

Proyecto Final

Manual técnico.....	2
Objetivos general.....	2
Alcances.....	2
Funciones y variables.....	2
Diagramas de flujo.....	12
Diagrama de Gantt.....	15
Elementos del proyecto.....	15
Análisis de costos.....	18
Conclusiones.....	19
Technical Manual.....	20
General Objective.....	20
Scope.....	20
Functions and Variables.....	21
Flowcharts.....	31
Gantt Diagram.....	33
Project Elements.....	33
Cost Analysis.....	37
Conclusions.....	37

Manual técnico

Objetivos general

Se tiene como objetivo general recrear uno de los espacios más significativos dentro de las infancias mexicanas, una feria de pueblo con diversas atracciones, además de que cuente con elementos del espacio como baños o árboles. Todo lo anterior utilizando herramientas como 3Ds Max para el modelado y texturizado de los objetos y OpenGL para la animación y el acomodo de los mismos.

Alcances

1. Modelado y diseño de entorno

Se desarrollará un entorno tridimensional que represente una feria, incorporando al menos cinco elementos clave como juegos de destreza, puesto de micheladas, caminos y estructuras decorativas, los cuales serán modelados utilizando herramientas como 3ds Max.

2. Importación y renderizado de modelos en OpenGL

Los modelos creados externamente serán exportados e integrados en la escena utilizando OpenGL, aplicando técnicas de carga de geometría, texturizado y transformación para su correcta visualización en tiempo real.

3. Implementación de cámara en primera o tercera persona

Se integrará un sistema de cámara que permita al usuario recorrer la feria utilizando teclas de dirección (WASD) y/o mouse, ofreciendo libertad de movimiento e inmersión en el entorno virtual.

4. Animación de objetos y personajes

Se animarán cuatro elementos dentro del entorno, simulando movimiento automático o controlado por el usuario mediante la interacción con teclas.

5. Simulación visual con iluminación básica

Se aplicarán técnicas de iluminación para simular condiciones de luz diurna o nocturna dentro del entorno, utilizando fuentes de luz ambiental o direccional para mejorar el realismo visual del proyecto.

Funciones y variables

Dentro de la organización del proyecto se tienen varias funciones y variables importantes. Empezando por las variables de los códigos window.cpp y window.h , a continuación se agrega una captura del código donde se declaran las variables, seguida de una tabla donde se explica el tipo de variable y el uso en particular para cada una de ellas.

```

        ~Window();
private:
    GLFWwindow *mainWindow;
    GLint width, height;
    GLfloat rotax, rotay, rotaz, articulacion1, articulacion2, articulacion3, articulacion4;
    int direccionPiernaIzq = 1;
    int direccionPiernaDer = -1;
    bool keys[1024];
    bool avanzaBomba;
    GLint bufferWidth, bufferHeight;
    void createCallbacks();
    GLfloat lastX;
    GLfloat lastY;
    GLfloat xChange;
    GLfloat yChange;
    GLfloat mueveLink, mueveLinkz, rotacionPersonajeY;
    bool mouseFirstMoved;
    static void ManejaTeclado(GLFWwindow* window, int key, int code, int action, int mode);
    static void ManejaMouse(GLFWwindow* window, double xPos, double yPos);
};


```

Variable	Tipo	Descripción
mainWindow	GLFWwindow *	Puntero a la ventana de GLFW, necesaria para contexto de OpenGL.
width, height	GLInt	Tamaño de la ventana.
bufferWidth, bufferHeight	GLInt	Tamaño real del framebuffer (usado en viewport).
keys[1024]	bool	Arreglo para rastrear el estado de cada tecla del teclado.
lastX, lastY, xChange, yChange	GLfloat	Variables para el manejo del mouse y el movimiento de cámara.
mueveLink, mueveLinkz	GLfloat	Posición del personaje (ej. X/Z) dentro de la escena.

<code>rotacionPersonajeY</code>	<code>GLfloat</code>	Ángulo de rotación del personaje para simular que gira.
<code>articulacion1,</code> <code>articulacion2,</code> <code>articulacion3, articulacion4</code>	<code>GLfloat</code>	Control de articulaciones del personaje (piernas, brazos, etc).
<code>direccionPiernaIzq,</code> <code>direccionPiernaDer</code>	<code>int</code>	Dirección en que se mueven las piernas (1 o -1).
<code>avanzaBomba</code>	<code>bool</code>	Indica si una bomba (objeto) está activa o en movimiento.
<code>brazoOscilando,</code> <code>articulacionBrazo,</code> <code>direccionBrazo</code>	<code>bool / float / int</code>	Control del movimiento oscilante del brazo (como saludar o atacar).
<code>mouseFirstMoved</code>	<code>bool</code>	Bandera para inicializar correctamente el movimiento del mouse.

A continuación se muestran las funciones y el uso designado para cada una de ellas.

- *Initialise()*: Inicializa GLFW, GLEW crea la ventana y el contexto en OpenGL.

```

int Window::Initialise()
{
    //Inicialización de GLFW
    if (!glfwInit())
    {
        printf("Falló inicializar GLFW");
        glfwTerminate();
        return 1;
    }
    //Asignando variables de GLFW y propiedades de ventana
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    //para solo usar el core profile de OpenGL y no tener retrocompatibilidad
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);

    //CREAR VENTANA
    mainWindow = glfwCreateWindow(width, height, "Proyecto Final CGeIHC: Feria", NULL, NULL);
}

```

- `createCallbacks():` Registra los callbacks del teclado y mouse con GLFW.

```

    < void Window::createCallbacks()
    {
        glfwSetKeyCallback(mainWindow, ManejaTeclado);
        glfwSetCursorPosCallback(mainWindow, ManejaMouse);
    }
}

```

- `ManejaTeclado():` Callback para manejar entradas de teclado, incluyendo movimiento del personaje, animaciones, y control de articulaciones. Dentro de esta función se establecen las teclas que son responsables de mover al personaje de link al mismo tiempo de cambiar la rotación de las misma

```

    < void Window::ManejaTeclado(GLFWwindow* window, int key, int code, int action, int mode)
    {
        Window* theWindow = static_cast<Window*>(glfwGetWindowUserPointer(window));

        if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        {
            glfwSetWindowShouldClose(window, GL_TRUE);
        }

        if (key == GLFW_KEY_Y)
        {
            theWindow->mueveLink += 1.0;
            theWindow->rotacionPersonajeY = 0.0f;
            // Pierna 1 (izquierda)
            theWindow->articulacion1 += 5.0f * theWindow->direccionPiernaIzq;
            if (theWindow->articulacion1 >= 45.0f)
                theWindow->direccionPiernaIzq = -1;
            else if (theWindow->articulacion1 <= -45.0f)
                theWindow->direccionPiernaIzq = 1;

            // Pierna 2 (derecha)
            theWindow->articulacion2 += 5.0f * theWindow->direccionPiernaDer;
            if (theWindow->articulacion2 >= 45.0f)
                theWindow->direccionPiernaDer = -1;
            else if (theWindow->articulacion2 <= -45.0f)
                theWindow->direccionPiernaDer = 1;
        }
    }
}

