Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Desenvolvimento de um jogo educacional para ensino de programação básica

Hugo Josue Lema das Neves

Dr. Diogo Fernando Trevisan (Orientador)

Dourados - MS
2025

Desenvolvimento de um jogo educacional para o ensino de programação básica

Hugo Josue Lema das Neves

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso II devidamente corrigida e defendida por Hugo Josue Lema das Neves e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Dourados, novembro de 2025.

Prof. Dr. Diogo Fernando Trevisan
(Orientador)

N424d Neves, Hugo Josue Lema das.

Desenvolvimento de um jogo educacional para ensino de programação básica / Hugo Josue Lema das Neves. – Dourados, MS: UEMS, 2025.
57 p.

Monografia (Graduação) – Sistemas de Informação – Universidade Estadual de Mato Grosso do Sul, 2025.
Orientador: Prof. Dr. Diogo Fernando Trevisan.

1. Jogos educacionais 2. Programação - Estudo e ensino 3. Lógica de programação 4. Jogos digitais I. Trevisan, Diogo Fernando II. Título

CDD 23. ed. - 005.13

Ficha Catalográfica elaborada pela bibliotecária da Universidade Estadual de Mato Grosso do Sul (UEMS)
Bruna Peruffo Vieira – CRB 1/2959

Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Desenvolvimento de um jogo educacional para ensino de programação básica

Hugo Josue Lema das Neves

Novembro de 2025

Banca Examinadora:

Prof. Dr. Diogo Fernando Trevisan (Orientador)
Área de Computação - UEMS

Prof. Dr. Evandro Cesar Bracht
Área de Computação - UEMS

Prof. Dr. Ricardo Luís Lachi
Área de Computação - UEMS

Dedico esta conquista à minha família, fonte de força e carinho, que permaneceu inabalável no apoio em todos os momentos, superando as dificuldades ao meu lado. Aos meus pais, por sua paciência exemplar e por cada ensinamento que moldou meu caminho. Esta realização é o fruto do apoio incondicional e da crença que sempre depositaram em mim.

"Construir é uma forma poderosa de aprender. Quando construímos um programa, estamos também construindo e refinando nosso entendimento sobre a lógica."

Seymour Papert

Agradecimentos

Expresso minha profunda gratidão à família, meu porto seguro, pelo incondicional apoio emocional e incentivo constante. Em especial, aos meus amados pais, Lucio Vladimir Lema Padilla e Luciane Canavarro Das Neves Lema, que representaram a estrutura base e a fonte de força para superar os desafios e prosseguir nesta jornada.

Ao Professor Doutor Diogo Fernando Trevisan, meu orientador, pela dedicação, paciência e valiosas contribuições. Suas sugestões técnicas, orientações precisas e correções rigorosas foram essenciais para garantir a estruturação e o rigor acadêmico deste projeto.

A todos os professores do curso de Sistemas de Informação da UEMS, pela transmissão do conhecimento que fundamentou minha formação profissional.

Finalmente, aos amigos e colegas, pelo apoio e companheirismo ao longo dos anos de graduação.

Resumo

O ensino de programação básica ainda representa um desafio, especialmente para inciantes. Nesse contexto, os jogos digitais surgem como uma alternativa didática eficaz, ao oferecerem um ambiente interativo e motivador que permite ao estudante aprender enquanto pratica. Este trabalho apresenta o desenvolvimento de um jogo educacional em 2D, com o objetivo de auxiliar no ensino de estruturas fundamentais da programação. O jogo foi projetado para introduzir de forma progressiva conceitos como sequência de comandos, estruturas condicionais, laços de repetição e funções, utilizando blocos de código no estilo *drag and drop*.

Como base para o projeto, são analisados exemplos históricos e contemporâneos de jogos e ferramentas voltados ao ensino de programação, como LOGO, Scratch e Grubibots, além da escolha da Godot Engine como plataforma de desenvolvimento.

Palavras-chave: jogos educacionais; programação básica; lógica de programação; Godot Engine; ensino.

Abstract

Teaching basic programming still represents a challenge, especially for beginners. In this context, digital games emerge as an effective teaching alternative, offering an interactive and motivating environment that allows students to learn while practicing. This work presents the development of a 2D educational game, focusing on assisting in the teaching of fundamental programming structures. The game was designed to progressively introduce concepts such as command sequences, conditional structures, repetition loops, and functions, using drag-and-drop code blocks.

As a basis for the project, historical and contemporary examples of games and tools focused on teaching programming, such as LOGO, Scratch, and Grubibots, are analyzed, along with the choice of the Godot Engine as the development platform.

Keywords: educational games; basic programming; programming logic; Godot Engine; teaching.

Sumário

Lista de Figuras

1. Introdução

O aprendizado de programação ainda representa um grande desafio. Muitas vezes, o primeiro contato com noções de programação e lógica ocorre em etapas posteriores, especialmente no Brasil, onde disciplinas de programação são vistas pela primeira vez no ensino superior ou em cursos de Ensino médio Técnico, o que não é acessível a todos os estudantes.

As estruturas básicas de programação, como variáveis, condicionais, repetições e funções são fundamentais para a compreensão da lógica computacional, mas podem ser abstratas e pouco atrativas quando apresentadas de forma tradicional. Nesse cenário, os jogos digitais surgem como uma alternativa lúdica e interativa para introduzir esses conceitos de maneira prática, permitindo que o aluno aprenda enquanto experimenta soluções e testa suas hipóteses.

A experiência em utilizar jogos educacionais para o ensino de programação baseia-se em marcos como a linguagem LOGO Abelson e DiSessa (1986), que introduziu o aprendizado de conceitos computacionais por meio de um ambiente interativo e posteriormente ferramentas como o Scratch, que possibilitaram estudantes a desenvolver projetos com blocos de programação visuais Resnick et al. (2009). Esses exemplos ilustram que a adoção de uma interface visual e interativa facilita a construção algorítmica.

Nesse contexto, a importância deste projeto está em propor o uso de uma ferramenta gráfica e interativa visando auxiliar o ensino de conteúdos teóricos inerentes ao ensino de programação.

Com base nessa proposta, o objetivo geral deste trabalho foi desenvolver um jogo educacional 2D para ensino de programação básica por meio de blocos de código que representem estruturas fundamentais de sequências de comandos, condicionais, repetições e funções.

Para alcançar esse objetivo geral, foram definidos os seguintes objetivos específicos, que detalham as etapas desenvolvidas neste trabalho:

- Definir a mecânica do jogo baseada em blocos de código no estilo *drag and drop*;
- Projetar e implementar os blocos que representem estruturas básicas da programação;

- Criar fases progressivas que introduzam gradualmente conceitos de programação;
- Desenvolver uma interface simples que facilite a interação do jogador com os blocos;
- Escolher as ferramentas e tecnologias adequadas para o desenvolvimento;
- Disponibilizar o jogo como *software* de código aberto, permitindo sua expansão e adaptação por outros desenvolvedores ou educadores.

Considerando os objetivos propostos, o desenvolvimento do trabalho foi organizado do seguinte modo: no **Capítulo 2** é feita uma revisão bibliográfica revisando estudos, trabalhos, jogos e ferramentas correlatos que fundamentam o desenvolvimento deste projeto. Em seguida, o **Capítulo 3** detalha o processo de desenvolvimento do jogo, incluindo as etapas de criação, ferramentas e tecnologias utilizadas, estrutura do jogo e implementação técnica das principais funcionalidades. Por fim, o **Capítulo 4** apresenta a conclusão final do projeto, avaliando os resultados obtidos, propondo melhorias e trabalhos futuros.

2. Revisão da Literatura

Nesta seção, é apresentada a revisão da literatura com o objetivo de analisar os conceitos e estudos mais relevantes que embasam o desenvolvimento do jogo.

A estrutura desta seção é organizada em subseções: a **subseção 2.1** traz jogos e ferramentas para ensino de programação. A **subseção 2.2** traz um levantamento da justificativa da escolha da *engine* para o desenvolvimento do jogo no trabalho. Por fim, a **subseção 2.3** apresenta as interfaces e funcionalidades da *engine* escolhida para o desenvolvimento do jogo.

2.1 Jogos e ferramentas para ensino de programação

O uso de jogos digitais como recurso educacional tem se expandido nas últimas décadas. Além de entreter, os jogos oferecem um ambiente interativo que favorece a aprendizagem ativa, permitindo que o estudante experimente, erre, ajuste e aprenda de forma prática.

Os autores Savi e Ulbricht (2008) relataram alguns resultados observados durante o processo de jogar algum tipo de jogo educacional, tais como a motivação, facilidade no aprendizado, desenvolvimento de habilidades cognitivas, aprendizado por descoberta, experiências de novas identidades, socialização, coordenação motora e comportamento expert. Isso leva à conclusão que jogos educacionais são ferramentas poderosas capazes de favorecer o aprendizado de diferentes indivíduos por meio das estratégias aplicadas em seu desenvolvimento. Nesse sentido, os jogos educacionais têm a capacidade de ensinar a teoria e prática integradas, eficazes ao aplicar o ensino de disciplinas, como a programação. Nessa área, os jogos assumem papel relevante, pois tornam mais acessíveis conceitos que, em métodos tradicionais, tendem a ser abstratos. Além disso, por meio de desafios progressivos, permitem que iniciantes compreendam gradualmente as estruturas básicas da lógica de programação.

Um marco pioneiro no ensino de programação foi a criação da linguagem de programação LOGO. Desenvolvida por Seymourt Papert e colaboradores do Massachussets Institute of Technology (MIT) durante a década de 1960, a linguagem LOGO utiliza a concepção de gráficos para ensinar

conceitos matemáticos e lógicos de forma intuitiva e visual (Papert, 2020). O estudante controla uma tartaruga que é desenhada em um espaço gráfico, movendo-a usando códigos simples como *forward* (avançar), *backward* (retroceder), *rigth* (direita) e *left* (esquerda), a execução desses comandos resulta na trajetória da tartaruga, que é representada na tela. O objetivo dos autores teve como propósito possibilitar que os estudantes interagissem com conceitos matemáticos e lógicos de forma concreta e visual, preservando a criatividade e incentivando o raciocínio próprio na resolução de problemas.

Dando continuidade a essa proposta, Abelson e DiSessa (1986) aprofundaram os estudos sobre o potencial da linguagem LOGO em *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*, onde os autores mostram como a tartaruga pode ser utilizada para demonstrar conceitos matemáticos mais complexos. Os autores destacaram que com a movimentação da tartaruga na tela, proporcionando figuras, auxiliava aos alunos a compreender conceitos de lógica e geometria, permitindo que ideias fossem exploradas por meio da experimentação prática, com os alunos aplicando seu próprio raciocínio ao programar a tartaruga.

Na **Figura 2.1** são mostrados exemplos de trajetórias realizadas pela tartaruga através de sua movimentação que geram desenhos de ângulos e triângulos. A abordagem visual introduzida pela linguagem LOGO possibilitou que conceitos ensinados anteriormente de maneira abstrata e rudimentar, agora pudessem ser ensinadas de forma prática e visual.

