INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

LUIS HENRIQUE DE MELO MARTINS

JOGO DA COBRINHA

CAMPOS DO JORDÃO 2024

RESUMO

O presente documento descreve o desenvolvimento de um jogo digital da cobrinha, implementado em C++ utilizando a biblioteca gráfica Raylib. O objetivo principal do projeto foi recriar o clássico jogo da cobrinha, onde o jogador controla uma cobra que cresce ao consumir alimentos, devendo evitar colisões com as bordas da tela e com o próprio corpo para continuar avançando. A metodologia utilizada consistiu na aplicação de conceitos básicos de programação orientada a objetos, manipulação de gráficos bidimensionais e interação com o usuário via controles do teclado. Foram desenvolvidas funções para controlar o movimento da cobrinha, detectar colisões e gerenciar o estado do jogo. Ao final, o jogo alcançou um nível satisfatório de funcionalidade e jogabilidade, permitindo que o jogador possa iniciar, escolher a dificuldade e encerrar a partida de forma intuitiva. Conclui-se que a utilização de Raylib facilitou a implementação dos recursos gráficos e de interação, tornando o desenvolvimento ágil e acessível, especialmente para jogos simples.

Palavras-Chave: Jogo da cobrinha; Raylib; C++; Desenvolvimento de jogos; Programação gráfica.

ABSTRACT

This document describes the development of a snake game, implemented in C++ using the Raylib graphics library. The main objective of the project was to recreate the classic snake game, where the player controls a snake that grows when consuming food, and must avoid collisions with the edges of the screen and with his own body to continue advancing. The methodology used consisted of applying basic concepts of object-oriented programming, manipulating two-dimensional graphics and interacting with the user via keyboard controls. Functions were developed to control the snake's movement, detect collisions and manage the game state. In the end, the game reached a satisfactory level of functionality and playability, allowing the player to start, choose the difficulty and end the game intuitively. It is concluded that the use of Raylib facilitated the implementation of graphic and interaction resources, making development agile and accessible, especially for simple games.

Keywords: Snake game; Raylib; C++; Game development; Graphic programming.

SUMÁRIO

1 INTRODUÇÃO

Este documento apresenta o desenvolvimento de um jogo digital inspirado no clássico “Jogo da Cobrinha”, elaborado em C++ com a biblioteca gráfica Raylib. A ideia de fazer esse jogo surgiu como uma oportunidade de consolidar conceitos de programação, especialmente no controle de gráficos bidimensionais e na criação de um sistema interativo de jogabilidade, além de colocar em prática alguns conceitos vistos em sala de aula. O projeto propõe uma experiência lúdica e funcional, com um desafio de programação que alia lógica e criatividade.

O objetivo principal foi desenvolver uma aplicação que simulasse o funcionamento do jogo original, permitindo ao jogador controlar uma cobrinha que cresce ao consumir alimentos e deve evitar colisões para permanecer no jogo. A implementação incluiu a criação de uma interface gráfica simples, tratamento de eventos de teclado e a programação de regras de colisão. Esses aspectos permitiram a criação de uma estrutura de jogo completa e funcional.

A organização deste documento segue uma estrutura lógica, abordando inicialmente os conceitos e o planejamento do desenvolvimento, passando pela metodologia empregada, pelos detalhes de implementação e, por fim, pelos resultados e conclusões obtidos.

1.1 Objetivos

Este trabalho tem por objetivo desenvolver um jogo digital da cobrinha utilizando a biblioteca Raylib, com foco na aplicação prática de conceitos de programação gráfica e interação em C++. O projeto visa recriar uma versão funcional, simples e intuitiva do jogo, proporcionando ao jogador uma experiência lúdica e desafiadora.

Para a consecução deste objetivo foram estabelecidos os seguintes objetivos específicos:

- Investigar as características e mecânicas dos jogos da cobrinha disponíveis atualmente, com o intuito de identificar funcionalidades essenciais e diferenciais relevantes;

- Propor uma estrutura de código modular e organizada, que permita o controle do movimento da cobrinha, a detecção de colisões e a geração de alimentos de forma eficiente;
- Implementar uma interface gráfica simples e interativa, usando a biblioteca Raylib, garantindo uma experiência de usuário fluida e acessível;
- Documentar todas as etapas do desenvolvimento comentando o código inteiro, descrevendo o processo de criação e as soluções aplicadas.

