Reporte Culebrita

Alumno: Solano Meza Angel Daniel

Clase: Paradigmas de Programación

Introducción

Este documento describe de manera general el código del juego "Culebrita", una versión del clásico juego de la serpiente implementado en C utilizando la biblioteca gráfica `raylib`. El objetivo del juego es mover una serpiente por la pantalla para comer frutas y crecer, evitando colisionar con los bordes de la pantalla o consigo misma.

Estructura del Código

Inclusiones y Definiciones

El código comienza incluyendo las librerías necesarias, `raylib` y algunas estándar de C, y define el tamaño de cada segmento de la serpiente:

```
#include "raylib.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define SQUARE_SIZE 31
```

Estructuras de Datos

Se definen estructuras para representar los segmentos de la serpiente (`SnakeSegment`), los nodos de la lista enlazada que conforman la serpiente (`SnakeNode`) y la fruta (`Food`):

```
typedef struct SnakeSegment {
    Vector2 position;
    Vector2 size;
    Vector2 speed;
    Texture2D sprite;
} SnakeSegment;

typedef struct SnakeNode {
    SnakeSegment segment;
    struct SnakeNode *next;
} SnakeNode;

typedef struct Food {
    Vector2 position;
```

```
    Vector2 size;
    bool active;
    Texture2D sprite;
} Food;
```

Declaración de Funciones

Se declaran las funciones utilizadas en el juego:

```
static void InitGame(void);
static void UpdateGame(Sound eat, Sound defeat);
static void DrawGame(Sound defeat, Sound eat, Music music);
static void DrawStartScreen(void);
static void UnloadGame(void);
static void UpdateDrawFrame(Sound defeat, Sound eat, Music music, Music
iniciar);
static void AddSegment(void);
static void FreeSnake(void);
```

Función Principal

La función `main` inicializa la ventana, carga los recursos de audio, inicia el juego y contiene el bucle principal:

```
int main(void) {
    InitWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "Culebrita");
    InitAudioDevice();

    Music iniciar = LoadMusicStream("..\\sonidos\\iniciar.mp3");
    Sound defeat = LoadSound("..\\sonidos\\derrota.mp3");
    Sound eat = LoadSound("..\\sonidos\\comer.mp3");
    Music music = LoadMusicStream("..\\sonidos\\musica.mp3");

    SetSoundVolume(defeat, 0.1f);
    SetSoundVolume(eat, 0.1f);
    SetMusicVolume(music, .75f);
    SetMusicVolume(iniciar, .75f);

    PlayMusicStream(iniciar);
    PlayMusicStream(music);

    InitGame();
    SetTargetFPS(60);

    while (!WindowShouldClose()) {
        UpdateDrawFrame(defeat, eat, music, iniciar);
    }
```

```
    UnloadSound(defeat);
    UnloadGame();
    CloseWindow();

    return 0;
}
```

Inicialización del Juego

La función `InitGame` inicializa los parámetros del juego y crea la serpiente inicial:

```
static void InitGame(void) {
    framesCounter = 0;
    gameOver = false;
    pause = false;

    counterTail = 1;
    offset.x = SCREEN_WIDTH % SQUARE_SIZE;
    offset.y = SCREEN_HEIGHT % SQUARE_SIZE;

    FreeSnake();

    snakeHead = (SnakeNode *)malloc(sizeof(SnakeNode));
    snakeHead->segment.position = (Vector2){offset.x / 2, offset.y / 2};
    snakeHead->segment.size = (Vector2){SQUARE_SIZE, SQUARE_SIZE};
    snakeHead->segment.speed = (Vector2){SQUARE_SIZE, 0};
    snakeHead->segment.sprite =
LoadTexture("C:\\Users\\Mitillo\\Universidad\\4to.
Semestre\\P.P\\Practica1\\cualquier\\texturas\\snake-head.png");
    snakeHead->next = NULL;

    fruit.size = (Vector2){SQUARE_SIZE, SQUARE_SIZE};
    fruit.sprite = LoadTexture("C:\\Users\\Mitillo\\Universidad\\4to.
Semestre\\P.P\\Practica1\\cualquier\\texturas\\fruta.png");
    fruit.active = false;
    AddSegment();
}
```

