Universidad Nacional Autónoma de México

Facultad de Ingeniería

División de Ingeniería Eléctrica

Computación Gráfica Avanzada

Profesor: ING. REYNALDO MARTELL AVILA

Semestre 2020-2

Grupo: 1

Proyecto final: MAZMORRA EMBRUJADA

Reporte del desarrollo del proyecto.

Alumnos:

Orozco Hernández Alexis

Zagoya Mellado Roberto Uriel

Introducción

“La mazmorra embrujada”, un lugar al otro del infierno, entre la vida y la muerte, entre la oscuridad, llena de almas perdidas e inquietas de todos aquellos que ya no están en este mundo, un lugar de asilo, para todos los seres incomprendidos, vagando, esperando el momento del juicio final, creado por un poder oscuro desconocido e inhumano, ha sido construido con el propósito de acabar con nuestro ridículo mundo como lo conocemos, alterando diferentes líneas de tiempo y espacio en todos los universos conocidos.

El tiempo se divide en tres etapas: presente, pasado y futuro, el mal que ha construido este lugar busca destruir estos tres tiempos, comenzando por nuestro futuro, el personaje Bender del universo de Futurama, es una clave para que el futuro no termine, ya que es quien por accidente evita que una guerra acabe con el futuro, por esto es que los poderes malévolos, han ocultado a este personaje dentro de la Mazmorra, así lograran completar su malévolo plan.

Debido a diferentes circunstancias, nuestro personaje Zelda ha perdido ya su universo, todo lo que conocía ya no existe, quedó devastado, para recuperarlo y evitar que a otros les suceda los mismo, y todo vuelva a la normalidad, ha aprendido diferentes habilidades, principalmente las de caminar entre universos, y líneas del tiempo, al fin ha encontrado esta mazmorra, primero tiene que salvar el futuro, ya que fue donde perdió su universo, por ende tiene que rescatar a Bender, es una pieza clave para establecer esa línea del tiempo, pero debe darse prisa ya que planean eliminarlo.

Esta misión no es sencilla, tiene que recorrer la mazmorra, buscando los 6 jarrones que juntos tienen el poder para abrir la puerta de la prisión donde se encuentra Bender, además, de tener que hallar la misma prisión, pero como ya se mencionó, dentro de la mazmorra, se encuentran los ejércitos de los verdaderos condenados, almas en pena, que ahora adquieren forma de fantasma, intolerantes a todo ser ajeno, su sed de sangre recorre los pasillos, imitan los llantos de las víctimas torturadas, arrastradas, encerradas y asesinadas, seres que nadie quiere ver ni en sueños y ahora en esta misión, esas pesadillas se vuelven realidad, por lo que nuestro personaje los tiene que enfrentar.

Objetivo

El objetivo es que nuestro personaje protagonista, la princesa Zelda, rescate a Bender, que está dentro de una celda en la mazmorra, pero también para abrir dicha celda, tiene que recolectar los 6 jarrones dentro de la misma, buscando, la prisión y los jarrones.

Nuestro personaje Zelda, debe recorrer la mazmorra embrujada, para poder liberar a Bender, el segundo personaje, para abrir la puerta donde está encerrado Bender, debe recolectar 6 jarrones, esto se realiza simplemente buscando los mismos y colocándose sobre ellos, de lo contrario dicha puerta se encontrará cerrada y no podremos rescatar a nuestro segundo personaje, la mazmorra tiene una estructura similar a la de un laberinto, para aumentar la dificultad del juego, una vez recolectado los jarrones, la puerta se abrirá, pero debes buscarla, y al tener contacto con el personaje Bender, finaliza el juego, lo sabrás, porque se mostrará un letrero indicando y además, te indicará que para reiniciar el juego presionas la letra Q.

Además, entre algunos pasillos se encuentran los fantasmas que son los enemigos de nuestro juego, pueden eliminarte y pierdes la partida, por lo que debes evitar tener contacto con ellos, tú también puedes eliminarlos, acertando uno o varios golpes, con la espada de la princesa Zelda, este ataque se realiza con la tecla E, y lo tienes que hacer acercándote a los fantasmas y presentándola, de lo contrario, no les causarás ningún daño.

Una vez completada la misión o si los enemigos te llegan a eliminar, tienes la opción de reiniciar la partida, comenzando desde el inicio es te juego, solo con presionar la tecla Q, como te lo indicará un letrero que aparecerá en la pantalla.

