (2007), consiste em utilizar o artefato para resolver uma ou mais instâncias do problema, envolvendo experimentação, simulação, estudo de caso, entre outros. Esta etapa é encontrada no Capítulo 5, no qual são mostrados os resultados obtidos do jogo por meio da análise das atividades, ferramentas e métodos utilizados.

3.2.5 Avaliação

Esta etapa consiste na avaliação e mensuração do artefato, comparando os objetivos com os resultados observados na etapa da demonstração. Para isso, são realizados testes de caixa-preta. Ao final, é feita uma avaliação analisando se é necessário retornar à etapa de projeto e desenvolvimento, para aprimorar a efetividade do artefato, ou se deve dar prosseguimento para a etapa de comunicação (PEFFERS *et al.,* 2007).

Para garantir a qualidade do *software*, os testes de caixa-preta são realizados para validar se os requisitos funcionais estão sendo atendidos, sem considerar os detalhes internos do código. Eles têm o objetivo de testar o sistema como um todo, focando nas entradas e saídas esperadas, ajudando a revelar problemas de funções incorretas ou omitidas, erros de interface, erros de desempenho, erros de iniciação e término (KOSCIANSKI; SOARES, 2007).

Para a obtenção dos casos de testes deste trabalho, foi utilizado o método de caixa-preta de particionamento de equivalência. Este método consiste em dividir o domínio de entrada do sistema em classes de equivalência. Cada classe representa um conjunto de valores que são tratados de maneira semelhante pelo sistema. Essas classes são utilizadas para criar casos de testes, garantindo uma cobertura eficaz das condições de entrada (PRESSMAN, 2011).

Os casos de testes são projetados para exercitar o máximo de atributos de uma classe de equivalência ao mesmo tempo, garantindo uma cobertura abrangente do domínio de entrada. O particionamento de equivalência ajuda a economizar tempo e recursos, concentrando-se nos casos de teste mais relevantes para a qualidade do sistema (PRESSMAN, 2011).

3.2.6 Comunicação

Nesta última etapa, foi necessário transmitir o problema, importância, utilidade e eficácia do artefato para pesquisadores e outros públicos relevantes. Por fim, foi realizada a escrita da conclusão da monografia, utilizando-se a estrutura padrão de um processo de pesquisa, contendo a definição do problema, a revisão da literatura, o desenvolvimento de hipóteses, a coleta de dados, a análise, os resultados, a discussão e a conclusão (PEFFERS *et al.*, 2007).

4 DESENVOLVIMENTO

Este capítulo aborda o desenvolvimento do *software* deste trabalho. As Seções 4.1, 4.2 e 4.3 abordam, respectivamente, o desenvolvimento da primeira, segunda e terceira fases do jogo na plataforma Unity.

Neste trabalho, foram utilizados os recursos digitais gratuitos disponíveis nos sites da Unity Asset Store, OpenGameArt e itch.io. Os *assets* englobam uma variedade ampla de elementos, como imagens, modelos 3D, texturas, animações e efeitos sonoros. As *sprites* são tipicamente imagens bidimensionais que representam personagens, objetos e cenários em jogos. Esses recursos são cruciais para a criação de experiências imersivas e envolventes para os jogadores, adicionando profundidade aos projetos. As Figuras 7(a) e 7(b) mostram as *sprites* utilizadas como modelos para as caixas de diálogos, pergaminhos e botões durante as fases do jogo.

Figura 7 - Modelos *Sprites*

(a) – Modelo Caixa e Botões (b) – Pergaminho

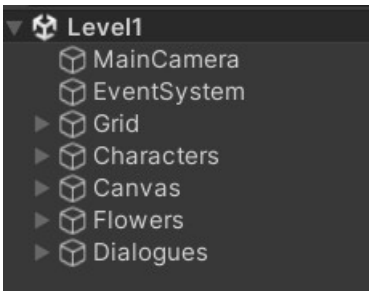
Fonte: Elaborado pela autora, 2023.

A seguir, são detalhados os principais aspectos das interações, elementos técnicos incorporados na Unity para as fases do jogo, o jogador e os objetos de interação.

4.1 Primeira fase

Cada fase do jogo é representada como uma *Scene* (cena). De acordo com a documentação da Unity (2022), esses espaços de trabalho são designados para conter elementos como ambientes, personagens e obstáculos, com a finalidade de organizar o conteúdo do jogo e facilitar a criação e o gerenciamento de distintos níveis. Cada cena inclui *GameObjects* associados, que são os objetos que representam personagens, adereços e cenários. Como evidenciado na Figura 8, na primeira fase, são identificados sete principais objetos, sendo eles: *MainCamera*, *EventSystem*, *Grid*, *Characters*, *Canvas*, *Flowers* e *Dialogues*.

Figura 8 - *GameObjects* Primeira Fase



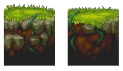
Fonte: Elaborado pela autora, 2024.

O objeto *MainCamera* representa a câmera principal na cena, que define a perspectiva pela qual os elementos do jogo são visualizados. O *script CameraFollow* vinculado ao objeto controla dinamicamente essa câmera, usando a posição da personagem principal, chamada Morgana, para calcular uma nova posição, aplicando uma suavização de interpolação linear. A câmera mantém uma altura constante e é limitada horizontalmente entre os valores das variáveis *minX* e *maxX*. Isso resulta em um acompanhamento suave pelo campo de visão da câmera, proporcionando uma experiência controlada e visualmente agradável.

O objeto *EventSystem* é uma estrutura que gerencia e distribui eventos de entrada no jogo, sendo que cada cena contém apenas um. Atua como um sistema central para capturar e encaminhar eventos de entrada do usuário, como cliques, toques, teclas pressionadas, entre outros.

O objeto Grid é utilizado para armazenar e organizar elementos em uma grade bidimensional, facilitando o posicionamento e a estruturação de objetos em um ambiente 2D. Contém três objetos adicionais chamados *Background1*, *Background2* e *Background3*, responsáveis por criar a ambientação do jogo na primeira fase, utilizando *sprites* para compor o fundo do cenário, como mostrado nas Figuras 9(a), 9(b) e 9(c).

Figura 9 - *Sprites* de Fundo da Primeira Fase



(a) – *Background1*

(b) – *Background2*

(c) – *Background3*

Fonte: OPEN GAME ART, 2022.

O *Background1* é mais complexo, sendo o único que interage diretamente com o jogador, e, por isso, inclui interações físicas e colisões. Possui o componente *Tilemap* para

definir a aparência visual do mapa de blocos para construir o ambiente 2D; *Tilemap Renderer*, para renderizar os blocos na cena; *Tilemap Collider 2D*, para gerenciar colisões bidimensionais; *Rigidbody 2D*, para possibilitar interações físicas entre o jogador e o ambiente; e *Composite Collider 2D*, para otimizar a detecção de colisões. Esses atributos são fundamentais para a jogabilidade, permitindo uma interação de maneira controlada e responsiva com o ambiente do jogo.

Os objetos *Background2* e *Background3* são mais simples, contendo apenas os componentes *Tilemap* e *Tilemap Renderer,* que se destinam principalmente à renderização visual, sem a necessidade de interações com o jogador.

O objeto *Characters* agrupa e organiza os objetos relacionados aos personagens de interação da primeira fase, Morgana e Magnus. O objeto Morgana possui diversos componentes, incluindo *Sprite Renderer*, *Rigidbody 2D*, *Box Collider 2D*, *Animator* e o *script* Morgana.

O componente *Sprite Renderer* recebe o *sprite* associado ao objeto do jogador e tem a função de exibi-lo na tela durante a execução do jogo. A personagem Morgana possui três conjuntos de *sprites*, cada um com sequências de *frames* para ações como respirar, correr, pular, atacar, dano e morte. Além disso, há versões desses conjuntos para cada uma das possibilidades que o jogador pode escolher: sem armamento, com espada e com cajado. Essas variações são ilustradas nas Figuras 10, 11 e 12.

Figura 10 - *Sprites* Morgana sem arma

Fonte: Adaptado de Asset Store, 2021a.

Figura 11 - *Sprites* Morgana com cajado

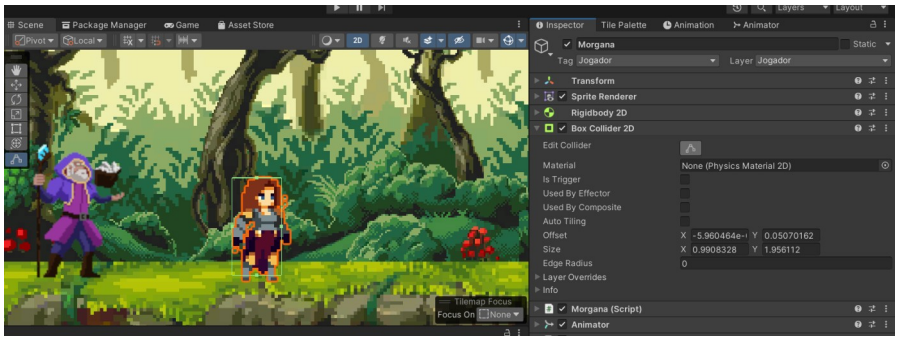
Fonte: Adaptado de Asset Store, 2021a.

Figura 12 - *Sprites* Morgana com espada

Fonte: ASSET STORE, 2021a.

O *Rigidbody 2D* simula a física do objeto, possibilitando seu movimento e interação com forças como gravidade e colisões. Quanto ao *Box Collider 2D*, define a área de colisão do objeto, determinando onde pode interagir com outros elementos no jogo, criando uma "área invisível" ao redor do objeto. No editor, essa área é representada por uma linha de coloração verde em volta da personagem, como mostrado na Figura 13.

Figura 13 - Colisão Morgana



Fonte: Elaborado pela autora, 2024.

O *Animator* controla as animações do objeto, possibilitando transições entre diferentes estados de animação. É através deste componente que se definem diversas animações, como andar, pular e atacar, gerenciando as transições com base em condições, como a entrada do jogador.

O *script* Morgana é responsável por controlar o único personagem jogável. Contém variáveis e parâmetros, como o *Rigidbody2D*, para controlar a física do personagem, e o *Animator*, para gerenciar animações e botões para interações do jogador. Além disso, o *script* interage com outros elementos do jogo, como o vilão, flores a serem coletadas e objetos a serem ativados no decorrer da fase.

O método *Start* é responsável por inicializar referências e definir valores iniciais, como o número inicial de flores coletadas e a adição de *listeners*, que são utilizados para responder a eventos, como os cliques dos botões para selecionar as armas disponíveis no jogo. O método *SelectWeapon* é utilizado para escolher e exibir a animação da arma selecionada pelo jogador, além de salvar essa escolha usando *PlayerPrefs*. Esta classe permite salvar e recuperar dados simples de forma persistente, possibilitando que o jogador mantenha a escolha da arma ao longo das fases do jogo.

O método *Update* é chamado a cada quadro do jogo e atualiza parâmetros no *Animator* para controlar as animações de pulo e movimento, verificando a direção do movimento do jogador e executando o pulo quando necessário. O método *ActivateObject* é responsável por ativar um objeto específico quando determinada condição é satisfeita.

A função *CheckInput* é chamada para verificar as entradas do jogador, permitindo pular se o personagem estiver no chão. O método *Jump* é responsável pelo pulo duplo, verificando e reduzindo o número de pulos disponíveis para que o jogador possa dar somente dois pulos seguidos. O método *OnTriggerEnter2D* é acionado quando o personagem colide com um objeto que possui a *tag Flowers*, sendo utilizado para coletar flores, atualizar a

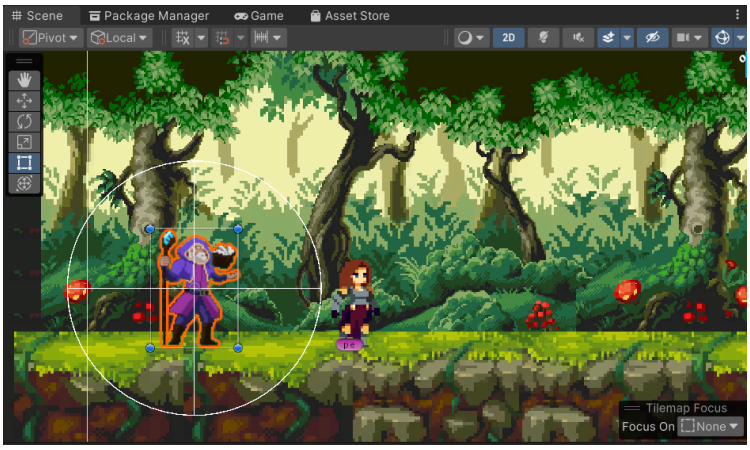
contagem e verificar se a coleta foi concluída.

