INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

TIFANY LUIZA DE JESUS MOREIRA

DESENVOLVIMENTO DE JOGO “LABIRINTO DAS SOMBRAS” EM C++ UTILIZANDO A BIBLIOTECA RAYLIB PARA A DISCIPLINA DE PROGRAMAÇÃO ORIENTADA A OBJETOS

CAMPOS DO JORDÃO

2025

RESUMO

Este relatório detalha o desenvolvimento e a implementação de "Labirinto das Sombras", um jogo de labirinto dinâmico e original para computador, construído integralmente em C++ com o auxílio da biblioteca Raylib. O jogo desafia o jogador a navegar por labirintos de dificuldade crescente, evitando obstáculos móveis e armadilhas, enquanto coleta moedas e power-ups estratégicos, tudo isso dentro de um prazo global estabelecido. A progressão pelos níveis introduz maior complexidade no design do labirinto e acelera o comportamento dos elementos adversos, testando a capacidade de reação e o planejamento tático do jogador. A pontuação é incrementada pela aquisição de moedas e bônus por fases completadas, com o tempo restante funcionando como um fator crítico para o sucesso final. A arquitetura do projeto foi pensada sob os princípios da Programação Orientada a Objetos, resultando em um código modular, de fácil leitura e adaptável para expansões, além de documentação detalhada para futuras manutenções, também.

Palavras-Chave: Jogo, C++, Raylib, Programação Orientada a Objetos, Labirinto, Desenvolvimento de Jogos.

SUMÁRIO

1 INTRODUÇÃO

No cenário contemporâneo da computação, o desenvolvimento de jogos digitais emerge como um campo de aplicação prática altamente relevante para a síntese de conhecimentos teóricos e a exploração de desafios computacionais intrincados. Este trabalho se insere nesse contexto, apresentando o "Labirinto das Sombras", um jogo eletrônico original, integralmente concebido e implementado em C++ utilizando a biblioteca Raylib. A escolha da linguagem e da biblioteca foi uma premissa estabelecida pela disciplina de Programação Orientada a Objetos, orientando o desenvolvimento para a exploração de suas capacidades na criação de um sistema interativo.

O "Labirinto das Sombras" foi arquitetado para oferecer uma experiência de labirinto dinâmico, onde o jogador é desafiado a navegar por fases cuja complexidade aumenta progressivamente. A essência da jogabilidade reside na habilidade de desviar de obstáculos — incluindo paredes móveis e elementos que aparecem e desaparecem — enquanto se coleta moedas e power-ups estratégicos que oferecem vantagens no percurso. O objetivo primordial é alcançar a saída de cada labirinto antes que um tempo limite global se esgote, adicionando uma camada crítica de estratégia, visto que o cronômetro não é reiniciado a cada nova fase. Essa concepção visou testar a capacidade de adaptação e o raciocínio rápido do jogador diante de um ambiente em constante transformação.

A finalidade primordial deste projeto transcende a mera entrega de um produto funcional; ele representa um exercício prático e aprofundado na aplicação dos princípios da Programação Orientada a Objetos. O desenvolvimento teve como objetivo demonstrar, de forma tangível, a utilização de conceitos como encapsulamento, herança e polimorfismo na construção de um sistema complexo e interativo. A escolha do C++ e da Raylib, apesar de predefinida, justificou-se plenamente ao permitir um controle granular sobre a arquitetura do jogo e ao fornecer as ferramentas essenciais para a prototipagem rápida e a implementação gráfica. A criação de um jogo inédito, especificamente para este propósito acadêmico, reforça o compromisso com a originalidade e a exploração de soluções criativas dentro das restrições do projeto.

O processo de desenvolvimento seguiu uma metodologia que enfatizou a modelagem de cada elemento do jogo como uma entidade orientada a objetos. Essa abordagem permitiu a criação de classes distintas para o jogador, os diferentes tipos de paredes (fixas, móveis e especiais), os itens coletáveis (moedas e power-ups) e o objetivo final, cada qual com suas responsabilidades e comportamentos bem definidos. Os fundamentos teóricos que embasam este projeto incluem o ciclo de jogo (game loop), que coordena a atualização lógica e a renderização gráfica, e a implementação de mecânicas de detecção de colisões. Além disso, a gestão de estados do jogo (menu, jogando, pausado) e a inclusão de um sistema de partículas para efeitos visuais contribuíram para a robustez da aplicação e para uma experiência de usuário mais imersiva, consolidando o aprendizado teórico em uma aplicação prática.

