Paradigmas de la programación

Hernández Ceseña Iván Fernando

Práctica 1.

Snake en C

29 de Mayo del 2024

Instrucciones

crear el juego de Snake en C siguiendo los siguientes pasos:

- Listas enlazadas

- No usar variables globales

- No usar codigo espagueti

Desarrollo

Este juego fue creado con la libreria Raylib la cual nos permite desarrollar juegos en lenguaje C de manera sencilla.

Elementos del juego:

- Serpiente: esta es el jugador, esta compuesta por nodos que representan cada parte de la serpiente.

- Comida: esta es el objetivo del juego, al comerla la serpiente crece.

Declaraciones estructuras

```c
// Estructuras
typedef struct Nodo {
    Vector2 posicion;
    Vector2 velocidad;
    struct Nodo* siguiente;
} Nodo;

typedef struct {
    Nodo* cabeza;
    Nodo* cola;
    Color colorCabeza;
    Color colorCuerpo;
} Serpiente;

typedef struct {
    Vector2 posicion;
    Vector2 tamano;
    bool activa;
    Color color;
} Comida;

typedef struct {
    int anchoPantalla;
    int altoPantalla;
    int contadorFrames;
    bool juegoTerminado;
    Comida fruta;
    Serpiente serpiente;
    Vector2 offset;
} Juego;
```

Funciones

```
// Funciones principales
static void InicializarJuego(Juego* juego);
static void ActualizarJuego(Juego* juego);
static void DibujarJuego(const Juego* juego);
static void ActualizarDibujoFrame(Juego* juego);

// Funciones para el manejo de la serpiente
static Nodo* CrearNodo(Vector2 posicion, Vector2 velocidad);
static void AnadirNodo(Serpiente* serpiente);
static void LiberarSerpiente(Serpiente* serpiente);
```

Inicializacion del juego

```
void InicializarJuego(Juego* juego)
{
    juego->contadorFrames = 0;
    juego->juegoTerminado = false;

    juego->offset.x = juego->anchoPantalla % TAMANO_CUADRADO;
    juego->offset.y = juego->altoPantalla % TAMANO_CUADRADO;

    // inicializa la serpiente
    juego->serpiente.cabeza = CrearNodo((Vector2){ juego->offset.x /
    2, juego->offset.y / 2 }, (Vector2){ TAMANO_CUADRADO, 0 });
    juego->serpiente.cola = juego->serpiente.cabeza;
    juego->serpiente.colorCabeza = DARKBLUE;
    juego->serpiente.colorCuerpo = BLUE;

    // inicializa la comida
    juego->fruta.tamano = (Vector2){ TAMANO_CUADRADO,
    TAMANO_CUADRADO };
    juego->fruta.color = SKYBLUE;
    juego->fruta.activa = false;
}
```

