INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

MATHEUS LOPES DE OLIVEIRA

Este documento se trata da entrega projeto final da disciplina de Programação Orientada a Objetos, do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - Campus Campos do Jordão (IFSP-CJO).

PROFESSOR: Paulo Giovani de Faria Zeferino.

Jogo de Carro com Raylib C++

CAMPOS DO JORDÃO

2025

RESUMO

O presente trabalho descreve o desenvolvimento de um jogo de corrida 2D, utilizando a biblioteca Raylib e a linguagem de programação C++. O objetivo principal do jogo é controlar um carro enquanto o jogador enfrenta inimigos e tenta sobreviver por o maior tempo possível, acumulando pontos ao longo do percurso. O projeto explora conceitos de programação de jogos, como manipulação de objetos gráficos, controle de entrada do usuário e detecção de colisões. O jogo inclui elementos como a movimentação do carro, a interação com obstáculos na pista e a presença de inimigos que tornam o jogo mais desafiador. A dinâmica de jogo é estruturada em torno da evasão de obstáculos e da competição contra o tempo, mantendo o jogador engajado através do aumento de dificuldade conforme o progresso. O sistema de pontuação e de "highscore" é implementado, permitindo ao jogador verificar seu desempenho ao longo do jogo e competir por melhores resultados. O trabalho também sugere diversas melhorias para enriquecer a experiência do usuário, como a adição de mais níveis de dificuldade, a inclusão de uma loja de carros para futuras versões, melhorias gráficas e sonoras, e uma sincronização mais eficiente entre os efeitos sonoros e visuais, como a resposta do ronco do motor e os sons de colisão.

Palavras-chave: Raylib, C++, Corrida, Detecção de Colisões, Sistema de Pontuação.

ABSTRACT

This project describes the development of a 2D racing game, using the Raylib library and the C++ programming language. The main goal of the game is to control a car while the player faces enemies and tries to survive for as long as possible, accumulating points along the way. The project explores game development concepts such as manipulation of graphical objects, user input control, and collision detection. The game includes elements such as car movement, interaction with obstacles on the track, and the presence of enemies that make the game more challenging. The gameplay is structured around avoiding obstacles and competing against time, keeping the player engaged through increasing difficulty as progress is made. The scoring and "highscore" systems are implemented, allowing the player to track their performance throughout the game and compete for better results. The work also suggests several improvements to enrich the user experience, such as adding more difficulty levels, the inclusion of a car shop in future versions, graphic and sound enhancements, and more efficient synchronization between sound and visual effects, such as engine sounds and collision noises.

Keywords: Raylib, C++, Racing, Collision Detection, Scoring System.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

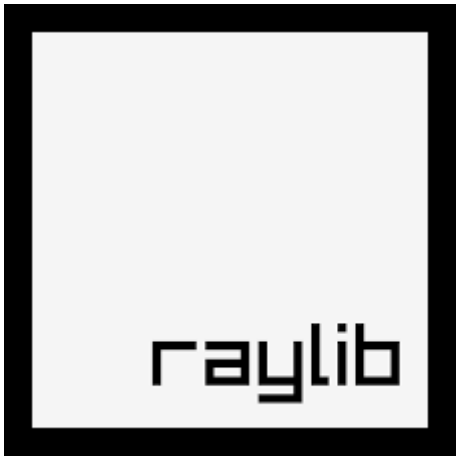

Figura 1 - Raylib

O desenvolvimento de jogos eletrônicos combina criatividade, tecnologia e lógica de programação, oferecendo um campo desafiador e dinâmico para estudantes e iniciantes na área. Este projeto tem como principal objetivo criar um jogo de corrida inspirado em clássicos de arcade, utilizando a linguagem C++ e a biblioteca gráfica Raylib.

A proposta do trabalho está centrada no aprendizado e aplicação prática de conceitos essenciais de programação orientada a objetos (POO), além da implementação de técnicas fundamentais no desenvolvimento de jogos, como manipulação gráfica, detecção de colisões e gerenciamento de eventos. A Raylib foi escolhida por ser uma biblioteca leve e acessível, facilitando a criação de elementos gráficos e sonoros com simplicidade.

Baseado no universo de corridas urbanas, o jogo desenvolvido apresenta um carro controlado pelo jogador, inimigos que aparecem na pista, obstáculos, sistema de pontuação e sons que trazem uma experiência de imersão para o jogador. Este projeto busca integrar aprendizado teórico e prático em uma experiência completa de desenvolvimento, aliando diversão e desafio.

Durante a construção do jogo, foram explorados conceitos como organização de classes, reaproveitamento de código, uso de estruturas como vetores e manipulação de arquivos para armazenamento de dados. Além disso, o projeto serviu como base para consolidar conhecimentos de lógica de programação e design de interação.

Este relatório detalha as etapas do desenvolvimento do jogo, desde a concepção e implementação até os testes finais, oferecendo uma visão crítica sobre os resultados obtidos e os conhecimentos adquiridos ao longo do processo.