2. METODOLOGIA

A concepção de "Labirinto das Sombras" iniciou-se, a princípio, apenas como um projeto acadêmico focado na aplicação prática dos conhecimentos adquiridos em Programação Orientada a Objetos, utilizando a linguagem C++ e a biblioteca Raylib como ferramentas predefinidas pela ementa da disciplina. A visão inicial para o jogo era criar uma experiência de labirinto que transcende a estaticidade, introduzindo elementos dinâmicos e um desafio progressivo que mantivesse o jogador engajado, fugindo do padrão de um Game comum de labirinto, em que, não se há muita variação do desenvolver do jogo — por padrão, o labirinto possui um início e fim, com o objetivo de encontrar o melhor caminho para o fim, sem adições de elementos que potencializam as partidas. O objetivo primordial era não apenas desenvolver um jogo funcional, mas, sobretudo, estruturá-lo de forma a evidenciar os benefícios de um design orientado a objetos em um ambiente interativo.

Para o desenvolvimento do projeto, as ferramentas essenciais empregadas foram:

- **Linguagem de Programação:** C++. A escolha da linguagem foi um requisito da disciplina, proporcionando a oportunidade de explorar recursos avançados e a melhor gestão de memória.

- **Biblioteca de Desenvolvimento de Jogos:** Raylib. Predefinida para o projeto, a Raylib ofereceu uma abstração eficaz para renderização gráfica, processamento de áudio e gerenciamento de entrada, permitindo ao desenvolvedor focar na lógica do jogo.
- **Ambiente de Desenvolvimento Integrado (IDE):** CLion, configurado com o compilador C++ para construção e depuração do projeto.
- **Sistema de Controle de Versão:** Git, integrado com dois repositórios no GitHub: CJOPROO, para publicação, sendo um repositório público, e outro privado para gerenciar o código-fonte, versionamento e os assets do projeto, facilitando o controle de alterações e a entrega do projeto.

O game foi desenvolvido com uma arquitetura orientada a objetos, onde cada elemento ativo no ambiente de jogo é representado por uma classe específica, seguindo um sistema de jogos bem estruturado. A classe **Game** atua como a ferramenta central do projeto, em que esta gerencia os diferentes estados do jogo (menu, em jogo, pausado, transição de nível, vitória, game over) e coordena a atualização e o desenho de todas as entidades definidas. O ciclo de vida do jogo é orquestrado por um laço principal (while (**!WindowShouldClose()**)) que, a cada iteração, executa as fases de **Update()** para a lógica e Draw() para a renderização. O método **Update()** da classe Game é responsável por avançar a simulação, processando entradas do jogador, atualizando posições de entidades móveis, decrementando temporizadores de efeitos e verificando as condições de vitória ou derrota. A passagem do tempo é calculada de forma independente da taxa de quadros (framerate) através do **delta time** (GetFrameTime()), garantindo que a fluidez da simulação seja consistente em diferentes ambientes de execução. A gestão de estados do jogo é controlada por uma máquina de estados simples, onde uma enumeração (**GameState**) e estruturas **switch** direcionam qual subconjunto de lógica e elementos visuais deve ser processado e renderizado em determinado momento, promovendo uma execução modular e prevenindo interferências entre as telas do jogo. Concomitantemente, o método **Draw()** se encarrega de renderizar todos os elementos visuais na tela, desde o fundo dinâmico e seus efeitos sutis até o jogador, obstáculos, coletáveis e a interface do usuário, garantindo uma representação visual coesa do estado atual do jogo.

A base para todos os objetos interativos deste projeto é a classe abstrata **Entidade**, que define propriedades comuns como a posição e as dimensões (representadas por um Rectangle), além de comportamentos fundamentais como **Update()** e **Draw()**, que são sobrescritos pelas classes derivadas. Essa hierarquia de classes é um exemplo direto de polimorfismo, permitindo que um vetor de ponteiros para **Entidade** (**std::vector<Wall*> walls**) trate diferentes tipos de objetos de forma unificada no ciclo de atualização e desenho do jogo, enquanto cada objeto individual executa sua própria lógica especializada. A partir de **Entidade**, foram especializadas diversas classes:

- **Player:** Representa o personagem controlável pelo jogador. A movimentação é implementada pela detecção contínua de teclas (WASD ou setas direcionais) via **IsKeyDown()**, ajustando a posição (**rect.x, rect.y**) em cada frame. A velocidade de deslocamento (**currentSpeed**) é multiplicada por **delta** para assegurar que o movimento seja suave e consistente independente do desempenho da máquina. Limites de tela são rigorosamente impostos, impedindo que o jogador ultrapasse a área visível do jogo. Atributos como vidas, invencibilidade (ativa por 1.5 segundos após dano) e duração de escudo/velocidade são geridos por temporizadores dedicados, os quais são decrementados a cada **Update()**, e sua ativação é acompanhada por efeitos visuais distintos no jogador (piscar, aura de escudo). Um rastro visual de partículas escuras é emitido constantemente pela posição do jogador, criando um efeito de fluidez e movimento.
- **Wall:** Constitui os obstáculos básicos do labirinto. Instâncias de Wall podem ser estáticas (velocidade zero) ou possuir vetores de velocidade que as impulsionam pelo cenário. A inversão da direção ocorre ao atingir os limites da tela, adicionando um elemento dinâmico e imprevisível ao labirinto.
- **ParedeEspecial:** Deriva de **Wall**, introduzindo um tipo de obstáculo que causa dano ao jogador ao contato e se distingue por um efeito visual pulsante e vibrante. Sua opacidade e brilho de borda alternam ritmicamente utilizando funções seno baseadas no tempo (**sinf(GetTime())**), alertando visualmente sobre sua periculosidade.
- **Coin:** Representa os colecionáveis que aumentam a pontuação em +100 do jogador ao serem tocados. Visualmente, as moedas apresentam um efeito de

pulsação de tamanho e um brilho sutil, chamando a atenção para sua coletabilidade.

- **PowerUp:** Oferece bônus temporários ao jogador. Esta classe utiliza uma enumeração (**PowerUpType**) para diferenciar os seguintes tipos de bônus, que aparecem após um **spawnTimer** aleatório:
    - **LIFE:** Concede uma vida adicional ao jogador, limitado ao máximo de três vidas (o incremento de vida é permanente até o limite máximo de vidas do jogador).
    - **SHIELD:** Ativa um escudo protetor que concede invencibilidade temporária a colisões com paredes ou ataques inimigos, com duração de 5.0 segundos. Visualmente, um círculo translúcido e pulsante envolve o jogador durante o efeito, com a intensidade do pulso variando para um feedback dinâmico.
    - **SPEED:** Aumenta temporariamente a velocidade de movimento do jogador, permitindo maior agilidade na travessia do labirinto, com duração de 4.0 segundos. Durante sua ativação, a velocidade base do jogador é aumentada em 50%. A classe PowerUp implementa um sistema de spawn temporizado para o reaparecimento desses bônus em locais aleatórios no cenário, garantindo uma distribuição dinâmica dos power-ups ao longo da partida.
- **Objetivo:** Marca o ponto de saída do labirinto. Ele é ativado apenas após todas as moedas de um nível serem coletadas, desencadeando uma animação visual de surgimento e um efeito de pulsação luminosa para sinalizar sua disponibilidade ao jogador.

A detecção de colisões é um pilar fundamental da interatividade do jogo, sendo implementada por meio da função **CheckCollisionRecs**() da Raylib, que verifica a sobreposição entre os retângulos de colisão das entidades. Este sistema é aplicado em cada **Update()** para uma ampla gama de interações: do jogador com paredes (levando a dano se não houver invencibilidade ou escudo), com moedas (resultando em coleta e pontuação), com power-ups (ativando seus respectivos bônus) e com o objetivo (avançando de nível). Cada colisão é tratada para disparar as lógicas e efeitos visuais correspondentes, além de decrementar uma vida do jogador. Um **ParticleSystem** foi cuidadosamente integrado para gerenciar efeitos

visuais dinâmicos, como explosões ao coletar itens ou impactos em colisões, adicionando um feedback visual tátil e aprimorando a imersão do usuário. A emissão de partículas em **ParticleSystem::Emit()** é parametrizável, permitindo controlar a quantidade, cor, velocidade e tamanho inicial das partículas manualmente (ou em implementações futuras, sendo isto feito dinamicamente), que se dispersam com velocidade aleatória em ângulos variados. No **Update()** do sistema de partículas, cada partícula tem sua posição atualizada e seu tempo de vida reduzido; ao expirar, são eficientemente removidas do vetor através de **std::remove_if** e **erase**, evitando acúmulo desnecessário de memória. A interface do usuário é facilitada pela classe **Button**, que encapsula a lógica de interação (verificação de **hover** do mouse com **CheckCollisionPointRec**() e detecção de **click** com **IsMouseButtonPressed()**) para elementos de menu e pause, promovendo uma navegação intuitiva e responsiva. O efeito de "screen shake" é aplicado na câmera através da alteração de seu **offset** de forma aleatória quando o jogador sofre dano, intensificando a sensação de impacto e a visualização desta interação com o game. A gestão de recursos (texturas, sons, fontes) é centralizada nos métodos **LoadResources()** e **UnloadResources()** da classe **Game**, garantindo carregamento e descarregamento eficientes ao longo do ciclo de vida do jogo e otimizando o uso de memória. O design de níveis foi configurado inteiramente manualmente, com a disposição de paredes já definidas, moedas e power-ups também adaptados para aumentar a dificuldade a cada progressão de level.