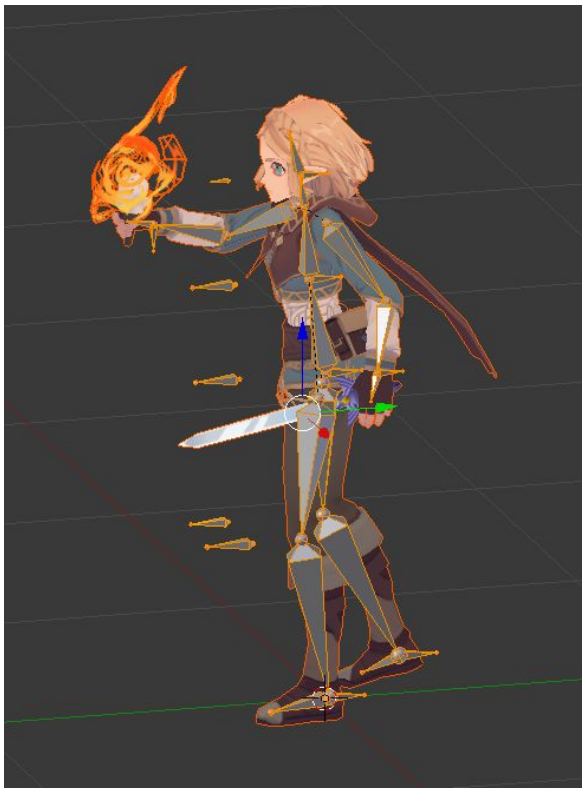
Desarrollo

Animación por huesos

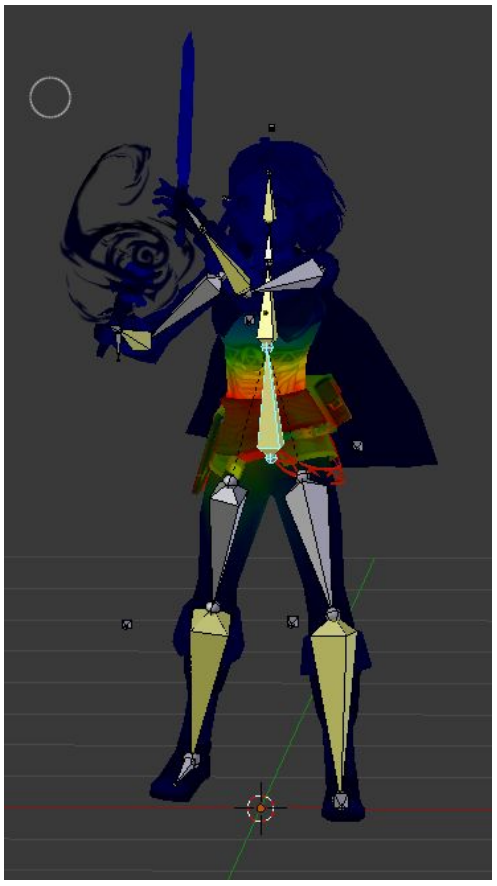
Las animaciones utilizadas para el personaje principal del juego fueron realizadas desde cero por nosotros en blender, para lograr las animaciones nos basamos en el procedimiento visto en clases para agregar huesos al modelo y posteriormente hacer los keyframes de cada animación. a continuación se describe de manera resumida el procesos seguido para crear las animaciones.

Creación del esqueleto

siguiendo el procedimiento visto en clases, agregamos huesos al modelo acomodandolos a la forma que tiene objeto en los pies, brazos, torso y cabeza teniendo como resultado el siguiente esqueleto para el personaje:



Posteriormente de tener todo acomodado como queríamos le asignamos pesos a los huesos para poder hacer que el modelo se deforme dependiendo de las posiciones (rotaciones y traslaciones) de cada hueso, modificando después el peso de cada hueso para poder obtener un “movimiento” o deformación más realista del modelo utilizando la herramienta de “weight paint”, donde seleccionamos cada hueso y modificamos el color de las zonas que queremos que se vean afectadas o no por el movimiento de dicho hueso, donde el color azul representa que no se verá afectada la zona.