1.1 Objetivo

O objetivo deste projeto é consolidar os conhecimentos adquiridos em programação, utilizando a linguagem C++ e o paradigma de orientação a objetos (POO). Além disso, o desenvolvimento busca aprimorar habilidades na criação de aplicações gráficas interativas, com o uso da biblioteca Raylib para construção de jogos 2D.

Os objetivos específicos incluem:

- **Desenvolvimento de um jogo de corrida**: Criar um jogo com temática de corrida urbana, onde o jogador controla um carro, enfrentando inimigos na pista, obstáculos e desafios. O jogo inclui sistemas de pontuação, vida, e mecânicas de colisão.
- **Aprofundar o conhecimento na linguagem C++:** Aplicar os conceitos da linguagem, como manipulação de vetores, estruturas condicionais e laços, além do uso de arquivos para salvar e carregar dados, como o highscore.
- **Praticar conceitos de orientação a objetos:** Estruturar o projeto de forma modular, implementando classes para representar os diferentes componentes do jogo (carros, inimigos, obstáculos, etc.). O uso de métodos e atributos encapsulados promove uma melhor organização do código e facilita a manutenção.
- **Aprender a utilizar a biblioteca Raylib:** Explorar os recursos de Raylib, incluindo renderização de imagens, detecção de colisão, manipulação de sons e gerenciamento de entrada do usuário.

- **Desenvolver uma aplicação com foco em usabilidade e diversão:** Criar um jogo funcional e envolvente, que apresente uma experiência fluida ao usuário, com elementos de desafio progressivo e recompensas (como aumento de pontuação e exibição do highscore).

Este projeto também serve como base para futuros desenvolvimentos, tanto no aprimoramento das mecânicas já implementadas quanto na adição de novas funcionalidades. É uma oportunidade de aplicar conceitos teóricos em um contexto prático e criativo, contribuindo para o aprendizado contínuo na área de programação de jogos.

1.2 Justificativa

A escolha deste tema, que envolve o desenvolvimento de um jogo de corrida inspirado em clássicos de arcade, é motivada por diversos aspectos que abrangem tanto interesses pessoais quanto objetivos educacionais relacionados ao aprendizado da programação e ao domínio de ferramentas específicas para o desenvolvimento de jogos. O gênero de corrida, com mecânicas de obstáculos e inimigos na pista, oferece um ambiente dinâmico e desafiador para explorar uma ampla gama de conceitos fundamentais de desenvolvimento de jogos, como controle de objetos, detecção de colisões e sistemas de pontuação.

A adoção desse tema se justifica, em primeiro lugar, pela oportunidade de aplicar conhecimentos adquiridos em programação orientada a objetos (POO) em um contexto prático, utilizando a linguagem C++ e a biblioteca Raylib.

1.3 Aporte Teórico

O desenvolvimento de jogos eletrônicos envolve a integração de diversos conhecimentos, como programação, design gráfico, interação do usuário e áudio. Este projeto, focado no desenvolvimento de um jogo 2D utilizando C++ e a biblioteca Raylib, visa explorar aspectos importantes da criação de jogos, como a programação orientada a objetos, o uso de ferramentas eficientes para a criação de jogos simples e a construção de uma experiência imersiva para o jogador.

- **Linguagem C++ no Desenvolvimento de Jogos:** A linguagem C++ é amplamente utilizada no desenvolvimento de jogos devido à sua alta performance e controle sobre os recursos do sistema. Esse controle direto sobre a memória é crucial em jogos de alto desempenho, como os jogos de ação e corrida, que exigem a otimização dos recursos. A linguagem também oferece suporte à programação orientada a objetos, o que permite a organização modular do código, essencial para o desenvolvimento de jogos complexos.

  Em jogos como o proposto, a programação orientada a objetos facilita a criação de classes que representam entidades do jogo, como o carro do jogador, inimigos e obstáculos na pista. Cada classe pode ser responsável por comportamentos específicos, o que organiza e torna o código mais eficiente e de fácil manutenção.
- **Programação Orientada a Objetos (POO):** A Programação Orientada a Objetos (POO) é um paradigma que organiza o código em torno de "objetos", instâncias de "classes" que possuem atributos e métodos. No contexto de jogos, a POO permite modelar as diversas entidades, como o carro, os inimigos e obstáculos, de maneira estruturada, facilitando sua manipulação e interação.

  Por exemplo, neste jogo, a classe Carro representa o carro do jogador, enquanto a classe Inimigo modela os carros inimigos e a classe Obstaculo representa os obstáculos na pista. Esses objetos interagem por meio de métodos específicos, como o movimento e a detecção de colisões, o que torna a estrutura do jogo mais flexível e escalável.
- **Biblioteca Raylib:** Raylib é uma biblioteca de desenvolvimento de jogos simples e eficiente, que facilita a criação de jogos 2D e 3D. Sua API foi projetada para ser acessível, permitindo que os desenvolvedores se concentrem na lógica do jogo em vez de lidar com detalhes complexos de baixo nível. Raylib oferece funções para manipulação gráfica, entrada do usuário e áudio, tornando-se uma ferramenta poderosa para criar jogos interativos.

