

Reporte Proyecto Final

"Lost Souls"

01/06/2024



Integrantes:

Argüello León Dante Moisés
Cruz Cedillo Daniel Alejandro
Yoaddan Yokaem Muro León

Índice

Introducción.....	2
Objetivo del juego (Gameplay).....	2
Diseño.....	2
Plan de trabajo.....	6
Licenciamiento de texturas, modelos y sonido.....	6
Precio estimado.....	11
Desarrollo.....	12
Resultados y trabajo a futuro.....	20
Conclusiones.....	20
Repositorio de Git:.....	21



Introducción.

OpenGL es una herramienta muy potente que nos permite crear muchas cosas, en esta ocasión lo utilizaremos para realizar un juego, el cual buscará utilizar lo visto a lo largo del semestre como lo es las colisiones, animaciones, sonido espacial, entre otras cosas.

Objetivo del juego (Gameplay).

El juego se ambienta en un laberinto, el cual contará con 2 tipos de enemigos, el primer tipo de enemigo es un fantasma el cual irá persiguiendo al personaje principal evitando que el usuario tenga tiempo de quedarse quieto y tenga que tomar decisiones rápidas, el segundo tipo de enemigo son los guardias, dichos guardias contarán con una ruta predefinida, el propósito del jugador es recolectar los cofres que se encuentran distribuidos en el laberinto. El juego busca ser complejo por lo que el jugador solo dispondrá de 1 vida, y con poca iluminación, el caballero que será el personaje usado por el jugador contará con una pequeña hada en su espalda que le dará un poco de luz. El escenario cuenta con antorchas esparcidas para ayudar con la visibilidad limitada.

Diseño.

Para el diseño del juego buscamos referencias relacionadas a Dark souls, ya que se buscó un estilo medieval, y para la configuración de la cámara buscamos algo inspirado en la utilizada en juegos como Diablo.



1.1 Imagen referencia de ambientación.

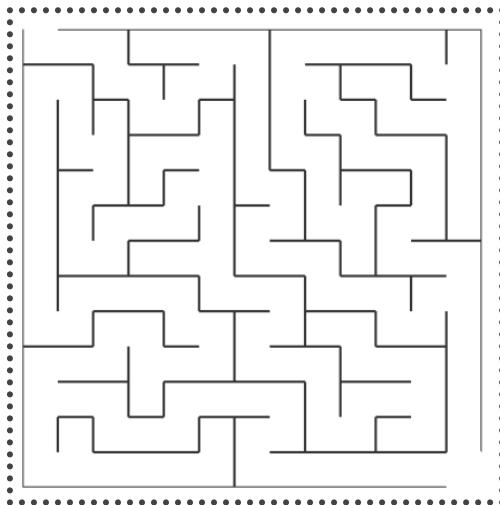


1.2 Imagen referencia de cámara.



Para el diseño de controles, no se realizó algo complejo, ya que el juego se pensó como un puzzle en el que solo tienes que moverte y esquivar a los enemigos, no se buscó pelear o algo similar.

Para el diseño del laberinto nos basamos en un laberinto generado de forma aleatoria, ya que el laberinto no se tomó de ningún lado fue necesario crearlo de forma independiente en blender.



1.3 Imagen referencia laberinto.



1.4 Laberinto en blender.

Se cuenta con un modelo de personaje principal acorde a la temática:





1.5 Personaje principal.
Puede moverse a lo largo del escenario en busca de los tesoros.

Cada tipo de enemigo contará con sus características:



1.6 Guardia.
Contarán con un circuito predefinido y en caso de entrar en contacto con el jugador el usuario morirá y el juego se reiniciará.



1.7 Fantasma.
Podrá atravesar paredes e irá siguiendo al jugador a lo largo del laberinto.



Se elaboraron un total de 3 pantallas, la primera es para cuando se inicia el juego, y nos dará la opción de start y exit.



1.8 Pantalla de Inicio del Juego.

Para las otras 2 pantallas que saldrán dependerá de si el usuario pierde o logra conseguir los 3 tesoros.



