Jogo Labirinto Lógico: Usando um serious game para introduzir o pensamento computacional e conceitos de programação

Bernardo Hipólito Mundim Porto

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Uberlândia - MG
2026

Bernardo Hipólito Mundim Porto

Jogo Labirinto Lógico: Usando um serious game para introduzir o pensamento computacional e conceitos de programação

Trabalho de Conclusão de Curso de Desenvolvimento de Software apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciências da Computação.

Área de concentração: Ciências da Computação

Orientador: Renato de Aquino Lopes

Uberlândia - MG
2026

Agradecimentos

Agradeço ao meu orientador, Professor Renato de Aquino Lopes, pela orientação e pelo conhecimento compartilhado ao longo deste trabalho, à minha mãe, Flávia Hipólito, por estar sempre presente e à minha namorada, Amandha Portilho, pelo companheirismo.

A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho, meus sinceros agradecimentos.

Resumo

O ensino de Pensamento Computacional enfrenta desafios como a desmotivação e a dificuldade de aprendizado de conceitos abstratos de programação, especialmente entre iniciantes. Este trabalho apresenta o desenvolvimento de um jogo sério voltado à introdução dos pilares do Pensamento Computacional, com ênfase em pensamento algorítmico e decomposição de problemas. O jogo foi implementado em Python com a biblioteca Pygame e integra um pipeline de interpretação próprio, composto por analisador léxico, parser e interpretador, que processa uma pseudo-linguagem de programação digitada pelo jogador e a converte em movimentos executados por um personagem em um labirinto. O sistema oferece dois modos de jogo (Principal, com escrita de código, e Secundário, com comandos direcionais), cinco níveis progressivos com variações aleatórias de mapa, editor de código com feedback de erros em tempo real, sistema de pontuação baseado em desempenho e progressão com fases desbloqueáveis. Os resultados demonstraram que o jogo é funcional e estável, constituindo uma ferramenta de código aberto que pode ser utilizada como material de referência para o ensino introdutório de programação e para o estudo de técnicas de implementação de interpretadores em ambientes de jogos.

Palavras-chave: Pensamento Computacional, Jogos sérios, Interpretador, Python, Ensino de programação.

Lista de ilustrações

Lista de tabelas

Lista de siglas

AST Arvore Sintática Abstrata

2D Bidimensional - *Two-Dimensional*

3D Tridimensional - *Three-Dimensional*

FPS Quadros por Segundo - *Frames Per Second*

PC Computador Pessoal - *Personal Computer*

SDL Camada Simples de Mídia Direta - *Simple DirectMedia Layer*

Sumário

Capítulo 1

Introdução

O Pensamento Computacional tem se consolidado como uma competência essencial na formação de profissionais de tecnologia, porém seu ensino introdutório ainda enfrenta desafios como desmotivação e dificuldade na assimilação de conceitos abstratos. Este trabalho delimita-se ao desenvolvimento de um jogo sério 2D, capaz de executar uma pseudo-linguagem de programação e movimentar um personagem em um labirinto. A relevância do projeto reside na combinação de dois elementos: a oferta de um ambiente lúdico que estimula o raciocínio algorítmico e a decomposição de problemas, e a disponibilização de um código-fonte aberto e documentado, que serve como material de referência tanto para o ensino de programação quanto para o estudo da implementação de um interpretador embutido em um jogo usando conceitos de compiladores.

1.1 Contextualização do Trabalho

1.1.1 Pensamento Computacional

Por se tratar de uma área recente, pesquisadores da área atribuem diferentes definições no artigo de Cansu (CANSU, 2019) são citadas algumas dessas definições, nas quais conceitos como abstração e pensamento algorítmico estão presentes direta ou indiretamente na maioria delas.

No caso da abstração, Não deve-se por exemplo saber apenas isolar as características de um problema, mas sim saber aplicar este conceito em diferentes níveis dependendo da necessidade (WING, 2006). Já o pensamento algoritmo descreve os passos de forma generalizada para independente do estado inicial do problema (entrada) o seu passo a passo (algoritmo) consiga solucioná-lo.

Dois marcos da história do Computational Thinking são os artigos de Wing (2006) e Papert (1996), como apontado por Cansu (2019). O primeiro cunhou o termo como é usado hoje e ainda é altamente citado por artigos mais recentes, que discutem a importância deste tipo de ensino para a geração atual; o segundo, mesmo que não usasse do

mesmo termo (ao invés de Computational Thinking era usado Procedural Thinking), descrevia as mesmas características. Ambos os autores, Wing e Papert, veem o Pensamento Computacional como uma disciplina fundamental. No entanto, é necessário questionar como tal conceito pode ser ensinado de forma acessível. É nesse contexto que surge o conceito de gamificação.

1.1.2 Gamificação

Gamificação se apresenta como uma ferramenta para auxílio na resolução de problemas, aumento da motivação e aumento do engajamento de determinados públicos (BUSARELLO, 2016), para alcançar estes objetivos, é necessário tornar uma tarefa que outrora pudesse ser repetitiva e entediante, em um sistema que gratifica o indivíduo, usando por exemplo cenários lúdicos, uma forma de ”disfarçar” o aprendizado em um ambiente mais agradável e palatável.

Um cenário lúdico que é comumente utilizado na literatura são jogos digitais, mas não qualquer tipo, apenas aqueles que caem dentro da definição de jogos sérios, tendo, como o nome indica, uma abordagem mais séria que jogos convencionais, onde o foco principal é a transmissão de ”algo” para o jogador, sendo este ”algo” uma mensagem, habilidade ou, no contexto da gamificação, a transmissão de conhecimento(LAAMARTI; EID; SADDIK, 2014).

Neste contexto, trabalhos recentes como o de Shadbad(SHADBAD et al., 2023) demonstram o processo de gamificação em conjunto com outras ferramentas, neste caso, laboratórios virtuais, no qual a inserção de elementos simples de jogos, como uma tabela de classificação melhoraram o desempenho dos alunos significativamente.

1.1.3 Jogos na educação

Tendo estabelecido os conceitos de Gamificação e jogos sérios, agora é possível explorar de qual forma os mesmos podem ser aplicados. No livro ”Jogando jogos digitais” (RITTERFELD; WEBER, 2006) o autor separa os tipos de jogos usados em educação em três categorias.

1. Paradigma Motivacional: neste paradigma, o aspecto de entretenimento dos jogos digitais é usado como um suporte, ajudando a desenvolver o interesse inicial, garantindo que áreas do conhecimento que não são estimulantes por si mesmas sejam mais facilmente digeridas.

2. Paradigma de Reforço: ainda se tratando do papel do entretenimento, o paradigma de reforço não o usa como um atrativo como o anterior, mas agora o mesmo é visto como uma ”recompensa”; por exemplo, progredir no jogo ou atingir uma pontuação alta. Cria-se assim um ciclo, onde para conseguir satisfação, o jogador

necessita melhorar no jogo, aprender suas mecânicas e, consequentemente, assimilar o conteúdo desejado.

3. Paradigma de Mistura: diferente dos paradigmas anteriores, este não distingue entretenimento de aprendizado. Como o nome sugere, ambos são combinados; desta forma, o aprendizado é feito de forma implícita, dependente do jogador se empenhar em jogar para desenvolver suas habilidades. Um exemplo seria jogos de simulação, onde se pode adquirir conhecimento pelo fato de as mecânicas do jogo serem fiéis às habilidades necessárias para realizar a tarefa na vida real.

No trabalho de Zirawaga (ZIRAWAGA; OLUSANYA; MADUKU, 2017), pode-se perceber a aplicação desses paradigmas na aplicação de jogos digitais para o ensino de história, caça-palavras que possuem apenas palavras relevantes para o ensino, palavras cruzadas que necessitam que o aluno lembre de informações históricas para serem finalizados, o artigo por sua vez conclui a versatilidade dos jogos digitais, podendo atender às diferentes necessidades, como pensamento lógico, aprendizado inicial ou reforço de aprendizado.

Outro trabalho, de Anastasiadis (ANASTASIADIS; LAMPROPOULOS; SIAKAS, 2018), discute seus reais benefícios, o artigo propõe que os mesmos não alcançam apenas o aprendizado fundamental, como desenvolvimento cognitivo e pensamento crítico, mas também aspectos sociais, como desenvolvimento de relações interpessoais e aumento de autoestima, demonstrando benefícios que vão além do aprendizado técnico.

Apesar da versatilidade, estes não são triviais de serem desenvolvidos, por consequência, surgiram ferramentas capazes de padronizar este processo e facilitá-lo. Como o uso de bibliotecas típicas para isso.

1.1.4 Biblioteca Pygame

A biblioteca Pygame é um conjunto de módulos Python construídos sobre a Camada Simples de Mídia Direta (SDL), oferecendo uma camada de abstração de alto nível que simplifica o desenvolvimento de aplicações multimídia, especialmente jogos. Sendo orientada a objetos, ela encapsula chamadas à SDL em C, permitindo que o programador controle, a criação e o dimensionamento da janela visual, imagens e texto diretamente na superfície da tela. O gerenciamento da área de exibição é feito por meio de objetos Surface, que representam regiões retangulares onde é possível desenhar pixels, carregar e transformar imagens, aplicar transparências e realizar operações de blit (cópia rápida entre superfícies).

Além do controle do que é exibido, oferece um sistema de detecção e tratamento de eventos, capaz de monitorar teclado, mouse e até mesmo eventos de janela como redimensionamento ou fechamento. O módulo pygame.time fornece relógios com controle de Quadros por Segundo (FPS) e temporizadores, para manter a lógica de atualização e renderização sincronizadas.

Para manipulação de elementos visuais, pygame.draw fornece funções otimizadas para desenhar linhas, círculos, polígonos e outras formas geométricas. Já o pygame.font trabalha com renderização de textos, utilizando fontes do sistema ou arquivos TrueType, possibilitando a criação de interfaces com menus, pontuações e diálogos.

Outra funcionalidade relevante é o suporte a sprites por meio do módulo pygame.sprite, que organiza objetos gráficos em grupos, facilitando a detecção de colisões e a atualização coletiva dos elementos. Tendo assim um pacote de funcionalidades completo para a criação de jogos.

1.2 Problema e Justificativas

O problema central que este trabalho pretende enfrentar é a dificuldade de introduzir os conceitos fundamentais do Pensamento Computacional e de programação a estudantes que almejam ingressar (ou acabaram de ingressar) em cursos superiores na área de computação. A literatura indica que o aprendizado inicial de programação é frequentemente percebido como uma atividade puramente técnica e desmotivadora, o que compromete o desenvolvimento de habilidades essenciais como abstração e pensamento algorítmico.

A justificativa para a escolha desse problema reside na necessidade de abordagens pedagógicas que engajem os alunos e facilitem a transição do pensamento concreto para o lógico-abstrato. Jogos sérios têm demonstrado potencial para melhorar a motivação e a retenção de conhecimento, configurando-se como uma alternativa promissora para complementar os métodos tradicionais de ensino. Assim, este trabalho se insere em um contexto de pesquisa que busca validar e expandir o uso de jogos digitais como ferramentas efetivas para o desenvolvimento do Pensamento Computacional.

1.3 Objetivos (Gerais e Específicos)

O objetivo geral deste trabalho é contribuir para o ensino introdutório de programação e Pensamento Computacional por meio da concepção de um jogo educativo digital.

Os objetivos específicos são:

1. Investigar, na literatura, os principais pilares do Pensamento Computacional abordados em jogos sérios e como estes são implementados.

2. Analisar as estratégias de gamificação adotadas em jogos correlatos, como Program Your Robot (KAZIMOGLU et al., 2012) e CodeCombat (INC., 2013), identificando seus potenciais e limitações.

3. Desenvolver um jogo do gênero labirinto que integre desafios de lógica e programação, voltado a iniciantes.

1.4 Método Proposto

Tomando como referência o diagrama de criação de serious games apresentado por (PENDLETON, 2020), o método proposto para o desenvolvimento deste trabalho organiza-se nas seguintes etapas:

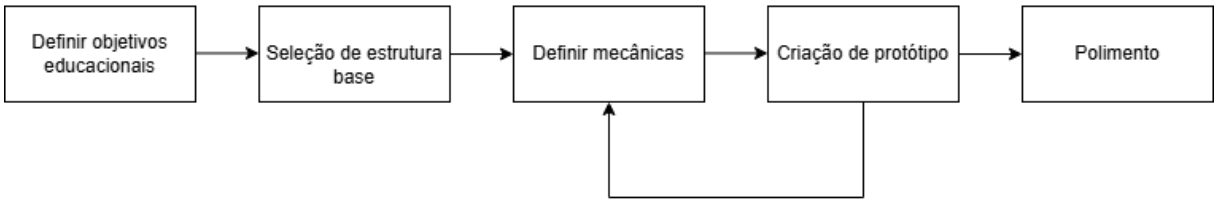

Figura 1 – Diagrama Método Proposto.

1. **Definição dos objetivos educacionais**: estabelecer os pilares do Pensamento Computacional abordados, bem como os conceitos básicos de programação a serem introduzidos.

2. **Seleção da estrutura base**: escolher a plataforma, a engine e o gênero do jogo — por exemplo, definir se a aplicação é para computador pessoal ou dispositivo móvel e qual o gênero adotado.

3. **Definição das mecânicas**: especificar as principais formas de interação do jogador com o jogo. Com base nos objetivos educacionais, pode-se, por exemplo, associar o conceito de abstração ao uso de funções ou o pensamento algorítmico a mecânicas que empreguem laços de repetição (loops).

4. **Criação de protótipo**: implementar as mecânicas escolhidas e avaliar se o ciclo de jogo resultante gera engajamento e, simultaneamente, cumpre o papel educativo. Nesta etapa, não são trabalhados os aspectos visuais finais nem o desempenho, havendo foco exclusivo na funcionalidade inicial. Caso o resultado não seja ideal, retorna-se à etapa de definição de mecânicas para os ajustes necessários.

5. **Polimento**: com o protótipo validado, são incorporados elementos como, interface visual e sprites (imagens de personagens e cenários), componentes que não eram estritamente necessários ao funcionamento do jogo, mas que enriquecem a experiência do usuário.

1.4.1 Definição dos Objetivos Educacionais

Levando em consideração os exemplos de jogos educacionais aplicados anteriormente, um pilar indispensável é o pensamento algorítmico, presente em todos eles. Quanto à escolha de um pilar secundário, abrem-se mais possibilidades: por se tratar de um jogo introdutório ao Pensamento Computacional, os artigos voltados a públicos similares (com pouco ou nenhum conhecimento prévio) optaram pela decomposição de problemas.

Sendo assim, o jogo foi criado seguindo os modelos de CodeCombat, Elevator Saga e Program Your Robot, nos quais o objetivo é que o jogador compreenda uma pseudo-linguagem de programação e conduza um personagem até o fim de um caminho. Outro modo criado apresenta os comandos fornecidos ao personagem e pergunta ao jogador em que lugar do mapa ele irá parar após executar os passos (ou até mesmo pedir que o jogador monte o trajeto).

1.4.2 Seleção de Estrutura Base

Primeiramente, para a decisão da plataforma a ser utilizada, haveriam essencialmente três opções: consoles, dispositivos móveis (jogos mobile) e PCs. Pela necessidade de escrita constante durante o jogo, seria inviável desenvolvê-lo para mobile. Considerando também a acessibilidade e a complexidade de desenvolvimento, pode-se igualmente excluir os consoles, restando, assim, a plataforma PC, na qual os jogos usados como modelo também foram desenvolvidos.

Com a plataforma definida, a questão central passou a ser a escolha da engine ou da linguagem a ser utilizada na codificação. Para isso, foi necessário compreender as necessidades do jogo. A mecânica central consiste em movimentar um personagem por um labirinto utilizando comandos discretos (cima, baixo, esquerda, direita), sobre um grid. Como a ação relevante (se mover para espaços adjacentes) é puramente lógica e independe da perspectiva visual, representá-la em um ambiente 3D não acrescentaria nada substancial. Pelo contrário, exigiria modelagem tridimensional, controle de câmera e maior poder de processamento, complexidades que não se justificam para um jogo cujo foco está no raciocínio algorítmico, e não nos gráficos. Portanto, a opção lógica foi manter a simplicidade e optar pelo modelo 2D, excluindo, assim, engines como Unity e Godot (focadas em jogos 3D). No entanto, durante a seleção e os testes de engines 2D, surgiu o problema da inviabilidade de construir uma pseudo-linguagem funcional dentro dessas ferramentas, uma vez que elas são voltadas para a criação de jogos com mecânicas de plataforma ou de aventura.

Dessa forma, o desenvolvimento se deu sem o auxílio de engines. Quanto à escolha da linguagem, se fosse um jogo computacionalmente pesado, faria sentido optar por uma linguagem de mais baixo nível, como C++, amplamente utilizada em jogos por sua capacidade de otimização. Entretanto, o jogo a ser desenvolvido não exige grande poder

computacional, o que dá a possibilidade de escolher uma linguagem de alto nível, como Python, que dispõe da biblioteca Pygame, ideal para a construção de jogos 2D. Assim, foi desenvolvido um compilador de uma linguagem simples em Python, conectado a um front-end feito em Pygame, onde ocorrerá o loop principal do jogo.

1.4.3 Definição de Mecânicas

Focando agora na progressão do jogo, pode-se dividir os comandos dados ao personagem em 4 categorias (estas estando em ordem crescente de complexidade):

1. **Comandos básicos**: Sendo a movimentação o núcleo do jogo, os comandos de deslocamento são elementares e, portanto, devem ser os primeiros a serem introduzidos. Esses comandos têm como única funcionalidade mover o jogador em quatro direções: para cima, para baixo, para a esquerda e para a direita.

2. **Comandos de repetição**: Uma vez dominada a movimentação básica, o passo lógico seguinte é capacitar o jogador a repetir um comando determinado número de vezes sem precisar reescrevê-lo manualmente. Isso exige o ensino de estruturas de repetição, como while e for.

3. **Comandos de decisão**: Em seguida, introduz-se a possibilidade de tomar diferentes caminhos no código conforme condições estabelecidas, ampliando o leque de soluções para percursos mais complexos com uma quantidade mínima de comandos.

