UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

COMPUTACIÓN GRÁFICA E INTERACCIÓN HUMANO COMPUTADORA

MANUAL TÉCNICO

“FERIA”

ESTUDIANTE:

319283064

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 05

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 20 de mayo del 2025

CALIFICACIÓN: _____

Contenido

Objetivo

Desarrollar una feria de juegos de destreza interactiva inspirada en los mundos animados de Mario Bros, Snoopy y El Increíble Mundo de Gumball.

El entorno incluirá juegos como lanzamiento de hacha, boliche, lanzamiento de dados, jaula de bateo, dardos y "golpea al topo".

Se incorporarán elementos decorativos típicos de una feria, tanto en los juegos como en el entorno en general, con el fin de ofrecer una experiencia que combine el uso de modelado 3D para escenarios, objetos y personajes, junto con un texturizado inspirado en los diferentes mundos. Además, se integrarán animaciones, un sistema de iluminación para mejorar la ambientación, y cámaras con distintos ángulos que permitirán al usuario recorrer e interactuar con la feria.

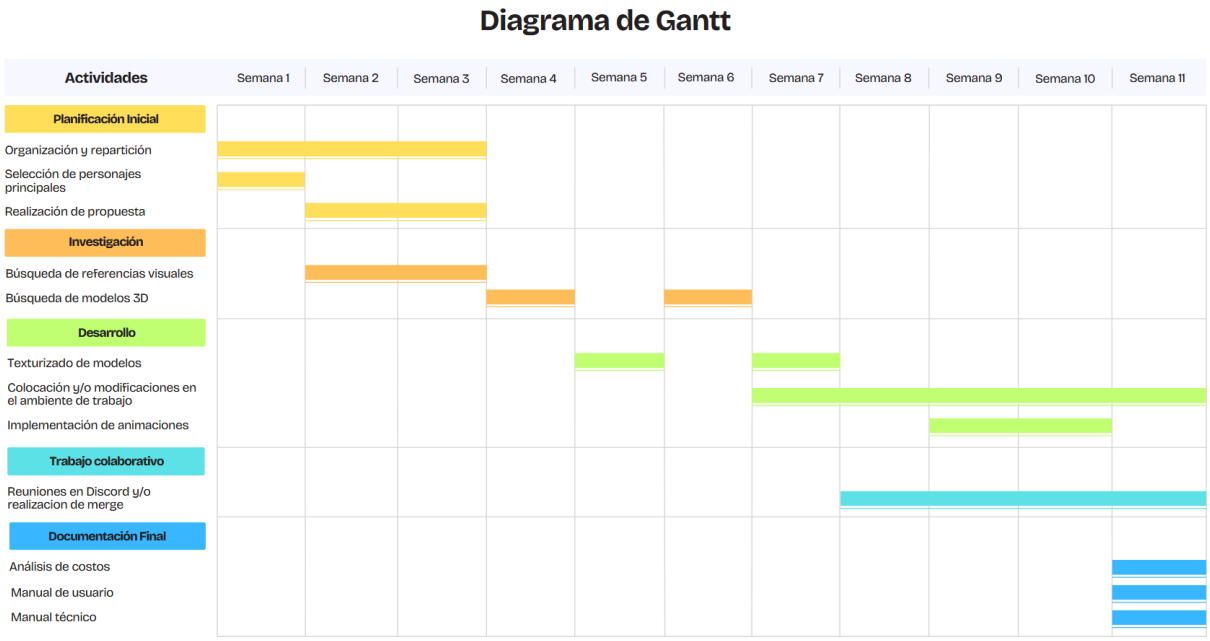
Para el desarrollo del proyecto se utilizarán herramientas como OpenGL y C++ para la programación e implementación, 3ds Max para el modelado y texturizado, Lightroom y GIMP para la edición y creación de texturas.