1.2 Justificativa

A relevância deste trabalho se fundamenta na importância de consolidar conceitos de programação e desenvolvimento de jogos em um contexto prático, proporcionando aos alunos um modelo de aplicação de conhecimentos em um projeto real. A elaboração de um jogo da cobrinha em C++ com a biblioteca Raylib permite explorar técnicas de programação gráfica, manipulação de eventos e lógica de colisão, conteúdos essenciais para o desenvolvimento de software interativo, além de motivações pessoais que incentivaram na criação desse jogo em específico.

1.3 Aspectos Metodológicos

O presente trabalho adotou uma metodologia prática e experimental. A implementação foi realizada com a biblioteca Raylib, que ofereceu suporte para a criação de uma interface gráfica simplificada e o controle de eventos do jogo. O desenvolvimento do projeto seguiu uma abordagem iterativa, com testes constantes para validação das funcionalidades implementadas, tais como o movimento da cobrinha, a detecção de colisões e o gerenciamento dos estados do jogo. Essa abordagem permitiu ajustes contínuos e a resolução de problemas técnicos de forma ágil.

2 METODOLOGIA

A metodologia utilizada no desenvolvimento deste trabalho baseou-se em um ciclo iterativo de planejamento, implementação, teste e ajustes. O objetivo foi recriar o clássico jogo da cobrinha utilizando ferramentas modernas de programação e desenvolvimento de software, proporcionando uma aplicação prática dos conceitos aprendidos no curso. A seguir, detalham-se as considerações iniciais, as ferramentas utilizadas e a descrição do projeto desenvolvido.

2.1 Considerações Iniciais

O projeto teve como objetivo principal criar um jogo digital funcional e intuitivo que seguisse as principais mecânicas do jogo da cobrinha. Como parte do planejamento, foi estabelecido que o jogo incluiria funcionalidades básicas, como controle do movimento da cobrinha por meio do teclado, crescimento da cobrinha ao consumir alimentos, e fim de jogo em caso de colisão com o próprio corpo ou com as bordas da tela. Além disso, o projeto buscou implementar uma interface gráfica simples, proporcionando uma experiência acessível ao jogador.

A principal motivação para a escolha deste tema foi a oportunidade de consolidar conhecimentos em programação gráfica e manipulação de eventos, além de explorar ferramentas voltadas ao desenvolvimento de jogos.

2.1 Ferramentas Utilizadas

O desenvolvimento do projeto utilizou as seguintes ferramentas e tecnologias:

- **Linguagem de programação**: C++, escolhida por conta de ser abordada em sala de aula e pela compatibilidade com a biblioteca Raylib.
- **Biblioteca gráfica**: Raylib, uma biblioteca de código aberto voltada para o desenvolvimento de jogos 2D e 3D, utilizada para criar a interface gráfica e manipular eventos.
- **Ambiente de desenvolvimento**: Visual Studio Code, que ofereceu suporte para edição de código, depuração e integração com compiladores.
- **Compilador**: g++, integrado ao Visual Studio Code, para compilar o código em C++.

- **Template**: Foi utilizado um template gratuito para uma melhor integração da Raylib com o Visual Studio Code.
- **Sistema operacional**: Windows, como plataforma de desenvolvimento e execução do jogo.

2.2 Descrição do Projeto

O jogo da cobrinha desenvolvido neste projeto segue as mecânicas clássicas do gênero. O jogador controla a cobrinha usando as teclas direcionais, com o objetivo de consumir alimentos gerados aleatoriamente na tela. A cada alimento consumido, a cobrinha cresce, aumentando o nível de dificuldade do jogo. O jogo termina se a cobrinha colidir com as bordas da tela ou com seu próprio corpo.