4. **Criação de funções**: Sendo o conceito mais complexo a ser abordado, a criação de funções revela a utilidade de reutilizar trechos de código, abstraindo a complexidade de múltiplas ações por meio de uma chamada de função.

1.4.4 Considerações sobre a metodologia

As etapas do protótipo e polimento, previstas no método proposto, serão detalhadas nos capítulos seguintes, dedicados à análise do projeto e ao desenvolvimento do sistema, nos quais se descreverão as decisões de implementação, os ajustes e a incorporação dos elementos visuais que conferem ao jogo sua aparência final. Desta forma, termina-se o estabelecimento da metodologia.

1.5 Organização do Trabalho

Este trabalho está organizado da seguinte maneira:

- ❏ No Capítulo 2, são apresentados os trabalhos relacionados.
- ❏ O Capítulo 3 detalha os requisitos do sistema.

- O Capítulo 4 trata da análise e projeto do sistema.
- O Capítulo 5 traz o desenvolvimento do sistema, discutindo os resultados alcançados e sugerindo trabalhos futuros.
- Por fim, o capítulo 6 traz a conclusão, discutindo os resultados alcançados e sugerindo trabalhos futuros.

1.6 IA-Generativa

Foi utilizada a LLM DeepSeek (DEEPSEEK, 2025) para correção ortográfica. O *prompt* usado em todos os parágrafos foi: "corrija os erros ortográficos do texto a seguir sem modificar nada da estrutura nem das palavras". Após isso, o texto gerado era revisado manualmente para garantir que a IA havia apenas acentuado palavras devidamente e reescrito as que sofreram erros tipográficos (erros de engano de digitação), garantindo assim que a ferramenta não tivesse alucinado e modificado trechos ou removido/adicionado palavras novas.

Capítulo 2

Trabalhos Relacionados

Dentre os trabalhos que relacionam Pensamento Computacional e jogos digitais, alguns concentram-se naqueles específicos para ensino de conceitos de programação, outros investigam como o ato de criar os mesmos pode revelar e aprimorar habilidades computacionais nos estudantes. Há ainda pesquisas que buscam comparar a eficácia dessa abordagem com métodos tradicionais de ensino, avaliando tanto a aquisição imediata de conhecimento quanto sua retenção em longo prazo. Os estudos aqui revisados abordam diferentes pilares do Pensamento Computacional, como pensamento algorítmico, abstração e reconhecimento de padrões, há também trabalhos que não se configuram como pesquisas científicas, mas seus projetos encapsulam a mesma categoria de jogo desenvolvida neste trabalho.

2.1 Code Combat

A plataforma CodeCombat(INC., 2013) configura-se como um ambiente educacional gamificado voltado ao ensino de programação, fundamentado em uma narrativa de RPG na qual o aprendiz controla um personagem exclusivamente por meio de comandos escritos em linguagens, como Python, JavaScript e C++ (Python Software Foundation, 2023; Ecma International, 2023; cppreference.com, 2023). O jogo foi originalmente desenvolvido em CoffeeScript(CoffeeScript Team, 2021) e é distribuído como uma aplicação web. O sistema opera diretamente no navegador, o que reduz significativamente as barreiras técnicas de entrada. Sua interface divide-se entre um editor de código e uma área de execução visual que exibe, em tempo real, o resultado das instruções submetidas. A progressão didática é estruturada em fases que introduzem gradualmente conceitos de programação, das sequências simples de movimentação a estruturas de controle, laços, condicionais, variáveis, funções e algoritmos mais elaborados, através de objetivos como derrotar inimigos ou coletar itens. Recursos como dicas progressivas, sistema de conquistas e pontuação por eficiência de código integram a gamificação.

Dessa forma, o CodeCombat estabelece uma plataforma não só para o aprendizado

de programação, mas também para o desenvolvimento do pensamento computacional, através de elementos narrativos e interativos.

Figura 2 – Nível 1 do Code Combat

Na Figura 2 podemos ver que a primeira fase começa simples, apenas necessitando dar comandos de movimento para o personagem parar faze-lo chegar ao fim.

2.2 Elevator Saga

O Elevator Saga(WOLFFELT, 2013) caracteriza-se como uma simulação orientada a eventos, implementada integralmente em JavaScript(Ecma International, 2023) e executada no navegador. Sua arquitetura é organizada em camadas independentes que separam a engine de simulação, a execução do código do usuário e a renderização gráfica.

A simulação é sustentada por um ciclo contínuo de atualização que modela a passagem do tempo e o comportamento físico dos elevadores, gerenciando três categorias de entidades, elevadores, passageiros e andares, por meio de uma interface de programação que restringe o acesso direto às propriedades internas desses elementos. O jogador deve programar, exclusivamente em JavaScript, as funções init e update que determinam o comportamento dos elevadores. Em init, são definidas as configurações iniciais, enquanto update é invocada a cada ciclo da simulação, recebendo informações sobre o estado do elevador, as solicitações de transporte e as restrições do ambiente, e devolvendo decisões como a direção de movimento e a abertura ou fechamento de portas. A interface visual apresenta o deslocamento dos elevadores e o fluxo de passageiros, oferecendo métricas de desempenho como tempo médio de espera e total transportado,funcionando assim como critérios de avaliação da solução implementada.

Dessa forma, o Elevator Saga configura-se como um ambiente de aprendizagem, que por meio de um problema concreto de logística e otimização, estimula o desenvolvimento de habilidades de abstração e planejamento algorítmico. Pode-se perceber isso na Figura 3 abaixo, a implementação de um looping e da chamada de função ”goToFloor” simulando o comportamento de um elevador através de lógica de programação.

Figura 3 – Nível 1 do Elevator Saga

2.3 Um jogo sério para o desenvolvimento do pensamento computacional e a aprendizagem de programação introdutória

O artigo (KAZIMOGLU et al., 2012) aponta a falta do Pensamento Computacional como principal motivo para a dificuldade no aprendizado de programação de computadores. Os autores apontam que os alunos veem a programação como uma atividade puramente técnica, mantendo-se em um nível superficial, não conseguindo desenvolver, por exemplo, um pensamento algorítmico.

Neste contexto foi apresentado como uma solução o jogo sério ”Program your robot”, desenvolvido pelo autor do artigo usando Adobe Flash CS5 (Adobe Systems, 2010b) na linguagem ActionScript 3 (Adobe Systems, 2010a), separado em fases; cada uma consiste em dar comandos para o robô através de um caminho com o objetivo de chegar ao final do mesmo. Os comandos são divididos entre ações de movimento e estruturas de programação básicas (decisão, looping e funções). Para incentivar o desenvolvimento do Pensamento

Computacional, jogadores recebem mais pontos nos níveis quando usam de forma eficiente as estruturas mencionadas.

Os autores aplicaram este jogo em 25 alunos com níveis diferentes de conhecimento, tendo majoritariamente uma recepção positiva, na qual acreditam que o jogo consegue introduzir pessoas sem conhecimento de programação aos conceitos básicos e à habilidade de resolução de problemas.

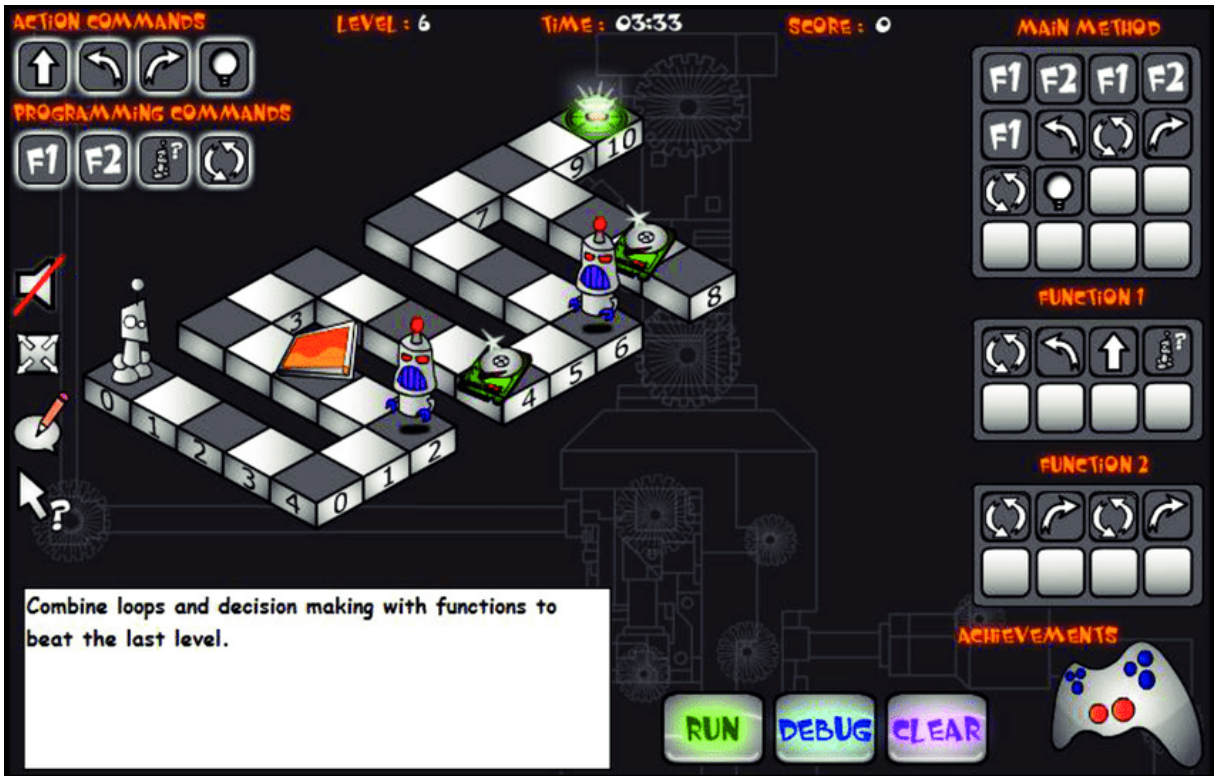

Figura 4 – Nível 6 do Program Your Robot

2.4 Meu Jogo Está Bom, Dr. Scratch?: Explorando o Desenvolvimento de Programação e Pensamento Computacional por meio de Métricas em Jogos Sérios Projetados por Estudantes para STEM

Troiano (TROIANO et al., 2019) usa uma abordagem diferente; agora os alunos não são avaliados ao jogar, mas sim ao desenvolver o próprio jogo. A linguagem utilizada foi Scratch, baseada em programação em blocos, especificamente mais didática para ensino de programação para crianças.

Os estudantes seriam avaliados em 7 características do Pensamento Computacional: abstração, representação de dados, controle de fluxo, lógica, paralelismo, sincronização

e interatividade do usuário. Cada uma sendo avaliada de 0 a 3 de acordo com o quão bem cada característica foi aplicada, para um total possível de 21 pontos, divididos em 1-7 (básico), 7-14 (desenvolvendo) e 15-21 (proficiente). Foram considerados a nota final de cada jogo (nomeada no artigo como proficiência em Pensamento Computacional) e como essa nota foi aumentando ao longo do desenvolvimento do jogo (desenvolvimento do Pensamento Computacional).

Por fim, os autores alcançaram resultados promissores. Dos 317 jogos avaliados em proficiência em Pensamento Computacional, a média de pontos foi 14.07, considerada na métrica anterior como proficiente (14-21). Analisando as características individuais, foi possível perceber que paralelismo, lógica e sincronização foram onde os alunos mais conseguiram notas 3 (proficiência). Em contrapartida, abstração consistentemente recebeu nota 1 (básico). Durante a avaliação do desenvolvimento de Pensamento Computacional é possível notar essa discrepância, onde os alunos que implementaram abstração normalmente o faziam no final do desenvolvimento, mostrando ser uma habilidade de maior complexidade de ensino quando se usa como ferramenta o desenvolvimento de jogos.

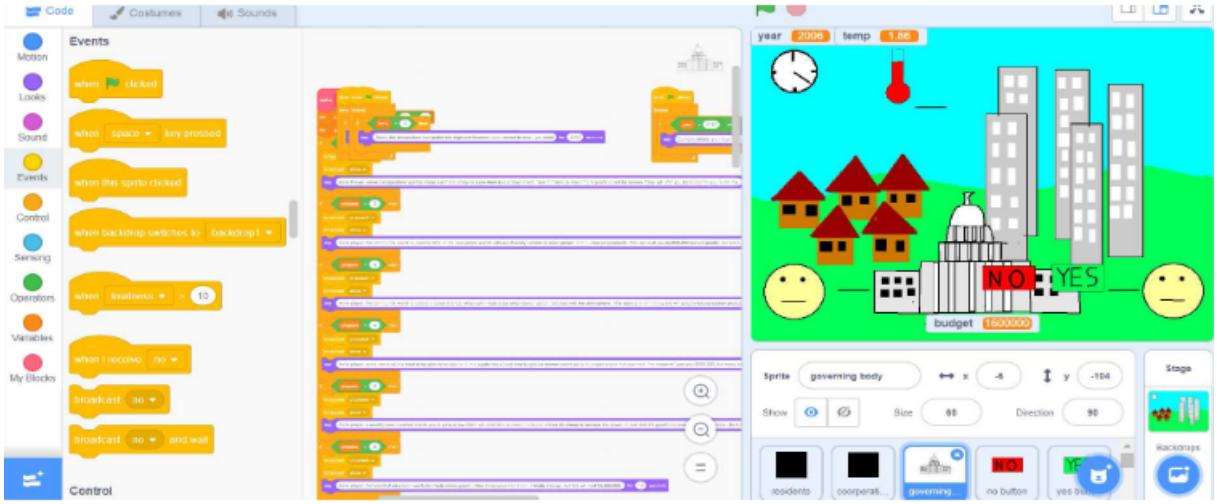

Figura 5 – Um dos jogos serios desenvolvidos pelos alunos

2.5 CTArcade: Pensamento Computacional com jogos em crianças em idade escolar

Já o artigo de Lee(LEE et al., 2014), consiste em introduzir 18 crianças a uma plataforma virtual (CTArcade), onde foi desenvolvida uma versão diferente do ”jogo da velha”. A diferença se dá que os estudantes não jogam diretamente o jogo, mas sim programam um personagem; este personagem obedece à sequência de regras definidas, por exemplo: ”sempre marcar o quadrado do centro, se ele já não estiver marcado”. Após a programação do personagem, ele é colocado em um torneio com outras máquinas, onde o aluno pode avaliar se sua estratégia foi bem-sucedida ou precisa de ajustes.

Comparado ao ”jogo da velha” tradicional, jogado no papel, os pesquisadores conseguiram avaliar as diferenças em desenvolvimentos de características do Pensamento Computacional. Quando jogado no CTArcade, os alunos articularam em média 5 instâncias de pensamento algorítmico, enquanto no papel quase nunca o faziam (média entre 0 e 1). Mas, em contraste, o enfoque da versão virtual no pensamento algorítmico prejudica a habilidade de reconhecimento de padrões, que possui uma média significativamente maior no jogo em papel. Os autores concluem que suas descobertas são importantes não apenas para pesquisadores de Interação Humano-Computador, mas também para pesquisadores na área da educação.

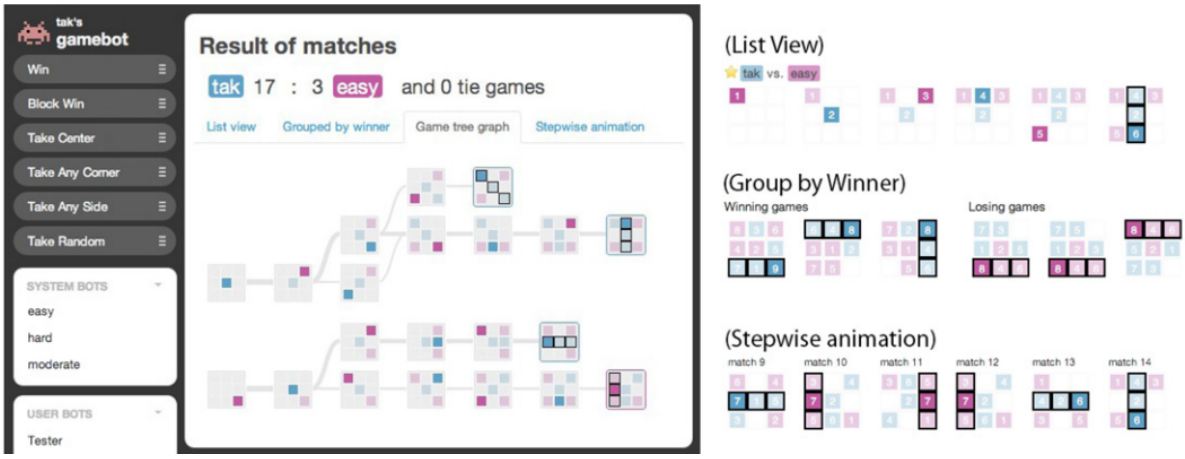

Figura 6 – Exemplo de um personagem programado em um torneiro

2.6 Microjogos educacionais de computador são envolventes e eficazes para a aquisição de conhecimento no ensino médio? Um estudo quase-experimental

O estudo (BROM; PREUSS; KLEMENT, 2011) utilizou um design quase-experimental envolvendo 100 estudantes. A metodologia consistiu em uma sessão de 90 minutos iniciada por uma aula teórica expositiva de 40 minutos sobre aprendizagem animal. Em seguida, as turmas foram divididas aleatoriamente em dois grupos: o grupo experimental, que jogou o micro-jogo Orbis Pictus Bestialis (OPB) por 20 minutos com orientação do professor, e o outro recebeu uma aula extra de 20 minutos com materiais multimídia. O objetivo do jogo era reforçar conhecimentos vistos previamente na aula. A eficácia foi avaliada por meio de testes de conhecimentos imediatos e um teste de retenção realizado após um mês, além de questionários o valor educacional percebido.