  Neste projeto, Raylib é utilizada para gerenciar gráficos, como o controle de sprites e texturas, e para lidar com a entrada do usuário por meio do teclado e mouse. Além disso, a biblioteca fornece funcionalidades para a detecção de colisões, essencial para a interação realista entre os carros e obstáculos do jogo.

- **Design de Jogos:** O design de jogos envolve a criação de mecânicas que garantem uma experiência divertida e desafiadora para o jogador. Em um jogo como o proposto, o jogador controla um carro e deve desviar de obstáculos e inimigos, enquanto o jogo avalia a pontuação e as vidas restantes. A definição de regras claras, como a perda de vidas ao colidir com inimigos ou obstáculos, é essencial para a jogabilidade.

  Além disso, o design de jogos envolve a criação de uma experiência visual e sonora imersiva. O uso de gráficos atrativos e efeitos sonoros dinâmicos contribui para o ambiente do jogo, tornando-o mais envolvente. A música de fundo e os efeitos de colisão, por exemplo, são recursos que ajudam a manter o jogador imerso na experiência.
- **Mecânicas de Colisão e Inteligência Artificial:** A mecânica de colisão é crucial para garantir a interação entre os objetos do jogo de forma realista. A biblioteca Raylib oferece funções que facilitam a detecção de colisões entre objetos retangulares, permitindo que o jogo reaja corretamente a eventos como colisões entre o carro do jogador e os inimigos ou obstáculos.

  A inteligência artificial (IA) dos inimigos, embora simples, é um componente importante. Os carros inimigos se movem de forma coordenada e buscam barrar a passagem do carro do jogador, criando um desafio contínuo. A movimentação dos inimigos e a detecção de colisões tornam o jogo mais dinâmico e interessante.

2 METODOLOGIA

A metodologia empregada neste trabalho foi fundamentada em uma combinação de pesquisa bibliográfica e aplicação prática dos conhecimentos adquiridos, com o objetivo de desenvolver um jogo 2D de corrida urbana, utilizando a biblioteca Raylib. Através da utilização da Programação Orientada a Objetos (POO) e da biblioteca mencionada, foi possível criar um sistema modular e eficiente para o gerenciamento das diversas funcionalidades do jogo, como movimentação, detecção de colisões, pontuação, interação com obstáculos e controle dos inimigos.

A primeira etapa do desenvolvimento envolveu uma pesquisa bibliográfica sobre a biblioteca Raylib. A pesquisa anterior, entregue para a disciplina de Programação, foi essencial para a fundamentação do uso da biblioteca, permitindo o entendimento de suas funcionalidades, como a renderização de gráficos, controle de entrada do usuário e manipulação de áudio. Essa pesquisa serviu como base para o estudo da documentação oficial da Raylib, que foi fundamental para a integração da biblioteca ao projeto. Além disso, a documentação ofereceu insights sobre a utilização de funções como LoadTexture, DrawRectangle, PlayMusicStream e CheckCollisionRecs, as quais foram aplicadas diretamente no desenvolvimento do jogo, oferecendo uma estrutura gráfica simples, mas eficiente. Por fim, foi realizada a instalação das ferramentas para o desenvolvimento.

2.1 Ferramentas Utilizadas

O desenvolvimento deste projeto foi realizado utilizando um conjunto de ferramentas poderosas e amplamente utilizadas para a criação de jogos 2D. A seguir, será apresentada uma descrição detalhada de cada uma dessas ferramentas: Raylib, C++, GCC Compiler e VSCode.

2.1.1 Raylib

Raylib é uma biblioteca gráfica de código aberto desenvolvida em C, projetada para facilitar a criação de jogos e aplicações multimídia. Ela fornece uma série de funções simples e eficientes para trabalhar com gráficos 2D e 3D, áudio e entrada do usuário. Raylib é particularmente popular entre iniciantes devido à sua sintaxe intuitiva e à vasta documentação, o que torna a criação de jogos acessível e sem a complexidade de outras bibliotecas mais robustas. No contexto deste projeto, a Raylib foi utilizada para lidar com as operações de renderização gráfica, como o desenho de objetos na tela, manipulação de sons e músicas, e o gerenciamento de eventos de entrada do usuário, como o movimento da nave e o disparo de lasers. Além disso, a biblioteca também foi responsável pela detecção de colisões, uma funcionalidade essencial para o funcionamento do jogo. A utilização da Raylib facilitou o desenvolvimento do jogo, proporcionando ferramentas prontas para uso e reduzindo a necessidade de escrever código complexo para tarefas comuns.