Alcance

1. **Modelado**: Llevar a cabo el modelado de las fachadas de cada juego, manteniendo una estructura simétrica que unifique el estilo general de la feria, decorar cada juego con elementos de los universos de Mario Bros, Snoopy y El Increíble Mundo de Gumball, y ambientar el entorno con modelos como árboles, bancas, postes, personajes, etc.
2. **Texturizado**: Aplicar texturas a cada modelo utilizando herramientas de edición de imágenes como GIMP o Lightroom ajustando sus dimensiones o editándolas según las necesidades.
3. **Animación**: Implementar animaciones interactivas en cada uno de los juegos, así como animaciones personalizadas para personajes y elementos decorativos.
4. **Iluminación**: Desarrollar un sistema de iluminación que permita iluminar el exterior de la feria, el interior de los puestos y simular la luz solar, incorporando una luz ambiental. Este sistema deberá incluir el encendido y apagado de luces automáticas según el ciclo de día y noche, así como el control de las luces del interior de los puestos mediante el uso del teclado.
5. **Cámaras**: Desarrollar tres tipos de cámaras para visualizar la feria: una aérea para tener una vista general, una en tercera persona para interactuar con la feria y una frontal centrada en los juegos para una vista amplia.

6. **Interacción del usuario**: Permitir al usuario recorrer libremente la feria desde una vista en tercera persona, así como activar juegos y animaciones mediante el uso del teclado.

Cronograma



Costo

Para facilitar la comprensión de los costos asociados a este proyecto, se presenta un desglose de los distintos tipos de gastos identificados. Aunque el proyecto fue desarrollado por un equipo de tres personas, los costos aquí expuestos corresponden al trabajo realizado por una sola persona.

- **Costo de recursos humanos**

El desarrollo del proyecto fue realizado por una sola persona con un nivel de experiencia intermedio en el lenguaje de programación C++. La carga total de trabajo fue de 100 horas.

Considerando una tarifa promedio de $110.00 MXN por hora, el costo total por concepto de recursos humanos se calcula de la siguiente manera:

Recursos humanos: $11,000.00 MXN

- **Costos fijos**

Los costos fijos corresponden a aquellos gastos que se mantienen constantes durante el proyecto, independientemente de la cantidad de trabajo realizada. En

este caso, al tratarse de una persona trabajando desde casa, se consideraron los siguientes:

- **Electricidad:** Se utilizó una laptop Asus TUF Gaming A15, con un consumo estimado mensual de $430.00 MXN. Para una duración del proyecto de 2.75 meses, el total es: $1,182.50 MXN

- **Internet:** Se empleó el servicio de internet Totalplay, con un costo mensual estimado de $420.00 MXN. Para 2.75 meses, se tiene: $1,155.00 MXN

- **Depreciación de equipo:** El equipo utilizado tiene un costo aproximado de $26,000.00 MXN y una vida útil estimada de 60 meses, lo que representa una depreciación mensual de $433.33 MXN. Para 2.75 meses: $1,191.67 MXN

Costos fijos: $3,529.17 MXN

- **Costos variables**

Los costos variables hacen referencia a recursos cuyo uso depende directamente de la ejecución del proyecto. En este caso:

- **Software:** Se emplearon herramientas gratuitas como Visual Studio 2022, GitHub, 3ds Max, GIMP y Lightroom.

Costos variables: $0.00 MXN

- **Costo neto**

Sumando los tres tipos de costos previamente descritos, se obtiene:

- Recursos Humanos: $11,000.00
- Costos Fijos: $3,529.17
- Costos Variables: $0.00

Costo neto: $14,529.17 MXN

- **Costo con margen de ganancia**

Para asegurar la rentabilidad del proyecto, se consideró un margen de ganancia del 30%, resultando en:

- Ganancia estimada: $4,358.75 MXN

Costo con margen de ganancia: $18,887.92 MXN

- **Costo final**

Al subtotal con utilidad se le añaden los impuestos correspondientes.

- **IVA (16%):** $3,022.07 MXN

Costo final: $21,909.99 MXN

El monto final de $21,909.99 MXN por persona representa un precio que cubre todos los costos operativos, los recursos utilizados, la remuneración profesional individual y las obligaciones fiscales correspondientes. Este valor asegura la viabilidad económica del proyecto y una compensación justa por el trabajo aportado por cada integrante.