O objeto Magnus possui somente os componentes *Sprite Renderer* e *Animator*, indicando ser um Personagem Não Jogável (NPC). Também possui quatro *scripts*: *MagnusDialogue1*, *MagnusDialogue2*, *MagnusDialogue3* e *Deliver*. Esses scripts lidam com interações de diálogo entre o jogador e o personagem Magnus, onde diferentes diálogos são acionados com base na proximidade do jogador e interações específicas.

De modo geral, cada *script* de diálogo controla uma interação do jogador com o Magnus. Possuem as variáveis *speechText* e *actorNameText* para armazenar o texto do diálogo e o nome do NPC, além de *playLayer*, para definir a máscara de camada de interação, e *radious*, para representar o raio da área de interação. Mantêm referências aos *scripts* de controle de diálogo, *flags* como *onRadious*, que indicam se o jogador está dentro da área de interação, enquanto *dialogueStarted* controla se o diálogo foi iniciado.

Os métodos *Start* inicializam as referências aos *scripts* de controle de diálogo, *FixedUpdate* verifica a interação a cada atualização fixa e *Update* analisa se a tecla ou botão de confirmação foi pressionado para iniciar o diálogo. Os métodos *Interact* verificam se o jogador está dentro da área de interação, utilizando *Physics2D.OverlapCircle*, para detectar colisões, e *OnDrawGizmosSelected*, para desenhar uma esfera representativa da área de interação no Editor Unity, como mostrado na Figura 14. Essa estrutura permite uma interação personalizada com diferentes personagens e diálogos no jogo.

Figura 14 - Colisão Magnus



Fonte: Elaborado pela autora, 2024.

O *script Deliver* verifica se o jogador está dentro do raio de interação do objeto Magnus e se coletou todas as flores. Quando estas condições são cumpridas, é ativado um objeto no cenário, potencialmente vinculado a uma progressão na história do jogo.

Neste *script*, são utilizadas variáveis para referenciar dois objetos, *flowerBox* e

collectFlowers, que serão ativados e desativados, respectivamente. Há também uma variável para definir o *interactionRadius*, que especifica a distância de interação com o NPC. No método *Start*, uma referência ao *script* Morgana é obtida através da função *FindObjectOfType*, garantindo que o objeto Morgana esteja na mesma hierarquia.

O método *Update* verifica se o jogador está dentro do raio de interação e se coletou as cinco flores. Caso todas as condições sejam atendidas, o método *ActivateObject* é chamado, ativando o *flowerBox*.

O objeto *Canvas*, na Unity, é usado para representar a área na qual se colocam elementos de interface do usuário (*UI*), como botões, imagens, textos etc. Funciona como um contêiner para organizar e exibir elementos de *UI* na tela do jogo. Nesta fase, possui sete componentes vinculados: *DialogueBox*, *FlowersIcon*, *FlowerBox*, *SelectionBox*, *StaffCode*, *SwordCode* e *PotionBox*.

O *DialogueBox* é a imagem que será mostrada quando um diálogo for iniciado entre o jogador e o NPC disponível na cena. Já *FlowersIcon* é a imagem exibida no canto superior direito, indicando ao jogador quantas flores foram coletadas na fase. Esse valor é verificado no *script Deliver* do objeto *Magnus* e atualizado à medida que as flores são coletadas. *FlowerBox* é uma imagem ativada somente quando as cinco flores são coletadas e contém um botão vinculado, que, ao ser pressionado, inicia o próximo diálogo entre o jogador e o NPC.

O objeto *SelectionBox* é uma imagem que representa uma caixa de seleção contendo três botões vinculados. Dois desses botões são imagens das armas que o jogador pode escolher no início do jogo: o cajado e a espada. Ao selecionar um deles, este permanecerá pressionado até que o outro seja pressionado, e vice-versa. Além disso, quando o botão do cajado é pressionado, a imagem *StaffCode* é exibida, e, quando o botão da espada é pressionado, a imagem *SwordCode* é mostrada. Essas imagens contêm o código escrito em Java para representar a classe arma e suas instâncias do cajado e da espada.

O *script* chamado *WeaponSelection*, vinculado ao objeto *SelectionBox*, controla a lógica de seleção de armas no jogo. Isso inclui a interação com os botões de seleção do cajado e da espada, bem como um botão de confirmação. O *script* é responsável por atualizar os objetos visuais do cajado e da espada com base na seleção do jogador, além de gerenciar a desativação da caixa de seleção após a confirmação da escolha.

A imagem *PotionBox* é ativada quando todas as interações necessárias da primeira fase forem completadas. Contém um botão vinculado que, ao ser pressionado, direciona o jogador para a segunda fase do jogo.

O objeto *Flowers* desempenha o papel de um contêiner para os *prefabs* flower1, flower2, flower3, flower4 e flower5. Os *prefabs* são modelos de objetos que podem ser pré-fabricados e reutilizados em diferentes partes do jogo, facilitando o desenvolvimento e garantindo consistência. Esses modelos representam as flores que podem ser encontradas e coletadas pelo jogador dentro do jogo.

Por fim, o objeto *Dialogues* é responsável por controlar os diálogos que ocorrem durante a primeira fase do jogo. É um contêiner que possui três objetos vinculados: *Dialogues1*, *Dialogues2* e *Dialogues3*, cada um possuindo um *script* próprio para gerenciar o diálogo em um momento específico.

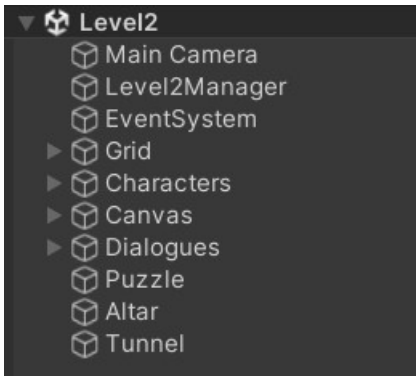
O *script* associado ao objeto *Dialogues1* controla o primeiro diálogo que ocorre. Ao inciar a fase, o diálogo é exibido gradualmente, letra por letra, em uma caixa de texto (*DialogueBox*), dando a impressão de que o texto está sendo digitado. O diálogo é ativado quando um NPC invoca o método *Speech*, fornecendo as frases a serem exibidas e o nome do NPC envolvido. O método *TypeSentence* é uma *coroutine* que exibe as frases gradualmente, enquanto *NextSentence* avança para a próxima frase quando a frase atual é totalmente exibida.

Os *scripts* de *Dialogues2* e *Dialogues3* seguem um padrão semelhante, controlando diálogos específicos que ocorrem em momentos distintos da fase. Cada um deles exibe o diálogo correspondente, avançando para a próxima frase quando o jogador pressiona um botão com o símbolo de seta (»), localizado no canto inferior direito da caixa. Ao final do diálogo, o objeto é desativado, e outro é ativado para dar continuidade ao jogo. Esses *scripts* permitem a criação e execução de múltiplos diálogos de forma dinâmica e organizada dentro do jogo.

4.2 Segunda fase

Após finalizar a primeira fase e compreender a plataforma, tornou-se mais fácil realizar a segunda *sprint*. Portanto, após adaptar os *sprites* a serem usados, foi iniciado o desenvolvimento da segunda fase. Nesta etapa do jogo, como mostrado na Figura 15, identificaram-se dez objetos principais: *MainCamera, Level2Manager, EventSystem, Grid, Characters, Canvas, Dialogues, Puzzle, Altar e Tunnel.*

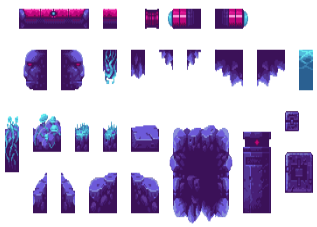
Figura 15 - *GameObjects* Segunda Fase



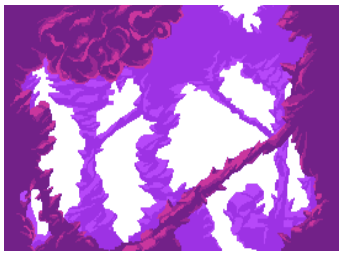
Fonte: Elaborado pela autora, 2024.

Os objetos *MainCamera*, *EventSystem*, *Grid* e *Dialogues* seguem o mesmo padrão apresentado na Seção 4.1. Nas Figuras 16(a), 16(b) e 16(c), são mostrados os *sprites* utilizados no objeto *Grid* para a criação da ambientação do jogo na segunda fase. O objeto *Level2Manager* possui o *script Level2*, que é responsável por verificar a escolha da arma do jogador, recuperar essa escolha e, com base nela, ativar a animação apropriada no componente *Animator* associado à personagem Morgana. Isso permite que o jogo reaja dinamicamente à escolha da arma do jogador, exibindo as animações corretas correspondentes às diferentes armas disponíveis.

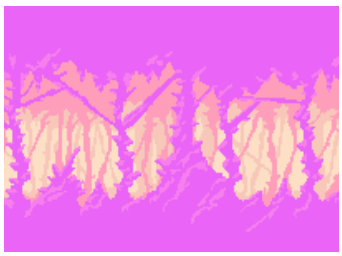
Figura 16 - *Sprites* de Fundo da Segunda Fase



(a) – *Background1*



(b) – *Background2*



(c) – *Background3*

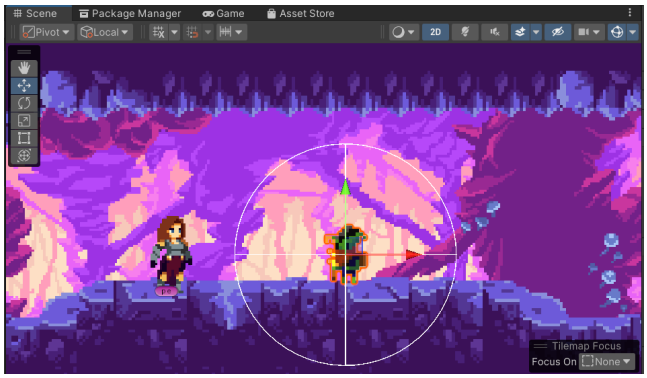
Fonte: ASSET STORE, 2017.

O *script Level2* contém o método *Start*, o qual verifica se a escolha da arma feita na primeira fase do jogo foi salva, consultando os dados do jogador através do *PlayerPrefs*. Dependendo da arma selecionada, uma animação correspondente é ativada no componente *animatorMorgana*. Se a arma selecionada for o cajado, o parâmetro *booleano staff* é definido como verdadeiro, ativando a animação correspondente ao uso do cajado, e o mesmo acontece para a escolha da espada, ativando o parâmetro *sword*.

O objeto *Characters* inclui os dois personagens que interagem na segunda fase: Morgana e Gux. O objeto Morgana é idêntico ao mostrado anteriormente, na Seção 4.1; já o

objeto Gux é similar ao objeto Magnus, possuindo os mesmos componentes e os três *scripts* *GuxDialogue1*, *GuxDialogue2* e *GuxDialogue3*. Esses *scripts* seguem o mesmo padrão dos encontrados no objeto Magnus, lidando com as interações de diálogo entre o jogador e o personagem Gux, ativando diferentes diálogos com base na proximidade do jogador e em interações específicas. Na Figura 17, é possível verificar o raio de interação do objeto Gux.

Figura 17 - Colisão Gux



Fonte: Elaborado pela autora, 2024.

O objeto Canvas possui seis objetos vinculados: *DialogueBox*, *EnigmaBox*, C*rystalIcon*, *DeliverCrystal*, C*rystalCode e FollowPath.* A explicação para *DialogueBox* é descrita na Seção 4.1. *EnigmaBox* é uma imagem com três botões na qual serão mostradas as três perguntas feitas pelo guardião, com suas três opções de respostas ao jogador. *CrystalIcon* é a imagem exibida no canto superior direito, indicando ao jogador que o cristal lhe foi entregue.

DeliverCrystal é uma imagem que é mostrada quando o jogador tiver finalizado o enigma, possuir o cristal e estiver no raio de interação do altar em que deve deixar o cristal. C*rystalCode* é um pergaminho que contém o código em Java referente a aprimorar a arma. *FollowPath* é uma imagem mostrada ao jogador ao final da fase para que siga caminho e vá para a próxima fase.

O objeto *Puzzle* desempenha um papel fundamental na implementação de um sistema de seleção de perguntas e respostas dentro do jogo, o enigma. O *script* associado a este objeto, denominado *QuestionSelection*, controla o comportamento desse sistema, configurando as perguntas e respostas que serão apresentadas ao jogador. Isso é feito por meio da inicialização de *arrays* que armazenam as perguntas e as respostas possíveis para cada uma delas.