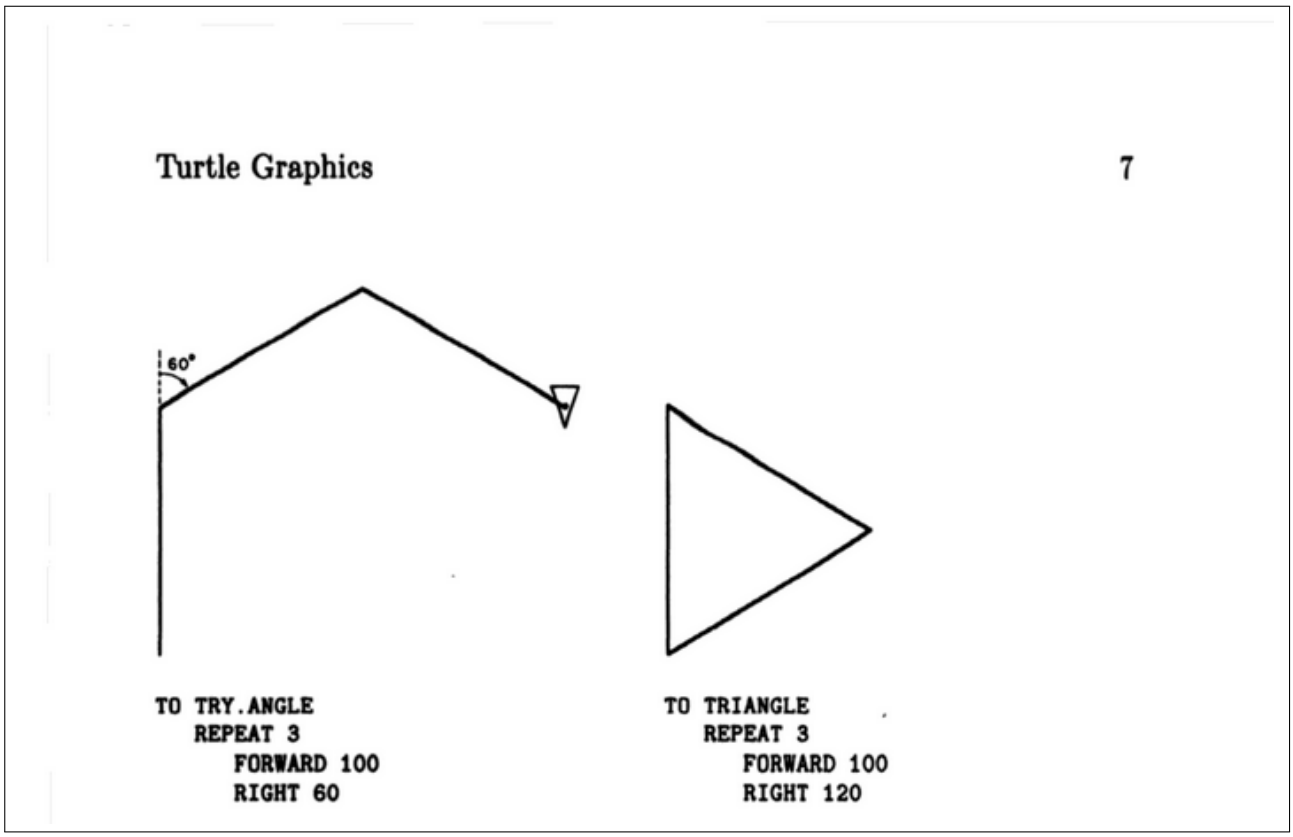

Figura 2.1 – Figuras projetadas pelo movimento da tartaruga na linguagem LOGO.
Fonte: Abelson e DiSessa (1986, p. 7).

A partir dessa abordagem visual pioneira, consolidou-se a compreensão de que, para o aprendizado da programação, é essencial o desenvolvimento de habilidades de raciocínio lógico, fundamentais para a solução eficiente de problemas. Essa lógica algorítmica é a base da computação, sendo traduzida na capacidade de definir uma sequência de passos necessários para atingir um objetivo. Ao praticar a programação, o estudante é naturalmente levado a aprimorar essa forma de pensar, o que manifesta na habilidade de estruturar e executar algoritmos.

Em consequência direta dessa prática algorítmica, o estudante passa a desenvolver um conjunto mais amplo de habilidades de resolução de problemas, como a decomposição e a abstração. Esta coleção de habilidades é comumente referenciada na literatura como Pensamento Computacional (PC), um conceito amplamente discutido e popularizado por Jeannette Wing (WING, 2006). Portanto, o foco primário do ensino de programação por meio de jogos reside na construção lógica, sendo o desenvolvimento do Pensamento Computacional (PC) um efeito colateral benéfico e desejado dessa prática.

Nesse contexto, os jogos educacionais, como a própria linguagem LOGO, são ferramentas de excelência. Eles exigem que o jogador aplique e estruture algoritmos de forma prática para superar desafios lúdicos, tornando a resolução de problemas um processo engajador e concreto, não apenas teórico.

Com o avanço da tecnologia, surgiram novas plataformas voltadas ao ensino introdutório de programação. Um ambiente de desenvolvimento de jogos bastante conhecido é o Scratch, proposto por Resnick et al. (2009). O Scratch permite a visualização de blocos que representam partes de conceitos da programação, como repetições, condicionais, variáveis, etc. Nela é possível criar jogos completos dentro do ambiente de desenvolvimento. A **Figura 2.2** apresenta o ambiente de desenvolvimento de um projeto na plataforma Scratch, onde todos os blocos estão disponíveis aos usuários ao lado esquerdo da tela, contendo blocos para movimento, rotação, condição, etc. Além dos blocos básicos, existem blocos que emitem efeitos sonoros, controlam valor de variáveis e sensores para aproximação de um elemento com outro. Ao lado direito há a área de visualização, podendo ser ilustrado o desenvolvimento do jogo conforme a montagem dos blocos de código.

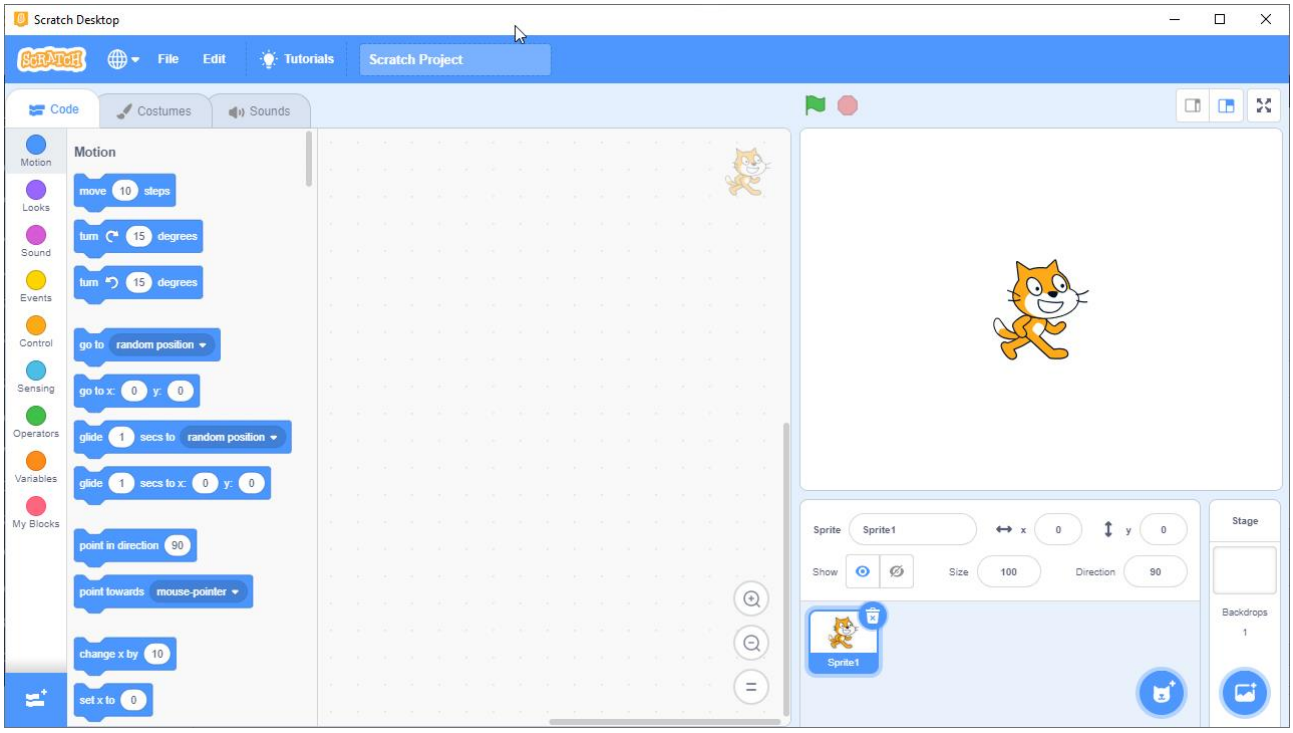

Figura 2.2 – Imagem do ambiente de desenvolvimento Scratch.
Fonte: Captura de tela realizada pelo autor na plataforma (MIT Media Lab, 2025).

É relevante ressaltar que essa abrangência permite que o indivíduo crie por meio dos blocos disponíveis seu próprio jogo com a sua própria linha de raciocínio, ou seja, o desenvolvimento do próprio jogo em questão é um aprendizado, pois o criador do jogo deve aprender a lógica trabalhada por trás dos blocos disponíveis, obtendo aprendizado enquanto cria ou joga. Brennan e Resnick (2012) enfatizam em seu estudo que se interessam em despertar a capacidade criativa dos jovens, no contexto a plataforma Scratch desenvolvida proporciona um ambiente de programação que permite aos jovens criar suas próprias histórias interativas, jogos e simulações, além de compartilhar suas criações com a comunidade, promovendo um aprendizado colaborativo e criativo.

Outro exemplo de jogo é o Grubibots, desenvolvido por de Oliveira, G. A., de Bettio, R. W., Rodarte, A. P. e Ferrari, F. B. (2014), é outro exemplo prático de como jogos podem ser usados para ensinar programação básica. Grubibots torna acessível o aprendizado de forma intuitiva e educativa em sua metodologia. Nele, os jogadores manipulam blocos de código, que ao serem selecionados e soltos em uma determinada área de execução, formam uma sequência de ações. Isso permite aos usuários a visualização do passo a passo que eles mesmo construíram, incentivando a compreensão prática dos conceitos de programação.