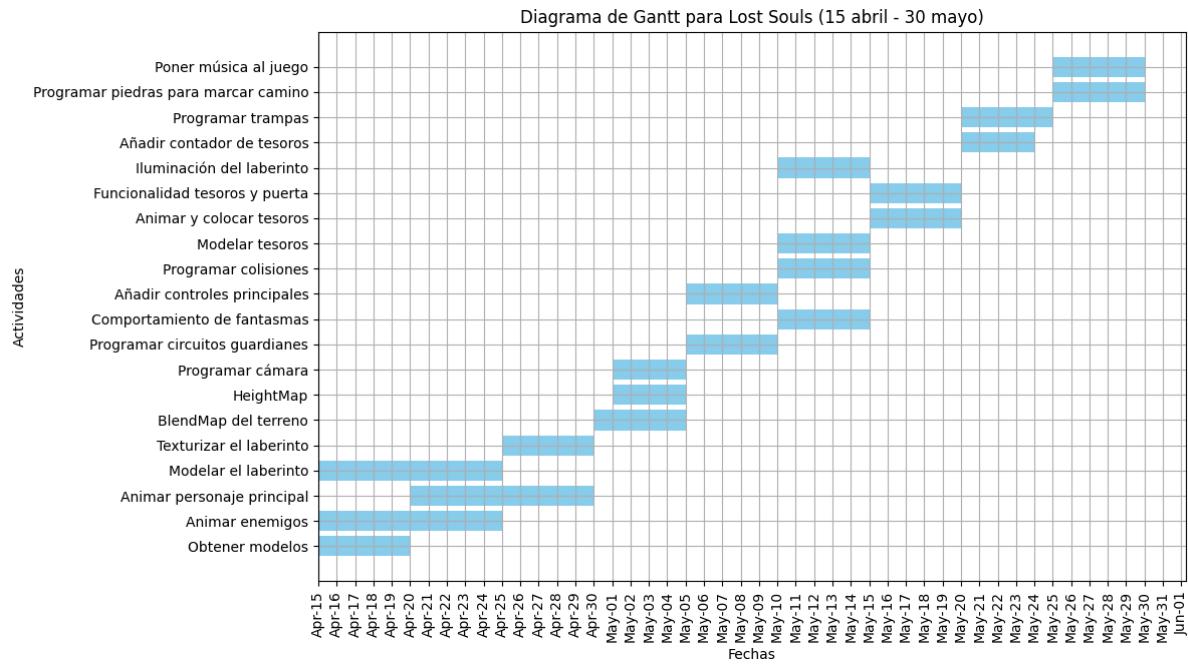
1.9 Pantalla al perder partida.



1.10 Pantalla al Ganar.

Plan de trabajo.

El Diagrama de Gantt a continuación muestra los tiempos estimados de trabajo que buscamos dedicar a cada parte del proyecto, consideramos dar tiempos holgados para cada parte por posibles imprevistos o complicaciones o modificaciones que llegasen a surgir.



2.1 Diagrama de Gantt para desarrollo de Lost Souls.

Tratamos de seguir el plan de trabajo inicial, pero conforme se fue desarrollando el juego se modificaron un poco las mecánicas que se planteaban inicialmente:

- Trampas: Se planeaban usar para facilitar al personaje al deambular por el laberinto, no se agregaron por que los enemigos no son muy rápidos y el jugador puede esquivarlos sin problemas
- Se retiro la idea de abrir una puerta a juntar los tesoros porque afectaba mucho a los fps
- Se quitó la mecánica de poner piedras para marcar el camino, en cambio se le agregó un hada al personaje principal que le ayudará a iluminar el camino, como el laberinto no era muy grande optamos por cambiar la mecánica para dar complejidad.

Licenciamiento de texturas, modelos y sonido.

Todos los modelos utilizados son de licencia gratuita o de autoría propia.





3.1 Modelo de Guardian.

Models124717. The Models Resource.

https://www.models-resource.com/pc_computer/darksoulsii/model/63523/



3.2 Modelo de Fantasma.

GianaSistersFan64. The Models Resource.

https://www.models-resource.com/pc_computer/gianasisterstwisteddreams/model/45697/



3.3 Modelo de antorcha.

Models124717. The Models Resource.

https://www.models-resource.com/pc_computer/darksouls/model/62530/





3.4 Modelo de hada en botella.

Centrix the Dodo. The Models Resource.

https://www.models-resource.com/nintendo_switch/supersmashbrosultimate/model/32617/



3.5 Modelo del personaje principal.

Ozci. The Models Resource.

<https://www.models-resource.com/wii/zangekinoreginkleiv/model/26209/>

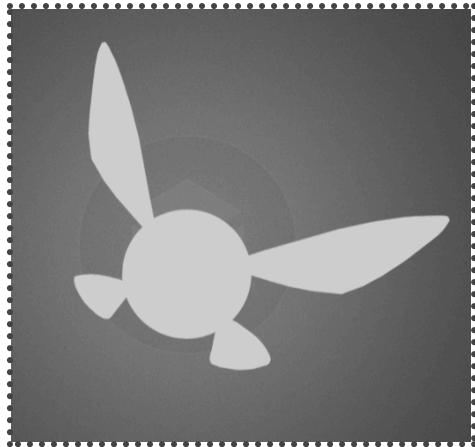


3.6 Modelo de fuego de las antorchas.

zer0nim. Sketchfab.

<https://sketchfab.com/3d-models/bomberman-fire-8e482145eed419980fabf073fc13c9>





3.7 Modelo del hada.
darkewne. Sketchfab.

<https://sketchfab.com/3d-models/navi-fairy-of-link-zelda-724f7dbdbc8440edb7cfddb1abfd0a>

71



3.8 Modelo del laberinto.
De nuestra autoría.

En cuanto a texturas solo se utilizó el siguiente paquete para texturizar el laberinto y el terreno. Además del skybox.