O jogo é composto por três níveis distintos, cada um projetado para introduzir novos desafios e intensificar a experiência:

- **Nível 1:** Apresenta 5 moedas a serem coletadas, distribuídas estrategicamente para guiar o jogador pelos primeiros desafios. As paredes (instâncias de **Wall**) possuem velocidades variadas, com movimento predominantemente vertical variando entre 70 e 100 unidades/segundo, e horizontal de 50 unidades/segundo, proporcionando um desafio inicial moderado e servindo como introdução às mecânicas dinâmicas.
- **Nível 2:** Contém 7 moedas, um aumento na quantidade que demanda mais exploração. Este nível introduz a **ParedeEspecial**, com movimentos verticais mais rápidos, entre 140 e 150 unidades/segundo, exigindo maior precisão e

tempo de reação do jogador para evitar o dano e consequentemente, a perda de uma vida. As paredes normais mantêm um ritmo similar ao nível anterior, mas com algumas velocidades ligeiramente aumentadas (100 a 120 unidades/segundo) para intensificar a dificuldade e a fluidez do cenário.

- **Nível 3:** O nível mais desafiador, com 10 moedas para coletar. As **ParedeEspecial** operam em velocidades significativamente maiores, atingindo entre 210 e 220 unidades/segundo verticalmente, e algumas paredes móveis simples também são mais rápidas, entre 180 e 200 unidades/segundo, elevando drasticamente a dificuldade de desvio. A complexidade do layout é acentuada, com mais obstáculos estáticos e móveis dispostos de forma densa, exigindo alta coordenação, planejamento e aproveitamento estratégico dos power-ups para uma travessia bem-sucedida.

Além disso, fora construído o diagrama de classes que representa a construção deste projeto:

<Acesse em: https://github.com/tifanymoreira/CJOPROO/tree/main/UML>

3. RESULTADOS OBTIDOS

A concretização do projeto "Labirinto das Sombras" resultou em uma aplicação funcional que reflete as escolhas de design e a implementação das mecânicas detalhadas na metodologia. O jogo opera em um ambiente gráfico bidimensional, com elementos visuais que buscam criar uma atmosfera coesa com a temática sombria proposta, utilizando a paleta de cores definida. A interface do usuário é clara e direta, proporcionando ao jogador todas as informações necessárias para acompanhar seu progresso e interagir com o sistema.

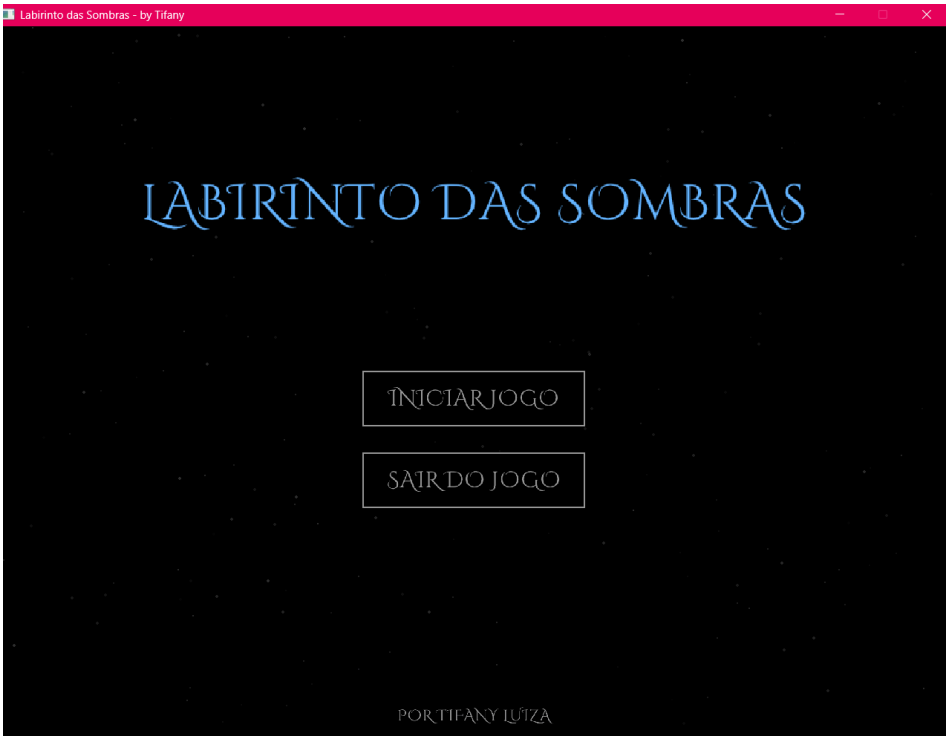

Figura 1: Tela do Menu Principal do jogo 'Labirinto das Sombras'. Fonte: Tifany Luiza, captura de tela.