A função *Start* é chamada quando o objeto *EnigmaBox* é ativado na cena, sendo inicializadas as perguntas e respostas, configurando os *arrays*. Em seguida, é mostrada a pergunta inicial, chamando a função *Question*, que define o texto da pergunta e as opções de

resposta nos elementos da interface do usuário. Finalmente, adiciona *listeners* de clique aos botões de resposta. Isso é feito utilizando-se a função *onClick.AddListener*, que associa uma função a ser chamada quando um botão é clicado.

A função *Respond* é chamada quando um dos botões de resposta é clicado. Recebe a resposta selecionada como parâmetro e verifica se é correta com base na pergunta atual. Isso é feito comparando-se a resposta selecionada com a resposta correta armazenada no *array* de respostas. Dependendo do resultado, avança para a próxima pergunta (se a resposta estiver correta) ou exibe um alerta (se a resposta estiver incorreta).

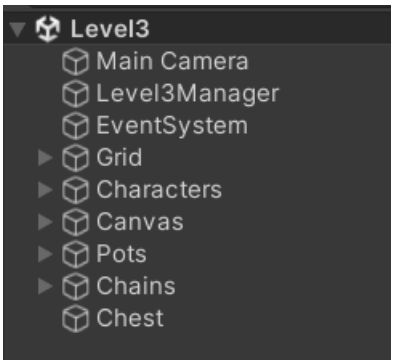
A função *AdvanceQuestion* avança para a próxima pergunta, atualizando o índice da pergunta atual e configurando a próxima pergunta, chamando *Question*. Antes de avançar, verifica se ainda há perguntas restantes para evitar acessar índices fora do limite do *array*. O processo de seleção de perguntas e respostas continua até que todas as perguntas sejam respondidas. Quando todas as perguntas são respondidas corretamente, o sistema indica ao jogador que concluiu todas as perguntas.

O objeto *Altar* possui o *script DeliverAltar*, que verifica se o jogador está dentro de um raio de interação especificado e se coletou o cristal. Quando estas condições são satisfeitas, a imagem *DeliverCrystal* é exibida, incentivando o jogador a colocar o cristal no altar para aprimorar sua arma, conforme indicado pelo código apresentado na imagem *CrystalCode*. Após a conclusão do aprimoramento da arma, a imagem *FollowPath* é exibida, orientando o jogador para o próximo passo no jogo. Por fim, o objeto *Tunnel*, com base no *script SwapScene2*, permite que o jogador seja teletransportado para a próxima fase do jogo quando entrar na área de interação do túnel.

4.3 Terceira fase

Após a conclusão da segunda fase, avançou-se para o desenvolvimento da terceira *sprint*. Na terceira fase do jogo, como mostrado na Figura 18, foram identificados nove principais objetos: *MainCamera*, *Level3Manager*, *EventSystem*, *Grid*, *Characters*, *Canvas*, *Pots*, *Chains* e *Chest*. Os objetos *MainCamera*, *EventSystem* e *Grid* seguem o mesmo padrão apresentado na Seção 4.1.

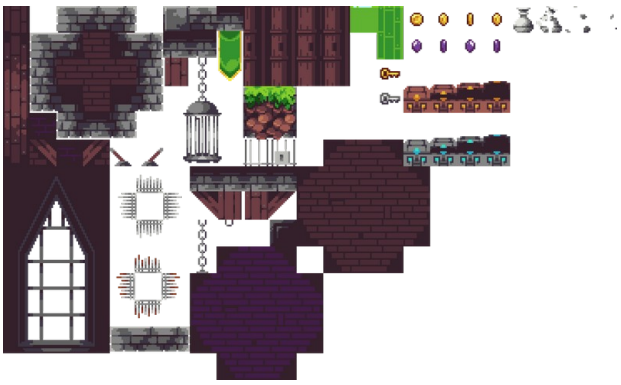
Figura 18 - *GameObjects* Terceira Fase



Fonte: Elaborado pela autora, 2024.

Na Figura 19, são mostrados os *sprites* utilizados no objeto *Grid* para a criação da ambientação do jogo na terceira fase. O objeto *Level3Manager* verifica se a escolha da arma foi salva nas preferências do jogador e, em seguida, ativa a animação correspondente com base nessa escolha. Os objetos *Pots* e *Chains* desempenham o mesmo papel que o objeto *Flowers*, mostrado na Seção 4.1. Contudo, esses objetos não interagem com o jogador, servindo apenas como imagens de decoração no ambiente do jogo.

Figura 19 - *Sprites* de Fundo da Terceira Fase



Fonte: ITCH.IO, 2020.

O objeto *Characters* engloba os dois personagens centrais da terceira fase: Morgana e *Villain*. Morgana é semelhante ao que foi apresentado na Seção 4.1; entretanto, são adicionados os *scripts FightMorgana* e *HealthMorgana*, além do componente *LifeBar*, que representa a barra de vida do jogador.

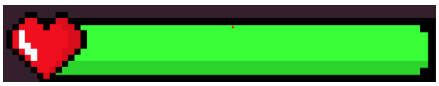
O *script FightMorgana* gerencia as funcionalidades de ataque para o personagem Morgana. A função *Update* verifica se a tecla “C” foi pressionada, acionando a função *EnableAnimationAttack* quando detectada. Essa função, por sua vez, ativa a animação de ataque no *Animator* e realiza um ataque, causando danos a todos os inimigos dentro de uma área definida pelo ponto de ataque. A detecção de inimigos é feita através do uso de

OverlapCircleAll, e a aplicação do dano é realizada chamando-se a função *TakeDamage* do componente *HealthVillain* associado aos inimigos atingidos. O método *OnDrawGizmosSelected* desenha uma esfera de alcance de ataque para visualização no Editor Unity.

O *script HealthMorgana* é responsável por gerenciar a saúde da personagem Morgana. Inicializa a vida máxima da personagem, exibe sua barra de vida correspondente e controla o dano recebido durante o jogo. Quando Morgana sofre dano, o método *TakeDamage* é ativado, diminuindo sua vida e ativando uma animação de dano. Se sua vida chegar a zero, o método *Die* é acionado, desencadeando uma sequência de eventos, incluindo a desativação do objeto Morgana e a exibição de uma mensagem de retorno para a terceira fase, indicando que o jogador perdeu a batalha.

Os objetos Morgana e vilão compartilham a mesma lógica do objeto *LifeBar*, que inclui um componente *Slider* para visualizar a barra da saúde. O *script* associado é encarregado de gerenciar esse componente. O método *SetMaxSaude* estabelece o valor máximo da barra de vida com base na saúde total do personagem. O método *SetDelize* atualiza dinamicamente o valor atual conforme a saúde do personagem, refletindo visualmente a diminuição da barra à medida que a luta progride. Morgana inicia a batalha com 1000 de saúde, enquanto o vilão, com 500. Nas Figuras 20(a) e 20(b), são mostrados os *sprites* da barra de vida.

Figura 20 - *Sprites* Barra de Vida



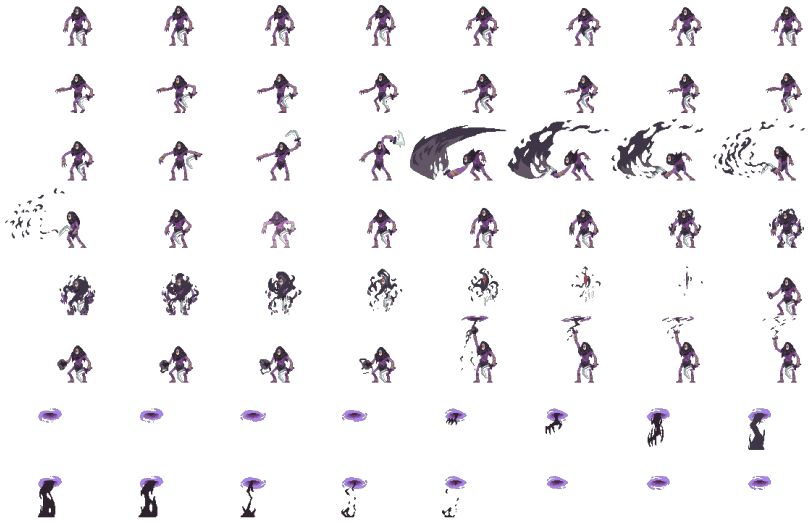
(a) – Barra de Vida Morgana

(b) – Barra de Vida Vilão

Fonte: Elaborado pela autora, 2024.

O objeto *Villain* é composto por uma variedade de componentes, incluindo *Sprite Renderer*, *Animator*, *Box Collider 2D* e *Rigidbody 2D.* Além disso, inclui o objeto *LifeBar* e quatro *scripts*: *Vilain*, *HealthVillain*, *FightVillain* e *VillainMovement*. Estes *scripts* cooperam para criar uma experiência de jogo dinâmica e envolvente, permitindo que o vilão interaja de várias maneiras com o jogador, incluindo combates, movimentos e reações à sua própria saúde. O vilão é caracterizado por um conjunto de *sprites* que abrangem sequências de *frames* para ações como respirar, andar, atacar, sofrer dano e morrer, como ilustrado na Figura 21.

Figura 21 - *Sprites* Vilão



Fonte: ASSET STORE, 2021b.

O *script Villain* assume o controle das interações entre o vilão e o jogador, empregando o método *Interact* para verificar se o jogador está dentro da área de interação estipulada, usando colisores e máscaras de camada para detectar a presença do jogador. A função *LookAtPlayer* direciona o vilão na direção do jogador, utilizando sua posição para determinar a orientação adequada.

O *script HealthVillain* controla a saúde do vilão, e o método *TakeDamage* é acionado quando o jogador pressiona a tecla "C" para executar um ataque e está dentro do alcance de dano do vilão. Com base no dano recebido, a saúde é reduzida, desencadeando animações de dano e diminuindo a representação visual da barra de vida. Quando a saúde chega a zero, o método *Die* gerencia seu estado, iniciando uma série de eventos relacionados à sua morte, como desativar o objeto vilão e exibir uma mensagem final do jogo indicando a vitória do jogador na batalha.

O *script VillainMovement* emprega o método *Update* para coordenar o movimento do Vilão em direção ao jogador, utilizando o *Rigidbody* para deslocamento. Também calcula a distância entre o vilão e o jogador, assegurando que os ataques sejam acionados somente quando o jogador estiver ao alcance. Por outro lado, o *script FightVillain* gerencia o comportamento de combate. A função principal, *AttackAnimation*, ativa a animação de ataque e causa dano ao inimigo próximo. Além disso, controla o intervalo entre os ataques para evitar que sejam realizados repetidamente, em rápida sucessão.

O objeto *Canvas* é composto por seis objetos principais: *OpenChest*, *Parchment1*, *Parchment2*, *Parchment3*, *FinalBox* e *ReturnBox*. *OpenChest* é uma imagem interativa que o

jogador acessa ao se aproximar do objeto *Chest*, exibindo uma mensagem com um botão para abrir. Se o baú for aberto, os objetos *Parchment1*, *Parchment2* e *Parchment3* serão ativados, apresentando imagens de pergaminhos com escrituras antigas e trechos de código para contextualizar o jogador.

O objeto *Chest* possui o *script BauActivate*, que é responsável por verificar se o jogador entrou no raio de interação e ativar a imagem do *Parchment1*. No método *Update*, é calculada a distância entre o baú e o jogador usando-se a função *Vector2.Distance*, medindo-se a distância euclidiana entre as posições do baú e do jogador. Cada pergaminho possui um botão vinculado para avançar para o próximo. Após a leitura dos pergaminhos, o jogador deve enfrentar o vilão. Ao vencer a luta, o objeto *FinalBox* é ativado, exibindo uma mensagem de vitória. Em caso de derrota, o objeto *ReturnBox* é ativado, oferecendo um botão para retornar à terceira fase e tentar novamente.

5 RESULTADOS E DISCUSSÃO

O presente Capítulo objetiva apresentar os resultados finais obtidos com o trabalho. A Seção 5.1 apresenta o resultado das fases do jogo, destacando as interações e metáforas utilizadas. A Seção 5.2 descreve a fase de testes, enquanto a Seção 5.3 trata da escrita do manual de acesso ao jogo, que será disponibilizado no repositório.

5.1 Jogo

O registro do tempo em jogos educacionais fornece uma visão clara do progresso dos alunos, permitindo aos desenvolvedores e educadores avaliar o desempenho, identificar possíveis áreas de dificuldade e ajustar o conteúdo do jogo conforme necessário. No Quadro 4, foi registrado o tempo gasto pela autora para completar cada fase do jogo, realizando todas as leituras, interações e desafios. No total, foram necessários 18 minutos, 57 segundos e 58 milésimos de segundos para concluir todas as fases.