```

```

if (key == GLFW_KEY_U)
{
    theWindow->mueveLink -= 1.0;
    theWindow->rotacionPersonajeY = 180.0f;
    theWindow->articulacion1 += 5.0f * theWindow->direccionPiernaIzq;
    if (theWindow->articulacion1 >= 45.0f)
        theWindow->direccionPiernaIzq = -1;
    else if (theWindow->articulacion1 <= -45.0f)
        theWindow->direccionPiernaIzq = 1;

    // Pierna 2 (derecha)
    theWindow->articulacion2 += 5.0f * theWindow->direccionPiernaDer;
    if (theWindow->articulacion2 >= 45.0f)
        theWindow->direccionPiernaDer = -1;
    else if (theWindow->articulacion2 <= -45.0f)
        theWindow->direccionPiernaDer = 1;

}
if (key == GLFW_KEY_J)
{
    theWindow->mueveLinkz += 1.0;
    theWindow->rotacionPersonajeY = -90.0f;
    // Pierna 1 (izquierda)
    theWindow->articulacion1 += 5.0f * theWindow->direccionPiernaIzq;
    if (theWindow->articulacion1 >= 45.0f)
        theWindow->direccionPiernaIzq = -1;
    else if (theWindow->articulacion1 <= -45.0f)
        theWindow->direccionPiernaIzq = 1;

    // Pierna 2 (derecha)
    theWindow->articulacion2 += 5.0f * theWindow->direccionPiernaDer;
    if (theWindow->articulacion2 >= 45.0f)
        theWindow->direccionPiernaDer = -1;
    else if (theWindow->articulacion2 <= -45.0f)
        theWindow->direccionPiernaDer = 1;
}

if (key == GLFW_KEY_H)
{
    theWindow->mueveLinkz -= 1.0;
    theWindow->rotacionPersonajeY = 90.0f;
    theWindow->articulacion1 += 5.0f * theWindow->direccionPiernaIzq;
    if (theWindow->articulacion1 >= 45.0f)
        theWindow->direccionPiernaIzq = -1;
    else if (theWindow->articulacion1 <= -45.0f)
        theWindow->direccionPiernaIzq = 1;

    // Pierna 2 (derecha)
    theWindow->articulacion2 += 5.0f * theWindow->direccionPiernaDer;
    if (theWindow->articulacion2 >= 45.0f)
        theWindow->direccionPiernaDer = -1;
    else if (theWindow->articulacion2 <= -45.0f)
        theWindow->direccionPiernaDer = 1;
}

if (key == GLFW_KEY_O)
{
    theWindow->articulacion3 += 5.0f * theWindow->direccionBrazo;
    if (theWindow->articulacion3 >= 0.0f)
        theWindow->direccionBrazo = -1;
    else if (theWindow->articulacion3 <= -90.0f)
        theWindow->direccionBrazo = 1;
}

```

Además encargarse de mover al personaje de Link también se ocupa una tecla para simular que el personaje de Clarence está “golpeando” a los topos

- *ManejaMouse():*Callback para detectar movimiento del mouse (control de cámara).

```

    v void Window::ManejaMouse(GLFWwindow* window, double xPos, double yPos)
    {
        Window* theWindow = static_cast<Window*>(glfwGetWindowUserPointer(window));

        if (theWindow->mouseFirstMoved)
        {
            theWindow->lastX = xPos;
            theWindow->lastY = yPos;
            theWindow->mouseFirstMoved = false;
        }

        theWindow->xChange = xPos - theWindow->lastX;
        theWindow->yChange = theWindow->lastY - yPos;

        theWindow->lastX = xPos;
        theWindow->lastY = yPos;
    }
}

```

- *getXChange()* y *getYChange()*: Devuelven el cambio en el movimiento del mouse desde el último frame.

```

    v GLfloat Window::getXChange()
    {
        GLfloat theChange = xChange;
        xChange = 0.0f;
        return theChange;
    }

    v GLfloat Window::getYChange()
    {
        GLfloat theChange = yChange;
        yChange = 0.0f;
        return theChange;
    }

```

Dentro del main se tienen las siguientes variables:

```

float angulovaria = 0.0f;
bool cambio;
bool avanzaSumo;
bool avanza;
bool avanzaTopo = false;
float movTopo;
float movTopoo;
float rotaSumo;
float movSumo;
int aux;

Window mainWindow;
std::vector<Mesh*> meshList;
std::vector<Shader> shaderList;

```

Variable	Tipo	Descripción
mainWindow	Window	Clase principal que gestiona la ventana y entrada del usuario
Camera	Camera	Maneja la posición, dirección y controles de cámara (WASD, mose)
shaderList	std::vector<Shader>	Lista de shaders usados para renderizar con iluminación.
meshList	std::vector<Mesh*>	Lista de mallas utilizadas en el entorno.
movSumo	float	Controla la posición del personaje Sumo
rotaSumo	float	Angulo para poder rotar a Sumo
movTopo, movTopoo	float	Controla el movimiento de los topos
avanzaSumo, avanzaTopo	bool	Flags que controlan si los personajes avanzan o retroceden.
deltaTime	float	Control del tiempo entre frames para animaciones suaves.
letraUVs	std::map<char, glm::vec4>	Mapa para acceder a coordenadas UV de letras (texto 3D).

Dentro de las funciones del archivo main estan las siguientes funciones:

- *CreateObjects*: Con esta función se crean las mallas base del entorno.

```

    void CreateObjects()
    {
        unsigned int floorIndices[] = {
            0, 2, 1,
            1, 2, 3
        };
        //Piso que mide 20 unidades
        GLfloat floorVertices[] = {
            -10.0f, 0.0f, -10.0f,    0.0f, 0.0f,      0.0f, -1.0f, 0.0f,
            10.0f, 0.0f, -10.0f,    1.0f, 0.0f, 0.0f, -1.0f, 0.0f,
            -10.0f, 0.0f, 10.0f,    0.0f, 1.0f, 0.0f, -1.0f, 0.0f,
            10.0f, 0.0f, 10.0f,    1.0f, 1.0f, 0.0f, -1.0f, 0.0f
        };

        Mesh* obj1 = new Mesh();
        obj1->CreateMesh(floorVertices, floorIndices, 32, 6);
        meshList.push_back(obj1);

        //agarra solo una region letra
        unsigned int letraIndices[] = {
            0, 1, 2,
            0, 2, 3,
        };

        GLfloat letraVertices[] = {
            -0.5f, 0.0f, 0.5f,      0.0f, 0.00f,      0.0f, -1.0f, 0.0f,
            0.5f, 0.0f, 0.5f,      0.11f, 0.00f,      0.0f, -1.0f, 0.0f,
            0.5f, 0.0f, -0.5f,     0.11f, 0.33f,      0.0f, -1.0f, 0.0f,
            -0.5f, 0.0f, -0.5f,    0.0f, 0.33f,      0.0f, -1.0f, 0.0f,
        };

        Mesh* obj2 = new Mesh();
        obj2->CreateMesh(letraVertices, letraIndices, 32, 6);
        meshList.push_back(obj2);
    }
}

```

- *RendeRextoEstatico*: Con ayuda de esta función se dibuja texto letra por letra en la escena con transformaciones.

```

void RenderTextoEstatico(const std::string& texto, glm::vec3 posicionInicial,
float separacion, GLuint uniformModel, GLuint uniformTextureOffset) {
    for (size_t i = 0; i < texto.size(); ++i) {
        char letra = texto[i];

        // Verifica si la letra está en el mapa de UVs
        if (letraUVs.find(letra) == letraUVs.end()) continue;

        glm::vec4 uv = letraUVs[letra]; // Coordenadas UV de la letra
        glm::vec2 toffset = glm::vec2(uv.x, uv.y);

        // Calcula la posición de la letra
        glm::vec3 posicionLetra = posicionInicial + glm::vec3(0.0f, 0.0f, i * separacion);

        // Renderiza la letra
        glm::mat4 model = glm::mat4(1.0f);
        model = glm::translate(model, posicionLetra);
        model = glm::rotate(model, 90.0f * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación al plano XY
        model = glm::rotate(model, 90.0f * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación al plano XY
        model = glm::scale(model, glm::vec3(1.4f, 4.5f, 4.5f)); // Aplica la escala
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
        glUniform2fv(uniformTextureOffset, 1, glm::value_ptr(toffset));
        LetrasTexture.UseTexture();
        meshList[1]->RenderMesh(); // Usa un plano para renderizar la letra
    }
}

```

- *main*: Dentro de esta función se cargaron todos los modelos que se utilizaron, esto se hace accediendo a las respectivas carpetas donde se encuentra cada uno de ellos.

```
//Clarence
ClarenceC_M = Model();
ClarenceC_M.LoadModel("Models/Clarence/cuerpoClarence.obj");
ClarencePD_M = Model();
ClarencePD_M.LoadModel("Models/Clarence/piernaDerClarence.obj");
ClarencePI_M = Model();
ClarencePI_M.LoadModel("Models/Clarence/piernaIzqClarence.obj");
ClarenceBD_M = Model();
ClarenceBD_M.LoadModel("Models/Clarence/brazoDerClarence.obj");
ClarenceBI_M = Model();
ClarenceBI_M.LoadModel("Models/Clarence/brazoIzqClarence.obj");
//Jeff
JeffC_M = Model();
JeffC_M.LoadModel("Models/jeff/source/jeffcuerpo.obj");
JeffPD_M = Model();
JeffPD_M.LoadModel("Models/jeff/source/jeffpiernaDer.obj");
JeffPI_M = Model();
JeffPI_M.LoadModel("Models/jeff/source/jeffpiernaIzq.obj");
JeffBD_M = Model();
JeffBD_M.LoadModel("Models/jeff/source/jeffbrazoDer.obj");
JeffBI_M = Model();
JeffBI_M.LoadModel("Models/jeff/source/jeffbrazoIzq.obj");
```

