UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Proyecto Final – Recreación de Pacman

Computación Gráfica Avanzada

Integrantes:

- Colón Palacios Emmanuel
- Guerrero Prado Issac Alexander

Semestre 2024-2

CALIFICACIÓN: _____

1. Introducción

Con la finalidad de representar lo aprendido a lo largo del curso de *Computación Gráfica Avanzada* se requería de realizar un proyecto final que introdujera un *gameplay* dentro de un entorno virtual con la gran mayoría de técnicas que pudimos ver durante las sesiones de este semestre, como animaciones, mapeo y texturización de un terreno hecho con la técnica de Blendmap, lógica de un personaje como su movimiento dentro del entorno, iluminación, colisiones, etc.

Buscando una idea de proyecto que satisfaga dichos requerimientos anteriores, se optó por recrear el célebre videojuego PacMan, realizando la lógica de dicho juego desde cero mediante la herramienta de OpenGL.

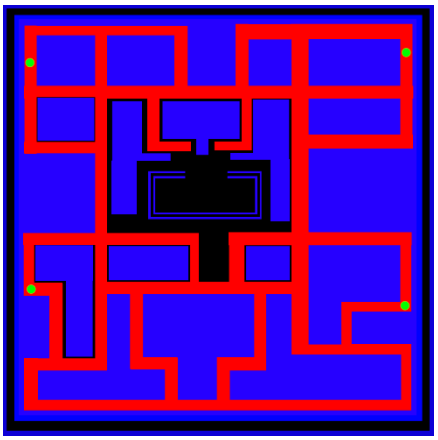
2. Objetivo del juego

Recrear el diseño del célebre videojuego Pacman utilizando las técnicas vistas en el curso de Computación Gráfica Avanzada, de tal manera que obtengamos un gameplay interesante para el usuario final y que a este le represente un reto.

3. Diseño

El proyecto constará de un solo escenario, con la lógica de movimiento de los fantasmas fija y con un comportamiento determinado de los mismos al agarrar un *power-up* nuestro personaje principal. Por la misma razón, nuestros mapas de alturas y blend serán más cuadrados de lo habitual.

- BlendMap



- HeightMap

Para el usuario final la interfaz de este proyecto debe ser algo intuitivo, por lo cual también contaremos con una pantalla de inicio y una pantalla de Game Over para cuando el usuario gane o pierda.

- Inicio

- GameOver

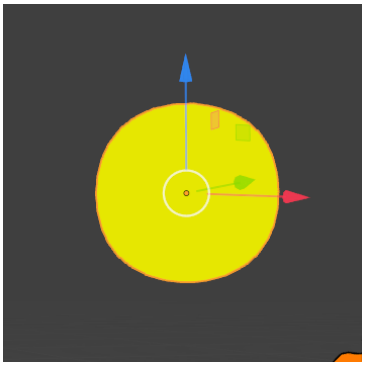
También, como ya se mencionó, contamos con los personajes clásicos de pacman, fantasmas y Pacman, además de tener un modelo de una esfera, que tiene dos

tipos de lógicas, la primera únicamente suma puntos al *score* dentro del juego y la segunda es para sumar puntos y además agregar el Power-Up a nuestro personaje principal.

- Pacman

- Fantasmas

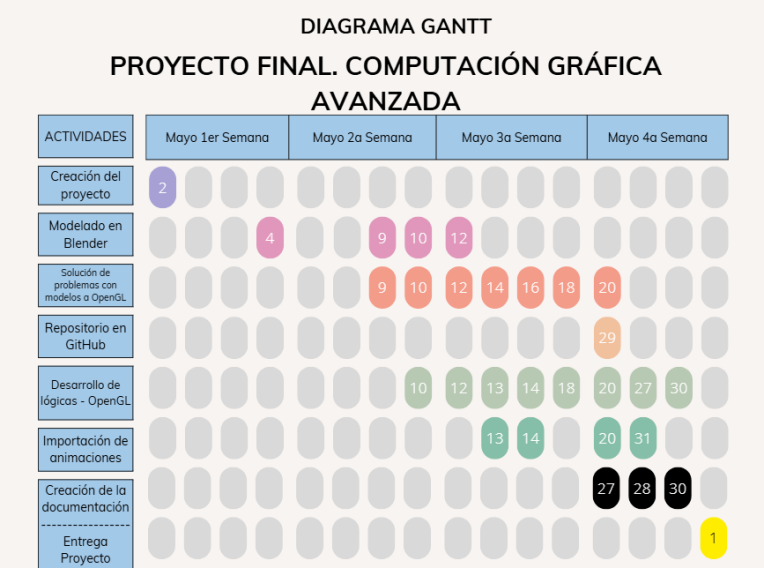
- Esfera de puntos y Power-Up



4. Plan de trabajo

El plan principal para sacar a flote este proyecto consistió de trabajo paralelo y con tiempo, más o menos se explica viendo el diagrama de Gantt resultante del desarrollo de este proyecto.