Quadro 4 - Registro de tempo

Fase	Tempo gasto
1	08:29:46
2	05:38:29
3	04:49:55

Fonte: Elaborado pela autora, 2024.

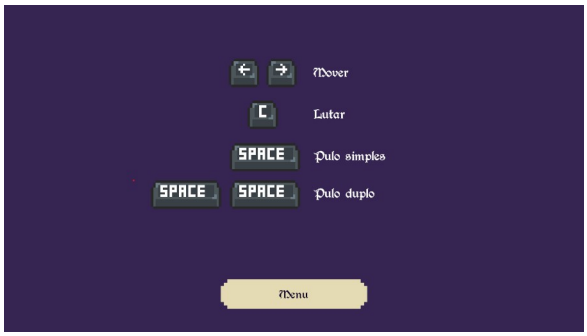
Ao iniciar o jogo, é apresentada a tela inicial, conforme mostrado na Figura 22. Nessa tela, é necessário pressionar a tecla “Enter” para acessar o menu principal. No menu, há opções para jogar, receber instruções sobre como jogar e sair do jogo, como ilustrado na Figura 23(a). Se o jogador optar por clicar no botão “Como Jogar”, será direcionado para a tela mostrada na Figura 23(b), na qual todos os comandos do jogo são detalhados.

Figura 22 - Tela Inicial

Fonte: Elaborado pela autora, 2024.

Figura 23 - Menu Principal

(a) – Tela de Opções



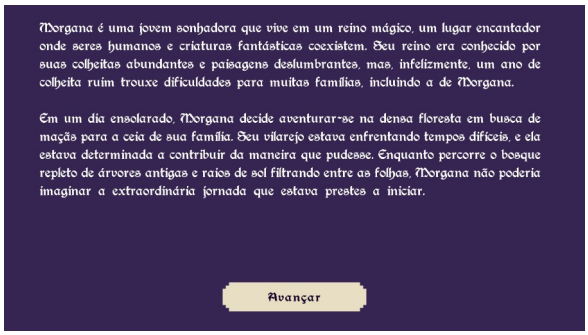
(b) – Tela de Comandos

Fonte: Elaborado pela autora, 2024.

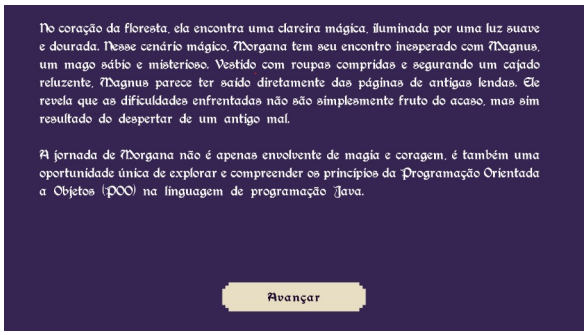
Ao selecionar o botão “Jogar”, no menu principal, o usuário é conduzido para duas telas de ambientação. Essas telas foram elaboradas com o propósito de oferecer ao jogador uma compreensão do contexto do jogo antes de iniciar a experiência. Na primeira tela, mostrada na Figura 24(a), o usuário é introduzido à protagonista do jogo, Morgana, com informações cruciais sobre sua identidade, localização e sua situação atual.

Na segunda tela, o enredo principal do jogo é introduzido de maneira sucinta e envolvente, conforme mostrado na Figura 24(b). O jogador recebe uma breve descrição do desafio inicial enfrentado por Morgana, a personagem principal, juntamente com seus objetivos. Estas telas de ambientação não apenas situam o jogador no universo do jogo, mas também despertam sua curiosidade e o preparam para a aventura que está prestes a começar.

Figura 24 - Telas de Ambientação



(a) – Apresentação da Protagonista



(b) – Chamado da Aventura

Fonte: Elaborado pela autora, 2024.

Ao iniciar a primeira fase, o jogador é imerso em um diálogo liderado pelo mentor, Magnus. Durante essa interação, há uma introdução, seguida por um convite, para iniciar a jornada do jogo. Logo após, são apresentadas explicações sobre os conceitos de POO que serão aplicados nesta etapa. A Figura 25 proporciona uma visão geral do ambiente.

Figura 25 - Tela Inicial da Primeira Fase

Fonte: Elaborado pela autora, 2024.

A abordagem baseia-se na utilização de metáforas para ensinar POO. Após a introdução dos conceitos de classe, objeto, método e atributo, o jogador é direcionado a escolher entre duas armas distintas: um cajado e uma espada. Cada uma dessas armas é tratada como uma instância da classe genérica “Arma”, como mostrado no código Java, apresentado no pergaminho ao lado direito da tela de escolha. Essa classe funciona como um modelo abstrato que encapsula os atributos e métodos compartilhados entre as armas disponíveis no jogo.

Ao escolher a espada, por exemplo, o jogador cria uma instância específica dessa arma, que herda os atributos e métodos definidos na classe “Arma”, mas que também possui suas próprias características. Da mesma forma, ao optar pelo cajado, uma instância distinta é

criada, refletindo as propriedades específicas dessa arma, conforme indicado no pergaminho.

Essa abordagem não apenas proporciona uma experiência interativa e envolvente para o jogador, mas também facilita a compreensão dos conceitos abstratos da programação orientada a objetos ao associá-los diretamente a objetos tangíveis dentro do contexto do jogo. Nas Figuras 26(a) e 26(b), é apresentada a seleção da espada e do cajado, juntamente com seus pergaminhos contendo os códigos correspondentes. Os trechos de todos os códigos presentes no jogo podem ser encontrados no Apêndice B.

Figura 26 - Telas da Seleção da Arma do Personagem

(a) – Seleção da Arma Espada

(b) – Seleção da Arma Cajado

Fonte: Elaborado pela autora, 2024.

Após a seleção das armas, Morgana é instruída a prosseguir em sua jornada em direção a uma caverna, onde ela poderá aprimorar sua arma para enfrentar o monstro que a aguarda. No entanto, para acessar a caverna, o jogador precisa de uma poção de teletransporte. Magnus se oferece para prepará-la, mas carece de um ingrediente crucial: cinco flores azuis, encontradas na floresta onde estão. Assim, o jogador deve explorar a floresta em busca dessas cinco flores azuis, como mostrado na Figura 27.

Figura 27 - Tela de Coletar Flores

Fonte: Elaborado pela autora, 2024.

Para auxiliá-lo, no canto superior direito da tela, o jogador terá um contador para acompanhar quantas flores já foram encontradas. Após reunir todos os ingredientes necessários, o jogador deve retornar a Magnus, que preparará a poção de teletransporte. Ao beber a poção, Morgana avançará para a próxima fase, conforme mostrado na Figura 28.

Figura 28 - Tela Receber Poção

Fonte: Elaborado pela autora, 2024.

Na segunda fase, retratada na Figura 29, o jogador se encontra diante do guardião da caverna, chamado Gux. É proposto ao jogador um desafio que concederá acesso ao cristal necessário para fortalecer sua arma. Este desafio consiste em responder três perguntas simples relacionadas aos conceitos de POO, reforçando os conhecimentos adquiridos anteriormente.

Figura 29 - Tela Inicial da Segunda Fase

Fonte: Elaborado pela autora, 2024.

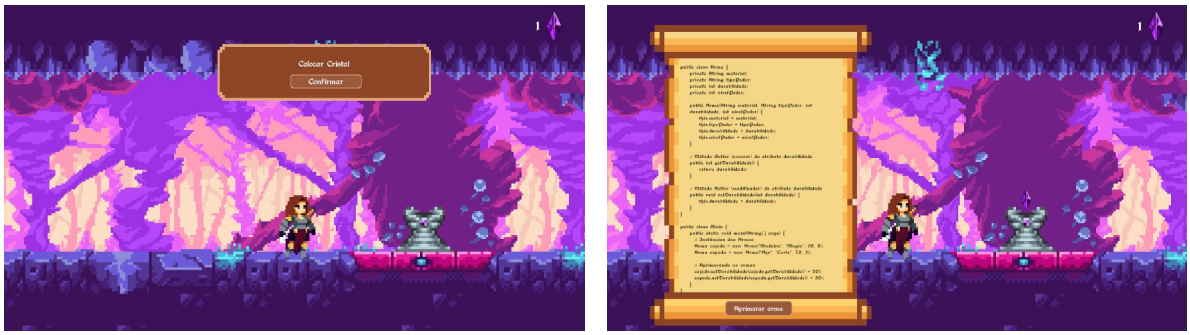
Conforme o jogador for respondendo corretamente cada questão do desafio, avançará para a próxima, até finalizar as três questões. No entanto, se errar, permanecerá na questão até responder corretamente. Durante esse período, receberá uma mensagem de alerta indicando que a resposta está incorreta e que deve tentar novamente para prosseguir. A Figura 30 mostra uma das questões no momento em que o jogador errou a resposta e está exibindo a mensagem de alerta.

Figura 30 - Tela Questão Enigma

Fonte: Elaborado pela autora, 2024.

Após responder corretamente as três questões, o jogador receberá o cristal. Em seguida, serão fornecidas explicações sobre os conceitos de *getter*, *setter* e encapsulamento. Ao colocar o cristal no altar, o jogador terá acesso ao pergaminho contendo o código em Java, que apresenta a mesma classe “Arma” da fase anterior, porém, com métodos para acessar e modificar a durabilidade da arma do jogador. Nas Figuras 31(a) e 31(b), são exibidos o altar e o pergaminho com o código.

Figura 31 - Telas Aprimorar Armamento

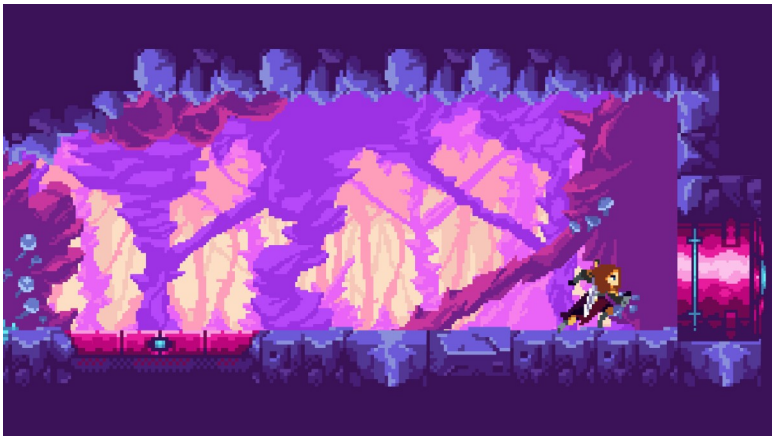


(a) – Interação Altar (b) – Pergaminho com código *setter* e *getter*

Fonte: Elaborado pela autora, 2024.

Como metáfora, o cristal permite que o jogador aprimore os atributos da arma, tornando-a uma extensão mais poderosa de suas habilidades. Atua como um catalisador, potencializando os atributos da arma e capacitando o jogador para enfrentar desafios mais difíceis no futuro. Ao clicar no botão associado ao pergaminho para aplicar essas melhorias, o jogador receberá uma mensagem para continuar avançando, adentrando em um túnel que o levará à fase final de sua jornada, conforme representado na Figura 32.

Figura 32 - Personagem Entrando no Túnel



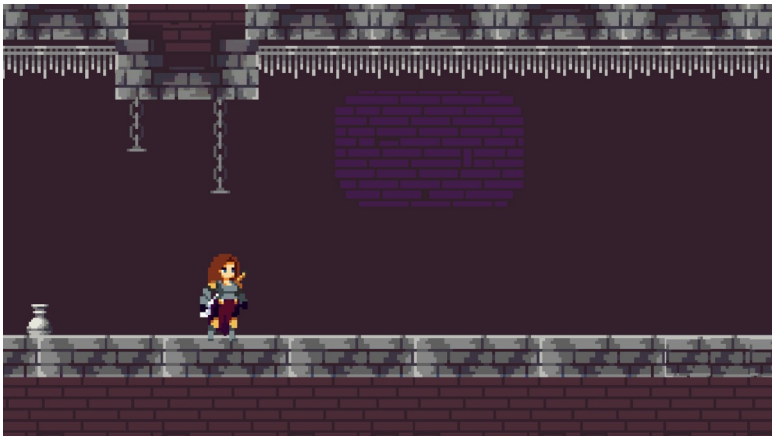
Fonte: Elaborado pela autora, 2024.

Na terceira fase, o jogador chega ao castelo onde o vilão o aguarda, conforme exposto na Figura 33. O jogador deve encontrar e abrir o baú para ler os pergaminhos que contêm as histórias de um nobre guerreiro que, há muito tempo, conseguiu derrotar o monstro. O pergaminho revela ao jogador que somente sua herdeira, Morgana, teria a capacidade de enfrentar o monstro que retornou.

