Paradigmas de la programación

Diego D. Quiros Vicencio

Manual de reportes.

Fecha de Entrega: 30 de mayo de 2024.

Reporte Practica 1 de la Clase Paradigmas de Programacion

Alumno Diego Demian Quiros Vicencio - 372688

Minijuego Snake

Instrucciones

Recrear el juego Snake en Lenguaje C siguiendo los siguientes *Requisitos*:

- Listas Enlazadas
- NO usar variables globales
- NO usar codigo espagueti

El desarrollo del juego lo realice haciendo uso de la libreria "Raylib" la cual esta enfocada en el desarrollo de juegos en lenguaje C.

Desarrollo

Los elementos fundamentales en este juego son:

Serpiente

Es el principal protagonista, el jugador, la cual interactuara directamente con el mapa, el objetivo es consumir la mayor cantidad de manzanas, por cada manzana que coma la serpiente crecera su tamaño y si colisiona con su cuerpo... Se acaba el juego.

Manzanas

Estos son otros objetos con distintas propiedades, se generan en algun lado aleatorio del mapa y cuando la vibora las come, estas desaparecen y vuelven a aparecer en una ubicacion aleatoria.

Partes fundamentales del codigo.

Macros para la pantalla y librerias

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

#define ANCHO_PANTALLA 1280
#define ALTO_PANTALLA 720
#define TAMANO_CUADRADO 20
#define LONGITUD_MAX_SERPIENTE 100
#define MAX_COMIDA 4
```

Estructuras y nodos para las listas enlazadas

```
typedef struct Nodo {
    Vector2 posicion;
    struct Nodo* siguiente;
}Nodo;

typedef struct {
    Nodo* cabeza;
    Vector2 velocidad;
    int longitud;
}Serpiente;

typedef struct Comida {
    Vector2 posicion;
    bool activa;
} Comida;
```

Prototipos de funciones

Estas son las declaraciones de las funciones que se utilizaran a lo largo del programa, declararlas al principio es una buena practica para la estructura del programa.

```
void IniciarJuego(Serpiente* serpiente, Comida* comida);
void ActualizarJuego(Serpiente* serpiente, Comida* comida);
void DibujarJuego(Serpiente* serpiente, Comida* comida);
void DibujarFondo();
void ReiniciarJuego(Serpiente* serpiente, Comida* comida);
```

Iniciar Juego

Esta función se encarga de inicializar el estado del juego. Crea la serpiente y la comida en posiciones aleatorias dentro de la ventana de juego.

```
void IniciarJuego(Serpiente* serpiente, Comida* comida)
{
    if (gameOver) return;
    // Inicializa la serpiente en el centro de la pantalla
    serpiente->cabeza = (Nodo*)malloc(sizeof(Nodo));
    serpiente->cabeza->posicion = Vector2{ ANCHO_PANTALLA / 2.0f, ALTO_PANTALLA /
2.0f };
    serpiente->cabeza->siguiente = NULL;
    serpiente->velocidad = Vector2{ TAMANO_CUADRADO, 0 };
    serpiente->longitud = 1;
    //Inicializa la comida en posiciones aleatorias
    for (int i = 0; i < MAX_COMIDA; i++) {
        comida[i].posicion = Vector2{ static_cast<float>(GetRandomValue(0,
```

```
(ANCHO_PANTALLA / TAMANO_CUADRADO) - 1) * TAMANO_CUADRADO),
                            static_cast<float>(GetRandomValue(0, (ALTO_PANTALLA /
TAMANO_CUADRADO) - 1) * TAMANO_CUADRADO) };
        comida[i].activa = true;
    }
}
```

Actualizar Juego

Esta función se encarga de actualizar el estado del juego en cada iteración del bucle principal. Gestiona el movimiento de la serpiente, detecta colisiones con la comida y con ella misma, y actualiza la longitud de la serpiente.

```
void ActualizarJuego(Serpiente* serpiente, Comida* comida)
{

    // Movimiento de la serpiente, verifica que no pueda moverse en la dirección
opuesta
    if (IsKeyPressed(KEY_RIGHT) && serpiente->velocidad.x == 0) { serpiente-
>velocidad = Vector2{ TAMANO_CUADRADO, 0 }; }
    if (IsKeyPressed(KEY_LEFT) && serpiente->velocidad.x == 0) { serpiente-
>velocidad = Vector2{ -TAMANO_CUADRADO, 0 }; }
    if (IsKeyPressed(KEY_UP) && serpiente->velocidad.y == 0) { serpiente-
>velocidad = Vector2{ 0, -TAMANO_CUADRADO }; }
    if (IsKeyPressed(KEY_DOWN) && serpiente->velocidad.y == 0) { serpiente-
>velocidad = Vector2{ 0, TAMANO_CUADRADO }; }


    // Mueve la serpiente en la dirección que se encuentra actualmente y agrega un
nuevo nodo en la cabeza de la serpiente
    Nodo* nuevoNodo = (Nodo*)malloc(sizeof(Nodo));
    nuevoNodo->posicion = Vector2{ serpiente->cabeza->posicion.x + serpiente-
>velocidad.x,
                                   serpiente->cabeza->posicion.y + serpiente-
>velocidad.y };
    nuevoNodo->siguiente = serpiente->cabeza;
    serpiente->cabeza = nuevoNodo;
    serpiente->longitud++;