Os resultados indicaram que o uso do micro-jogo é tão eficaz quanto a instrução tradicional para o conhecimento a curto prazo, mas apresenta uma vantagem significativa na retenção a longo prazo. Além disso, o estudo demonstrou que o uso do jogo não prejudicou o aprendizado de informações transmitidas na aula expositiva que não foram reforçadas no mesmo. Embora cerca de 90% dos alunos tenham gostado do jogo, o grupo da aula tradicional atribuiu um valor educacional percebido ligeiramente superior ao seminário em comparação ao grupo do jogo.

Figura 7 – Imagem da tela inicial do jogo

2.7 Tabela de Comparação

Observa-se na Tabela 1 que o pilar mais abordado nos artigos analisados é o Pensamento Algorítmico. Outro pilar verificado é a Abstração, discutida nos artigos (KAZIMOGLU et al., 2012) e (TROIANO et al., 2019) Contudo, os resultados indicam desafios significativos em seu ensino. Isso pode se dar pelas características do estudo realizado, sendo majoritariamente estudos preliminares e exploratórios, estando, portanto, ainda numa fase de formulação de hipóteses. Na Tabela 1 os pilares do Pensamento Computacional estão associados aos seguintes número: 1 = Pensamento Algorítmico, 2 = Decomposição de Problemas, 3 = Reconhecimento de Padrões e 4 = Abstração.

Os jogos desenvolvidos focam em gêneros nos quais o raciocínio lógico e o conhecimento prévio são esperados do jogador, aspectos estes que auxiliam o público-alvo, crianças e pessoas iniciantes em programação, a desenvolverem o Pensamento Computacional. Diferentemente de gêneros como jogos de corrida ou ação, cujo desafio é baseado em habilidades motoras.

Em todos os artigos, utilizaram-se de computadores para os jogos, possivelmente por sua maior acessibilidade; três deles podem ser acessados pelo navegador, sem necessidade de instalação ou do uso de hardware específico.

Artigo	**Tipo do estudo**	**Pilares**	**Jogo desenvolvido**	**Público-alvo**	**Estratégias de engajamento(Gamificação)**	**Plataforma**	**Foco de ensino do artigo**
(KAZIMOGLU et al., 2012)	Estudo preliminar/exploratório	1,2,3,4	Jogo de plataforma	Iniciantes em programação	Progressão em níveis, sistema de pontos	Computador	Desenvolver o pensamento computacional e introduzir conceitos de programação com desafios lógicos progressivos.
(TROIANO et al., 2019)	Estudo de caso	1,4	NA	Alunos de 13 a 14 anos	NA	Computador	Introduzir os pilares de abstração e algoritmos do pensamento computacional a estudantes do ensino fundamental
(LEE et al., 2014)	Estudo exploratório	1,3	Jogo de estratégia	Alunos de 7 a 11 anos	Torneios, seleção de dificuldade	Computador	Ensinar os conceitos de reconhecimento de padrões e projeto de algoritmos para crianças por meio de um jogo de estratégia
(BROM; PREUSS; KLEMENT, 2011)	Estudo piloto/quase-experimental	1,2	Simulador	Alunos do ensino médio	Progressão em níveis	Computador	Ensinar fundamentos de ciência da computação a alunos do ensino médios
(INC., 2013)	NA	1,2,3,4	RPG de aventura	Iniciantes em programação de todas as idades	Progressão em níveis, conquistas	Computador	Ensino de programação introdutória com sintaxe real (Python, JavaScript)
(WOLFFELT, 2013)	NA	1,2,3,4	Simulador	Estudantes de programação básica	Métricas de desempenho, metas de eficiência	Computador	Otimização de algoritmos de escalonamento e tomada de decisão em tempo real
Meu Trabalho	Pesquisa de desenvolvimento	1,2	Jogo de Labirinto	Pessoas sem conhecimento de programação	Pontuação e leaderboard	Computador	Introduzir pessoas que querem cursar ensino superior em faculdades de programação ao pensamento computacional

Tabela 1 – Análise de artigos sobre serious games e Pensamento Computacional

Capítulo 3

Requisitos do Sistema

Este capítulo tem como objetivo descrever em linhas gerais o sistema desenvolvido, assim como seu público alvo e seus requisitos (funcionais e não funcionais).

3.1 Descrição geral do sistema

Primeiramente, para a decisão da plataforma a ser utilizada, haveria essencialmente três opções: consoles, dispositivos móveis (jogos mobile) e Computador Pessoal (PC). Como o jogo utiliza bastante do teclado (escrita de pseudo-código), seria inviável desenvolvê-lo para mobile. Considerando também a acessibilidade e a complexidade de desenvolvimento, podería-se igualmente excluir os consoles, restando, assim, a plataforma PC, na qual os jogos usados como modelo também foram desenvolvidos.

Com a plataforma definida, a questão central passou a ser a escolha da engine ou da linguagem a ser utilizada na codificação. Para isso, foi necessário compreender as necessidades do jogo. Como o jogo se baseia em um labirinto em grade resolvido por comandos discretos de movimento, uma perspectiva Bidimensional (2D) seria suficiente, enquanto um ambiente Tridimensional (3D) adicionaria complexidade visual e de câmera que não contribuiria para o aprendizado do Pensamento Computacional. Assim, o modelo 2D, similar ao adotado pelo CodeCombat, exclui da escolha engines focadas em jogos 3D como Unity(Unity Technologies, 2023) e Godot (Godot Engine, 2023).

No entanto, durante a seleção e os testes de engines 2D, surgiu o problema da inviabilidade de construir uma pseudolinguagem funcional dentro dessas ferramentas, uma vez que elas são voltadas para a criação de jogos com mecânicas de plataforma ou de aventura.

Dessa forma, o desenvolvimento se deu sem o auxílio de engines. Quanto à escolha da linguagem, se estivesse tratando de um jogo computacionalmente pesado, faria sentido optar por uma linguagem de mais baixo nível, como C++, amplamente utilizada em jogos por sua capacidade de otimização. Entretanto, o jogo desenvolvido não exige grande poder computacional, o que nos dá a possibilidade de escolher uma linguagem de alto nível, como

Python, que dispõe da biblioteca Pygame, ideal para a construção de jogos 2D. Assim, foi desenvolvido um compilador de uma linguagem simples em Python, conectado a um front-end feito em Pygame, onde ocorre o loop principal do jogo.

3.2 Abrangência e sistemas relacionados

Desta forma, as principais funcionalidades oferecidas são:

1. Entrada de código pelo jogador, digitado diretamente no ambiente do jogo;
2. CInterpretação com análise léxica, sintática e semântica para extrair as instruções de movimento;
3. Visualização do labirinto e do personagem em visão superior, utilizando a biblioteca Pygame;
4. Execução animada dos movimentos sobre o labirinto, conforme os comandos interpretados;

Tendo como restrições(escopo negativo):

1. Versões para dispositivos móveis ou consoles: dependente do teclado para a digitação do pseudo-código, tornando inviável a adaptação para telas touch ou gamepads; além disso, o desenvolvimento para consoles implicaria maior complexidade e menor acessibilidade ao público-alvo inicial.
2. Criação de fases ou labirintos pelo próprio jogador (editor de níveis): o foco está na resolução dos desafios pré-definidos, que foram projetados para explorar conceitos específicos de programação.
3. Suporte a múltiplos jogadores online, leaderboard e persistência de progresso em nuvem: o jogo é executado localmente, sem sem servidores e suporte online.
4. Integração com linguagens de programação reais (Python, C etc.): a pseudolinguagem interna é limitada e não possui a sintaxe ou as bibliotecas de uma linguagem de propósito geral.

Como mencionado anteriormente, o sistema é independente e totalmente autocontido. Não se comunica com nenhum outro software externo, servidor ou serviço web. Toda a lógica do jogo reside em uma única aplicação, sem dependência de engines de jogos de terceiros, apenas da biblioteca Pygame e de bibliotecas auxiliares (como a biblioteca ”time”).

3.2.1 Looping Principal

O funcionamento do sistema pode ser compreendido como um ciclo contínuo de transformação de código em movimento (Figura 8). A cada interação, o jogador fornece um programa, e o jogo o converte em uma animação visual dentro do labirinto. Esse ciclo é composto pelas seguintes etapas:

1. **Escrita do código.** Utilizando o editor de texto, o jogador escreve um programa.
2. **Interpretação.** Ao solicitar a execução (tecla `F5`), o texto é extraído do editor e submetido ao pipeline de interpretação, que se divide em três etapas consecutivas:
   - ❑ **Lexer — Análise Léxica.** O analisador léxico percorre o código-fonte caractere por caractere e o decompõe em unidades significativas chamadas *tokens*. Ele identifica números, identificadores (nomes de variáveis e funções), operadores aritméticos e relacionais, delimitadores (parênteses, chaves, ponto-e-vírgula) e palavras reservadas como `if`, `else`, `while`, `def` e `return`. Espaços em branco e quebras de linha são descartados, mas suas posições são registradas para que, em caso de erro, o sistema possa informar a linha onde ocorreu. Se o Lexer encontrar um caractere que não corresponde a nenhum padrão da linguagem ele interrompe a análise e sinaliza o erro, impedindo que o código inválido prossiga para as etapas seguintes.
   - ❑ **Parser — Análise Sintática.** De posse da lista de tokens fornecida pelo Lexer, o analisador sintático verifica se a sequência obedece às regras gramaticais da pseudo-linguagem. Ele opera por descida recursiva: cada construção da linguagem (`while`, `if`, atribuições, definições de funções) é tratada por um método específico que consome os tokens esperados. Enquanto analisa, o Parser constrói a Arvore Sintática Abstrata (AST), uma estrutura hierárquica em que cada nó representa uma construção do programa, como um while por exemplo..
   - ❑ **Interpreter — Interpretação.** O interpretador percorre a AST utilizando o padrão de projeto Visitor. Para cada tipo de nó da árvore, existe um método "visit" correspondente que sabe como executá-lo. Quando um comando de movimento (`move_up`, `move_right`, etc.) é executado, o interpretador o registra em uma lista de movimentos.
3. **Execução dos movimentos.** Com a lista de comandos gerada pelo Interpreter, o Pygame:
   - ❑ Consulta a matriz do mapa para verificar se a célula de destino está livre.
   - ❑ Se o caminho estiver desimpedido move o player.

- ❏ Atualiza a matriz.

4. **Verificação de vitória.** Após cada movimento, o sistema confere se o personagem alcançou o ponto de chegada (`C`). Em caso positivo, a fase é concluída e a pontuação é calculada.

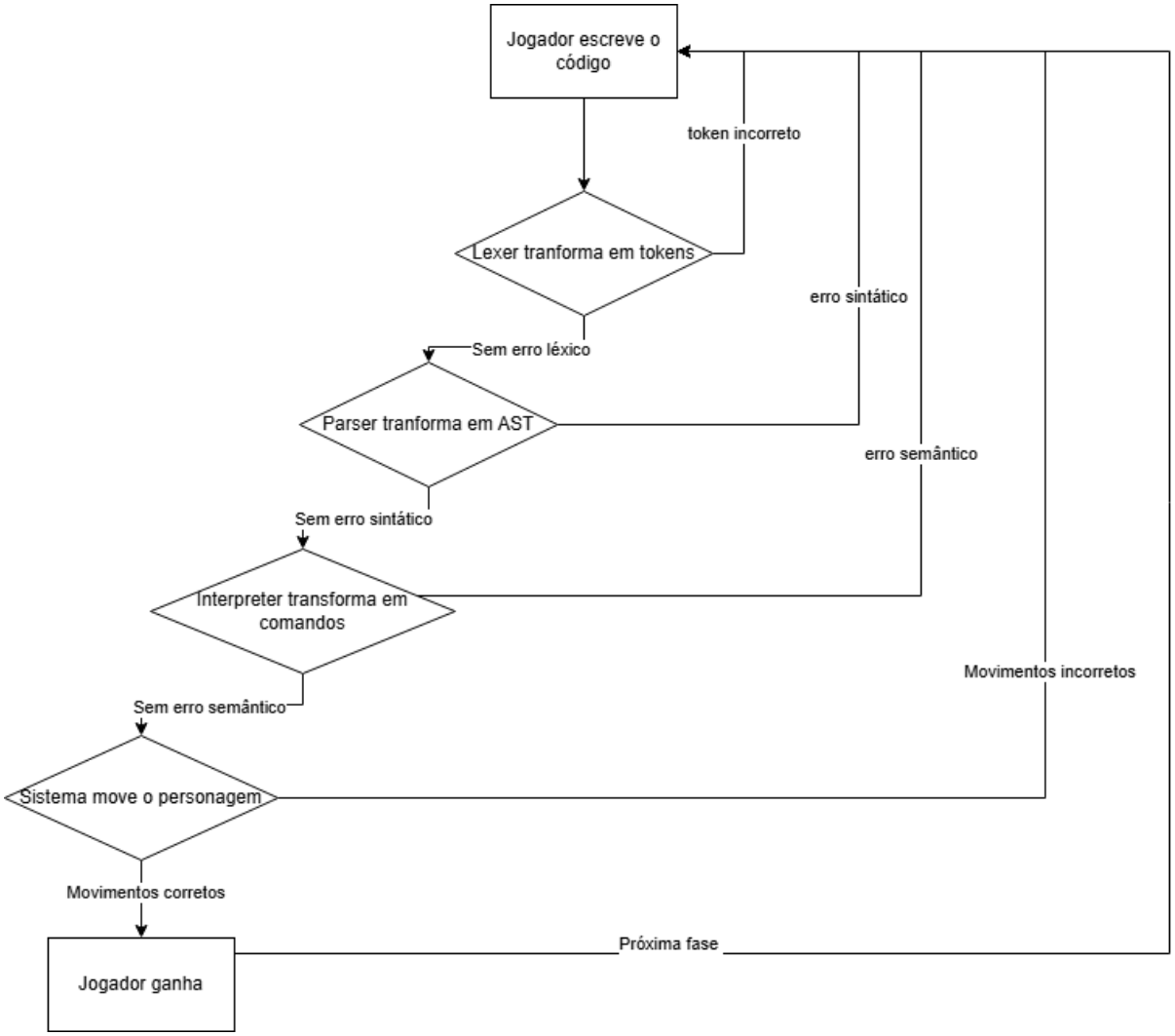

Figura 8 – Diagrama Looping principal

3.2.2 Descrição dos usuários

Diferentemente dos artigos mencionados na contextualização, o público-alvo deste trabalho são os universitários, mais especificamente aqueles que ingressam em cursos voltados para a programação (Engenharia da Computação, Ciências da Computação e Sistemas de Informação), os quais apresentam uma taxa de evasão superior à média dos demais cursos (Instituto Semesp, 2023).

Um dos motivos para essa evasão elevada é a complexidade das disciplinas de programação e de matemática (ALVIM et al., 2024), somada à expectativa dos professores universitários de que os alunos já possuam conhecimentos prévios antes de ingressarem na universidade (RODRIGUES, 2013). Portanto, o jogo tem como objetivo auxiliar o aluno na preparação prévia para essas disciplinas, introduzindo conceitos fundamentais e facilitando, posteriormente, o aprendizado de conteúdos mais complexos.

3.3 Requisitos funcionais - RF

Com isto em mente, precisa definir os casos de uso que o jogador seja capaz de acessar 3 funcionalidades, jogar o modo principal, o modo secundário e por fim, visualizar sua pontuação para cada um dos níveis.

3.3.1 Diagrama de Caso de Uso do sistema

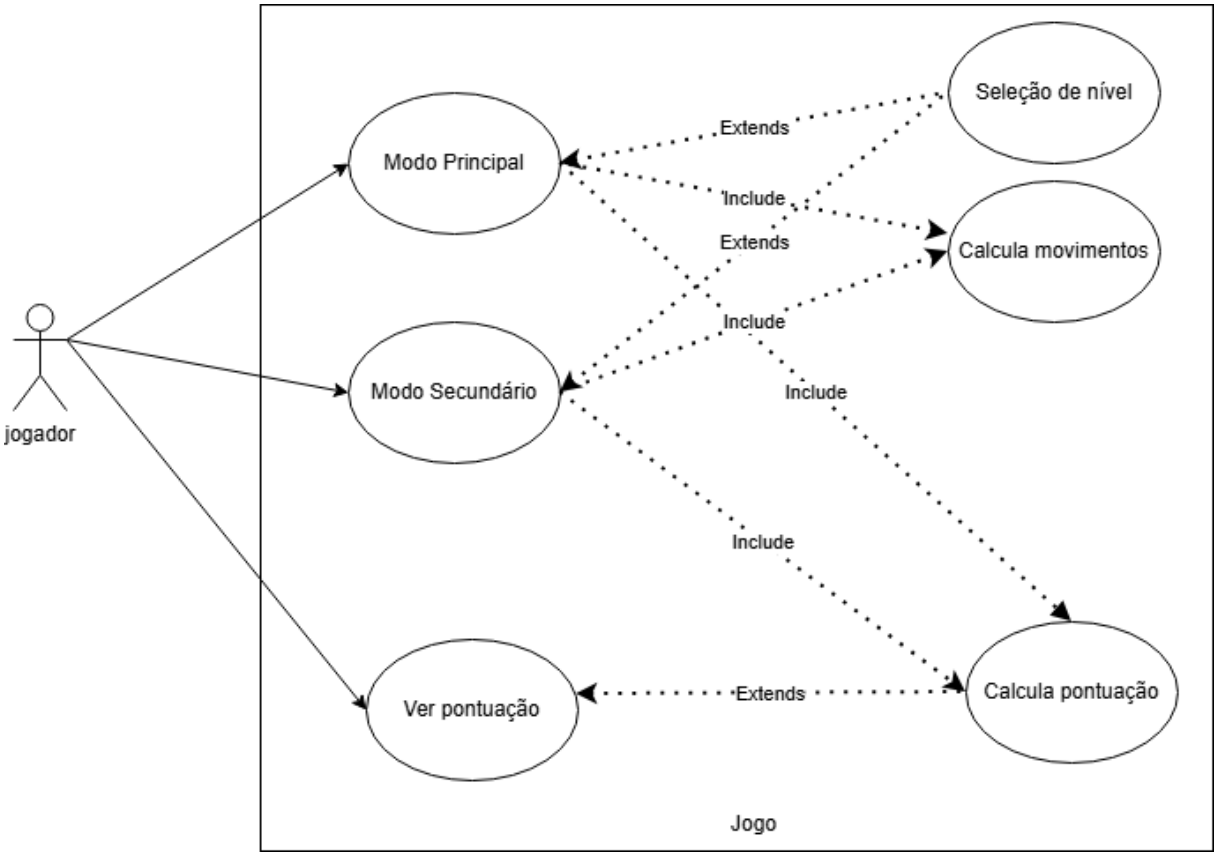

Figura 9 – Diagrama de caso de uso

Como descrito na Figura 9, o jogador consegue acessar 3 casos, porém estes possuem requisitos auxiliares para manter o funcionamento do sistema, por exemplo, não é possível exibir a pontuação sem antes calculá-la, e por sua vez, para o calculo da pontuação, é necessário terminar uma fase.