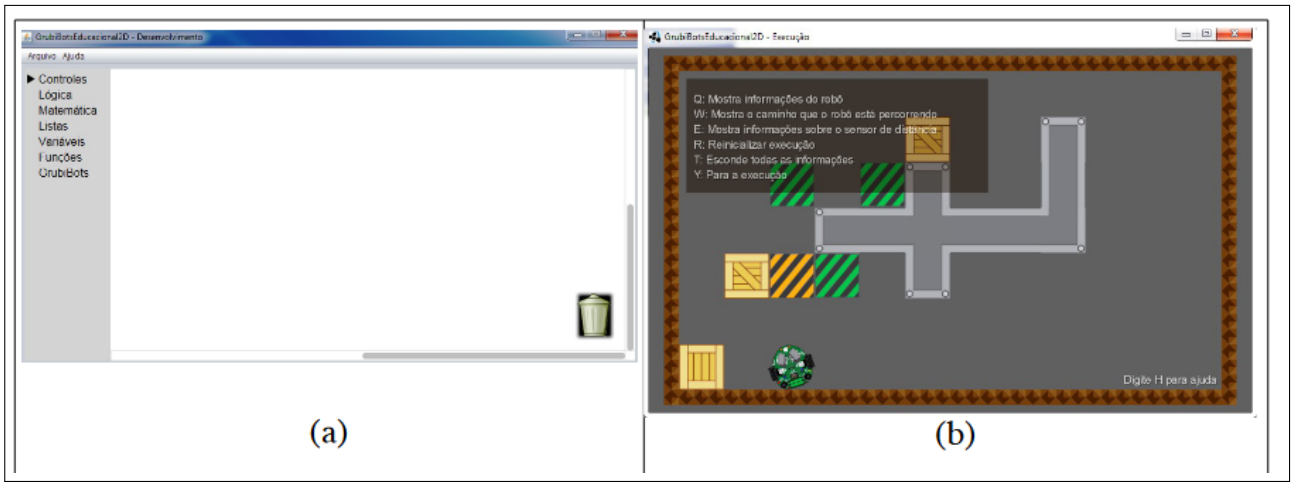

Figura 2.3 – (a) Ambiente de desenvolvimento, (b) Ambiente de execução.
Fonte: de Oliveira et al. (2014, p. 587).

Na **Figura 2.3-(a)** é apresentada a área de desenvolvimento, onde são exibidos o controle de ferramentas, que representam as funcionalidades do jogo. Nessa área os jogadores podem criar algoritmos por meio de blocos interconectados que contém instruções e estruturas lógicas. Por sua vez a **Figura 2.3-(b)** refere-se a área de execução dos algoritmos, que se desenrola em um ambiente bidimensional. Esse espaço inclui componentes como blocos tracejados, representando o início e fim de um trajeto e caixas fechadas, as quais representam obstáculos. Assim como outros jogos educacionais, a medida que o jogo avança e apresenta novos conceitos ao jogador, o nível de dificuldade aumenta. Esse tipo de progressão na dificuldade é essencial para determinar o grau de aprendizado que esta sendo absorvido pelo jogador e para desenvolver suas habilidades computacionais. Além de tornar acessível e divertido o aprendizado, Grubibots reforça os conceitos centrais da programação e facilita a transição do ambiente do jogo para situações reais que envolvem a capacidade de resolução de problemas (de Oliveira et al., 2014).

Outro jogo, intitulado O Sequestro de Magrafo, desenvolvido por Alencar et al. (2020), tem o objetivo de auxiliar as crianças usando o conceito de grafos, bastante úteis na computação. Nesse trabalho, foi realizado uma avaliação heurística para medir a eficácia do jogo na prática com estudantes. Esses dados foram separados por métricas, onde cada métrica era de acordo com algo relacionado ao jogo, sendo uma das métricas de mais sucesso apresentadas foi a Confiança dos alunos ao interagir com o jogo. Essa métrica que se refere ao nível de confiança e segurança dos alunos em sua capacidade de resolver desafios propostos, obteve 90% de aprovação entre concordo parcialmente e concordo totalmente que se sentiam confiantes durante a experiência. A **Figura** 2.4 ilustra esses resultados e demostra o impacto positivo do jogo no engajamento e aprendizagem dos estudantes.

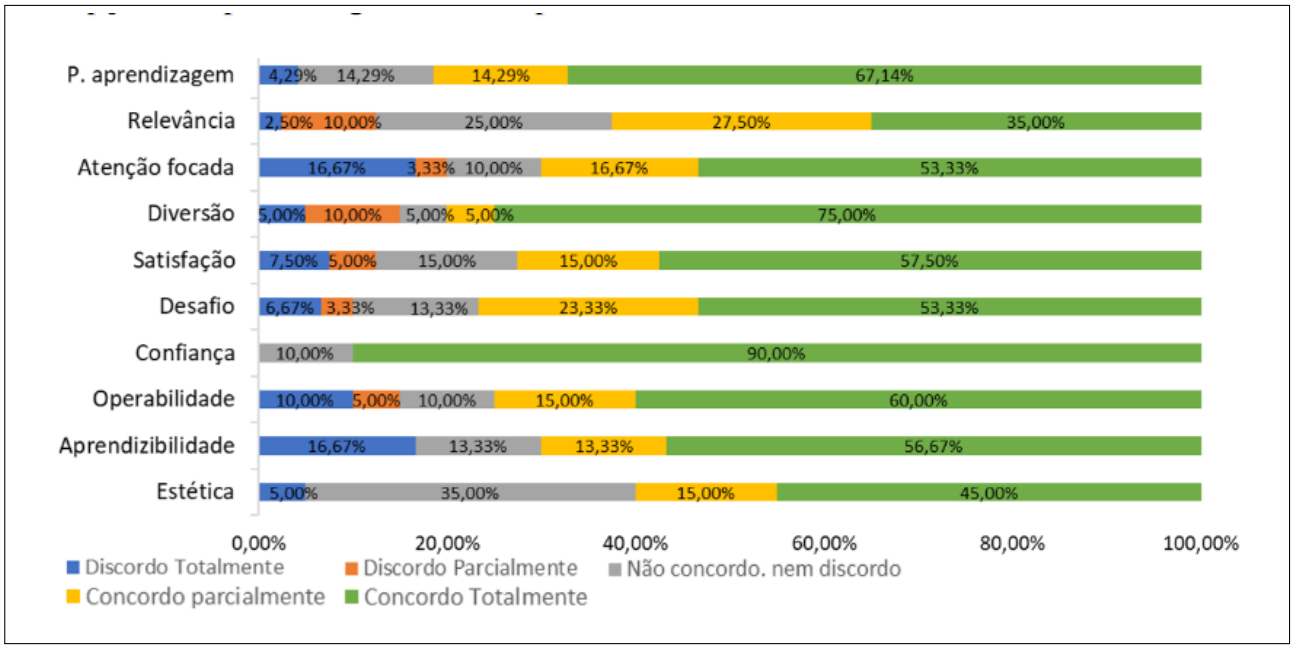

Figura 2.4 – Imagem contendo o feedback dos jogadores.
Fonte: Alencar et al. (2020, p. 4).

Isso demonstra que por meio de um simples jogo incorporando conceitos da computação é possível despertar a confiança em estudantes. Essa confiança, uma vez estabelecida, pode ser usada como grande incentivo de aprendizagem para fortalecimento do raciocínio lógico e da capacidade de resolver problemas, habilidades de alta utilidade na área de programação.

Além disso, há diversos outros jogos mais recentes que também exploram o ensino de programação. Como por exemplo Hello Food Macena et al. (2022) utiliza uma metáfora de cozinha para introduzir conceitos básicos de algoritmos e sequências de instruções. Outro exemplo é o Super Think Wash densevolvido por Dutra et al. (2021), que simula atividades cotidianas, desafiando o jogador a organizar tarefas em ordem lógica. Esses exemplos demonstram que jogos podem ser adaptados a diferentes contextos temáticos, mas sempre com o objetivo de estimular a compreensão de estruturas lógicas e de programação.

Considerando os trabalhos relacionados descritos anteriormente, este trabalho propõe desenvolver um jogo educacional 2D voltado ao ensino de programação básica.

2.2 Desenvolvimento de jogos

Para realizar o desenvolvimento de qualquer jogo é necessário saber qual *engine* utilizar, pois elas facilitam o desenvolvimento, possuindo muitas funcionalidades necessárias para o desenvolvimento de jogos já implementadas, como *rendering*, entrada de dados, áudio, rede, inteligência artifi-

cial entre outros. Assim, foi realizada uma análise comparativa entre algumas das principais *engines* utilizadas no desenvolvimento de jogos como a Unity, Unreal e a Godot Engine, com foco em escolher uma que seja leve, intuitiva e a curva de aprendizado rápida. Cada *engine* apresenta características específicas voltadas a diferentes níveis de complexidade e propósitos de desenvolvimento.

2.2.1 Unity

A Unity Technologies (2024) é uma das engines mais utilizadas na indústria de jogos digitais, sendo amplamente reconhecida por sua versatilidade e compatibilidade com múltiplas plataformas, como *desktop*, dispositivos móveis, *web* e consoles. Possui uma interface intuitiva e uma vasta documentação oficial, além de uma comunidade ativa que disponibiliza tutoriais e recursos em abundância. Essa acessibilidade torna a Unity uma ferramenta poderosa tanto para desenvolvedores independentes quanto para grandes estúdios.

Entretanto, essa amplitude de recursos pode representar um desafio para iniciantes. A curva de aprendizado é consideravelmente acentuada, sobretudo pela necessidade de dominar a linguagem C# Microsoft Corporation (2024), que é a principal forma de programar na *engine*. Além disso, apesar de oferecer suporte para jogos 2D, a Unity é originalmente voltada ao desenvolvimento 3D, o que pode tornar o processo de criação de jogos bidimensionais mais complexo do que o necessário. O uso de componentes e sistemas de física muitas vezes exige ajustes manuais, e o ambiente de desenvolvimento tende a ser mais pesado, o que pode dificultar a execução em máquinas de menor desempenho. Assim, embora seja uma *engine* robusta e consolidada, pode não ser a opção mais prática para projetos de pequeno porte.

2.2.2 Unreal

A Unreal Engine Games (2024) é amplamente reconhecida por seu alto desempenho e pela capacidade de produzir gráficos de qualidade superior. Sendo amplamente utilizada em jogos 3D de grande escala, a Unreal oferece recursos avançados como iluminação realista, renderização de última geração e um sistema de física detalhado. Seu diferencial está no sistema *Blueprints*[1], que permite a criação de lógicas de jogo por meio de blocos visuais, reduzindo a necessidade de codificação direta.

[1] *Blueprints* são um sistema de script visual completo que permite a criação de lógicas de jogabilidade, eventos e interatividade do jogo sem escrever código de programação tradicional.

No entanto, a Unreal também apresenta limitações para projetos de pequeno porte. Por exigir hardware mais potente e um conhecimento mais profundo em C++ Stroustrup (2013) e nos próprios *Blueprints*, a engine acaba se tornando menos acessível a desenvolvedores iniciantes. Além disso, apesar de oferecer suporte a jogos 2D, a Unreal é fortemente orientada ao desenvolvimento 3D, o que implica em uma sobrecarga de recursos e complexidade desnecessária para aplicações simples.