Documentación

Archivo Main

- **hacha(...)**

Renderiza todos los modelos correspondientes al puesto de lanzamiento de hacha. Se encarga de aplicar las transformaciones necesarias y dibujar en pantalla objetos como la hacha, el soporte y elementos decorativos del entorno inmediato del puesto.

Parámetros:

glm::mat4& model: matriz de modelo usada para transformaciones.

GLuint uniformModel: ubicación del uniforme model en el shader.

std::vector<Model*>& objetosHacha: lista de punteros a los modelos del puesto.

- **dados(…)**

Dibuja los elementos del puesto de lanzamiento de dados, incluidos la mesa, dado, hongos decorativos y cuadros temáticos. También gestiona su transformación y ubicación dentro de la escena principal.

Parámetros:

glm::mat4& model: matriz de transformación.

GLuint uniformModel: ubicación del uniforme de transformación.

std::vector<Model*>& objetosDados: modelos relacionados con el juego de dados.

- **ambientacion(…)**

Renderiza todos los objetos de ambientación general de la feria: árboles, luminarias, bancas, basureros, laguna, entre otros. Sirve para dar vida y coherencia visual al entorno no jugable.

Parámetros:

glm::mat4& model: matriz de transformación global.

GLuint uniformModel: uniforme de transformación en el shader.

std::vector<Model*>& objetosAmbientacion: modelos del escenario estático.

- **comida(…)**

Dibuja los puestos de comida y sus respectivos elementos como palomitas, helados, algodones de azúcar. Añade atractivo visual y realismo temático a la feria.

Parámetros:

glm::mat4& model: matriz utilizada para ubicar modelos.

GLuint uniformModel: ubicación del uniforme en el shader.

std::vector<Model*>& objetosComida: modelos que representan comida y puestos.

- **NPCs(…)**

Renderiza los modelos de los personajes no jugables (NPCs) como Snoopy, Charlie Brown, Bowser, Donkey Kong, etc. Están repartidos por la feria con fines decorativos y de ambientación.

Parámetros:

glm::mat4& model: matriz de transformación común.

GLuint uniformModel: referencia al uniforme del modelo.

std::vector<Model*>& personajesNPCs: lista de personajes a renderizar.

- **zonaFotos(…)**

Renderiza una sección temática pensada como zona de fotos, que incluye personajes posando y escenarios decorativos. Se utilizan transformaciones animadas para ciertos elementos (alas, patas, etc.).

Parámetros:

glm::mat4& model: matriz base para transformaciones.

GLuint uniformModel: ubicación del uniforme model.

std::vector<Model*>& objetosZonaFotos: modelos que componen el escenario.

float deltaTime: tiempo entre frames, usado para animaciones.

- **arco(…)**

Dibuja el arco de entrada a la feria, incluyendo las puertas y el letrero animado. Esta función aplica desplazamientos de textura (offset UV) para animar la escritura del letrero.

Parámetros:

glm::mat4& model: matriz de transformación.

GLuint uniformModel: uniforme del modelo.

GLuint uniformTextureOffset: desplazamiento uniforme para animación UV.

std::vector<Model*>& objetosArco: modelos estructurales del arco.

float deltaTime: delta de tiempo para animación.

Texture& letreroTexture: textura con caracteres del letrero.

std::vector<Mesh*>& meshList: lista de mallas, usada aquí para renderizar los caracteres del letrero.

- **actualizarHacha(float velocidad)**

Actualiza la rotación y desplazamiento del modelo de hacha lanzado, simulando su trayectoria de vuelo. Se utiliza un valor delta (velocidad) para lograr una animación fluida en el tiempo.

Parámetro:

velocidad: tiempo escalado (delta time) que regula la velocidad de la animación.

- **actualizarDados(float velocidad)**

Administra la rotación y rebote del dado después de ser lanzado. Permite una simulación física básica del movimiento del dado sobre la mesa de juego.

Parámetro:

velocidad: valor proporcional al delta time, controla el tiempo de animación.

- **lanzarHacha()**

Activa la animación de lanzamiento del modelo de hacha, comenzando su trayectoria hacia el objetivo. Normalmente se activa con la tecla T.