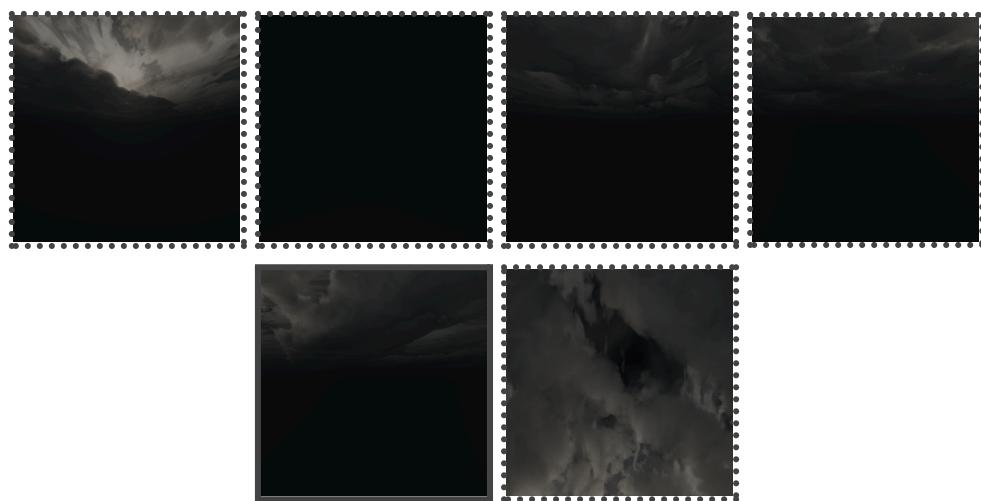


Random Talking Bush. The Models Resource.

https://www.textures-resource.com/xbox_360/sonicthehedgehog2006/texture/17704/

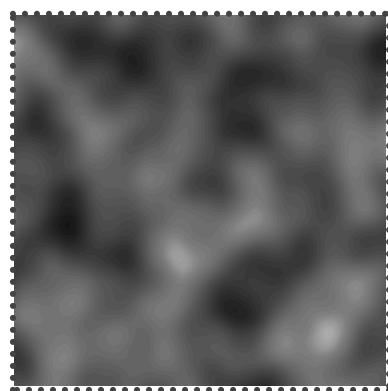


3.9 Textura del laberinto basada en la original.
De nuestra autoría.



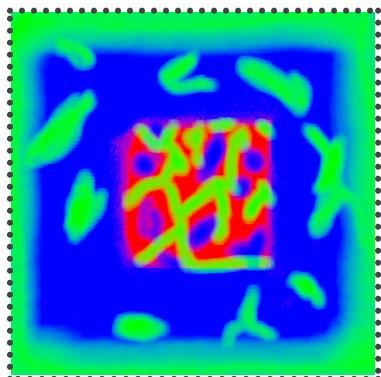
3.10 Skybox.
avianx. CgTrader.

<https://www.cgtrader.com/free-3d-models/textures/natural-textures/hdri-freebie-series>



3.11 Mapa de alturas.

De nuestra autoría.



3.12 Mapa de mezclas.

De nuestra autoría.



3.13 Imágenes de interfaz.

De nuestra autoría (incluyendo las de cofres).

De igual manera todos los sonidos utilizados son de uso gratuito.

Sonido de fantasma. Pixabay. <https://pixabay.com/es/sound-effects/ghost-6979/>

Sonido de guardia. Pixabay. <https://pixabay.com/es/sound-effects/metal-footsteps-14727/>

Sonido de fuego. Pixabay. <https://pixabay.com/es/sound-effects/burninospfire-6936/>

Sonido de fondo. Lachm. Pixabay.

<https://pixabay.com/es/sound-effects/sound-ambience-sonido-ambiente-23-14211/>

Precio estimado

Costos fijos de desarrollo		
Concepto	Mes / Unitario	Proyecto (7 semanas)



Internet.	\$550.00	\$962.50
Equipo.	\$15,000.00	\$15,000.00
Software de diseño 2D.	\$0.00	\$0.00
Software de diseño 3D.	\$0.00	\$0.00
Imágenes.	\$0.00	\$0.00
Modelos 3D.	\$0.00	\$0.00
Sonidos.	\$0.00	\$0.00
Costos variables de desarrollo		
Concepto	Mes	Proyecto (7 semanas)
Electricidad.	\$300.00	\$525.00
Sueldo de desarrollador.	\$18,000.00	\$31,500.00
Sueldo de diseñador 3D.	\$25,000.00	\$833.33 X 1 dia
Costos pasivos		
Concepto	Valor	Proyecto (7 semanas)
Mantenimiento.	\$3,000.00	\$3,000.00
Actualización del software	\$5,000.00	\$5,000.00
Costo total		
		\$56,820.83

Por lo tanto su costo de venta sería de \$79,549.162. Con un beneficio del 40%.



Animaciones.

En esta práctica, se utilizaron animaciones generadas con Mixamo para dar vida a los personajes del juego. Para el personaje principal, se implementaron dos animaciones clave: una para el estado "Idle" y otra para el estado "Walking". Estos estados fueron manejados utilizando índices específicos que permitieron alternar fluidamente entre las animaciones según las acciones del jugador. Las animaciones en formato FBX fueron cargadas y renderizadas en OpenGL, lo que facilitó la integración y el manejo eficiente de las secuencias de movimiento.

El guardia, por su parte, utilizó una única animación de caminata, también generada en Mixamo y almacenada en formato FBX. Esta animación proporcionó el movimiento necesario para patrullar el entorno del juego, añadiendo un nivel adicional de dinamismo y realismo. La elección de Mixamo para generar las animaciones permitió un proceso de desarrollo más ágil y la obtención de resultados de alta calidad. La compatibilidad con OpenGL aseguró que las animaciones se integraran sin problemas en el motor gráfico del juego, mejorando la experiencia visual y la inmersión del jugador.

Desarrollo.

