INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

MATHEUS LOPES DE OLIVEIRA

Pesquisa sobre a Biblioteca Raylib

CAMPOS DO JORDÃO

2024

RESUMO

Este estudo aborda a Raylib, uma biblioteca de código aberto amplamente utilizada no desenvolvimento de jogos e gráficos 2D e 3D, com ênfase na simplicidade e acessibilidade. Desenvolvida em 2013 por Ramon Santamaria, a Raylib foi concebida para auxiliar no ensino de desenvolvimento de jogos, proporcionando uma interface gráfica intuitiva e de fácil uso. A biblioteca, escrita em C e utilizando a API OpenGL, destaca-se por sua compatibilidade com múltiplas plataformas, como Windows, Linux, macOS e Android, além de permitir a prototipagem rápida e o aprendizado prático de conceitos gráficos. Este trabalho apresenta exemplos práticos de código em Raylib e discute sua utilização em jogos independentes, como clones de *Flappy Bird* e *Bomberman*. A pesquisa destaca a importância da Raylib como ferramenta educacional, proporcionando um ambiente acessível para iniciantes no desenvolvimento de jogos e gráficos.

Palavras-chave: Raylib, desenvolvimento de jogos, gráficos 2D/3D, programação gráfica, Ramon Santamaria.

ABSTRACT

This study explores Raylib, an open-source library widely used in the development of 2D and 3D games and graphics, with a focus on simplicity and accessibility. Developed in 2013 by Ramon Santamaria, Raylib was designed to assist in teaching game development, offering an intuitive and easy-to-use graphical interface. Written in C and using the OpenGL API, the library stands out for its compatibility with multiple platforms such as Windows, Linux, macOS, and Android, and allows for rapid prototyping and practical learning of graphical concepts. This paper presents practical code examples using Raylib and discusses its application in indie games, such as *Flappy Bird* and *Bomberman* clones. The research highlights Raylib's importance as an educational tool, providing an accessible environment for beginners in game and graphics development.

Keywords: Raylib, game development, 2D/3D graphics, graphical programming, Ramon Santamaria.

LISTA DE ILUSTRAÇÕES

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

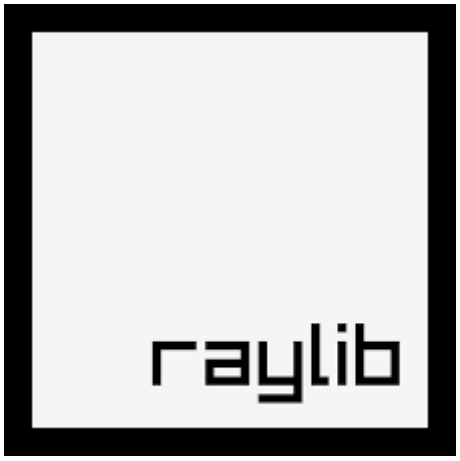

Figura 1 - Raylib

A *Raylib* é uma biblioteca de código aberto voltada ao desenvolvimento de jogos e gráficos em 2D e 3D, amplamente reconhecida por sua simplicidade e acessibilidade. Desenvolvida com o propósito de ser uma ferramenta intuitiva, a *Raylib* é especialmente utilizada por iniciantes no campo de programação gráfica e desenvolvimento de jogos. Seu principal diferencial reside na facilidade de uso, permitindo a criação de gráficos e jogos sem a complexidade inerente a bibliotecas mais robustas, como OpenGL ou DirectX.

1.1 História e Desenvolvimento

Figura 2 - Ramon Santamaria

A *Raylib* foi criada em 2013 por **Ramon Santamaria**, inicialmente como um projeto pessoal para auxiliar no ensino de desenvolvimento de jogos a seus alunos. Ramon identificou a carência de ferramentas simples que permitissem uma compreensão mais acessível dos conceitos de gráficos, sem sacrificar funcionalidades essenciais. Desenvolvida em linguagem **C**, a *Raylib* utiliza a API OpenGL para manipulação gráfica, proporcionando uma interface de programação simples e eficaz para o desenvolvimento de aplicações gráficas.

A filosofia por trás da criação da *Raylib* era a simplicidade, e por isso, ela foi projetada para ser minimalista e de fácil uso, focando no aprendizado prático de gráficos e programação de jogos. Com o passar dos anos, a biblioteca ganhou notoriedade, e a comunidade de desenvolvedores contribuiu significativamente para seu crescimento, tanto em funcionalidades quanto em portabilidade, sendo compatível com várias plataformas, como Windows, Linux, macOS, Android e dispositivos embarcados, como o Raspberry Pi.