a. Diagrama de Gantt



5. Licenciamiento de modelos, sonido, etc.

Modelos de Pacman y Fantasmas:

- Origen: Los modelos 3D de Pacman y los fantasmas fueron creados específicamente para este proyecto utilizando software de modelado 3D.
- Licencia: Los modelos de fantasmas fueron realizados con mérito propio, sin embargo el modelo principal fue rescatado de SketchFab, de uso libre y disponible para descargar, mientras que las animaciones de IDLE y de WALK fueron sacadas de Mixamo, colección de animaciones de uso libre.
- URL de la licencia: https://www.mixamo.com/ y https://tinyurl.com/3cvht7uy

Modelos de Laberinto y Puntos:

- Origen: Los modelos del laberinto y los puntos recolectables también fueron creados específicamente para este proyecto.

- Licencia: Fueron realizados con mérito propio.

Efectos de Sonido del Juego:

- Origen: Los efectos de sonido fueron obtenidos de una biblioteca de sonidos con licencia libre, compatible con proyectos de código abierto.
- Licencia: Los efectos de sonido se distribuyen bajo la Licencia Creative Commons Attribution 3.0 Unported (CC BY 3.0). Esto permite el uso, distribución y modificación de los sonidos siempre que se atribuya adecuadamente a los autores originales.
- URL de la licencia: https://creativecommons.org/licenses/by/3.0/

Música de Fondo:

- Origen: La música de fondo utilizada en el juego fue creada específicamente para este proyecto por un compositor independiente.
- Licencia: La música de fondo se distribuye bajo una licencia personalizada que permite su uso exclusivo en este proyecto, prohibiendo su redistribución o modificación sin el consentimiento explícito del compositor.

6. Precio estimado

Características del Proyecto

- Un Solo Escenario: Laberinto clásico de Pacman con texturización y mapeo de alturas.
- Lógica de Personajes: Movimiento de fantasmas y Pacman, incluyendo comportamiento con power-ups.
- Interfaz de Usuario: Pantalla de inicio y pantalla de Game Over.
- Modelos Clásicos: Pacman, fantasmas y esferas de puntos/power-ups.

Público Objetivo

- Jugadores Casuales: Usuarios que buscan una versión gratuita y entretenida del clásico Pacman.
- Desarrolladores Independientes: Que pueden estudiar y modificar el proyecto como base para sus propios juegos.

Modelo de Negocio

- Free-to-Play con Anuncios

Una opción viable para monetizar el proyecto sin infringir derechos de propiedad intelectual es lanzarlo como un juego free-to-play (gratuito) con anuncios integrados. Este modelo permite a los usuarios jugar sin costo, generando ingresos a través de publicidad in-game.

Ventajas:

- Accesibilidad para un amplio público.
- Generación de ingresos pasivos.
- No requiere que los usuarios paguen por el juego, aumentando la base de usuarios.

Venta del Proyecto (Con Licencias Educativas y Personales)

Si se desea ofrecer el proyecto para su estudio o modificación, se puede establecer un precio simbólico, asegurándose de no infringir derechos de propiedad intelectual.

- Precio para Estudiantes/Desarrolladores: Entre $50 y $200 MXN, considerando que es un recurso básico e independiente.
- Precio para Educadores/Instituciones: Entre $200 y $350 MXN, como material de referencia.