    // Verifica si la serpiente se sale de la pantalla, si sale por un lado,
aparece por el otro
    if (serpiente->cabeza->posicion.x >= ANCHO_PANTALLA) serpiente->cabeza-
>posicion.x = 0;
    else if (serpiente->cabeza->posicion.x < 0) serpiente->cabeza->posicion.x =
ANCHO_PANTALLA - TAMANO_CUADRADO;
    if (serpiente->cabeza->posicion.y >= ALTO_PANTALLA) serpiente->cabeza-
>posicion.y = 0;
    else if (serpiente->cabeza->posicion.y < 0) serpiente->cabeza->posicion.y =
ALTO_PANTALLA - TAMANO_CUADRADO;
```

```
    // Verifica si la serpiente se come la comida y agrega un nuevo nodo en la
cola de la serpiente
    for (int i = 0; i < MAX_COMIDA; i++) {
        if (comida[i].activa && serpiente->cabeza->posicion.x ==
comida[i].posicion.x && serpiente->cabeza->posicion.y == comida[i].posicion.y) {
            comida[i].posicion = Vector2{ static_cast<float>(GetRandomValue(0,
(ANCHO_PANTALLA / TAMANO_CUADRADO) - 1) * TAMANO_CUADRADO),
                                          static_cast<float>(GetRandomValue(0,
(ALTO_PANTALLA / TAMANO_CUADRADO) - 1) * TAMANO_CUADRADO) };
            comida[i].activa = true;
            serpiente->longitud++;
        }
    }

    // Verifica si la serpiente se come a si misma
    Nodo* temp = serpiente->cabeza;
    for (int i = 0; i < serpiente->longitud; i++) {
        if (temp->siguiente == NULL) {
            break;
        }
        temp = temp->siguiente;
    }
```

DibujarJuego

Esta función se encarga de dibujar el estado actual del juego en la ventana. Dibuja la serpiente, la comida y el fondo del juego.

```
void DibujarJuego(Serpiente* serpiente, Comida* comida)
{
    // Dibuja el juego
    //
    if (gameOver) {
        DrawText("Perdiste :(", ANCHO_PANTALLA / 2 - MeasureText("Perdiste :(",
40) / 2, ALTO_PANTALLA / 2 - 40, 40, RED);
        DrawText("Presiona [ESPACIO] para reiniciar", ANCHO_PANTALLA / 2 -
MeasureText("Presiona [ESPACIO] para reiniciar", 20) / 2, ALTO_PANTALLA / 2, 20,
DARKGRAY);
    }
    else {
        // Dibuja la serpiente en la pantalla
        DibujarFondo();
        Nodo* temp = serpiente->cabeza;
        while (temp != NULL) {
            DrawRectangleV(temp->posicion, Vector2{ TAMANO_CUADRADO,
TAMANO_CUADRADO }, BLUE);
            temp = temp->siguiente;
        }
        // Dibuja la comida
        for (int i = 0; i < MAX_COMIDA; i++) {
            if (comida[i].activa) {
```

```
                DrawRectangleV(comida[i].posicion, Vector2{ TAMANO_CUADRADO,
TAMANO_CUADRADO }, RED);
            }
        }

    }
}
```

DibujarFondo

Esta función dibuja el fondo del juego. Genera un patrón de cuadrados de dos colores en la ventana.

```
void DibujarFondo() {
    // Dibuja el fondo del juego con cuadros verdes y oscuros intercalados en la
pantalla
    for (int i = 0; i < ANCHO_PANTALLA / TAMANO_CUADRADO; i++) {
        for (int j = 0; j < ALTO_PANTALLA / TAMANO_CUADRADO; j++) {
            if ((i + j) % 2 == 0) {
                DrawRectangle(i * TAMANO_CUADRADO, j * TAMANO_CUADRADO,
TAMANO_CUADRADO, TAMANO_CUADRADO, GREEN);
            }
            else {
                DrawRectangle(i * TAMANO_CUADRADO, j * TAMANO_CUADRADO,
TAMANO_CUADRADO, TAMANO_CUADRADO, DARKGREEN);
            }
        }
    }
}
```

ReiniciarJuego

Esta función reinicia el estado del juego cuando el jugador pierde. Libera la memoria utilizada por la serpiente anterior, reinicia la serpiente y la comida, y restablece el indicador de juego terminado (gameOver) a falso.

```
void ReiniciarJuego(Serpiente* serpiente, Comida* comida) {
    // Reinicia el juego
    while (serpiente->cabeza != NULL) {
        // Libera la memoria de la serpiente
        Nodo* eliminar = serpiente->cabeza;
        serpiente->cabeza = serpiente->cabeza->siguiente;
        free(eliminar);
    }
    // Inicia el juego nuevamente
    IniciarJuego(serpiente, comida);
    gameOver = false;

}
```

Resultado