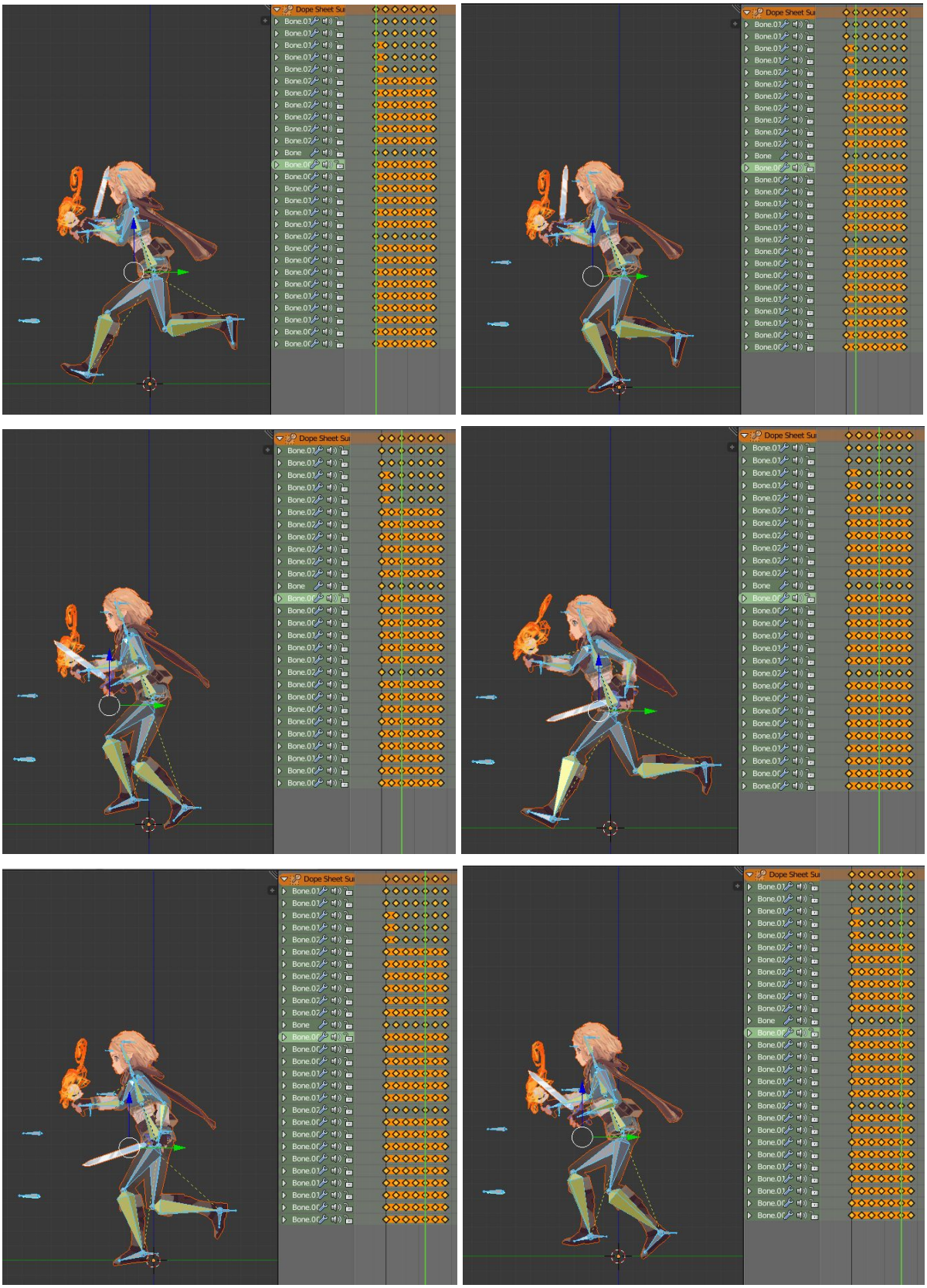


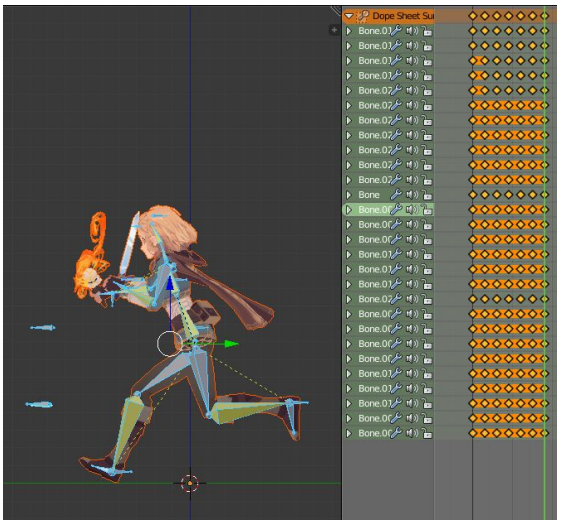
hasta aquí realmente nos encontramos con un problema, y es que no todos los huesos se podían mover ni rotar, desconocemos la razón de esta situación por lo cual buscamos en internet más formas de hacer este tipo de animación, encontrando entonces que se podría agregar otros huesos al modelo los cuales serían como una referencia para hacer los movimientos, es decir, si ese hueso se mueve afecta las transformaciones de los huesos que ya teníamos. Realizando esto logramos hacer que los huesos se movieran y ahora si empezar a trabajar en las animación (agregamos tantos huesos como consideramos necesarios y en las zonas que creíamos más convenientes para los movimientos del modelo). estos huesos se pueden ver en la primera imagen, los cuales son los huesos perpendiculares al esqueleto del modelo.

animaciones creadas

Decidimos crear 5 animaciones para el personaje, las cuales son: caminar, atacar, morir, correr y una animación para cuando no se esté moviendo el personaje (idle). Para realizar las animaciones modificamos las posiciones de los huesos para poder obtener uno a uno los keyframes que consideramos necesarios para que la animación se viera de la mejor manera posible.

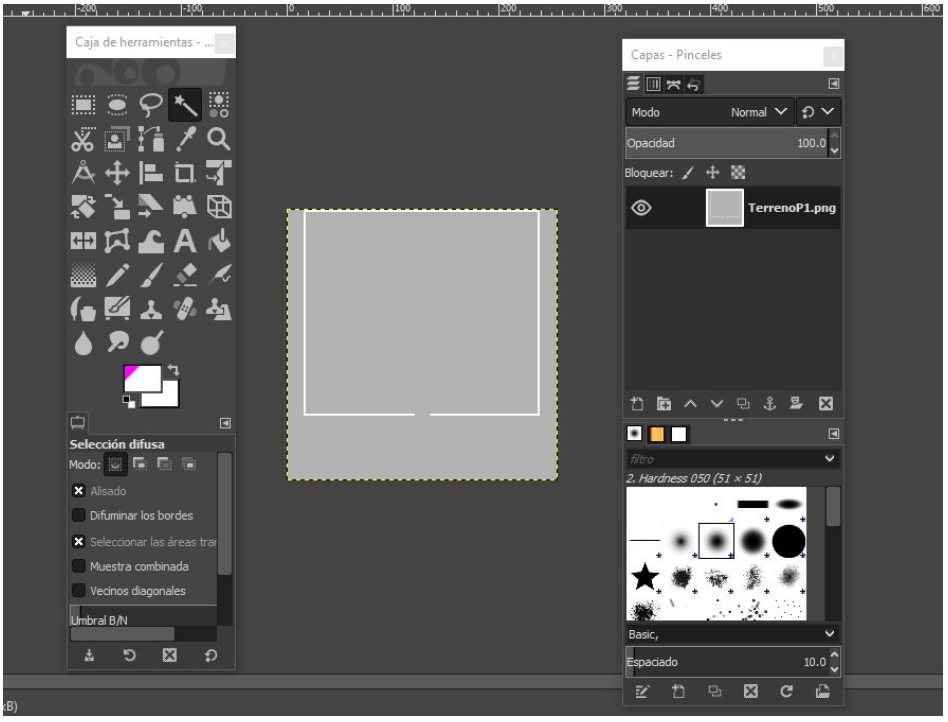
en las siguientes imagenes se veran uno a uno los keyframes creados para la animación de correr.