Además se cargaron las texturas para el skybox junto con el material de nuestro protagonista y la luz principal del ambiente.

```
std::vector<std::string> skyboxFaces;
skyboxFaces.push_back("Textures/Skybox/cube_right.png");
skyboxFaces.push_back("Textures/Skybox/cube_left.png");
skyboxFaces.push_back("Textures/Skybox/cube_down.png");
skyboxFaces.push_back("Textures/Skybox/cube_up.png");
skyboxFaces.push_back("Textures/Skybox/cube_back.png");
skyboxFaces.push_back("Textures/Skybox/cube_front.png");

skybox = Skybox(skyboxFaces);

Material_link = Material(0.0f, 1);

//luz direccional, sólo 1 y siempre debe de existir
mainLight = DirectionalLight(1.0f, 1.0f, 1.0f,
    0.3f, 0.3f,
    0.0f, -1.0f, 0.0f);
//contador de luces puntuales
```

Dentro de este main también se tiene un loop el cual se ejecuta mientras que la ventana esté abierta, dentro de este loop se utiliza deltaTime para poder manejar las animaciones de los distintos personajes. Dentro de este proyecto se decidió darle movimiento al personaje

de Sumo de manera automática, como se visualiza a continuación:

```
if (avanzaSumo) {
    if (movSumo< 100.0f)
    {
        movSumo += movOffset * deltaTime;
    }
    else {
        avanzaSumo = false;
        rotaSumo = 180.0f;
    }
}
else {
    if (movSumo > -1.0f)
    {
        movSumo -= movOffset * deltaTime;
    }
    else {
        avanzaSumo = true;
        rotaSumo = 0.0f;
    }
}
```

El personaje de sumo se acomodó de manera jerárquica por lo que simplemente se utiliza la variable “movSumo” para trasladarlo

```
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(30.0f, 0.90f, 0.0f + (movSumo)));
model = glm::rotate(model, glm::radians(rotaSumo), glm::vec3(0.0f, 1.0f, 0.0f));
//model = glm::translate(model, glm::vec3(10.0f, 0.90f, -92.0f));
model = glm::scale(model, glm::vec3(20.05f, 20.05f, 20.05f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SumoC_M.RenderModel();

modelaux = model;
model = glm::translate(model, glm::vec3(-0.014f, 0.022f, -0.0f));
model = glm::rotate(model, 15.0f * sin(angulovaria * 0.1f) * toRadians, glm::vec3(-1.0f, 0.0f, -0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SumoPD_M.RenderModel();
model = modelaux;
modelaux = model;
model = glm::translate(model, glm::vec3(0.02f, 0.022f, -0.0f));
model = glm::rotate(model, 15.0f * sin(angulovaria * 0.1f + glm::pi<float>()) * toRadians, glm::vec3(-1.0f, 0.0f, -0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SumoPI_M.RenderModel();
model = modelaux;
modelaux = model;
model = glm::translate(model, glm::vec3(0.015f, 0.106f, 0.001f));
model = glm::rotate(model, 15.0f * sin(angulovaria * 0.1f) * toRadians, glm::vec3(-1.0f, 0.0f, -0.0f));
model = glm::rotate(model, -45 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SumoBI_M.RenderModel();
```

Se realizó algo similar para el movimiento de los topos

```
// avanzaTopo {
//     if (movTopo < 0.0f)
//     {
//         movTopo += movOffset * deltaTime;
//     }
//     else {
//         avanzaTopo = false;
//     }
//     if (movTopoo > -0.30f)
//     {
//         movTopoo -= movOffset * deltaTime;
//     }

// se {
//     if (movTopo > -0.30f)
//     {
//         movTopo -= movOffset * deltaTime;
//     }
//     else {
//         avanzaTopo = true;
//     }
//     if (movTopoo < 0.0f)
//     {
//         movTopoo += movOffset * deltaTime;
//     }
// }
```

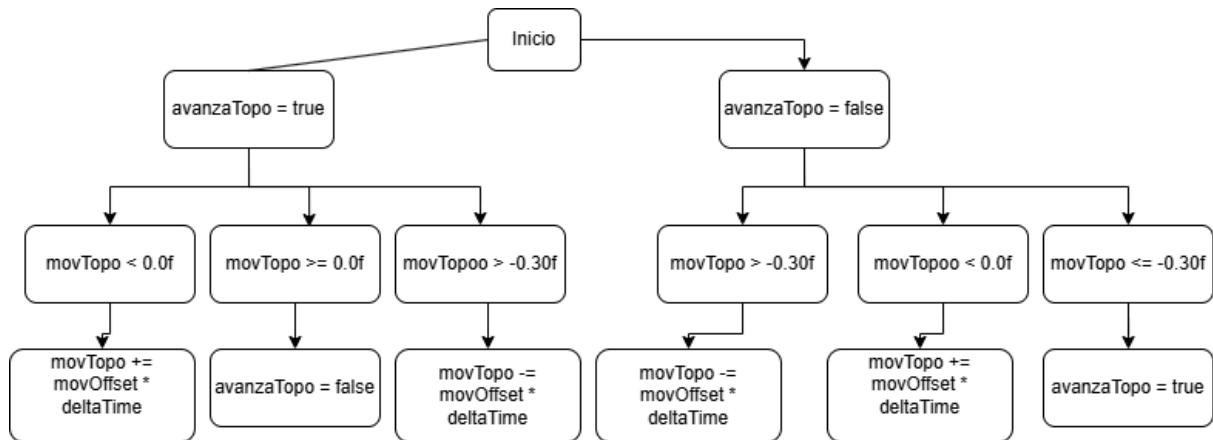
A la hora de mandarlos a llamar solo fue cuestión de asignar una u otra variable a cada uno de los topos.

Las últimas dos animaciones utilizan variables globales las cuales son modificadas dentro del archivo de windows.cpp mediante entradas por parte del usuario con el teclado, para el caso del protagonista, al tener movimiento en ambos ejes, se declaró la variable “muestraLink” para mover al personaje en x y “muestraLinkz” para moverlo en el eje z.

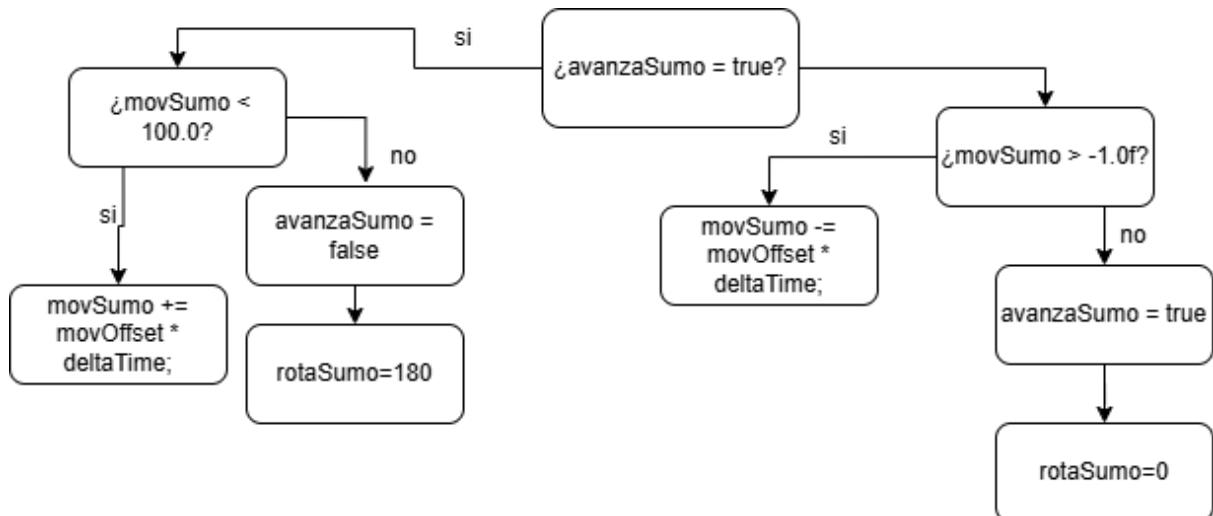
```
//Cuerpo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(13.0f, 3.25f, -22.0f));
model = glm::translate(model, glm::vec3(-0.0f + mainWindow.getmuestraLink(), 2.0f, -3.0f));
model = glm::translate(model, glm::vec3(-0.0f, 0.0f, 0.0f + mainWindow.getmuestraLinkz()));
//model = glm::translate(model, glm::vec3(0.0f, 3.85f, 0.0f));
model = glm::scale(model, glm::vec3(2.2f, 2.2f, 2.2f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotacionPersonajeY()), glm::vec3(0.0f, 1.0f, 0.0f));
posInicial = model;
```