O projeto foi estruturado de forma modular, com funções específicas para controle do movimento da cobrinha, geração e reposicionamento dos alimentos, e detecção de colisões. Além disso, foi implementado um sistema de estados que permite alternar entre a tela inicial (menu), o jogo em execução e a tela de game over. A interface gráfica utiliza elementos básicos, como desenhos de retângulos e textos, garantindo simplicidade e clareza.

3 RESULTADOS OBTIDOS

Nesta seção, serão apresentados os resultados do desenvolvimento do **Jogo da Cobrinha**, começando com a exibição das telas que compõem o jogo, como o menu principal, a seleção de dificuldade e a tela principal de gameplay. Após isso, será realizada uma análise detalhada do código, explicando as principais funções, seus objetivos, e em que momentos são chamadas durante a execução do jogo. Essa abordagem busca destacar como cada parte do sistema contribui para o funcionamento completo do projeto.

3.1 Telas do Projeto

O jogo está dividido em três telas principais:

1. **Tela de Menu (main.cpp)**: Aqui, o jogador pode iniciar o jogo ou sair. Quando o jogador clica em "Jogar", a tela de seleção de dificuldade é exibida.

2. **Tela de Seleção de Dificuldade (difficultScreen.cpp)**: O jogador escolhe a dificuldade (Fácil, Normal ou Difícil), que ajusta a velocidade do jogo.
3. **Tela de Jogo (gameScreen.cpp)**: Esta é a tela principal onde a lógica do jogo acontece. A cobra se move, come a comida, cresce, e o jogador perde se colidir com as paredes ou com seu próprio corpo.

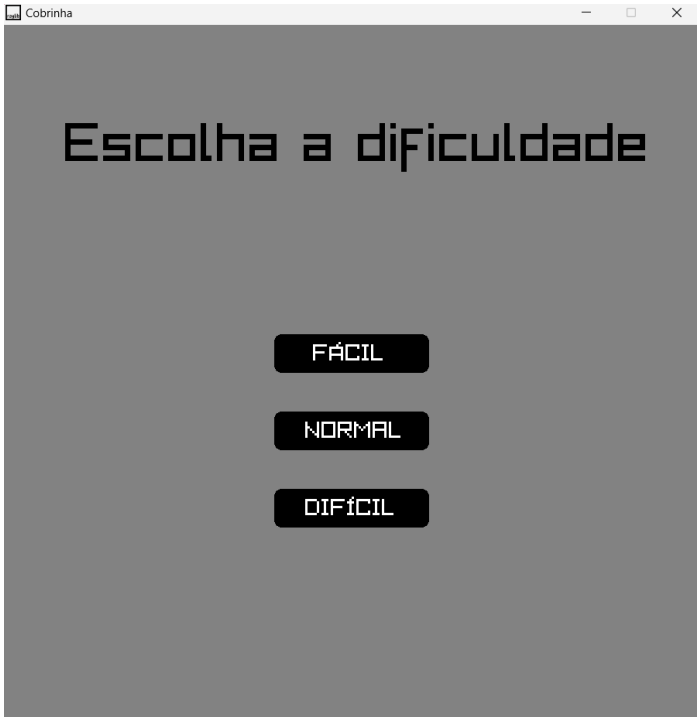
3.1.1 Tela 1: Menu Iniciar

- O menu inicial é onde o jogador escolhe se deseja jogar ou sair.
- **Componentes**: Dois botões (Jogar e Sair), um título no topo ("Jogo da Cobrinha") e uma imagem do time (no canto inferior direito).
- **Função**: O código realiza a verificação de cliques nos botões e, dependendo da ação do usuário, transita para a próxima tela ou encerra o jogo.