- **lanzarDados()**

Simula el lanzamiento del dado, activando su rotación y desplazamiento con rebote. Normalmente responde a la tecla U.

Archivos por separado

- **Dados.cpp**

Variables

glm::vec3 posDado1, posDado2: posiciones actuales de cada dado.

glm::vec3 rotDado1, rotDado2: rotaciones acumuladas de los dados.

float velY: velocidad vertical para simular gravedad.

int rebotes: número de rebotes restantes.

bool lanzandoDados: si la animación de lanzamiento está activa.

Funciones

void dados(glm::mat4 model, GLuint uniformModel, std::vector<Model*> objetosDados): Renderiza el entorno completo del puesto de dados.

void renderDado(glm::mat4 model, GLuint uniformModel, Model& dado, glm::vec3 posicion, glm::vec3 rotacion): Dibuja un dado con posición y rotación específicas.

void renderStand3(glm::mat4 model, GLuint uniformModel, Model& stand): Renderiza el modelo del stand.

void renderMesaDados(glm::mat4 model, GLuint uniformModel, Model& mesa, glm::vec3 posicion): Renderiza una mesa en la posición especificada.

void renderHongo(glm::mat4 model, GLuint uniformModel, Model& hongo, glm::vec3 posicion, int rotY): Renderiza un hongo decorativo con rotación Y.

void renderCuadro(glm::mat4 model, GLuint uniformModel, Model& cuadro, glm::vec3 posicion, float escala): Renderiza un cuadro decorativo con escalado.

void actualizarDados(float deltaTime): Actualiza la animación de los dados en movimiento.

void lanzarDados(): Inicia la animación de lanzamiento de los dados.

- **Hacha.cpp**

Variables

float avanceHacha: desplazamiento hacia adelante del hacha.

float rotacionHacha: rotación durante el vuelo.

bool lanzandoHacha: si el hacha está en animación.

bool impactoHacha: si ha impactado.

float tiempoEsperaHacha: tiempo antes de reiniciar.

Funciones

void hacha(glm::mat4 model, GLuint uniformModel, std::vector<Model*> objetosHacha): Renderiza el puesto de hacha completo.

void renderStand(glm::mat4 model, GLuint uniformModel, Model& stand, Model& reja, Model& tabla, glm::vec3 posicion): Renderiza el stand y sus decoraciones.

void renderHachas(glm::mat4 model, GLuint uniformModel, Model& hacha, glm::vec3 posicion): Renderiza hachas estática y animada.

void renderObjetivos(glm::mat4 model, GLuint uniformModel, Model& objetivo, glm::vec3 posicion): Renderiza los blancos del juego.

void lanzarHacha(): Activa el lanzamiento animado del hacha.

void actualizarHacha(float deltaTime): Actualiza posición y rotación del hacha en vuelo.

- **ZonaFotos.cpp**

Variables

float vuelaSnoopy, velocidadSnoopy, amplitud: control del vuelo circular de Snoopy.

float snoopyY, snoopyZ, movSnoopy, movOffsetSnoopy: desplazamientos.

bool avanzaSnoopy: controla dirección del movimiento.

Funciones

void zonaFotos(glm::mat4 model, GLuint uniformModel, std::vector<Model*> objetosZonaFotos, float deltaTime): Renderiza y anima a los personajes en la zona de fotos.

void renderSnoopyAviador(glm::mat4 model, GLuint uniformModel, std::vector<Model*> objetosZonaFotos): Renderiza y anima al personaje Snoopy simulando que está volando como un aviador. Se mueve en una trayectoria circular (en los ejes Y-Z) mientras se desplaza lateralmente en el eje X, haciendo un recorrido de ida y vuelta. La animación combina rotaciones suaves y desplazamientos periódicos.

- **NPCs.cpp**

void NPCs(glm::mat4 model, GLuint uniformModel, std::vector<Model*> personajesNPCs): Renderiza todos los NPCs en sus posiciones.