Diagramas de flujo

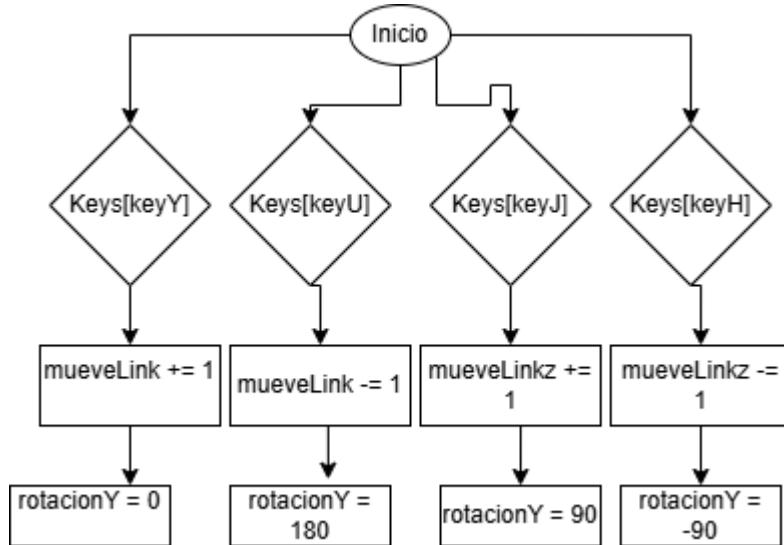
A continuación se muestra el diagrama de flujo de la animación de los topos:



Animación del personaje de Sumo:



Animación de Link:



Animación de Clarence:

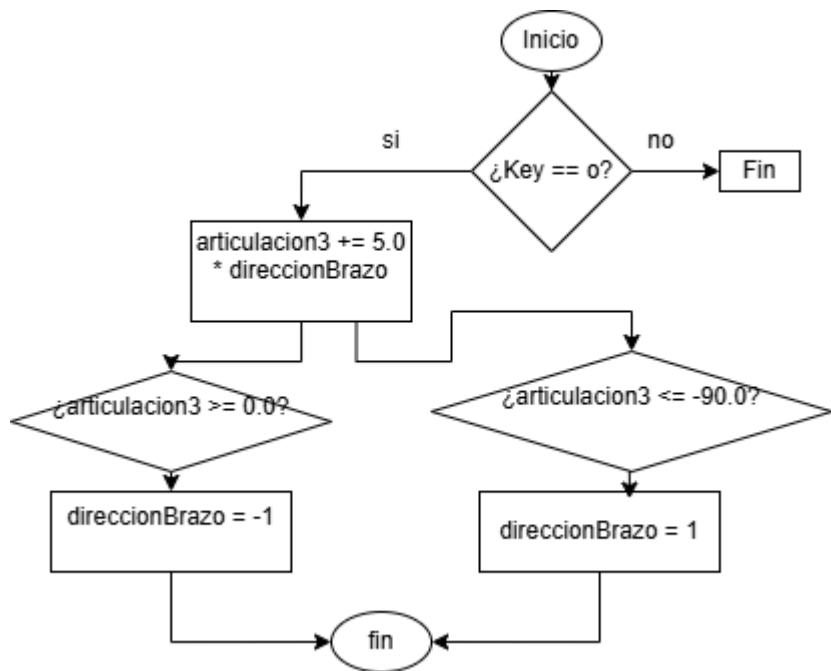


Diagrama de Gantt

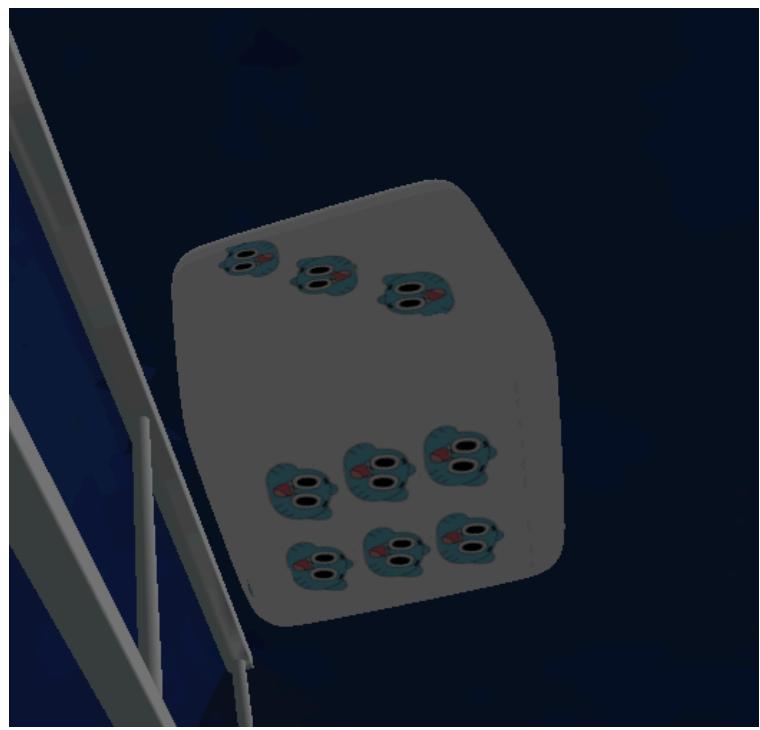
Actividad	Abril Semana 1	Abril Semana 2	Abril Semana 3	Abril Semana 4	Mayo Semana 1	Mayo Semana 2	Mayo Semana 3
Investigación y planificación	[Timeline Bar]						
Modelado de objetos en 3ds Max		[Timeline Bar]	[Timeline Bar]				
Animación 1			[Timeline Bar]	[Timeline Bar]			
Animación 2				[Timeline Bar]	[Timeline Bar]		
Animación 3					[Timeline Bar]	[Timeline Bar]	
Animación 4						[Timeline Bar]	[Timeline Bar]
Manual técnico				[Timeline Bar]	[Timeline Bar]		
Manual de usuario					[Timeline Bar]	[Timeline Bar]	
Manual de costos						[Timeline Bar]	[Timeline Bar]
Revisión y corrección final							[Timeline Bar]

Elementos del proyecto

Los elementos creados para este proyecto son los siguientes:



Recreación de juego de dados con temática de El increíble mundo de Gumball



Dado recreado con temática de “El increíble mundo de Gumball”



Golpea al Topo con temática de “Clarence”



Globos con dardos



Elementos del espacio



Fachada de la feria

Análisis de costos

Concepto	Descripción	Cantidad	Costo Unitario (MXN)	Costo Total (MXN)
Software 3Ds Max	Licencia para	1	\$4,000	\$4,000
Hardware	Computadora para diseño y renderizado	1	\$15,000	\$15,000
Horas de trabajo	Desarrollo y modelado (50 horas)	50	\$150	\$7,500
Horas de trabajo	Programación y animación (40 horas)	40	\$150	\$6,000
Otros gastos	Recursos y materiales varios			\$500
Total				\$33,000
Precio de venta		\$49,000		
Ganancia Total		\$16,000		

Conclusiones

Con este proyecto se logró integrar exitosamente herramientas de programación como los son OpenGL y modelado 3D con 3ds Max, lo cual permitió la generación de animaciones funcionales y visualmente coherentes. A lo largo del desarrollo, se fortalecieron tanto habilidades técnicas como creativas, evidenciadas en la creación de cuatro animaciones las cuales combinaron gran parte de lo aprendido a lo largo del curso.