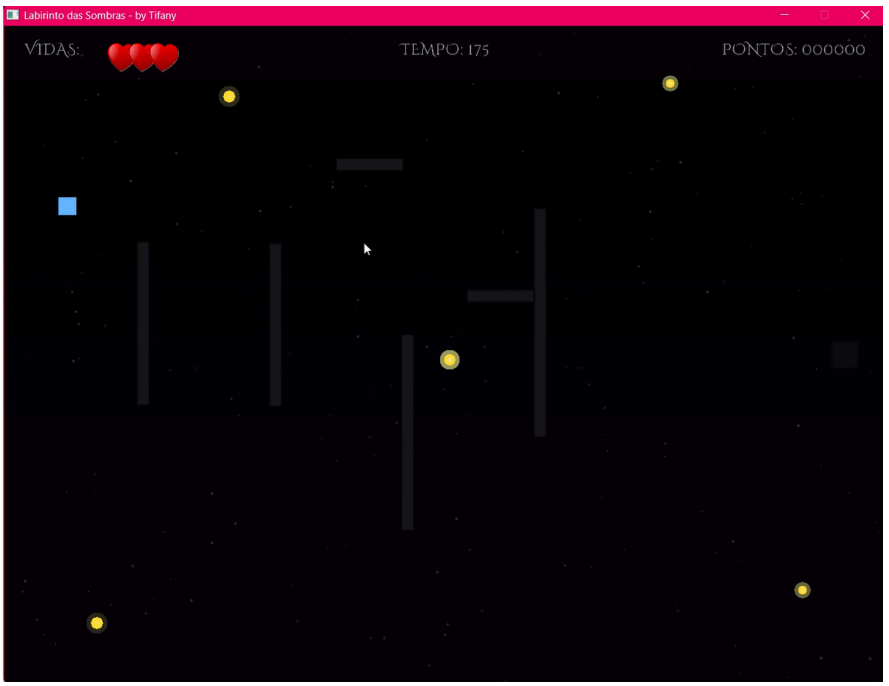

Figura 2: Cena do game, fase 1, exibindo o jogador durante partida com moedas e paredes visíveis, além de status de vidas, tempo restante e score. Fonte: Tifany Luiza, captura de tela.

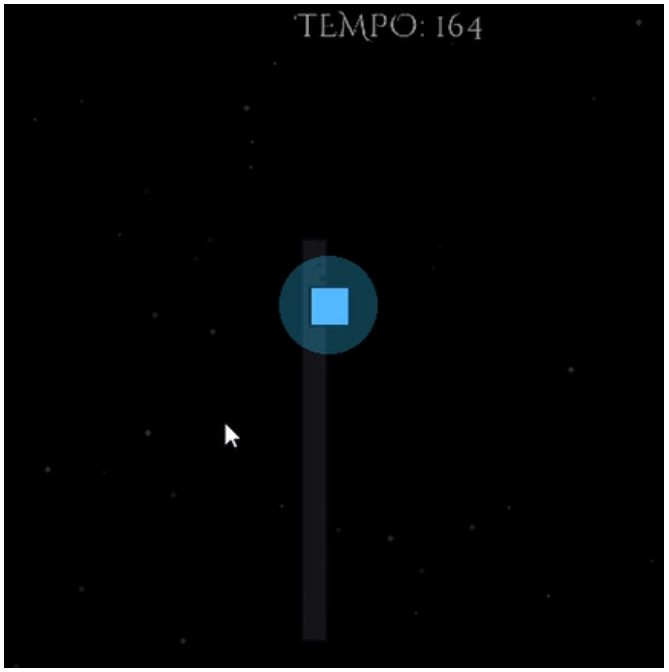

Figura 3: Jogador com power-up de escudo ativo, ilustrando o efeito visual do bônus e o seu efeito de protegê-lo contra impacto da parede. Fonte: Tifany Luiza, captura de tela.

