Universidad Nacional Autónoma de México

Facultad de Ingeniería

PROYECTO FINAL

MANUAL TÉCNICO

Cortez Ibarra Derek
Medina Cruz Josué Emanuel

Grupo: 4

Semestre: 2021-I

Fecha de entrega: 29 de enero de 2021

Manual Técnico

Objetivos

El objetivo del presente proyecto es el de desarrollar un espacio virtual en la que se observe la casa de “Coraje el Perro Cobarde” y la sala de la casa. Así como un ático con temática de Pokémon,

La casa tendrá los muebles que se muestran en la imagen, en este caso los modelos de la televisión, mecedora, mazo, sillón, lámpara, coraje, puerta y máscara.

El ático tendrá de muebles una televisión, una laptop, un Nintendo switch, una silla giratoria, un libro, un pikachu de juguete, un reloj y una lámpara en forma de pokebola.

Se tendrán 10 animaciones, 4 de ellas complejas y 6 simples en las que algunos de los objetos mencionados serán animados.

Alcance del proyecto

En primer lugar, la fachada contará con el modelado de la casa completa por fuera y contar con las ventanas, chimenea y techo exterior como en la animación original de Coraje, asimismo contará con el modelo de un molino de viento y la camioneta de Justo al exterior de la casa.

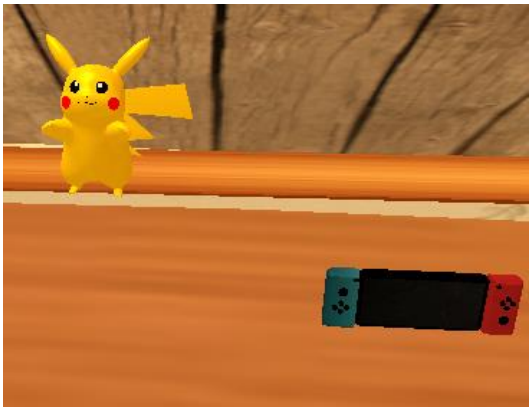
En el interior de la casa se modelará la sala de la casa la cual cuenta con los siguientes objetos:

- Coraje
- Mecedora
- Sillón
- Televisión
- Lampara
- Máscara de Justo

- Mazo (modelado con primitivas)
- Escaleras para el segundo piso
- Puerta
- Alfombra de la sala
- Silla giratoria
- Reloj
- Televisión en el ático
- Lampara en forma de pokebola
- Cama
- Alfombra
- Nintendo switch
- Mesa
- Libro
- Pikachu de juguete

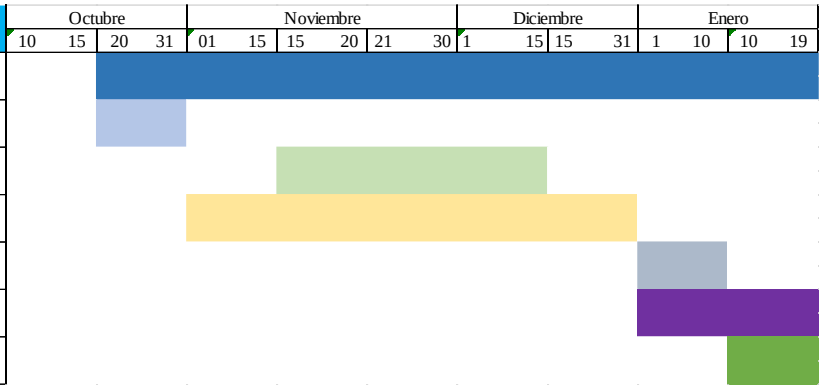
El proyecto en conjunto debe contener 9 animaciones de las cuales 4 de ellas son realizadas por key frames y 5 sencillas

Las animaciones complejas serán: el movimiento de la máscara de Justo simulando que quiere atrapar a Coraje, la segunda será Coraje tratando de escapar de la máscara y se esconderá detrás de la mecedora, la tercera será el movimiento y la rotación de la Silla giratoria y la cuarta serán los saltos junto al mortal hacia atrás del Pikachu.