Para el desarrollo del proyecto se realizó la importación de los modelos que ambientan el juego, y se le dio la inicialización necesaria.

```
// Modelos externos
Model laberinto; //Laberinto principal.
Model fantasma; //Enemigo que atraviesa paredes.
Model guardial; //Enemigo con ruta predefinida.
Model guardia2;
Model guardia3;
Model modelAntorcha; //Antorchas.
Model tesoro;
Model fuego; //Fuego animado.
Model hada; //Hada.
```

4.1 Modelos para ambientar y decorar el escenario.

Se dieron los posicionamientos de los elementos fijos como las antorchas, cofres y demás modelos que estarán estáticos en el escenario y solo tendrán la función de decorar o iluminar.

```
// Posicion antorchas
std::vector<glm::vec3> torchesPosition = {
    glm::vec3(-1.79279, 0, 2.06897),
    glm::vec3(-2.1736, 0, 36.6826),
    glm::vec3(36.1438, 0, -1.00821),
    glm::vec3(9.27195, 0, -21.005),
    glm::vec3(-31.1512, 0, -12.7198),
    glm::vec3(-37.0462, 0, 16.0555)
};
```

4.2 Posicionamiento de las antorchas.



Se establecieron las matrices para los modelos que tendrán movimiento como es el caso del fantasma, el personaje principal, los guardias, entre otros.

```
// Model matrix definitions
glm::mat4 modelMatrixLaberinto = glm::mat4(1.0f);
glm::mat4 modelMatrixFantasma = glm::mat4(1.0f);
glm::mat4 modelMatrixGuardia1 = glm::mat4(1.0f);
glm::mat4 modelMatrixGuardia2 = glm::mat4(1.0f);
glm::mat4 modelMatrixGuardia3 = glm::mat4(1.0f);
glm::mat4 modelMatrixHada = glm::mat4(1.0f);
```

4.3 Matrices para los modelos.

Para el movimiento del personaje principal utilizamos el código para animar objetos con huesos, el cual permite contener las animaciones dentro del modelo y setearlas con ayuda de un índice, haciendo su uso el más accesible y eficiente.

```
/*
 * Objetos animados por huesos
 */
glm::vec3 ejey = glm::normalize(terrain.getNormalTerrain(modelMatrixMainCharacter[3][0], modelMatrixMainCharacter[3][2]));
glm::vec3 ejex = glm::vec3(modelMatrixMainCharacter[0]);
glm::vec3 ejez = glm::normalize(glm::cross(ejex, ejey));
ejex = glm::normalize(glm::cross(ejey, ejez));
modelMatrixMainCharacter[0] = glm::vec4(ejex, 0.0);
modelMatrixMainCharacter[1] = glm::vec4(ejey, 0.0);
modelMatrixMainCharacter[2] = glm::vec4(ejez, 0.0);
modelMatrixMainCharacter[3][1] = terrain.getHeightTerrain(modelMatrixMainCharacter[3][0], modelMatrixMainCharacter[3][2]);
glm::mat4 modelMatrixMainCharacterBody = glm::mat4(modelMatrixMainCharacter);
//modelMatrixMainCharacterBody = glm::scale(modelMatrixMainCharacterBody, glm::vec3(0.021f)); //Se debe comentar despues
modelMainCharacter.setAnimationIndex(animationMainCharacterIndex);
modelMainCharacter.render(modelMatrixMainCharacterBody);

modelMatrixHada = modelMatrixMainCharacter;
modelMatrixHada = glm::translate(modelMatrixHada, glm::vec3(0.174625f, 1.552f, -0.5f));
hada.render(modelMatrixHada);
```

4.4 Movimiento del personaje principal.

En la parte del fantasma se le agregaron las instrucciones necesarias para que siguiera al personaje principal a lo largo del mapa, se le dio una velocidad baja para que le diera oportunidad al jugador de maniobrar y no ser atrapado tan fácilmente por él.

```
// Interpolación lineal para mover el fantasma hacia la posición de Mayow
float interpolationFactor = 0.001f; // Ajusta este valor para cambiar la velocidad de seguimiento
glm::vec3 newPositionFantasma = glm::mix(glm::vec3(modelMatrixFantasma[3]), glm::vec3(modelMatrixMainCharacter[3]), interpolationFactor);

// Calcular la dirección hacia Mayow y ajustar la orientación del fantasma
glm::vec3 directionFantasma = glm::normalize(glm::vec3(modelMatrixMainCharacter[3]) - newPositionFantasma);
glm::vec3 upFantasma = glm::vec3(0.0f, 1.0f, 0.0f);
glm::vec3 rightFantasma = glm::normalize(glm::cross(upFantasma, directionFantasma));
upFantasma = glm::cross(directionFantasma, rightFantasma);
```

4.5 Instrucciones para seguimiento del personaje principal.

Tanto al personaje principal como algunos enemigos se le dio un punto de partida inicial con el propósito de indicarle al jugador desde donde puede iniciar y los enemigos para que puedan realizar sus animaciones en el lugar que les corresponde.



```

modelMatrixMainCharacter = glm::translate(modelMatrixMainCharacter, glm::vec3(34.677f, 0.05f, 37.0987f));
modelMatrixMainCharacter = glm::rotate(modelMatrixMainCharacter, glm::radians(-90.0f), glm::vec3(0, 1, 0));

//Guardias
modelMatrixGuardia1 = glm::translate(modelMatrixGuardia1, glm::vec3(29.0f, 0.05f, -5.0f));
modelMatrixGuardia1 = glm::rotate(modelMatrixGuardia1, glm::radians(180.0f), glm::vec3(0, 1, 0));
modelMatrixGuardia1 = glm::scale(modelMatrixGuardia1, glm::vec3(1.45));