3.1.2 Tela 2: Seleção de dificuldade

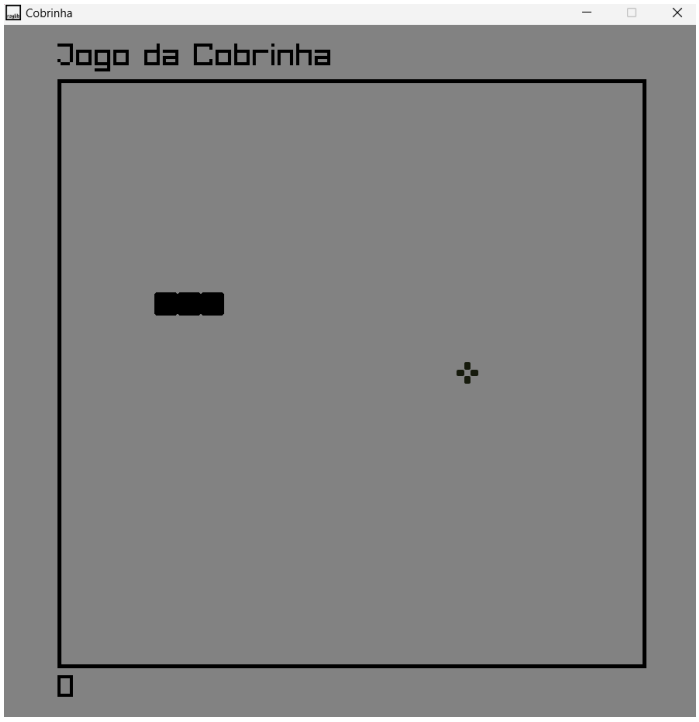
- Após o menu, o jogador escolhe a dificuldade do jogo: Fácil, Normal ou Difícil.
- **Componentes**: Três botões (Fácil, Normal, Difícil), um título na parte superior ("Escolha a dificuldade").

- **Função**: O código verifica a seleção do jogador e ajusta a dificuldade do jogo, o que influencia a velocidade de movimentação da cobra.



3.1.3 Tela 3: Jogo

- Esta é a tela principal do jogo, onde a cobra se move pelo grid, coleta a comida e cresce.
- **Componentes**: A cobra, a comida, o score, a borda da tela e sons de interação (comer e colisões com a parede).
- **Função**: A cobra se move com base nas teclas de direção. O jogador pontua cada vez que a cobra come a comida. Se a cobra bate nas paredes ou em si mesma, o jogo reinicia.



3.2 Detalhamento do Código

3.2.1 Classe Menu() – (main.cpp)

A classe Menu é responsável por gerenciar a interface inicial do jogo, apresentada ao jogador. Ela organiza o layout do menu principal e captura as interações do usuário para navegar entre as opções. O Menu guia o usuário para as próximas ações com uma interface simples e funcional.

Métodos e Atributos da Classe Menu:

1. **Construtor e Layout:**

- Define os botões "Jogar", "Dificuldade" e "Sair", estabelecendo suas posições e dimensões.
- Exibe o título do jogo e organiza elementos visuais com bordas e alinhamento.
-

Exemplo de código do layout do Menu:

```
// Exibe o titulo e os textos nos botões
    DrawText("Jogo da", 150, 120, 70, BLACK);
    DrawText("Cobrinha", 230, 250, 100, BLACK);
    DrawText("JOGAR", playButton.x + 50, playButton.y + 10, 30,
WHITE);
    DrawText("SAIR", exitButton.x + 60, exitButton.y + 10, 30,
WHITE);

```

2. **Método de Interação:**

- Detecta cliques do mouse nos botões usando CheckCollisionPointRec().
- Chama as telas apropriadas dependendo do botão clicado.

Exemplo de código para detectar cliques:

```
// Verifica se o botão Jogar foi clicado
    if (CheckCollisionPointRec(GetMousePosition(), playButton) &&
IsMouseButtonPressed(MOUSE_BUTTON_LEFT)) {
        UnloadTexture(textura);
        gameRunning = 1;
    }

```

3.2.2 Tela de dificuldade – (difficultScreen.cpp)

Essa tela permite ao jogador selecionar o nível de dificuldade, que controla a velocidade do jogo.

Funcionamento:

- Apresenta três botões: "Fácil", "Normal" e "Difícil".
- Cada botão retorna um valor correspondente ao intervalo de tempo para os *updates* do jogo.