Cuadro de actividades y Diagrama de Gantt

Actividad	Duración	Inicio	Final
Proyecto Final de Lab. de Comp.Graf.	53 días	21/10/2020	19/01/2021
Análsis de requerimientos	5 días	21/10/2020	27/10/2020
Modelado de objetos principales	16 días	17/11/2020	08/12/2020
Modelado de la casa	36 días	02/11/2021	05/01/2021
Modelado de objetos extra	8 días	04/01/2021	13/01/2021
Animaciones	8 días	07/01/2021	17/01/2021
Manuales técnico y usuario	3 días	15/01/2021	19/01/2021



Documentación del código

Key Frames

Como se mencionó anteriormente se realizaron 2 animaciones por key frames en las cuales se cuentan con dos estructuras _frame2 y _frame3 los cuales corresponden a la máscara y a Coraje respectivamente

```
//Animacion 2 Mascara
typedef struct _frame2
{
	//Variables para GUARDAR Key Frames
	float posX;		//Variable para PosicionX
	float posY;		//Variable para PosicionY
	float posZ;		//Variable para PosicionZ
	float incX;		//Variable para IncrementoX
	float incY;		//Variable para IncrementoY
	float incZ;		//Variable para IncrementoZ

	float rotInc6, rotMascara;

}FRAME2;

//Animacion 3 Coraje
typedef struct _frame3
{
	//Variables para GUARDAR Key Frames
	float posX;		//Variable para PosicionX
	float posY;		//Variable para PosicionY
```

```
        float posZ;             //Variable para PosicionZ
        float incX;             //Variable para IncrementoX
        float incY;             //Variable para IncrementoY
        float incZ;             //Variable para IncrementoZ

        float rotInc7, rotCoraje;

}FRAME3;
```

Posteriormente se definen las dos funciones respectivas de **resetElement2()** y **resetElements3()** las cuales tienen la tarea de iniciar las variables en el key frame 0

Asimismo, se tienen las funciones de **interpolation2()** e **interpolation3()** que se encargan de calcular los punto intermedios entre cada key frame.

```
void interpolation3(void)
{

        KeyFrame3[playIndex3].incX = (KeyFrame3[playIndex3 + 1].posX -
KeyFrame3[playIndex3].posX) / i_max_steps;
        KeyFrame3[playIndex3].incY = (KeyFrame3[playIndex3 + 1].posY -
KeyFrame3[playIndex3].posY) / i_max_steps;
        KeyFrame3[playIndex3].incZ = (KeyFrame3[playIndex3 + 1].posZ -
KeyFrame3[playIndex3].posZ) / i_max_steps;

        KeyFrame3[playIndex3].rotInc7 = (KeyFrame3[playIndex3 + 1].rotCoraje -
KeyFrame3[playIndex3].rotCoraje) / i_max_steps;
}
```

En donde la variable **i_max_steps** define la cantidad de puntos intermedios que tiene cada frame

Por otro lado, la función que permite activar la animación mediante el teclado es **KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)** ya que identifica cuando el usuario presiona una tecla para activar la animación.

La función **animación()** detecta cuando la animación se activa.

Animaciones complejas

Animación de la máscara

Esta animación inicia con la Tecla ‘2’

Como se puede ver en el siguiente código el cual se encuentra en **KeyCallback**

```
if (keys[GLFW_KEY_2] && (FrameIndex > 1))
	{
		if (play2 == false )
		{
			MposX = 0;
			MposY = 0;
			MposZ = 0;
			rotMascara = 0;

			resetElements2();
			//First Interpolation
			interpolation2();

			play2 = true;
			playIndex2 = 0;
			i_curr_steps2 = 0;
		}
		else
		{
			play2 = false;
		}
	}
```

El cual a su vez invoca a **play2** el cual se encuentra en **animación()**

```
if (play2)
	{
		if (i_curr_steps2 >= i_max_steps)
		{
			playIndex2++;
			if (playIndex2 > FrameIndex - 2)
			{
				printf("Animacion de la mascara\n");
				playIndex2 = 0;
				play2 = false;
			}
			else //Next frame interpolations
			{
				i_curr_steps2 = 0; //Reset counter
				//Interpolation
```

```
                    interpolation2();
                }
            }
            else
            {
                //Draw animation
                MposX += KeyFrame2[playIndex2].incX;
                MposY += KeyFrame2[playIndex2].incY;
                MposZ += KeyFrame2[playIndex2].incZ;

                rotMascara += KeyFrame2[playIndex2].rotInc6;

                i_curr_steps2++;
            }
        }
```

Al iniciarse **play2**, se repite el proceso de interpolación hasta llegar a i_max_steps de la misma forma hace este proceso hasta que **playIndex2** llegue al máximo de frames.