Cada una renderiza al personaje correspondiente con posición y rotación.

void renderSnoopy(glm::mat4 model, GLuint uniformModel, Model& personaje, glm::vec3 posicion, int grados)

void renderPeppermint(glm::mat4 model, GLuint uniformModel, Model& personaje, glm::vec3 posicion, int grados)

void renderCharlie(glm::mat4 model, GLuint uniformModel, Model& personaje, glm::vec3 posicion, int grados)

- **Tierra.cpp**

Funciones

void tierra(glm::mat4 model, GLuint uniformModel, Texture& tierra, std::vector<Mesh*> meshList): Renderiza todo el terreno general con textura.

void renderTierra(glm::mat4 model, GLuint uniformModel, Texture& tierra, Mesh& piso, glm::vec3 posicion, glm::vec3 escala): Renderiza un bloque de terreno texturizado en posición y tamaño dados.

- **Ambientación.cpp**

Funciones

void ambientacion(glm::mat4 model, GLuint uniformModel, std::vector<Model*> objetosAmbientacion): Renderiza árboles, bancas, luminarias y otros elementos decorativos.

Cada función posiciona y dibuja un objeto del entorno

void renderLuminaria1(glm::mat4 model, GLuint uniformModel, Model& modelo, glm::vec3 posicion)

void renderBote1(glm::mat4 model, GLuint uniformModel, Model& modelo, glm::vec3 posicion, int grados)

void renderBote2(glm::mat4 model, GLuint uniformModel, Model& modelo, glm::vec3 posicion, int grados)

- **Arco.cpp**

Variables

std::vector<glm::vec2> mensajeLetrero: coordenadas UV de letras.

float tiempoLetrero: control de tiempo para mostrar letras.

int posicionLetrero: letra actual en pantalla.

glm::vec2 toffset: desplazamiento UV animado.

float anguloPuerta, velocidadPuerta: apertura de puertas.

bool abrirPuerta: estado de apertura.

Funciones

void arco(glm::mat4 model, GLuint uniformModel, GLuint uniformTextureOffset, std::vector<Model*> objetosArco, float deltaTime, Texture& letrero, std::vector<Mesh*> meshList): Renderiza el arco de entrada con animación de puertas y letrero.

void inicializarMensajeLetrero(): Inicializa las coordenadas para animar texto en el letrero.

Funciones dadas por el Ing. Jose Roque RG

- **mainWindow.Initialise()**

Inicializa la ventana principal de OpenGL con GLFW y GLEW, estableciendo los parámetros de contexto gráfico, tamaño de buffer, y capturando eventos.

Parámetros:

No recibe parámetros directamente (usa los valores definidos al construir mainWindow).

- **CreateObjects()**

Crea varias mallas geométricas primitivas (pirámide, piso, vegetación, letrero) con sus vértices e índices, y las almacena en meshList.

- **CrearDado()**

Construye y almacena la malla de un cubo con texturas aplicadas a cada cara, simulando un dado. Se guarda en meshList.

- **CreateShaders()**

Carga los archivos de shader (vertex y fragment) desde las rutas vShader y fShader, y los agrega al vector shaderList.

- **LoadModel(...)**

Carga un modelo .obj desde una ruta de archivo específica al objeto Model. Se llama muchas veces para cada objeto 3D de la feria (puestos, ambientación, personajes, etc.).

Parámetros:

const char* fileName: ruta relativa al archivo .obj del modelo.

- **Skybox(...)**

Crea una caja de cielo con 6 imágenes que se renderiza en segundo plano para simular un entorno envolvente. Se crean dos: una para el día y otra para la noche.

Parámetros:

std::vector<std::string> faceLocations: rutas de imágenes para cada cara del cubo (derecha, izquierda, abajo, arriba, frente, atrás).

- **Material(...)**

Crea un material con propiedades de reflectancia para aplicar en iluminación especular.

Parámetros:

float specIntensity: intensidad del reflejo especular.

float shininess: brillo del material (mayor valor = reflexión más pequeña y brillante).

- **DirectionalLight(...)**

Constructor que inicializa una luz direccional, la cual simula una fuente de luz lejana como el sol. No tiene posición, solo dirección, y afecta de manera uniforme todos los objetos en la escena.

Parámetros:

GLfloat red, green, blue: componentes del color de la luz.