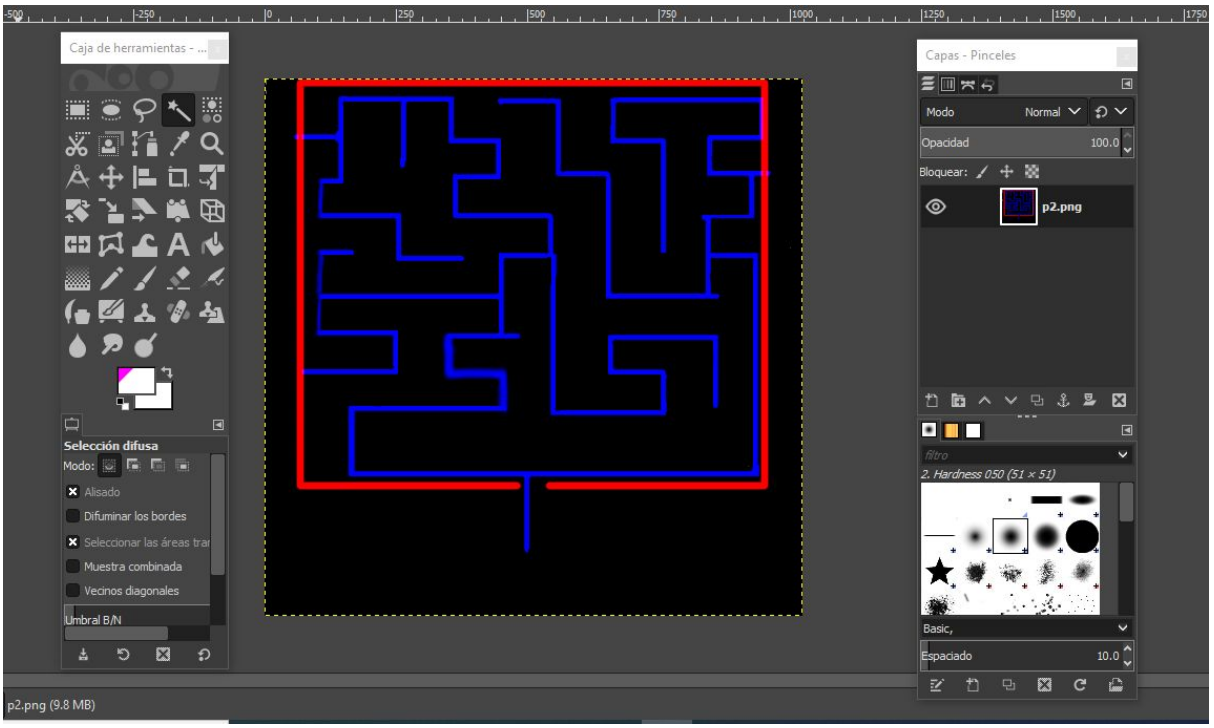


Mapa de alturas de alturas:

Para las alturas del terreno usamos GIMP, como se vieron en prácticas anteriores, creamos una imagen de 256 pixeles y se le asignó el fondo de gris con formato RGB(70,70,70), para después colocar diferentes “líneas” de colores más cercano al blanco, como es el caso del contorno de la mazmorra que se usó un formato RGB(150,150,150), esto para tener una textura más elevada, como se mencionó, para nuestro caso, solo lo usamos para el contorno de la mazmorra, ya que por razones estéticas y por las colisiones, optamos por usar modelos



Lo mismo se usó para las texturas:



En el color azul montamos la textura del camino de sangre y en la roja la textura para el contorno del mapa de alturas debido a que, son imágenes de diferente tamaño, se optó por usar un excel, para realizar las transformaciones:

	Gimpx1024		a	Gimpox256			OpenGL			Centro en OpenGL		
	X	Z					X	Z		X	Z	Distancia
	923.648	768		230.912	192		80.4	50				
	855.04	768		213.76	192		67	50				
	768.432	768		192.108	192		50.084375	50				
	717.824	768		179.456	192		40.2	50				
	649.216	768		162.304	192		26.8	50				
	580.608	768		145.152	192		13.4	50				
				0	0		-100	-100				
	192	688		48	172		-62.5	34.375		6.8359375	34.375	

Convirtiendo de las coordenadas de 1024 a 256 pixeles y después a opengl, para colocar objetos posteriormente:

en las líneas de código que cargamos las textura del suelo son las siguientes:

```
930     // Definiendo la textura a utilizar
931     //Texture textureTerrainBackground("../Textures/grassy2.png");
932     Texture textureTerrainBackground("../Textures/Suelo1.png");
933     // Carga el mapa de bits (FIBITMAP es el tipo de dato de la libreria)
934     bitmap = textureTerrainBackground.loadImage();
935     // Convertimos el mapa de bits en un arreglo unidimensional de tipo unsigned char
936     data = textureTerrainBackground.convertToData(bitmap, imageWidth,
```

para cargar la textura del camino de sangre:

```
1030    // Definiendo la textura a utilizar
1031    //Texture textureTerrainB("../Textures/path.png");
1032    Texture textureTerrainB("../Textures/Sangre1.png");
1033    // Carga el mapa de bits (FIBITMAP es el tipo de dato de la libreria)
1034    bitmap = textureTerrainB.loadImage();
1035    // Convertimos el mapa de bits en un arreglo unidimensional de tipo unsigned char
1036    data = textureTerrainB.convertToData(bitmap, imageWidth,
1037        imageHeight);
1038    // Creando la textura con id 1
1039    glGenTextures(1, &textureTerrainBID);
1040    // Enlazar esa textura a una tipo de textura de 2D.
```

Para cargar la textura del contorno, y de la altura del mapa:

```
963
964     // Definiendo la textura a utilizar
965     Texture textureTerrainR("../Textures/Acero.png");
966     // Carga el mapa de bits (FIBITMAP es el tipo de dato de la libreria)
967     bitmap = textureTerrainR.loadImage();
968     // Convertimos el mapa de bits en un arreglo unidimensional de tipo unsigned char
969     data = textureTerrainR.convertToData(bitmap, imageWidth,
970         imageHeight);
971     // Creando la textura con id 1
972     glGenTextures(1, &textureTerrainRID);
973     // Enlazar esa textura a una tipo de textura de 2D.
```

Como se puede observar, cada textura se carga en un color diferente

Mapa de alturas:

Audio:

Declaramos variables:

```
// OpenAL Defines
//Definimos el numero de buffers que vamos a crear.
//Para agregar una  septima, modificamos a 10.
#define NUM_BUFFERS 9 //agregamos una 9 funete de audio, ponemos 10 en los dos
#define NUM_SOURCES 9
#define NUM_ENVIRONMENTS 1
// Listener
//Para las fuente de sonido de nuestros mismos datos
ALfloat listenerPos[] = { 0.0, 0.0, 4.0 };
ALfloat listenerVel[] = { 0.0, 0.0, 0.0 };
ALfloat listenerOri[] = { 0.0, 0.0, 1.0, 0.0, 1.0, 0.0 };
```

Los diferentes vectores, para cada uno de los buffer, en nuestro caso habrá 9 buffer

```
// Buffers
//Arreglo de buffers, con el numero definido anteriormente
ALuint buffer[NUM_BUFFERS];//Arreglo de buffers, con el numero definido anteriormente
ALuint source[NUM_SOURCES];
ALuint environment[NUM_ENVIRONMENTS];
```