7. Desarrollo

- Inicialización de modelos

En esta sección se hará caso especial a la inicialización de los muros de nuestro mapa de Pacman y se mencionará de manera superficial los métodos que lleva consigo el introducir los modelos de esferas, fantasmas y Pacman.

```
346   std::vector<glm::vec3> wallPosition = {
347       glm::vec3(-30.5, 0, -21),
348       glm::vec3(-28.5, 0, -21),
349       glm::vec3(-26.5, 0, -21),
350       glm::vec3(-24.5, 0, -21),
351       glm::vec3(-22.5, 0, -21),
352       glm::vec3(-20.5, 0, -21),
353       glm::vec3(-18.5, 0, -21),
354       glm::vec3(-16.5, 0, -21),
355       glm::vec3(-31.1, 0, -20),
356       glm::vec3(-31.1, 0, -18),
357       glm::vec3(-31.1, 0, -16),
358       glm::vec3(-31.1, 0, -14),
359       glm::vec3(-31.1, 0, -12),
360       glm::vec3(-31.1, 0, -10),
361       glm::vec3(-31.1, 0, -8),
362       glm::vec3(-31.1, 0, -6),
363       glm::vec3(-31.1, 0, -4),
364       glm::vec3(-31.1, 0, -2.5),
365       glm::vec3(-30.3, 0, -1.9),
366       glm::vec3(-30.3, 0, -1.9),
367       glm::vec3(-28.3, 0, -1.9),
368       glm::vec3(-26.3, 0, -1.9),
369       glm::vec3(-24.3, 0, -1.9),
370       glm::vec3(-22.3, 0, -1.9),
371       glm::vec3(-20.3, 0, -1.9),
372       glm::vec3(-18.3, 0, -1.9),
373       glm::vec3(-16.3, 0, -1.9),
```

Al igual que en las lecciones para inicializar modelos de lámparas fijas mediante un mapa de texturas en nuestro terreno, se realiza el mismo procedimiento para los muros de nuestro mapa de Pacman, dando datos relevantes sobre la posición y la orientación de los mismos.

```
std::vector<float> wallOrientation = {
    //spawn fantasmas
    0 , 0, 0, 0, 0, 0, 0, 0,
    90, 90, 90, 90, 90, 90, 90, 90, 90, 90,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    90,
    0,0,0,0,0,0,0,
    //centro izquierda
    90,90,90,90,
    90,90, 90,
    0,0, 0

};
```

Igual es importante rescatar la adición de sus respectivos coliders, para que el personaje principal (usuario) no pueda traspasar el mapa.

```
boxWalls.init();
boxWalls.setShader(&shaderMulLighting);
```

Se inicializa el modelo del muro como cualquier modelo dentro de nuestro entorno, cargándolo y seteándole un shader, para luego ser texturizado.

```
modelWall.loadModel("../models/wall/fall.fbx");
modelWall.setShader(&shaderMulLighting);
modelLargeWall.loadModel("../models/box/box.fbx");
modelLargeWall.setShader(&shaderMulLighting);
```

Por último, al finalizar la ejecución del programa se procede a destruir en memoria el modelo.

```
modelGhostPink.destroy
modelWall.destroy();
```

Para el resto de modelos se omite el caso de la introducción de posición y orientación dentro del entorno virtual, simplemente se setean los modelos y se

procede a colocar los mismos métodos para introducirlos y destruirlos al momento de terminar la ejecución.

```
//Pacman
mayowModelAnimate.setShader(&shaderMulLighting);
```

En el caso del personaje principal, al estar modelado y animado por huesos, se procede a setear la lógica del mismo.

```
/*******************************************
 * Objetos animados por huesos
 * *****************************************/
glm::vec3 ejey = glm::normalize(terrain.getNormalTerrain(modelMatrixMayow[3][0], modelMatrixMayow[3][2]))
glm::vec3 ejex = glm::vec3(modelMatrixMayow[0]);
glm::vec3 ejez = glm::normalize(glm::cross(ejex, ejey));
ejex = glm::normalize(glm::cross(ejey, ejez));
modelMatrixMayow[0] = glm::vec4(ejex, 0.0);
modelMatrixMayow[1] = glm::vec4(ejey, 0.0);
modelMatrixMayow[2] = glm::vec4(ejez, 0.0);
modelMatrixMayow[3][1] = -GRAVITY * tmv * tmv + 3.5 * tmv + terrain.getHeightTerrain(modelMatrixMayow[3]
tmv = currTime - startTimeJump;
if(modelMatrixMayow[3][1] < terrain.getHeightTerrain(modelMatrixMayow[3][0], modelMatrixMayow[3][2])){
    isJump = false;
    modelMatrixMayow[3][1] = terrain.getHeightTerrain(modelMatrixMayow[3][0], modelMatrixMayow[3][2]);
}
glm::mat4 modelMatrixMayowBody = glm::mat4(modelMatrixMayow);
modelMatrixMayowBody = glm::scale(modelMatrixMayowBody, glm::vec3(0.021f));
mayowModelAnimate.setAnimationIndex(animationMayowIndex);
mayowModelAnimate.render(modelMatrixMayowBody);
```