GLfloat ambientIntensity: intensidad con la que la luz afecta las zonas no iluminadas directamente (luz ambiente).

GLfloat diffuseIntensity: intensidad de la luz directa sobre los objetos.

GLfloat xDir, yDir, zDir: dirección en la que apunta la luz.

- **PointLight(...)**

Constructor de una luz puntual, la cual emite luz en todas las direcciones desde una posición específica. Este tipo de luz se atenúa con la distancia, simulando una lámpara o farol.

Parámetros:

GLfloat red, green, blue: componentes de color de la luz.

GLfloat ambientIntensity: intensidad ambiental.

GLfloat diffuseIntensity: intensidad difusa.

GLfloat xPos, yPos, zPos: posición de la luz en el mundo 3D.

GLfloat constant, linear, exponent: coeficientes de atenuación de la luz. Estos determinan cómo disminuye la intensidad con la distancia (modelo de atenuación estándar).

- **SpotLight(...)**

Constructor de una luz tipo foco (spotlight), que es como una luz puntual pero con una dirección definida y un ángulo de apertura. Es útil para simular linternas, faros o reflectores.

Parámetros:

GLfloat red, green, blue: componentes del color.

GLfloat ambientIntensity: luz ambiental emitida.

GLfloat diffuseIntensity: intensidad difusa.

GLfloat xPos, yPos, zPos: posición en el espacio.

GLfloat xDir, yDir, zDir: dirección hacia la que apunta el foco.

GLfloat constant, linear, exponent: coeficientes de atenuación.

GLfloat edge: ángulo de apertura del cono de luz (en grados), define el límite de iluminación del foco.

- **getCamaraActiva()**

Devuelve el ID entero de la cámara actualmente seleccionada por el usuario (0 = aérea, 1 = tercera persona, 2 = cámara de puestos).

- **mouseControl(x, y)**

Recibe los desplazamientos del mouse en X y Y, y ajusta la orientación (yaw, pitch) de la cámara activa.

Parámetros:

GLfloat x: desplazamiento horizontal del mouse.

GLfloat y: desplazamiento vertical del mouse.

- **calculateViewMatrix()**

Devuelve la matriz de vista (glm::mat4) calculada a partir de la posición y dirección de la cámara. Usada para transformar la escena desde el punto de vista del usuario.

- **UseShader(...)**

Activa el shader compilado previamente para empezar a usarlo en las operaciones de dibujo.

- **SetSpotLights(...)**

Envía un arreglo de luces tipo spotlight al shader activo. Cada luz contiene información de color, dirección, atenuación, intensidad y ángulo de corte. Se usa para iluminar zonas específicas de la escena como los juegos individuales (bateo, bolos, etc.).

Parámetros:

SpotLight* lights: puntero a un arreglo de luces SpotLight activas.

unsigned int lightCount: cantidad de luces SpotLight que se deben enviar al shader

- **SetPointLights(...)**

Enlaza al shader un conjunto de luces puntuales, como faroles o postes. Estas luces emiten en todas las direcciones desde un punto y se atenúan con la distancia.

Parámetros:

PointLight* lights: arreglo de luces PointLight.

unsigned int lightCount: número de luces a activar y pasar al shader.

- **SetDirectionalLight(...)**

Asigna una única luz direccional al shader, la cual simula una fuente de luz como el sol. Esta luz afecta toda la escena desde una dirección fija.

Parámetros:

DirectionalLight* dLight: puntero a una instancia de luz direccional con los parámetros ya definidos.

- **RenderModel()**

Dibuja en pantalla un modelo previamente cargado en OpenGL.

- **swapBuffers()**

Intercambia el buffer de dibujo en la ventana para mostrar la imagen renderizada. Hace parte del ciclo de renderizado en tiempo real.

Conclusiones

Este proyecto representó una oportunidad para aplicar los conocimientos adquiridos en la materia de Computación Gráfica e Interacción Humano Computadora. Se logró combinar modelado, texturizado, iluminación, cámaras e interacción en un entorno complejo como lo es una feria de juegos, inspirada en mundos como Mario Bros, Snoopy y Gumball. Aunque el mayor reto fue la complejidad del proyecto, especialmente por los requerimientos y el tiempo, resultó gratificante ver cómo todo fue tomando forma.