2.1.2 C++

C++ é uma das linguagens de programação mais populares e poderosas, amplamente utilizada no desenvolvimento de jogos, sistemas de alto desempenho e aplicações que exigem manipulação direta de hardware. A escolha do C++ para este projeto foi motivada pela sua performance e pela facilidade com que se pode trabalhar com a Programação Orientada a Objetos (POO), um paradigma que facilita a modularização e a organização do código. A utilização de C++ permitiu que o desenvolvimento do jogo fosse realizado de forma estruturada, com a criação de classes que encapsulam as diversas entidades do jogo, como o carro do jogador, os carros inimigos, os obstáculos e as faixas de corrida. Além disso, o C++ também foi fundamental para a implementação de funcionalidades avançadas de controle de fluxo, como a verificação de colisões e a manipulação de eventos de entrada, essenciais para uma jogabilidade dinâmica e fluida.

2.1.3 GCC Compiler

O GCC (GNU Compiler Collection) é um compilador de código aberto que suporta diversas linguagens de programação, incluindo C e C++. Ele é amplamente utilizado na comunidade de desenvolvedores devido à sua robustez, performance e capacidade de compilar código para diferentes plataformas. O GCC Compiler foi utilizado neste projeto para compilar o código C++ e gerar o arquivo executável do jogo. Sua integração com sistemas operacionais como Linux, Windows e macOS garantiu que o projeto fosse facilmente portável entre diferentes plataformas, sem a necessidade de ajustes significativos no código. O GCC também possui recursos avançados de otimização, permitindo que o jogo seja executado de forma eficiente, mesmo em máquinas com recursos limitados.

2.1.4 Visual Studio Code

O Visual Studio Code (VSCode) é um editor de código-fonte leve, mas poderoso, que oferece suporte a uma ampla variedade de linguagens de programação, incluindo C++.

2.1.5 Google Gemini

O Gemini foi uma ferramenta crucial no processo de criação das pixel arts do jogo. Foi utilizado para gerar todas as imagens e sprites do jogo, incluindo os carros, inimigos e imagens de fundo. Com a sua capacidade de criar e editar pixel art de maneira intuitiva e eficiente, o Gemini facilitou a criação de gráficos personalizados para o jogo, alinhando-se perfeitamente ao estilo visual desejado. A ferramenta permitiu que os gráficos fossem facilmente ajustados e refinados, garantindo um design visual coeso e dinâmico que contribuiu significativamente para a estética do jogo.

2.1.6 Epidemic Sound

Para os efeitos sonoros e a música de fundo, o Epidemic Sound foi utilizado. Esta plataforma forneceu uma vasta biblioteca de sons livres de copyright, essenciais para garantir que o jogo tivesse uma trilha sonora de alta qualidade e que os efeitos sonoros fossem únicos, sem preocupações com direitos autorais. A pesquisa no Epidemic Sound permitiu escolher trilhas e sons que se encaixam perfeitamente na atmosfera do jogo, contribuindo para a imersão do jogador.

Essas ferramentas, combinadas com o uso de conceitos de Programação Orientada a Objetos (POO), foram fundamentais para o desenvolvimento e sucesso do projeto, proporcionando tanto a base técnica quanto a parte visual e sonora envolvente do jogo.

2.2 Desenvolvimento

A segunda etapa foi a aplicação prática do conhecimento adquirido. Para isso, foi necessário planejar a estrutura do jogo, identificando os elementos principais, como o carro do jogador, os inimigos, os obstáculos e as faixas de corrida. A escolha de Programação Orientada a Objetos foi decisiva para modularizar o código, tornando-o mais legível e flexível. Cada componente do jogo foi implementado como uma classe independente, o que permitiu a criação de objetos com responsabilidades bem definidas e a implementação de interações entre eles.

A seguir, será apresentada uma descrição detalhada das principais classes, arquivos e suas funcionalidades dentro do jogo:

- **Main:** É o principal responsável pela lógica central do jogo, funcionando como o controlador geral das mecânicas e das interações entre os objetos. Entre suas funções principais, destacam-se a inicialização dos elementos do jogo, a atualização do estado do jogo a cada quadro, a renderização dos objetos na tela e o tratamento das entradas do jogador.

  Adicionalmente, o arquivo é responsável por iniciar o jogo, verificar as colisões entre os diversos elementos do jogo, calcular a pontuação e gerenciar as vidas do jogador. Dentro dela, a função AtualizarScore() é utilizada para calcular a pontuação do jogador, com base no tempo que ele permanece vivo, multiplicando o tempo por 5. A função SalvarRecorde() é chamada quando o jogo termina, salvando a pontuação e o nome do jogador no arquivo recordes.txt. Já a função CarregarRecordes() carrega os melhores recordes salvos no arquivo, organizando-os de forma que os 10 maiores scores sejam exibidos.