Esta narrativa estabelece uma poderosa metáfora no conceito de herança na POO, associando a linhagem de Morgana com a habilidade de enfrentar o desafio, assim como a herança de classes na programação, onde uma classe pode herdar atributos e métodos de

outra.

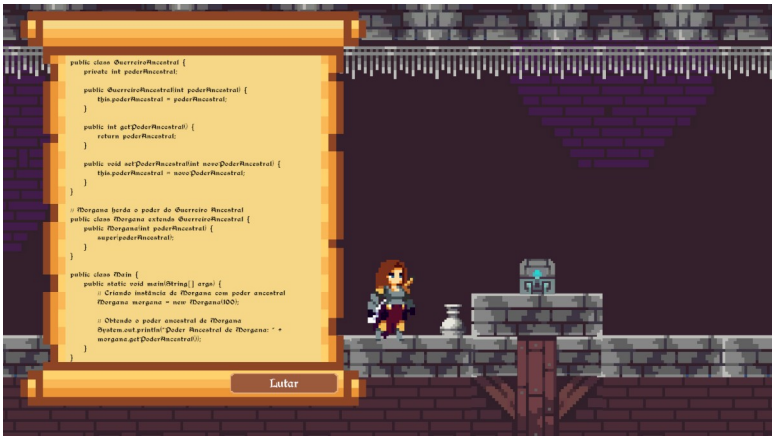
Figura 33 - Tela Inicial da Terceira Fase



Fonte: Elaborado pela autora, 2024.

Ao final da história, o jogador encontra um pergaminho adicional contendo um código em Java que ilustra a herança da classe “Guerreiro” para Morgana, evidenciando que possui a força necessária para enfrentar o monstro, como mostrado na Figura 34. Esta representação reforça a ideia de herança na programação, demonstrando como uma classe (Morgana) pode herdar características de outra classe (Guerreiro), permitindo-a enfrentar desafios com base nos atributos e métodos transmitidos.

Figura 34 - Tela Inicial da Terceira Fase



Fonte: Elaborado pela autora, 2024.

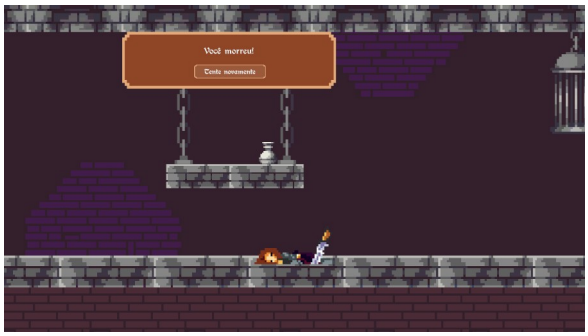
Após a leitura de todos os pergaminhos, o jogador finalmente encontra o monstro no interior do castelo e se prepara para enfrentá-lo em uma batalha decisiva. Ao se aproximar, tanto o jogador quanto o vilão recebem barras de vida para representar a saúde de cada um durante o combate.

A inclusão das barras de vida é fundamental para proporcionar uma experiência de jogo mais imersiva e estratégica, permitindo que o jogador acompanhe visualmente o

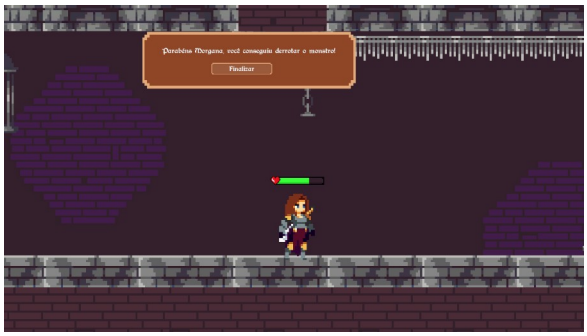
progresso da batalha, avaliando o impacto de seus ataques e os danos recebidos do oponente. Além disso, as barras de vida fornecem feedback claro sobre a condição atual do jogador e do vilão, aumentando a tensão e a emoção do confronto.

Caso o jogador seja derrotado na batalha, receberá uma mensagem de retorno, indicando a necessidade de refazer a terceira fase novamente, conforme ilustrado na Figura 35(a). No entanto, se sair vitorioso do embate, será direcionado para a mensagem mostrada na Figura 35(b), prosseguindo, assim, em sua jornada rumo à conclusão do jogo.

Figura 35 - Mensagens Finais



(a) – Mensagem Retorno

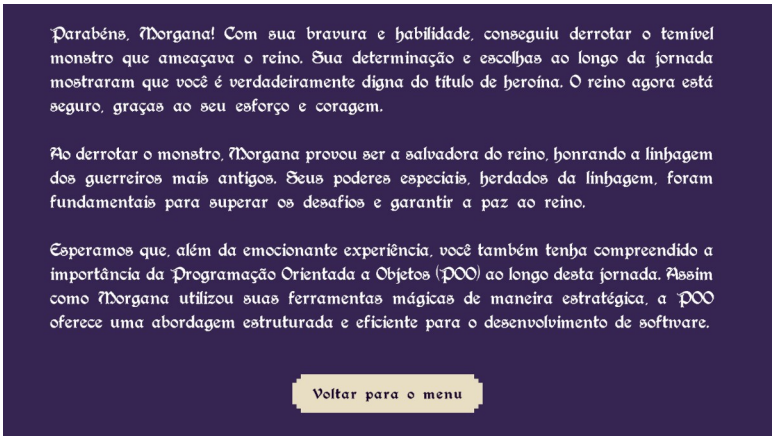


(b) – Mensagem Vitória

Fonte: Elaborado pela autora, 2024.

Por fim, como exibido na Figura 36, após a vitória do jogador, é mostrada uma tela que contém um texto que ressalta o papel heroico do jogador, Morgana, destacando a aplicação dos conceitos aprendidos ao longo da jornada e enfatizando a importância da programação orientada a objetos de forma significativa e relevante para a experiência do jogo.

Figura 36 - Tela Inicial da Terceira Fase



Fonte: Elaborado pela autora, 2024.

5.2 Testes

Para se realizar a quarta *sprint*, foi necessário analisar todas as fases em conjunto e verificar se as funcionalidades estavam de acordo com o que foi proposto nos requisitos funcionais e não funcionais descritos na Seção 3.2.3.6. Durante esse processo, foram conduzidos testes de caixa-preta para garantir a integridade e a eficiência do sistema. Caso alguma anomalia fosse identificada, era registrada para posterior correção. Além disso, efetuou-se uma revisão completa do jogo para identificar áreas de melhoria e oportunidades de otimização.

Para conduzir os testes, foi empregado o método de caixa-preta de particionamento de equivalência. As condições de entrada foram definidas com base no estado atual do jogador dentro do jogo. As classes válidas representam ações esperadas, como a coleta de itens, enquanto as classes inválidas englobam ações inesperadas ou falhas, como tentativas de realizar ações sem os requisitos necessários. Os casos de teste foram projetados para avaliar diferentes aspectos do jogo, abrangendo uma variedade de cenários específicos. Os valores correspondem aos dados específicos utilizados para cada ação ou interação, enquanto o status indica se a ação foi bem-sucedida ou não.

Foram conduzidos seis testes, cada um abordando as interações principais do jogo. Estas interações incluem: acesso ao menu, seleção da arma, obtenção da poção, resolução do enigma, inserir o cristal no altar e confronto com o vilão.

Para cada teste realizado, foram criados dois quadros para documentar os resultados. O primeiro quadro destaca a entrada do teste, juntamente com as classes válidas e inválidas relacionadas a essa entrada. Este quadro oferece uma visão geral das diferentes maneiras de realizar a ação em teste e das possíveis falhas associadas. O segundo quadro é dedicado aos casos de teste específicos dentro das classes mencionadas anteriormente. Detalha as ações executadas em cada caso de teste, os valores utilizados e os resultados esperados em comparação com os resultados obtidos.

Os Quadros 5 e 6 apresentam os resultados do primeiro teste, que envolveu iniciar o jogo e acessar o menu principal seguindo as instruções exibidas na tela para o usuário: "Pressione ‘Enter’ para jogar". O jogador deve ser direcionado para o menu apenas se pressionar a tecla "Enter". O teste foi realizado para verificar o comportamento das outras teclas disponíveis.

Quadro 5 - Primeiro Teste: Acessar menu

Entrada	Classes válidas	Classes inválidas
Acessar menu	C1 - Tecla Enter	C2 - Qualquer outra tecla

Fonte: Elaborado pela autora, 2024.

Quadro 6 - Resultados do Primeiro Caso de Teste

Caso de Teste	Valores	Status Esperado	Status Obtido
C1	Clique da tecla “Enter”	Válido	Válido
C2	Clique de qualquer outra tecla	Inválido	Inválido

Fonte: Elaborado pela autora, 2024.

Os Quadros 7 e 8 exibem os resultados do segundo teste, que abordou o momento do jogo em que o usuário precisa escolher uma arma. Este processo foi realizado seguindo-se uma tela que apresenta três botões: dois botões com imagens das armas disponíveis (o cajado e a espada) e o botão de confirmação. O jogador só deve receber uma arma se pressionar o botão correspondente à arma desejada e, em seguida, o botão de confirmação.

Quadro 7 - Segundo Teste: Escolher a arma

Entrada	Classes válidas	Classes inválidas
Escolher a arma	C1 – Escolher cajado e confirmar C2 – Escolher espada e confirmar	C3 – Não escolher, mas confirmar C4 – Não confirmar escolha C5 – Não escolher

Fonte: Elaborado pela autora, 2024.

Quadro 8 - Resultados do Segundo Caso de Teste

Caso de Teste	Valores	Status Esperado	Status Obtido
C1	Clique no botão do cajado e no botão confirmar	Válido	Válido
C2	Clique no botão da espada e no botão confirmar	Válido	Válido
C3	Não clicar no botão do cajado ou da espada, mas clicar no botão confirmar	Inválido	Válido
C4	Clique no botão do cajado ou da espada e não clicar no botão confirmar	Inválido	Inválido
C5	Não clicar em nenhum botão	Inválido	Inválido

Fonte: Elaborado pela autora, 2024.

Os Quadros 9 e 10 exibem os resultados do terceiro teste, que tratou do momento do jogo em que o usuário precisa retornar com as cinco flores coletadas e entregá-las a Magnus para receber a poção que o levará à segunda fase do jogo. O jogador só deve ser capaz de receber a poção após coletar todas as flores.

Quadro 9 - Terceiro Teste: Receber poção

Entrada	Classes válidas	Classes inválidas
Receber poção	C1 – Contador *flowers* = 5 e está dentro do raio de interação	C2 – Contador *flowers* = 5 e não está dentro do raio de interação C3 – Contador *flowers* != 5

Fonte: Elaborado pela autora, 2024.

Quadro 10 - Resultados do Terceiro Caso de Teste

Caso de Teste	Valores	Status Esperado	Status Obtido
C1	Todas as flores coletadas e o jogador está no raio de interação	Válido	Válido
C2	Todas as flores coletadas e o jogador não está no raio de interação do objeto Magnus	Inválido	Inválido
C3	Não coletar todas as flores	Inválido	Inválido

Fonte: Elaborado pela autora, 2024.

Os Quadros 11 e 12 apresentam os resultados do quarto teste, que abordou o momento do jogo em que o usuário precisa responder corretamente as três questões do enigma proposto por Gux na caverna. A cada pergunta respondida incorretamente, o jogador receberá um aviso de alerta para indicar que a resposta está errada e deve tentar novamente até acertar. As questões são de escolha única, com apenas uma resposta correta para cada pergunta. O jogador só deve ser capaz de solucionar o enigma se responder corretamente todas as questões.

Quadro 11 - Quarto Teste: Solucionar enigma

Entrada	Classes válidas	Classes inválidas
Solucionar enigma	C1 – Responder corretamente às três questões	C2 – Primeira questão incorreta C3 – Segunda questão incorreta C4 – Terceira questão incorreta C5 – Nenhuma questão respondida

Fonte: Elaborado pela autora, 2024.

Quadro 12 - Resultados do Quarto Caso de teste

Caso de Teste	Valores	Status Esperado	Status Obtido
C1	Clique no botão “Modelo” para a primeira questão, “Objeto” para a segunda questão e “Método” para a terceira questão	Válido	Válido
C2	Clique no botão “Método” ou “Instância” para a primeira questão	Inválido	Inválido
C3	Clique no botão “Atributo” ou “Método” para a segunda questão	Inválido	Inválido
C4	Clique no botão “Atributo” ou “Instância” para a terceira questão	Inválido	Inválido
C5	Não clicar em nenhum botão	Inválido	Inválido

Fonte: Elaborado pela autora, 2024.