Animación de coraje

Esta animación inicia con la Tecla ‘3’

Al igual que la anterior es muy similar solo que cambia el recorrido que Coraje realiza

```
//Mascara
KeyFrame2[0].posX = 0;
KeyFrame2[0].posY = 0;
KeyFrame2[0].posZ = 0;
KeyFrame2[0].rotMascara = 0;

KeyFrame2[1].posX = 2.0;
KeyFrame2[1].posY = 0;
KeyFrame2[1].posZ = 0.0;
KeyFrame2[1].rotMascara = 0;

KeyFrame2[2].posX = 2.0;
KeyFrame2[2].posY = 0.0;
KeyFrame2[2].posZ = 0.0;
KeyFrame2[2].rotMascara = -90;

KeyFrame2[3].posX = 2.0;
KeyFrame2[3].posY = 0.0;
KeyFrame2[3].posZ = -3.0;
KeyFrame2[3].rotMascara = -90;

KeyFrame2[4].posX = 2.0;
KeyFrame2[4].posY = 1.0;
KeyFrame2[4].posZ = -3.0;
KeyFrame2[4].rotMascara = -90;

KeyFrame2[5].posX = 2.0;
KeyFrame2[5].posY = 1.0;
KeyFrame2[5].posZ = -3.0;
KeyFrame2[5].rotMascara = -180;

KeyFrame2[6].posX = -0.5;
```

```
KeyFrame2[6].posY = 1.0;
KeyFrame2[6].posZ = -3.0;
KeyFrame2[6].rotMascara = 0;

KeyFrame2[7].posX = 0;
KeyFrame2[7].posY = 1.0;
KeyFrame2[7].posZ = 0;
KeyFrame2[7].rotMascara = 180;
///Coraje
KeyFrame3[0].posX = 0;
KeyFrame3[0].posY = 0;
KeyFrame3[0].posZ = 0;
KeyFrame3[0].rotCoraje = 0;

KeyFrame3[1].posX = 5.0;
KeyFrame3[1].posY = 0;
KeyFrame3[1].posZ = 0.0;
KeyFrame3[1].rotCoraje = 0;

KeyFrame3[2].posX = 5.0;
KeyFrame3[2].posY = 0.0;
KeyFrame3[2].posZ = 0.0;
KeyFrame3[2].rotCoraje = -90;

KeyFrame3[3].posX = 5.0;
KeyFrame3[3].posY = 0.0;
KeyFrame3[3].posZ = -6.0;
KeyFrame3[3].rotCoraje = -90;

KeyFrame3[4].posX = 26.0;
KeyFrame3[4].posY = 0.0;
KeyFrame3[4].posZ = -6.0;
KeyFrame3[4].rotCoraje = -90;

KeyFrame3[5].posX = 26.0;
KeyFrame3[5].posY = 0.0;
KeyFrame3[5].posZ = -6.0;
KeyFrame3[5].rotCoraje = 45;

KeyFrame3[6].posX = 26.0;
KeyFrame3[6].posY = 0.0;
KeyFrame3[6].posZ = 6.0;
KeyFrame3[6].rotCoraje = 45;
```



Animación de silla

Esta animación inicia con la Tecla ‘L’

Al iniciarse **play6**, se repite el proceso de interpolación hasta llegar a i_max_steps de la misma forma hace este proceso hasta que **playIndex** llegue al máximo de frames.

```
//Silla
	KeyFrame[0].rotSilla = 0;
	KeyFrame[1].rotSilla = 360;
	KeyFrame[0].movSilla = 0;
	KeyFrame[1].movSilla = 3.5;


if (play6)
	{
		if (i_curr_steps >= i_max_steps) //end of animation between frames?
		{
			playIndex++;
			if (playIndex > FrameIndex - 2)//end of total animation?
```

```
                {
                        playIndex = 0;
                        play6 = false;
                }
                else //Next frame interpolations
                {
                        i_curr_steps = 0; //Reset counter
                                                        //Interpolation
                        interpolation();
                }
        }
        else
        {
                //Draw animation
                posX += KeyFrame[playIndex].incX;
                posY += KeyFrame[playIndex].incY;
                posZ += KeyFrame[playIndex].incZ;

                rotSilla += KeyFrame[playIndex].inc1;
                movSilla += KeyFrame[playIndex].inc2;
                i_curr_steps++;
        }

}
```