  A função DesenharTelaGameOver() renderiza a tela de Game Over, exibindo a pontuação e um botão para retornar ao menu principal. Além disso, a classe Game também implementa as funções de atualização do estado do jogo (Atualizar()) e renderização dos componentes gráficos (Desenhar()), além de controlar as colisões entre os elementos do jogo e a reinicialização do jogo através dos métodos SalvarRecorde() e CarregarRecordes().

  Este arquivo também gerencia o fundo da pista e o movimento de rolagem. O método Atualizar() atualiza o movimento do fundo da pista, criando a ilusão de que o jogador está avançando. Quando o fundo atinge o final da tela, ele é reiniciado. A função Desenhar() renderiza o fundo da pista e as faixas de limite, mantendo a visualização contínua e dinâmica durante o jogo.

  A função DesenharBotaoArredondado() é utilizada para desenhar botões com bordas arredondadas na tela, como no menu principal e na tela de Game Over. Ela também trata os eventos de clique do mouse, permitindo que o jogador interaja com os botões para iniciar o jogo ou retornar ao menu.

  A função DesenharMenu() renderiza o menu principal, exibindo o nome do jogador (caso tenha sido inserido) e os botões de "Jogar" e "Recordes". Ela lida com a interação do mouse, permitindo que o jogador inicie o jogo ou acesse a tela de recordes.

Por fim, a função DesenharEntradaNome() exibe a tela onde o jogador pode inserir seu nome antes de iniciar o jogo. Nessa tela, o jogador pode digitar seu nome e, ao clicar no botão "OK", ele avança para o jogo.

- **Jogador:** é responsável pela movimentação do carro do jogador na pista. A função CarregarTextura() carrega a imagem que representa o carro, e o método Atualizar() lida com a movimentação do carro, controlada pelas teclas pressionadas pelo jogador (setas ou teclas WASD). Além disso, o método Desenhar() é responsável por renderizar o carro na tela com a textura carregada, e o método GetRetangulo() retorna um retângulo que representa a área do carro, utilizado para detectar colisões com outros objetos, como inimigos e obstáculos.
- **Inimigo:** A classe Inimigo representa os carros inimigos que aparecem na pista e que o jogador deve evitar. O método Atualizar() atualiza a posição do inimigo com base em sua velocidade, enquanto o método Desenhar() renderiza o inimigo na tela. A função ForaDaTela() verifica se o inimigo saiu da tela e, caso sim, ele é removido. O método VisivelNaTela() garante que apenas os inimigos que ainda estão visíveis na tela sejam considerados para colisões, e o método GetRetangulo() retorna o retângulo de colisão do inimigo, utilizado para detectar se houve alguma colisão com o carro do jogador.
- **GeradorInimigos:** A classe GeradorInimigos é responsável por criar inimigos de forma aleatória na pista. A função CarregarTexturas() carrega as texturas dos inimigos, organizadas por tipos de inimigos. O método Atualizar() verifica o tempo de spawn e, quando o tempo é atingido, cria novos inimigos nas faixas disponíveis da pista. A função Desenhar() renderiza todos os inimigos na tela, enquanto o método Resetar() limpa todos os inimigos gerados e reinicia o contador de tempo de spawn.

A interação entre essas classes é essencial para o funcionamento do jogo. O Main gerencia a criação e atualização das instâncias das classes Jogador, Inimigo, GeradorInimigos, garantindo a interação entre elas e o bom funcionamento da dinâmica do jogo. A utilização de objetos de diferentes classes permite modularizar as funcionalidades, facilitando a manutenção e a expansão do código.

2.3 Estudos Realizados

A pesquisa também envolveu o estudo de técnicas de acesso a arquivos de texto, como a leitura e escrita de arquivos para salvar o highscore do jogo. A função SalvarRecorde() foi implementada com base em conceitos aprendidos sobre manipulação de arquivos em C++, permitindo que o jogador consiga registrar a maior pontuação alcançada. Essa função garante que, ao término de cada partida, a pontuação do jogador seja armazenada corretamente e, no caso de um novo recorde, seja atualizado no arquivo “recordes.txt”.

Adicionalmente, foi realizada uma pesquisa sobre o design de jogos, focando na criação de uma experiência desafiadora e divertida. O equilíbrio entre a velocidade do carro do jogador, o tempo de spawn dos inimigos e a dificuldade dos obstáculos foi ajustado através de testes contínuos durante o desenvolvimento. As mudanças nos parâmetros, como a velocidade de rolagem do fundo e a quantidade de inimigos gerados, foram feitas com base no feedback da jogabilidade para garantir que o jogo fosse dinâmico e interessante, sem se tornar frustrante para o jogador.