Para que se pueda reproducir:

```
//Se agrega un true, por cada fuente.
std::vector<bool> sourcesPlay = { true, true, true, true, true, true, false, true, true };
```

Configuración de las fuentes de sonido

```
//Aceleracion
//Condfiguración de la fuente de sonido, el comportamiento de que tan fuerte se escucha
//Así como la ganancia(de una señal, que tan fuerte se escucha)
//Cambiamos el indice del arreglo
alSourcef(source[3], AL_PITCH, 1.0f);//que tan rapido se reproduce el sonido
alSourcef(source[3], AL_GAIN, 1.0f);//ganancia, rango de 0 a 1, es decir 1, el volumen normal del audio
alSourcefv(source[3], AL_POSITION, source3Pos);//posicion de la fuente, cambia constantemente, con respeco al modelo
alSourcefv(source[3], AL_VELOCITY, source3Vel);//Velocidad, no se ocupa, pero hay que configurarla
alSourcei(source[3], AL_BUFFER, buffer[3]);//Buffer asosiado a esa fuente
alSourcei(source[3], AL_LOOPING, AL_TRUE);//Una vez que ya empezo a reproducir, si se va a repetir el sonido, para que no se repita
alSourcef(source[3], AL_MAX_DISTANCE, 500);//Umbral, para saber que tanto nos alejamos, para que se escuche menos
```

Posición del sonido

```
//indicarle cual es la posición del sonido,
source3Pos[0] = 13.4;
source3Pos[1] = 10.0;
source3Pos[2] = 52.8;
//Indicamos cual es la fuente de sonido que queremos enviarle
//Posicion y valor
alSourcefv(source[3], AL_POSITION, source3Pos);
```

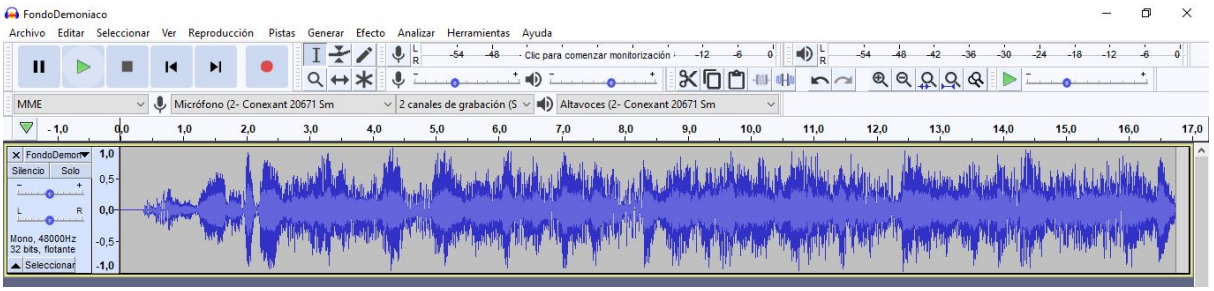
Finalmente le damos la reproducción del mismo:

```
shaderParticlesFire.turnOn();
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureParticleFireID);
glDepthMask(GL_FALSE);
glBindVertexArray(particleArray[drawBuf]);
glVertexAttribDivisor(0, 1);
glVertexAttribDivisor(1, 1);
glVertexAttribDivisor(2, 1);
glDrawArraysInstanced(GL_TRIANGLES, 0, 6, nParticlesFire);
glBindVertexArray(0);
glDepthMask(GL_TRUE);
drawBuf = 1 - drawBuf;
shaderParticlesFire.turnOff();

/**************************+
* Open AL sound data
*/
source1Pos[0] = modelFireParticles[3].x;
source1Pos[1] = modelFireParticles[3].y;
source1Pos[2] = modelFireParticles[3].z;
alSourcefv(source[1], AL_POSITION, source1Pos);
```

Para nuestro caso usamos los siguientes audios, para editar cada audio s e utilizó audacity:

Audacity:



Para volverlo mono y manejar un poco los decibeles.

Colliders

Para el caso del as colisiones, se utilizó las siguientes condiciones:

Para que el fantasma, baje la vida o pueda ser eliminado:

```
2162            for (; jt != collidersOBB.end(); jt++) {
2163                if (testSphereOBox(std::get<0>(it->second),
2164                    std::get<0>(jt->second))) {
2165                    if (it->first == "fantasma0" && fantasmasVivos[0] == true) {
2166                        if (animationIndex == 1 || animationIndex == 0 || animationIndex == 2) {
2167                            vida = vida - 1;
2168                            std::cout << "Colision " << it->first << " with "
2169                                << jt->first << vida << std::endl;
2170                            if (vida <= 0) {
2171                                vivo = false;
2172                            }
2173                        }
2174                        else if (animationIndex == 3) {
2175                            fantasmasVidas[0] = fantasmasVidas[0] - 1;
2176                            std::cout << "Vida fantasma" << fantasmasVidas[0] << std::endl;
2177                            if (fantasmasVidas[0] <= 0) {
2178                                //fantasmaVivo = false;
2179                                fantasmasVivos[0] = false;
2180                            }
```

Básicamente lo que se hizo, es verificar si había colisión y evaluar que el índice de animación, si no estaba en modo ataque el personaje de Zelda, entonces el fantasma le restaba vida, de lo contrario el personaje le restaba vida al fantasma y lo elimina, esto para cada uno de los fantasmas, verificando su etiqueta, fantasma0, fantasma1, etc. Cbe resaltar que se usaron colliders de esferas, para los fantasmas debido a su forma.

```
2096                std::cout << "Colision " << it->first << " with "
2097                    << jt->first << std::endl;
2098                int op;
2099                std::istringstream(it->first) >> op;
2100                jarronE[op] = false;
2101
2102                cuentaJarrones = 0;
2103                for (int i = 0; i <= jarronPosition.size(); i++) {
2104                    if (jarronE[i] == false) {
2105                        cuentaJarrones++;
2106                    }
2107                }
2108                if (cuentaJarrones == 6) {
2109                    rotBuzzLeftArm = (-1.57); abrir = 10;
2110                    rotBuzzRightArm = (-1.57);
2111                }
2112                //std::cout << jarronEtiqueta[0] << std::endl;
2113                dibujaJarron = false;
2114                isCollision = true;
2115            }
```

Para la colisión de los jarrones, se realizó la creación de unas nuevas cajas para los colliders, debido a que se requería que pudieran ser traspasados sin detenerse, y los demás objetos con las cajas no podían permitir el paso, es decir el personaje puede atravesar a los jarrones, pero los jarrones al personaje no, pero estos nunca lo intentara ya que son estáticos cada que se haga colisión con estos, el jarrón desaparecerá y se hará un recuento del total de jarrones, en caso de ser 6, la puerta final, donde esta Bender, nuestro personaje a rescatar, se abrirá, rotando los puertas, y desplazando la colisión que las rodea.