Se setean sus respectivos COLLIDERS.

```
// Collider de mayow
AbstractModel::OBB mayowCollider;
glm::mat4 modelmatrixColliderMayow = glm::mat4(modelMatrixMayow);
modelmatrixColliderMayow = glm::rotate(modelmatrixColliderMayow,
        glm::radians(-90.0f), glm::vec3(1, 0, 0));
// Set the orientation of
mayowCollider.u = glm::qu
modelmatrixColliderMayow
modelmatrixColliderMayow
        glm::vec3(mayowModelAnimate.getObb().c.x,
                mayowModelAnimate.getObb().c.y,
                mayowModelAnimate.getObb().c.z));
mayowCollider.e = mayowModelAnimate.getObb().e * glm::vec3(0.021, 0.021, 0.021) * glm::vec3(0.787401
mayowCollider.c = glm::vec3(modelmatrixColliderMayow[3]);
addOrUpdateColliders(collidersOBB, "mayow", mayowCollider, modelMatrixMayow);
```

```
constexpr glm::highp_vec3::vec(float a, float b, float c)

+19 overloads

Explicit conversions (From section 5.4.1 Conversion and scalar constructors of GLSL 1.30.08 sp
```

```
if (itCollision->first.compare("mayow") == 0)
    modelMatrixMayow = std::get<1>(obbBuscado->second);
if (itCollision->first.compare("dart") == 0)
    modelMatrixDart = std::get<1>(obbBuscado->second);
```

Se procede a destruir los modelos anteriormente llamados en caso del cierre de la aplicación para ahorrar memoria.

```
mayowModelAnimate.destroy(); // Pacman
cowboyModelAnimate.destroy();
guardianModelAnimate.destroy();
cyborgModelAnimate.destroy();
modelFountain.destroy();
modelGhostRed.destroy();
modelGhostBlue.destroy();
modelGhostOrange.destroy();
modelGhostPink.destroy();
modelWall.destroy();
```

- Lógica de audio

Al igual que lo vimos en la práctica de OpenAL, se procede a generar buffers donde estarán almacenados los archivos con extensión *.wav*. Posteriormente, se setea la lógica para escucharlo respectivamente desde la cámara de tercera persona y más información requerida del espacio virtual para su correcto funcionamiento.

```
// Generate buffers, or else no sound will happen!
alGenBuffers(NUM_BUFFERS, buffer);
buffer[0] = alutCreateBufferFromFile("../sounds/pacman_beginning.wav");
buffer[1] = alutCreateBufferFromFile("../sounds/pacman_chomp.wav");
buffer[2] = alutCreateBufferFromFile("../sounds/pacman_death.wav");
buffer[3] = alutCreateBufferFromFile("../sounds/pacman_eatghost.wav");
int errorAlut = alutGetError();
if (errorAlut != ALUT_ERROR_NO_ERROR){
    printf("- Error open files with alut texto en openGL %d !!\n", errorAlut);
    exit(2);
}
```

```
/*****************************+
 * Open AL sound data
 */

source1Pos[0] = modelMatrixMayow[3].x;
source1Pos[1] = modelMatrixMayow[3].y;
source1Pos[2] = modelMatrixMayow[3].z;
alSourcefv(source[1], AL_POSITION, source1Pos);

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

glm::vec3 upModel = glm::normalize(modelMatrixMayow[1]);
glm::vec3 frontModel = glm::normalize(modelMatrixMayow[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;
```

- Lógica de Fantasmas

Cada fantasma cuenta con una lógica de trayectoria dentro de un Switch-Case, este es el ejemplo del fantasma rojo.

```
switch (state){
case 0:
    if(numberAdvance == 0){
        maxAdvance = 13.6;
        direction = -1;
    }
    else if(numberAdvance == 1){
        maxAdvance = 8.6;
        direction = -1;
    }
    else if(numberAdvance == 2){
        maxAdvance = 19;
        direction = -1;
    }
    else if(numberAdvance == 3){
        maxAdvance = 23.5;
        direction = 1;
    }
```

```
case 1:
    modelMatrixGhostRed = glm::translate(modelMatrixGhostRed, glm::vec3(0.0f, 0.45f, 0.0f));
    //modelMatrixGhostRed = glm::rotate(modelMatrixGhostRed, glm::radians(-90.0f), glm::vec3(0, 1,
    advanceCount += 0.45;
    if(advanceCount > maxAdvance){
        advanceCount = 0;
        numberAdvance++;
        state = 2;
    }
    break;
case 2:
    modelMatrixGhostRed = glm::rotate(modelMatrixGhostRed, glm::radians(giroFantasma), glm::vec3(0,
    rotCount += giroFantasma;
    if(rotCount >= 90.0f){
        rotCount = 0;
        state = 0;
        if(numberAdvance > 20)
            numberAdvance = 5;
    }
    break;

default:
    break;
}
```

