UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA: INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN (TI)

ASIGNATURA: ESTRUCTURA DE DATOS – NRC: 14128

TEMÁTICA: PROYECTO DE TERCER PARCIAL

GRUPO: JUEGO DE LA SERPIENTE

COLINA MORALES MATEO RUBEN

CEDEÑO ZAPATA JHOIS MICAEL

CHANCUSIG GUAMÁN ERICK ARIEL

RAMÍREZ CEVALLOS CARLOS ESTÉFANO

TUTOR: ING. ALVAREZ JIMENEZ MAYRA ISABEL

FECHA: 07/03/2024

Contenido

ÍNDICE DE FIGURAS

Fase y explicación del código de programación

Inclusión de bibliotecas

- SFML/Graphics.hpp y SFML/Audio.hpp son bibliotecas para gráficos y audio en SFML.
- Vector es una biblioteca para contenedores dinámicos de C++.
- iostream es la biblioteca estándar de C++ para entrada/salida

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <vector>
#include <iostream>
```

Figura 1. Bibliotecas usadas en el programa

Espacio de nombres y estructura

- Se utiliza using namespace para evitar escribir repetidamente std:: y sf::.
- Se define una estructura llamada Punto con dos variables enteras x e y.

```
using namespace std;
using namespace sf;

struct Punto {
    int x, y;
};
```

Figura 2. Definición de estructura

Funciones de colisión

- **ColisionConSerpiente:** Esta función verifica si la cabeza de la serpiente colisiona con alguna parte de su cuerpo. Recorre el vector de puntos que representa la serpiente y compara las coordenadas de la cabeza con cada punto.

```
bool colisionConSerpiente(const Punto& cabeza, const vector<Punto>& serpiente) {
    for (auto& parte : serpiente) {
        if (parte.x == cabeza.x && parte.y == cabeza.y)
            return true;
    }
    return false;
}
```

Figura 3. Función ColisiónConSerpiente

ColisionConBorde: Verifica si la cabeza de la serpiente colisiona con el borde de la ventana. Utiliza las dimensiones de la ventana para determinar si la posición de la cabeza está fuera de los límites.

```
// Función para verificar si la cabeza de la serpiente colisiona con el borde de la ventana
bool colisionConBorde(const Punto& cabeza, const RenderWindow& ventana) {
    return static_cast<unsigned int>(cabeza.x) < 0 || static_cast<unsigned int>(cabeza.y) < 0 || static_cast<unsigned int>(cabeza.x) >= ventana.getSize().x /
}
```

Figura 4. Función ColisionConParedes

Funciones implementadas para cargar sonidos y música

La función cargar sonidos carga los archivos de sonido para la comida, colisiones y logros. Si hay algún problema con la carga, muestra un mensaje de error y termina la ejecución del programa.

```
// Función para cargar los sonidos
void cargarSonidos(SoundBuffer& bufferComer, SoundBuffer& bufferColision, SoundBuffer& bufferHito, Sound& sonidoComer, Sound& sonidoColision, Sound& sonidoHi
    if (!bufferComer.loadFromFile("comer.wav")) {
        cout << "Error al cargar el archivo de sonido 'comer.wav'" << endl;
        exit(-1);
    }
    sonidoComer.setBuffer(bufferComer);

    if (!bufferColision.loadFromFile("colision.wav")) {
        cout << "Error al cargar el archivo de sonido 'colision.wav'" << endl;
        exit(-1);
    }
    sonidoColision.setBuffer(bufferColision);

    if (!bufferHito.loadFromFile("logro.wav")) {
        cout << "Error al cargar el archivo de sonido 'logro.wav'" << endl;
        exit(-1);
    }
    sonidoHito.setBuffer(bufferHito);
}
```

Figura 5. Función para cargar los sonidos

La función cargar música, carga un archivo de música de fondo. Si hay un problema con la carga, muestra un mensaje de error y termina la ejecución. Configura la música para que se repita continuamente

```
// Función para cargar la música de fondo
void cargarMusica(Music& musicaFondo) {
    if (!musicaFondo.openFromFile("musica.wav")) {
        cout << "Error al cargar el archivo de música 'musica.wav'" << endl;
        exit(-1);
    }
    musicaFondo.setLoop(true); // hace que la música se repita
    musicaFondo.play();
}
```

Figura 6. Función para cargar la música de fondo

Función para crear la ventana en modo de pantalla completa

Crea una ventana en modo de pantalla completa con el título "Juego de Serpiente". También configura un objeto RectangleShape llamado pixel que se usará para representar elementos gráficos en el juego.