Por fim, a integração de todas as classes e funcionalidades foi realizada com testes contínuos e ajustes, de modo a garantir o correto funcionamento da detecção de colisões, movimentação do jogador, e renderização dos elementos na tela. O processo de depuração e otimização foi conduzido utilizando as ferramentas de debugging da IDE, além de observações em tempo real da execução do jogo. Esses testes ajudaram a identificar e corrigir problemas de desempenho, como a taxa de atualização dos gráficos e a gestão de eventos do teclado, que são cruciais para uma experiência fluida.

Em resumo, a metodologia aplicada envolveu uma combinação de pesquisa bibliográfica e desenvolvimento prático, utilizando a biblioteca Raylib como ferramenta central. A pesquisa anterior sobre a Raylib, juntamente com a aplicação dos conceitos de POO, foi fundamental para o sucesso do desenvolvimento do jogo. Além disso, as fontes de pesquisa forneceram os fundamentos necessários para a criação de um jogo funcional, intuitivo e agradável de jogar, proporcionando uma experiência imersiva ao usuário.

3 RESULTADOS OBTIDOS

Os resultados obtidos neste projeto são demonstrados por meio de capturas de tela (prints) do jogo em funcionamento. Essas imagens ilustram os diferentes aspectos e etapas do jogo, incluindo a interface gráfica, as interações do jogador e as dinâmicas de colisões e movimentação. O fluxo do jogo, desde a inicialização até o encerramento, é visível nas capturas, proporcionando uma visão detalhada de como o sistema de pontuação, a interação com os inimigos e a destruição dos obstáculos são realizados.

Os prints também mostram os momentos em que o jogador interage com o jogo, como quando o carro do jogador se move pela pista, desvia dos inimigos. A tela de "Game Over" e as transições de fase podem ser observadas, fornecendo uma visão clara do estado do jogo em diferentes pontos de progresso. A pontuação do jogador e o momento de colisão entre o carro do jogador e os inimigos também são exibidos nas capturas, permitindo uma compreensão visual das dinâmicas do jogo durante a execução.