2.2.3 A escolha pela Godot

Diante das características observadas nas engines anteriores, a Godot apresentou-se como a opção mais adequada para o desenvolvimento deste projeto. Por ser uma ferramenta de código aberto e multiplataforma, oferece ampla liberdade de customização e uma curva de aprendizado mais acessível para iniciantes. Sua arquitetura baseada em nós e cenas favorece a organização modular do jogo, tornando o processo de criação mais intuitivo.

Segundo o *site* oficial, "Godot Engine é um motor de jogos multiplataforma repleto de recursos para criar jogos 2D e 3D a partir de uma interface unificada. Ela fornece um conjunto abrangente de ferramentas comuns para que os usuários possam concentrar-se em criar jogos sem precisar 're-inventar a roda'. Os jogos podem ser exportados num clique para várias plataformas, incluindo as principais plataformas de *desktop* (Linux, macOS, Windows), plataformas móveis (Android, iOS), as baseadas na Web e também *consoles*. A Godot tem uma vasta comunidade, por ser gratuita e de código aberto sob a licença MIT, que não é restritiva. A *engine* é totalmente independente e voltada para a comunidade, sendo mantida pela Godot Foundation, uma organização sem fins lucrativos" (Juan Linietsky e the Godot community, 2024). Essas características reforçam os motivos que justificam a escolha da Godot para o presente projeto. A *engine* foi projetada com forte foco em jogos 2D, o que simplifica significativamente o processo de desenvolvimento sem a necessidade de recursos gráficos avançados, sendo leve e não exigindo capacidade elevada de *hardware*. A linguagem GDScript, inspirada em Python, contribui para a compreensão rápida do funcionamento das funcionalidades principais da *engine*, tornando-a ideal para projetos de pequeno porte. Assim, a escolha pela Godot reflete a busca por uma *engine* leve, intuitiva e que tenha uma curva de aprendizado rápida, assim como a análise proposta inicialmente.

2.3 A Godot Engine

Após a análise comparativa entre diferentes *engines* de desenvolvimento de jogos, a Godot foi definida como a mais adequada para este projeto. Nesta seção, são apresentadas as principais interfaces e funcionalidades da Godot Engine.

Ao iniciar no ambiente de desenvolvimento da Godot Engine, após a criação de um novo projeto, a interface é o primeiro elemento a ser apresentado ao usuário, sendo o ponto de partida para a manipulação e criação de cenas e nós. A **Figura 2.5** mostra a interface principal da *engine* Godot.

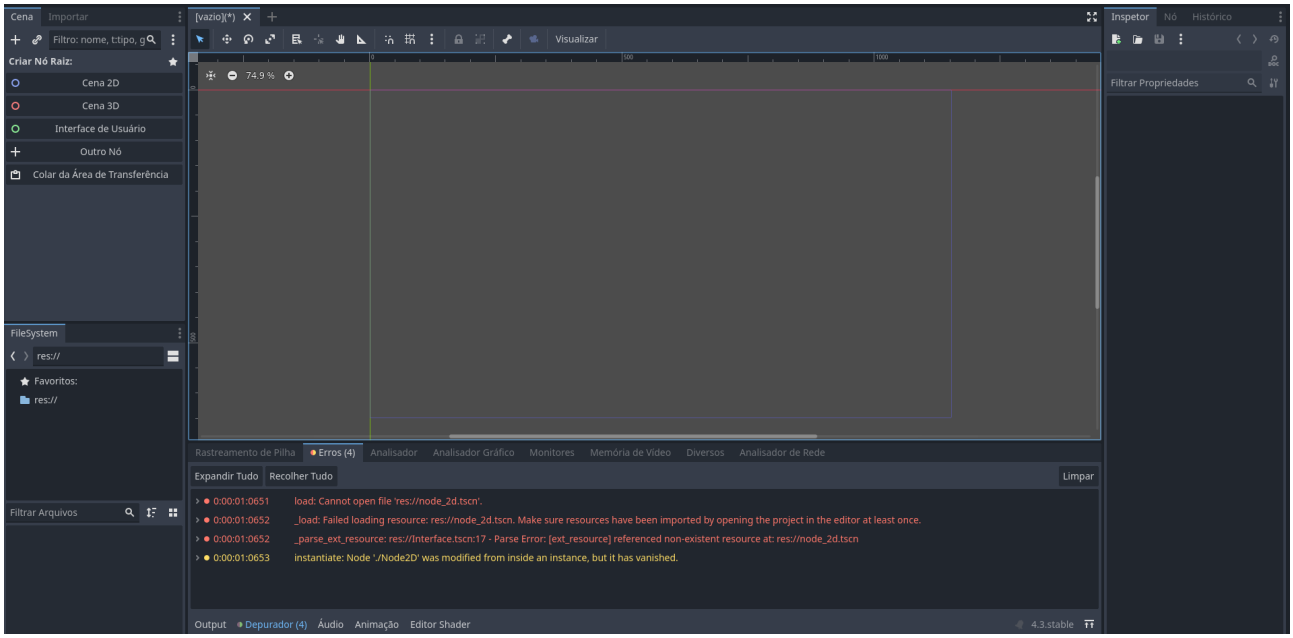

Figura 2.5 – Imagem da Interface de desenvolvimento da Godot Engine.
Fonte: Autor.

A **Figura 2.6** mostra a área de criação de nós, sendo possível a criação de 4 nós principais, como **Cena2D**, **Cena3D**, **Interface do Usuário** e a opção **Outro**, esta última possibilitando a adição de um novo nó, provendo propriedades diferentes dos demais já apresentados. Cada nó na Godot possui um conjunto próprio de propriedades e funções, podendo ser atualizado a cada quadro de execução e estendido com novas características.

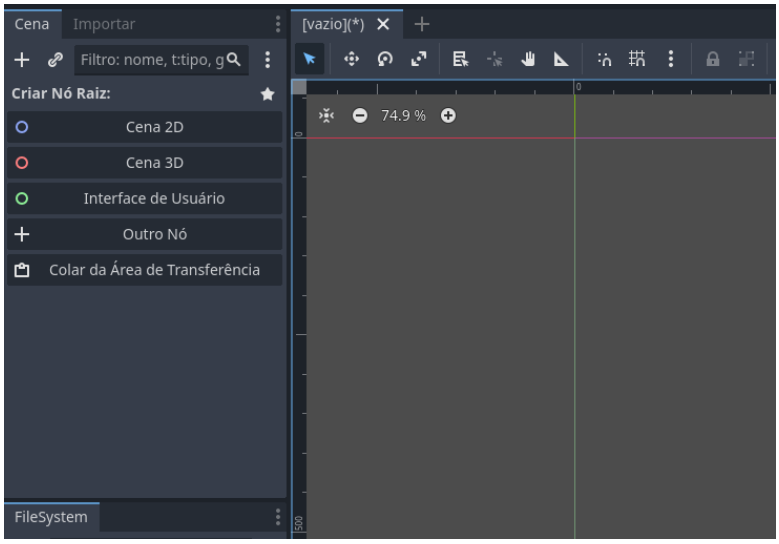

Figura 2.6 – Área de Criação de Nós.
Fonte: Autor.

Cada nó é organizado em uma estrutura de árvore hierárquica, chamada "árvore de cenas", cada cena pode possuir múltiplos nós que herdam propriedades e permitem a reutilização de componentes, assim garantindo a modularidade no *design* do jogo. Para criar uma cena é necessário criar um nó raiz e então salvar a cena em algum lugar do dispositivo.

Na **Figura 2.7**, na parte inferior esquerda, é possível ver quantas cenas foram criadas de acordo com o lugar onde o desenvolvedor salvou a cena. Garantindo o controle de cenas criadas e sua visualização, assim como um atalho para abertura da cena.

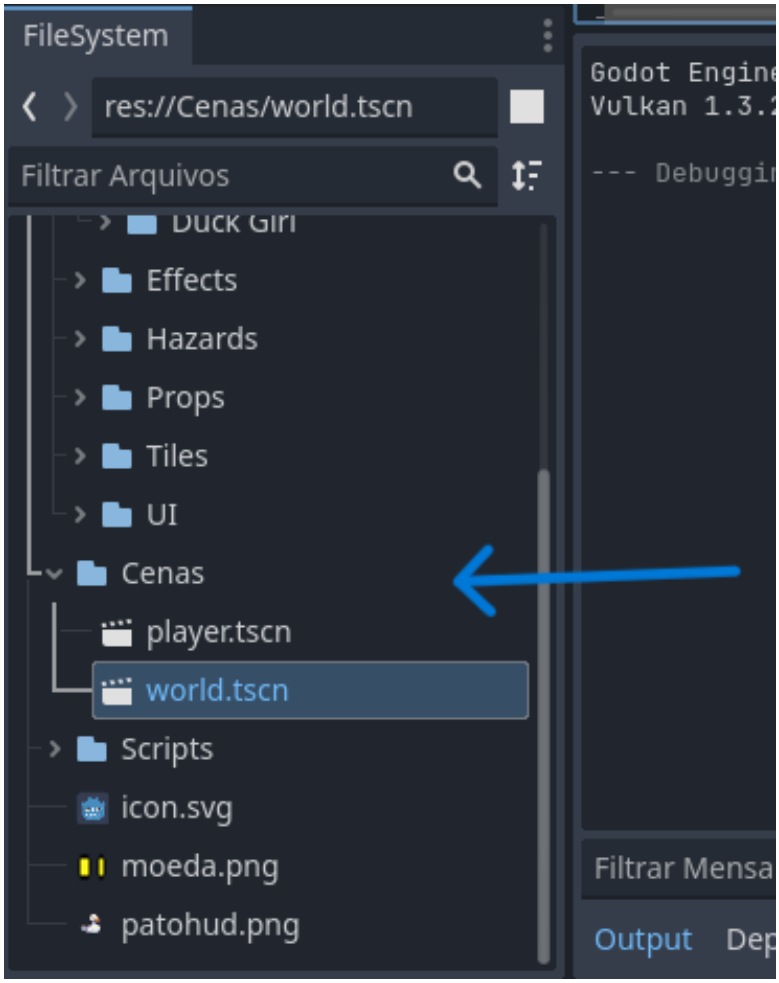

Figura 2.7 – Local das cenas salvas.
Fonte: Autor.

Na **Figura 2.8** é mostrada a área do "Inspetor", localizada na parte superior direita, é possível a visualização de características de cada nó selecionado na árvore hierárquica. Tais como modificar a coordenada da posição do nó, sua visibilidade dentro do jogo, aplicar filtros de textura, etc.