Donde cuando llegue a cierta posición, rotará de una u otra forma continuando así su trayectoria, si el jugador se cruza con alguno de los fantasmas y se detecta alguna colisión este puede perder una vida o ganar puntos, dependiendo de si el jugador principal tiene activo el Power-Up dado por una de las esferas más grandes dentro del escenario.

- Lógica de Pacman

Pacman debe contener un collider para así detectar tanto colisiones con las paredes del mapa, los fantasmas y principalmente las esferas de puntos para poder así sumar al score final, además de que así se puede detectar si gana o no el juego.

```
// Collider de pacman
AbstractModel::OBB mayowCollider;
glm::mat4 modelmatrixColliderMayow = glm::mat4(modelMatrixMayow);
modelmatrixColliderMayow = glm::rotate(modelmatrixColliderMayow,
        glm::radians(-90.0f), glm::vec3(1, 0, 0));
// Set the orientation of collider before doing the scale
mayowCollider.u = glm::quat_cast(modelmatrixColliderMayow);
//modelmatrixColliderMayow = glm::scale(modelmatrixColliderMayow, glm::vec3(0.021, 0.021, 0.021));
modelmatrixColliderMayow = glm::scale(modelmatrixColliderMayow, glm::vec3(0.25, 0.4, 0.5));
modelmatrixColliderMayow = glm::translate(modelmatrixColliderMayow,
        glm::vec3(mayowModelAnimate.getObb().c.x,
                mayowModelAnimate.getObb().c.y,
                mayowModelAnimate.getObb().c.z));
mayowCollider.e = mayowModelAnimate.getObb().e * glm::vec3(0.25, 0.4, 0.5) * glm::vec3(0.787401574,
mayowCollider.c = glm::vec3(modelmatrixColliderMayow[3]);
addOrUpdateColliders(collidersOBB, "mayow", mayowCollider, modelMatrixMayow);
```

El modelo de igual manera cuenta con respuesta al terreno desde el mapa de alturas.

```
glm::vec3 ejey = glm::normalize(terrain.getNormalTerrain(modelMatrixMayow[3][0], modelMatrixMayow[3][2]
glm::vec3 ejex = glm::vec3(modelMatrixMayow[0]);
glm::vec3 ejez = glm::normalize(glm::cross(ejex, ejey));
ejex = glm::normalize(glm::cross(ejey, ejez));
modelMatrixMayow[0] = glm::vec4(ejex, 0.0);
modelMatrixMayow[1] = glm::vec4(ejey, 0.0);
modelMatrixMayow[2] = glm::vec4(ejez, 0.0);
modelMatrixMayow[3][1] = -GRAVITY * tmv * tmv + 3.5 * tmv + terrain.getHeightTerrain(modelMatrixMayow[3]
tmv = currTime - startTimeJump;
if(modelMatrixMayow[3][1] < terrain.getHeightTerrain(modelMatrixMayow[3][0], modelMatrixMayow[3][2])){
    isJump = false;
    modelMatrixMayow[3][1] = terrain.getHeightTerrain(modelMatrixMayow[3][0], modelMatrixMayow[3][2]);
}
glm::mat4 modelMatrixMayowBody = glm::mat4(modelMatrixMayow);
modelMatrixMayowBody = glm::scale(modelMatrixMayowBody, glm::vec3(0.005f));
mayowModelAnimate.setAnimationIndex(animationMayowIndex);
mayowModelAnimate.render(modelMatrixMayowBody);
```

Si Pacman come a alguno de los fantasmas con el Power-Up activado, entonces Pacman obtendrá 300 puntos extras y mandará al fantasma al punto de inicio, además de que finalizará con su Power-Up.

```
else{
    if(jt->first == "mayow" && powerUp && it->first == "blinky")
        {
            reset_blinky = true;
            score += 300;
            alSourcePlay(source[3]);
            powerUp = false;
        }
    if(jt->first == "mayow" && powerUp && it->first == "pinky")
        {
            reset_pinky = true;
            score += 300;
            alSourcePlay(source[3]);
            powerUp = false;
        }
    if(jt->first == "mayow" && powerUp && it->first == "inky")
        {
            reset_inky = true;
            score += 300;
            alSourcePlay(source[3]);
            powerUp = false;
        }
```