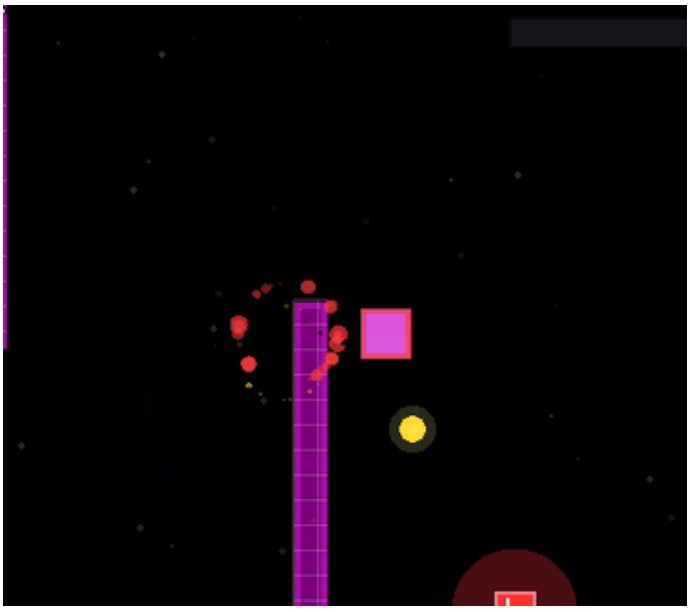

Figura 4: Colisão do jogador com uma Parede Especial, com feedback visual de impacto. Fonte: Tifany Luiza, captura de tela.

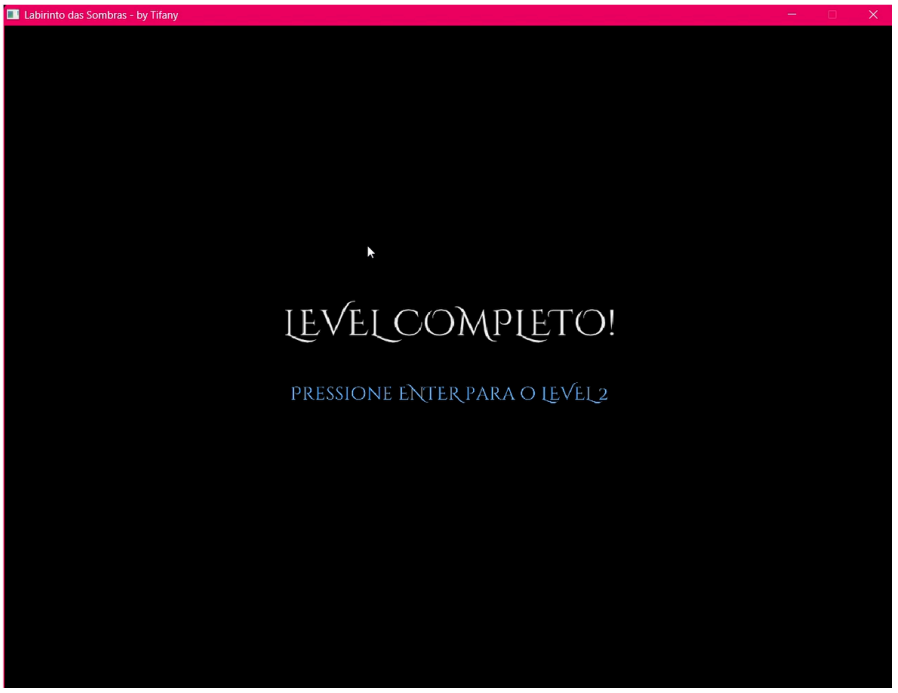

Figura 5: Tela de Transição de Nível, indicando a conclusão de uma fase e qual a próxima fase a ser iniciada. Fonte: Tifany Luiza, captura de tela.