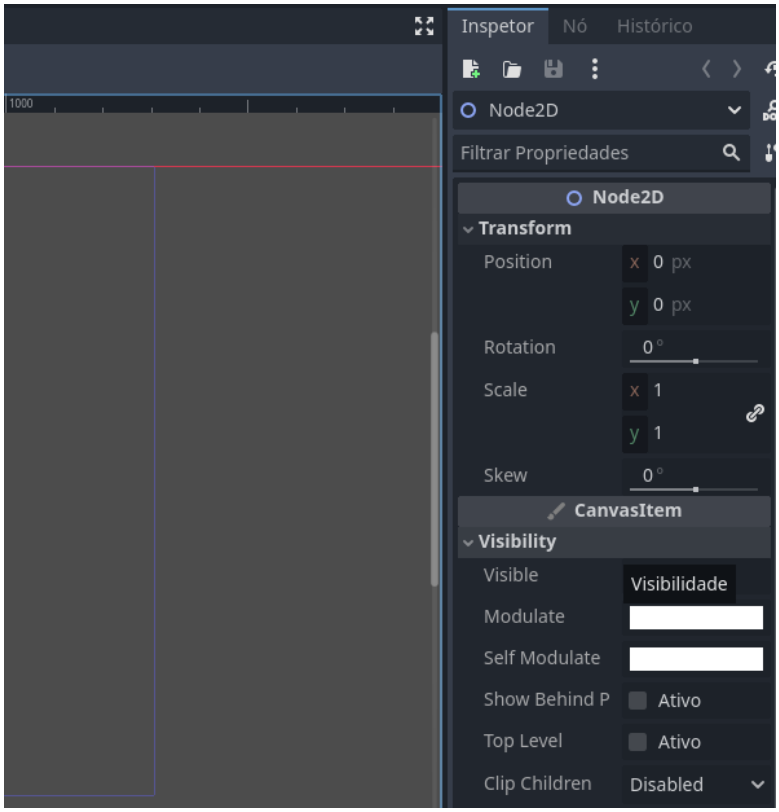

Figura 2.8 – Área do Inspetor.
Fonte: Autor.

A área do inspetor nos permite modificar diferentes valores dos nós selecionados, garantindo uma interface intuitiva e amigável ao uso.

No decorrer do desenvolvimento utilizando a Godot Engine, o editor de *scripts*, localizado ao selecionar um nó e adicionar um novo *script* através dele, oferece suporte para GDScript, semelhante a sintaxe da linguagem Python (Python Software Foundation, 2024). O editor de *script* é importante para modificar elementos dentro do jogo e adicionar funcionalidades que serão necessárias para que se construa a estrutura básica do jogo. A **Figura 2.9** mostra o editor de *scripts* presente na Godot.

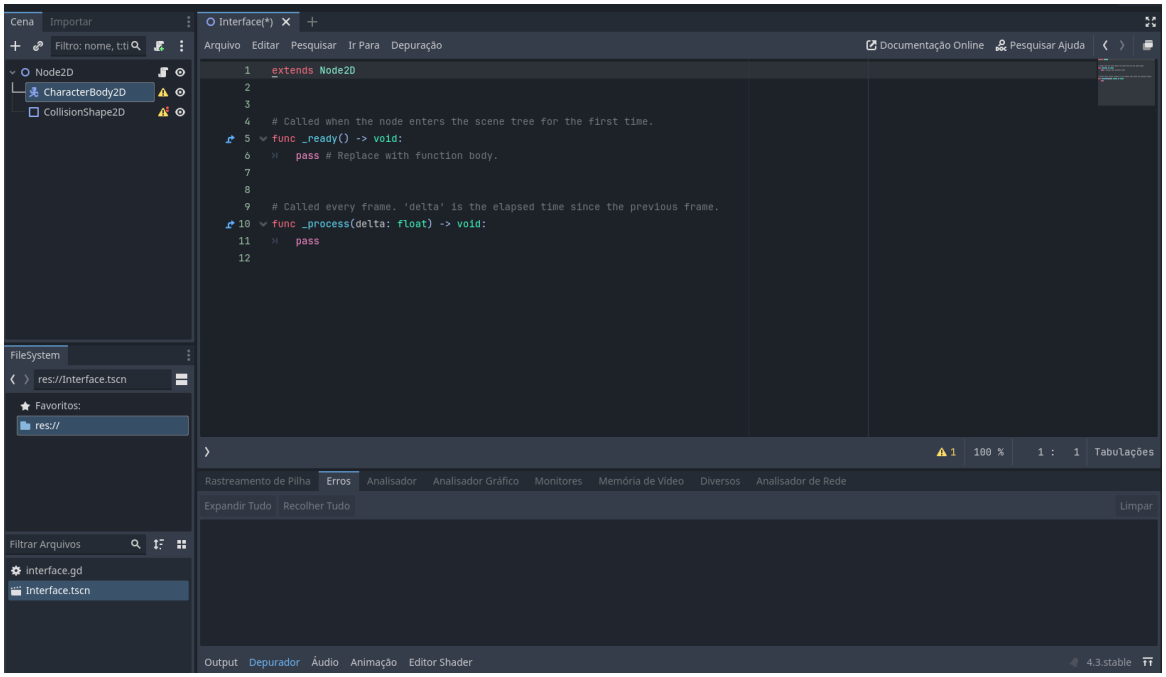

Figura 2.9 – Editor de Script.
Fonte: Autor.

Através da inserção de *scripts* em nós específicos, é possível definir e ajustar as ações e reações de personagens ou elementos do jogo, contribuindo diretamente para a dinâmica e a experiência do jogo.

Dentro desse contexto, a *engine* Godot utiliza um sistema chamado sinais, que funciona como um mecanismo de comunicação entre nós. Quando um evento ocorre, como por exemplo o pressionamento de um botão, a colisão de um objeto ou a conclusão de uma animação, o nó responsável emite um sinal. Outros nós podem então se conectar a esse sinal e executar funções em resposta ao evento. Esse modelo permite que diferentes partes do jogo interajam entre si sem dependerem de referências diretas, promovendo um *design* mais modular, organizado e de fácil manutenção.

Na **Figura 2.10** mostra o acesso a área para visualização desses sinais de cada nó, ao lado do botão inspetor há o botão nó, nele contém todos os sinais referentes ao nó que foi selecionado, assim como características próprias e valores de retorno.

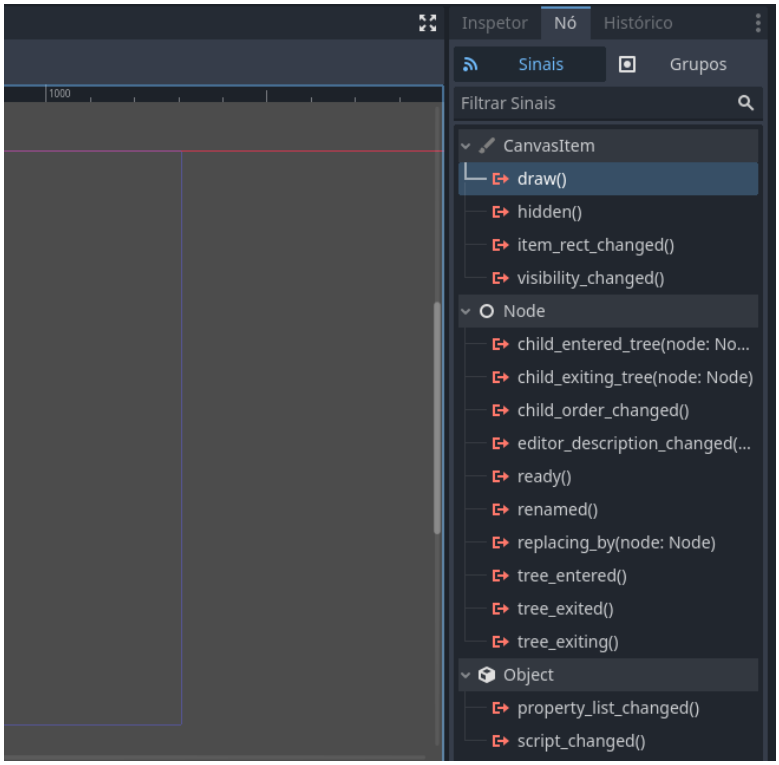

Figura 2.10 – Área de visualização de sinais referente a um nó selecionado.
Fonte: Autor.

Na **Figura 2.11** há o botão *play* para executar o projeto como um todo na parte superior direita, assim sendo todas as cenas são executadas em conjunto, podendo ser visualizado a evolução do desenvolvimento do jogo.

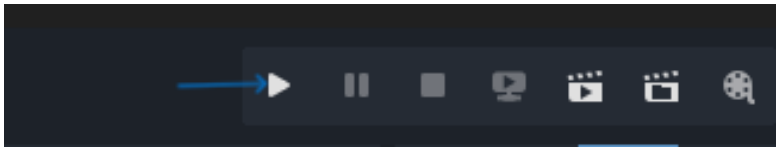

Figura 2.11 – Botão de execução de cena.
Fonte: Autor.

A partir da revisão apresentada, foi possível compreender os principais conceitos, ferramentas e tecnologias que fundamentam o desenvolvimento de jogos educacionais voltados ao ensino de programação, bem como identificar a escolha da *engine* mais adequada com base no estudo realizado. Assim, com a *engine* definida e suas principais características descritas, o capítulo seguinte apresenta a metodologia adotada para o desenvolvimento do projeto, detalhando as etapas, ferramentas e procedimentos utilizados na construção do jogo proposto.

3. Desenvolvimento do Jogo

O desenvolvimento do jogo educacional foi estruturado de forma a unir aspectos técnicos de implementação com objetivos pedagógicos voltados ao ensino de programação básica. Para isso, adotou-se o uso de um Documento de Design do Jogo (Game Design Document – GDD), que serviu como guia central durante todo o processo de desenvolvimento, conforme proposto por Schuytema (2008).

O GDD organiza as principais informações relacionadas ao projeto, como mecânicas de jogo, blocos de código, fases, níveis de dificuldade e interfaces gráficas. Essa documentação foi fundamental para garantir consistência e clareza durante a implementação técnica do jogo. O conteúdo completo do GDD encontra-se no **Apêndice A**.

3.1 Etapas de desenvolvimento

O processo de desenvolvimento seguiu três etapas principais:

- Protótipo inicial: implementação de um modelo simples de arrastar e soltar blocos de código para validar a mecânica;
- Desenvolvimento das funcionalidades principais: criação de blocos representando estruturas de programação;
- Construção de fases e testes: elaboração das quatro fases principais, com níveis de dificuldade crescentes, além de testes de jogabilidade, desempenho e clareza pedagógica.

3.2 Ferramentas e Tecnologias

O jogo foi desenvolvido utilizando a Godot Engine, escolhida por ser uma ferramenta gratuita, de código aberto e multiplataforma, que oferece suporte nativo para o desenvolvimento de jogos 2D. A linguagem utilizada foi o GDScript, nativa da Godot, com sintaxe semelhante à do Python, o que facilitou a implementação da lógica do jogo e das interações com os blocos de código.

Para a criação dos elementos visuais, foram utilizadas as ferramentas Piskel Piskel (2025) e LibreSprite LibreSprite (2025), ambas voltadas para produção de arte em pixel art[1], alguns *assets*[2] de terceiros foram utilizados no projeto, retirados do *site Itch.io*, o **Apêndice B** demonstra os nomes dos pacotes que foram utilizados e a licença de uso.