El modelo del personaje principal cuenta con dos animaciones almacenadas en un modelo FBX, una animación de reposo y otra animación de movimiento.

```
// Mayow
mayowModelAnimate.loadModel("../models/Pacman/HappyWalk3.fbx");
mayowModelAnimate.setShader(&shaderMulLighting);
```

Dichas animaciones funcionan tanto a la respuesta de un mando como a la respuesta del teclado.

```
if(fabs(axes[1]) > 0.2){
    modelMatrixMayow = glm::translate(modelMatrixMayow, glm::vec3(0, 0, -axes[1] * 0.1));
    animationMayowIndex = 2;
}if(fabs(axes[0]) > 0.2){
    modelMatrixMayow = glm::rotate(modelMatrixMayow, glm::radians(-axes[0] * 0.5f), glm::vec3(0, 1,
    animationMayowIndex = 2;
}
```

```
// Controles de Pacman
if(vidas > 0)
{
    if (modelSelected == 0 && glfwGetKey(window, GLFW_KEY_LEFT) == GLFW_PRESS){
        modelMatrixMayow = glm::rotate(modelMatrixMayow, 0.06f, glm::vec3(0, 1, 0));
        animationMayowIndex = 2;
    } else if (modelSelected == 0 && glfwGetKey(window, GLFW_KEY_RIGHT) == GLFW_PRESS){
        modelMatrixMayow = glm::rotate(modelMatrixMayow, -0.06f, glm::vec3(0, 1, 0));
        animationMayowIndex = 2;
    }
    if (modelSelected == 0 && glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS){
        modelMatrixMayow = glm::translate(modelMatrixMayow, glm::vec3(0.0, 0.0, 0.6));
        animationMayowIndex = 2;
    }
    else if (modelSelected == 0 && glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS){
        modelMatrixMayow = glm::translate(modelMatrixMayow, glm::vec3(0.0, 0.0, -0.6));
        animationMayowIndex = 2;
    }
```

De igual manera existe una lógica para detectar si el usuario terminó con todas las esferas que suman puntos en el mapa, al igual que si el personaje principal Pacman pierde una o todas sus vidas.

```
if(spheres_to_win == 0 && !gameFinished)
{
    modelMatrixMayow = glm::mat4(1.0);
    modelMatrixMayow = glm::translate(modelMatrixMayow, glm::vec3(0.0f, 0.05f, 5.0f));
    modelMatrixMayow = glm::rotate(modelMatrixMayow, glm::radians(-90.0f), glm::vec3(0, 1, 0));
    textureActivaID = textureVictoryID;
    alSourcePause(source[1]);
    alSourcePlay(source[4]);
    gameFinished = true;
}

if(reset_player){
    modelMatrixMayow = glm::mat4(1.0);
    modelMatrixMayow = glm::translate(modelMatrixMayow, glm::vec3(0.0f, 0.05f, 5.0f));
    modelMatrixMayow = glm::rotate(modelMatrixMayow, glm::radians(-90.0f), glm::vec3(0, 1, 0));
    vidas--;
    reset_player = false;
    switch(vidas){
        ALint source0State;
        case 3: textureActivaID = textureScreenID;
                alSourcePause(source[1]);
                alSourcePlay(source[2]);
                do {
                    alGetSourcei(source[2], AL_SOURCE_STATE, &source0State);
                do {
                    alGetSourcei(source[2], AL_SOURCE_STATE, &source0State);
                } while (source0State == AL_PLAYING);
                // Adjuntar el segundo buffer al source[0]
                alSourceQueueBuffers(source[0], 1, &buffer[1]);
                alSourcePlay(source[1]);
                break;
        case 1: textureActivaID = textureScreenID3;
                alSourcePause(source[1]);
                alSourcePlay(source[2]);
                do {
                    alGetSourcei(source[2], AL_SOURCE_STATE, &source0State);
                } while (source0State == AL_PLAYING);
                // Adjuntar el segundo buffer al source[0]
                alSourceQueueBuffers(source[0], 1, &buffer[1]);
                alSourcePlay(source[1]);
                break;
        case 0:
                textureActivaID = textureGameOverID;
                alSourcePause(source[1]);
                break;
        }
    std::cout << "menos una vida :(" << std::endl;
```