3.3.1.1 [RF001] Modo Principal

- ❑ **Descrição:** Permite ao jogador digitar um programa na pseudolinguagem, submetê-lo à compilação e interpretação e, em caso de sucesso, observar a animação do personagem executando os comandos de movimento dentro do labirinto. O propósito é traduzir o código do jogador em ações visíveis, possibilitando a resolução da fase.
- ❑ **Ator:** Jogador.
- ❑ **Prioridade:** Essencial.
- ❑ **Entradas e pré-condições:**
  - O jogo está em execução e uma fase encontra-se carregada e visível na tela.
  - O editor de código está acessível e vazio.
  - Entrada fornecida pelo ator: um texto contendo comandos válidos na pseudolinguagem (sequências de movimentos, estruturas de repetição, decisões e definições de funções).
- ❑ **Saídas e pós-condições:**
  - **Sucesso:** O personagem é animado deslocando-se no labirinto conforme os comandos interpretados. Se a posição final corresponder à saída, a fase é concluída e o sistema exibe uma mensagem de vitória, liberando a fase seguinte.
  - **Falha por erros de código:** Nenhum movimento é executado. Uma mensagem de erro (léxico, sintático ou semântico) é exibida, e o controle retorna ao editor para correção.
  - **Falha por não alcance da saída:** O personagem se move, mas não atinge a saída; o jogador pode editar o código e tentar novamente.
- ❑ **Fluxo de eventos principal:**
  1. O jogador digita o código na área de edição e executa o código.
  2. O sistema processa o arquivo:
     a) O lexer converte o texto em uma sequência de tokens;
     b) O parser valida a estrutura e constrói a AST;
     c) O interpretador percorre a AST e grava a lista ordenada de comandos de movimento (move_right, move_left, move_up, move_down) em uma lista.
  3. o Pygame inicia a animação: lê cada comando e desloca o personagem na tela na direção correspondente.

4. Se o personagem alcançar a coordenada da saída, o sistema pausa a animação, exibe uma mensagem de "Fase Concluída" e habilita o botão de avançar para a próxima fase.

❑ **Fluxos secundários (alternativos e de exceção):**

1. **Erro de compilação (léxico, sintático ou semântico):**
   a) Durante o processamento, o lexer, parser ou interpretador identifica um token inválido, uma construção gramatical incorreta ou um erro semântico (como uso de função inexistente).
   b) O sistema interrompe imediatamente a análise, descarta qualquer movimento parcial e exibe uma mensagem de erro (ex.: "Erro na linha 3: comando "move_let" não reconhecido").
   c) O controle retorna ao editor, onde o cursor pode ser posicionado na linha do erro. Nenhuma animação é realizada.
2. **Execução sem alcançar a saída:**
   a) A lista de movimentos é executada completamente, mas o personagem termina em uma posição diferente da saída.
   b) O sistema mantém o personagem na posição final e aguarda nova ação do jogador (editar código, executar novamente ou reiniciar a fase).

3.3.1.2 [RF002] Modo Secundário

❑ **Descrição:** O jogo fornece um programa escrito em pseudolinguagem. O jogador deve controlar o personagem utilizando as setas direcionais do teclado, executando exatamente os mesmos comandos descritos nesse código. Dessa forma, demonstra que domina a linguagem, mesmo sem tê-la escrito.

❑ **Ator:** Jogador

❑ **Prioridade:** Importante

❑ **Entradas e pré condições:**

- O jogo está em execução e uma fase encontra-se carregada e visível na tela.
- O código está fornecido pelo jogo para leitura do jogador
- O personagem ocupa a posição inicial da fase.
- Entrada Fornecida pelo Ator: movimentação usando as setas direcionais

❑ **Saídas e pós condições:**

- **Sucesso:** O jogador consegue com sucesso interpretar o código e reproduzir os comandos, chegando a saída, concluindo a fase e recebendo uma mensagem de vitória, podendo prosseguir para a próxima fase.
- **Falha por não alcance da saída:** O personagem se move, mas não atinge a saída; O jogador deve reiniciar a fase para tentar novamente.

❑ **Fluxo de eventos principal**

1. O jogador movimenta-se pelo labirinto.
2. O sistema verifica se é o movimento correto, se sim continua.
3. o Pygame inicia a animação: dado o comando desloca o personagem na tela na direção correspondente.
4. Se o personagem alcançar a coordenada da saída, o sistema pausa a animação, exibe uma mensagem de "Fase Concluída" e habilita o botão de avançar para a próxima fase.

❑ **Fluxos secundários**

1. **Execução sem alcançar a saída:**
   a) Durante a execução dos movimentos o jogador produz um comando não correspondente ao código
   b) O jogador reinicia a fase para tentar novamente.

3.3.1.3 [RF003] Ver Pontuação

❑ **Descrição:** Apresentar ao final de cada nível a pontuação do jogador, permitindo saber o quão bem ele foi, sendo também possível visualizar todas as maiores pontuações em um menu separado.

❑ **Ator:** Jogador

❑ **Prioridade:** Desejável

❑ **Entradas e pré condições:**

- O jogador concluiu alguma fase, tendo assim no mínimo uma pontuação para ser exibida

❑ **Saídas e pós condições:**

- **Sucesso:** O jogo consegue recuperar as pontuações e exibi-las na tela .
- **Falha por não possuir pontuações:** Por não haver nenhuma fase completa(ou falha na recuperação das pontuações) o jogo exibe a tela de pontuações com todos os campos com valor 0.

❑ **Fluxo de eventos principal**

1. O jogador faz a requisição
2. O sistema faz uma chamada ao caso de uso calcula pontuações.
3. Dada a resposta, o Pygame exibe na tela.

❑ **Fluxos secundários**

1. **Não possui pontuação:**
   a) O caso de uso calcula pontuação retorna Null.
   b) Exibe todas as pontuações como 0

3.3.1.4 [RF004] Seleção de Nível

❑ **Descrição:** Caso o jogador queira voltar em níveis que já foram conclúidos, um sistema de seleção de níveis é necessária, sendo utilizada para ambos os modos de jogo.

❑ **Ator:** NA

❑ **Prioridade:** Importante

❑ **Entradas e pré condições:**

- O jogo está em execução.
- Pelo menos um nível já foi concluído.

❑ **Saídas e pós condições:**

- **Sucesso:** O jogador consegue selecionar uma fase, carregando ela diretamente e voltando para o caso de uso modo principal ou modo secundário carregando o nível correspondente.
- **Falha por não ter concluído uma fase:** É necessário concluir uma fase para poder utilizar esta opção, senão será redirecionado para o primeiro nível

❑ **Fluxo de eventos principal**

1. O jogador escolhe um dos dois modos.
2. Uma lista das fases é exibida.
3. Um dos níveis é escolhido e carregado.

❑ **Fluxos secundários**

1. **Nível bloqueado:**
   a) Tenta-se escolher um nível no qual o jogador não desbloqueou ainda
   b) O sistema não permite a escolha do nível e aguarda uma escolha adequada.

3.3.1.5 [RF005] Calcula Pontuação

- **Descrição:** No final de uma fase a pontuação é calculada automaticamente e salva num arquivo local txt, para assim ser utilizado na ”visualização de pontos”.
- **Ator:** NA
- **Prioridade:** Desejável
- **Entradas e pré condições:**
  - Um nível acaba de ser concluído
- **Saídas e pós condições:**
  - **Sucesso:** O sistema consegue calcular a pontuação e salvar no arquivo.
  - **Falha por cálculo:** O sistema não consegue calcular a pontuação, logo não é possível salvar o progresso.
  - **Falha por Salvamento:** O sistema não consegue salvar a pontuação, da mesma forma não conseguindo salvar o progresso.
- **Fluxo de eventos principal**
  1. Uma fase é concluída.
  2. Enumera a quantidade de comandos utilizados para concluir a fase(modo principal), ou enumera o tempo necessário para conclusão (modo secundário).
  3. Calcula a pontuação baseado nos dados coletados,
  4. Salva o número junto com ID da fase correspondente, caso já exista, a maior pontuação é priorizada.
- **Fluxos secundários**
  1. **Não é possível resgatar número de comandos ou tempo:**
     a) Erro durante a coleta de informação.
     b) Notifica o jogador que não é possível salvar a pontuação.
  2. **Não é possível salvar:**
     a) Não é possível escrever no arquivo txt (ou não é possível achá-lo).
     b) Notifica o jogador que não é possível salvar a pontuação.+

3.3.1.6 [RF006] Calcula Movimentos

- **Descrição:** O sistema deve permitir que o jogador mova o personagem pelo mapa de duas formas, dependendo do modo de jogo. No modo Principal, ao pressionar a tecla F5, o código digitado no editor é compilado e os comandos de movimento gerados são executados sequencialmente, com animação de deslizamento. No modo Secundário, o jogador move o personagem diretamente com as setas do teclado (cima, baixo, esquerda, direita), um movimento por tecla.
- **Ator:** NA
- **Prioridade:** Essencial
- **Entradas e pré-condições:**
  - Modo Principal: o código foi escrito no editor e não há erro de compilação; o mapa está carregado e o personagem posicionado.
  - Modo Secundário: o mapa está carregado, o personagem posicionado e o tutorial já foi concluído.
- **Saídas e pós-condições:**
  - **Sucesso:** O personagem desloca-se para a direção desejada, desde que o caminho esteja livre. O tilemap é atualizado com a nova posição.
  - **Falha (caminho bloqueado):** O personagem permanece na posição atual; nenhuma animação é executada.
  - **Falha (erro de sintaxe ou limite excedido):** No modo Principal, se o código contiver erro ou exceder o limite de comandos/passos, uma mensagem de erro é exibida no lugar da referência, e o personagem não se move.
- **Fluxo de eventos principal (modo Principal)**
  1. O jogador escreve um código no editor e pressiona F5.
  2. O sistema compila o código.
  3. Para cada comando da lista:
     a) O sistema verifica se o destino não é uma parede (B).
     b) Se o caminho estiver livre, o sistema executa uma animação de deslizamento (slide_to) do personagem da posição atual até a nova posição.
     c) Atualiza a posição lógica do jogador no tilemap (remove P da posição anterior, coloca P na nova).
  4. Ao final de todos os comandos, se a nova posição for o checkpoint (C), o sistema registra a vitória e calcula a pontuação.

❑ **Fluxo de eventos principal (modo Secundário)**

1. O jogador pressiona uma das setas direcionais (cima, baixo, esquerda, direita).
2. O sistema verifica se o destino não é uma parede (B).
3. Se o caminho estiver livre, o sistema executa a animação de deslizamento e atualiza a posição lógica.
4. Se o movimento corresponder ao comando esperado na sequência guia, o sistema avança para o próximo comando da sequência.
5. Ao reproduzir o código fornecido, o nível acaba com uma tela de vitória.

❑ **Fluxos secundários**

1. **Caminho bloqueado (ambos os modos):**
   a) O sistema detecta que o destino é B.
   b) O personagem não se move; o estado do jogo permanece inalterado.
2. **Erro de compilação ou loop infinito (modo Principal):**
   a) O compilador retorna um erro.
   b) O sistema exibe a mensagem de erro na área de diálogo e interrompe a execução. O personagem não se move.
3. **Movimento incorreto (modo Secundário):**
   a) O jogador pressiona uma seta que não corresponde ao próximo comando da sequência guia.
   b) O jogador precisa pressionar F7 para reiniciar a fase e tentar novamente.

3.4 Requisitos Não Funcionais - RNF

A partir dos requisitos funcionais descritos, definem-se os não funcionais. Para oferecer uma boa experiência ao usuário, o jogo precisa ser, acima de tudo, responsivo. Desempenho e confiabilidade tornam-se, portanto, requisitos essenciais, juntamente com a usabilidade, garantindo que o jogador não se sinta perdido nem desengajado durante uma fase.

3.4.1 Usabilidade

3.4.1.1 [RNF001] Tutorial Eficiente

❑ **Descrição:** Para que o jogador continue querendo progredir nas fases, é essencial que os tutoriais sejam claros, permitindo-o ter todas as ferramentas em sua disposição para aquele determinado nível.

- ❏ **Prioridade:** Importante.
- ❏ **Caso(s) de uso associado(s):**Modo Principal e Modo Secundário.

3.4.1.2 [RNF002] Dificuldade Progressiva

- ❏ **Descrição:** Para que o jogador não se sinta desestimulado, é preciso implementar níveis. Cada um, comparado ao imediatamente anterior, não deve ser nem muito fácil nem muito difícil, criando-se, assim, uma curva de aprendizado suave e eficiente.
- ❏ **Prioridade:** Importante.
- ❏ **Caso(s) de uso associado(s):**Modo Principal e Modo Secundário.

3.4.2 Confiabilidade

3.4.2.1 [RNF001] Disponibilidade da Pontuação

- ❏ **Descrição:** Para não desestimular o usuário ao término de uma fase, o sistema de pontuação deve ser o mais à prova de falhas possível.
- ❏ **Prioridade:** Desejável.
- ❏ **Caso(s) de uso associado(s):** Calculo de Pontuação

3.4.3 Desempenho

3.4.3.1 [RNF001] Tempo de Resposta

- ❏ **Descrição:** Tanto na execução de comandos quanto no término de um nível(exibição de pontuação) o sistema não pode demorar mais do 1 segundo para responder, mantendo a jogabilidade fluida.
- ❏ **Prioridade:** Essencial
- ❏ **Caso(s) de uso associado(s):** Modo Principal, Modo Secundário, Ver Pontuação.

3.4.3.2 [RNF002] Taxa de Quadros

- ❏ **Descrição:** Da mesma forma, o jogo durante sua execução não pode alcançar uma taxa de quadros por segundo inferior a 30, o mínimo necessário para manter a fluidez.
- ❏ **Prioridade:** Essencial
- ❏ **Caso(s) de uso associado(s):** Modo Principal, Modo Secundário, Ver Pontuação.

3.4.3.3 [RNF003] Carregamento de Nível

- ❑ **Descrição:** Como o carregamento de uma fase exige maior poder computacional, não estará limitado ao tempo de resposta de 1 segundo definido anteriormente. Ainda assim, precisa ser rápido, alcançando no máximo 3 segundos.
- ❑ **Prioridade:** Importante
- ❑ **Caso(s) de uso associado(s):** Modo Principal, Modo Secundário.

3.5 Considerações Sobre o Capítulo

Com os requisitos devidamente documentados, o próximo capítulo dará continuidade ao processo de engenharia de software, focando na análise e projeto do sistema, descrevendo como foram implementadas as restrições e funcionalidades definidas neste capítulo.

Capítulo 4

Análise e Projeto do Sistema

Tendo os requisitos estabelecidos, é preciso definir o projeto concreto do sistema. Este capítulo dedica-se, portanto, à análise e ao projeto da solução, com os requisitos levantados e validados nos requisitos. Para isso, adota-se uma visão geral da arquitetura e realiza-se a modelagem da solução por meio de diagramas, desenvolvendo desde modelos conceituais iniciais, como o Modelo de Domínio, até representações mais próximas do software final, a exemplo do Diagrama de Classes.

4.1 Modelo de Domínio do Sistema

O sistema é estruturado em torno da classe central Game, que mantém o estado ativo do programa, modo escolhido (Principal ou Secundário), nível, mapa, personagem e diálogo. No modo Principal, o jogador escreve pseudocódigo em um editor de texto integrado e, ao executá-lo, um pipeline de compilação (Lexer, Parser e Interpreter) transforma o código em uma sequência de movimentos que o personagem realiza no mapa; já no modo Secundário, o jogador controla o personagem diretamente com as setas, enquanto o editor exibe um código-guia não editável. Ao alcançar o checkpoint, a pontuação é calculada e persistida em arquivos de texto, atualizando os recordes quando superados. Desta forma, Game centraliza a execução, delegando tarefas específicas a módulos como o editor de texto, o sistema de animação, o compilador e o gerenciador de pontuações, como pode ser visto na Figura 10 abaixo.