```

4.6 Coordenadas y orientación inicial de los modelos.

En la parte de la iluminación se dio muy poca, ya que el propósito es que casi no puedas ver a tus alrededores y se vuelva difícil descifrar donde se encuentra el enemigo o si el fantasma está cerca.

```

/***********************
 * Propiedades Luz direccional
***********************/
shaderMullighting.setVectorFloat3("viewPos", glm::value_ptr(camera->getPosition()));
shaderMullighting.setVectorFloat3("directionalLight.light.ambient", glm::value_ptr(glm::vec3(0.0, 0.0, 0.0)));
shaderMullighting.setVectorFloat3("directionallight.light.diffuse", glm::value_ptr(glm::vec3(0.1, 0.1, 0.1)));
shaderMullighting.setVectorFloat3("directionallight.light.specular", glm::value_ptr(glm::vec3(0.0, 0.0, 0.0)));
shaderMullighting.setVectorFloat3("directionalLight.direction", glm::value_ptr(glm::vec3(-0.7071, -0.7071, -0.7071)));

shaderTerrain.setVectorFloat3("viewPos", glm::value_ptr(camera->getPosition()));
shaderTerrain.setVectorFloat3("directionallight.light.ambient", glm::value_ptr(glm::vec3(0.0, 0.0, 0.0)));
shaderTerrain.setVectorFloat3("directionallight.light.diffuse", glm::value_ptr(glm::vec3(0.1, 0.1, 0.1)));
shaderTerrain.setVectorFloat3("directionallight.light.specular", glm::value_ptr(glm::vec3(0.0, 0.0, 0.0)));
shaderTerrain.setVectorFloat3("directionallight.direction", glm::value_ptr(glm::vec3(-0.7071, -0.7071, -0.7071)));

/***********************

```

4.7 Luz del escenario.

Pero para no hacer imposible el juego y ayudar al jugador se agregaron elementos como el hada, y las antorchas que darán iluminación al jugador, estos le permiten ver el mapa pero no verlo todo.

```

// Pointlight del hada
glm::mat4 matrixAdjustFairy = glm::mat4(1.0);
matrixAdjustFairy = glm::translate(matrixAdjustFairy, glm::vec3(modelMatrixHada[3]));
glm::vec3 FairyPosition = glm::vec3(matrixAdjustFairy[3]);
shaderMullighting.setVectorFloat3("pointLights[" + std::to_string(torchesPosition.size()) + "].light.ambient", glm::value_ptr(glm::vec3(0.1, 0.1, 0.1)));
shaderMullighting.setVectorFloat3("pointLights[" + std::to_string(torchesPosition.size()) + "].light.diffuse", glm::value_ptr(glm::vec3(0.1, 0.1, 0.1)));
shaderMullighting.setVectorFloat3("pointLights[" + std::to_string(torchesPosition.size()) + "].light.specular", glm::value_ptr(glm::vec3(0.1, 0.1, 0.1)));
shaderMullighting.setVectorFloat3("pointLights[" + std::to_string(torchesPosition.size()) + "].position", glm::value_ptr(FairyPosition));
shaderMullighting.setFloat("pointLights[" + std::to_string(torchesPosition.size()) + "].constant", 1.0);
shaderMullighting.setFloat("pointLights[" + std::to_string(torchesPosition.size()) + "].linear", 0.09);
shaderMullighting.setFloat("pointLights[" + std::to_string(torchesPosition.size()) + "].quadratic", 0.02);
shaderTerrain.setVectorFloat3("pointlights[" + std::to_string(torchesPosition.size()) + "].light.ambient", glm::value_ptr(glm::vec3(0.1, 0.1, 0.1)));

```

4.8 Luz de los modelos.

Se agregaron Colliders a todos los modelos como las antorchas y paredes con el propósito de no poder atravesar los muros y solo poder seguir los caminos, y se agregaron colliders a los enemigos, así como al personaje principal para que si entran en contacto poder indicar que el jugador perdió. Así como también se agregaron colliders para poder recoger los tesoros del laberinto.



```

// Antorchas colliders
for (int i = 0; i < torchesPosition.size(); i++){
    AbstractModel::OBB lampCollider;
    glm::mat4 modelMatrixColliderLamp = glm::mat4(1.0);
    modelMatrixColliderLamp = glm::translate(modelMatrixColliderLamp, torchesPosition[i]);
    modelMatrixColliderLamp = glm::rotate(modelMatrixColliderLamp, glm::radians(torchesOrientation[i]),
                                         glm::vec3(0, 1, 0));
    addOrUpdateColliders(collidersOBB, "Antorcha-" + std::to_string(i), lampCollider, modelMatrixColliderLamp);
    // Set the orientation of collider before doing the scale
    lampCollider.u = glm::quat_cast(modelMatrixColliderLamp);
}