- Lógica de esferas

Debido a que el mapa no tiene diagonales, se ajustan y clasifican las esferas de puntuación en eje X y e eje Y.

```
std::vector<glm::vec3> spherePosition = {
    //Esquina1 cuadrante 1
    glm::vec3(-11.2, 0, 9.6),
    //Esferas en X
    glm::vec3(-14.2, 0, 9.6),
    glm::vec3(-19.2, 0, 9.6),
    glm::vec3(-24.2, 0, 9.6),
    glm::vec3(-29.2, 0, 9.6),
    glm::vec3(-34.2, 0, 9.6),
    glm::vec3(-39.2, 0, 9.6),
    glm::vec3(-44.2, 0, 9.6),
    glm::vec3(-49.2, 0, 9.6),
    glm::vec3(-54.2, 0, 9.6),
    glm::vec3(-59.2, 0, 9.6),
    glm::vec3(-64.2, 0, 9.6),
    glm::vec3(-69.2, 0, 9.6),
    glm::vec3(-74.2, 0, 9.6),
    glm::vec3(-79.2, 0, 9.6),
    glm::vec3(-84.2, 0, 9.6),
    //Esferas en Y
    glm::vec3(-11.2, 0, 16.6),
    glm::vec3(-11.2, 0, 21.6),
    glm::vec3(-11.2, 0, 26.6),
    glm::vec3(-11.2, 0, 31.6),
```

La lógica de estas está basada en qué tantas y qué tipo de esfera agarra el personaje principal, de igual manera cuentan con un collider para detectar la colisión de Pacman sobre las esferas y así sumar puntos o darle un power-up al usuario.

```
if(spheres_to_win == 0 && !gameFinished)
{
    modelMatrixMayow = glm::mat4(1.0);
    modelMatrixMayow = glm::translate(modelMatrixMayow, glm::vec3(0.0f, 0.05f, 5.0f));
    modelMatrixMayow = glm::rotate(modelMatrixMayow, glm::radians(-90.0f), glm::vec3(0, 1, 0));
    textureActivaID = textureVictoryID;
    alSourcePause(source[1]);
    alSourcePlay(source[4]);
    gameFinished = true;
}
```

Básicamente detecta que Pacman ya haya tomado todas las esferas, de tal manera que activa el overlay de victoria y endgame al usuario.

```
for(int i = 0; i < spherePosition.size(); i++){
    if(spheresCollisions[i] == false)
    {
        spherePosition[i].y = terrain.getHeightTerrain(spherePosition[i].x, spherePosition[i].z) + 1.0f;
        modelSphere.setPosition(spherePosition[i]);
        modelSphere.setScale(glm::vec3(0.75));
        //modelSphere.setOrientation(glm::vec3(0, lamp1Orientation[i], 0));
        modelSphere.render();
    }
}
```

```
for(int i = 0; i < powerUpPosition.size(); i++){
    if(powerUpCollisions[i] == false)
    {
        powerUpPosition[i].y = terrain.getHeightTerrain(powerUpPosition[i].x, powerUpPosition[i].z) + 1.
        modelSphere.setPosition(powerUpPosition[i]);
        modelSphere.setScale(glm::vec3(1.25));
        //modelSphere.setOrientation(glm::vec3(0, lamp1Orientation[i], 0));
        modelSphere.render();
    }
}
```

Estos dos últimos fragmentos de código son los dos tipos de esferas que detecta, power-up o normal.

```
//sphere colliders

for (int i = 0; i < spherePosition.size(); i++){
    if(spheresCollisions[i] == false)
    {
    AbstractModel::SBB sphereCollider;
    glm::mat4 modelMatrixColliderSphere = glm::mat4(1.0);
    modelMatrixColliderSphere = glm::translate(modelMatrixColliderSphere, spherePosition[i]);
    addOrUpdateColliders(collidersSBB, "sphere-" + std::to_string(i), sphereCollider, modelMatrixCol
    // Set the orientation of collider before doing the scale
    modelMatrixColliderSphere = glm::scale(modelMatrixColliderSphere, glm::vec3(0.5, 0.5, 0.5));
    modelMatrixColliderSphere = glm::translate(modelMatrixColliderSphere, modelSphere.getSbb().c);
    sphereCollider.c = glm::vec3(modelMatrixColliderSphere[3]);
    sphereCollider.ratio = modelSphere.getSbb().ratio * 0.5;
    std::get<0>(collidersSBB.find("sphere-" + std::to_string(i))->second) = sphereCollider;
    }
}
```

Esta es la lógica para setearle colliders a las esferas normales, así mismo es para las esferas Power-Up.