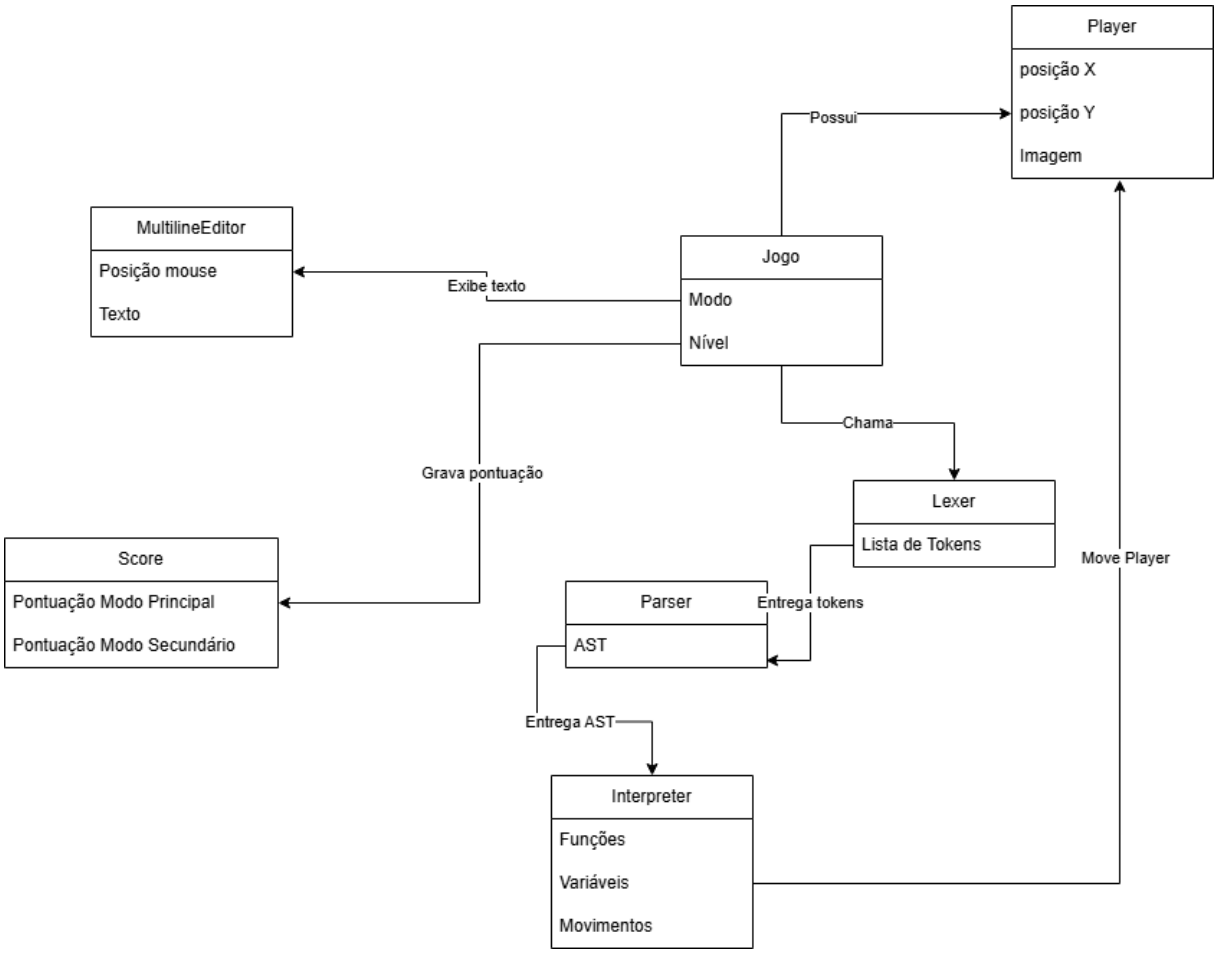

Figura 10 – Modelo Conceitual do Sistema.

4.2 Diagrama de Classe do Sistema

Com base no modelo conceitual discutido anteriormente, foi possível elaborar um diagrama de classes completo (Figura 11). Observando a nova representação, nota-se a inclusão de classes voltadas à interface do jogo. Essa camada de menu é responsável por conduzir o jogador até a classe principal Game, onde ele pode selecionar um nível e o modo que será executado(principal ou secundário), ou até a tela de Pontuação, que exibe os recordes alcançados.

Além do jogador, o cenário do jogo exigiu a modelagem de outros elementos concretos, como o chão, as paredes do labirinto e o ponto de chegada. Todos eles necessários para compor visualmente cada fase e estabelecer as regras de movimentação e vitória.

Apesar dessas adições, a estrutura central do sistema permanece praticamente a mesma: Game continua sendo a classe principal que utiliza as classes auxiliares, e o processamento do código escrito pelo usuário ainda é confiado a um pipeline, formado pelo Lexer, pelo Parser e pelo Interpreter, que transforma instruções textuais em comandos dentro do jogo.

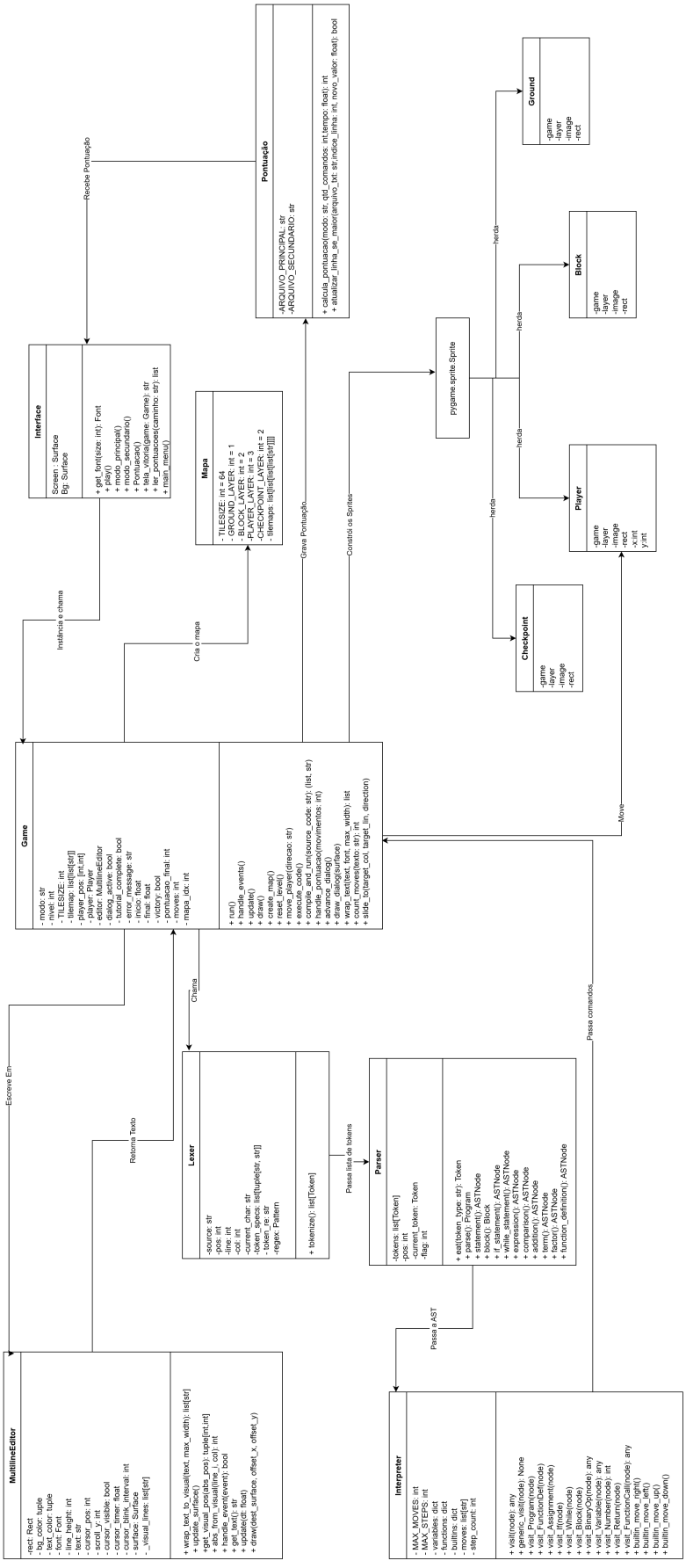

Figura 11 – Diagrama de Classes.

4.2.1 Considerações do Capítulo

Neste capítulo, foram apresentados os modelos conceituais e estruturais que fundamentam a arquitetura do sistema. O Modelo de Domínio destacou a classe Game como núcleo. Essa visão inicial foi refinada por meio do Diagrama de Classes, que acrescentou a camada de interface (telas de menu e pontuação) e os elementos concretos do cenário (chão, paredes, ponto de chegada).

Com a base conceitual e estrutural estabelecida, o próximo capítulo abordará o desenvolvimento do sistema, detalhando as tecnologias utilizadas e as decisões de implementação. Os diagramas serviram como guias para a construção prática da solução.

Capítulo 5

Desenvolvimento do Sistema

5.1 Processo de Desenvolvimento

O desenvolvimento do sistema foi conduzido por meio de um processo iterativo, porém com requisitos que evoluíram à medida que a implementação avançava. Por se tratar de um trabalho realizado por um único desenvolvedor, adotou-se uma abordagem prática que combinou elementos de prototipação e ciclos curtos de codificação e testes

5.1.1 Interpretador

A primeira etapa, executada pelo Lexer, transforma a sequência de caracteres do código-fonte em uma lista de tokens. Utilizando expressões regulares compiladas, como pode-se ver na Figura 12 o analisador léxico percorre o texto e identifica números, identificadores (nomes de variáveis e funções), operadores aritméticos e relacionais, delimitadores (parênteses, chaves, ponto-e-vírgula) e palavras reservadas como if, else, while, def e return. Espaços em branco e quebras de linha são descartados, mas utilizados para rastrear o número da linha e da coluna de cada token, informação necessária para a geração de mensagens de erro. Caso um caractere não corresponda a nenhum padrão esperado, o Lexer interrompe a análise e sinaliza o erro, impedindo que o código inválido prossiga para as fases seguintes.

```
f tokenize(self):
  tokens = []
  for mo in self.regex.finditer(self.source):
      #tipo do token, EQ NE etc
      kind = mo.lastgroup
      #valor especifico, como x,5
      value = mo.group()
      #pula espaco em branco e avanca 1 coluna
      if kind == 'SKIP':
          self.col += len(value)
          continue
      #pula \n e avanca 1 linha e posiciona a coluna no 1
      elif kind == 'NEWLINE':
          self.line += 1
          self.col = 1
          continue
      #palavras reservadas serao tratadas como seu proprio token em maisculo
      elif kind == 'IDENT' and value in ('if', 'else', 'while','def', 'return'):
          kind = value.upper()
      elif kind == 'MISMATCH':
          raise Exception(f"Erro: Caractere inesperado '{value}' na linha {self.line}, coluna {self.col}")
      # atribui o valor dos tokens
      token = Token(kind, value, self.line, self.col)
      tokens.append(token)
      self.col += len(value)
  #no final adiciona EOF
  tokens.append(Token('EOF', '', self.line, self.col))
  return tokens
```

Figura 12 – Função de Tokenização.

Em seguida, o Parser consome a lista de tokens e verifica se a sequência obedece à gramática da linguagem, construindo simultaneamente uma AST. A análise é realizada por descida recursiva, técnica em que cada regra gramatical é implementada como um método da classe Parser. A hierarquia de precedência de operadores é garantida pela organização em cascata dos métodos. Comandos como atribuições, condicionais e laços são tratados em métodos específicos, que montam os nós correspondentes da AST. Erros sintáticos, como a ausência de um ponto-e-vírgula ou de um fecha-chaves, são detectados pelo método eat, que consome um token esperado ou sinaliza a falha, permitindo que o jogo exiba uma mensagem ao usuário.

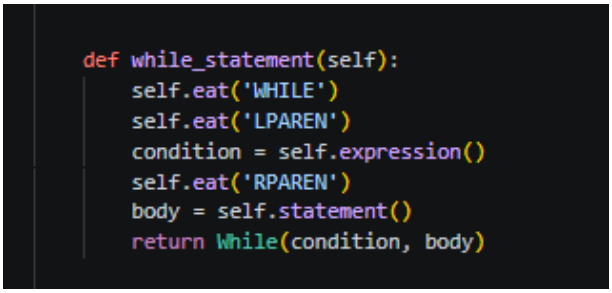

Figura 13 – Estrutura while no Parser.

No exemplo da Figura 13, o tratamento da estrutura while mostra a lógica do Parser. A gramática da pseudo-linguagem estabelece que o laço de repetição deve obedecer ao formato: ”while (expressão) comando”. Para reconhecer essa construção, o parser usa o método while_statement().

Inicialmente, o método invoca a função eat para consumir o token correspondente à palavra reservada WHILE. Em seguida, consome o parêntese de abertura LPAREN, que delimita o início da condição do laço. Nesse ponto, o controle é transferido ao método expression(), responsável por analisar a expressão booleana que determinará se o corpo do laço será executado. Permitindo que condições complexas, com operadores aritméticos e relacionais combinados, sejam corretamente interpretadas.

Após o processamento da condição, o parser consome o parêntese de fechamento RPAREN e passa a tratar o corpo do laço. Para isso, chama o método statement(), que pode retornar tanto um bloco delimitado por chaves, contendo múltiplos comandos. Assim a linguagem aceitaria por exemplo: ”while (i < 3) { move_up(); i=i+1; }”.

Por fim, o Interpreter percorre a AST e executa as ações correspondentes a cada nó. Para cada classe de nó (por exemplo, While, Assignment, FunctionCall), há um método "visit"que implementa a semântica daquela construção, como na Figura 14 em que o visit_while irá executar o body do while enquanto sua condição for verdadeira. Além disto o interpretador mantém um dicionário de variáveis e um dicionário de funções definidas pelo usuário, além de um conjunto de funções embutidas (move_up, move_down, move_left, move_right) que, quando chamadas, registram os comandos de movimento em uma lista.

Para evitar que loops infinitos travem o jogo, foram estabelecidos limites de segurança: um número máximo de iterações (MAX_STEPS) para o laço while demonstrado na Figura 14 e um número máximo de comandos de movimento (MAX_MOVES). Ao exceder esses limites, o interpretador lança uma exceção, que é capturada pelo jogo e exibida como mensagem de erro.

```
def visit_While(self, node):
    while self.visit(node.condition):
        self.step_count += 1
        if self.step_count > self.MAX_STEPS:
            raise Exception("Limite de operações excedido. Possível loop infinito.")
        self.visit(node.body)
```

Figura 14 – Estrutura While no Interpretador.

5.1.2 Mapa

O ambiente visual do jogo é construído sobre uma representação textual do mapa, que consiste em uma matriz de caracteres na qual cada célula corresponde a um tipo de elemento do cenário (Figura 15 e Figura 17). A matriz é armazenada como uma lista de strings, em que o caractere 'B' indica uma parede (bloco intransponível), 'P' marca a posição inicial do personagem, 'C' sinaliza o ponto de chegada (checkpoint) e '.' representa o piso livre por onde o jogador pode se deslocar.

Para conferir variedade e rejogabilidade, cada um dos cinco níveis do jogo possui cinco variações de mapa, totalizando vinte e cinco configurações possíveis. Essas variações são agrupadas em uma lista de strings tilemaps[nivel][variacao], ao iniciar uma fase, um índice aleatório (self.mapa_idx) é sorteado, determinando qual das cinco variantes será utilizada naquela partida.

A construção visual do cenário ocorre no método create_map (Figura 16), que percorre cada célula da matriz selecionada. Para toda posição (i, j), independentemente do caractere, é instanciado um objeto Ground, que representa o piso e é adicionado aos grupos all_sprites e grounds. Em seguida, conforme o caractere encontrado, são criados os elementos específicos: se 'B', instancia-se um Block (parede), adicionado a blocks e all_sprites; se 'P', cria-se o objeto Player e registra-se sua posição lógica em player_pos; se 'C', gera-se um CheckPoint, inserido no grupo check_points. Dessa forma, a matriz de caracteres é integralmente traduzida em uma hierarquia de sprites prontos para exibição e interação.

Figura 15 – Exemplo de Mapa.

```
def create_map(self):
    self.inicio = time.time()
    if hasattr(self, 'player') and self.player:
        self.player.kill()
        self.player = None
    self.grounds.empty()
    self.blocks.empty()
    self.check_points.empty()
    self.virtual_screen.fill("black")

    for i, row in enumerate(self.tilemap):
        for j, column in enumerate(row):
            ground = Ground(self, j, i)
            self.grounds.add(ground)
            if column == 'B':
                block = Block(self, j, i)
                self.blocks.add(block)
            if column == 'P':
                self.player = Player(self, j, i)
                self.player_pos = [i, j]
            if column == 'C':
                check_point = CheckPoint(self, j, i)
                self.check_points.add(check_point)
```

Figura 16 – Função de criação de mapa.

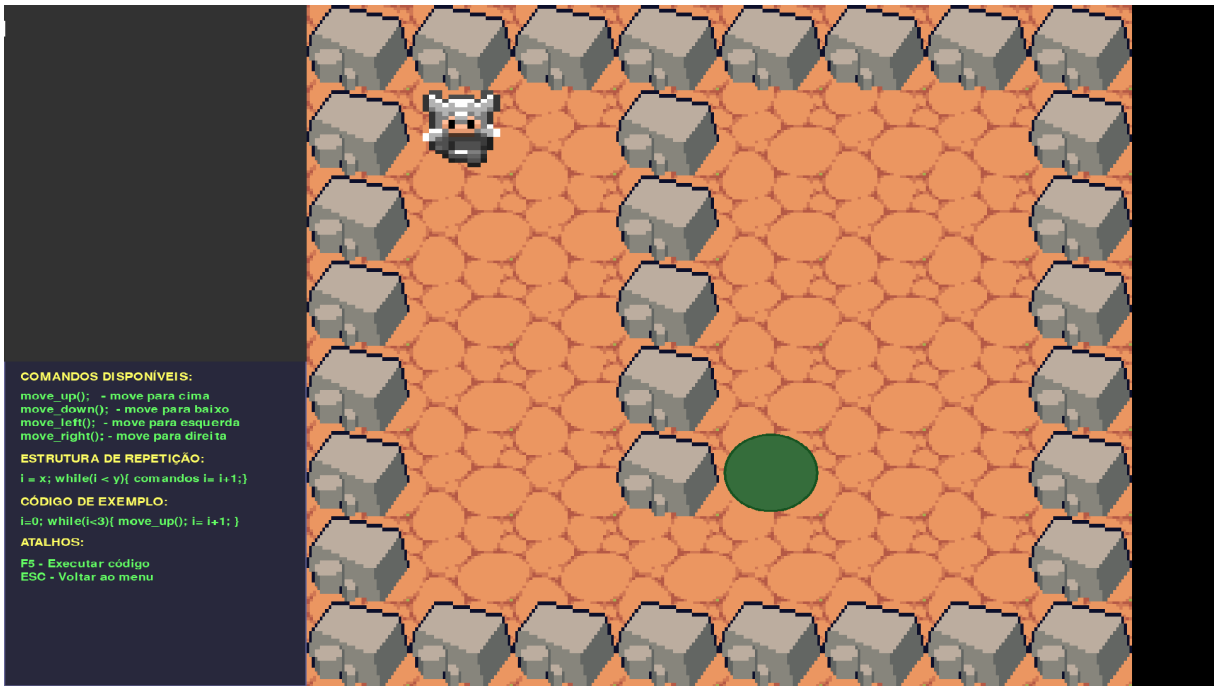

Figura 17 – Mapa criado pela Matriz da Figura 15.