A biblioteca é distribuída sob a licença **zlib/libpng**, o que permite sua utilização gratuita tanto em projetos de caráter pessoal quanto em iniciativas comerciais. A contínua colaboração entre a comunidade e Ramon Santamaria impulsionou a incorporação de novos recursos, a otimização de seu desempenho e a adaptação a diferentes compiladores e versões mais recentes das plataformas de desenvolvimento.

2 EM QUAIS ÁREAS A RAYLIB É UTILIZADA

A Raylib tem se consolidado como uma ferramenta amplamente utilizada no **desenvolvimento de jogos educacionais**, prototipagem rápida de jogos e construção de ferramentas gráficas. A simplicidade de sua arquitetura a torna particularmente adequada para iniciantes, que encontram na biblioteca um ambiente propício para aprender conceitos fundamentais de programação gráfica e design de jogos.

Além de seu uso na criação de jogos, a Raylib tem sido utilizada em outras áreas, como **visualização de dados**, **simulações gráficas** e desenvolvimento de **aplicações interativas**. Em projetos acadêmicos, a biblioteca desempenha um papel importante, principalmente por sua facilidade na implementação de gráficos e animações, tornando-se uma escolha popular entre estudantes e pesquisadores.

Figura 3 – Wave Collector

A comunidade ativa em torno da Raylib também contribui com exemplos práticos, tutoriais e projetos desenvolvidos, reforçando seu papel como uma ferramenta educativa. Além disso, sua portabilidade permitiu que desenvolvedores experimentassem a criação de protótipos para diferentes plataformas, incluindo consoles como o **PSP** e o **Nintendo Switch**, demonstrando a versatilidade e adaptabilidade da Raylib em diversos contextos.

3 ALGUNS EXEMPLOS DE CÓDIGOS

A seguir, apresenta-se um exemplo simples de código utilizando Raylib, no qual um círculo é desenhado na tela, e o usuário pode mover esse círculo utilizando as teclas de direção:

```c
#include "raylib.h"

int main(void) {
    // Inicializa a janela
    InitWindow(800, 600, "Exemplo Raylib - Movimento");

    // Posição inicial do círculo
    Vector2 position = { 400, 300 };

    SetTargetFPS(60); // Limita a 60 FPS

    // Loop principal
    while (!WindowShouldClose()) {
        // Movimenta o círculo
        if (IsKeyDown(KEY_RIGHT)) position.x += 5.0f;
        if (IsKeyDown(KEY_LEFT)) position.x -= 5.0f;
        if (IsKeyDown(KEY_UP)) position.y -= 5.0f;
        if (IsKeyDown(KEY_DOWN)) position.y += 5.0f;

        // Inicia o processo de renderização
        BeginDrawing();
        ClearBackground(RAYWHITE); // Limpa a tela
```

```
        DrawCircleV(position, 50, BLUE); // Desenha o círculo

        DrawText("Use as setas para mover o círculo", 10, 10, 20, DARKGRAY);

        EndDrawing(); // Finaliza o processo de renderização
    }

    CloseWindow(); // Fecha a janela

    return 0;
}
```

Algoritmo 1 – Círculo

Outro exemplo, é o código do jogo “Snake”, um clássico que tem como objetivo coletar quadrados na tela evitando bater nos cantos dela, os quadrados acabam formando uma cobra. O código foi disponibilizado no repositório do Github da própria biblioteca Raylib:

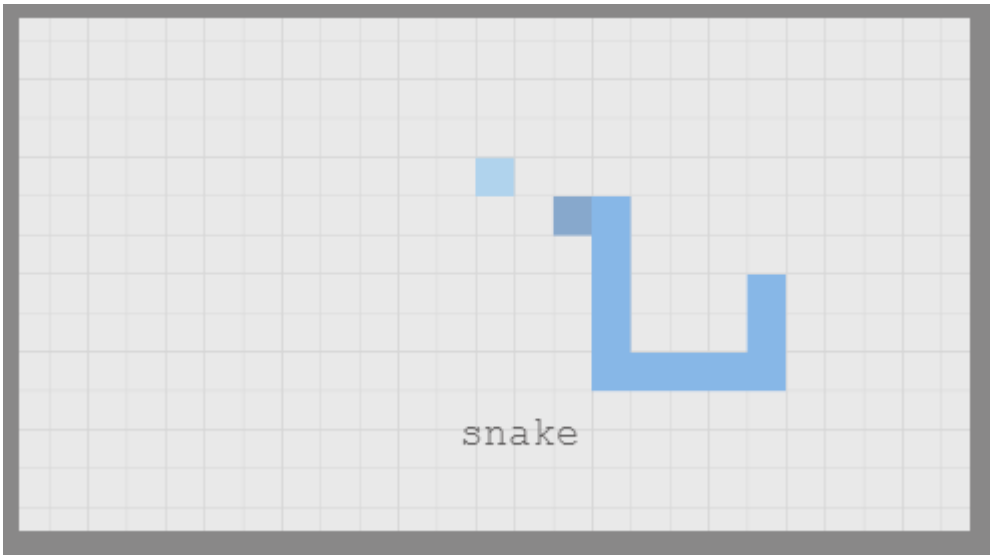

Figura 4 – Snake

```
/*******************************************************************************************
*
*   raylib - classic game: snake
*
*   Sample game developed by Ian Eito, Albert Martos and Ramon Santamaria
*
*   This game has been created using raylib v1.3 (www.raylib.com)
*   raylib is licensed under an unmodified zlib/libpng license (View raylib.h for details)
*
*   Copyright (c) 2015 Ramon Santamaria (@raysan5)
*
********************************************************************************************/

#include "raylib.h"

#if defined(PLATFORM_WEB)
```

```
    #include <emscripten/emscripten.h>
#endif

//----------------------------------------------------------------------------------
// Some Defines
//----------------------------------------------------------------------------------
#define SNAKE_LENGTH   256
#define SQUARE_SIZE     31

//----------------------------------------------------------------------------------
// Types and Structures Definition
//----------------------------------------------------------------------------------
typedef struct Snake {
    Vector2 position;
    Vector2 size;
    Vector2 speed;
    Color color;
} Snake;

typedef struct Food {
    Vector2 position;
    Vector2 size;
    bool active;
    Color color;
} Food;

//----------------------------------------------------------------------------------
// Global Variables Declaration
//----------------------------------------------------------------------------------
static const int screenWidth = 800;
static const int screenHeight = 450;

static int framesCounter = 0;
static bool gameOver = false;
static bool pause = false;

static Food fruit = { 0 };
static Snake snake[SNAKE_LENGTH] = { 0 };
static Vector2 snakePosition[SNAKE_LENGTH] = { 0 };
static bool allowMove = false;
static Vector2 offset = { 0 };
static int counterTail = 0;

//----------------------------------------------------------------------------------
// Module Functions Declaration (local)
//----------------------------------------------------------------------------------
static void InitGame(void);         // Initialize game
static void UpdateGame(void);       // Update game (one frame)
static void DrawGame(void);         // Draw game (one frame)
static void UnloadGame(void);       // Unload game
static void UpdateDrawFrame(void);  // Update and Draw (one frame)

//----------------------------------------------------------------------------------
// Program main entry point
//----------------------------------------------------------------------------------
int main(void)
{
    // Initialization (Note windowTitle is unused on Android)
    //---------------------------------------------------------
    InitWindow(screenWidth, screenHeight, "classic game: snake");

    InitGame();
```

```
#if defined(PLATFORM_WEB)
    emscripten_set_main_loop(UpdateDrawFrame, 60, 1);
#else
    SetTargetFPS(60);
    //--------------------------------------------------------------------------------------

    // Main game loop
    while (!WindowShouldClose())    // Detect window close button or ESC key
    {
        // Update and Draw
        //----------------------------------------------------------------------------------
        UpdateDrawFrame();
        //----------------------------------------------------------------------------------
    }
#endif
    // De-Initialization
    //--------------------------------------------------------------------------------------
    UnloadGame();         // Unload loaded data (textures, sounds, models...)

    CloseWindow();        // Close window and OpenGL context
    //--------------------------------------------------------------------------------------

    return 0;
}

//------------------------------------------------------------------------------------
// Module Functions Definitions (local)
//------------------------------------------------------------------------------------

// Initialize game variables
void InitGame(void)
{
    framesCounter = 0;
    gameOver = false;
    pause = false;

    counterTail = 1;
    allowMove = false;

    offset.x = screenWidth%SQUARE_SIZE;
    offset.y = screenHeight%SQUARE_SIZE;

    for (int i = 0; i < SNAKE_LENGTH; i++)
    {
        snake[i].position = (Vector2){ offset.x/2, offset.y/2 };
        snake[i].size = (Vector2){ SQUARE_SIZE, SQUARE_SIZE };
        snake[i].speed = (Vector2){ SQUARE_SIZE, 0 };

        if (i == 0) snake[i].color = DARKBLUE;
        else snake[i].color = BLUE;
    }

    for (int i = 0; i < SNAKE_LENGTH; i++)
    {
        snakePosition[i] = (Vector2){ 0.0f, 0.0f };
    }

    fruit.size = (Vector2){ SQUARE_SIZE, SQUARE_SIZE };
    fruit.color = SKYBLUE;
    fruit.active = false;
}
```

```
// Update game (one frame)
void UpdateGame(void)
{
    if (!gameOver)
    {
        if (IsKeyPressed('P')) pause = !pause;

        if (!pause)
        {