Exemplo de código para retornar a dificuldade escolhida:

```
// Verifica se a dificuldade escolhida foi fácil
if (CheckCollisionPointRec(GetMousePosition(), easyButton) &&
IsMouseButtonPressed(MOUSE_BUTTON_LEFT)) {
    return 0.2;
}

// Verifica se a dificuldade escolhida foi normal
if (CheckCollisionPointRec(GetMousePosition(), normalButton) &&
IsMouseButtonPressed(MOUSE_BUTTON_LEFT)) {
    return 0.1;
}

// Verifica se a dificuldade escolhida foi dificil
if (CheckCollisionPointRec(GetMousePosition(), hardButton) &&
IsMouseButtonPressed(MOUSE_BUTTON_LEFT)) {
    return 0.05;
}
```

Quanto menor o intervalo retornado, maior a velocidade e dificuldade do jogo.

3.2.3 Funções do Jogo – (gameScreen.cpp)

ElementoNoDeque(Vector2 posicaoComida, deque<Vector2> corpo):

- **Descrição:** Verifica se a comida é gerada em uma posição ocupada pelo corpo da cobra.

Exemplo do código:

```
if (Vector2Equals(corpo[i], posicaoComida)) { return true; }
```

UpdateAconteceu(double intervalo):

- **Descrição:** Sincroniza o tempo de atualização com a dificuldade escolhida através do parâmetro intervalo.

Exemplo do código:

```
if (tempoAtual - ultimoUpdate >= intervalo) { ultimoUpdate =
tempoAtual; return true; }
```

3.2.4 Classes do Jogo – (gameScreen.cpp)

Classe Cobra

Atributos:

- **deque<Vector2> corpo:** Representa os segmentos da cobra no grid.
- **Vector2 direcao:** Determina a direção do movimento.
- **bool crescimento:** Controla se a cobra deve crescer.

Métodos:

- **Draw():** Desenha cada segmento do corpo no grid.

```
// Criando um retangulo
Rectangle pedacoCorpo = Rectangle{borda + x * tamanhoCelula, borda + y
* tamanhoCelula, (float)tamanhoCelula, (float)tamanhoCelula};

// Imprimindo o retangulo criado com bordas arredondadas
DrawRectangleRounded(pedacoCorpo, 0.3, 6, BLACK);
```

- **Update():** Atualiza a posição da cobra e verifica se ela deve crescer.

```
// Faz a "animação" de andar adicionando um elemento no vector,
somando as coordenadas da cabeça com a direção desejada
// e retirando o último elemento do vector caso o crescimento seja
false, dando a impressão da "animação"
corpo.push_front(Vector2Add(corpo[0], direcao));

// Faz a verificação se a cobra deve crescer ou não
if (crescimento == true) {
    crescimento = false;
} else {
// Função que retira o último valor do vector2 caso crescimento =
false
    corpo.pop_back();
}
```

Classe Comida

Atributos:

- **Vector2 posicao:** Armazena a posição da comida.
- **Texture2D texture:** Define a textura da comida.

Método Principal:

- **GenerateRandomPos():** Gera uma posição aleatória para a comida, verificando se não colide com o corpo da cobra.

```
while (ElementoNoDeque(posicao, corpoCobra)) { posicao =
GenerateRandomCell(); }
```

Classe Game

A classe Game foi projetada para centralizar e organizar os elementos principais do jogo, facilitando a interação entre as diversas classes, funções e dados necessários para o funcionamento do sistema. Ela age como uma camada de controle, coordenando a lógica do jogo e promovendo uma comunicação mais eficiente entre objetos como a cobra e a comida.

Finalidade:

A classe serve para encapsular as principais funcionalidades do jogo, como:

- Gerenciar a lógica de colisão (cobra com paredes, comida ou com o próprio corpo).
- Atualizar o estado do jogo com base nos inputs do jogador e nas regras estabelecidas.
- Garantir que os elementos sejam desenhados corretamente a cada frame.
- Armazenar e manipular dados globais do jogo, como pontuação e estado (running).

Essa abordagem modular facilita a manutenção e expansão do código, permitindo que outros elementos sejam adicionados sem comprometer a estrutura existente. Em resumo, a classe Game atua como um "orquestrador", promovendo uma integração eficiente e fluida entre os componentes do jogo.

Função gameScreen(difficult):

Essa função integra todas as partes do jogo.

Descrição:

- Inicializa o jogo e executa o loop principal.