Cada sprite é posicionado na tela multiplicando-se suas coordenadas lógicas (x, y) pelo tamanho do tile (TILESIZE), garantindo que todos os elementos se alinhem. As camadas de renderização são controladas pelo atributo Layer: o piso (GROUND_LAYER) é desenhado primeiro; em seguida, os blocos e o checkpoint (BLOCK_LAYER e CHECKPOINT_LAYER) aparecem sobre o piso; e, por fim, o personagem (PLAYER_LAYER) é exibido no topo, assegurando a correta sobreposição visual.

A movimentação do personagem é guiada pela matriz de caracteres. Antes de qualquer deslocamento, o método move_player (Figura 18) consulta a célula de destino: se ela contiver 'B', o movimento é bloqueado e o personagem permanece na posição atual. Caso contrário, a célula de origem é alterada para '.' e a célula de destino passa a conter 'P', refletindo a nova posição lógica. Essa atualização da matriz permite que as verificações de colisão e vitória, quando a posição coincide com 'C', sejam realizadas.

A transição entre as posições é acompanhada por uma animação de deslizamento implementada no método slide_to. Utilizando uma folha de sprites (Spritesheet), o sistema extrai quadros específicos para cada direção (cima, baixo, esquerda, direita), formando uma sequência de animação. Durante um intervalo de tempo definido por move_duration, a posição do personagem é interpolada linearmente entre a origem e o destino, enquanto os quadros são trocados periodicamente.

```
def move_player(self, direcao):
    col, lin = self.player_pos[0], self.player_pos[1]
    target_col, target_lin = col, lin
    if direcao == "direita":
        target_lin = lin + 1
    elif direcao == "esquerda":
        target_lin = lin - 1
    elif direcao == "cima":
        target_col = col - 1
    elif direcao == "baixo":
        target_col = col + 1
    else:
        return False  # direção inválida

    # Verifica se o destino é um B
    if self.tilemap[target_col][target_lin] == 'B':
        return False
    # Atualiza o tilemap
    self.tilemap[col][lin] = '.'
    self.tilemap[target_col][target_lin] = 'P'
    self.player_pos = [target_col, target_lin]

    # Executa animação
    self.is_sliding = True
    self.slide_to(target_col, target_lin, direcao)
    self.is_sliding = False

    return True
```

Figura 18 – Função de mover o jogador.

5.1.3 Editor de Texto

A interação do jogador com o código-fonte do jogo é mediada por um editor de texto próprio, pois a biblioteca Pygame não oferece um campo de texto multilinha adequado para a edição de pseudocódigo,o componente gerencia apenas uma linha e não dispõe de recursos como quebra automática ou navegação por setas. Diante dessa limitação, implementou-se uma classe dedicada que assumisse tanto a captura das teclas digitadas quanto a exibição formatada do código, atendendo às necessidades de um ambiente de programação integrado ao jogo.

O MultilineEditor foi concebido como uma superfície retangular que mantém internamente uma string completa, com quebras de linha representadas pelo caractere `\n`. Um algoritmo de quebra automática percorre as palavras e, sempre que a adição de uma nova palavra excederia o limite horizontal, inicia uma nova linha visual (Figura 19). Essas linhas são armazenadas em visual_lines e redesenhadas sempre que o conteúdo é alterado. Para permitir a edição, o editor processa eventos de teclado como inserção de caracteres, remoção com Backspace e Delete, navegação com as setas direcionais e posicionamento por clique do mouse.

```python
def _wrap_text_to_visual(self, text, max_width):
    #Quebra o texto em linhas visuais conforme a largura
    paragraphs = text.split('\n')
    visual = []
    for para in paragraphs:
        if para == "":
            visual.append("")
            continue
        words = para.split(' ')
        cur_line = ""
        for w in words:
            while self.font.size(w)[0] > max_width:
                part = w
                while self.font.size(part)[0] > max_width and len(part) > 1:
                    part = part[:-1]
                if part == "":
                    part = w[0]
                visual.append(cur_line + part if cur_line else part)
                w = w[len(part):]
                cur_line = ""
                if not w:
                    break
            if not w:
                continue
            trial = cur_line + (" " if cur_line else "") + w
            if self.font.size(trial)[0] > max_width:
                visual.append(cur_line)
                cur_line = w
            else:
                cur_line = trial
        if cur_line:
            visual.append(cur_line)
    return visual
```

Figura 19 – Função para exibição visual do código.

No modo Principal, o editor é o principal meio de interação do jogador com o jogo. Após a conclusão do tutorial, o editor é liberado para digitação, e o jogador pode escrever seu pseudocódigo utilizando a sintaxe da pseudo-linguagem (Figura 20). Quando o jogador pressiona a tecla F5, o texto completo é extraído do editor por meio do método get_text() e encaminhado ao pipeline de interpretação. Se o código for válido, a lista de movimentos resultante é executada sequencialmente, e o personagem se desloca pelo mapa com animações. Caso contrário, o erro é imediatamente reportado ao jogador.

Quando o interpretador retorna um erro, a mensagem é armazenada no atributo error_message da classe Game. Na área de diálogo, localizada na metade inferior do painel esquerdo, uma caixa vermelha substitui temporariamente o texto de referência e exibe a descrição do problema. Após alguns segundos, a mensagem desaparece automaticamente, e o texto de referência retorna, permitindo que o jogador corrija o código e tente novamente.

O sistema de tutorial, por sua vez, utiliza a mesma área de diálogo para guiar o jogador nos primeiros momentos da fase. Durante o tutorial, um personagem animado surge ao lado de um balão de fala, e mensagens explicativas são exibidas sequencialmente. O jogador avança as mensagens pressionando ENTER, até que o diálogo se encerre e a área passe a mostrar um texto de referência contendo os comandos disponíveis e as teclas de atalho (Figura 21).

No modo Secundário, o editor assume uma função diferente. Em vez de permitir a digitação livre, ele é preenchido automaticamente com um código de exemplo, um dos cinco códigos predefinidos para o nível. Esse código serve como modelo que o jogador deve seguir, mas a interação não se dá pela escrita: o jogador movimenta o personagem diretamente com as setas do teclado. A cada tecla pressionada, o sistema compara a direção escolhida com o próximo comando da sequência esperada, extraída de COMANDOS_PARA_OS_CODIGOS. Se o movimento corresponder, a sequência avança. Ao completar todos os movimentos corretamente, a vitória é registrada. Dessa forma, o editor atua como um expositor de código, permitindo que jogadores sem experiência prévia com sintaxe de programação observem a estrutura de um programa funcional e a relacionem com as ações executadas no jogo.

Em resumo, o MultilineEditor viabiliza a escrita e a execução de código no modo Principal, oferece resposta para erros, conduz o jogador por meio de um tutorial e no modo Secundário, exibe o código de referência.

Figura 20 – Exemplo de Código escrito no editor.

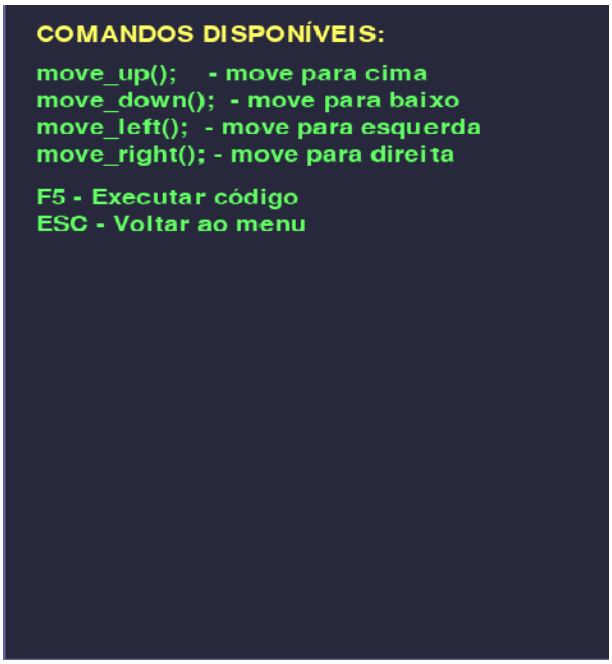

Figura 21 – Exemplo de comandos de referência.

5.1.4 Pontuação

A pontuação é calculada automaticamente ao final de cada fase bem-sucedida, atribuindo ao jogador um valor numérico entre 0 e 2000 que reflete seu desempenho na resolução do problema proposto, caso a fase não seja concluída (jogador sai da fase por exemplo) é atribuída a pontuação padrão 0.

Pode-se ver na Figura 22 o cálculo da pontuação difere conforme o modo de jogo. No modo Principal, a métrica utilizada é a quantidade de comandos de movimento escritos no editor de texto, capturado pela função count_moves. A fórmula 2000 - (quantidade_de_comandos * 100) recompensa a economia de comandos de movimentação, incentivando o jogador a buscar soluções mais eficientes. Já no modo Secundário, em que o jogador controla o personagem diretamente com as setas, a pontuação é baseada no tempo decorrido desde o início da fase até a chegada ao ponto de verificação, obtido pela diferença entre time.time() e o instante de início da fase, registrado em self.inicio durante a criação do mapa. A expressão 2000 - (tempo_em_segundos * 10) valoriza a agilidade na execução da sequência de movimentos, estimulando o raciocínio rápido e a compreensão dos comandos.

Também na Figura 22 se vê que valores calculados são persistidos em arquivos de texto simples, pontuacao_Principal.txt e pontuacao_Secundaria.txt, localizados no diretório do próprio projeto. Cada arquivo armazena cinco linhas, correspondentes aos cinco níveis do jogo, e cada linha contém a maior pontuação já obtida naquele nível. O módulo pontuacao.py implementa a função atualizar_linha_se_maior, que realiza a atualização condicional: o novo valor só substitui o anterior se for estritamente maior, preservando o recorde do jogador. Caso o arquivo ainda não exista,situação da primeira execução do jogo, o sistema o cria automaticamente com valores iniciais iguais a zero.

A exibição das pontuações ocorre em uma tela específica do menu principal, acessível pelo botão "PONTUAÇÕES". A função ler_pontuacoes recupera os dados de ambos os arquivos e os organiza em uma lista para cada modo. A interface apresenta os resultados em duas colunas, com os cinco níveis dispostos verticalmente. Cada célula exibe o número da fase e sua respectiva pontuação recorde, formatada como valor inteiro.

```python
def calcula_pontuacao(modo,qtd_comandos,tempo):
    if modo =="Principal":
        pontuacao = 2000 - (qtd_comandos *100)
        return pontuacao
    else:
        pontuacao = int(2000 - (tempo*10))
        return pontuacao

def atualizar_linha_se_maior(arquivo_txt, indice_linha, novo_valor):
    # Se o arquivo não existe, cria com zeros para 5 níveis
    if not os.path.exists(arquivo_txt):
        os.makedirs(os.path.dirname(arquivo_txt), exist_ok=True)  # garante que o diretório exista
        with open(arquivo_txt, 'w', encoding='utf-8') as f:
            for _ in range(5):      # 5 níveis
                f.write("0\n")

    # Lê todas as linhas
    with open(arquivo_txt, 'r', encoding='utf-8') as f:
        linhas = f.readlines()

    # Verifica se a linha existe
    if indice_linha < 0 or indice_linha >= len(linhas):
        print("Índice inválido")
        return False

    # Obtém o valor atual da linha
    try:
        valor_atual = float(linhas[indice_linha].strip())
    except ValueError:
        print("A linha não contém um número válido")
        return False

    # Compara e substitui se o novo valor for maior
    if novo_valor > valor_atual:
        linhas[indice_linha] = f"{novo_valor}\n"
        with open(arquivo_txt, 'w', encoding='utf-8') as f:
            f.writelines(linhas)
        return True
    return False
```

Figura 22 – Cálculo e atualização da pontuação.

5.1.5 Interface

A interface foi estruturada em um conjunto de telas interdependentes, gerenciadas pelo módulo Interface/main.py, que gerencia a navegação entre o menu principal, as telas de seleção de modo e nível, a tela de jogo, a tela de pontuações e a tela de vitória. Cada uma dessas telas foi implementada como uma função independente, que utiliza os recursos de renderização e captura de eventos do Pygame para construir a interação.

A tela de menu principal (Figura 23) constitui o ponto de entrada do sistema. Sobre um fundo redimensionado para preencher toda a resolução do monitor, são dispostos três botões, "JOGAR", "PONTUAÇÕES" e "SAIR", centralizados horizontalmente. As imagens dos botões são carregadas e escaladas proporcionalmente, garantindo que, independentemente da resolução do dispositivo, os elementos mantenham suas proporções e permaneçam acessíveis. A partir dessa tela, o jogador pode iniciar uma partida, consultar seus recordes ou encerrar a aplicação.

Ao selecionar "JOGAR", o usuário é conduzido a uma tela de escolha de modo, onde opta entre as modalidades Principal e Secundário. Em ambos os casos, a progressão é direcionada para uma tela de seleção de nível, que exibe cinco botões correspondentes às fases disponíveis. Os níveis ainda não desbloqueados são exibidos em cinza e não respondem a cliques, enquanto os níveis já liberados aparecem em branco. O estado de bloqueio é determinado pela leitura dos arquivos de progresso, atualizados automaticamente a cada vitória.

A tela de jogo representa a porção mais complexa da interface e é controlada pela classe Game que integra as 3 porções do desenvolvimento descritas anteriormente, o Mapa, o Editor de Texto e o Interpretador.

A tela de pontuação também descrita anteriormente apresenta os recordes salvos de ambos os modos de jogo. Duas colunas, uma para o modo Principal e outra para o modo Secundário

Por fim, a tela de vitória é exibida automaticamente ao concluir uma fase com sucesso. Além de apresentar a pontuação obtida, ela oferece dois botões: "MENU", que retorna ao menu principal, e "PRÓXIMO NÍVEL", que avança diretamente para a fase seguinte (desde que haja uma).

```python
def main_menu():
    while True:
        SCREEN.fill("black")
        SCREEN.blit(BG, (0, 0))
        MENU_MOUSE_POS = pygame.mouse.get_pos()

        MENU_TEXT = get_font(100).render("MAIN MENU", True, "#b68f40")
        MENU_RECT = MENU_TEXT.get_rect(center=(NATIVA_LARGURA // 2, escala_y(100)))
        SCREEN.blit(MENU_TEXT, MENU_RECT)

        # Botões com imagens escaladas
        play_img = pygame.image.load("Interface/assets/Play Rect.png")
        play_img = pygame.transform.smoothscale(play_img, (escala_x(380), escala_y(100)))
        PLAY_BUTTON = Button(image=play_img,
                             pos=(NATIVA_LARGURA // 2, escala_y(250)), text_input="JOGAR", font=get_font(75),
                             base_color="#d7fcd4", hovering_color="White")

        pontuacao_img = pygame.image.load("Interface/assets/Pontuacao Rect.png")
        pontuacao_img = pygame.transform.smoothscale(pontuacao_img, (escala_x(760), escala_y(100)))
        PONTUACAO_BUTTON = Button(image=pontuacao_img, pos=(NATIVA_LARGURA // 2, escala_y(400)),
                                  text_input="PONTUAÇÕES", font=get_font(75),
                                  base_color="#d7fcd4", hovering_color="White")

        quit_img = pygame.image.load("Interface/assets/Quit Rect.png")
        quit_img = pygame.transform.smoothscale(quit_img, (escala_x(380), escala_y(100)))
        QUIT_BUTTON = Button(image=quit_img,
                             pos=(NATIVA_LARGURA // 2, escala_y(550)), text_input="SAIR", font=get_font(75),
                             base_color="#d7fcd4", hovering_color="White")

        for button in [PLAY_BUTTON, PONTUACAO_BUTTON, QUIT_BUTTON]:
            button.changeColor(MENU_MOUSE_POS)
            button.update(SCREEN)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if PLAY_BUTTON.checkForInput(MENU_MOUSE_POS):
                    play()
                if PONTUACAO_BUTTON.checkForInput(MENU_MOUSE_POS):
                    Pontuacao()
                if QUIT_BUTTON.checkForInput(MENU_MOUSE_POS):
                    pygame.quit()
                    sys.exit()

        pygame.display.update()
```

Figura 23 – Código Menu Principal.

Figura 24 – Interface Menu Principal.

5.2 Arquitetura do Sistema

O sistema foi concebido como uma aplicação autossuficiente, executada integralmente no computador do usuário, sem dependência de servidores remotos, comunicação em rede ou banco de dados externos. Essa escolha deve-se à natureza do jogo que não requer funcionalidades colaborativas ou acesso simultâneo de múltiplos usuários. A arquitetura interna organiza-se em camadas lógicas, cada qual com responsabilidades bem definidas, facilitando a manutenção e a expansão do software.

5.2.1 Tipo de Arquitetura

A arquitetura lógica do sistema é composta por três camadas principais:

1. **Camada de Interface:** responsável pelas telas de menu, seleção de modo e nível, exibição de pontuações e tela de vitória. Utiliza diretamente os recursos gráficos do Pygame para renderização e captura de eventos de mouse e teclado. Comunica-se com a camada de Jogo instanciando objetos da classe `Game` e lendo seus atributos após o término de cada partida.

2. **Camada de Jogo:** núcleo da aplicação, onde reside o estado completo de uma partida (mapa, personagem, diálogo, editor de texto). Coordena os demais componentes, como sprites (`Player`, `Ground`, `Block`, `CheckPoint`) e o editor de código (`MultilineEditor`). Invoca a camada de Interpretação quando o jogador solicita a execução do pseudocódigo.

3. **Camada de Interpretação:** pipeline formado por `Lexer`, `Parser` e `Interpreter`. O Lexer converte o código-fonte em tokens; o Parser analisa esses tokens e gera uma AST; o Interpreter percorre a AST e produz a sequência de comandos de movimento executados pelo personagem. Essa camada é independente da interface gráfica, tendo sido inicialmente testada em ambiente isolado.