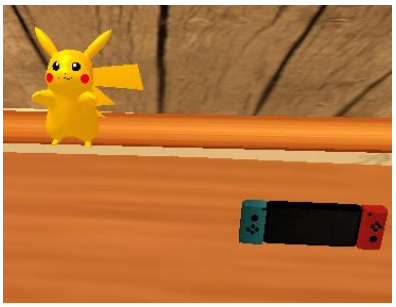
Animación de pikachu

Esta animación inicia con la Tecla ‘K’. Los saltos funcionan como un circuito de tres partes, la primera inicia el movimiento hacia arriba, la segunda baja el juguete y la tercera reinicia el primer movimiento.

```
if (circuito)
        {
                if (recorrido1)
                {
                        if (movKitY > 2.5)
                        {
                                recorrido1 = false;
                                recorrido2 = true;
                        }
                }
                if (recorrido2)
                {
                        movKitY -= 0.015f;
                        if (movKitY < 0)
                        {
                                recorrido2 = false;
                                recorrido3 = true;

                        }
                }
                if (recorrido3)
                {
                        movKitY += 0.03f;
                        if (movKitY > 2.5)
                        {
                                recorrido3 = false;
                                recorrido1 = true;
                        }
                }
        }

//Pikachu
        KeyFrame[0].rotPikachu = 0;
        KeyFrame[1].rotPikachu = 360;
```



Animaciones Simples

Animación de la puerta

Esta animación inicia con la Tecla ‘1’

Dentro de la función **animación()** se declara la animación de la puerta de la siguiente forma

```
if (play4)
{
	if (rotDoor > -90.0f) {
		rotDoor -= 1.0f;
	}

	else {
		if (rotDoor <= 90.0f) {
			rotDoor += 1.0f;
		}
		play4 = false;
	}
}
```

En donde se puede ver la rotación de la puerta la cual se inicializa en cero al inicio del código `float rotDoor = 0;` y al ser mayor a -90 va a decrementar -1.0 grados hasta llegar a 90 grados lo cual se puede ver en la apertura de la puerta.

Y en **KeyCallback** se activa con la tecla 1

```
if (keys[GLFW_KEY_1])
	{
		if (play4 == false)
		{
			rotDoor = 0.0f;
			play4 = true;
		}
	}
```

Animación de la mecedora

Esta animación inicia con la Tecla ‘4’

Dentro de la función **animación()** se declara la animación de la puerta de la siguiente forma

```
if (play)
	{
		if (rotMec > -10.0f) {
			rotMec -= 0.1f;
		}
		else {
			play = false;
		}
	}
```

Al igual que con la apertura de la puerta, en este caso la mecedora rota menos hasta -10.0 grados e incrementa de 0.1

En **KeyCallback** se activa con ‘4’

```
if (keys[GLFW_KEY_4])
	{
		if (play == false)
		{
			rotMec = 0.0f;
			play = true;
		}
	}
```

Animación que simula el encendido y apagado de la TV

Esta animación inicia con la Tecla ‘N’ para el encendido y con la Tecla ‘M’ para el apagado

En la TV se simula el encendido y apagado de esta mediante el traslado de la pantalla con la imagen del show de Coraje

Se activa con play 10 el cual suma de 0.01 unidades hasta llegar a 0.07 que es la posición en la que se encuentra la pantalla de la televisión y con play 10_1 se realiza su apagado

```
else if (play10) {
	if (movPan > 0.07f) {
		play10 = false;
	}
	else {
		movPan += 0.01f;
	}
}
else if (play10_1) {
	if (movPan < 0.0f) {
		play10_1 = false;
	}
	else {
		movPan -= 0.01f;
	}
}
```

Y en **DoMovement()** se activa con la tecla ‘N’ así como el encendido de la luz.