Colisión con Bender:

```
2120        for (std::map<std::string,
2121            std::tuple<AbstractModel::OBB, glm::mat4, glm::mat4> >::iterator it =
2122            collidersOBB.begin(); it != collidersOBB.end(); it++) {
2123            bool isCollision = false;
2124            for (std::map<std::string,
2125                std::tuple<AbstractModel::OBB, glm::mat4, glm::mat4> >::iterator jt =
2126                collidersOBB.begin(); jt != collidersOBB.end(); jt++) {
2127                if (it != jt
2128                    && testOBBOBB(std::get<0>(it->second),
2129                        std::get<0>(jt->second))) {
2130                    if (it->first == "Princesa" && jt->first == "mayow") {
2131                        complet = true;
2132                    }
2133                    isCollision = true;
2134                }
2135            }
2136            addOrUpdateCollisionDetection(collisionDetection, it->first, isCollision);
2137        }
```

Para estas colisiones, simplemente se verifica de que exista, para después asignar un valor a true, que bloqueará los controles y también desplegará el letrero de que se ha ganado la partida.

```
1510    if (modelSelected == 2 && glfwGetKey(window, GLFW_KEY_LEFT) == GLFW_PRESS && vivo == true && complet == false) {
1511        modelMatrixMayow = glm::rotate(modelMatrixMayow, glm::radians(5.0f), glm::vec3(0, 1, 0));
1512        animationIndex = 0;
1513    }
```

Los controles verifican que este valor es false, si es verdadero y aún está vivo, se pueden accionar, de lo contrario no.

Controles

Lo más importante a destacar sobre los controles, es que, con el botón de reinicio, restablece todos los valores como los jarrones que se usan, las vidas de todos los fantasmas y la del personaje protagonista.

```
if (modelSelected == 2 && glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS) {
    //modelMatrixMayow = glm::rotate(modelMatrixMayow, glm::radians(3.0f), glm::vec3(0, 1, 0));

    //animationIndex = 3;
    if (vivo == false || complet == true) {
        vivo = true;
        //modelMatrixMayow = glm::translate(modelMatrixMayow, glm::vec3(0, 0, 60));
        modelMatrixMayow = glm::translate(modelMatrixArco, glm::vec3(0, 0, 10));
        modelMatrixMayow = glm::rotate(modelMatrixMayow, glm::radians(-180.0f), glm::vec3(0, 1, 0));
        vida = 10;
        animationIndex = 1;
        for (int i = 0; i < fantasmaPosition.size(); i++) {
            fantasmasVivos[i] = true;
            fantasmasVidas[i] = 2;
        }
        //trasladaPersonaje = true;
        //jarronE[] = { true, true, true, true , true , true };
        for (int i = 0; i < jarronOrientation.size(); i++) {
            jarronE[i] = true;
        }

        complet = false;
        //abrir = 0;
        rotBuzzLeftArm = (0.0); abrir = 0.0;
        rotBuzzRightArm = (0.0);

    }
```