- Piskel foi utilizado para o *design* dos personagens, inimigos e objetos do cenário, garantindo um estilo visual coerente e leve;
- LibreSprite foi utilizado para a criação e animação dos *sprites*[3], permitindo suavizar os movimentos e transições entre ações.

Além disso, o Github Github (2025) foi utilizado como repositório para controle de versão e disponibilização do código fonte, permitindo a colaboração futura do projeto .

3.3 Estruturação do Jogo

O jogo é estruturado em fases progressivas, cada uma apresentando um novo conceito de programação básica. O objetivo é que o jogador aprenda, de forma prática e lúdica, noções fundamentais como sequência de comandos, condicionais, repetições e funções. O jogador interage com blocos de código em estilo *drag and drop* (arrastar e soltar), organizando-os em uma área de montagem para resolver os desafios propostos. A correta combinação dos blocos possibilita a execução de algoritmos que fazem o personagem avançar no jogo.

[1] Pixel art é uma forma de arte digital em que as imagens são criadas e editadas em nível de pixel, caracterizando-se pelo uso de baixa resolução e pela estética visual de gráficos de jogos clássicos.

[2] *Assets* são os recursos gráficos, sonoros e visuais utilizados no desenvolvimento de um jogo, como imagens, sons, animações, modelos e texturas, que compõem os elementos visuais e auditivos da aplicação

[3] *Sprites* são imagens bidimensionais utilizadas para representar personagens, objetos ou elementos visuais que se movem de forma independente no ambiente do jogo, sendo amplamente empregados no desenvolvimento de jogos 2D.

O funcionamento do jogo baseia-se em quatro tipos de blocos:

- Blocos de movimento: permitem o deslocamento do personagem em quatro direções dentro do percurso disponível no jogo;
- Blocos de condicionais: realizam verificações lógicas baseadas em condições específicas;
- Blocos de repetição: executam uma sequência de ações repetidamente enquanto uma condição é verdadeira;
- Blocos de função: agrupam comandos reutilizáveis para modularidade do código.

3.4 Desenvolvimento Técnico do Jogo

O desenvolvimento do jogo foi realizado na Godot Engine, utilizando sua estrutura hierárquica de nós, que permite organizar cada elemento do jogo como parte de uma árvore de cenas. Essa estrutura foi fundamental para separar a lógica de funcionamento dos blocos de código, a interface do usuário e o comportamento do personagem. O *design* dos blocos de código pode ser consultado no **Apêndice A.8**.

Cada bloco de código do jogo foi desenvolvido como uma cena independente, contendo os seguintes componentes principais:

- Node raiz: do tipo **Control**, utilizado como base da interface do bloco, permitindo que ele seja arrastável;
- TextureRect: responsável pela imagem do bloco, mantendo a identidade visual de cada tipo de instrução (movimento, condição, repetição, função);
- Script (GDScript): vinculado ao nó raiz, responsável por definir o comportamento do bloco, como o evento de encaixe, soltura e execução da ação;
- Signal (sinal): utilizado para comunicar eventos entre blocos e entre a área de montagem e o personagem principal, garantindo independência entre as partes do código.

A **Figura 3.1** apresenta visualmente a área de montagem e a interação dos blocos. A mecânica de *drag and drop* (arrastar e soltar) é a fundação da interface de programação por blocos e requer a coordenação entre o *script* do bloco e o *script* da área de montagem.

Figura 3.1 – Interação dos blocos com a área de montagem.
Fonte: Autor.

O processo de arrasto é iniciado no bloco. Se o jogador estiver arrastando um bloco novo da paleta de comandos, o sistema primeiramente cria um clone exato do bloco, garantindo que o original permaneça disponível e que o clone herde todas as propriedades necessárias (tipo, textura, contador de repetição, entre outros).

Em seguida, o controle passa para a área de montagem. Esta área atua como um validador, verificando continuamente se o bloco pode ser solto no local onde o *mouse* se encontra. A validação assegura duas condições essenciais:

- Se o objeto arrastado é, de fato, um bloco de programação válido;
- O cumprimento de regras de encaixe específicas, como a restrição de soltar blocos de função dentro da própria área de montagem da função.

Se a validação for positiva, o sistema indica visualmente ao jogador que o bloco pode ser solto. A rotina final gerencia o encaixe, sendo responsável por adicionar o bloco à estrutura do algoritmo, remover o bloco de sua posição anterior (se aplicável) e tratar a troca de posição de outros comandos na área de montagem.

3.4.1 Interpretação e Execução do Algoritmo

A execução do algoritmo visual construído pelo jogador é o ponto crucial do sistema. Diferentemente da montagem, que é reativa (baseada em eventos do *mouse*), a execução é procedural e controlada pelo interpretador do jogo.

O processo é disparado por uma função de *callback* que serve como o ponto de entrada da execução. A primeira etapa é gerar uma estrutura imutável que copia a hierarquia de blocos montada pelo jogador para uma lista interna do Godot. Essa imutabilidade é vital para garantir que o algoritmo não seja alterado caso o jogador tente interagir com os blocos enquanto a execução está em andamento.

A lógica de execução é então iniciada por uma rotina de interpretação, que atua como o coração do interpretador. Ela percorre sequencialmente a estrutura de blocos e processa os comandos de movimento. Estruturas como repetição (*Loops*) e funções são tratadas através da implementação de recursividade lógica: ao encontrar o bloco de repetição, a rotina de execução é utilizada novamente para processar o subconjunto de blocos internos pelo número de vezes definido, e o mesmo princípio é aplicado para blocos de função.

Ao final de cada movimento, o sistema executa verificações cruciais, como a colisão com o inimigo e a validação de que o personagem alcançou um *checkpoint* válido. Se qualquer validação de colisão ou movimento inválido falhar, o processo de execução é interrompido imediatamente, e o personagem é reposicionado no último *checkpoint* alcançado. Essa arquitetura recursiva e modular assegura que o sistema possa interpretar de forma robusta qualquer combinação válida de sequência de movimentos, repetição e função.

3.4.2 Desenvolvimento das fases do jogo

Antes de qualquer fase ser desbloqueada, o jogador deve concluir a fase tutorial, que apresenta a mecânica básica do jogo, como o sistema *drag and drop* (arrastar e soltar) e as principais áreas da interface. Cada fase subsequente foi planejada para introduzir gradualmente um novo conceito de programação, de forma prática e contextualizada. Assim, o aprendizado ocorre à medida que o jogador interage com os blocos e compreende seu funcionamento dentro do ambiente de jogo.

Dessa forma, o jogo foi estruturado em quatro fases principais, cada uma projetada para introduzir gradualmente um novo conceito de programação e consolidar o aprendizado por meio da prática. As fases desenvolvidas são:

1. Fase movimento: aborda o deslocamento do personagem;
2. Fase condicional: apresenta o uso de verificações lógicas;
3. Fase repetição: trabalha a execução de instruções em laço;
4. Fase função: introduz a modularização e reutilização de código.

Na sequência, cada uma dessas fases é descrita em detalhes, com destaque para suas mecânicas e interfaces correspondentes.

3.4.2.1 Fase movimento

Nesta fase inicial, o jogador aprende a utilizar os blocos de movimento, que representam as direções possíveis do personagem (cima, baixo, esquerda e direita).

Na **Figura 3.2** é possível visualizar os blocos de movimento soltos na área de montagem da fase, assim sendo cada bloco corresponde a um *tile* no percurso do personagem, ou seja, a cada bloco executado o personagem se desloca um espaço de acordo com a direção do bloco escolhido. A área de montagem permite até 13 blocos por sequência, incentivando o jogador a pensar na ordem correta das instruções.

Figura 3.2 – Visualização dos blocos de movimento na área de montagem.
Fonte: Autor.

3.4.2.2 Fase condicional

A segunda fase introduz o bloco de condicional, responsável por avaliar situações lógicas. O jogador precisa usar o bloco condicional para verificar se existe um inimigo à frente e, caso positivo, eliminá-lo.

Na **Figura 3.3** é possível visualizar o inimigo fixo no percurso, se o jogador não incluir a verificação fornecida pelo bloco condicional, ocorrerá uma colisão e o personagem será reiniciado a partir do último *checkpoint*.

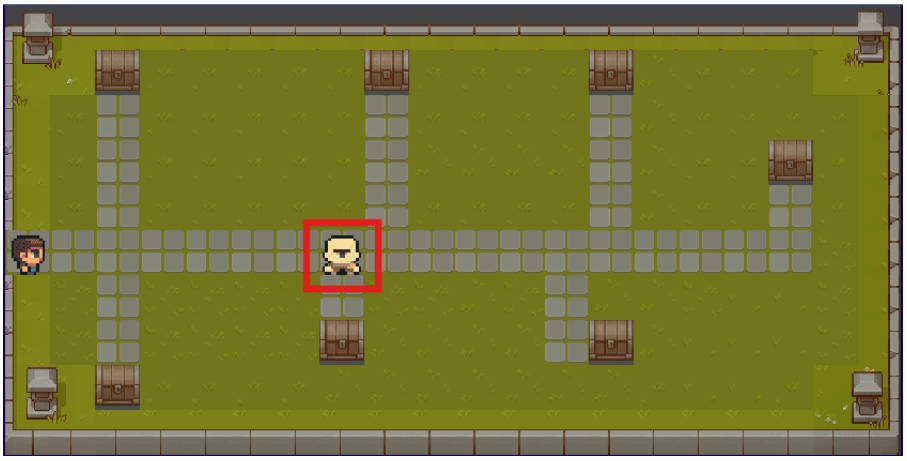

Figura 3.3 – Inimigo no percurso do personagem.
Fonte: Autor.

3.4.2.3 Fase repetição

A terceira fase apresenta o bloco de repetição, que permite executar uma sequência de ações múltiplas vezes.

Na **Figura 3.4** é possível visualizar o uso do bloco de repetição para mover o personagem dentro da área de montagem, esse bloco é formado por dois elementos: `repita_início` e `repita_fim`. Todos os blocos inseridos entre esses delimitadores são executados conforme o valor definido no campo do bloco `repita_início`, que varia de 1 a 10. Caso o jogador esqueça de fechar o bloco com `repita_fim`, as instruções seguintes não serão executadas corretamente.

Figura 3.4 – Representação do bloco de repetição na área de montagem.
Fonte: Autor.

3.4.2.4 Fase função

Na última fase, é introduzido o bloco de função, que reforça o conceito de modularidade e reutilização de código.

Na **Figura 3.5** é apresentada uma nova área de montagem, localizada na parte inferior da área de montagem principal, onde o jogador pode construir uma função personalizada composta por outros blocos. Essa função só será executada quando o jogador incluir o bloco de chamada da função na área de montagem principal.