```

4.9 Colliders de los Modelos.

Para los enemigos que tienen rutas fijas se les establece una máquina de estados para cada uno, ya que no siguen rutas similares.

```

//Guardia 2
switch (stateG2)
{
    case 0:
        if(posG2 == 0 || posG2 == 4 || posG2 == 7 || posG2 == 11){
            maxAdvanceG2 = 5.0;
            stateG2 = 1;
        }
        //Derecha
        if(posG2 == 1 || posG2 == 3 || posG2 == 12 || posG2 == 13){
            stateG2 = 3;
        }
        if(posG2 == 2 || posG2 == 9){
            maxAdvanceG2 = 12.0;
            stateG2 = 1;
        }
        //Izquierda
        if(posG2 == 5 || posG2 == 6 || posG2 == 8 || posG2 == 10){
            stateG2 = 2;
        }
        if(posG2 == 14){
            posG2 = 0;
        }
        break;
    //Avanzar
    case 1:
        modelMatrixGuardia2 = glm::translate(modelMatrixGuardia2, glm::vec3(0.0f, 0.0f, avanceG2));
        contAdvanceG2 += avanceG2;
        if (contAdvanceG2 > maxAdvanceG2){
            contAdvanceG2 = 0.0;
            stateG2 = 0;
            posG2++;
        }
        break;
    //Girar izquierda
    case 2:
}

```

4.10 Máquina de estado de Guardian

Para los sonidos espaciales se usó OpenAL y se inicializan los vectores de posiciones y velocidad. Seguido de los buffer y los sonidos que se reproducen al iniciar automáticamente el juego.

```

alutGenBuffers(NUM_BUFFERS, buffer);
//Buffer del fantasma
buffer[0] = alutCreateBufferFromFile("../sounds/fantasma.wav");
//Buffer del fuego
buffer[1] = alutCreateBufferFromFile("../sounds/fuego.wav");
//Buffer de fondo (Es estereo para que no le afecte el sonido espacial)
buffer[2] = alutCreateBufferFromFile("../sounds/background.wav");
//Buffer de guardias
buffer[3] = alutCreateBufferFromFile("../sounds/guardia.wav");

```

4.11 Buffers cargando sonidos.



Luego se configuró cada source de sonido con sus respectivas configuraciones de audio como la distancia máxima a la que se escucha, su pitch y su ganancia.

```
// Modelo de OpenAL - IMPORTANTE
alDistanceModel(AL_LINEAR_DISTANCE_CLAMPED);

// Parametros sonido del fantasma
alSourcef(source[0], AL_PITCH, 1.5f);
alSourcef(source[0], AL_GAIN, 0.8f);
alSourcefv(source[0], AL_POSITION, source0Pos);
alSourcefv(source[0], AL_VELOCITY, source0Vel);
alSourcei(source[0], AL_BUFFER, buffer[0]);
alSourcei(source[0], AL_LOOPING, AL_TRUE);
alSourcef(source[0], AL_MAX_DISTANCE, 12.0f);
```

4.12 Configuración de las fuentes de audio.

Finalmente se asignaron las posiciones a las fuentes de audio y se asignó la posición y orientación del personaje principal al listener de OpenAL.

```
source10Pos[0] = modelMatrixGuardia3[3].x;
source10Pos[1] = modelMatrixGuardia3[3].y;
source10Pos[2] = modelMatrixGuardia3[3].z;
alSourcefv(source[10], AL_POSITION, source10Pos);

// Listener for the Thris person camera
listenerPos[0] = modelMatrixMainCharacter[3].x;
listenerPos[1] = modelMatrixMainCharacter[3].y;
listenerPos[2] = modelMatrixMainCharacter[3].z;
alListenerfv(AL_POSITION, listenerPos);

glm::vec3 upModel = glm::normalize(modelMatrixMainCharacter[1]);
glm::vec3 frontModel = glm::normalize(modelMatrixMainCharacter[2]);

listenerOri[0] = frontModel.x;
listenerOri[1] = frontModel.y;
listenerOri[2] = frontModel.z;
listenerOri[3] = upModel.x;
listenerOri[4] = upModel.y;
listenerOri[5] = upModel.z;
```

4.13 Posiciones de fuentes y listener de OpenAL.

Para el mando se configuró la función processInput donde leemos el joystick, sus ejes y sus botones, y los almacenamos por paso por referencia. En la imagen también se aprecia el cambio de interfaz por medio de las texture ID's.



```

int buttonCount;

if (exitApp || glfwWindowShouldClose(window) != 0) {
    return false;
}

if (!iniciaPartida) {
    // Comprobación de presionar Enter (teclado) o botón X (mando)
    presionarEnter = glfwGetKey(window, GLFW_KEY_ENTER) == GLFW_PRESS || (glfwJoystickPresent(GLFW_JOYSTICK_1) && glfwGetJoystickButtons(GLFW_JOYSTICK_1, &buttonCount)[1] == GLFW_PRESS);
    if (textureActivaID == textureInitID && presionarEnter) {
        iniciaPartida = true;
        textureActivaID = textureScreenID;
    } else if (textureActivaID == textureInit2ID && presionarEnter) {
        exitApp = true;
    } else {
        // Comprobación de presionar CTRL izquierdo (teclado) o botón Cuadrado (mando)
    }
}