```cpp
// Función para crear la ventana en modo de pantalla completa
void crearVentana(RenderWindow& ventana, RectangleShape& pixel) {
    ventana.create(VideoMode::getDesktopMode(), "Juego de Serpiente", Style::Fullscreen);
    pixel.setSize(Vector2f(20, 20));
    pixel.setOutlineThickness(1);
    pixel.setOutlineColor(Color::Black);
}
```

Figura 7. Función para correr el juego en pantalla completa

Función para inicializar la serpiente y la comida generada

Inicializa la serpiente y la comida. La serpiente comienza con una posición aleatoria y la comida también se coloca en una posición aleatoria que no coincide con ninguna parte de la serpiente.

```cpp
// Función para inicializar la serpiente y la comida
void inicializarSerpienteYComida(vector<Punto>& serpiente, vector<Punto>& comidas, Punto& comida) {
    serpiente.push_back({rand() % 40, rand() % 30}); // Posición inicial aleatoria de la serpiente

    bool comidaEnSerpiente;
    do {
        comidaEnSerpiente = false;
        comida = {rand() % 40, rand() % 30}; // Posición inicial aleatoria de la comida
        for (Punto p : serpiente) {
            if (p.x == comida.x && p.y == comida.y) {
                comidaEnSerpiente = true;
                break;
            }
        }
    } while (comidaEnSerpiente);
    comidas.push_back(comida);
}
```

Figura 8. Función que hace aparecer comida y serpiente

Funciones para crear la fuente y crear el texto de la puntuación

La función CargarFuente carga una fuente específica para el juego llamada "Montserrat-Black.ttf". Si hay algún problema con la carga, muestra un mensaje de error y termina la ejecución.

```cpp
// Función para cargar la fuente
void cargarFuente(Font& fuente) {
    fuente.loadFromFile("Montserrat-Black.ttf");
}
```

Figura 9. Función que extrae la fuente Montrserrat

La función creartextoPuntuación configura el texto de la puntuación. Le asigna una fuente, tamaño, color y posición específicos en la ventana. Este texto se usará para mostrar la puntuación actual del jugador.

```cpp
// Función para crear el texto de la puntuación
void crearTextoPuntuacion(Text& textoPuntuacion, Font& fuente) {
    textoPuntuacion.setFont(fuente);
    textoPuntuacion.setCharacterSize(24);
    textoPuntuacion.setFillColor(Color::White);
    textoPuntuacion.setPosition(10, 10);
}
```

Figura 10. Función para crear texto de puntuación

Función para seleccionar la velocidad

```
// Función para crear el texto de la velocidad
void crearTextoVelocidad(Text& textoVelocidad, Font& fuente) {
    textoVelocidad.setFont(fuente);
    textoVelocidad.setString("\n\n+30 puntos: Velocidad aumentada\n+100 puntos: Logro desbloqueado\n+150 puntos: Otra bola más de comida");
    textoVelocidad.setCharacterSize(24);
    textoVelocidad.setFillColor(Color::White);
    textoVelocidad.setPosition(10, 40); // Posiciona el texto debajo del texto de la puntuación
}
```

Figura 11. Función para crear los textos de subida de puntos

La función crearTextoVelocidad tiene como propósito inicializar el objeto de texto destinado a mostrar información sobre la velocidad del juego y ciertos logros asociados. Esta función toma como entrada un objeto text para representar el texto de velocidad; la fuente utilizada para la apariencia del texto, y configura su contenido, tamaño, color y posición en la ventana de juego. La salida de la función es la modificación del objeto text pasado como argumento, listo para ser dibujado en la ventana.

La lógica de la función implica la asignación de la fuente a utilizar, la configuración del contenido del texto para describir logros y cambios de velocidad asociados a la puntuación del jugador, y finalmente, la posición del texto en la ventana. Este texto informativo se sitúa debajo del texto de la puntuación, proporcionando detalles sobre los logros alcanzados y cómo estos afectan la dinámica del juego, brindando así una experiencia interactiva y clara para el jugador.

Función para crear los botones