La organización y gestión del tiempo fueron clave para lograr el cumplimiento de los objetivos, gracias a una planificación estructurada mediante un diagrama de Gantt que facilitó la distribución eficiente de tareas en las fases de modelado, animación y documentación. Asimismo, la elaboración de manuales técnico, de usuario y de costos aseguró una documentación clara y replicable del sistema desarrollado.

A pesar de los desafíos enfrentados dentro de este proyecto, como el manejo de coordenadas 3D, la sincronización de movimientos y la optimización del renderizado, estos fueron superados mediante investigación y pruebas iterativas.

Technical Manual

General Objective

The general objective is to recreate one of the most meaningful spaces in Mexican childhood: a traditional town fair with various attractions, including elements such as restrooms and trees. This will be achieved using tools like 3Ds Max for modeling and texturing objects, and OpenGL for animation and scene arrangement.

Scope

Modeling and Environment Design

A three-dimensional environment representing a fair will be developed, incorporating at least five key elements such as skill games, a michelada stand, walkways, and decorative structures. These will be modeled using tools like 3Ds Max..

Importing and Rendering Models in OpenGL

The externally created models will be exported and integrated into the scene using OpenGL, applying geometry loading, texturing, and transformation techniques for proper real-time visualization.

First-Person or Third-Person Camera Implementation

A camera system will be integrated to allow the user to explore the fair using directional keys (WASD) and/or the mouse, providing freedom of movement and immersion within the virtual environment.

Object and Character Animation

Four elements in the environment will be animated, simulating automatic movement or user-controlled actions through key interactions.

Visual Simulation with Basic Lighting

Lighting techniques will be applied to simulate daytime or nighttime conditions within the environment, using ambient or directional light sources to enhance the visual realism of the project.

Functions and Variables

Within the project's structure, several important functions and variables are used. Starting with the variables from the window.cpp and window.h files, a screenshot of the code where the variables are declared is included below, followed by a table explaining each variable's type and specific use.

```
| ~Window();  
private:  
    GLFWwindow *mainWindow;  
    GLint width, height;  
    GLfloat rotax, rotay, rotaz, articulacion1, articulacion2, articulacion3, articulacion4;  
    int direccionPiernaIzq = 1;  
    int direccionPiernaDer = -1;  
    bool keys[1024];  
    bool avanzaBomba;  
    GLint bufferWidth, bufferHeight;  
    void createCallbacks();  
    GLfloat lastX;  
    GLfloat lastY;  
    GLfloat xChange;  
    GLfloat yChange;  
    GLfloat mueveLink, mueveLinkz, rotacionPersonajeY;  
    bool mouseFirstMoved;  
    static void ManejaTeclado(GLFWwindow* window, int key, int code, int action, int mode);  
    static void ManejaMouse(GLFWwindow* window, double xPos, double yPos);  
};
```

Variable	Type	Description
mainWindow	GLFWwindow*	Pointer to the GLFW window, required for the OpenGL context.
width, height	GLInt	Window size.
bufferWidth, bufferHeight	GLInt	Actual framebuffer size (used in the viewport).
keys[1024]	bool	Array to track the state of each keyboard key.

<code>lastX, lastY, xChange, yChange</code>	<code>GLfloat</code>	Variables for mouse handling and camera movement.
<code>mueveLink, mueveLinkz</code>	<code>GLfloat</code>	Character's position (e.g., X/Z) within the scene.
<code>rotacionPersonajeY</code>	<code>GLfloat</code>	Character's rotation angle to simulate turning.
<code>articulacion1, articulacion2, articulacion3, articulacion4</code>	<code>GLfloat</code>	Control of character joints (legs, arms, etc.).
<code>direccionPiernaIzq, direccionPiernaDer</code>	<code>int</code>	Direction in which the legs move (1 or -1).
<code>avanzaBomba</code>	<code>bool</code>	Indicates whether a bomb (object) is active or in motion.
<code>brazoOscilando, articulacionBrazo, direccionBrazo</code>	<code>bool / float / int</code>	Controls for arm oscillation movement (like waving or attacking).
<code>mouseFirstMoved</code>	<code>bool</code>	Flag to correctly initialize mouse movement.

The following are the functions and their designated uses:

- *Initialise():Initializes GLFW and GLEW, and creates the window and OpenGL context.*

```

int Window::Initialise()
{
    //Inicialización de GLFW
    if (!glfwInit())
    {
        printf("Falló inicializar GLFW");
        glfwTerminate();
        return 1;
    }
    //Asignando variables de GLFW y propiedades de ventana
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    //para solo usar el core profile de OpenGL y no tener retrocompatibilidad
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);

    //CREAR VENTANA
    mainWindow = glfwCreateWindow(width, height, "Proyecto Final CGeIHC: Feria", NULL, NULL);
}

```

- *createCallbacks()*: Registers keyboard and mouse callbacks with GLFW.

```

void Window::createCallbacks()
{
    glfwSetKeyCallback(mainWindow, ManejaTeclado);
    glfwSetCursorPosCallback(mainWindow, ManejaMouse);
}

```

- *ManejaTeclado()*: Callback to handle keyboard input, including character movement, animations, and joint control. This function defines the keys responsible for moving the Link character while also updating its rotation accordingly.

```

void Window::ManejaTeclado(GLFWwindow* window, int key, int code, int action, int mode)
{
    Window* theWindow = static_cast<Window*>(glfwGetWindowUserPointer(window));

    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }
    if (key == GLFW_KEY_Y)
    {
        theWindow->mueveLink += 1.0;
        theWindow->rotacionPersonajeY = 0.0f;
        // Pierna 1 (izquierda)
        theWindow->articulacion1 += 5.0f * theWindow->direccionPiernaIzq;
        if (theWindow->articulacion1 >= 45.0f)
            theWindow->direccionPiernaIzq = -1;
        else if (theWindow->articulacion1 <= -45.0f)
            theWindow->direccionPiernaIzq = 1;

        // Pierna 2 (derecha)
        theWindow->articulacion2 += 5.0f * theWindow->direccionPiernaDer;
        if (theWindow->articulacion2 >= 45.0f)
            theWindow->direccionPiernaDer = -1;
        else if (theWindow->articulacion2 <= -45.0f)
            theWindow->direccionPiernaDer = 1;
    }
}

```

```

if (key == GLFW_KEY_U)
{
    theWindow->mueveLink -= 1.0;
    theWindow->rotacionPersonajeY = 180.0f;
    theWindow->articulacion1 += 5.0f * theWindow->direccionPiernaIzq;
    if (theWindow->articulacion1 >= 45.0f)
        theWindow->direccionPiernaIzq = -1;
    else if (theWindow->articulacion1 <= -45.0f)
        theWindow->direccionPiernaIzq = 1;

    // Pierna 2 (derecha)
    theWindow->articulacion2 += 5.0f * theWindow->direccionPiernaDer;
    if (theWindow->articulacion2 >= 45.0f)
        theWindow->direccionPiernaDer = -1;
    else if (theWindow->articulacion2 <= -45.0f)
        theWindow->direccionPiernaDer = 1;

}
if (key == GLFW_KEY_J)
{
    theWindow->mueveLinkz += 1.0;
    theWindow->rotacionPersonajeY = -90.0f;
    // Pierna 1 (izquierda)
    theWindow->articulacion1 += 5.0f * theWindow->direccionPiernaIzq;
    if (theWindow->articulacion1 >= 45.0f)
        theWindow->direccionPiernaIzq = -1;
    else if (theWindow->articulacion1 <= -45.0f)
        theWindow->direccionPiernaIzq = 1;

    // Pierna 2 (derecha)
    theWindow->articulacion2 += 5.0f * theWindow->direccionPiernaDer;
    if (theWindow->articulacion2 >= 45.0f)
        theWindow->direccionPiernaDer = -1;
    else if (theWindow->articulacion2 <= -45.0f)
        theWindow->direccionPiernaDer = 1;
}

if (key == GLFW_KEY_H)
{
    theWindow->mueveLinkz -= 1.0;
    theWindow->rotacionPersonajeY = 90.0f;
    theWindow->articulacion1 += 5.0f * theWindow->direccionPiernaIzq;
    if (theWindow->articulacion1 >= 45.0f)
        theWindow->direccionPiernaIzq = -1;
    else if (theWindow->articulacion1 <= -45.0f)
        theWindow->direccionPiernaIzq = 1;

    // Pierna 2 (derecha)
    theWindow->articulacion2 += 5.0f * theWindow->direccionPiernaDer;
    if (theWindow->articulacion2 >= 45.0f)
        theWindow->direccionPiernaDer = -1;
    else if (theWindow->articulacion2 <= -45.0f)
        theWindow->direccionPiernaDer = 1;
}

if (key == GLFW_KEY_O)
{
    theWindow->articulacion3 += 5.0f * theWindow->direccionBrazo;
    if (theWindow->articulacion3 >= 0.0f)
        theWindow->direccionBrazo = -1;
    else if (theWindow->articulacion3 <= -90.0f)
        theWindow->direccionBrazo = 1;
}

```

In addition to moving the Link character, a key is also used to simulate the Clarence character "hitting" the moles.