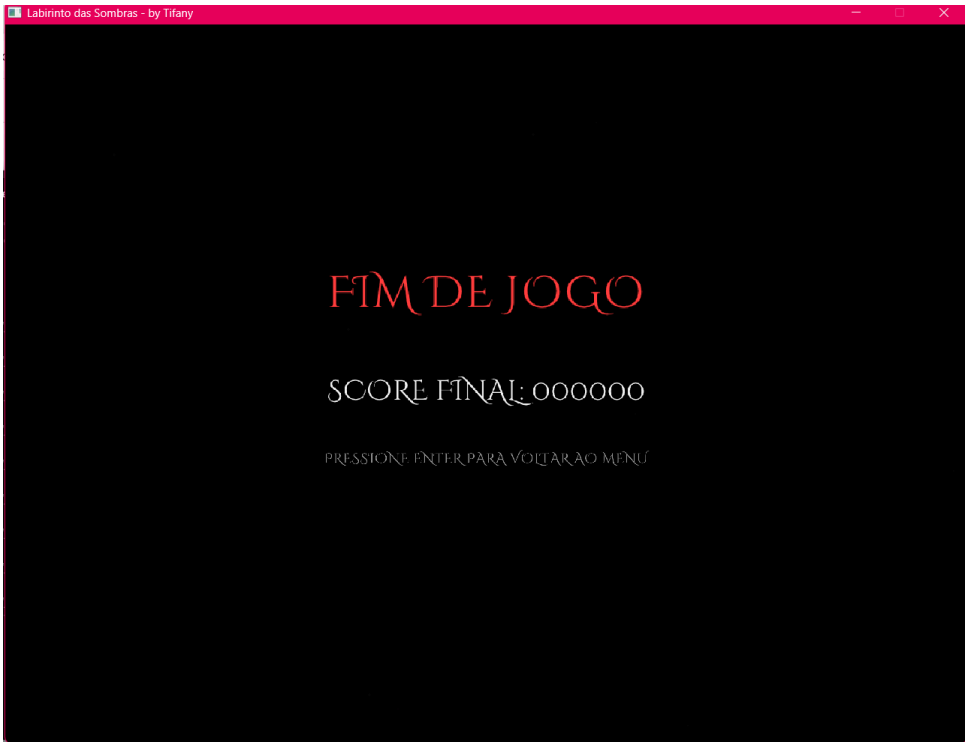

Figura 6: Tela de DERROTA, exibindo a pontuação final da partida. Fonte: Tifany Luiza, captura de tela.

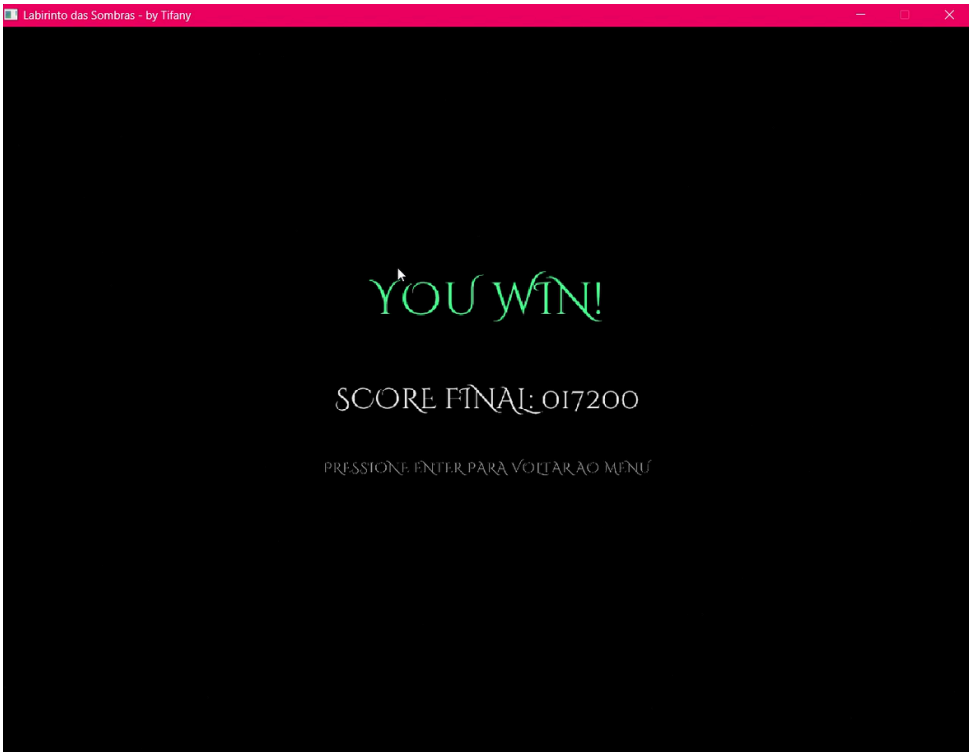

Figura 7: Tela de VITÓRIA, exibindo a pontuação final da partida. Fonte: Tifany Luiza, captura de tela.

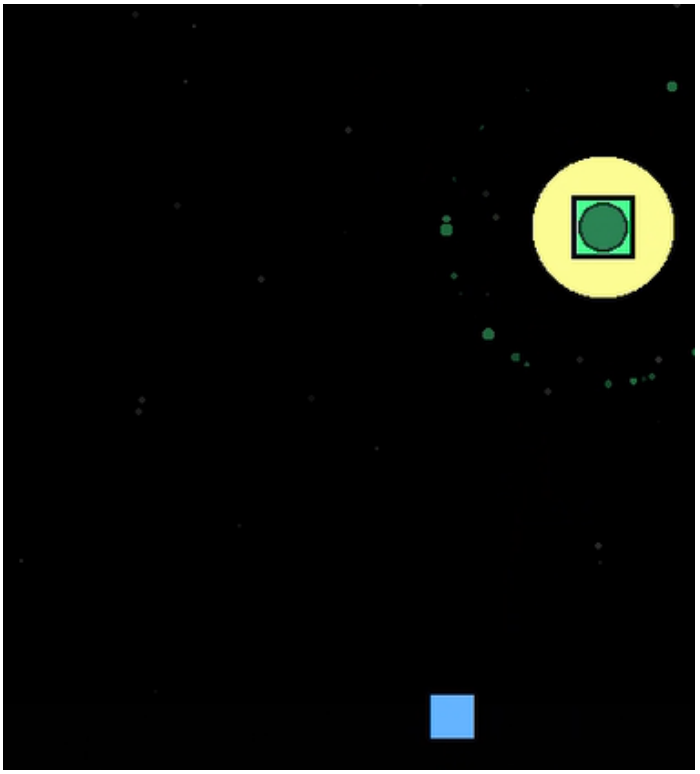

Figura 8: Momento em que o campo de objetivo é habilitado após a coleta de todas as moedas do level. Fonte: Tifany Luiza, captura de tela.