Os Quadros 13 e 14 exibem os resultados do quinto teste, que tratou do momento do jogo em que o usuário precisa colocar o cristal recebido por Gux no altar. O jogador só deve ser capaz de realizar essa ação se possuir o cristal e estiver dentro do raio de interação do altar.

Quadro 13 - Quinto Texto: Cristal no altar

Entrada	Classes válidas	Classes inválidas
Cristal no altar	C1 – Contador *crystal* = 1 e está no raio de interação	C2 – Contador *crystal* = 1 e não está no raio de interação C3 – Contador *crystal* != 1

Fonte: Elaborado pela autora, 2024.

Quadro 14 - Resultados do Quinto Caso de Teste

Caso de Teste	Valores	Status Esperado	Status Obtido
C1	Cristal coletado e o jogador está no raio de interação do objeto altar	Válido	Válido
C2	Cristal coletado e o jogador não está no raio de interação do objeto altar	Inválido	Inválido
C3	Não coletou o cristal	Inválido	Inválido

Fonte: Elaborado pela autora, 2024.

Os Quadros 15 e 16 exibem os resultados do sexto teste, que explorou o momento do jogo em que o usuário encontra o vilão para a batalha final. A batalha só deve ser iniciada se o

jogador tiver concluído a leitura dos pergaminhos encontrados no baú no início da terceira fase. O desfecho da batalha pode resultar na derrota do jogador ou do monstro.

Quadro 15 - Sexto Teste: Luta

Entrada	Classes válidas	Classes inválidas
Luta	C1 – Monstro derrotado C2 – Jogador derrotado	C3 – Ataque antecipado

Fonte: Elaborado pela autora, 2024.

Quadro 16 - Resultados do Sexto Caso de teste

Caso de Teste	Valores	Status Esperado	Status Obtido
C1	O jogador derrota o monstro	Válido	Válido
C2	O monstro derrota o jogador	Válido	Válido
C3	O jogador ataca o monstro antes de ler os pergaminhos	Inválido	Válido

Fonte: Elaborado pela autora, 2024.

Para a etapa de avaliação dos testes, foi necessário verificar as colunas de status esperado e status obtido de cada caso de teste. Se elas forem iguais, não houve nenhuma falha. Caso estejam diferentes, indica que o teste identificou uma falha. A partir dos testes realizados, foram detectadas duas falhas no sistema. As falhas foram encontradas no segundo e sexto teste.

A primeira falha ocorreu no terceiro caso do segundo teste. Para este caso, foi considerado que o jogador estava na tela de escolha da arma e não selecionou o botão do cajado ou da espada, mas clicou no botão de confirmar. Como resultado, o jogador não recebeu nenhuma arma. No entanto, o diálogo seguinte, que deveria ser exibido somente quando o jogador adquirisse a arma, foi ativado atrás da tela da seleção, indicando uma falha, como mostra a Figura 37.

Figura 37 - Primeira Falha

Fonte: Elaborado pela autora, 2024.

Se o jogador pressionar o botão sem ter selecionado uma arma, nenhum retorno deveria acontecer. Para resolver esse problema, foi criada uma condição no *script* *WeaponSelection* para considerar a ação do clique no botão de confirmar somente se algum dos botões do cajado ou espada estiver selecionado.

A segunda falha foi encontrada no terceiro caso do sexto teste. Ocorre quando o jogador, após encontrar o baú na terceira fase, não termina de ler os pergaminhos e vai diretamente ao encontro do vilão no final do mapa, iniciando a briga. Isso faz com que as telas com os pergaminhos permaneçam ativas, limitando o campo de batalha, conforme mostrado na Figura 38. Para corrigir esse problema, uma condição foi adicionada ao *script* *Villain*. O objeto do vilão só será ativado para visão e interação com o jogador se estiver finalizado a interação dos pergaminhos do baú.

Figura 38 - Segunda Falha

Fonte: Elaborado pela autora, 2024.

5.3 Manual para acesso

A quinta *sprint* se baseia na criação de uma sequência de instruções simples e dos requisitos necessários de como o usuário poderá acessar o jogo. As instruções juntamente com uma breve descrição sobre o jogo serão encontradas pelo usuário em um arquivo de texto chamado "*README*" no repositório. Durante o processo de desenvolvimento, foi decidido que o repositório do jogo será o GitHub, por ser de fácil acesso e colaboração. O jogador deve seguir as etapas abaixo:

- Pré-requisitos
  - Sistema operacional Windows.
- Instalando o jogo Morgana
  1. Acesse o link[2] fornecido, direcionando-o à página do GitHub onde o jogo está disponível.
  2. Na página, encontre a opção para baixar o jogo.
  3. Após o download, descompacte o arquivo, revelando diversas pastas e arquivos.
     Importante: Não faça alterações nos arquivos do jogo.
  4. No diretório descompactado, localize e abra o arquivo executável do jogo, denominado "JogoMorgana.exe".

A partir desse momento, o usuário estará pronto para aproveitar da experiência oferecida pelo Jogo Morgana.

2 Jogo disponibilizado para acesso no GitHub através deste link.

6 CONCLUSÃO

O presente projeto apresentou o desenvolvimento de um jogo digital educativo, com a finalidade de ensinar os princípios básicos da programação orientada a objetos de maneira envolvente e imersiva, fazendo uso da plataforma de desenvolvimento Unity. Para alcançar esse objetivo, foram empregadas técnicas de gamificação, as quais tornam o processo de aprendizagem mais envolvente. Essas estratégias criam um ambiente dinâmico e interativo, facilitando a compreensão e a aplicação dos conceitos abordados.

Desenvolveu-se uma narrativa ambientada em um contexto medieval. Em seguida, foram selecionados e categorizados os principais conceitos de POO ensinados por meio dessa história, distribuídos em três níveis de dificuldade correspondentes às fases do jogo. Essa estratégia visa oferecer aos jogadores uma experiência de aprendizado progressiva e adaptativa. A etapa de desenvolvimento do jogo foi a mais complexa e demorada, principalmente devido à falta de conhecimento prévio sobre a plataforma e o processo de criação de jogos, o que tornou o progresso mais difícil.

Enquanto isso, é importante observar que o mercado de jogos está em constante crescimento. Segundo Newzoo (2022), o Brasil se destacou como o terceiro maior mercado consumidor de jogos do mundo, com aproximadamente 102 milhões de jogadores. Em termos de receita, o país ficou em 10º lugar, movimentando cerca de 2,6 bilhões de dólares no mesmo ano. Esses números evidenciam a importância e o potencial significativo do setor de jogos. No entanto, a falta de oferta regular de oportunidades relacionadas ao desenvolvimento de jogos no curso pode indicar uma negligência em reconhecer e promover habilidades relevantes para os alunos em um campo de estudo tão próspero e dinâmico.

Ao finalizar o desenvolvimento, foram conduzidos testes para garantir a qualidade e a integridade do sistema. Por meio de testes de caixa-preta e análise detalhada dos resultados, identificaram-se áreas de melhoria e oportunidades de otimização em diversas interações-chave do jogo. As falhas encontradas durante os testes ressaltaram a necessidade contínua de revisão e aprimoramento do código, visando proporcionar uma experiência de usuário fluida e livre de problemas. Para solucionar tais falhas, foram aplicadas medidas como a inclusão de condições nos *scripts* do jogo, a fim de aprimorar tanto a funcionalidade quanto a consistência do sistema.

Como trabalhos futuros, percebe-se a oportunidade de realizar testes com alunos em uma disciplina, o que poderia ser efetuado por meio da execução de um estudo de caso com

um grupo de estudantes, para reforçar a validação da eficácia da ferramenta. Outra possibilidade seria expandir o jogo, adicionando mais fases, para abordar uma gama mais ampla de conceitos, como polimorfismo, agregação e interface.

A adição de efeitos sonoros está prevista como um elemento para aprimorar a experiência do usuário e aumentar o nível de imersão no jogo. Além disso, incorporar sistemas de pontuação conforme o avanço nas etapas e estabelecer um *ranking* competitivo entre os jogadores pode aumentar o engajamento dos alunos, motivando-os a aprimorar suas habilidades no jogo, tornando a experiência de aprendizado em programação ainda mais envolvente.

A implementação da funcionalidade de salvar o progresso do jogo pode possibilitar que os jogadores retomem a partir do ponto em que pararam, proporcionando uma experiência mais conveniente e permitindo que os usuários avancem de forma mais fluida. Além disso, a criação do jogo em uma resolução maior contribuirá para uma experiência visual imersiva para os jogadores.

Na segunda fase do jogo, um enigma é apresentado ao jogador, no qual as perguntas a serem respondidas são fixas. Embora esse método possa ajudar os alunos com dificuldades a melhorarem seu desempenho, também pode tornar a ferramenta inflexível. Como trabalho futuro, seria interessante implementar um banco de questões com uma variedade de perguntas relacionadas à disciplina de programação orientada a objetos. Isso permitiria uma experiência mais dinâmica e adaptável, oferecendo desafios diferentes em cada sessão de jogo, evitando a repetição excessiva de perguntas.

Finalmente, permitir que o jogador escolha a linguagem de programação para apresentar os conceitos no início do jogo, com os códigos correspondentes a essa linguagem, ampliará a flexibilidade do jogo e garantirá uma experiência mais personalizada para cada jogador, adaptando-se às preferências individuais e ao nível de familiaridade com diferentes linguagens de programação.

REFERÊNCIAS

ALBERTAZZI, Deise; FERREIRA, Marcelo G. G. F.; FORCELLINI, Fernando A. **A Wide View on Gamification**. Technology, Knowledge and Learning, p. 191–202, 2018. Disponível em: https://doi.org/10.1007/s10758-018-9374-z. Acesso em: 22 mai. 2023.

ALBUQUERQUE, Marcos R. E. de. **Processo de construção do jogo Tetris no Unity como incentivo à aprendizagem de programação**. Universidade Federal Rural do Semi-Árido. Monografia de Bacharel em Ciência e Tecnologia, Pudos Ferros, 2021. Disponível em: https://repositorio.ufersa.edu.br/handle/prefix/7772. Acesso em: 22 jun. 2023.

ALVES, Gelderson B. **Estudo comparativo entre engines de desenvolvimento de jogos 2D**. Trabalho de Conclusão de Curso (Graduação em Engenharia de Software)-Universidade Federal do Ceará, Campus de Quixadá, Quixadá, 2020. Disponível em: http://www.repositorio.ufc.br/handle/riufc/58827. Acesso em: 20 abr. 2023.

ALVES, Marcia. M.; BATTAIOLA, André. L.; CEZAROTTO, Matheus. A. Representação gráfica para a inserção de elementos da narrativa na animação educacion: Graphic representation for inserting narrative elements in educational animations. **InfoDesign**: Revista Brasileira de Design da Informação, v. 13, p. 1–21, 2016. Disponível em: https://www.infodesign.org.br/infodesign/article/view/424. Acesso em: 28 maio. 2023.

AMARAL, Laurence R. do; SILVA, Gláucia B. S.; PANTALEÃO, Eliana. **Simpósio Brasileiro de Informática na Educação -** SBIE. Maceió, 2015. Disponível em: https://www.researchgate.net/publication/284508496. Acesso em: 27 mai. 2023.

ASSETSTORE, **Warrior Free Asset**. 2021. Disponível em: https://assetstore.unity.com/packages/2d/characters/warrior-free-asset-195707#releases. Acesso em: 19 out. 2023.

ASSETSTORE, **Bringer Of Death (free)**. 2021. Disponível em: https://assetstore.unity.com/packages/2d/characters/bringer-of-death-free-195719#releases. Acesso em: 22 nov. 2023.

ASSETSTORE, **Warped Caves**. 2017. Disponível em: https://assetstore.unity.com/packages/2d/characters/warped-caves-103250#releases. Acesso em: 24 out. 2023.

BARNES, David J.; KÖLLING, Michael. **Programação Orientada a Objetos com Java:** Uma Introdução Prática usando o Blue J. São Paulo: Pearson Education, 2004. Acesso em: 23 jun. 2023

BARTIÉ, Alexandre. **Garantia da Qualidade de Software**: adquirindo maturidade organizacional. Rio de Janeiro, Elsevier, 2002.

BAX, Marcello P. **Design science:** filosofia da pesquisa em ciência da informação e tecnologia. Ciência da Informação, v. 42, n. 2, 2015. Disponível em: https://revista.ibict.br/ciinf/article/view/1388. Acesso em: 12 maio. 2023.