Actualización del Juego

La función `UpdateGame` maneja la lógica del juego, incluyendo el movimiento de la serpiente, la generación de frutas y las colisiones:

```
static void UpdateGame(Sound eat, Sound defeat) {
    if (!gameOver) {
        if (IsKeyPressed('P')) {
            pause = !pause;
        }
```

```
        if (!pause) {
            if (IsKeyPressed(KEY_RIGHT) && (snakeHead->segment.speed.x == 0)) {
                snakeHead->segment.speed = (Vector2){SQUARE_SIZE, 0};
            }
            if (IsKeyPressed(KEY_LEFT) && (snakeHead->segment.speed.x == 0)) {
                snakeHead->segment.speed = (Vector2){-SQUARE_SIZE, 0};
            }
            if (IsKeyPressed(KEY_UP) && (snakeHead->segment.speed.y == 0)) {
                snakeHead->segment.speed = (Vector2){0, -SQUARE_SIZE};
            }
            if (IsKeyPressed(KEY_DOWN) && (snakeHead->segment.speed.y == 0)) {
                snakeHead->segment.speed = (Vector2){0, SQUARE_SIZE};
            }

            if ((framesCounter % 5) == 0) {
                SnakeNode *current = snakeHead;
                Vector2 previousPosition = current->segment.position;
                Vector2 tempPosition;

                current->segment.position.x += current->segment.speed.x;
                current->segment.position.y += current->segment.speed.y;

                while (current->next != NULL) {
                    current = current->next;
                    tempPosition = current->segment.position;
                    current->segment.position = previousPosition;
                    previousPosition = tempPosition;
                }
            }

            if ((snakeHead->segment.position.x >= SCREEN_WIDTH - offset.x) ||
                (snakeHead->segment.position.y >= SCREEN_HEIGHT - offset.y) ||
                (snakeHead->segment.position.x < 0) || (snakeHead-
>segment.position.y < 0)) {
                gameOver = true;
            }

            SnakeNode *current = snakeHead->next;
            while (current != NULL) {
                if ((snakeHead->segment.position.x == current-
>segment.position.x) &&
                    (snakeHead->segment.position.y == current-
>segment.position.y)) {
                    gameOver = true;
                    break;
                }
                current = current->next;
            }

            if (!fruit.active) {
                fruit.active = true;
                fruit.position = (Vector2){GetRandomValue(0, (SCREEN_WIDTH /
```

```
SQUARE_SIZE) - 1) * SQUARE_SIZE + offset.x / 2,
                                        GetRandomValue(0, (SCREEN_HEIGHT /
SQUARE_SIZE) - 1) * SQUARE_SIZE + offset.y / 2};

                current = snakeHead;
                while (current != NULL) {
                    while ((fruit.position.x == current->segment.position.x) &&
                           (fruit.position.y == current->segment.position.y)) {
                        fruit.position = (Vector2){GetRandomValue(0,
(SCREEN_WIDTH / SQUARE_SIZE) - 1) * SQUARE_SIZE + offset.x / 2,
                                                   GetRandomValue(0,
(SCREEN_HEIGHT / SQUARE_SIZE) - 1) * SQUARE_SIZE + offset.y / 2};
                        current = snakeHead;
                    }
                    current = current->next;
                }
            }

            if ((snakeHead->segment.position.x < (fruit.position.x +
fruit.size.x) &&
                 (snakeHead->segment.position.x + snakeHead->segment.size.x) >
fruit.position.x) &&
                (snakeHead->segment.position.y < (fruit.position.y +
fruit.size.y) &&
                 (snakeHead->segment.position.y + snakeHead->segment.size.y) >
fruit.position.y)) {
                PlaySound(eat);
                AddSegment();
                fruit.active = false;
            }

            framesCounter++;
            if (gameOver) {
                PlaySound(defeat);
            }
        }
    } else {
        if (IsKeyPressed(KEY_ENTER)) {
            StopSound(defeat);
            InitGame();
            gameOver = false;
        }
    }
}
```

Dibujo del Juego

La función `DrawGame` se encarga de renderizar los elementos del juego en pantalla:

```
static void DrawGame(Sound defeat, Sound eat, Music music) {
    char counterText[20];
    Rectangle frameRec = {0.0f, 0.0f, (float)SQUARE_SIZE, (float)SQUARE_SIZE};
    BeginDrawing();

    ClearBackground(SKYBLUE);

    if (!gameOver) {
        UpdateMusicStream(music);

        for (int i = 0; i < SCREEN_WIDTH / SQUARE_SIZE + 1; i++) {
            DrawLineV((Vector2){SQUARE_SIZE * i + offset.x /

 2, offset.y / 2},
                      (Vector2){SQUARE_SIZE * i + offset.x / 2, SCREEN_HEIGHT -
offset.y / 2}, DARKGRAY);
        }

        for (int i = 0; i < SCREEN_HEIGHT / SQUARE_SIZE + 1; i++) {
            DrawLineV((Vector2){offset.x / 2, SQUARE_SIZE * i + offset.y / 2},
                      (Vector2){SCREEN_WIDTH - offset.x / 2, SQUARE_SIZE * i +
offset.y / 2}, DARKGRAY);
        }

        SnakeNode *current = snakeHead;
        while (current != NULL) {
            DrawTextureRec(current->segment.sprite, frameRec, Vector2{current-
>segment.position.x, current->segment.position.y}, RAYWHITE);
            current = current->next;
        }

        DrawTextureRec(fruit.sprite, frameRec, Vector2{fruit.position.x,
fruit.position.y}, RAYWHITE);
        sprintf(counterText, "PUNTUACION: %d", counterTail - 1);
        DrawText(counterText, 20, 430, 10, BLACK);
        if (pause) {
            DrawText("PAUSA", SCREEN_WIDTH / 2 - MeasureText("PAUSA", 40) / 2,
SCREEN_HEIGHT / 2 - 40, 40, BLACK);
        } else {
            DrawText("PAUSA [P] ", 720, 430, 10, BLACK);
            DrawText("SALIR [ESC] ", 720, 415, 10, BLACK);
        }
    } else {
        DrawText("CULEBRITA", SCREEN_WIDTH / 2 - MeasureText("CULEBRITA", 30) /
2, SCREEN_HEIGHT / 2 - 90, 30, RED);
        DrawText("PRESIONA [ENTER] PARA JUGAR DE NUEVO", SCREEN_WIDTH / 2 -
MeasureText("PRESIONA [ENTER] PARA JUGAR DE NUEVO", 20) / 2, SCREEN_HEIGHT / 2
- 50, 20, BLACK);
    }

    EndDrawing();
}
```

Otros Métodos

Además, el código contiene funciones para añadir segmentos a la serpiente (`AddSegment`), liberar la memoria de la serpiente (`FreeSnake`), dibujar la pantalla de inicio (`DrawStartScreen`) y manejar la lógica de actualización y dibujo del juego (`UpdateDrawFrame`).

Conclusión

Este reporte ofrece una visión general del código del juego "Culebrita", explicando sus componentes principales y la lógica que permite el funcionamiento del juego. El uso de `raylib` facilita la implementación de la lógica de juego y la gestión de recursos gráficos y de audio.