```

4.14 Muestra de lectura de los botones del mando.

En el caso de los ejes de los joysticks se leen sus valores y se modifica la cámara en base a ellos directamente, se limitó el eje y de la cámara a no ser cercano a 0 ni menor a él para evitar que el jugador haga trampa observando el escenario por debajo.

```

if (glfwJoystickPresent(GLFW_JOYSTICK_1) == GL_TRUE) {
    //std::cout << "Esta presente el joystick" << std::endl;
    int axesCount;
    const float * axes = glfwGetJoystickAxes(GLFW_JOYSTICK_1, &axesCount);
    //DEBUG ENTRADAS
    /*std::cout << "Número de ejes disponibles :=>" << axesCount << std::endl;
    std::cout << "Left Stick X axis: " << axes[0] << std::endl;
    std::cout << "Left Stick Y axis: " << axes[1] << std::endl;
    std::cout << "Left Trigger/L2: " << axes[3] << std::endl;
    std::cout << "Right Stick X axis: " << axes[2] << std::endl;
    std::cout << "Right Stick Y axis: " << axes[5] << std::endl;
    std::cout << "Right Trigger/R2: " << axes[4] << std::endl;*/

    if(fabs(axes[1]) > 0.2){
        modelMatrixMainCharacter = glm::translate(modelMatrixMainCharacter, glm::vec3(0, 0, -axes[1] * 0.1));
        animationMainCharacterIndex = 0;
    }if(fabs(axes[0]) > 0.2{
        modelMatrixMainCharacter = glm::rotate(modelMatrixMainCharacter, glm::radians(-axes[0] * 1.5f), glm::vec3(0, 1, 0));
        animationMainCharacterIndex = 0;
    }

    if(fabs(axes[2]) > 0.2){
        camera->mouseMoveCamera(axes[2], 0.0, deltaTime);
    }if(fabs(axes[5]) > 0.2{
        if (axes[5] >= 0) {
            // Permitir el movimiento hacia arriba sin restricciones
            camera->mouseMoveCamera(0.0, axes[5], deltaTime);
        } else {
            // Mueve la cámara hacia abajo solo si la altura actual es mayor que el límite mínimo
            if (camera->getPosition().y > 2.0) {
                camera->mouseMoveCamera(0.0, axes[5], deltaTime);
            }
        }
    }
}

```

4.15 Muestra de lectura de joystick y limitaciones de cámara.

Para los tesoros se tiene un contador que funciona como selector en un switch, en base a este valor la interfaz va cambiando al número de tesoros que hemos recogido hasta el momento. En caso de completarlos todos se activa una bandera que marca el fin de la partida y muestra la pantalla de victoria.



```

        } else {
            if (finPartida) {
                if(treasuresCollected == 3){
                    textureActivaID = textureWinID;
                } else {
                    textureActivaID = textureGameOverID;
                }
                for(unsigned int i = 0; i < sourcesPlay.size(); i++){
                    sourcesPlay[i] = false;
                    alSourceStop(source[i]);
                    // Limpia el buffer asociado
                    alSourcei(source[i], AL_BUFFER, 0);
                    // Verifica si hay algún error de OpenAL
                    ALenum error = alGetError();
                    if (error != AL_NO_ERROR)
                        std::cerr << "Error al detener la fuente " << i << ":" << error << std::endl;
                }
            } // Comprobación de presionar Enter (teclado) o botón Start (mando)
            presionarStart = glfwGetKey(window, GLFW_KEY_ENTER) == GLFW_PRESS || (glfwJoystickPresent(GLFW_JOYSTICK_1) && glfwGetJoystickButtons(GLFW_JOYSTICK_1, &buttonCount)[9] == GLFW_PRESS);
            if (presionarStart) {
                exitApp = true;
            }
        } else {
            switch (treasuresCollected) {
                case 0:
                    textureActivaID = textureCounter0ID;
                    break;
                case 1:
                    textureActivaID = textureCounter1ID;
                    break;
                case 2:
                    textureActivaID = textureCounter2ID;
                    break;
                case 3:
                    textureActivaID = textureCounter3ID;
                    finPartida = true;
                    break;
                default:
                    std::cout << "Número de tesoros fuera de rango." << std::endl;
                    break;
            }
        }
    }
}

```

4.16 Muestra del cambio de interfaz al recoger tesoros, ganar y morir.

Para hacer el cambio de texturas se tiene una textura activa que va variando de ID, justo en esta parte se tiene el render de texto también, donde le indicamos posición, escala y color con canal alfa además de nuestro mensaje deseado. Para esto fue necesario editar los archivos `FontTypeRendering.h` y `FontTypeRendering.cpp` para que recibiera estos parámetros adicionales. Además tiene condición para que desaparezca en caso de victoria o derrota y que no aparezca durante el menú inicial.

```

/************* Render de imagen de frente *****/
shaderTexture.setMatrix4("projection", 1, false, glm::value_ptr(glm::mat4(1.0)));
shaderTexture.setMatrix4("view", 1, false, glm::value_ptr(glm::mat4(1.0)));
glActiveTexture(GL_TEXTURE0);
 glBindTexture(GL_TEXTURE_2D, textureActivaID);
 shaderTexture.setInt("outTexture", 0);
boxIntro.render();
float colorText[4] = {1.0f, 0.0f, 0.0f, 0.5f}; // Rojo
if(iniciaPartida && !finPartida)
    modelText->render("Recolecta los tesoros", -0.9, -0.8, 2.5, colorText);
//glfwSwapBuffers(window);