Actualización del juego

```c
void ActualizarJuego(Juego* juego)
{
    if (!(juego->juegoTerminado))
    {
        // control del jugador
        if (IsKeyPressed(KEY_RIGHT) && (juego->serpiente.cabeza->velocidad.x == 0))
        {
            juego->serpiente.cabeza->velocidad = (Vector2){ TAMANO_CUADRADO, 0 };
        }
        if (IsKeyPressed(KEY_LEFT) && (juego->serpiente.cabeza->velocidad.x == 0))
        {
            juego->serpiente.cabeza->velocidad = (Vector2){ -TAMANO_CUADRADO, 0 };
        }
        if (IsKeyPressed(KEY_UP) && (juego->serpiente.cabeza->velocidad.y == 0))
        {
            juego->serpiente.cabeza->velocidad = (Vector2){ 0, -TAMANO_CUADRADO };
        }
        if (IsKeyPressed(KEY_DOWN) && (juego->serpiente.cabeza->velocidad.y == 0))
        {
            juego->serpiente.cabeza->velocidad = (Vector2){ 0, TAMANO_CUADRADO };
        }

        // movimiento de la serpiente
        if ((juego->contadorFrames % 5) == 0)
        {
            Nodo* actual = juego->serpiente.cabeza;
            Vector2 posicionAnterior = actual->posicion;
            Vector2 posicionTemp;

            actual->posicion.x += actual->velocidad.x;
            actual->posicion.y += actual->velocidad.y;

            while (actual->siguiente != NULL)
            {
                actual = actual->siguiente;
                posicionTemp = actual->posicion;
                actual->posicion = posicionAnterior;
                posicionAnterior = posicionTemp;
            }
        }

        // colision con los limites de la matriz
        if ((juego->serpiente.cabeza->posicion.x >= (juego->anchoPantalla - juego->offset.x)) ||
            (juego->serpiente.cabeza->posicion.y >= (juego->altoPantalla - juego->offset.y)) ||
            (juego->serpiente.cabeza->posicion.x < 0) || (juego->serpiente.cabeza->posicion.y < 0))
        {
            juego->juegoTerminado = true;
        }
```

```
        // colision con la cola de la serpiente
        Nodo* actual = juego->serpiente.cabeza->siguiente;
        while (actual != NULL)
        {
            if ((juego->serpiente.cabeza->posicion.x == actual->posicion.x) && (juego->serpiente.cabeza->posicion.y == actual->posicion.y))
            {
                juego->juegoTerminado = true;
            }
            actual = actual->siguiente;
        }

        // colision de la cabeza de la serpiente con la comida
        if ((juego->serpiente.cabeza->posicion.x < (juego->fruta.posicion.x + juego->fruta.tamano.x) && (juego->serpiente.cabeza->posicion.x + TAMANO_CUADRADO) > juego->fruta.posicion.x) &&
            (juego->serpiente.cabeza->posicion.y < (juego->fruta.posicion.y + juego->fruta.tamano.y) && (juego->serpiente.cabeza->posicion.y + TAMANO_CUADRADO) > juego->fruta.posicion.y))
        {
            AnadirNodo(&(juego->serpiente));
            juego->fruta.activa = false;
        }

        // posicion de la fruta
        if (!(juego->fruta.activa))
        {
            juego->fruta.activa = true;
            juego->fruta.posicion = (Vector2){ GetRandomValue(0, (juego->anchoPantalla / TAMANO_CUADRADO) - 1) * TAMANO_CUADRADO + juego->offset.x / 2, GetRandomValue(0, (juego->altoPantalla / TAMANO_CUADRADO) - 1) * TAMANO_CUADRADO +
            juego->offset.y / 2 };

            Nodo* actual = juego->serpiente.cabeza;
            while (actual != NULL)
            {
                while ((juego->fruta.posicion.x == actual->posicion.x) && (juego->fruta.posicion.y == actual->posicion.y))
                {
                    juego->fruta.posicion = (Vector2){ GetRandomValue(0, (juego->anchoPantalla / TAMANO_CUADRADO) - 1) * TAMANO_CUADRADO + juego->offset.x / 2, GetRandomValue(0, (juego->altoPantalla / TAMANO_CUADRADO) - 1) *
                    TAMANO_CUADRADO + juego->offset.y / 2 };
                    actual = juego->serpiente.cabeza;
                }
                actual = actual->siguiente;
            }
        }

        juego->contadorFrames++;
    }
    else
    {
        if (IsKeyPressed(KEY_ENTER))
        {
            LiberarSerpiente(&(juego->serpiente)); // Libera la memoria de la serpiente antes de reiniciar el juego
            InicializarJuego(juego);
            juego->juegoTerminado = false;
        }
    }
}
```

Dibujo del juego

```
void DibujarJuego(const Juego* juego)
{
    BeginDrawing();

    ClearBackground(RAYWHITE);

    if (!(juego->juegoTerminado))
    {
        // dibuja la matriz
        for (int i = 0; i < juego->anchoPantalla / TAMANO_CUADRADO + 1; i++)
        {
            DrawLineV((Vector2){ TAMANO_CUADRADO * i + juego->offset.x / 2, juego->offset.y / 2 },
            (Vector2){ TAMANO_CUADRADO * i + juego->offset.x / 2, juego->altoPantalla - juego->offset.
            y / 2 }, LIGHTGRAY);
        }

        for (int i = 0; i < juego->altoPantalla / TAMANO_CUADRADO + 1; i++)
        {
            DrawLineV((Vector2){ juego->offset.x / 2, TAMANO_CUADRADO * i + juego->offset.y / 2 },
            (Vector2){ juego->anchoPantalla - juego->offset.x / 2, TAMANO_CUADRADO * i + juego->offset.
            y / 2 }, LIGHTGRAY);
        }

        // dibuja la serpiente
        Nodo* actual = juego->serpiente.cabeza;
        while (actual != NULL)
        {
            DrawRectangleV(actual->posicion, (Vector2){ TAMANO_CUADRADO, TAMANO_CUADRADO }, (actual ==
            juego->serpiente.cabeza) ? juego->serpiente.colorCabeza : juego->serpiente.colorCuerpo);
            actual = actual->siguiente;
        }

        // dibuja la comida
        DrawRectangleV(juego->fruta.posicion, juego->fruta.tamano, juego->fruta.color);
    }
    else
    {
        DrawText("PRESIONA [ENTER] PARA JUGAR DE NUEVO O ESC PARA CERRAR EL JUEGO", GetScreenWidth() /
        2 - MeasureText("PRESIONA [ENTER] PARA JUGAR DE NUEVO O ESC PARA CERRAR EL JUEGO", 20) / 2,
        GetScreenHeight() / 2 - 50, 20, GRAY);
    }

    EndDrawing();
}
```

Funcion para actualizar y dibujar el juego

```
void ActualizarDibujoFrame(Juego* juego)
{
    ActualizarJuego(juego);
    DibujarJuego(juego);
}
```

Imagen del Juego