```
// Función para crear los botones
void crearBotones(Text& botonJugar, Text& botonCerrar, Text& botonReintentar, Text& botonFinalizar, Font& fuente, RenderWindow& ventana) {
    botonJugar.setFont(fuente);
    botonJugar.setString("Jugar");
    botonJugar.setCharacterSize(50);
    botonJugar.setFillColor(Color::Green);
    botonJugar.setPosition(ventana.getSize().x / 2 - botonJugar.getGlobalBounds().width / 2 - 250, ventana.getSize().y / 2 + 350); // Mueve el

    botonCerrar.setFont(fuente);
    botonCerrar.setString("Cerrar");
    botonCerrar.setCharacterSize(50);
    botonCerrar.setFillColor(Color::Red);
    botonCerrar.setPosition(ventana.getSize().x / 2 - botonCerrar.getGlobalBounds().width / 2 + 250, ventana.getSize().y / 2 + 350); // Mueve

    botonReintentar.setFont(fuente);
    botonReintentar.setString("Reintentar");
    botonReintentar.setCharacterSize(50);
    botonReintentar.setFillColor(Color::Yellow);
    botonReintentar.setPosition(ventana.getSize().x / 2 - botonReintentar.getGlobalBounds().width / 2 - 200, ventana.getSize().y / 2 + 200);

    botonFinalizar.setFont(fuente);
    botonFinalizar.setString("Finalizar");
    botonFinalizar.setCharacterSize(50);
    botonFinalizar.setFillColor(Color::Red);
    botonFinalizar.setPosition(ventana.getSize().x / 2 - botonFinalizar.getGlobalBounds().width / 2 + 200, ventana.getSize().y / 2 + 200);
}
```

Figura 12. Función para crear botones

La función crearBotones tiene como objetivo inicializar los objetos de texto destinados a representar botones interactivos en la interfaz del juego. Recibe como entrada varios objetos Text que representan los botones "Jugar", "Cerrar", "Reintentar" y

"Finalizar", además de la fuente (fuente) utilizada para el estilo de los textos y la ventana (ventana) donde se visualizarán estos elementos. La salida de la función es la modificación de estos objetos Text, listos para ser renderizados en la ventana del juego.

La lógica de la función incluye la configuración de cada botón con su respectivo texto, tamaño, color y posición en la ventana. Cada botón se coloca estratégicamente para proporcionar una interfaz intuitiva y facilitar la interacción del usuario. Estos botones permiten realizar acciones clave en el juego, como iniciar una partida, cerrar la aplicación, reintentar el juego tras una partida finalizada y finalizar la aplicación desde la pantalla de resultados. En conjunto, la función crearBotones contribuye a una experiencia de usuario clara y accesible en el juego Snake.

Función para seleccionar la dificultad

```
// Función para crear los botones de dificultad
void crearBotonesDificultad(Text& botonNormal, Text& botonDificil, Text& botonMuyDificil, Font& fuente, RenderWindow& ventana) {
    botonNormal.setFont(fuente);
    botonNormal.setString("Normal (Velocidad normal)");
    botonNormal.setCharacterSize(50);
    botonNormal.setFillColor(Color::White);
    botonNormal.setPosition(ventana.getSize().x / 2 - botonNormal.getGlobalBounds().width / 2, ventana.getSize().y / 2 - 100);

    botonDificil.setFont(fuente);
    botonDificil.setString("Difícil (Velocidad rápida)");
    botonDificil.setCharacterSize(50);
    botonDificil.setFillColor(Color::White);
    botonDificil.setPosition(ventana.getSize().x / 2 - botonDificil.getGlobalBounds().width / 2, ventana.getSize().y / 2);

    botonMuyDificil.setFont(fuente);
    botonMuyDificil.setString("Muy difícil (Velocidad muy rápida)");
    botonMuyDificil.setCharacterSize(50);
    botonMuyDificil.setFillColor(Color::White);
    botonMuyDificil.setPosition(ventana.getSize().x / 2 - botonMuyDificil.getGlobalBounds().width / 2, ventana.getSize().y / 2 + 100);
}
```

Figura 13. Función para crear los botones de dificultad

La función `crearBotonesDificultad` tiene como propósito principal inicializar los objetos de texto que representan botones de selección de dificultad en la interfaz del juego. Recibe como entrada varios objetos `Text`, tales como "Normal (Velocidad normal)", "Difícil (Velocidad rápida)" y "Muy difícil (Velocidad muy rápida)", junto con la fuente (`fuente`) utilizada para el estilo del texto y la ventana (`ventana`) en la que se mostrarán estos elementos. La salida de la función es la modificación de estos objetos `Text`, preparados para ser renderizados en la ventana del juego.

La lógica de la función implica la configuración de cada botón de dificultad con su respectivo texto, tamaño, color y posición en la ventana; estos botones ofrecen al jugador opciones de dificultad que afectan la velocidad del juego, brindando una experiencia adaptada a las preferencias y habilidades del usuario.

La disposición estratégica de los botones en la ventana facilita la elección del nivel de dificultad.

La función `crearBotonesDificultad` contribuye a la personalización de la experiencia de juego, permitiendo a los jugadores ajustar la dificultad según sus preferencias.

Función para poner los elementos de la pantalla inicial