- Lógica de puntuación

La lógica de puntuación radica en un texto dentro del entorno OpenGL que suma puntos para el score final del usuario, puede ser mayor o igual a la cantidad de puntos que te otorgan todas las esferas del mapa.

```
// Se inicializa el model de render text
modelText = new FontTypeRendering::FontTypeRendering(screenWidth, screenHeight);
modelText->Initialize();
```

8. Resultados y trabajo a futuro

A pesar de los logros alcanzados, existen varias áreas en las que se puede seguir trabajando para mejorar y expandir el proyecto:

Optimización del Desempeño:

- Mejorar la eficiencia del código y la utilización de recursos para asegurar un rendimiento óptimo en una variedad de dispositivos.

Expansión de Funcionalidades:

- Nuevos Niveles y Escenarios: Crear nuevos laberintos y escenarios para diversificar el gameplay y aumentar el desafío para los jugadores.
- Modos de Juego Adicionales: Implementar diferentes modos de juego, como un modo contrarreloj o un modo de supervivencia.

Mejoras en la Interfaz de Usuario:

- Desarrollar un menú de opciones más completo que permita a los usuarios personalizar la experiencia de juego (ajustes de sonido, gráficos, controles, etc.).

Integración con Redes Sociales:

- Añadir funcionalidades que permitan a los jugadores compartir sus puntuaciones y logros en redes sociales, lo que puede ayudar a aumentar la visibilidad y popularidad del juego.

Expansión a Otras Plataformas:

- Adaptar el juego para que sea compatible con diferentes sistemas operativos y dispositivos móviles, para alcanzar a un público más amplio.

Feedback y Actualizaciones:

- Recoger y analizar el feedback de los jugadores para identificar áreas de mejora y lanzar actualizaciones regulares que mantengan el interés y la satisfacción de los usuarios.

9. Conclusiones

- Colón Palacios Emmanuel

Este proyecto no solo valida la eficacia de las técnicas enseñadas en el curso, sino que también proporciona una base robusta para futuros desarrollos en la creación de videojuegos con OpenGL y otras plataformas de desarrollo de entornos virtuales.

El proyecto de recreación del videojuego Pacman ha demostrado ser una implementación exitosa de las técnicas avanzadas de computación gráfica aprendidas en el curso. La utilización de animaciones, mapeo y texturización de terrenos, iluminación y detección de colisiones ha resultado en un gameplay sólido y atractivo. Los jugadores pueden experimentar un entorno visualmente detallado y una lógica de juego bien definida que presenta un desafío adecuado con un nivel de escalamiento muy aceptable.

- Guerrero Prado Issac Alexander

La recreación de Pacman utilizando OpenGL ha servido como una excelente plataforma para aplicar y validar los conceptos de computación gráfica avanzada adquiridos durante el curso. La implementación efectiva de técnicas como blend mapping, animaciones, colisiones y desarrollo de una lógica de programación propia ha resultado en un juego que no solo es entretenido, sino que también es técnicamente competente.

Además, el éxito alcanzado con esta recreación sugiere un gran potencial para la expansión del proyecto, incluyendo la incorporación de nuevas funcionalidades y la optimización del rendimiento, abriendo posibilidades para futuros desarrollos y mejoras.

10. Referencias

JadeandPals. (8 de April de 2024). *sketchfab/JadeandPals*. Obtenido de sketchfab/JadeandPals: https://sketchfab.com/3d-models/pacman-4534b38bac154e31964a369a7807016b?sscid=51k8_116s4j&utm_source=shareasale&utm_medium=affiliate&utm_campaign=745788_1272560

Martell Ávila, R. (2024 de 05 de 31). *Youtube*. Obtenido de Reynaldo Martell Avila: https://www.youtube.com/@reynaldomartellavila9051

Mixamo, Adobe. (31 de 05 de 2024). *Adobe*. Obtenido de Mixamo: https://www.mixamo.com/

OpenGL. (31 de 05 de 2024). *OpenGL*. Obtenido de The Industry's Foundation for High Performance Graphics: https://www.opengl.org/