Con la tecla ‘M’ se apaga y la imagen desaparece

```
//TV encendido y apagado------------------------
	if (keys[GLFW_KEY_N])
	{
		play10 = !play10;
		active2 = !active2;
		if (active2)
			LightP2 = glm::vec3(25.0f, 59.0f, 215.0f);

	}
	if (keys[GLFW_KEY_M])
	{
		play10_1 = !play10_1;
```

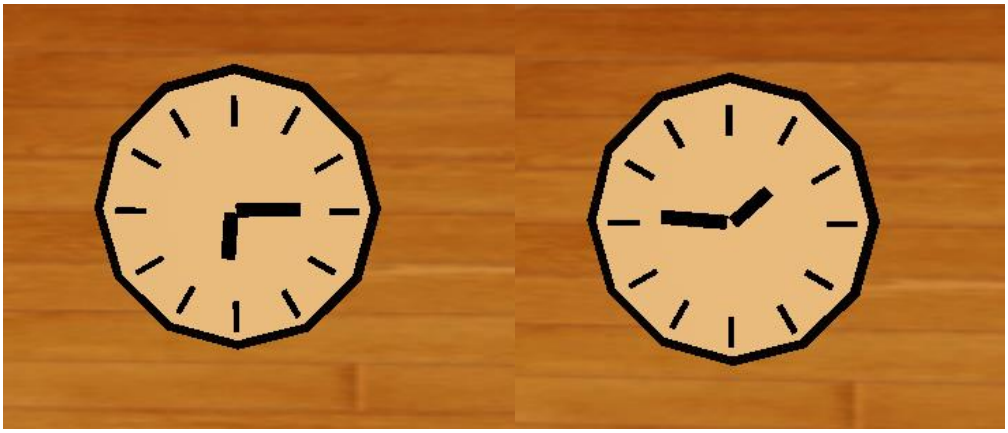
```
        active2 = !active2;
        if (active2)
            LightP2 = glm::vec3(0.0f, 0.0f, 0.0f);

}
```

Animación que simula un reloj

Esta animación inicia automáticamente

En el reloj se muestra la simulación de un objeto real al tener dos manecillas las cuales van moviéndose a un ritmo diferente dependiendo si es manecilla de minutos o de horas



```
//Reloj (2)
view = camera.GetViewMatrix();
//tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::rotate(model, -(float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

```
Reloj3.Draw(lightingShader);
//Reloj (3)
view = camera.GetViewMatrix();
//tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::rotate(model, -(float)glfwGetTime() / 50, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Reloj2.Draw(lightingShader);
```

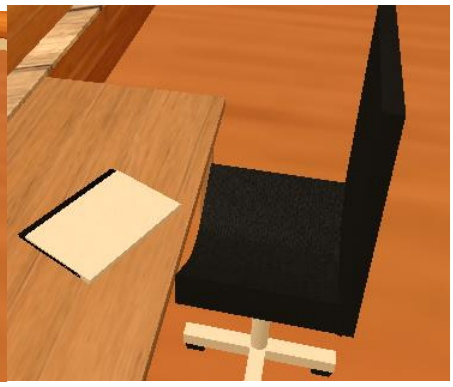
Animación de computadora

Esta animación inicia con las teclas ‘C’ y ‘V’

En la computadora simula la apertura y el cierre de ésta

Se activa con play9 y play9_1 el cual suma o resta unidades hasta llegar a 45 grados de apertura o 0 grados para que no traspase la base de la computadora

```
if (play9) {
	if (rotCompu > 45.0) {
		play9 = false;
	}
	else {
		rotCompu += 0.45f;
	}
}
if (play9_1) {
	if (rotCompu < 0.0) {
		play9_1 = false;
	}
	else {
		rotCompu -= 0.45f;
	}
}
```

Animación de lampara

Esta animación inicia la tecla espacio

Simula el encendido de una lampara real, debido a las de los modelos, se decidió que el rango de iluminación de la lampara fuera de 15 puntos.