```

4.17 Muestra del render de texto.

También hay una neblina completamente negra. La cual se renderiza con el shader de la luz, el terreno y el skybox para que se aprecie en todos ellos.



```

/*
 * Propiedades de neblina
 */
shaderMulLighting.setVectorFloat3("fogColor", glm::value_ptr(glm::vec3(0.0, 0.0, 0.0)));
shaderTerrain.setVectorFloat3("fogColor", glm::value_ptr(glm::vec3(0.0, 0.0, 0.0)));
shaderSkybox.setVectorFloat3("fogColor", glm::value_ptr(glm::vec3(0.0, 0.0, 0.0)));

/*

```

4.18 Configuración del color de la neblina en los shaders.

Resultados y trabajo a futuro.

Los resultados fueron buenos, resultó en un juego complicado pero no imposible, el cual tiene la posibilidad de darle más niveles en el futuro en los cuales sean laberintos más complejos, salgan mas enemigos, e incluso se vayan agregando mecánicas que se descartaron conforme se progrese en los niveles para dar la sensación de progreso que es muy parecida en los videojuegos. También es un concepto que permite muchas formas de generar enemigos interesantes, como cofres trampa por ejemplo.

En general a futuro se podrían añadir mecánicas descartadas como las piedras y el escape tras recoger los tesoros.

Conclusiones.

- Argüello León Dante Moisés:

Desarrollar un videojuego puede suponer un reto, pero es una aplicación en donde se puede demostrar bien el conocimiento adquirido a lo largo del curso de Computación Gráfica Avanzada. Para poder desarrollarlo, se debe tener presente cada uno de los pasos y funciones que son ejecutadas a lo largo del Game Loop, y me ha quedado más claro cómo es que en un ciclo del Game Loop debemos garantizar que las funciones de transformación principalmente, y las funciones que complementan a la ambientación del entorno virtual influyen en el tiempo que el equipo de cómputo se tardará en dibujar los frames, pues se debe hacer todo de manera óptima para poder obtener al menos 30 FPS. Si bien logramos implementar con éxito muchas de las funcionalidades deseadas, el proceso tuvo desafíos en la gestión de recursos, la compatibilidad de formatos y la correcta eliminación de colisionadores, pues fueron áreas que requirieron especial atención y esfuerzo.

- Cruz Cedillo Daniel Alejandro:

La práctica presentó bastante retos que se fueron corrigiendo, hubo momentos en que la pantalla no dejaba de parpadear y se corrigió quitando unas líneas, se enderezó la pantalla de inicio, por que estaba invertida, fue bastante tardado diseñar las máquinas de estado para algunas rutas. Pero a final de cuentas se logró hacer funcionar, también el juego presenta la opción a mejorarse en el futuro ya que se puede agregar sistema de niveles en



un futuro, en el cual conforme más dificultad haya más enemigos y puede agregarse mecánicas adicionales al personaje. La parte más difícil fue crear y posicionar todo en el escenario así como calcular que se desplacen los modelos de forma correcta en el escenario.

- Yoaddan Yokaem Muro León:

A lo largo del desarrollo del proyecto se presentaron varias complicaciones, para empezar no se tenía el código de referencia de últimas prácticas con interfaz funcional, lo que llevó a revisar porque la interfaz no funcionaba logrando corregirla pero con un bug de parpadeo que se solucionó al comentar la línea de swapBuffers al renderizar la imagen de la interfaz. Por otro lado también se tuvo problemas con la implementación de audio debido al descuido de usar audio estéreo en lugar de mono (esto sirvió después para al momento de añadir música al nivel solo se dejó en estéreo y ya) y a la falta de una línea de configuración que define el modelo que usa OpenAL para hacer la atenuación, esto se corrigió añadiendo la línea correspondiente. También se tuvieron percances con las sombras al inicio pero se solucionó rápidamente agregando las líneas que asignaban la matriz de proyección y de vista en el shader de la luz y del terreno a la matrixLightSpace. Además destacó el uso de herramientas vistas en clase con otros fines como la interpolación, para otros fines propios por ejemplo, se usó la función mix para hacer una interpolación entre las coordenadas del personaje principal y el fantasma, ocasionando que el fantasma persigue siempre al jugador.

En general el proyecto funcionó para reforzar los temas del curso tales como las sombras, el textRendering, el mapa de alturas, el mapa de mezclas, las interfaces, el sonido espacial con OpenAL, las colisiones, las entradas de mando, las animaciones, la neblina y la iluminación.

Repositorio de Git:

<https://github.com/Yoaddan/LostSoulsCGA>

En caso de que el programa se deseé ejecutar en Linux, deberá cambiar el contenido de la carpeta “sounds” por los archivos siguientes. Debido a una incompatibilidad de la codificación del formato .wav, en esta carpeta el formato es .wav de 16 bits signed PCM.

https://drive.google.com/drive/folders/1hratZUrwN1gyZ_dyMInhlJEwvbsx_s0D?usp=sharing

Para consultar los videos se debe acceder al siguiente enlace.

<https://drive.google.com/drive/folders/1pJccb9C6wFfSShVxU5Xju6xhN85Px-QK?usp=drivelink>