BRAGA, Juliana C. *et al.* **Desafios para o Desenvolvimento de Objetos de Aprendizagem Reutilizáveis e de Qualidade**. Anais do Desafie Workshop de Desafios da Computação Aplicada à Educação, Curitiba, P.1-11, 2012. Disponível em: http://www2.sbc.org.br/csbc2012/anais_csbc/eventos/desafie/artigos/desafie2012%20-%20Desafios%20para%20o%20Desenvolvimento%20de%20Objetos%20de%20Aprendizagem%20Reutilizaveis%20e%20de%20Qualidade.pdf. Acesso em: 20 abr. 2023.

CARVALHO, Marcio F. de *et al.* **Livro mágico da gamificação**. Porto Alegre, 2020. Disponível em: https://repositorio.ifrs.edu.br/xmlui/handle/123456789/214. Acesso em: 21 jun. 2023.

CAVALCANTE, Carlos H. L.; PEREIRA, Maria L. A. **Comparativo entre Game Engines como Etapa Inicial para o Desenvolvimento de um Jogo de Educação Financeira**. Congresso sobre Tecnologias na Educação, Fortaleza, Junho de 2018. Disponível em: http://ceur-ws.org/Vol-2185/CtrlE_2018_paper_110.pdf. Acesso em: 22 abr. 2023.

DA SILVA, Leuson M. *et al.* **POOGame**: Um Jogo Sério para o Ensino de Programação Orientada a Objetos. WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, 24. Porto Alegre: Sociedade Brasileira de Computação, 2016. p. 2333-2342. Disponível em: https://doi.org/10.5753/wei.2016.9677. Acesso em: 22 mai. 2023.

DAHL, Ole-Johan, MYHRHAUG, Bjorn.; NYGAARD, Kristen. **Simula 67 Common Base Language.** Norsk Regnesentral, 1968.

DETERDING, Sebastian *et al*. **From Game Design Elements to Gamefulness**: Defining Gamification. Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, 2011. Disponível em: 10.1145/2181037.2181040. Acesso em: 18 abr. 2023.

FALKEMBACH, Gilse A. M. **O Lúdico e os Jogos Educacionais.** Universidade Federal do Rio Grande do Sul, 2006. Disponível em: http://matpraticas.pbworks.com/w/file/fetch/85177681/Leitura_1.pdf. Acesso em: 19 mai. 2023.

FARRER, Harry. *et al.* **Programação Estruturada de Computadores** – Algoritmos estruturados. 3. ed. Rio de Janeiro: Gen/LTC, 1999.

GALAFASSI, Fabiane F. P.; GLUZ, João C.; GALAFASSI, Cristiano. **Análise Crítica das Pesquisas Recentes sobre as Tecnologias de Objetos de Aprendizagem e Ambientes Virtuais de Aprendizagem.** Revista Brasileira de Informática na Educação, 2014. Disponível em:https://www.researchgate.net/profile/Fabiane-Galafassi/publication/272703608_Analise_Critica_das_Pesquisas_Recentes_sobre_as_Tecnologias_de_Objetos_de_Aprendizagem_e_Ambientes_Virtuais_de_Aprendizagem/links/5a61de9baca272a158176fe7/Analise-Critica-das-Pesquisas-Recentes-sobre-as-Tecnologias-de-Objetos-de-Aprendizagem-e-Ambientes-Virtuais-de-Aprendizagem.pdf. Acesso em: 24 mai. 2023.

GERHARDT, Tatiana E.; SILVEIRA, Denise T. **Métodos de Pesquisa**. 1. ed. Porto Alegre: Editora da UFRGS, 2009. Disponível em: http://hdl.handle.net/10183/52806. Acesso em: 18 abr. 2023.

GODOT. **The game engine you've been waiting** for. Disponível em: https://godotengine.org/. Acesso em: 22 mai. 2023.

GREENFOOT. **About Greenfoot:** Software. Disponível em: https://www.greenfoot.org/overview. Acesso em: 25 mai. 2023.

GUEDES, Gilleanes T. A. **UML 2** - Uma abordagem prática. Novatec Editora, 2011.

HODGINS, Wayne. **The Future os Learning Objetcs**. e-Technologies in Engineering Education: Learning Outcomes Providing Future Possibilities. USA, p.281-298, 2002. Disponível em: https://dc.engconfintl.org/etechnologies/11. Acesso em: 22 mai. 2023.

HOLM, Ulrika; HUKE, Viktor. **Gamification in a Workshop Enviroment**. Chalmers University of Technology, p. 5–33, 2020. Disponível em: https://odr.chalmers.se/handle/20.500.12380/300993. Acesso em: 19 mai. 2023.

IEEE SPECTRUM. **The Top Programming Languages 2023**. 2023. Disponível em: https://spectrum.ieee.org/the-top-programming-languages-2023. Acesso em: 07 mar. 2024.

ITCH.IO. **Generic assets pack series**. 2022. Disponível em: https://bakudas.itch.io/generic-dungeon-pack. Acesso em: 5 nov. 2023.

TORETTI JÚNIOR, Nelson. **Desenvolvimento de Jogos Digitais 2D com Unity3D**. Faculdade de Tecnologia de Americana. Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais, Americana, 2013. Disponível em: https://ric.cps.sp.gov.br/bitstream/123456789/1258/1/20132S_TORETTIJUNIORNelson_TCCPD1222.pdf. Acesso em: 19 mai. 2023.

KAPP, Karl. **The gamification of learning and instruction**: Game-based methods and strategies for training and education. Pfeiffer, San Francisco, 2012. Disponível em: 9781118096345. Acesso em: 18 abr. 2023.

KÖLLING, Michael. **The problem of teaching object-oriented programming, Part 1**: Languages. Journal of Object-Oriented Programming, v. 11, p. 8-15, 1999. Disponível em: https://www.researchgate.net/publication/220278819_The_Problem_of_Teaching_Object-Oriented_Programming_Part_I_Languages. Acesso em 19 abr. 2023.

KOSCIANSKI, André; SOARES, Michel dos S. **Qualidade de Software**: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. Ed. 2, Novatec Editora, 2007.

KLOPFER, Eric *et al.* **Moving learning games forward obstacles, opportunities and openness.** The Education Arcade, Massachusetts Institute of Technology, 2009. Disponível em: https://education.mit.edu/wp-content/uploads/2018/10/MovingLearningGamesForward_EdArcade.pdf. Acesso em 19 abr. 2023.

MONQUEIRO, Julio C. B. **Programação Orientada a Objetos:** uma introdução. Hardware, 2007. Disponível em: https://www.hardware.com.br/artigos/programacao-orientada-objetos/. Acesso em: 20 abr. 2023.

NELSON, Mark J. **Soviet and American Precursors to the Gamification of Work.** Proceedings of the 16th International Academic MindTrek Conference, p. 23-26, 2012. Disponível em: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2115483. Acesso em: 25 mai. 2023.

NEWZOO, **Top countries and markets by video game revenue**, 2022. Disponível em: https://newzoo.com/resources/rankings/top-10-countries-by-game-revenues. Acesso em: 02 fev. 2024.

OPENGAMEART, **SunnyLand Forest of Illusion**. 2022. Disponível em: https://opengameart.org/content/sunnyland-forest-of-illusion. Acesso em: 19 out. 2023.

PEFFERS, Ken *et al*. **A design science research methodology for information systems research. Journal of Management Information Systems**, 24. p. 45-77, 2007. Disponivel em: https://www.researchgate.net/publication/284503626. Acesso em: 26 mai. 2023.

PIMENTA, André S. G. M. **Uma Arquitetura de Modularização de Componentes para Desenvolvimento de jogos sérios na Unity Engine.** Universidade Federal de Ouro Preto. Monografia apresentada ao Curso de Ciência da Computação, Ouro Preto, 2022. Disponível em: http://www.monografias.ufop.br/handle/35400000/4346. Acesso em: 19 jun. 2023.

PRESSMAN, Roger S. **Engenharia de software: uma abordagem profissional**. 7. ed. Porto Alegre: AMGH, 2011.

REBOUÇAS, Ayla D.; MAIA, Dennys L.; SCAICO, Pasqueline D. **Objetos de Aprendizagem:** da Definição ao Desenvolvimento, Passando pela Sala de Aula. Informática na Educação: ambientes de aprendizagem, objetos de aprendizagem e empreendedorismo, Porto Alegre (Série Informática na Educação, v.5), 2021. Disponível em: http://ieducacao.ceie-br.org/objetos-aprendizagem. Acesso em 17 abr. 2023.

RODRIGUES, Luciene C.; NOGUEIRA, Giovani C.; QUEIROGA, Ana. **Experiências no ensino de Programação Orientada a Objetos:** RoboCode, Greenfoot e Jogos de Tabuleiro no Ensino Superior. XXIII Workshop de Informática na Escola, p. 10, 2017. Disponível em: https://sol.sbc.org.br/index.php/wie/article/view/16295. Acesso em: 22 mai. 2023.

SANTOS, Rafael. **Introdução à Programação Orientada a Objetos usando Java**. 2. ed. Rio de Janeiro: Elsevier, 2003.

SEBESTA, Robert. W. **Conceitos de Linguagens de Programação.** 9. ed. Porto Alegre: Bookman, 2011. ISBN 9788577807918.

SILVA FILHO, Gerôncio O. da *et al.* **Unity**: Criando jogos e outras aplicações multi-plataforma. Introdução ao desenvolvimento de softwares para analistas do comportamento, p.138-155, 2018. Disponível em: https://www.researchgate.net/profile/Hernando-Neves-Filho/publication/323905795_Unity_Criando_jogos_e_outras_aplicacoes_multi-plataforma/links/5ab1eb52aca2721710ffd26a/Unity-Criando-jogos-e-outras-aplicacoes-multi-plataforma.pdf. Acesso em: 26 mai. 2023.

SILVA, Lays R. A. da; QUEIROZ, Ruy J. G. B, de. **Aprendizagem baseada em jogos**: Uma reflexão sobre o modelo de currículo da Quest to Learn. WORKSHOP DE INFORMÁTICA NA ESCOLA, p. 86-90, 2014. Disponível em: https://doi.org/10.5753/cbie.wie.2014.86. Acesso em: 18 abr. 2023.

SOMMERVILLE, Ian. **Engenharia de Software.** Pearson, São Paulo, 2011. ed. 9.

SCHWABER, Ken; SUTHERLAND, Jeff. **O Guia Definitivo para o Scrum:** As Regras do Jogo. 2020. Disponível em: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Portuguese-European.pdf. Acesso em: 26 mai. 2023.

SCRUM. **What is Scrum?**. Disponível em: https://www.scrum.org/learning-series/what-is-scrum. Acesso em: 25 mai. 2023.

UNITY. **Unity User Manual: Scenes**. 2022. Disponível em: https://docs.unity3d.com/Manual/CreatingScenes.html. Acesso em: 15 out. 2023.

VAREJÃO, Flávio M. **Linguagens de Programação**: Conceitos e Técnicas. Elsevier, 2004.

VAHLDICK, Adilson. **Uma experiência lúdica no ensino de programação orientada a objetos.** Anais do XVIII Simpósio Brasileiro de Informática na Educação (SBIE), São Paulo, p. 1-8, 2007. Disponível em: https://www.researchgate.net/publication/267767400_Uma_Experiencia_Ludica_no_Ensino_de_Programacao_Orientada_a_Objetos. Acesso em: 19 abr. 2023.

WAZLAWICK, Raul. S. **Metodologia de Pesquisa para Ciência da Computação.** Rio de Janeiro: Elsevier, 2009. v. 3. Acesso em: 12 mai. 2023.

WILEY, David A. **Learning object design and sequencing theory**. 2000. Ph.D. thesis, Brigham Young University, United States. Disponível em: https://www.learntechlib.org/p/120055/. Acesso em: 19 abr. 2023.

APÊNDICES

APÊNDICE A – HISTÓRIA DO JOGO

Fase 1 – Os Mistérios da Floresta

Morgana é uma jovem sonhadora que vive em um reino mágico. Um dia, enquanto procurava maçãs pela floresta para a ceia de sua família, que enfrentava dificuldades devido à colheita ruim daquele ano, ela tem um encontro inesperado: um mago vestido como nas histórias, com vestes compridas e um cajado reluzente. Ele se apresenta como Magnus e diz que as adversidades que assolam sua vila não são por acaso; um monstro temido despertou de um período sombrio do passado e requer a ajuda de Morgana para salvar o reino.

O sábio oferece a Morgana a chance de escolher uma classe com poderes mágicos para iniciar essa aventura. Antes de começar sua jornada, ele lhe proporciona a oportunidade de selecionar sua arma de combate. Morgana pode optar entre um cajado ou uma espada, sendo que cada opção possui atributos e habilidades específicas.