4. CONCLUSÃO

O projeto apresentado no discorrer deste relatório representou a concretização dos objetivos propostos para o projeto acadêmico, demonstrando a capacidade de aplicar os princípios da Programação Orientada a Objetos (POO) em um ambiente prático e interativo. O jogo finalizado reflete um esforço bem-sucedido na criação de uma experiência original, utilizando a linguagem C++ e a biblioteca Raylib como ferramentas fundamentais. A arquitetura orientada a objetos adotada, com a definição clara de classes para entidades como Player, Wall, Coin, PowerUp e Objetivo, juntamente com a hierarquia da classe base Entidade, provou ser altamente eficaz na gestão da complexidade inerente a um sistema de jogo dinâmico. Essa abordagem não apenas facilitou a implementação de lógicas complexas para movimentação, colisões e estados de jogo, mas também estabeleceu uma base robusta e de fácil manutenção para futuras expansões. O jogo atende à proposta de um labirinto não estático, com obstáculos e elementos que evoluem em velocidade e quantidade, garantindo um desafio progressivo ao jogador ao longo dos três níveis implementados.

Embora "Labirinto das Sombras" tenha alcançado seus objetivos primários e demonstre bem os conceitos de POO em um contexto funcional, diversas oportunidades de aprimoramento e expansão foram identificadas ao longo do processo de desenvolvimento, mas que não foram executadas devido à prazos e falta de oportunidade de melhor aprofundamento nos estudos das ferramentas utilizadas, as quais poderiam elevar a experiência de jogo e a complexidade do projeto:

- **Expansão de Conteúdo:** A inclusão de mais níveis, talvez com layouts gerados proceduralmente, aumentaria significativamente a longevidade do jogo.
- **Diversificação de Obstáculos e Inimigos:** Introduzir novos tipos de paredes com comportamentos mais complexos ou até mesmo inimigos que patrulhem o labirinto adicionaria camadas de desafio e estratégia.
- **Novos Power-ups e Habilidades:** Criar power-ups com efeitos mais variados

ou conceder ao jogador habilidades ativas (como por exemplo, disparos de tiros contra as Walls que, quando recebido X tiros, seriam destruídas) o que poderia enriquecer a jogabilidade e as escolhas táticas.

- **Sistema de Áudio Aprimorado:** Expandir a trilha sonora e incluir uma maior variedade de efeitos sonoros, especialmente para as interações e efeitos visuais, poderia aprofundar a imersão.
- **Interface e Experiência do Usuário (UX) Refinadas:** Aprimorar o menu de opções para permitir que o jogador configure controles, volume ou aspectos gráficos, e adicionar um sistema de feedback visual mais detalhado para o cronômetro, seriam melhorias valiosas, bem como, também, possibilitar o restart e pause do game.
- **Implementação de Persistência de Dados:** Adicionar a capacidade de salvar o progresso do jogo permitiria aos jogadores competir e retomar partidas.
- **Gerenciamento de Memória Otimizado:** A substituição de ponteiros brutos por ponteiros inteligentes (como **std::unique_ptr**) para a gestão de entidades dinâmicas, como as paredes, tornaria o código mais seguro e menos propenso a vazamentos de memória, embora não fosse uma barreira funcional no escopo atual do projeto.

As sugestões apresentadas representam caminhos potenciais para o aprofundamento do projeto, tanto em termos de complexidade de software quanto de engajamento do jogador, reafirmando a base sólida que foi estabelecida com a aplicação dos princípios de Programação Orientada a Objetos.

REFERÊNCIAS

RAYLIB DEVELOPMENT TEAM. **Raylib cheatsheet**. *raylib.com*, [s.d.]. Disponível em: https://www.raylib.com/cheatsheet.html. Acesso em 02 jun. de 2025.

GEEKSFORGEEKS. **Object Oriented Programming (OOP) Concepts in C++**. *GeeksforGeeks*,[s.d.]. Disponível em: https://www.geeksforgeeks.org/object-oriented-programming-oops-concepts-in-cpp/. Acesso em: 07 jun. 2025.

CPPREFERENCE.COM. **C++ language reference**. *cppreference.com*, [s.d.]. Disponível em: https://en.cppreference.com/w/cpp/language. Acesso em: 22 mai. 2025.