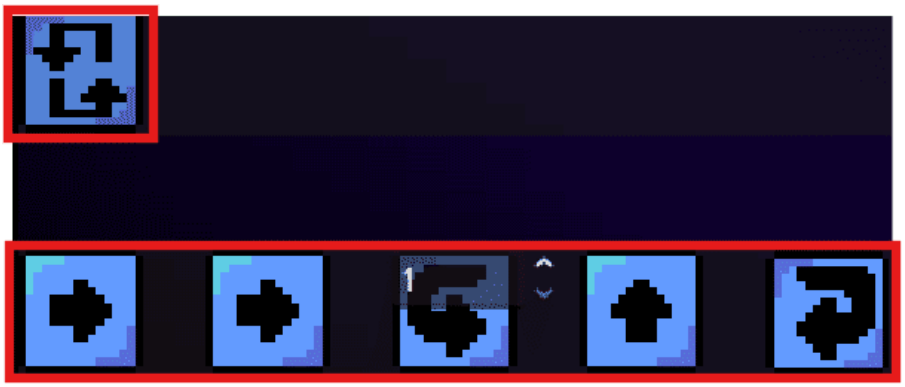

Figura 3.5 – Representação do bloco de função na área de montagem.
Fonte: Autor.

Cada fase, portanto, combina uma nova mecânica de jogo com um novo conceito de programação, promovendo o aprendizado de forma prática e contextualizada. Ao progredir, o jogador desenvolve pilares fundamentais da programação como lógica, abstração e resolução de problemas.

3.5 Desenvolvimento Completo

A versão final do jogo contempla mecânicas estáveis de arrastar e soltar, fases que introduzem gradualmente os conceitos de programação, e interfaces simples para facilitar a interação do jogador. O **Apêndice A** apresenta em detalhes os elementos que compõem o jogo, como os blocos de código, níveis e protótipos de interface.

Dessa forma, o desenvolvimento do jogo foi concluído. A estrutura final alcançada reflete a consolidação dos objetivos definidos no início do projeto, garantindo uma experiência prática voltada ao aprendizado de conceitos básicos de programação. A seguir, no **Capítulo 4** são apresentados os resultados obtidos com o desenvolvimento do jogo, bem como as possibilidades de aprimoramento futuro do trabalho.

4. Conclusão

O desenvolvimento deste jogo educacional teve como foco unir conhecimentos de lógica de programação e *design* de jogos. Nesse sentido, o projeto apresenta uma ferramenta que introduz conceitos básicos de programação por meio da *engine* Godot. O jogo desenvolvido é disponibilizado como software de código aberto, permitindo que outros pesquisadores e desenvolvedores possam dar continuidade ao projeto e realizar aprimoramentos. O código-fonte está acessível publicamente em: `https://github.com/MusgoNato/TCC`.

4.1 Resultados Obtidos

Esta seção apresenta os resultados obtidos a partir do desenvolvimento do jogo educacional proposto. O principal objetivo foi criar uma ferramenta que auxiliasse a compreender conceitos básicos de programação, utilizando blocos de código que representam instruções básicas.

O produto final consiste em um jogo funcional, construído com base em elementos que simulam estruturas fundamentais da lógica de programação, como condições, repetições e funções, possibilitando que o jogador aprenda conceitos práticos ao longo das fases. A seguir, são destacados os principais resultados alcançados:

- Conforme fora planejado, o jogo foi desenvolvido para ser utilizado como ferramenta complementar no processo de ensino-aprendizagem de programação;
- A apresentação de uma ferramenta que permite ilustrar de forma sistematizada estruturas básicas da programação, como condições, repetições e funções.

Durante o processo de desenvolvimento, surgiram alguns desafios, especialmente na implementação da estrutura condicional completa (*if* e *else*). Portanto, optou-se por uma versão simplificada do bloco condicional, representando apenas o *if*.

Outro desafio foi a gestão de tempo, já que o desenvolvimento ocorreu paralelamente a outras atividades. Mesmo com essas limitações, foi possível entregar uma versão funcional e estável do jogo.

Assim, considera-se que o objetivo proposto foi alcançado, uma vez que o jogo cumpre sua função de introduzir conceitos básicos de programação através de blocos de código. Espera-se que este projeto sirva como base para futuras melhorias e inspire novas iniciativas voltadas ao uso de jogos como ferramenta de apoio ao ensino.

Como perspectiva de trabalhos futuros, propõe-se:

- Implementar a estrutura condicional completa (*if* e *else*);
- Criar novas fases com conceitos avançados de lógica e algoritmos;
- Adicionar elementos sonoros para as interfaces.

Referências bibliográficas

Abelson, H. e DiSessa, A. (1986). *Turtle geometry: The computer as a medium for exploring mathematics*. MIT press.

Alencar, L., Pires, F. e Pessoa, M. (2020). Criação de um jogo para desenvolver o pensamento computacional percorrendo caminhos eulerianos. Em *Anais do XXVIII Workshop sobre Educação em Computação*, páginas 111–115. SBC.

Brennan, K. e Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. Em *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, volume 1, página 25.

de Oliveira, G. A., de Bettio, R. W., Rodarte, A. P. e Ferrari, F. B. (2014). Grubibots educacional: jogo para o ensino de algoritmos na educação básica. Em *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 25, página 584.

Dutra, T. C., Felipe, D., Gasparini, I. e Maschio, E. (2021). Super thinkwash: Um jogo digital educacional inspirado na vida real para desenvolvimento do pensamento computacional em crianças. Em *Anais do XXXII Simpósio Brasileiro de Informática na Educação*, páginas 292–303. SBC.

Games, E. (2024). Unreal engine documentation. Acesso em: 13 nov. 2025.

Github (2025). Tcc. `https://github.com/MusgoNato/TCC`. Acesso em: 11 nov. 2025.

Juan Linietsky, A. M. e the Godot community (2024). Godot docs - master branch. Acessado em: 22 de agosto de 2024.

LibreSprite (2025). Libresprite - animated sprite editor & pixel art tool. Acesso em: 12 nov. 2025.

Macena, J., Pires, F. e Melo, R. (2022). Hello food: uma jornada de aprendizagem lúdica em algoritmos, programação e pensamento computacional. Em *Anais do XXXIII Simpósio Brasileiro de Informática na Educação*, páginas 561–572. SBC.

Microsoft Corporation (2024). C# programming guide. Acesso em: 12 nov. 2025.

MIT Media Lab (2025). Scratch – imagine, program, share. Acesso em: 13 nov. 2025.

Papert, S. A. (2020). *Mindstorms: Children, computers, and powerful ideas*. Basic books.

Piskel (2025). Piskel - free online sprite editor. Acesso em: 12 nov. 2025.

Python Software Foundation (2024). *Python Documentation*. Acessado em: 10 nov. 2025.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. et al. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11):60–67.

Savi, R. e Ulbricht, V. R. (2008). Jogos digitais educacionais: benefícios e desafios. *Revista Novas Tecnologias na Educação*, 6(1).

Schuytema, P. (2008). *Design de Games: Uma abordagem prática*. Cengage Learning.

Stroustrup, B. (2013). *The C++ Programming Language*. Addison-Wesley, Boston, 4 edition.

Technologies, U. (2024). Unity documentation. Acesso em: 13 nov. 2025.

WING, J. M. (2006). Computational thinking. *Communications of the ACM, vol. 49, nž. 3, March 2006, pp. 33-35*.

A. Document Design

O Modelo de Document de Design (GDD), apresenta a ideia do jogo, trazendo as características que serão encontradas ao jogá-lo e suas funcionalidades principais, o modelo deste GDD seguiu por partes o modelo apresentado por Paul Schuytema Schuytema (2008), em seu livro "Design de Games: Uma abordagem Prática", porém em um formato mais simples. O documento será alimentado a medida que o jogo é desenvolvido, mecânicas aprimoradas ou implementações de novas funcionalidades serão todas escritas no documento.

Este apêndice está dividido em subseções. A **subseção A.1** introduz a proposta geral do jogo. A **subseção A.2** define o propósito formativo do projeto e a **subseção A.3** situa a narrativa e a ambientação. Em seguida, a **subseção A.4** descreve a mecânica central utilizada. Na sequência a **subseção A.5** apresenta os blocos de código utilizados no jogo. A **subseção A.6** apresenta a progressão do jogo e a **subseção A.7** exibe os modelos visuais elaborados para cada interface. Por fim, a **subseção A.8** apresenta a estrutura visual dos blocos de código utilizados no jogo e a **subseção A.9** descreve as formas de interação do jogador com o sistema.

A.1 High Concept

Um jogo educacional em 2D cujo objetivo é ensinar conceitos básicos de programação. O jogador utiliza blocos de código arrastáveis para montar algoritmos que solucionam desafios de lógica, avançando em fases progressivas.

A.2 Objetivo Educacional

O objetivo é introduzir estruturas fundamentais da programação de forma interativa, visual e prática. Entre os conceitos abordados estão:

1. Movimento;
2. Condicionais;
3. Repetições;
4. Funções.

A.3 Contexto do Jogo

A narrativa apresenta um personagem controlado pelo jogador que deve resolver desafios lógicos para avançar em diferentes fases. Cada desafio é superado por meio da montagem correta de algoritmos com blocos de código.

A.4 Mecânica Central

O jogador resolve desafios com blocos de códigos que representam uma instrução na programação, como repetições, condições, funções, etc.

O método *drag and drop* (arrastar e soltar) permite o jogador organizar os blocos de acordo com seu raciocínio enquanto joga.

A.5 Blocos de código

Os blocos de código que são utilizados baseiam-se nas instruções apresentadas em diversas linguagens de programação:

1. Movimento: controla o movimento que o personagem será capaz de realizar, como andar para frente, retroceder, ir a esquerda ou direita;
2. Condicionais: verificam se uma condição é falsa ou verdadeira para realizar uma determinada ação;
3. Repetição (*Loops*): executa ações repetidas enquanto uma condição verdadeira;
4. Funções: agrupa blocos para que estes sejam reutilizados, garantindo uma modularidade dentro do jogo.

A.6 Níveis

Cada nível representará um novo desafio dentro do jogo, trazendo conceitos novos da programação. A dificuldade aumentará análoga aos novos conceitos de programação introduzidos em cada nível. O jogador será avaliado ao final de completar um nível por meio de uma pontuação. A seguir é mostrada a sequência de conceitos a serem apresentados durante o jogo em cada nível:

1. Movimento;
2. Condicionais;
3. Repetições;
4. Funções.

A.7 Protótipo de Interfaces

A interface do jogo foi planejada para ser simples e com elementos visualmente agradáveis, para dar uma ambientação de jogo, porém que não fuja do foco educacional. Devido a isso neste documento é apresentado modelos das interfaces do jogo desenvolvido.

A.7.1 Interface Inicial

A interface inicial é a que reflete o jogo como um todo, a partir dela se tira premissas sobre o jogo em questão. Por esse motivo, optou-se por uma interface simples, com poucos elementos chamativos (ver **Figura A.1**).