```
// Función para crear el texto de la pantalla de inicio
void crearTextoInicio(Text& textoInicio1, Text& textoInicio2, Font& fuente, RenderWindow& ventana) {
    textoInicio1.setFont(fuente);
    textoInicio1.setString("JUEGO DE LA SERPIENTE\n\n Proyecto Tercer Parcial\n\n             NRC 14180");
    textoInicio1.setCharacterSize(50);
    textoInicio1.setFillColor(Color::White);
    textoInicio1.setPosition(ventana.getSize().x / 2 - textoInicio1.getGlobalBounds().width / 2, ventana.getSize().y / 9);

    textoInicio2.setFont(fuente);
    textoInicio2.setString
    ("Desarrollado por:\n\nColina Morales Mateo Rubén\nCedeño Zapata Jhois Micael\nChancusig Guaman Erick Ariel\nRamírez Cevallos Carlos Estéfano");
    textoInicio2.setCharacterSize(50);
    textoInicio2.setFillColor(Color::White);
    textoInicio2.setPosition(ventana.getSize().x / 2 - 500 - textoInicio2.getGlobalBounds().width / 2, ventana.getSize().y / 2.3);
}
```

Figura 14.Función para crear texto de pantalla inicial

La función `crearTextoInicio` tiene como objetivo principal inicializar objetos de texto para mostrar información sobre el inicio del juego, incluyendo el título del juego, detalles del proyecto y los nombres de los desarrolladores.

La función recibe como entrada varios objetos `Text`, tales como "JUEGO DE LA SERPIENTE" y la información del proyecto y los desarrolladores; también toma la fuente (`fuente`) utilizada para el estilo del texto y la ventana (`ventana`) donde se visualizarán estos elementos. La salida de la función es la modificación de estos objetos `Text`, listos para ser renderizados en la ventana del juego.

La lógica de la función incluye la configuración del contenido, tamaño, color y posición de cada objeto de texto, creando así una pantalla de inicio informativa y atractiva; estos textos forman parte de la presentación inicial del juego, proporcionando detalles sobre el proyecto y reconociendo a los desarrolladores responsables.

La disposición de los textos en la ventana garantiza una presentación ordenada y estéticamente agradable. En conjunto, la función `crearTextoInicio` contribuye a establecer un contexto informativo y atractivo para los jugadores al iniciar el juego Snake.

Función que permite empezar a jugar

```
// Función para iniciar el juego
void iniciarJuego(bool& juegoIniciado, bool& juegoTerminado, vector<Punto>& serpiente,
                  vector<Punto>& comidas, Punto& comida, Punto& dir, int& puntuacion, int& logros) {
    juegoIniciado = true;
    juegoTerminado = false;
    serpiente.clear();
    serpiente.push_back({rand() % 40, rand() % 30}); // Posición inicial aleatoria de la serpiente
    comidas.clear();
    bool comidaEnSerpiente;
    do {
        comidaEnSerpiente = false;
        comida = {rand() % 40, rand() % 30}; // Posición inicial aleatoria de la comida
        for (Punto p : serpiente) {
            if (p.x == comida.x && p.y == comida.y) {
                comidaEnSerpiente = true;
                break;
            }
        }
    } while (comidaEnSerpiente);
    comidas.push_back(comida);

    dir = {1, 0};
    puntuacion = 0;
    logros = 0;
}
```

Figura 15. Función para inicializar el programa

La función `iniciarJuego` en el código del juego tiene como propósito reiniciar o iniciar el juego con valores iniciales. Las entradas de esta función incluyen referencias a variables que representan el estado del juego, la serpiente, las comidas, la posición de la comida, la dirección de la serpiente, la puntuación y los logros. La función no tiene salidas explícitas, ya que modifica directamente las variables de estado proporcionadas.

La lógica de la función comienza estableciendo el estado del juego como iniciado y no terminado. Luego, limpia la serpiente y las comidas actuales, generando nuevas posiciones aleatorias para la serpiente y la comida. La dirección se reinicia a una dirección predeterminada, y la puntuación y los logros se establecen en cero. En resumen, la función restablece todos los elementos del juego a sus valores iniciales, preparando el entorno para un nuevo inicio del juego.