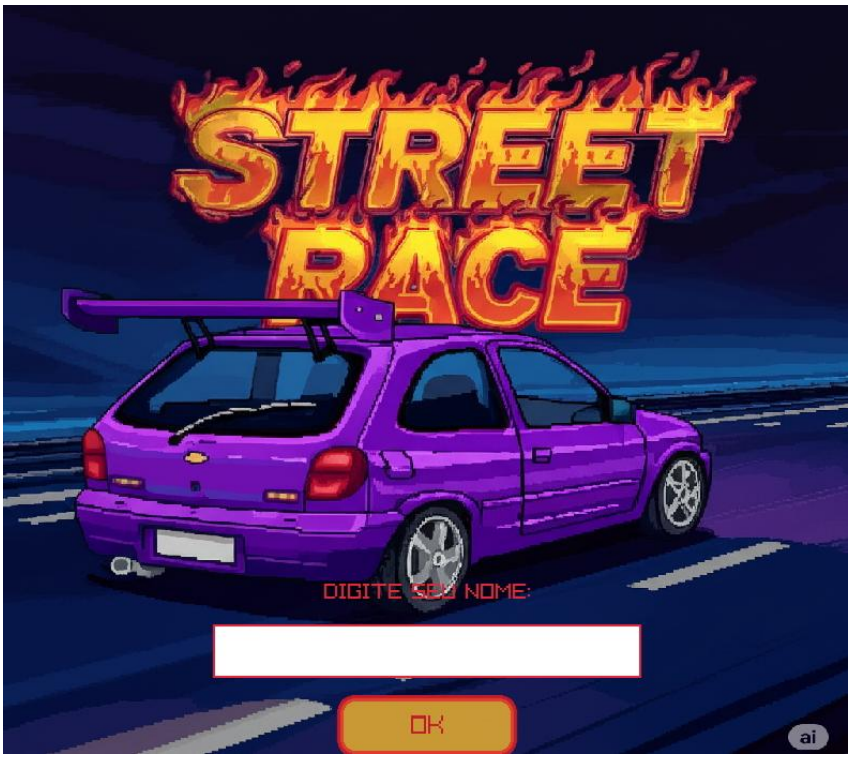

Figura 2 – Jogo Início

Figura 3 – Tela de Recordes

Figura 4 – Menu Principal

Figura 5 – Jogo Iniciado

Figura 6 – Score do Jogador

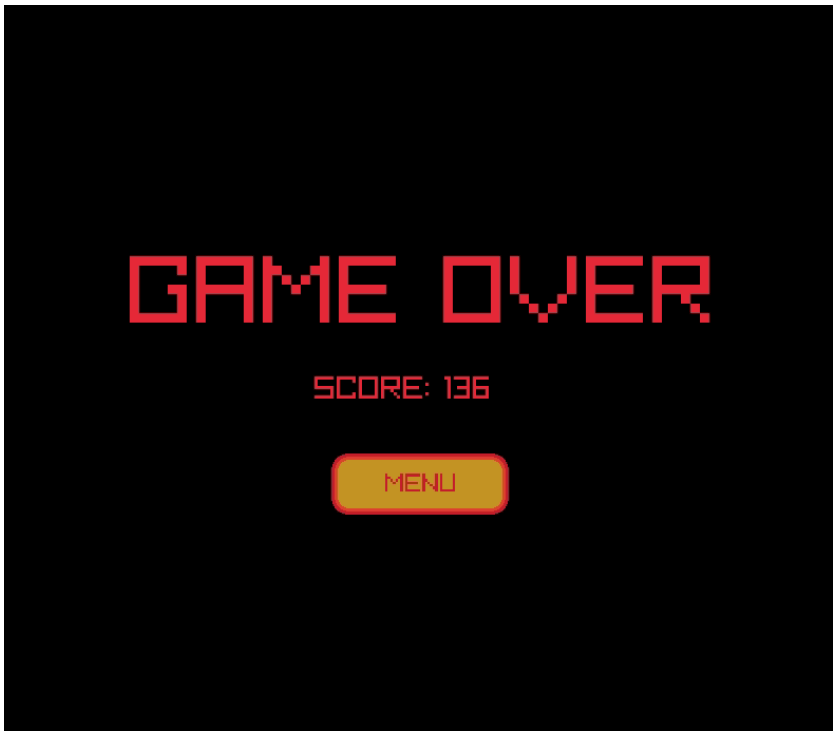

Figura 7 – Tela de Game Over

4 CONCLUSÃO

O projeto do jogo desenvolvido oferece uma experiência divertida e desafiadora, proporcionando uma interação dinâmica entre o jogador e os elementos do jogo, como o carro do jogador, os inimigos na pista, e a pontuação. A utilização da biblioteca Raylib garantiu uma implementação gráfica e sonora eficaz, permitindo que os elementos visuais do jogo, como a pista, os carros e o fundo, fossem renderizados de maneira fluida e envolvente. Além disso, a biblioteca facilitou a manipulação de sons e a entrada do usuário, criando uma experiência interativa completa. O código em C++ possibilitou um controle preciso dos elementos do jogo, desde a movimentação do carro até a detecção de colisões, permitindo ajustes finos na jogabilidade.

No entanto, como em qualquer projeto de desenvolvimento, há sempre oportunidades para melhorias.

4.1 Melhorias na Sincronização dos Efeitos Sonoros

Um ponto que poderia ser aprimorado é a sincronização dos efeitos sonoros. Embora os sons de buzina, colisões e o ronco do motor estejam presentes, poderia

haver uma melhoria no tempo de resposta entre os efeitos sonoros e as ações no jogo.

Por exemplo, o som do ronco do motor poderia ser mais distinto e reagir de forma mais sensível à aceleração do carro, criando uma sensação mais imersiva de velocidade. Além disso, o som das colisões, tanto com os inimigos quanto com os obstáculos, poderia ser mais impactante, aumentando a intensidade da experiência quando o jogador bate no trânsito ou em outros carros.

A adição de transições sonoras mais suaves entre os diferentes estados do jogo, como a tela de "Game Over" e o retorno ao menu, também ajudaria a criar uma experiência auditiva mais fluida.

4.2 Inclusão de Novos Níveis e Aumento de Complexidade

Outro ponto que pode ser melhorado é a inclusão de múltiplos níveis de dificuldade, o que proporcionaria ao jogador uma experiência mais variada e desafiadora. Atualmente, o jogo segue uma estrutura linear, onde os inimigos aparecem conforme o tempo.

A introdução de novos níveis, com diferentes tipos de inimigos e obstáculos, poderia aumentar a longevidade e o apelo do jogo. Por exemplo, em níveis mais difíceis, os carros inimigos poderiam acelerar mais rápido ou começar a realizar movimentos mais erráticos, desafiando o jogador a melhorar suas habilidades.

Além disso, ao finalizar um nível, o jogador poderia ser recompensado com um aumento na dificuldade, com novos carros inimigos mais rápidos e maiores, criando uma progressão contínua e uma maior motivação para continuar jogando.

4.3 Implementação de uma Loja de Carros com Sistema de Coins

Uma adição interessante para aumentar a imersão e o fator de progressão do jogo seria a implementação de uma loja de carros dentro do jogo. A loja permitiria que o jogador comprasse novos carros, com diferentes características, como velocidade, aceleração e manobrabilidade, usando coins adquiridos ao completar as fases do

jogo. Cada fase poderia recompensar o jogador com uma quantidade de coins dependendo da sua performance, como o tempo de conclusão, o número de inimigos derrotados ou o número de obstáculos evitados.

A loja poderia ter uma variedade de carros, incluindo modelos nacionais, com aparência e desempenho distintos. O jogador teria a possibilidade de escolher entre diferentes carros, cada um com suas características únicas que poderiam influenciar sua experiência de jogo, como carros mais rápidos ou mais resistentes. Além disso, a loja poderia ter um sistema de desbloqueio progressivo, onde os carros mais poderosos ou exclusivos ficariam disponíveis conforme o jogador avançasse nas fases e acumulasse mais coins.