- *ManejaMouse():*Callback to detect mouse movement (camera control).

```

    v void Window::ManejaMouse(GLFWwindow* window, double xPos, double yPos)
    {
        Window* theWindow = static_cast<Window*>(glfwGetWindowUserPointer(window));

        if (theWindow->mouseFirstMoved)
        {
            theWindow->lastX = xPos;
            theWindow->lastY = yPos;
            theWindow->mouseFirstMoved = false;
        }

        theWindow->xChange = xPos - theWindow->lastX;
        theWindow->yChange = theWindow->lastY - yPos;

        theWindow->lastX = xPos;
        theWindow->lastY = yPos;
    }
}

```

- *getXChange()* and *getYChange()*: Return the change in mouse movement since the last frame.

```

    v GLfloat Window::getXChange()
    {
        GLfloat theChange = xChange;
        xChange = 0.0f;
        return theChange;
    }

    v GLfloat Window::getYChange()
    {
        GLfloat theChange = yChange;
        yChange = 0.0f;
        return theChange;
    }

```

Within the main function, the following variables are used:

```

float angulovaria = 0.0f;
bool cambio;
bool avanzaSumo;
bool avanza;
bool avanzaTopo = false;
float movTopo;
float movTopoo;
float rotaSumo;
float movSumo;
int aux;

Window mainWindow;
std::vector<Mesh*> meshList;
std::vector<Shader> shaderList;

```

Variable	Type	Description
<code>mainWindow</code>	<code>Window</code>	Main class that manages the window and user input.
<code>Camera</code>	<code>Camera</code>	Handles camera position, direction, and controls (WASD, mouse).
<code>shaderList</code>	<code>std::vector<Shader></code>	List of shaders used for rendering with lighting.
<code>meshList</code>	<code>std::vector<Mesh*></code>	List of meshes used in the environment.
<code>movSumo</code>	<code>float</code>	Controls the Sumo character's position.
<code>rotaSumo</code>	<code>float</code>	Angle used to rotate the Sumo character.
<code>movTopo,</code> <code>movTopoo</code>	<code>float</code>	Controls the movement of the moles.
<code>avanzaSumo,</code> <code>avanzaTopo</code>	<code>bool</code>	Flags that control whether the characters move forward or backward.
<code>deltaTime</code>	<code>float</code>	Controls the time between frames for smooth animations.
<code>letraUVs</code>	<code>std::map<char, glm::vec4></code>	Map used to access UV coordinates of letters (3D text).

Within the functions of the main file, the following are included:

- *CreateObjects*: This function creates the base meshes of the environment.

```
void CreateObjects()
{
    unsigned int floorIndices[] = {
        0, 2, 1,
        1, 2, 3
    };
    //Piso que mide 20 unidades
    GLfloat floorVertices[] = {
        -10.0f, 0.0f, -10.0f,    0.0f, 0.0f,      0.0f, -1.0f, 0.0f,
        10.0f, 0.0f, -10.0f,    1.0f, 0.0f, 0.0f, -1.0f, 0.0f,
        -10.0f, 0.0f, 10.0f,    0.0f, 1.0f, 0.0f, -1.0f, 0.0f,
        10.0f, 0.0f, 10.0f,    1.0f, 1.0f, 0.0f, -1.0f, 0.0f
    };

    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(floorVertices, floorIndices, 32, 6);
    meshList.push_back(obj1);

    //agarra solo una region letra
    unsigned int letraIndices[] = {
        0, 1, 2,
        0, 2, 3,
    };

    GLfloat letraVertices[] = {
        -0.5f, 0.0f, 0.5f,      0.0f, 0.00f,      0.0f, -1.0f, 0.0f,
        0.5f, 0.0f, 0.5f,      0.11f, 0.00f,      0.0f, -1.0f, 0.0f,
        0.5f, 0.0f, -0.5f,     0.11f, 0.33f,      0.0f, -1.0f, 0.0f,
        -0.5f, 0.0f, -0.5f,    0.0f, 0.33f,      0.0f, -1.0f, 0.0f,
    };

    Mesh* obj2 = new Mesh();
    obj2->CreateMesh(letraVertices, letraIndices, 32, 6);
    meshList.push_back(obj2);
}
```

- *RendeRextoEstatico*: This function is used to draw text letter by letter in the scene using transformations.

```

void RenderTextoEstatico(const std::string& texto, glm::vec3 posicionInicial,
    float separacion, GLuint uniformModel, GLuint uniformTextureOffset) {
    for (size_t i = 0; i < texto.size(); ++i) {
        char letra = texto[i];

        // Verifica si la letra está en el mapa de UVs
        if (LetraUVs.find(letra) == LetraUVs.end()) continue;

        glm::vec4 uv = LetraUVs[letra]; // Coordenadas UV de la letra
        glm::vec2 toffset = glm::vec2(uv.x, uv.y);

        // Calcula la posición de la letra
        glm::vec3 posicionLetra = posicionInicial + glm::vec3(0.0f, 0.0f, i * separacion);

        // Renderiza la letra
        glm::mat4 model = glm::mat4(1.0f);
        model = glm::translate(model, posicionLetra);
        model = glm::rotate(model, 90.0f * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación al plano XY
        model = glm::rotate(model, 90.0f * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación al plano XY
        model = glm::scale(model, glm::vec3(1.4f, 4.5f, 4.5f)); // Aplica la escala
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
        glUniform2fv(uniformTextureOffset, 1, glm::value_ptr(toffset));
        LetrasTexture.UseTexture();
        meshList[1]->RenderMesh(); // Usa un plano para renderizar la letra
    }
}

```

- *main*: Within this function, all the models used in the project are loaded by accessing the corresponding folders where each one is located.

```

//Clarence
ClarenceC_M = Model();
ClarenceC_M.LoadModel("Models/Clarence/cuerpoClarence.obj");
ClarencePD_M = Model();
ClarencePD_M.LoadModel("Models/Clarence/piernaDerClarence.obj");
ClarencePI_M = Model();
ClarencePI_M.LoadModel("Models/Clarence/piernaIzqClarence.obj");
ClarenceBD_M = Model();
ClarenceBD_M.LoadModel("Models/Clarence/brazoDerClarence.obj");
ClarenceBI_M = Model();
ClarenceBI_M.LoadModel("Models/Clarence/brazoIzqClarence.obj");
//Jeff
JeffC_M = Model();
JeffC_M.LoadModel("Models/jeff/source/jeffcuerpo.obj");
JeffPD_M = Model();
JeffPD_M.LoadModel("Models/jeff/source/jeffpiernaDer.obj");
JeffPI_M = Model();
JeffPI_M.LoadModel("Models/jeff/source/jeffpiernaIzq.obj");
JeffBD_M = Model();
JeffBD_M.LoadModel("Models/jeff/source/jeffbrazoDer.obj");
JeffBI_M = Model();
JeffBI_M.LoadModel("Models/jeff/source/jeffbrazoIzq.obj");