```
// Función principal
int main() {
    SoundBuffer bufferComer, bufferColision, bufferHito;
    Sound sonidoComer, sonidoColision, sonidoHito;
    cargarSonidos(bufferComer, bufferColision, bufferHito, sonidoComer, sonidoColision, sonidoHito);

    Music musicaFondo;
    cargarMusica(musicaFondo);

    RenderWindow ventana;
    RectangleShape pixel;
    crearVentana(ventana, pixel);

    vector<Punto> serpiente;
    vector<Punto> comidas;
    Punto comida;
    inicializarSerpienteYComida(serpiente, comidas, comida);

    Font fuente;
    cargarFuente(fuente);

    Text textoPuntuacion;
    crearTextoPuntuacion(textoPuntuacion, fuente);

    Text textoVelocidad;
    crearTextoVelocidad(textoVelocidad, fuente);

    Text botonJugar, botonCerrar, botonReintentar, botonFinalizar;
    crearBotones(botonJugar, botonCerrar, botonReintentar, botonFinalizar, fuente, ventana);

    Text botonNormal, botonDificil, botonMuyDificil;
    crearBotonesDificultad(botonNormal, botonDificil, botonMuyDificil, fuente, ventana);
```

Figura 16. Función principal y llamada a funciones previas

Primero, hemos creado tres objetos ‘SoundBuffer’ y tres objetos ‘Sound’ para manejar los efectos de sonido del juego, que incluyen el sonido de comer, el sonido de colisión y el sonido de golpe. Estos sonidos se cargan mediante la función ‘cargarSonidos’.

Además, utilizamos un objeto Music para manejar la música de fondo del juego. Este se carga mediante la función ‘cargarMusica’; para la interfaz gráfica del juego, hemos creado un objeto ‘RenderWindow’ para la ventana del juego y un objeto ‘RectangleShape’ para representar los píxeles en el juego; estos se configuran mediante la función ‘crearVentana’.

En cuanto a la lógica del juego, usamos dos vectores de Punto para representar la serpiente y las comidas en el juego, y un objeto Punto para representar una comida individual. Estos se inicializan mediante la función ‘inicializarSerpienteYComida’.

Para la presentación visual, creamos un objeto Font para manejar la fuente del juego, que se carga mediante la función ‘cargarFuente’. Luego, hemos creado varios objetos Text para representar diferentes elementos de texto en el juego, como la puntuación, la velocidad, los botones y los mensajes de inicio.

```
Text textoInicio1, textoInicio2;
crearTextoInicio(textoInicio1, textoInicio2, fuente, ventana);

bool juegoIniciado = false;
bool juegoTerminado = false;
bool comidaEnSerpiente;

Punto dir = {1, 0};
int puntuacion = 0;
int logros = 0;
int velocidad = 100;
```

Figura 17. Inicialización de variables

En esta sección inicializaremos las variables que en este caso son 'juegoIniciado', 'juegoTerminado', declaramos 'comidaEnSerpiente' y luego colocamos en cero la puntuación, logros y la velocidad. Esto lo hacemos para que al inicio de juego las variables tengan estos valores predefinidos y prevenir errores como que se asignen valores que no son. Se hace esto siempre al iniciar al juego, pues no sería recomendable que al iniciar el juego salga una puntuación aleatoria y no cero, cuando se está iniciando.

```
while (ventana.isOpen()) {
    Event evento;
    while (ventana.pollEvent(evento)) {
        if (evento.type == Event::Closed)
            ventana.close();
        else if (evento.type == Event::MouseButtonPressed) {
            if (botonJugar.getGlobalBounds().contains(ventana.mapPixelToCoords(Mouse::getPosition(ventana)))) {
                // Muestra la pantalla de selección de dificultad
                ventana.clear();
                ventana.draw(botonNormal);
                ventana.draw(botonDificil);
                ventana.draw(botonMuyDificil);
                ventana.display();
                while (true) {
                    Event eventoDificultad;
                    while (ventana.pollEvent(eventoDificultad)) {
                        if (eventoDificultad.type == Event::MouseButtonPressed) {
                            if (botonNormal.getGlobalBounds().contains(ventana.mapPixelToCoords(Mouse::getPosition(ventana)))) {
                                velocidad = 100; // Velocidad normal
                                iniciarJuego(juegoIniciado, juegoTerminado, serpiente, comidas, comida, dir, puntuacion, logros);
                                break;
                            }
                            else if (botonDificil.getGlobalBounds().contains(ventana.mapPixelToCoords(Mouse::getPosition(ventana)))) {
                                velocidad = 70; // Velocidad rápida
                                iniciarJuego(juegoIniciado, juegoTerminado, serpiente, comidas, comida, dir, puntuacion, logros);
                                break;
                            }
                            else if (botonMuyDificil.getGlobalBounds().contains(ventana.mapPixelToCoords(Mouse::getPosition(ventana)))) {
                                velocidad = 40; // Velocidad muy rápida
                                iniciarJuego(juegoIniciado, juegoTerminado, serpiente, comidas, comida, dir, puntuacion, logros);
                                break;
                            }
                        }
                    }
                }
```

Figura 18. Parte del bucle principal

Esta es parte del bucle principal del juego. En este segmento, se manejan los eventos de la ventana y se controla la interacción del usuario con la interfaz del juego. El bucle while externo se ejecuta mientras la ventana del juego esté abierta. Dentro de este bucle, se crea un objeto Event y se usa otro bucle while para procesar todos los eventos pendientes. Si el evento es de tipo Closed, se cierra la ventana del juego. Si el evento es de tipo MouseButtonPressed, se verifica si el botón de jugar ha sido presionado. Si es así, se

muestra la pantalla de selección de dificultad, que incluye tres botones: normal, difícil y muy difícil.

Luego, se entra en otro bucle while para manejar la selección de dificultad. Dentro de este bucle, se procesan los eventos de la ventana y si se presiona el botón del mouse, se verifica cuál botón de dificultad ha sido presionado. Dependiendo del botón presionado, se establece la velocidad del juego y se inicia el juego con la función iniciarJuego. Este proceso se repite hasta que el juego se inicie, momento en el que se rompe el bucle de selección de dificultad.