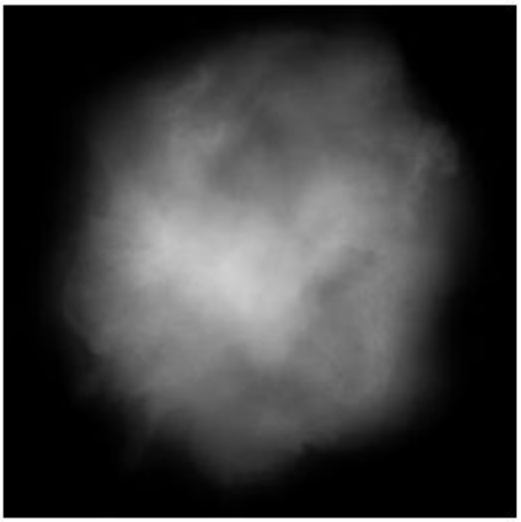
Partículas

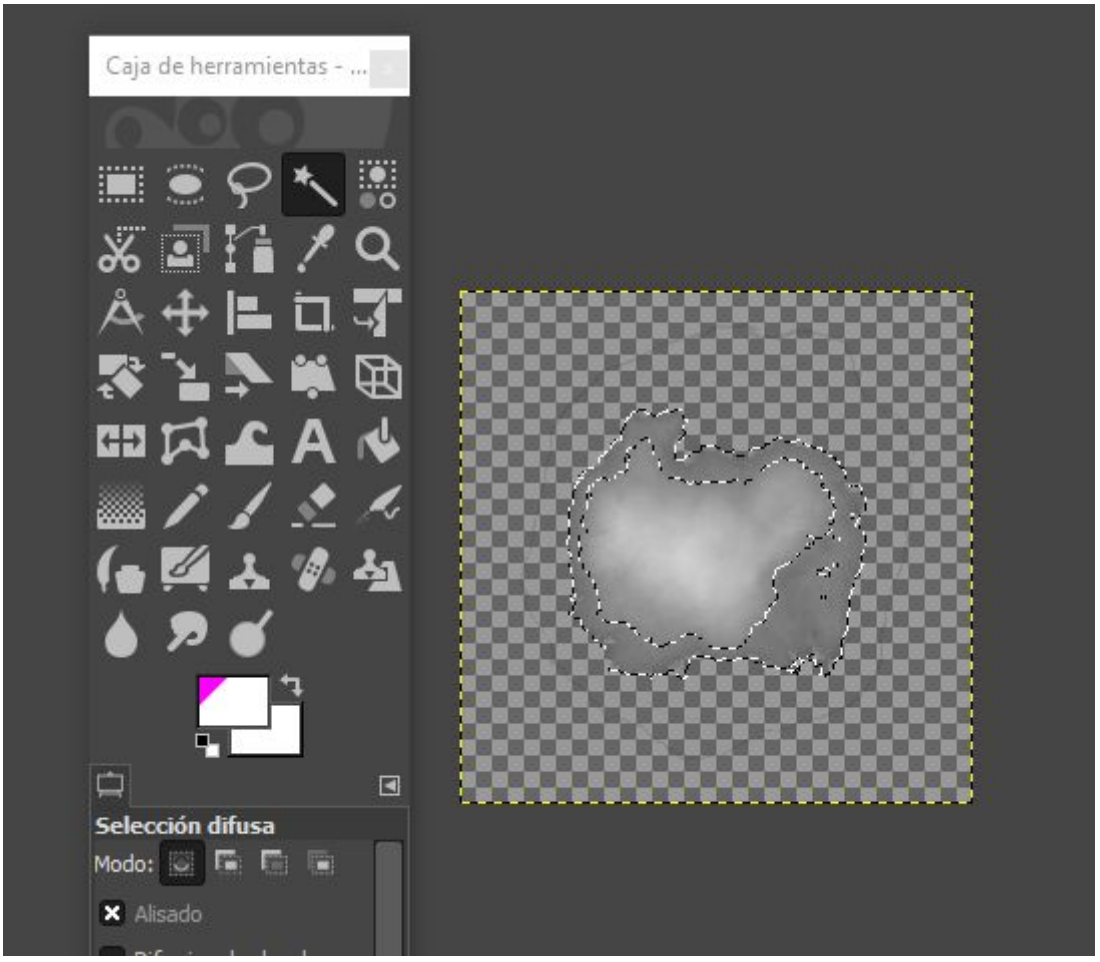
Para nuestro caso se utilizaron las partículas de humo(las que utilizamos para el agua en prácticas anteriores, solo que con otra textura, de humo) y de fuego, las partículas de fuego se utilizaron para las antorchas, mientras que para las de humo, simula un geiser en la entrada principal, solo que este sale de una jaula.

Cargamos la imagen

```
//Texture textureParticlesFountain("../Textures/bluewater.png");
Texture textureParticlesFountain("../Textures/smokeTransparencia.png");
bitmap = textureParticlesFountain.loadImage();
data = textureParticlesFountain.convertToData(bitmap, imageWidth, imageHeight);
glGenTextures(1, &textureParticleFountainID);
glBindTexture(GL_TEXTURE_2D, textureParticleFountainID);
// set the texture wrapping parameters
```

Previamente editada con GIMP, para agregar transparencia.





Disminuimos la gravedad, para dar el efecto deseado:

```
3001            shaderParticlesFountain.setInt("ParticleTex", 0);
3002            shaderParticlesFountain.setVectorFloat3("Gravity", glm::value_ptr(glm::vec3(0.0f, -0.03f, 0.0f)));
3003            shaderParticlesFountain.setMatrix4("model", 1, false, glm::value_ptr(modelMatrixParticlesFountain));
3004            glBindVertexArray(VAOParticles);
```

Así como parámetros de tiempo:

```
621     float time = 0.0f;
622     //float rate = 0.00075f;
623     float rate = 0.001f;
624     for (unsigned int i = 0; i < nParticles; i++) {
625         data[i] = time;
626         time += rate;
627     }
```

Para las partículas del fuego de las antorchas, declaramos las posiciones que se van a declarar:

```
467     // Blending model unsorted
468     std::map<std::string, glm::vec3> blendingUnsorted = {
469             {"fountain", glm::vec3(0.0, 0.0, 52.0)},
470             {"fire1", glm::vec3(15.4, 0.0, 55.0)},
471             {"fire2", glm::vec3(-11.4, 0.0, 55.0)},
472             {"fire3", glm::vec3(-46.8671875, 0, -56.5)},
473             {"fire4", glm::vec3(50.4375, 0.0, -27.0)},
474             {"fire5", glm::vec3(-55.03125, 0.0, 15.8)},
475             {"fire6", glm::vec3(69.9, 0.0, 38.7)},
476     };
477
```

Cargamos su textura:

```
1119        Texture textureParticleFire("../Textures/fire.png");
1120        bitmap = textureParticleFire.loadImage();
1121        data = textureParticleFire.convertToData(bitmap, imageWidth, imageHeight);
1122        glGenTextures(1, &textureParticleFireID);
1123        glBindTexture(GL_TEXTURE_2D, textureParticleFireID);
```

Y copiamos el proceso de renderizado, con sus valores, para cada uno de los fuegos:

```
3013    else if (renderParticles && it->second.first.compare("fire1") == 0) {
3014        /**********
3015         * Init Render particles systems
3016         */
3017        lastTimeParticlesAnimationFire = currTimeParticlesAnimationFire;
3018        currTimeParticlesAnimationFire = TimeManager::Instance().GetTime();
3019
3020        shaderParticlesFire.setInt("Pass", 1);
3021        shaderParticlesFire.setFloat("Time", currTimeParticlesAnimationFire);
3022        shaderParticlesFire.setFloat("DeltaT", currTimeParticlesAnimationFire - lastTimeParticlesAnimationFire);
3023
3024        glActiveTexture(GL_TEXTURE1);
3025        glBindTexture(GL_TEXTURE_1D, texId);
3026        glEnable(GL_RASTERIZER_DISCARD);
3027        glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, feedback[drawBuf]);
3028        glBeginTransformFeedback(GL_POINTS);
3029        glBindVertexArray(particleArray[1 - drawBuf]);
3030        glVertexAttribDivisor(0, 0);
3031        glVertexAttribDivisor(1, 0);
3032        glVertexAttribDivisor(2, 0);
3033        glDrawArrays(GL_POINTS, 0, nParticlesFire);
3034        glEndTransformFeedback();
3035        glDisable(GL_RASTERIZER_DISCARD);
3036
3037        shaderParticlesFire.setInt("Pass", 2);
3038        glm::mat4 modelFireParticles = glm::mat4(1.0);
3039        modelFireParticles = glm::translate(modelFireParticles, it->second.second);
3040        modelFireParticles[3][1] = terrain.getHeightTerrain(modelFireParticles[3][0], modelFireParticles[3][2]) + 9.0;
```

