

**UNIVERSIDAD CENTRAL DEL
ECUADOR**

**FACULTAD DE INGENIERÍA Y CIENCIAS
APLICADAS**

COMPUTACIÓN

PROYECTO DE VIDEOJUEGOS



TEMA:

Proyecto Unity: Demo Técnica de Gráficos en Unity 6, uso de Billboarding, Mapeo topológico, Skybox y Ray casting

INTEGRANTES:

- JOFFRE DAVID ARIAS BASANTES
- ANDRES SEBASTIAN LARA VIANA
- ANGELO FABRICIO PUJOTA PINEDA
- PABLO FERNANDO SIMBAÑA PULUPA

FECHA: 09/02/2026

Contenido

1. Introducción	2
1.1 Idea del proyecto.	2
1.2 Idea del proyecto	2
1.3 Código	2
2. Repositorios y recursos	5
2.1 Videos de YouTube.	5
3. Prompts	6
4. Actividades por integrante	7

1. Introducción

1.1 Idea del proyecto.

La idea principal fue crear un **entorno inmersivo de aprendizaje** donde el usuario no solo lea sobre conceptos teóricos de computación gráfica, sino que los experimente de forma interactiva. Se buscó unificar cuatro pilares:

- **Geometría:** Mapeo topológico y Billboarding.
- **Entorno:** Skyboxes/Skydomes.
- **Interacción Física:** Ray Casting.

1.2 Idea del proyecto

El desarrollo fue incremental:

Inicio: Establecer un controlador de personaje en primera/tercera persona para permitir la exploración.

Escenario: Generación de un terreno con elevaciones para representar el
Mapeo Topológico: Representar un minimapa 2d en un entorno 3d y poder visualizar desde el mapa el movimiento del mundo.

Atmósfera: Implementación de un material procedural para el **Skybox**, definiendo los límites visuales del mundo.

Optimización Visual: Uso de **Billboarding** para representar vegetación (árboles) eficiente que siempre mira al jugador.

Interactividad: Programación de un sistema de **Ray Casting** para detectar objetos en el espacio y generar efectos visuales (láser e impactos).

1.3 Código

Skybox

Controlador del cielo:

The screenshot shows the Unity Editor's code editor window with the script `ControladorCielo.cs` open. The window has tabs for `ControladorTiempo.cs*` and `ControladorCielo.cs`. The code implements a player controller with movement logic based on mouse input for aiming and keyboard WASD for movement, along with a spacebar jump and shift key controls for vertical movement.

```
using UnityEngine;
using UnityEngine.InputSystem; // Importante para Unity 6

public class ControladorCielo : MonoBehaviour
{
    [Header("Configuración de Movimiento")]
    public float velocidad = 5f;
    public float velocidadVuelo = 5f; // Velocidad para subir/bajar
    public float sensibilidadMouse = 0.5f;

    [Header("Referencias")]
    public Transform camaraJugador;

    private float rotacionX = 0f;
    private CharacterController controller;

    void Start()
    {
        controller = GetComponent<CharacterController>();
        Cursor.lockState = CursorLockMode.Locked; // Oculta el mouse
        if (camaraJugador == null) camaraJugador = Camera.main.transform;
    }

    void Update()
    {
        // --- 1. MIRAR (Mouse) ---
        Vector2 mouseDelta = Mouse.current.delta.ReadValue() * sensibilidadMouse;

        rotacionX -= mouseDelta.y;
        rotacionX = Mathf.Clamp(rotacionX, -90f, 90f);

        camaraJugador.localRotation = Quaternion.Euler(rotacionX, 0f, 0f);
        transform.Rotate(Vector3.up * mouseDelta.x);

        // --- 2. MOVERSE (WASD) ---
        Vector2 inputMovimiento = Vector2.zero;
        if (Keyboard.current.wKey.isPressed) inputMovimiento.y = 1;
        if (Keyboard.current.sKey.isPressed) inputMovimiento.y = -1;
        if (Keyboard.current.aKey.isPressed) inputMovimiento.x = -1;
        if (Keyboard.current.dKey.isPressed) inputMovimiento.x = 1;

        // Dirección horizontal (frente y lados)
        Vector3 movimientoHorizontal = transform.right * inputMovimiento.x + transform.forward * inputMovimiento.y;

        // --- 3. VOLAR (Espacio y Shift) ---
        float movimientoVertical = 0f;

        // Si presiona ESPACIO -> Sube
        if (Keyboard.current.spaceKey.isPressed)
        {
            movimientoVertical = 1f;
        }
        // Si presiona SHIFT IZQUIERDO -> Baja (Útil para acomodarse)
        else if (Keyboard.current.leftShiftKey.isPressed)
        {
            movimientoVertical = -1f;
        }

        // Combinamos todo: (Movimiento Horizontal * velocidad) + (Movimiento Vertical * velocidadVuelo)
        // Usamos Vector3.up para que siempre suba hacia el cielo absoluto, no hacia donde miras
        Vector3 movimientoFinal = (movimientoHorizontal * velocidad) + (Vector3.up * movimientoVertical * velocidadVuelo);

        // Mover el CharacterController
        controller.Move(movimientoFinal * Time.deltaTime);
    }
}
```