```

In addition, the textures for the skybox are loaded along with the material for the main character and the primary ambient light.

```

std::vector<std::string> skyboxFaces;
skyboxFaces.push_back("Textures/Skybox/cube_right.png");
skyboxFaces.push_back("Textures/Skybox/cube_left.png");
skyboxFaces.push_back("Textures/Skybox/cube_down.png");
skyboxFaces.push_back("Textures/Skybox/cube_up.png");
skyboxFaces.push_back("Textures/Skybox/cube_back.png");
skyboxFaces.push_back("Textures/Skybox/cube_front.png");

skybox = Skybox(skyboxFaces);

Material_link = Material(0.0f, 1);

//luz direccional, sólo 1 y siempre debe de existir
mainLight = DirectionalLight(1.0f, 1.0f, 1.0f,
    0.3f, 0.3f,
    0.0f, -1.0f, 0.0f);
//contador de luces puntuales

```

This main function also includes a loop that runs as long as the window is open. Inside this loop, deltaTime is used to control the animations of the different characters. In this project, the Sumo character was given automatic movement, as shown below:

```

if (avanzaSumo) {
    if (movSumo < 100.0f)
    {
        movSumo += movOffset * deltaTime;
    }
    else {
        avanzaSumo = false;
        rotaSumo = 180.0f;
    }
}
else {
    if (movSumo > -1.0f)
    {
        movSumo -= movOffset * deltaTime;
    }
    else {
        avanzaSumo = true;
        rotaSumo = 0.0f;
    }
}

```

The Sumo character was arranged hierarchically, so the “movSumo” variable is simply used to translate it

```
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(30.0f, 0.90f, 0.0f + (movSumo)));
model = glm::rotate(model, glm::radians(rotaSumo), glm::vec3(0.0f, 1.0f, 0.0f));
//model = glm::translate(model, glm::vec3(10.0f, 0.90f, -92.0f));
model = glm::scale(model, glm::vec3(20.05f, 20.05f, 20.05f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SumoC_M.RenderModel();

modelaux = model;
model = glm::translate(model, glm::vec3(-0.014f, 0.022f, -0.0f));
model = glm::rotate(model, 15.0f * sin(angulovaria * 0.1f) * toRadians, glm::vec3(-1.0f, 0.0f, -0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SumoPD_M.RenderModel();
model = modelaux;
modelaux = model;
model = glm::translate(model, glm::vec3(0.02f, 0.022f, -0.0f));
model = glm::rotate(model, 15.0f * sin(angulovaria * 0.1f + glm::pi<float>()) * toRadians, glm::vec3(-1.0f, 0.0f, -0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SumoPI_M.RenderModel();
model = modelaux;
modelaux = model;
model = glm::translate(model, glm::vec3(0.015f, 0.106f, 0.001f));
model = glm::rotate(model, 15.0f * sin(angulovaria * 0.1f) * toRadians, glm::vec3(-1.0f, 0.0f, -0.0f));
model = glm::rotate(model, -45 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SumoBI_M.RenderModel();
```

A similar approach was used for the movement of the moles.

```
    if (avanzaTopo) {
        if (movTopo < 0.0f)
        {
            movTopo += movOffset * deltaTime;
        }
        else {
            avanzaTopo = false;
        }
        if (movTopoo > -0.30f)
        {
            movTopoo -= movOffset * deltaTime;
        }
    }

    se {
        if (movTopo > -0.30f)
        {
            movTopo -= movOffset * deltaTime;
        }
        else {
            avanzaTopo = true;
        }
        if (movTopoo < 0.0f)
        {
            movTopoo += movOffset * deltaTime;
        }
    }
```

When calling them, it was just a matter of assigning one variable or another to each of the moles.

The last two animations use global variables, which are modified within the `window.cpp` file through user keyboard input.

In the case of the protagonist, since it has movement on both axes, the variable was declared accordingly.

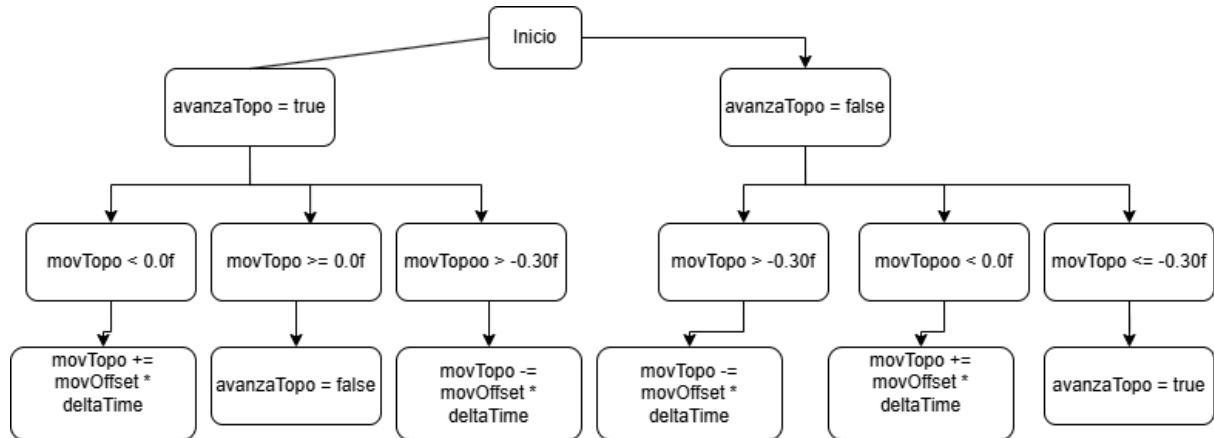
```

//Cuerpo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(13.0f, 3.25f, -22.0f));
model = glm::translate(model, glm::vec3(-0.0f + mainWindow.getmueveLink(), 2.0f, -3.0f));
model = glm::translate(model, glm::vec3(-0.0f, 0.0f, 0.0f + mainWindow.getmueveLinkz()));
//model = glm::translate(model, glm::vec3(0.0f, 3.85f, 0.0f));
model = glm::scale(model, glm::vec3(2.2f, 2.2f, 2.2f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, radians(mainWindow.getrotacionPersonajeY()), glm::vec3(0.0f, 1.0f, 0.0f));
posInicial = model;

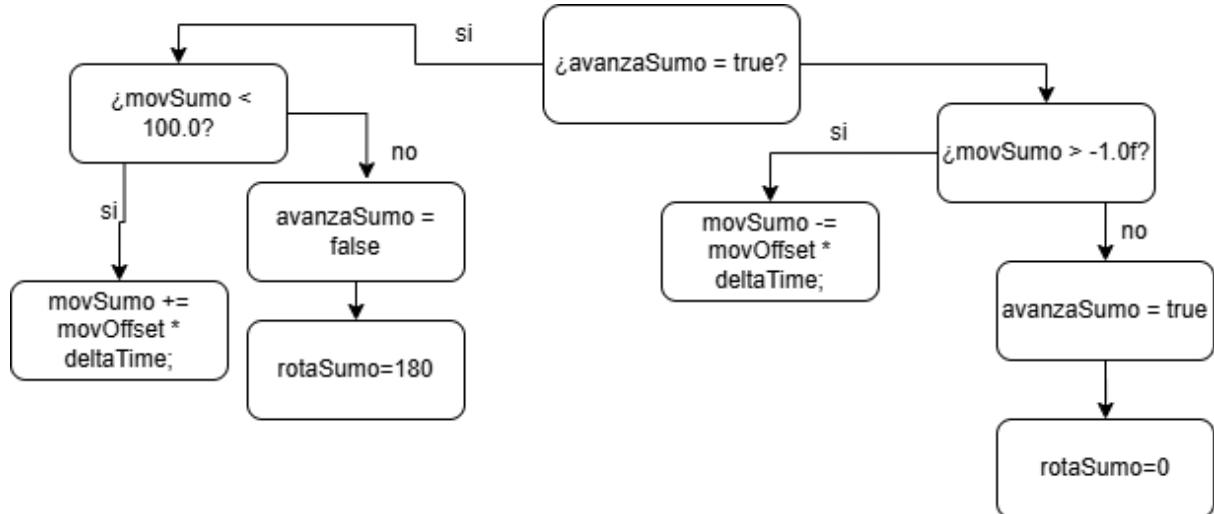
```