Durante el desarrollo se trabajaron distintos aspectos como el diseño de los puestos, la ambientación del entorno con decoraciones temáticas, la implementación de un sistema de iluminación automático con ciclo día-noche, el uso de cámaras que ofrecen diferentes perspectivas, las animaciones para cada uno de los juegos y de los personajes principales, entre muchas otras cosas. También se añadió la interacción mediante teclado, permitiendo al usuario interactuar con el entorno.

Este proyecto permitió aplicar lo aprendido en clase de una manera creativa y técnica a la vez. Aunque fue demandante en complejidad y tiempo, fue una experiencia enriquecedora y entretenida, que además fomentó el trabajo colaborativo y la comunicación.

Links Modelos:

Casa de Snoopy: https://sketchfab.com/3d-models/snoopy-7ca97398a08c4e9fb9a216c6982564dc

Algodon de azucar: https://www.cgtrader.com/items/4617534/download-page

Tabla: https://sketchfab.com/3d-models/cutting-board-61eadc1d646c47ac892ce418677dfb76#download

Snoopy: https://sketchfab.com/3d-models/snoopy-72116d2e288f4c45a11f323d76142a6c#download

Luminarias: https://www.turbosquid.com/es/3d-models/free-max-model-classical-street-light/965356

Botes de basura: https://www.turbosquid.com/es/3d-models/free-obj-model-street-bin/1106299

Casa (cabinas de juegos): https://free3d.com/3d-model/old-farm-house-91130.html

Peppermint: https://sketchfab.com/3d-models/wii-u-the-peanuts-movie-snoopys-grand-adventur-dbe36b120be94397b0327c22efbcfe75

Woodstock: https://sketchfab.com/3d-models/wii-u-the-peanuts-movie-woodstock-d7a8747fc83f49e194dd9ccde0cc2d49

Charlie Brown: https://sketchfab.com/3d-models/wii-u-the-peanuts-movie-charlie-brown-9acef7efdb42429ab0f79be59c193d41

Puesto de comida: https://free3d.com/es/modelo-3d/temporary-food-stalls-80503.html

Mesa de dados: https://free3d.com/es/modelo-3d/craps-table-v1--845885.html

Malla hacha: https://free3d.com/3d-model/fence-86772.html

Hacha: https://free3d.com/3d-model/ax-92952.html

Gumball waterson: https://sketchfab.com/3d-models/gumball-fusionfall-heroes-39c568e694e1478bb0e3ed81cde9c359

Librería Premios: https://sketchfab.com/3d-models/libreria-vuota-098205aa418f4237b361328a282e21e3

Objetivo de hacha: https://sketchfab.com/3d-models/archery-target-bfa271fc02a94640886a9dfc2cd6ad4a

Links Texturas:

Cuadros:

https://www.arte-mio.mx/MLM-763335879-set-nintendo-super-mario-bros-4-cuadros-en-tela-canvas-list-_JM

The Amazing World Of Gumball Clown Mystery Explained

peanuts-the-art-of-snoopy-history-snoopy-flying-ace-snoopy-come-home

Dado:

https://es.ssbwiki.com/wiki/Huevo_de_Yoshi

https://mario.fandom.com/es/wiki/Cheep_Cheep

https://cl.pinterest.com/pin/279645458107164172/

https://mario.fandom.com/es/wiki/Bill_Bala

https://mario.fandom.com/es/wiki/Flor_de_Fuego

https://mario.fandom.com/es/wiki/Champi%C3%B1%C3%B3n

https://www.pngwing.com/es/free-png-zhaxf

Pasto: https://img.freepik.com/fotos-premium/textura-hierba-verde-que-esta-hecha-empresa-empresa_612834-258.jpg

Skybox: https://opengameart.org/content/sky-box-sunny-day

Tabla: https://www.istockphoto.com/es/fotos/madera-clara-textura

Tierra:https://png.pngtree.com/thumb_back/fw800/background/20220711/pngtree-brown-paper-texture-ground-parcel-photo-image_999914.jpg