```
if (keys[GLFW_KEY_SPACE])
	{
		active = !active;
		if (active)
			LightP1 = glm::vec3(25.0f, 25.0f, 15.0f);
		else
			LightP1 = glm::vec3(0.0f, 0.0f, 0.0f);
	}
```

Apagado

Encendido

Para el traslado automático de un cuarto a otro se agregó lo siguiente a **MouseCallback()**

```
if (keys[GLFW_KEY_8] == GLFW_PRESS)
	{
		camera.position.x = 10.0f;
		camera.position.y = 100.0f;
		camera.position.z = 50.0f;
	}

	if (keys[GLFW_KEY_9] == GLFW_PRESS)
	{
		camera.position.x = 10.0f;
		camera.position.y = 40.0f;
```

```
            camera.position.z = 50.0f;
        }
```

En este caso con la tecla 9 nos desplazamos automáticamente a la sala de Coraje y con la tecla 8 al cuarto de arriba en donde se encuentra Pikachu.

Main()

Algo importante a mencionar es que se debe inicializar cada frame para que haga la animación de forma correcta como en el caso de Coraje y la máscara

Para el dibujo del mazo por primitivas se realizó el siguiente código

```
// Diffuse map
        image2 = stbi_load("images/madera.jpg", &textureWidth2, &textureHeight2, &nrChannels2, 0);
        glBindTexture(GL_TEXTURE_2D, texture3);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, textureWidth2, textureHeight2, 0, GL_RGB, GL_UNSIGNED_BYTE, image2);
        glGenerateMipmap(GL_TEXTURE_2D);
        if (image2)
        {
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, textureWidth2, textureHeight2, 0, GL_RGB, GL_UNSIGNED_BYTE, image2);
            glGenerateMipmap(GL_TEXTURE_2D);
        }
        else
        {
            std::cout << "Failed to load texture" << std::endl;
        }
        stbi_image_free(image2);

        // Specular map
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

```
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);

	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST_MIPMAP_NEAREST);

	image2 = stbi_load("images/madera.jpg", &textureWidth2, &textureHeight2, &nrChannels2, 0);

	glBindTexture(GL_TEXTURE_2D, texture4);

	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, textureWidth2, textureHeight2, 0, GL_RGB, GL_UNSIGNED_BYTE, image2);

	glGenerateMipmap(GL_TEXTURE_2D);

	if (image2)

	{

		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, textureWidth2, textureHeight2, 0, GL_RGB, GL_UNSIGNED_BYTE, image2);

		glGenerateMipmap(GL_TEXTURE_2D);

	}

	else

	{

		std::cout << "Failed to load texture" << std::endl;

	}

	stbi_image_free(image2);


	glBindTexture(GL_TEXTURE_2D, 0);
```

En donde lo que se puede destacar es la carga de la textura del mazo y se colocan

```
//Set textures units
	lightingShader.Use();
	glUniform1i(glGetUniformLocation(lightingShader.Program, "text3"),2);
	glUniform1i(glGetUniformLocation(lightingShader.Program, "text4"), 3);
```

Y para poder modelarlo a partir de un cubo:

```
	//Cuerpo del mazo
	model = glm::mat4(1.0f);
	model = glm::scale(model, glm::vec3(8.0f, 5.0, 6.0));
	model = glm::translate(model, glm::vec3(7.5f, 4.0, 0.0f));
	glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
	glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
//Mango del mazo
model = glm::mat4(1.0f);
model = glm::scale(model, glm::vec3(2.0f, 2.0, 8.0));
model = glm::translate(model, glm::vec3(30.0f, 10.0, 0.8f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);

//Bind difuuse map
glBindTexture(GL_TEXTURE_2D, texture3);

//Bind specular map
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture4);

glBindVertexArray(VAO);

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 6);
```

En donde se puede ver el escalamiento de los mismos y su traslado, así como la carga de las texturas declaradas arriba

Referencias y créditos:

El código base utilizado para el proyecto fue proporcionado por el Ing. Carlos Aldair Roman Balbuena

Los modelos fueron obtenidos en *turbosquid.com*

Modelos realizados por:

- Escaleras - Por Fworx
- Nariz de la máscara - Por tarcisiothiago
- Dientes de la máscara - Por SpinQuad1976
- Camión de Justo - Por ERLHN
- Coraje - Por mnphmnmn
- Molino - Por companion_3d
- Lámpara - Por Designconnected
- TV - Por MartinDiavolo
- Mecedora - Por anilcat
- Sillón - Por Designconnected

Los modelos restantes fueron realizados por Josué E. Medina Cruz y y Derek Cortez Ibarra a través de Maya 2018