```
            else if (botonCerrar.getGlobalBounds().contains(ventana.mapPixelToCoords(Mouse::getPosition(ventana)))) {
                ventana.close();
            }
            else if (juegoTerminado && botonReintentar.getGlobalBounds().contains(ventana.mapPixelToCoords(Mouse::getPosition(ventana)))) {
                iniciarJuego(juegoIniciado, juegoTerminado, serpiente, comidas, comida, dir, puntuacion, logros);
            }
            else if (juegoTerminado && botonFinalizar.getGlobalBounds().contains(ventana.mapPixelToCoords(Mouse::getPosition(ventana)))) {
                ventana.close();
            }
        }
        else if (evento.type == Event::KeyPressed) {
            Punto nuevaDir;
            switch (evento.key.code) {
                case Keyboard::Up:    nuevaDir = { 0, -1}; break;
                case Keyboard::Down:  nuevaDir = { 0,  1}; break;
                case Keyboard::Left:  nuevaDir = {-1,  0}; break;
                case Keyboard::Right: nuevaDir = { 1,  0}; break;
                default: break;
            }
            // Comprueba si la nueva dirección es opuesta a la dirección actual
            if (nuevaDir.x != -dir.x || nuevaDir.y != -dir.y) {
                dir = nuevaDir;
            }
        }
    }

    if (!juegoIniciado) {
        ventana.clear();
        ventana.draw(textoInicio1);
        ventana.draw(textoInicio2);
        ventana.draw(botonJugar);
        ventana.draw(botonCerrar);
        ventana.display();
        continue;
    }
```

Figura 19. Control de la interacción del usuario

En esta sección, se manejan más eventos de la ventana y se controla la interacción del usuario con el juego. Si el juego ha iniciado y el botón de cerrar es presionado, la ventana del juego se cierra; si el juego ha terminado y el botón de reintentar es presionado, se reinicia el juego con la función iniciarJuego; si el juego ha terminado y el botón de finalizar es presionado, la ventana del juego se cierra.

Al presionar una tecla, se crea un objeto Punto para la nueva dirección de la serpiente. Dependiendo de la tecla presionada, se establece la nueva dirección; sin embargo, se verifica si la nueva dirección es opuesta a la dirección actual de la serpiente. Si es así, la dirección no cambia. Cuando no se ha iniciado, se limpia la ventana, se dibujan el texto de inicio y los botones de jugar y cerrar, y se muestra la ventana; luego, se salta el resto del bucle con la instrucción continue.

```
Punto siguiente = {serpiente.back().x + dir.x, serpiente.back().y + dir.y};
bool comio = false;
for (int i = 0; i < comidas.size(); i++) {
    if (siguiente.x == comidas[i].x && siguiente.y == comidas[i].y) {
        sonidoComer.play();
        do {
            comidaEnSerpiente = false;
            comida = {rand() % 40, rand() % 30}; // Nueva posición aleatoria de la comida
            for (Punto p : serpiente) {
                if (p.x == comida.x && p.y == comida.y) {
                    comidaEnSerpiente = true;
                    break;
                }
            }
            // Comprueba si la comida se genera dentro de los rectángulos de los textos
            if (textoPuntuacion.getGlobalBounds().contains(comida.x * 20, comida.y * 20) ||
                textoVelocidad.getGlobalBounds().contains(comida.x * 20, comida.y * 20)) {
                comidaEnSerpiente = true;
            }
        } while (comidaEnSerpiente);
        comidas[i] = comida;