A adição de uma loja de carros não só adicionaria uma camada de personalização ao jogo, permitindo que o jogador escolhesse o veículo que mais se adequa ao seu estilo de jogo, mas também proporcionaria um incentivo adicional para completar as fases, criando uma motivação extra para avançar no jogo e conquistar recompensas. Essa funcionalidade poderia ser acessada diretamente do menu principal ou entre as fases, oferecendo ao jogador a chance de melhorar sua performance com novos carros antes de enfrentar desafios mais difíceis.

4.4 Melhoria no Design Visual dos Elementos

Embora os gráficos básicos do jogo, como os carros e obstáculos, estejam adequadamente implementados, existe espaço para melhorias estéticas. O design visual dos elementos poderia ser mais detalhado, com texturas mais refinadas e animações mais fluidas. O carro do jogador, por exemplo, poderia ter animações mais detalhadas ao se mover, como a simulação de aceleração e desaceleração, e até efeitos visuais ao fazer curvas mais apertadas.

As colisões também poderiam ser melhoradas, com animações de destruição mais complexas, que tornariam os efeitos visuais mais impactantes, como quando o carro bate em um obstáculo ou inimigo. Além disso, a inclusão de efeitos visuais ao interagir com objetos no jogo, como partículas de poeira ou fumaça quando o carro derrapa, poderia tornar a jogabilidade mais envolvente.

Tais adições estéticas ajudariam a criar uma sensação mais intensa de ação e recompensariam o jogador visualmente, contribuindo para uma experiência mais satisfatória.

4.5 Expansão da Trilha Sonora

A trilha sonora do jogo, embora adequada para a temática de corrida, poderia ser expandida para incluir mais variações e faixas que mudassem de acordo com o andamento do jogo. Por exemplo, a música de fundo poderia mudar para um tema mais intenso e acelerado quando o jogador estiver se aproximando de um obstáculo ou no final de um nível, criando uma sensação de urgência e tensão. A diversidade de músicas ajudaria a manter o jogador mais imerso, proporcionando uma sensação de progressão e dinamismo no jogo. A música poderia também variar dependendo do nível de dificuldade, com músicas mais aceleradas para níveis mais difíceis e mais tranquilas para níveis mais fáceis.

4.6 Considerações Finais

Em resumo, o jogo desenvolvido oferece uma boa base para uma experiência de corrida envolvente, com mecânicas interessantes e um desempenho eficiente. No entanto, como sugerido, existem várias áreas de melhoria, tanto no aspecto técnico quanto estético, que poderiam enriquecer significativamente a experiência do jogador. A implementação de novos níveis de dificuldade, a adição de uma loja de carros e efeitos sonoros mais sincronizados são apenas algumas das sugestões que, se aplicadas, poderiam transformar o jogo em uma experiência ainda mais completa e imersiva. Essas melhorias contribuiriam para aumentar a longevidade e o apelo do jogo, tornando-o mais desafiador, dinâmico e visualmente interessante para os jogadores.

REFERÊNCIAS

Terminal Root. **"Crie Jogos para Windows, Linux e Web com Raylib C/C++".** Disponível em: https://www.youtube.com/watch?v=LZUMVdkWWrg . Acesso em: 1 maio 2025.

DOCUMENTAÇÃO oficial da RAYLIB. Disponível em: https://raylib.com/cheatsheet.html . Acesso em: 12 mai. 2025.

RAYLIB GitHub Repository. Disponível em: https://github.com/raysan5/raylib . Acesso em: 2 mai. 2025.

RAYLIB Wiki. Disponível em: https://github.com/raysan5/raylib/wiki . Acesso em: 5 mai. 2025.

RAYLIB Examples Repository. Disponível em: https://github.com/raysan5/raylib/tree/master/examples . Acesso em: 7 mai. 2025.

DISCORD Raylib Community. Disponível em: https://discord.gg/r . Acesso em: 29 mai. 2025.

STACK Overflow. "Raylib Tag". Disponível em: https://stackoverflow.com/questions/tagged/raylib . Acesso em: 6 jun. 2025.

GAMEDEV.NET Fórum. "Discussions on Raylib". Disponível em: https://www.gamedev.net/forums/ . Acesso em: 30 mai. 2025.

Raylib Game Programming Tutorial and Reforcing Refs. Playlist de tutoriais no YouTube. Disponível em: https://www.youtube.com/playlist?list=PL-NSm9WUHXM-FEpA3qto28UozmIxsZPX27 . Acesso em: 21 mai. 2025.

Programming With Nick. "**How to install raylib with C++ on Windows and use it with Visual Studio Code**". Vídeo no YouTube. Disponível em: https://www.youtube.com/watch?v=PaAcVk5jUd8 . Acesso em: 1 mai. 2025.