- Atualiza o estado da cobra e da comida com base no tempo e na interação do jogador.

Exemplo de uso do parâmetro de dificuldade:

```
// Atualiza o jogo no tempo atribuido na tela de seleção de
dificuldade
if(UpdateAconteceu(difficult)) {
   game.Update();
}
```

Detecção de teclas para mudar a direção da cobra:

```
if (IsKeyPressed(KEY_UP) && game.cobra.direcao.y != 1) {
   game.cobra.direcao = {0, -1};
   game.running = true;
}
```

Função int main() (main.cpp)

Essa é a função que conecta todas as telas do programa.

Descrição:

- Inicia a janela principal do jogo.
- Chama o menu, a tela de dificuldade e a tela do jogo conforme a interação do usuário.

Exemplo de integração:

```
// Caso o botão jogar for apertado...
if (gameRunning == 1) {
// Chama a tela de seleção de dificuldade
    dificuldade = difficultScreen();
// Chama a tela de jogo após escolher a dificuldade
    saiu = GameScreen(dificuldade);
}

```

Essa estrutura garante que o programa flua corretamente entre as telas e encerre apenas quando a janela for fechada.

5 CONCLUSÃO

O desenvolvimento do projeto do jogo da cobrinha foi uma experiência enriquecedora que permitiu consolidar conhecimentos em programação, estruturação de código e utilização de bibliotecas gráficas. O jogo alcançou os objetivos propostos, apresentando uma interface funcional, mecânicas desafiadoras e uma boa integração entre as telas e os elementos principais, como o controle da cobra e a interação com os alimentos.

O sistema de dificuldade configurável mostrou-se eficaz em diversificar a experiência do jogador, atendendo tanto iniciantes quanto aqueles que buscam maior desafio. Além disso, o uso de classes e funções bem definidas promoveu a organização do código, facilitando a manutenção e futuras expansões.

Sugestões para Melhorias:

Embora o projeto tenha alcançado um bom resultado, há espaço para aperfeiçoamentos. Algumas melhorias sugeridas incluem:

1. **Adição de Placares e Rankings:** Implementar um sistema de pontuação persistente que permita ao jogador comparar resultados de diferentes partidas.
2. **Melhoria nos Gráficos:** Utilizar texturas mais elaboradas para a cobra e os alimentos, além de animações para eventos como colisões.
3. **Música e Efeitos Sonoros:** Incluir uma trilha sonora e mais sons para interações.
4. **Portabilidade:** Adaptar o jogo para dispositivos móveis, aproveitando a popularidade desse tipo de entretenimento em plataformas como Android e iOS.

Com essas melhorias, o projeto pode oferecer uma experiência ainda mais rica e envolvente para os jogadores, além de servir como base para trabalhos futuros em jogos e interfaces gráficas.

REFERÊNCIAS

S RAYLIB. *raylib Documentation: Official*. Disponível em: https://www.raylib.com/cheatsheet.html. Acesso em: 17 nov. 2024.

GITHUB. *Raylib CPP Starter Template for VSCode*. Disponível em: https://codeload.github.com/educ8s/Raylib-CPP-Starter-Template-for-VSCODE/zip/refs/heads/main. Acesso em: 17 nov. 2024.

STACK OVERFLOW. *raylib tags*. Disponível em: https://stackoverflow.com/questions/tagged/raylib. Acesso em: 17 nov. 2024.

YOUTUBE. *Raylib Basics for Beginners - Learn to Make Games*. Disponível em: https://www.youtube.com/watch?v=FVUAwBgcqgo. Acesso em: 17 nov. 2024.

YOUTUBE. *Introduction to raylib - Getting Started Guide*. Disponível em: https://www.youtube.com/watch?v=ihLz8h6Jpkw. Acesso em: 17 nov. 2024.

COLIRU. *Compile-link-run: Raylib examples*. Disponível em: http://coliru.stacked-crooked.com/. Acesso em: 17 nov. 2024.

RAYLIB. *Learn how to use raylib - Examples and Demos*. Disponível em: https://github.com/raysan5/raylib. Acesso em: 17 nov. 2024.