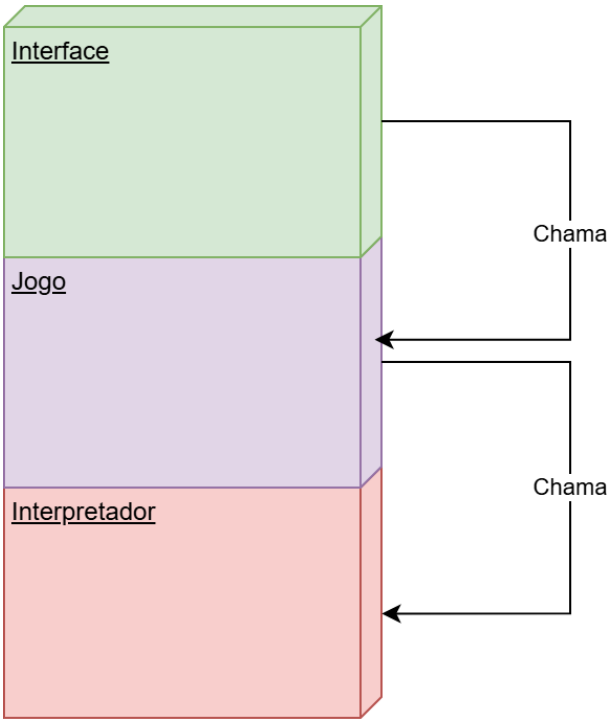

Figura 25 – Estrutura de Camadas.

A comunicação entre as camadas ocorre por chamadas diretas de funções como representado na Figura 25 e pela troca de estruturas de dados nativas do Python (listas, strings, objetos).

5.3 Padrões de Projetos Utilizados

Durante o desenvolvimento do sistema, alguns padrões de projeto foram empregados. O principal deles é o Visitor, que estrutura a compilação.

5.3.1 Visitor (Visitante)

O padrão Visitor foi aplicado na classe Interpreter, responsável por executar a AST gerada pelo Parser. A AST é composta por uma hierarquia de nós todos herdando de ASTNode. Cada nó representa uma construção da linguagem e possui uma estrutura específica.

Sem o Visitor, a execução da AST exigiria que cada classe de nó implementasse seu próprio método de interpretação, dificultando a manutenção. O Visitor resolve esse problema ao centralizar a lógica de processamento em uma única classe externa (o Interpreter), que ”visita” cada nó e executa a ação correspondente.

Para cada classe de nó (ex.: While), existe um método especializado visit_While(node) que sabe como executar a repetição. Se, futuramente, for necessário implementar nos funcionalidades, será necessário necessário somente a criação de um novo ”visit”.

5.3.2 Estratégia (Strategy) nos Modos de Jogo

A classe Game altera seu comportamento dependendo do modo de jogo ”Principal” ou ”Secundario”. A lógica interna é alterada por condicionais que selecionam diferentes conjuntos de ações: no modo Principal, o editor de texto é habilitado e o pipeline de compilação é utilizado; no modo Secundário, o editor é bloqueado e a movimentação é feita por setas. Sendo uma aplicação do padrão Strategy, em que o comportamento do objeto Game varia conforme um atributo de estado.

5.4 Tecnologias Utilizadas no Desenvolvimento

Nesta seção, são apresentadas as principais tecnologias empregadas na construção do sistema, abrangendo linguagens de programação, bibliotecas, ferramentas de apoio e configuração de hardware. Como o sistema é uma aplicação executada localmente, não foram utilizados recursos de rede ou bancos de dados externos.

5.4.1 Linguagens de Programação

A linguagem Python foi adotada por sua natureza de alto nível, que permite codificar de forma mais direta as funcionalidades exigidas pelo sistema, como manipulação de strings para o editor de código, leitura e escrita de arquivos para persistência, e processamento de expressões regulares para análise léxica. Além disso, o Python dispõe da biblioteca Pygame, específica para o desenvolvimento de jogos 2D, que oferece as abstrações necessárias para renderização gráfica, captura de eventos de teclado e mouse, controle de tempo e manipulação de sprites, elementos estes centrais do projeto. Por se tratar de um jogo leve, com gráficos 2d e lógica de execução limitada a poucos comandos por fase, não é necessário uma otimização robusta, o que elimina a necessidade de recorrer a linguagens de baixo nível (como C ou C++) que ofereceriam um maior controle sobre a performance a custo da simplicidade. Assim, Python se mostra a escolha ideal.

5.4.2 Bibliotecas

Diversos módulos da biblioteca padrão foram utilizados para tarefas específicas:

- Os e Sys: manipulação de caminhos de arquivos.
- Time: medição de tempo decorrido nas fases, essencial para o cálculo da pontuação.
- Re (Expressões Regulares): implementação do analisador léxico (Lexer), que tokeniza o código do jogador a partir de padrões definidos.
- Random: seleção aleatória de mapas dentre as variações disponíveis para cada nível.

5.5 Instalação e Configurações

Esta seção descreve os requisitos e procedimentos necessários para instalar e executar o sistema.

5.5.1 Pré-requisitos do Sistema

- **Sistema Operacional**: Windows 10 ou superior (64 bits). Compatibilidade com Linux ou macOS pode ser obtida ajustando-se os caminhos dos arquivos, mas o desenvolvimento e os testes concentraram-se na plataforma Windows.
- **Processador**: Dual-core com frequência de 1,5 GHz ou superior.
- **Memória RAM**: 2 GB disponíveis.
- **Espaço em disco**: aproximadamente 100 MB para o jogo e seus ativos (imagens, fontes, arquivos de dados).

- **Monitor**: resolução mínima de 1280×720 pixels.
- **Software**:
  - Python 3.10
  - Pygame

5.5.2 Processo de Instalação/Execução

1. Certificar de que o Python 3.10 está instalado
2. Instalar a biblioteca Pygame com o gerenciador de pacotes `pip`:

   ```
   pip install pygame
   ```

3. Copiar a pasta raiz do projeto para o computador de destino.
4. Abra um terminal na pasta Interface e execute:

   ```
   python main.py
   ```

   O jogo iniciará em tela cheia automaticamente.

5.6 Utilização do Sistema

Esta seção descreve como o sistema é executado e utilizado pelo usuário final. São apresentados o fluxo geral de navegação e as principais funcionalidades, ilustrados com telas capturadas durante a operação do jogo.

5.6.1 Processo de Execução do Sistema

O jogo é iniciado a partir da execução do arquivo `main.py` com o interpretador Python. Ao iniciar, o usuário é recebido pelo **Menu Principal**, onde pode escolher entre três opções: "JOGAR", que dá acesso aos modos de jogo; "PONTUAÇÕES", que exibe os recordes salvos; e "SAIR", que encerra a aplicação.

Ao selecionar "JOGAR", o usuário é conduzido à **Tela de Escolha de Modo**, onde decide entre o modo Principal e o modo Secundário.

Em qualquer um dos modos, a etapa seguinte é a **Tela de Seleção de Nível**. Os níveis já desbloqueados são exibidos em branco e podem ser selecionados; os bloqueados aparecem em cinza e não respondem a cliques. Uma vez escolhido o nível, a fase é carregada e o jogo tem início.

Durante a partida, a tela se divide em duas regiões principais: à esquerda, o editor de código na metade superior, e a área de diálogo na metade inferior; à direita, o mapa é exibido com o personagem, as paredes e o ponto de chegada.

No **modo Principal**, o tutorial é exibido no início de cada fase. O usuário pressiona `ENTER` para avançar as mensagens até que o diálogo se encerre e a área passe a mostrar a referência de comandos. O editor é então liberado para a digitação do código. Ao pressionar `F5`, o código é interpretado e executado, e o personagem se movimenta pelo mapa. Se houver erros, uma mensagem é exibida na área de diálogo. Ao atingir o ponto de chegada, a **Tela de Vitória** é apresentada, exibindo a pontuação obtida e oferecendo as opções de retornar ao menu ou avançar para o próximo nível.

No **modo Secundário**, o editor exibe um código de exemplo não editável, e o jogador movimenta o personagem com as setas direcionais. A cada tecla, o sistema verifica se o movimento corresponde ao comando esperado na sequência guia. Ao completar todos os movimentos corretamente, a vitória é registrada e a tela correspondente é exibida.

A qualquer momento, o jogador pode pressionar `ESC` para retornar ao menu principal, ou `F7` para reiniciar a fase atual.

5.6.2 Exemplos de Utilização das Principais Funcionalidades do Sistema

A seguir, são detalhadas as principais funcionalidades do sistema, com exemplos de utilização.

5.6.2.1 [RF001] Modo Principal

O modo Principal é a funcionalidade central do jogo, na qual o jogador escreve pseudocódigo para controlar o personagem. Após selecionar o modo Principal e um nível, a tela de jogo é carregada com o editor de código à esquerda e o mapa à direita. Um tutorial em balão de fala guia o jogador nas primeiras instruções. Ao pressionar ENTER repetidamente, o diálogo avança até ser substituído pelo texto de referência com os comandos disponíveis.

Com o editor liberado, o jogador pode digitar comandos como `move_right();`, `move_up();`, estruturas de repetição como `while` e definições de funções com `def`. Para executar o código, pressiona-se a tecla F5. O sistema então interpreta o texto: o Lexer gera os tokens, o Parser constrói a árvore sintática e o Interpreter produz a lista de movimentos. Se o código estiver correto, o personagem se move pelo mapa executando cada comando. Ao alcançar o checkpoint (C), a fase é concluída e a tela de vitória é exibida.

Caso o código contenha erros, como ”whil(i<3)” (Figura 27), uma caixa vermelha aparece na área de diálogo com a descrição do problema. O jogador pode corrigir o código e tentar novamente.

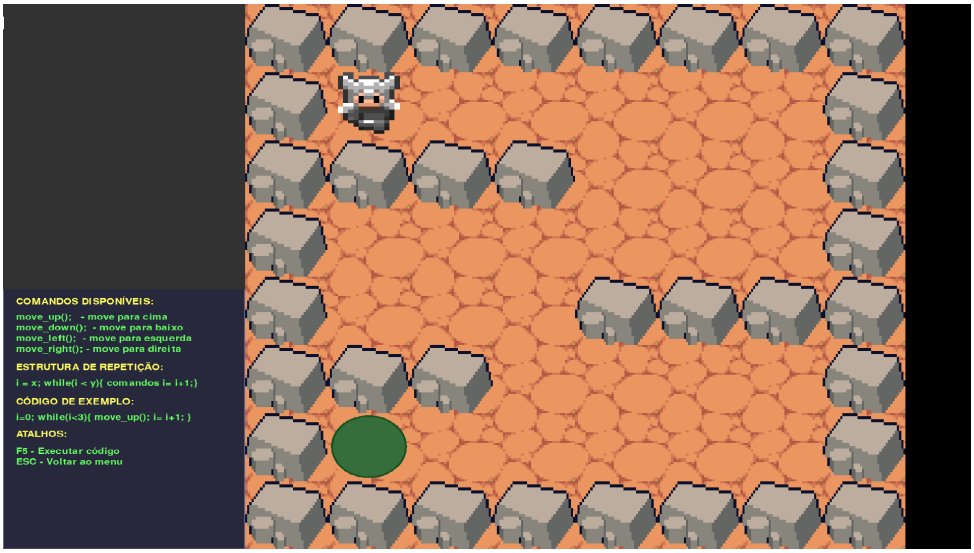

Figura 26 – Nível Modo Principal.

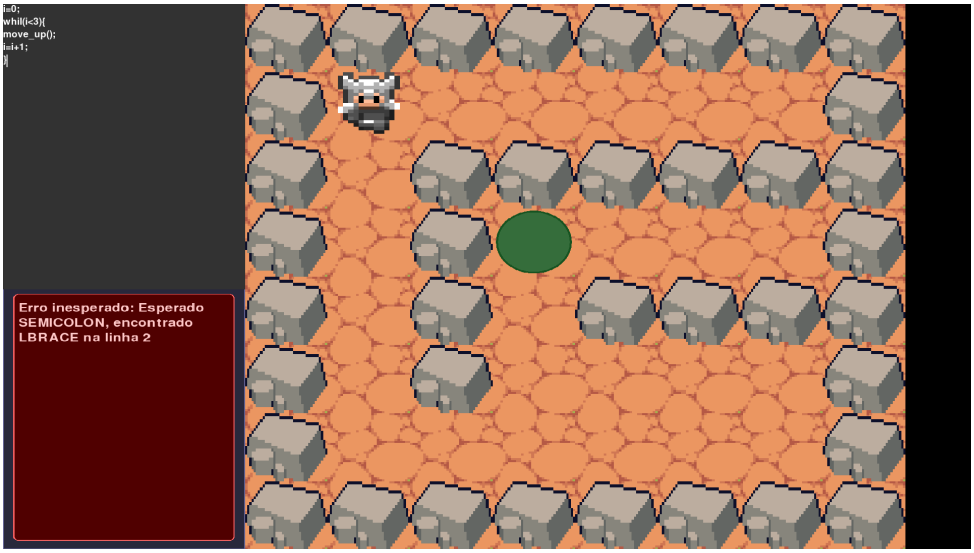

Figura 27 – Exemplo erro sintático.

5.6.2.2 [RF002] Modo Secundário

O modo Secundário (Figura 28) oferece uma abordagem complementar: em vez de escrever código, o jogador deve interpretar um programa já fornecido e reproduzi-lo com as setas do teclado. Após selecionar o modo e o nível, o editor exibe um código não editável, como:

```
move_up();
move_right();
move_down();
move_left();
```

O jogador observa o código e pressiona a seta correspondente a cada comando. Se

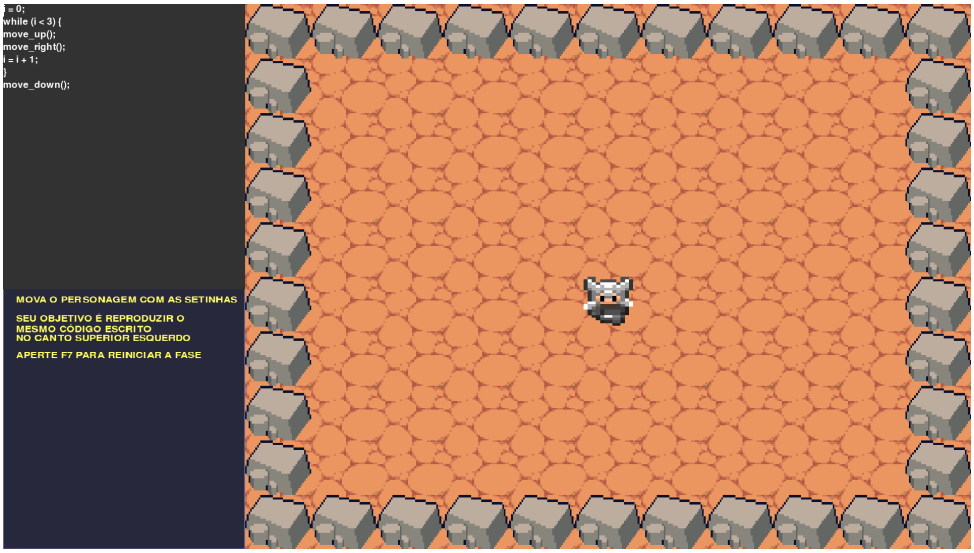

Figura 28 – Nível Modo Secundário.

a sequência estiver correta, o personagem se move e o sistema avança para o próximo comando. Se o jogador errar a direção a fase deve ser reiniciada com F7. Ao completar todos os movimentos na ordem correta, a vitória é registrada e a pontuação calculada.

Este modo permite que jogadores sem experiência em digitação de código compreendam a relação entre comandos escritos e ações no jogo.

5.6.2.3 [RF003] Visualização de Pontuações

As pontuações obtidas em cada nível podem ser consultadas de duas formas: ao final de cada fase, na tela de vitória, e a qualquer momento pelo menu principal, na opção ”PONTUAÇÕES”.

Na tela de pontuações (Figura 29), duas colunas são exibidas: à esquerda, os recordes do modo Principal; à direita, os do modo Secundário. Cada linha corresponde a um nível (1 a 5) e exibe a maior pontuação já alcançada. Os valores variam de 0 a 2000. Caso um nível nunca tenha sido concluído, a pontuação é exibida como zero. O botão ”VOLTAR” ou a tecla ESC retornam ao menu principal.

5.6.2.4 [RF004] Seleção de Nível

A qualquer momento, o jogador pode escolher um nível já desbloqueado por meio da tela de seleção (Figura 29). Após selecionar o modo de jogo, cinco botões são exibidos, cada um representando um nível. Níveis concluídos ou desbloqueados aparecem em branco; níveis ainda indisponíveis aparecem em cinza e não podem ser clicados.

Ao clicar em um nível disponível, a fase correspondente é carregada imediatamente. O progresso de desbloqueio é salvo automaticamente: ao vencer o nível 1, por exemplo, o ní-

Figura 29 – Tela de pontuação.

vel 2 torna-se acessível. Esse controle é mantido pelos arquivos `progresso_Principal.txt` e `progresso_Secundario.txt`.

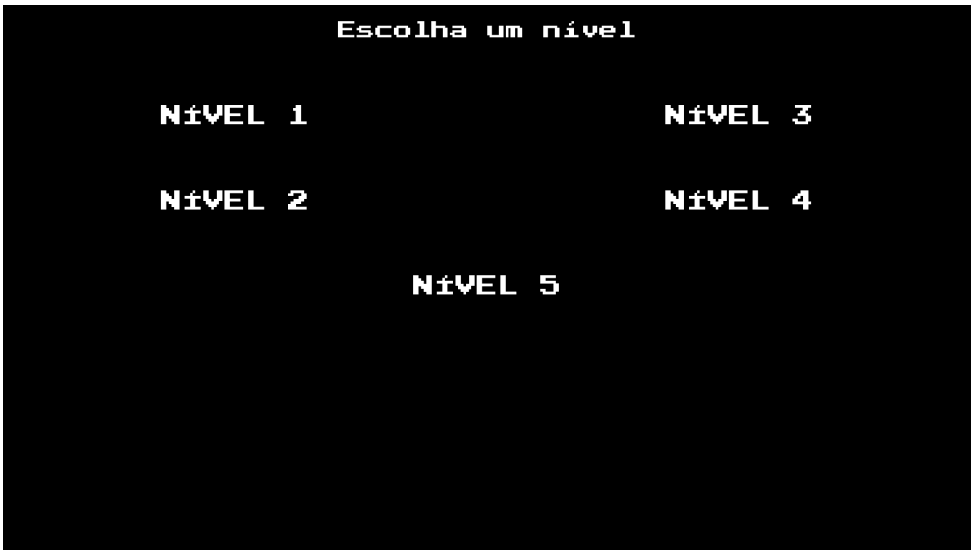

Figura 30 – Escolha de nível.