            // Player control
            if (IsKeyPressed(KEY_RIGHT) && (snake[0].speed.x == 0) && allowMove)
            {
                snake[0].speed = (Vector2){ SQUARE_SIZE, 0 };
                allowMove = false;
            }
            if (IsKeyPressed(KEY_LEFT) && (snake[0].speed.x == 0) && allowMove)
            {
                snake[0].speed = (Vector2){ -SQUARE_SIZE, 0 };
                allowMove = false;
            }
            if (IsKeyPressed(KEY_UP) && (snake[0].speed.y == 0) && allowMove)
            {
                snake[0].speed = (Vector2){ 0, -SQUARE_SIZE };
                allowMove = false;
            }
            if (IsKeyPressed(KEY_DOWN) && (snake[0].speed.y == 0) && allowMove)
            {
                snake[0].speed = (Vector2){ 0, SQUARE_SIZE };
                allowMove = false;
            }

            // Snake movement
            for (int i = 0; i < counterTail; i++) snakePosition[i] = snake[i].position;

            if ((framesCounter%5) == 0)
            {
                for (int i = 0; i < counterTail; i++)
                {
                    if (i == 0)
                    {
                        snake[0].position.x += snake[0].speed.x;
                        snake[0].position.y += snake[0].speed.y;
                        allowMove = true;
                    }
                    else snake[i].position = snakePosition[i-1];
                }
            }

            // Wall behaviour
            if (((snake[0].position.x) > (screenWidth - offset.x)) ||
                ((snake[0].position.y) > (screenHeight - offset.y)) ||
                (snake[0].position.x < 0) || (snake[0].position.y < 0))
            {
                gameOver = true;
            }

            // Collision with yourself
            for (int i = 1; i < counterTail; i++)
            {
                if ((snake[0].position.x == snake[i].position.x) && (snake[0].position.y ==
snake[i].position.y)) gameOver = true;
```

```
            }

            // Fruit position calculation
            if (!fruit.active)
            {
                fruit.active = true;
                fruit.position = (Vector2){ GetRandomValue(0, (screenWidth/SQUARE_SIZE) -
1)*SQUARE_SIZE + offset.x/2, GetRandomValue(0, (screenHeight/SQUARE_SIZE) - 1)*SQUARE_SIZE +
offset.y/2 };

                for (int i = 0; i < counterTail; i++)
                {
                    while ((fruit.position.x == snake[i].position.x) && (fruit.position.y ==
snake[i].position.y))
                    {
                        fruit.position = (Vector2){ GetRandomValue(0, (screenWidth/SQUARE_SIZE) -
1)*SQUARE_SIZE + offset.x/2, GetRandomValue(0, (screenHeight/SQUARE_SIZE) - 1)*SQUARE_SIZE +
offset.y/2 };
                        i = 0;
                    }
                }
            }

            // Collision
            if ((snake[0].position.x < (fruit.position.x + fruit.size.x) && (snake[0].position.x
+ snake[0].size.x) > fruit.position.x) &&
                (snake[0].position.y < (fruit.position.y + fruit.size.y) && (snake[0].position.y
+ snake[0].size.y) > fruit.position.y))
            {
                snake[counterTail].position = snakePosition[counterTail - 1];
                counterTail += 1;
                fruit.active = false;
            }

            framesCounter++;
        }
    }
    else
    {
        if (IsKeyPressed(KEY_ENTER))
        {
            InitGame();
            gameOver = false;
        }
    }
}

// Draw game (one frame)
void DrawGame(void)
{
    BeginDrawing();

        ClearBackground(RAYWHITE);

        if (!gameOver)
        {
            // Draw grid lines
            for (int i = 0; i < screenWidth/SQUARE_SIZE + 1; i++)
            {
                DrawLineV((Vector2){SQUARE_SIZE*i + offset.x/2, offset.y/2}, (Vec-
tor2){SQUARE_SIZE*i + offset.x/2, screenHeight - offset.y/2}, LIGHTGRAY);
            }
```

```
            for (int i = 0; i < screenHeight/SQUARE_SIZE + 1; i++)
            {
                DrawLineV((Vector2){offset.x/2, SQUARE_SIZE*i + offset.y/2}, (Vector2){scre-
enWidth - offset.x/2, SQUARE_SIZE*i + offset.y/2}, LIGHTGRAY);
            }

            // Draw snake
            for (int i = 0; i < counterTail; i++) DrawRectangleV(snake[i].position,
snake[i].size, snake[i].color);

            // Draw fruit to pick
            DrawRectangleV(fruit.position, fruit.size, fruit.color);

            if (pause) DrawText("GAME PAUSED", screenWidth/2 - MeasureText("GAME PAUSED", 40)/2,
screenHeight/2 - 40, 40, GRAY);
        }
        else DrawText("PRESS [ENTER] TO PLAY AGAIN", GetScreenWidth()/2 - MeasureText("PRESS [EN-
TER] TO PLAY AGAIN", 20)/2, GetScreenHeight()/2 - 50, 20, GRAY);

    EndDrawing();
}

// Unload game variables
void UnloadGame(void)
{
    // TODO: Unload all dynamic loaded data (textures, sounds, models...)
}

// Update and Draw (one frame)
void UpdateDrawFrame(void)
{
    UpdateGame();
    DrawGame();
}
```