Flowcharts

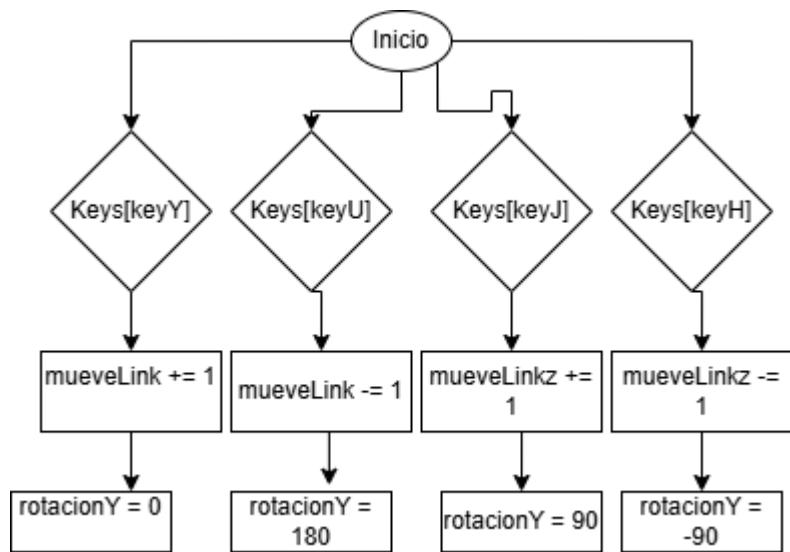
Below is the flowchart for the mole animation:



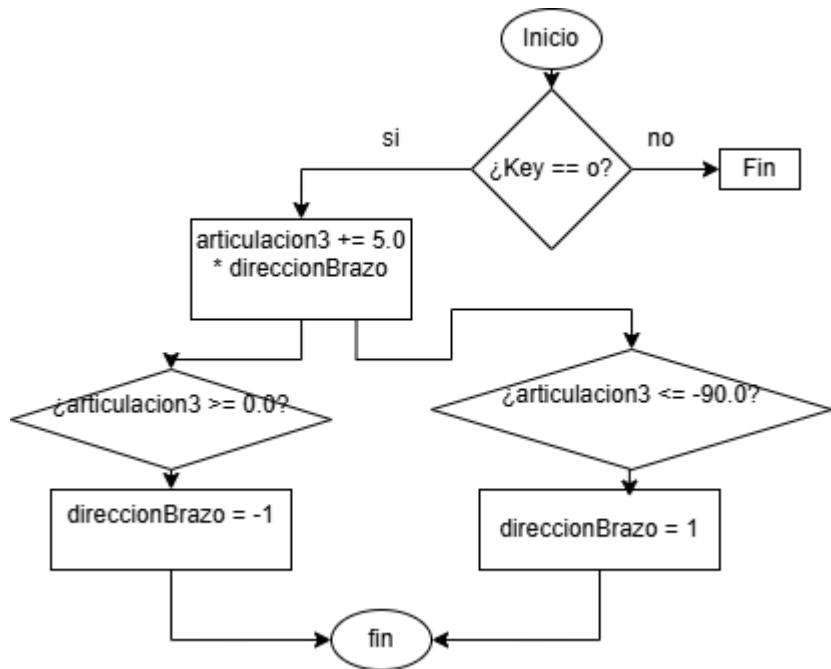
Sumo Character flowchart:



Link flowchart:



Clarence flowchart:



Gantt Diagram

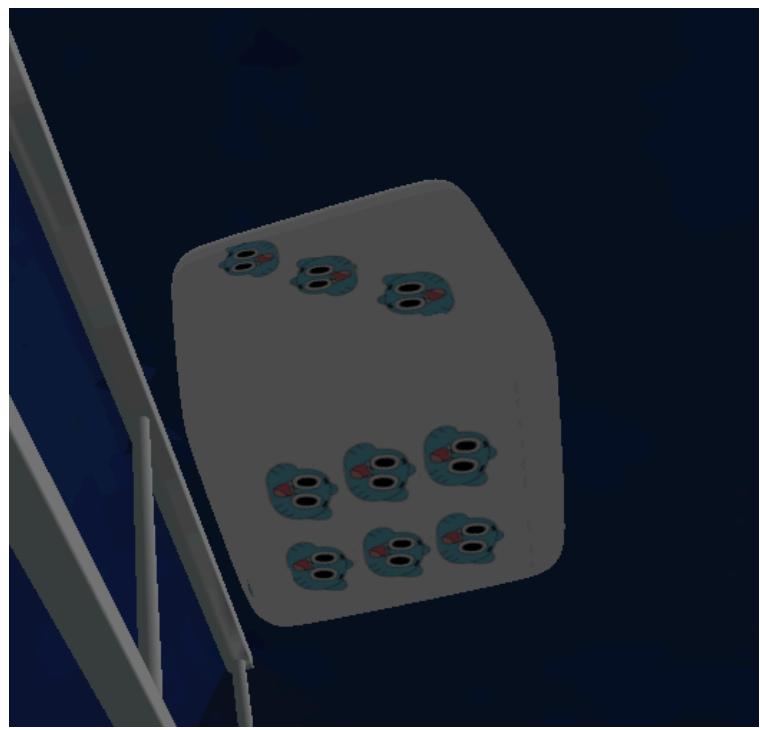
Actividad	Abril Semana 1	Abril Semana 2	Abril Semana 3	Abril Semana 4	Mayo Semana 1	Mayo Semana 2	Mayo Semana 3
Investigación y planificación							
Modelado de objetos en 3ds Max							
Animación 1							
Animación 2							
Animación 3							
Animación 4							
Manual técnico							
Manual de usuario							
Manual de costos							
Revisión y corrección final							

Project Elements

The elements created for this project are as follows::



Recreation of a dice game themed around *The Amazing World of Gumball*.



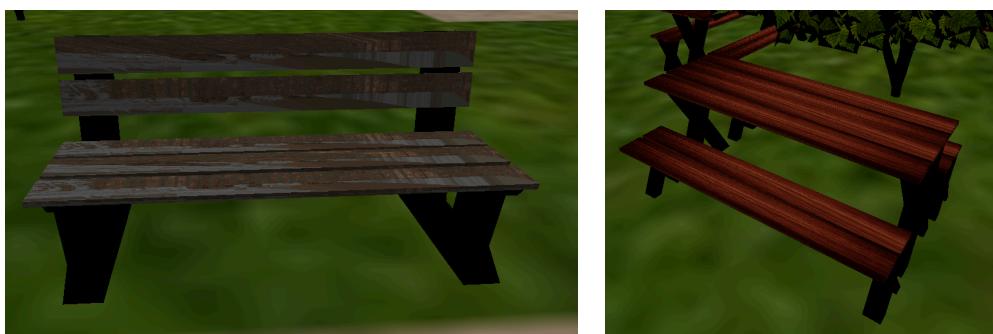
Dice recreated with a *The Amazing World of Gumball* theme.



“Whack the Mole” with a *Clarence* theme.



Balloons with Darts





Environmental Elements



Fair Entrance

Cost Analysis

Concept	Description	Quantity	Unit Cost(MXN)	Total Cost (MXN)
3Ds Max Software	License for 3D modeling	1	\$4,000	\$4,000
Hardware	Computer for design and rendering	1	\$15,000	\$15,000
Work Hours	Development and modeling (50 hours)	50	\$150	\$7,500
Work hours	Programming and animation (40 hours)	40	\$150	\$6,000
Other expenses	Various resources and materials			\$500
Total				\$33,000
Sale Price		\$49,000		
Total Profit		\$16,000		

Conclusions

This project successfully integrated programming tools such as OpenGL with 3D modeling using 3ds Max, enabling the creation of functional and visually coherent animations. Throughout the development process, both technical and creative skills were strengthened, as demonstrated by the creation of four animations that combined much of what was learned during the course.

Organization and time management were key to achieving the objectives, thanks to a structured plan using a Gantt chart that facilitated efficient task distribution across the modeling, animation, and documentation phases. Likewise, the preparation of technical, user, and cost manuals ensured clear and replicable documentation of the developed system.

Despite the challenges faced during this project, such as handling 3D coordinates, synchronizing movements, and optimizing rendering, these were overcome through research and iterative testing.