        puntuacion += 10;
        if (puntuacion % 50 == 0 && velocidad > 30) { // Aumenta la velocidad cada 50 puntos
            velocidad -= 5; // Disminuye el tiempo de espera, lo que aumenta la velocidad
        }
        if (puntuacion % 100 == 0) { // Desbloquea un logro cada 100 puntos
            logros++;
            sonidoHito.play();
        }
        if (puntuacion % 150 == 0) { // Agrega una bola extra de comida cada 150 puntos
            comidas.push_back({rand() % 40, rand() % 30});
        }
        comio = true;
        break;
    }
}
if (!comio) {
    serpiente.erase(serpiente.begin());
}
```

Figura 20. Comprobación de generación de comida, desbloqueo de logros y aumento de velocidad y comida

Este bloque de código inicia con la definición de un punto **siguiente**, que representa la siguiente posición de la cabeza de la serpiente según la dirección actual. Luego, se utiliza un bucle **for** para iterar sobre las comidas existentes y se verifica si la cabeza de la serpiente colisiona con alguna comida. Si hay una colisión, se ejecuta un conjunto de acciones:

- Reproducción de un sonido (**sonidoComer**) indicando que la serpiente ha comido.
- Se utiliza un bucle **do-while** para generar una nueva posición aleatoria para la comida, evitando que aparezca dentro de la serpiente o en las posiciones ocupadas por los textos de puntuación y velocidad.
- Se actualiza la posición de la comida en el vector de comidas.
- Incremento de la puntuación en 10 puntos.
- Verificación de ciertos hitos de puntuación para ajustar la velocidad del juego, desbloquear logros y agregar comidas adicionales.

Finalmente, si la cabeza de la serpiente no colisiona con ninguna comida, se elimina la cola de la serpiente mediante **serpiente.erase(serpiente.begin())**.

```
if (colisionConSerpiente(siguiente, serpiente) || colisionConBorde(siguiente, ventana)) {
    sonidoColision.play();
    // Fin del juego
    juegoIniciado = false;
    juegoTerminado = true;
    Text textoFinal;
    textoFinal.setFont(fuente);
    textoFinal.setString("PUNTUACION FINAL: " + to_string(puntuacion) + "\nLOGROS: " + to_string(logros) + "\n\nGracias por jugar!");
    textoFinal.setCharacterSize(50);
    textoFinal.setFillColor(Color::White);
    textoFinal.setPosition(ventana.getSize().x / 2 - textoFinal.getGlobalBounds().width / 2, ventana.getSize().y / 4);
    ventana.clear();
    ventana.draw(textoFinal);
    ventana.draw(botonReintentar);
    ventana.draw(botonFinalizar);
    ventana.display();
    while (juegoTerminado) {
        Event eventoFinal;
        while (ventana.pollEvent(eventoFinal)) {
            if (eventoFinal.type == Event::MouseButtonPressed) {
                if (botonReintentar.getGlobalBounds().contains(ventana.mapPixelToCoords(Mouse::getPosition(ventana)))) {
                    iniciarJuego(juegoIniciado, juegoTerminado, serpiente, comidas, comida, dir, puntuacion, logros);
                }
                if (botonFinalizar.getGlobalBounds().contains(ventana.mapPixelToCoords(Mouse::getPosition(ventana)))) {
                    ventana.close();
                }
            }
        }
        if (juegoIniciado) break;
    }
    continue;
}
serpiente.push_back(siguiente);
```

Figura 21. Verificación para colisiones

Después de gestionar la comida, se realiza una verificación para determinar si la cabeza de la serpiente colisiona con su propio cuerpo o con el borde de la ventana. Si se detecta una colisión, se ejecutan las siguientes acciones:

- Reproducción de un sonido (**sonidoColision**) indicando que la serpiente ha colisionado.
- Cambio de los estados **juegoIniciado** y **juegoTerminado** para detener la ejecución del bucle principal y mostrar la pantalla de fin de juego.
- Creación de un texto (**textoFinal**) que muestra la puntuación final y los logros obtenidos.
- Limpieza de la ventana y dibujo del texto final junto con los botones de reintentar y finalizar.
- Uso de un bucle de eventos para esperar la interacción del usuario.