Algoritmo 2 – Snake

4 EXEMPLOS DE JOGOS CRIADOS NO RAYLIB

Diversos jogos e projetos independentes foram criados com base na Raylib, demonstrando sua versatilidade e aplicabilidade em diferentes contextos de desenvolvimento. Alguns exemplos de destaque incluem:

Figura 5 – Asteroids

1. **Asteroids Clone**: Um jogo inspirado no clássico "Asteroids", recriado por desenvolvedores independentes utilizando a Raylib para aprender e implementar conceitos básicos de colisão e movimentação de objetos no espaço 2D.

Figura 6 – Flappy Bird

2. **Flappy Bird Clone**: Muitos desenvolvedores utilizam a Raylib para criar versões do famoso jogo "Flappy Bird", aproveitando-se da simplicidade de manipulação de sprites e detecção de colisões.

Figura 7 – Bomberman

3. **Bomberman**: Clones de "Bomberman" foram desenvolvidos com a biblioteca, explorando suas capacidades de renderização em 2D e movimentação de personagens, além de interação em tempo real entre jogadores.

Figura 8 – Rogue-like

4. **Jogos Rogue-like**: Raylib também tem sido utilizada para a criação de jogos do gênero rogue-like, caracterizados por mapas gerados de forma procedural e movimentação baseada em turnos, permitindo uma experiência de jogo dinâmica e desafiadora.

4 CONCLUSÃO

A Raylib destaca-se como uma ferramenta poderosa e acessível para o desenvolvimento de jogos e gráficos, especialmente voltada para iniciantes e educadores. Sua simplicidade permite uma curva de aprendizado rápida, sem comprometer a profundidade de recursos disponíveis para desenvolvedores mais experientes. A ampla compatibilidade com diferentes plataformas e linguagens, aliada a uma comunidade ativa, fortalece sua posição como uma das melhores opções para quem deseja explorar o desenvolvimento de jogos e aplicações gráficas interativas.

REFERÊNCIAS

Documentação oficial da RAYLIB. Disponível em: https://www.raylib.com/ . Acesso em: 3 setembro 2024.

RAYLIB GitHub Repository. Disponível em: https://github.com/raysan5/raylib. Acesso em: 3 setembro 2024.

Terminal Root. **"Crie Jogos para Windows, Linux e Web com Raylib C/C++".** Disponível em: https://www.youtube.com/watch?v=LZUMVdkWWrg . Acesso em: 4 setembro 2024.