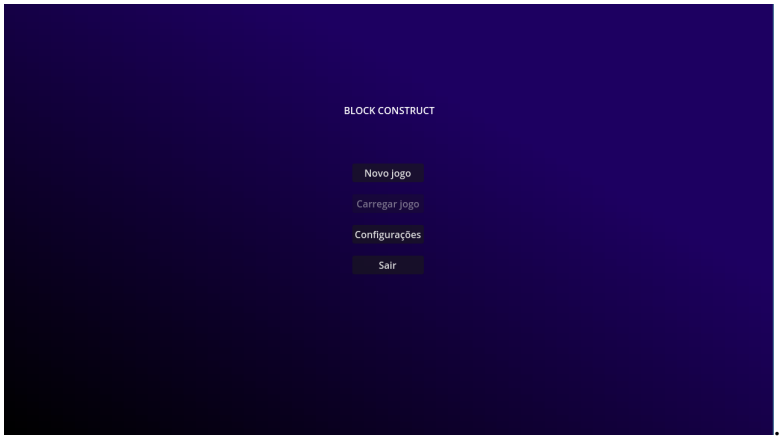

Figura A.1 – Interface inicial do Jogo.
Fonte: Autor.

A interface inicial é simples e direta, oferecendo uma ideia clara do jogo. Ela inclui funcionalidades e um botão para redirecionar para a interface de configurações do jogo, as funcionalidades da interface inicial são apresentadas a seguir:

- Novo jogo: funcionalidade para criar um novo arquivo de informações para o jogador e salvar essas informações em disco;
- Carregar jogo: funcionalidade para carregar um arquivo de salvamento existente, criado anteriormente com a opção "Novo jogo";
- Sair: funcionalidade para sair do jogo.

A.7.2 Interface da visualização dos níveis

Na interface de visualização dos níveis, foi projetado para que cada nível seja um mundo, cada mundo representa um novo conceito que será abordado. A visualização da interface pode ser dada a seguir na **Figura A.2**:

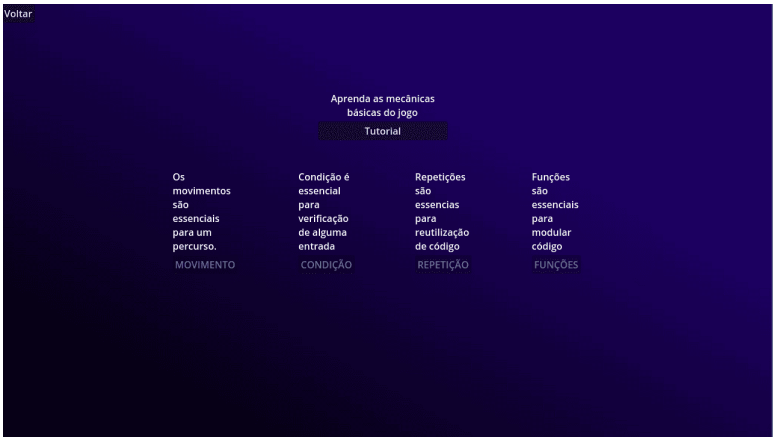

Figura A.2 – Interface da visualização das fases do jogo.
Fonte: Autor.

Caso o usuário prossiga na conclusão da fase, é liberado uma outra e assim sucessivamente. Cada botão como "Movimento", "Condicional", "Repetição"e "Função"redireciona para a interface da fase selecionada, acima destes botões existe uma breve explicação de qual será o conceito apresentado na fase.

A.7.3 Interface das fases

Cada fase trará um conceito novo. A **Figura A.3** apresenta uma fase selecionada.

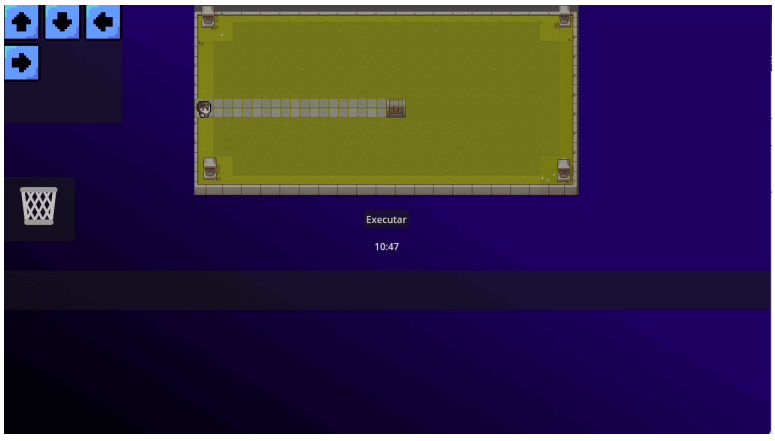

Figura A.3 – Interface da fase selecionada.
Fonte: Autor.

As fases contêm as seguintes estruturas:

1. Área dos blocos disponíveis: aqui estarão os blocos que conterão cada estrutura como condições, repetições e funções;

2. Área de exibição do algoritmo: responsável por exibir a execução do algoritmo construído por meio dos blocos na Área de encaixe dos blocos;

3. Área de encaixe dos blocos: responsável pela mecânica de encaixe dos blocos ao soltá-los, nesta seção é possível saber quais blocos encaixam com outros ao posicionar sobre a área de encaixe;

4. Área informativa: nesta seção são fornecidos alguns detalhes, dicas e informações complementares sobre a fase atual ou uso de algum bloco em específico.

A.8 Estrutura visual dos blocos arrastáveis

Nas subseções seguintes são exibidas imagens da estrutura dos blocos de código de cada fase dentro do jogo.

A.8.1 Bloco movimento

Figura A.4 – Visual dos blocos de movimento.
Fonte: Autor.

A.8.2 Bloco condicional

Figura A.5 – Visual do bloco condicional.
Fonte: Autor.

A.8.3 Bloco repetição

Figura A.6 – Visual do bloco repetição.
Fonte: Autor.

A.8.4 Bloco função

Figura A.7 – Visual do bloco função.
Fonte: Autor.

A.9 Controles

A interação do jogador é intuitiva e simplificada, sendo totalmente controlada pelo mouse. O foco da experiência está na construção lógica de algoritmos, utilizando blocos de encaixe. A visualização da execução dos algoritmos é processada e exibida automaticamente pelo sistema. A função do jogador é unicamente a de montar a sequência de blocos e, em seguida, iniciar a execução com o botão "Executar".

B. Documentação e comprovação de *assets*

Este apêndice tem por finalidade documentar a origem e as permissões de uso dos *assets* visuais. Todos os recursos foram obtidos na plataforma *Itch.io* sob licenças que permitem o uso comercial e, em alguns casos, dispensam a obrigatoriedade de atribuição de crédito.

A **Tabela B.1** apresenta de forma organizada os *assets*, listando sua categoria, o nome do pacote, o autor e as informações referentes à licença, permitindo uma visão geral clara sobre a origem e condições de uso de cada recurso.

Categoria do Asset	**Nome do Pacote (ou Item)**	**Autor (Nickname)**	**Licença e Restrições**
Personagens	Top Down character pack	TotusLotus	Uso livre, comercial permitido, sem atribuição.
Tileset (Mapa)	Pixel Art Top Down - Basic	Cainos	Uso comercial permitido, sem atribuição.
Setas direcionais do bloco movimento	Pixel Keys	It's Dev Time	LICENÇA NÃO DECLARADA. Uso assumido como educacional/não comercial.

Tabela B.1 – Lista de *assets* de Terceiros e Suas Licenças.

O *asset* denominado **Pixel Keys** do autor **It's Dev Time** foi obtido gratuitamente na plataforma *Itch.io*. Contudo, na ausência de uma declaração de licença no próprio site como os outros *assets* utilizados que autorize explicitamente o uso comercial, este trabalho adota o princípio legal de que os direitos autorais plenos permanecem com o criador. Como não há, neste caso, uma licença especificada que autorize o uso comercial ou educacional, considera-se que todos os direitos autorais permanecem com o criador. O uso deste material neste trabalho tem caráter exclusivamente ilustrativo e acadêmico, sem fins comerciais, estando restrito à demonstração do projeto desenvolvido. Caso o jogo venha a ser distribuído publicamente ou com fins comerciais, o referido *asset* deverá ser substituído por outro com licença claramente definida ou desenvolvido internamente.

B.1 Comprovação Visual de Uso

A **Figura B.1** e **Figura B.2** consistem em capturas de tela utilizadas para comprovar as licenças dos *assets* utilizados neste trabalho.

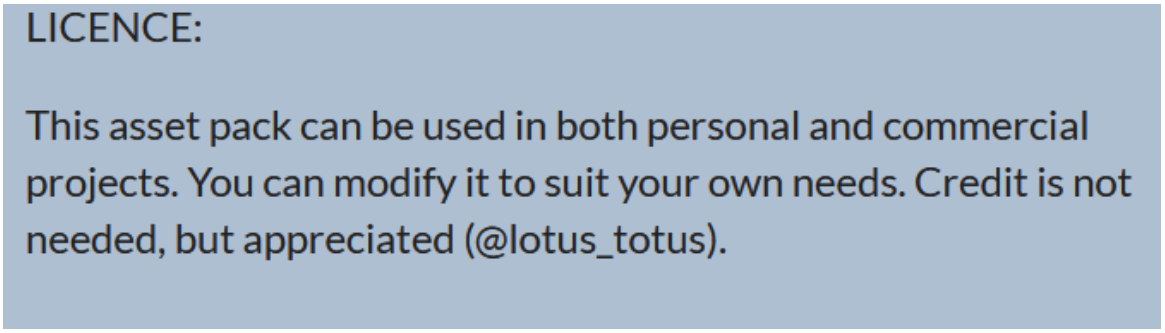

Figura B.1 – Comprovação da declaração de uso livre e irrestrito (sem atribuição) para os *assets* do autor TotusLotus.

Fonte: TotusLotus (Itch.io, 2025).

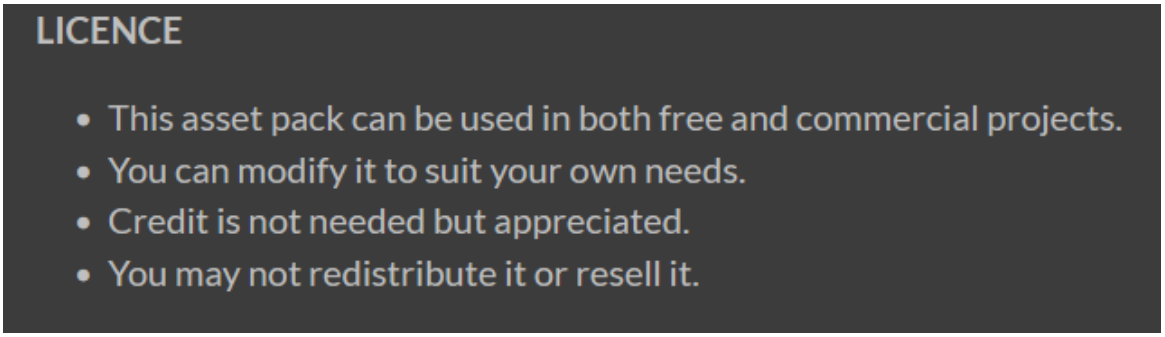

Figura B.2 – Comprovação da declaração de uso livre e irrestrito (sem atribuição) para os *assets* do autor Cainos.

Fonte: Cainos (Itch.io, 2025).