```
        ventana.clear();
        pixel.setFillColor(Color::Green);
        for (Punto p : serpiente) {
            pixel.setPosition(p.x * 20, p.y * 20);
            ventana.draw(pixel);
        }
        pixel.setFillColor(Color::Red);
        for (Punto p : comidas) {
            pixel.setPosition(p.x * 20, p.y * 20);
            ventana.draw(pixel);
        }

        // Actualiza y dibuja la puntuación
        textoPuntuacion.setString("Puntuacion: " + to_string(puntuacion) + "\nLogros: " + to_string(logros));
        ventana.draw(textoPuntuacion);

        // Dibuja el texto de la velocidad
        textoVelocidad.setString(
                "\n\n+50 puntos: Velocidad aumentada\n+100 puntos: Logro desbloqueado\n+150 puntos: Otra bola mas de comida");
        ventana.draw(textoVelocidad);

        ventana.display();

        sleep(milliseconds(velocidad));
    }

    return 0;
```

Figura 22. Actualización y dibujo de puntuación

Después de gestionar las colisiones y el fin del juego, se procede a actualizar la representación visual del juego en la ventana. En este proceso:

- Se añade la nueva posición de la cabeza de la serpiente al vector **serpiente**.
- Se limpia la ventana.
- Se dibujan la serpiente y la comida en sus posiciones actuales utilizando un bucle **for** y la función **draw** de la ventana.
- Se actualizan y dibujan los textos de puntuación y velocidad en la pantalla.
- Se refresca la ventana para mostrar los cambios.
- Se introduce un breve retardo en la ejecución (**sleep(milliseconds(velocidad))**) para controlar la velocidad del juego.

Conclusiones:

Desde una perspectiva general, la arquitectura modular y la utilización de funciones específicas contribuyen significativamente a la legibilidad y mantenimiento del código. El uso de estas estructuras da una base sólida para el desarrollo futuro; y una implementación efectiva del manejo de eventos demuestra un enfoque pragmático para las interacciones del usuario.

Asimismo, el diseño intuitivo facilita la comprensión y mejora la usabilidad.

La inclusión de efectos de sonido y música de fondo en el programa añade un componente inmersivo al juego; la correcta gestión de archivos de sonido, respaldada por mensajes de error, termina reflejando un sólido control de calidad y demuestra un manejo adecuado.

Finalmente, hablando por la interfaz de usuario, donde se termina presentando una disposición clara y accesible, con botones estratégicamente posicionados y textos informativos con una pantalla de inicio que proporciona detalles esenciales sobre el juego y sus desarrolladores, así como de la implementación de un sistema de velocidad progresiva en función de la puntuación aporta un elemento dinámico y desafiante al juego, enriqueciendo la experiencia del usuario.

Recomendaciones:

Recomendamos explorar oportunidades para optimizar secciones críticas del código, con el objetivo de mejorar la eficiencia y el rendimiento del juego.

- Ofrecer opciones de personalización, como la capacidad de ajustar la velocidad inicial o seleccionar niveles de dificultad desde el menú principal, para mejorar la flexibilidad del juego.
- Conducir pruebas exhaustivas en diversas situaciones para identificar y abordar posibles problemas, asegurando así una experiencia de usuario libre de errores

El código desarrollado en este informe demuestra un nivel competente de implementación y constituye una base robusta para un juego de la serpiente en C++. La implementación de estas y más recomendaciones dadas por otros usuarios, enriquecerá aún más la calidad y la experiencia de futuros usuarios, alineándose con estándares profesionales de desarrollo de software y cumpliendo la meta de un juego divertido y entretenido.

ENLACE A GITHUB:

https://github.com/micaelCZ/EstructuraDeDatos_GrupoD

ENLACE A YOUTUBE:

https://www.youtube.com/watch?v=QEtWT_6wNVc

Bibliografía

GeeksforGeeks. (2023, 1 de noviembre). SFML Graphics Library. Recuperado de https://www.geeksforgeeks.org/sfml-graphics-library-quick-tutorial/

Seleim, Y. (2023). Data_Structures-in-SFML: Popular Data Structures implemented in C++ and visualized with SFML. Recuperado de GitHub.

GeeksforGeeks. (2023). Snake Code in C++. Recuperado de GeeksforGeeks.

University of Adelaide. (2023). Introduction to Data Structures: Snake Game. Recuperado de University of Adelaide.

GitHub. (2023). SnakeGame: A classic Snake game implemented using Data Structures and Algorithm (DSA). Recuperado de GitHub.

GitHub. (2023). Snake-game-using-data-structures: A data structures project, we use doubly linked list to make a snake game using C++. Recuperado de GitHub.