5.6.2.5 [RF005] Cálculo de Pontuação

A pontuação é calculada automaticamente ao final de cada fase concluída com sucesso (Figura 31). O critério varia conforme o modo:

- **Modo Principal**: a pontuação baseia-se na quantidade de comandos de movimento executados. Quanto menos comandos, maior a pontuação, seguindo a fórmula `2000 - (comandos × 100)`. O valor mínimo é 0.

- **Modo Secundário**: a pontuação baseia-se no tempo decorrido desde o início da fase. Quanto mais rápido, maior a pontuação, seguindo a fórmula `2000 - (tempo_em_segundos × 10)`.

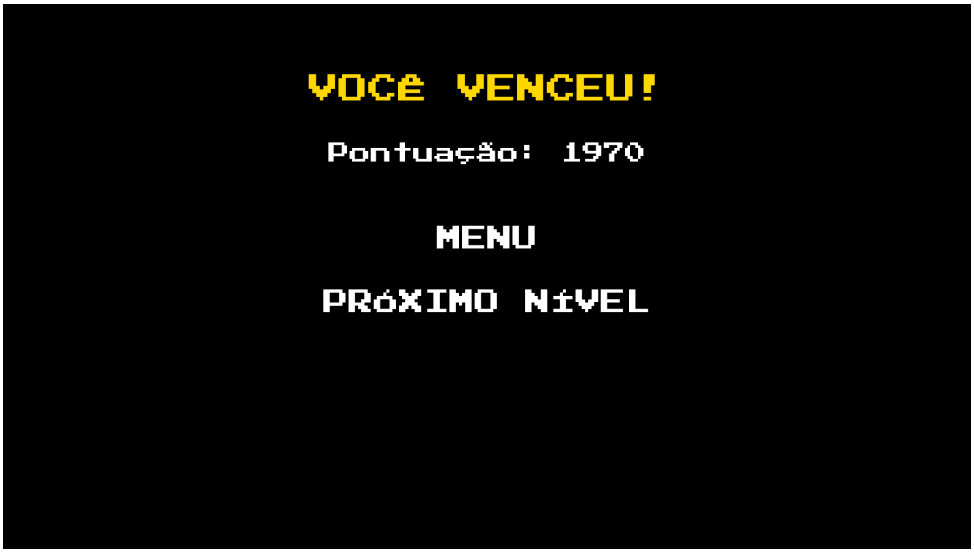

Figura 31 – Tela de vitória.

A pontuação é salva apenas se superar o recorde anterior para aquele nível. Os dados são armazenados nos arquivos `pontuacao_Principal.txt` e `pontuacao_Secundaria.txt`. Caso os arquivos não existam (primeira execução), são criados automaticamente com valores iniciais zero.

5.6.2.6 [RF006] Movimentação do Personagem

A movimentação do personagem é o núcleo da interação com o jogo. Ela ocorre de duas formas, dependendo do modo:

Modo Principal — execução por código:

1. O jogador digita o código no editor e pressiona F5.
2. O sistema interpreta o código e gera a lista de comandos de movimento.
3. Para cada comando, o sistema verifica se a célula de destino não é uma parede (B).
4. Se o caminho estiver livre, o personagem se move até a nova posição, com animação quadro a quadro obtida da spritesheet.
5. O tilemap é atualizado: a posição anterior vira piso (.) e a nova posição recebe o personagem (P).
6. Se o destino for o checkpoint (C), a vitória é registrada.

Modo Secundário — movimentação por setas:

1. O jogador pressiona uma seta direcional.
2. O sistema verifica se a célula de destino não é parede (B).
3. Se o caminho estiver livre, o personagem se move e o tilemap é atualizado.
4. O sistema compara o movimento com o próximo comando da sequência guia.
5. Se o movimento for incorreto, a fase deve ser reiniciada com F7.
6. Ao completar a sequência corretamente, a vitória é registrada.

Em ambos os modos, se o destino for uma parede, o personagem permanece na posição atual e nenhuma animação é executada. O jogador pode reiniciar a fase a qualquer momento pressionando F7.

5.7 Testes do Sistema

Para garantir a qualidade e a conformidade do software com os requisitos levantados, foi elaborado um plano de testes, que visa identificar defeitos, validar funcionalidades e assegurar que o comportamento do sistema. Sendo executados durante a implementação e novamente ao final do desenvolvimento do programa.

5.7.1 Plano de Testes

O plano de testes cobre os principais fluxos de execução do sistema, testando os casos positivos e os casos de erro.

- **Número:** 1
- **Descrição:** Iniciar o jogo e verificar se o menu principal é exibido em tela cheia, com o fundo redimensionado e os botões centralizados.
- **Resultado Esperado:** O menu principal ocupa toda a tela, sem bordas pretas; os botões ”JOGAR”, ”PONTUAÇÕES” e ”SAIR” aparecem centralizados e com tamanhos proporcionais à resolução.
- **Número:** 2
- **Descrição:** Navegar pelos menus: selecionar ”JOGAR” e qualquer um dos 2 modos e verificar se a tela de seleção de níveis é exibida com os botões corretos.

- **Resultado Esperado:** A tela de seleção de níveis aparece com cinco botões, sendo apenas os níveis já desbloqueados exibidos em branco; os demais em cinza e não clicáveis.

- **Número:** 3

- **Descrição:** No modo Principal, escolher um nível desbloqueado e verificar se o jogo inicia com o mapa, o editor de código e o diálogo tutorial visíveis.

- **Resultado Esperado:** A tela do jogo é dividida: editor de texto à esquerda, mapa à direita; na parte inferior esquerda, um personagem com balão de fala exibe a primeira mensagem do tutorial.

- **Número:** 4

- **Descrição:** Digitar um código simples (ex: move_right();) no editor e pressionar F5. Observar a movimentação do personagem no mapa.

- **Resultado Esperado:** O personagem se move uma casa para a direita, confirmando que o código foi executado com sucesso.

- **Número:** 5

- **Descrição:** Executar um código que leve o personagem a pisar no checkpoint (C). Verificar se a tela de vitória é exibida.

- **Resultado Esperado:** Ao atingir o checkpoint, o jogo exibe a tela de vitória, a pontuação calculada (entre 0 e 2000) e os botões "MENU" e "PRÓXIMO NÍVEL" (Se houver próximo nível).

- **Número:** 6

- **Descrição:** Na tela de vitória, pressionar "PRÓXIMO NÍVEL" e verificar se o nível seguinte é carregado e o progresso é salvo.

- **Resultado Esperado:** O jogo inicia o nível seguinte. O arquivo de progresso é atualizado com o novo nível desbloqueado. Na próxima visita ao menu, o nível completado está disponível.

- **Número:** 7

- **Descrição:** No modo Principal, escrever uma série de códigos com erros léxicos(ex: tokens que não existem), sintáticos (ex: não fechar chaves ao final de um while) e semânticas(ex: chamar uma variável que não foi instanciada ainda).

- **Resultado Esperado:** Nenhum movimento é executado. Na área de diálogo, aparece uma caixa vermelha com uma mensagem de erro informativa. Após alguns segundos, a mensagem desaparece e a escrita de código é reabilitada.

- **Número:** 8

- **Descrição:** Criar um código com loop infinito (ex.: `i=0; while(i<3)move_up();`) e executar com F5.

- **Resultado Esperado:** O código reconhece o looping infinito, pois chega ao limite de comandos e exibe uma mensagem de erro.

- **Número:** 9

- **Descrição:** No modo Secundário, mover o personagem com as setas do teclado conforme a sequência de comandos esperada.

- **Resultado Esperado:** Ao completar todos os movimentos corretos, a vitória é registrada e a pontuação é calculada com base no tempo decorrido.

- **Número:** 10

- **Descrição:** Acessar a tela de Pontuações pelo menu principal e verificar se os valores salvos são exibidos corretamente para ambos os modos.

- **Resultado Esperado:** A tela exibe cinco níveis, cada um com a pontuação do Modo Principal e do Modo Secundário. Os valores correspondem exatamente aos registros nos arquivos. Caso um valor nunca tenha sido gravado, o valor é zero.

- **Número:** 11

- **Descrição:** Pressionar F7 durante uma fase para reiniciar o nível.

- **Resultado Esperado:** O mapa é recriado a partir do mesmo maps_idx da fase atual; o personagem retorna à posição inicial; o diálogo tutorial é reativado; o cronômetro é reiniciado.

5.8 Repositório de Código

O projeto está hospedado no GitHub, acessível pelo link <https://github.com/bernardohmp/TCC>. O repositório contém o código fonte completo.

5.9 Resultados Alcançados com o Sistema

Ao final do desenvolvimento, obteve-se um jogo com 5 níveis de dificuldade progressiva por modo, cada um destes contendo 5 variações, portanto, ao total, o modo principal possui 25 mapas únicos e o modo secundário possui 25 códigos únicos, para promover a rejogabilidade. O sistema possui os modos Principal e Secundário, oferecendo, a escrita de pseudocódigo em um editor próprio e a resolução de desafios por meio de comandos direcionais. O editor de código exibe mensagens de erro para construções de erros sintáticos, loops infinitos ou comandos inexistentes, sem que o funcionamento do jogo seja interrompido. Dessa forma, o sistema alcança as funcionalidades propostas no capítulo de análise e projeto, possuindo os requisitos para ser um serious game, aplicando os aspectos de gamefication para promover o ensino do pensamento computacional.

5.10 Considerações Sobre o Capítulo

O desenvolvimento transcorreu em conformidade com o planejamento apresentado nos capítulos anteriores, e os resultados obtidos indicam que o sistema está apto para utilização. No próximo capítulo, serão dadas as conclusões gerais do trabalho, incluindo as contribuições, limitações e possíveis trabalhos futuros.

Capítulo 6

Conclusões

O objetivo geral do trabalho, desenvolver um jogo educativo capaz de introduzir os conceitos básicos do Pensamento Computacional e de programação, foi atingido com a construção de um software que integra principalmente a decomposição de problemas e o pensamento algorítmico.

Além disto, foi possível desenvolver uma interface que facilita-se a criação de novos níveis ou a expansão dos níveis existentes, evitando assim um projeto "hard-coded"difícil de ser modificado posteriormente.

6.1 Principais Contribuições

A principal contribuição deste trabalho reside na disponibilização de seu código fonte onde foram implementados um compilador em um jogo digital educativo. Ao documentar as decisões de projeto, a arquitetura e os padrões utilizados, como o Visitor no interpretador, o sistema oferece a outros estudantes e desenvolvedores um caso prático de como construir um compilador embutido em um ambiente de jogos. Permitindo que terceiros estudem desde a tokenização do pseudocódigo até a execução controlada dos comandos no jogo, servindo como material de referência.

6.2 Trabalhos Futuros

Embora o sistema tenha alcançado os objetivos propostos, algumas limitações foram identificadas e podem ser abordadas em trabalhos futuros.

Em primeiro lugar, a ampliação do conjunto de comandos disponíveis enriqueceria a experiência de programação, ou como em outros casos, a integração de linguagens de programação reais nas fases (como é feito no (INC., 2013), ao menos em níveis posteriores mais complexos.

Em segundo lugar, a implementação de um quadro de classificações (leaderboard) poderia intensificar o engajamento, introduzindo elementos de competição saudável entre

os estudantes. Associado a isso, um sistema de login com perfis individuais substituiria os arquivos de texto atuais, oferecendo maior organização dos dados.

Em terceiro lugar, o jogo, atualmente distribuído como uma aplicação local que exige instalação, pode ser adaptado para a plataforma web, eliminando a necessidade de download e configuração por parte do usuário e, consequentemente, ampliando o acesso a partir de qualquer dispositivo com um navegador.

Por fim, sugere-se a realização de estudos com turmas reais de programação, comparando o desempenho e a motivação de alunos que utilizaram o jogo com aqueles que seguiram apenas o método tradicional de ensino. Estes estudos forneceriam evidências quantitativas e qualitativas sobre a eficácia pedagógica da ferramenta, permitindo ajustes baseados nos dados.

Referências

Adobe Systems. **ActionScript 3.0 Reference**. [S.l.], 2010. Acesso em: 23 jul. 2024. Disponível em: <https://help.adobe.com/en_US/ActionScript/3.0_UsingAS3/>. Citado na página 21.

_____. **Adobe Flash Professional CS5**. [S.l.], 2010. Acesso em: 23 jul. 2024. Disponível em: <https://helpx.adobe.com/flash.html>. Citado na página 21.

ALVIM, Í. V. et al. Evasão nos cursos de graduação em computação no brasil. Universidade Estadual de Feira de Santana, 2024. Citado na página 33.

ANASTASIADIS, T.; LAMPROPOULOS, G.; SIAKAS, K. Digital game-based learning and serious games in education. **International Journal of Advances in Scientific Research and Engineering**, v. 4, n. 12, p. 139–144, 2018. Citado na página 13.

BROM, C.; PREUSS, M.; KLEMENT, D. Are educational computer micro-games engaging and effective for knowledge acquisition at high-schools? a quasi-experimental study. **Computers & Education**, Elsevier, v. 57, n. 3, p. 1971–1988, 2011. Citado 2 vezes nas páginas 25 e 28.

BUSARELLO, R. I. **Gamification: princípios e estratégias**. [S.l.]: Pimenta Cultural, 2016. Citado na página 12.

CANSU, F. K. An overview of computational thinking. **International journal of computer science education in schools**, 2019. Citado na página 11.

CoffeeScript Team. **CoffeeScript: Unfancy JavaScript**. 2021. Acesso em: 23 jul. 2024. Disponível em: <https://coffeescript.org/>. Citado na página 19.

cppreference.com. **C++ Reference**. 2023. Acesso em: 23 jul. 2024. Disponível em: <https://en.cppreference.com/w/>. Citado na página 19.

DEEPSEEK. **DeepSeek AI Assistant**. DeepSeek, 2025. Disponível em: <https://www.deepseek.com>. Citado na página 18.

Ecma International. **ECMAScript 2023 Language Specification**. 2023. Acesso em: 23 jul. 2024. Disponível em: <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>. Citado 2 vezes nas páginas 19 e 20.

Godot Engine. **Godot Engine 4.1 documentation**. [S.l.], 2023. Acesso em: 23 jul. 2024. Disponível em: <https://docs.godotengine.org/en/stable/>. Citado na página 29.

INC., C. **CodeCombat**. CodeCombat Inc., 2013. [Jogo de computador online]. Disponível em: <https://codecombat.com/>. Citado 4 vezes nas páginas 14, 19, 28 e 77.

Instituto Semesp. **Mapa do Ensino Superior 2023**. 2023. Dados do Censo da Educação Superior (INEP) 2021. Acesso em: 28 maio 2026. Disponível em: <https://www.semesp.org.br/wp-content/uploads/2023/06/mapa-ensino-superior-brasil-2023.pdf>. Citado na página 33.

KAZIMOGLU, C. et al. A serious game for developing computational thinking and learning introductory computer programming. **Procedia-Social and Behavioral Sciences**, Elsevier, v. 47, p. 1991–1999, 2012. Citado 4 vezes nas páginas 14, 21, 27 e 28.

LAAMARTI, F.; EID, M.; SADDIK, A. E. An overview of serious games. **International Journal of Computer Games Technology**, Wiley Online Library, v. 2014, n. 1, p. 358152, 2014. Citado na página 12.

LEE, T. Y. et al. Ctarcade: Computational thinking with games in school age children. **International Journal of Child-Computer Interaction**, Elsevier, v. 2, n. 1, p. 26–33, 2014. Citado 2 vezes nas páginas 24 e 28.

PAPERT, S. An exploration in the space of mathematics educations. **International Journal of Computers for Mathematical Learning**, v. 1, n. 1, p. 95–123, 1996. ISSN 1573-1766. Disponível em: <https://doi.org/10.1007/BF00191473>. Citado na página 11.

PENDLETON, A. J. **Introducing the game design matrix: A step-by-step process for creating serious games**. [S.l.], 2020. Citado na página 15.

Python Software Foundation. **Python Documentation**. [S.l.], 2023. Acesso em: 23 jul. 2024. Disponível em: <https://docs.python.org/3/>. Citado na página 19.

RITTERFELD, U.; WEBER, R. Video games for entertainment and education. **Playing video games: Motives, responses, and consequences**, p. 399–413, 2006. Citado na página 12.

RODRIGUES, F. S. Estudo sobre a evasão no curso de ciência da computação da ufrgs. 2013. Citado na página 33.

SHADBAD, F. et al. Best of both worlds: The inclusion of gamification in virtual lab environments to increase educational value. 2023. Citado na página 12.

TROIANO, G. M. et al. Is my game ok dr. scratch? exploring programming and computational thinking development via metrics in student-designed serious games for stem. In: **Proceedings of the 18th ACM international conference on interaction design and children**. [S.l.: s.n.], 2019. p. 208–219. Citado 3 vezes nas páginas 22, 27 e 28.

Unity Technologies. **Unity User Manual 2023.1**. [S.l.], 2023. Acesso em: 23 jul. 2024. Disponível em: <https://docs.unity3d.com/Manual/>. Citado na página 29.

WING, J. M. Computational thinking. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 3, p. 33–35, mar. 2006. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1118178.1118215>. Citado na página 11.

WOLFFELT, M. **Elevator Saga: The elevator programming game**. 2013. Jogo educativo online. Disponível em: <https://play.elevatorsaga.com/>. Citado 2 vezes nas páginas 20 e 28.

ZIRAWAGA, V. S.; OLUSANYA, A. I.; MADUKU, T. Gaming in education: Using games as a support tool to teach history. **Journal of Education and Practice**, ERIC, v. 8, n. 15, p. 55–64, 2017. Citado na página 13.