Para modificar la configuración del comportamiento como el alto y ancho:

```
487     // Definition for the particle system fire
488     GLuint initVelFire, startTimeFire;
489     GLuint VAOParticlesFire;
490     GLuint nParticlesFire = 2000;
491     GLuint posBuf[2], velBuf[2], age[2];
492     GLuint particleArray[2];
493     GLuint feedback[2];
494     GLuint drawBuf = 1;
495     float particleSize = 0.5, particleLifetime = 3.0;
496     double currTimeParticlesAnimationFire, lastTimeParticlesAnimationFire;
```

Partículas de humo:

Partículas de fuego:

Resultados

Como resultado del proyecto pudimos obtener un juego bastante cercano a lo que habíamos imaginado desde un principio, logrando la implementación de lo visto en clases. por ejemplo, logramos hacer que el personaje muera al ser tocado múltiples veces por algún fantasma, es decir que fuera perdiendo vida hasta morir y de igual manera hicimos que el personaje atacara con una espada y fuera quitando vida a los enemigos para poder matarlos y así “liberar” la zona.

También consideramos que pudimos darle un objetivo al juego, el cual consiste en recolectar los jarrones para poder abrir una puerta y rescatar a otro personaje, cosa que no habíamos considerado muy bien al inicio, es decir, el objetivo del juego no era claro y definido de buena manera pero logramos corregir ese problema.

En cuanto a la ambientación obtuvimos un buen resultado utilizando sonidos de fantasmas, la iluminación en las antorchas, así como las partículas para hacer que salga humo de ellas.

Por último, consideramos que a futuro se podría implementar que los enemigos sigan al personaje principal con la finalidad de poder agregar un poco más de dificultad al juego. tambien se podria implementar un ataque a distancia para los fantasmas usando partículas, como una especie de bola de energia o algo parecido, de igual manera se podría agregar un disparo de flechas al personaje junto con más enemigos. de igual manera se podrían hacer animaciones para los enemigos con el objetivo de hacer el juego más atractivo visualmente.

Conclusión

Desarrollar este proyecto nos ayudó a comprender de mejor manera todos los temas vistos en clase y sus implementaciones mediante opengl, es decir, pudimos entender cómo se debe realizar la implementación de la mejor manera posible para poder obtener un buen resultado de proyecto final, por ejemplo, pusimos en práctica lo que aprendimos de gimp para poder utilizar los mapas de alturas y así mismo usar otro mapa para poner texturas en sitios determinados de nuestra escena/mundo.

De igual manera pusimos en práctica lo que aprendimos de blender para poder editar modelos que queríamos usar para este proyecto, con lo cual pudimos hacer las modificaciones necesarias a cada modelo para que se pudiera cargar en opengl, por ejemplo, ajustamos o incluso modificamos las texturas de los modelos dado que en un inicio dichas texturas nos causaban problemas para poder ejecutar el programa. Además las animaciones del proyecto son creadas totalmente por nosotros desde cero utilizando blender, esto debido a que no nos funcionaban las animaciones que descargamos de páginas como mixamo. Desconocemos realmente cual sea el problema ya que en blender se podían ver bien las animaciones pero al momento de cargarlas en opengl los personajes hacían movimientos raros o incluso no se movían y sufrían deformaciones en sus mallas

También pusimos en práctica y entendimos mejor que modificaciones se deben de realizar a diferentes parámetros para poder tener un resultado cercano a la realidad y a lo que queríamos de manera visual y auditiva para ambientar nuestro juego en diferentes cosas como: luces, particulas y sonido. Estos temas los implementamos para dar el efecto de humo a las antorchas que se localizan en el juego y que dichas antorchas ilumen parte del mismo, además de agregar sonidos de fantasmas que se escuchan en ambas bocinas, con la finalidad de dar la sensación de que los enemigos están en todos lados y no se escuchen más de un lado que de otro logrando así un resultado que nos deja satisfechos.

En general consideramos que el proyecto fue muy entretenido de hacer y bastante didáctico para poder reforzar lo que vimos a lo largo del semestre, los mayores problemas que tuvimos para poder realizar el proyecto fueron básicamente dos: el primer problema tiene que ver con el hecho de que no nos podíamos dividir como tal el trabajo y después juntarlo,

dado que la computadora de Alexis Orozco Hernández no corre los programas que hemos realizado durante el semestre, por este motivo nuestro trabajo fue en su gran mayoría en conjunto mediante videollamadas.

el segundo problema tiene que ver más con los modelos y animaciones que queríamos usar, dado que hubo varios modelos que no supimos y no pudimos corregir los detalles que evitaban los pudiéramos cargar en opengl, lo cual consideramos nos atrasó un poco porque teníamos que buscar varios modelos, probar si cargaban, corregir problemas y después hacer las animaciones nosotros mismos, pero como ya se mencionó anteriormente esto nos sirvió bastante para poder poner en práctica lo que aprendimos de blender.

Ruta de git documentación:

https://github.com/Urielzm/CGA_PF_MazmorraEmbrujada_Documentacion_OHA_ZMRU.git

Ruta de git proyecto:

https://github.com/Urielzm/CGA_PF_MazmorraEmbrujada_Proyecto_OHA_ZMRU.git

Enlace del video:
https://drive.google.com/file/d/1iNis9tYlj4kN6APB-qDXxNwWQmu3HfvN/view?usp=sharing