Após Morgana selecionar sua arma, Magnus revela a existência de uma caverna secreta, conhecida apenas por alguns moradores da região. Para auxiliá-la a encontrá-la, Morgana recebe a orientação de buscar cinco flores na floresta, que serão utilizadas para preparar uma poção capaz de fornecer uma visão clara do caminho. Morgana parte em busca das flores e, ao reunir os itens necessários, retorna ao mago para entregá-los.

O Magnus então entrega a poção a ela, que a auxilia a visualizar o caminho para a caverna, e instrui Morgana a encontrá-la e enfrentar o guardião, que lhe concederá um cristal poderoso para aprimorar sua arma. No entanto, o sábio enfatiza a importância de controlar e proteger essa essência mágica. Ao ingerir a poção, Morgana desperta na caverna.

Fase 2 – Caverna dos Cristais

Dentro da caverna, Morgana encontra Gux, o guardião, que se apresenta e explica que, para continuar acessando a caverna e obter acesso ao cristal mágico, ela deve demonstrar sabedoria. Com um aceno de sua mão, Gux conjura três pergaminhos, cada um contendo uma pergunta intricada sobre POO, desafiando os limites do conhecimento de Morgana. Cada resposta correta parece desbloquear uma parte do enigma da caverna, revelando segredos antigos que ecoam através dos séculos.

Após responder corretamente, Gux entrega o cristal a ela e, em seguida, instrui Morgana sobre os conceitos de *getter* e *setter*. Ele explica como o cristal permite acessar e

modificar atributos da arma, enfatizando a importância de proteger o acesso direto aos atributos (encapsulamento) e utilizando métodos para controlar as modificações dos valores.

Em seguida, Morgana parte em direção ao altar e deposita o cristal. De repente, um pergaminho aparece diante dela, inscrito com códigos misteriosos em Java. Após essa interação, Morgana deve seguir para o túnel no final da caverna e avançar em direção ao castelo, onde o monstro a espera.

Fase 3 – O Desafio do Monstro

Morgana avança em sua jornada até alcançar um majestoso castelo, conforme Magnus havia informado que um poderoso monstro habitava esse lugar, e que Morgana teria que enfrentá-lo para concluir o jogo. Ao entrar no castelo, Morgana descobre pergaminhos ocultos em um baú, revelando um importante segredo: apenas o herdeiro dos guerreiros mais antigos possui o poder necessário para derrotar o temível monstro.

Para sua surpresa, ao ler os pergaminhos, ela descobre que tem em seu sangue a linhagem desse lendário guerreiro, herdando assim poderes especiais. Agora, confiante em sua herança, Morgana está pronta para utilizar suas habilidades especiais e demonstrar ao mundo que é digna do legado dos guerreiros mais antigos. O destino do jogo está em suas mãos, e ela está determinada a vencer.

O monstro final é um portador da morte, e Morgana luta bravamente para derrotá-lo. Após vencer a batalha, Morgana retorna à sua vila para reunir-se com sua família, sendo recebida com muitas comemorações e agradecimentos.

APÊNDICE B – CÓDIGOS DO JOGO

Este apêndice contém imagens com os trechos de todos os códigos na linguagem Java que o jogador tem acesso durante o jogo, conforme descrito na Seção 5.1.

Código 1 – Instância Espada

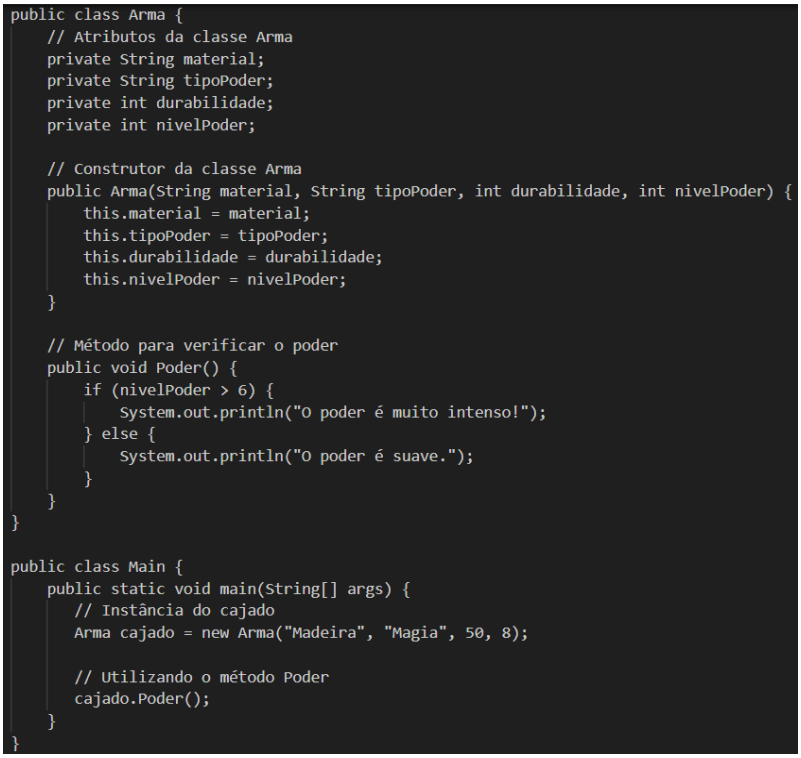
Figura 39 - Código Espada

```
public class Arma {
    // Atributos da classe Arma
    private String material;
    private String tipoPoder;
    private int durabilidade;
    private int nivelPoder;

    // Construtor da classe Arma
    public Arma(String material, String tipoPoder, int durabilidade, int nivelPoder) {
        this.material = material;
        this.tipoPoder = tipoPoder;
        this.durabilidade = durabilidade;
        this.nivelPoder = nivelPoder;
    }

    // Método para verificar o poder
    public void Poder() {
        if (nivelPoder > 6) {
            System.out.println("O poder é muito intenso!");
        } else {
            System.out.println("O poder é suave.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        // Instância da espada
        Arma espada = new Arma("Aço", "Corte", 70, 5);

        // Utilizando o método "verificarPoder"
        espada.Poder();
    }
}
```

Fonte: Elaborado pela autora, 2024.

Código 2 – Instância Cajado

Figura 40 - Código Cajado

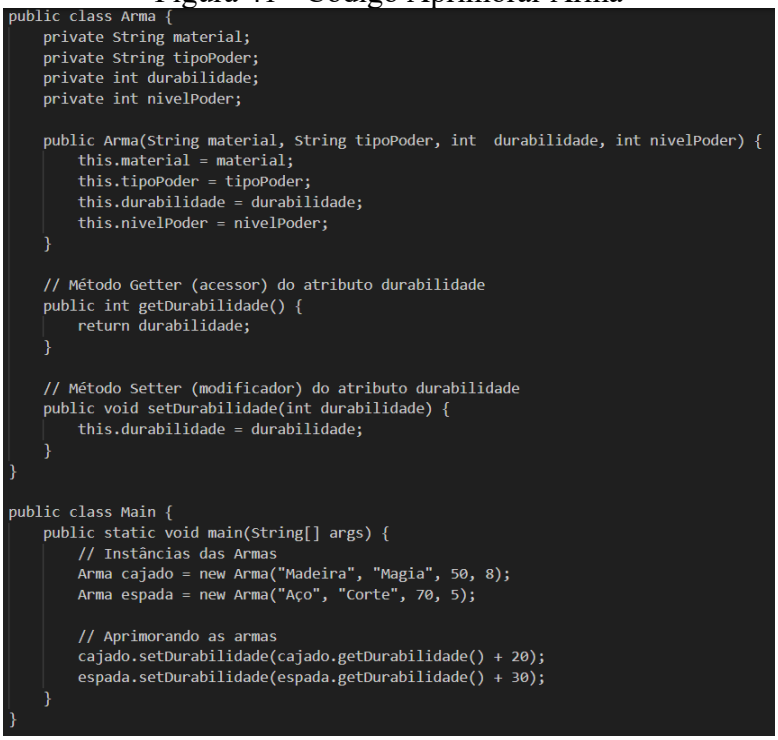


```
public class Arma {
    // Atributos da classe Arma
    private String material;
    private String tipoPoder;
    private int durabilidade;
    private int nivelPoder;

    // Construtor da classe Arma
    public Arma(String material, String tipoPoder, int durabilidade, int nivelPoder) {
        this.material = material;
        this.tipoPoder = tipoPoder;
        this.durabilidade = durabilidade;
        this.nivelPoder = nivelPoder;
    }

    // Método para verificar o poder
    public void Poder() {
        if (nivelPoder > 6) {
            System.out.println("O poder é muito intenso!");
        } else {
            System.out.println("O poder é suave.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
       // Instância do cajado
       Arma cajado = new Arma("Madeira", "Magia", 50, 8);

       // Utilizando o método Poder
       cajado.Poder();
    }
}
```

Fonte: Elaborado pela autora, 2024.

Código 3 – Aprimorar Arma

Figura 41 - Código Aprimorar Arma

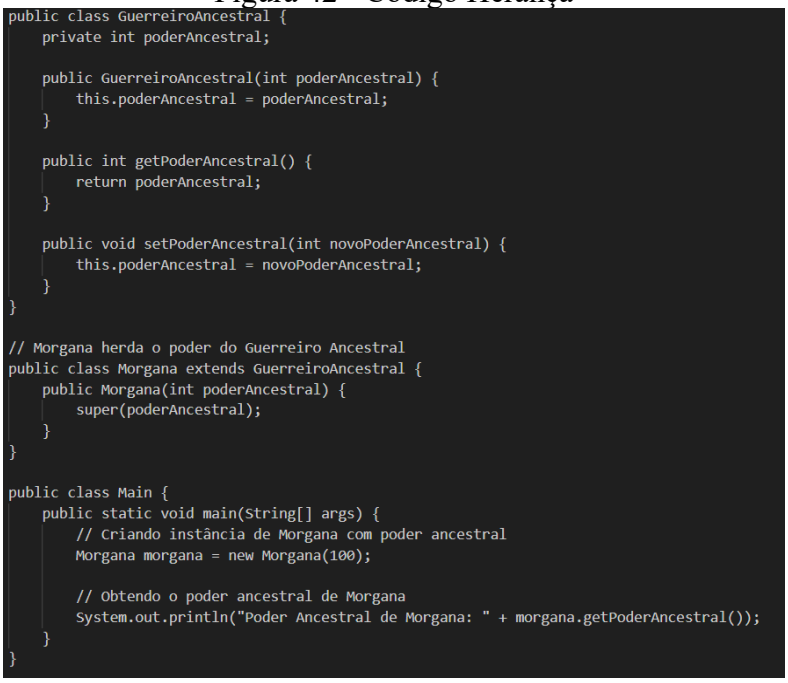


```
public class Arma {
    private String material;
    private String tipoPoder;
    private int durabilidade;
    private int nivelPoder;

    public Arma(String material, String tipoPoder, int  durabilidade, int nivelPoder) {
        this.material = material;
        this.tipoPoder = tipoPoder;
        this.durabilidade = durabilidade;
        this.nivelPoder = nivelPoder;
    }

    // Método Getter (acessor) do atributo durabilidade
    public int getDurabilidade() {
        return durabilidade;
    }

    // Método Setter (modificador) do atributo durabilidade
    public void setDurabilidade(int durabilidade) {
        this.durabilidade = durabilidade;
    }
}

public class Main {
    public static void main(String[] args) {
        // Instâncias das Armas
        Arma cajado = new Arma("Madeira", "Magia", 50, 8);
        Arma espada = new Arma("Aço", "Corte", 70, 5);

        // Aprimorando as armas
        cajado.setDurabilidade(cajado.getDurabilidade() + 20);
        espada.setDurabilidade(espada.getDurabilidade() + 30);
    }
}
```

Fonte: Elaborado pela autora, 2024.

Código 4 – Herança Morgana

Figura 42 - Código Herança



```
public class GuerreiroAncestral {
    private int poderAncestral;

    public GuerreiroAncestral(int poderAncestral) {
        this.poderAncestral = poderAncestral;
    }

    public int getPoderAncestral() {
        return poderAncestral;
    }

    public void setPoderAncestral(int novoPoderAncestral) {
        this.poderAncestral = novoPoderAncestral;
    }
}

// Morgana herda o poder do Guerreiro Ancestral
public class Morgana extends GuerreiroAncestral {
    public Morgana(int poderAncestral) {
        super(poderAncestral);
    }
}

public class Main {
    public static void main(String[] args) {
        // Criando instância de Morgana com poder ancestral
        Morgana morgana = new Morgana(100);

        // Obtendo o poder ancestral de Morgana
        System.out.println("Poder Ancestral de Morgana: " + morgana.getPoderAncestral());
    }
}
```

Fonte: Elaborado pela autora, 2024.