Se usa para que el jugador pueda moverse en el entorno

Controlador del tiempo:

ControladorTiempo.cs* ✎ X ControladorCielo.cs

Assembly-CSharp

```

    using UnityEngine;
    using UnityEngine.InputSystem; // Requerido para Unity 6

    public class ControladorTiempo : MonoBehaviour
    {
        [Header("Configuración de Luz")]
        public Light luzSolar; // Arrastra tu Directional Light aquí
        public float velocidadTiempo = 20f; // Qué tan rápido avanza al presionar la tecla

        void Update()
        {
            // Verificamos si la tecla 'T' está siendo presionada
            if (Keyboard.current.tKey.isPressed)
            {
                AvanzarTiempo();
            }

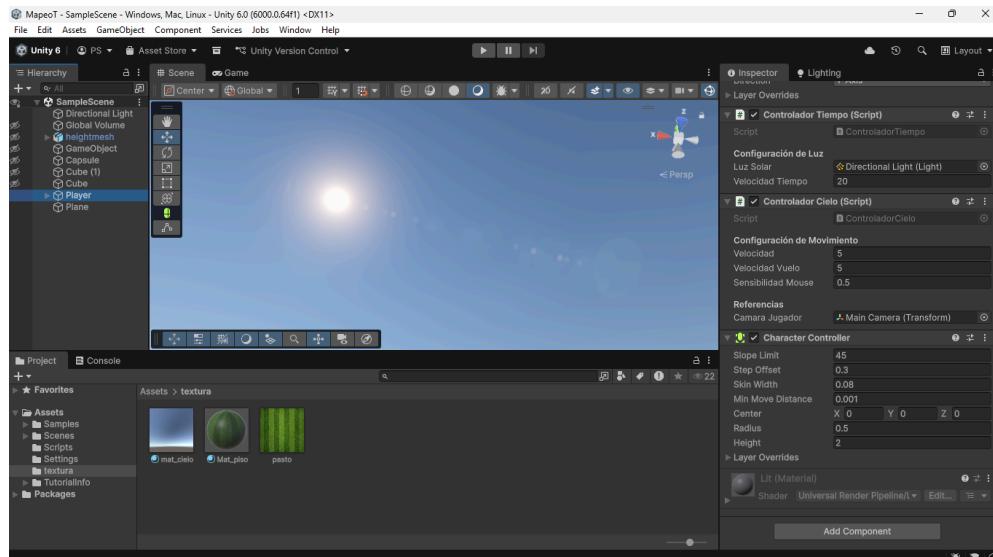
            // Opcional: Tecla 'R' para retroceder el tiempo
            if (Keyboard.current.rKey.isPressed)
            {
                RetrocederTiempo();
            }
        }

        void AvanzarTiempo()
        {
            if (luzSolar != null)
            {
                // Rotamos la luz en el eje X para simular el paso del sol
                luzSolar.transform.Rotate(Vector3.right * velocidadTiempo * Time.deltaTime);
            }
        }

        void RetrocederTiempo()
        {
            if (luzSolar != null)
            {
                luzSolar.transform.Rotate(Vector3.left * velocidadTiempo * Time.deltaTime);
            }
        }
    }

```

Se usa para hacer la transición del día y de la noche



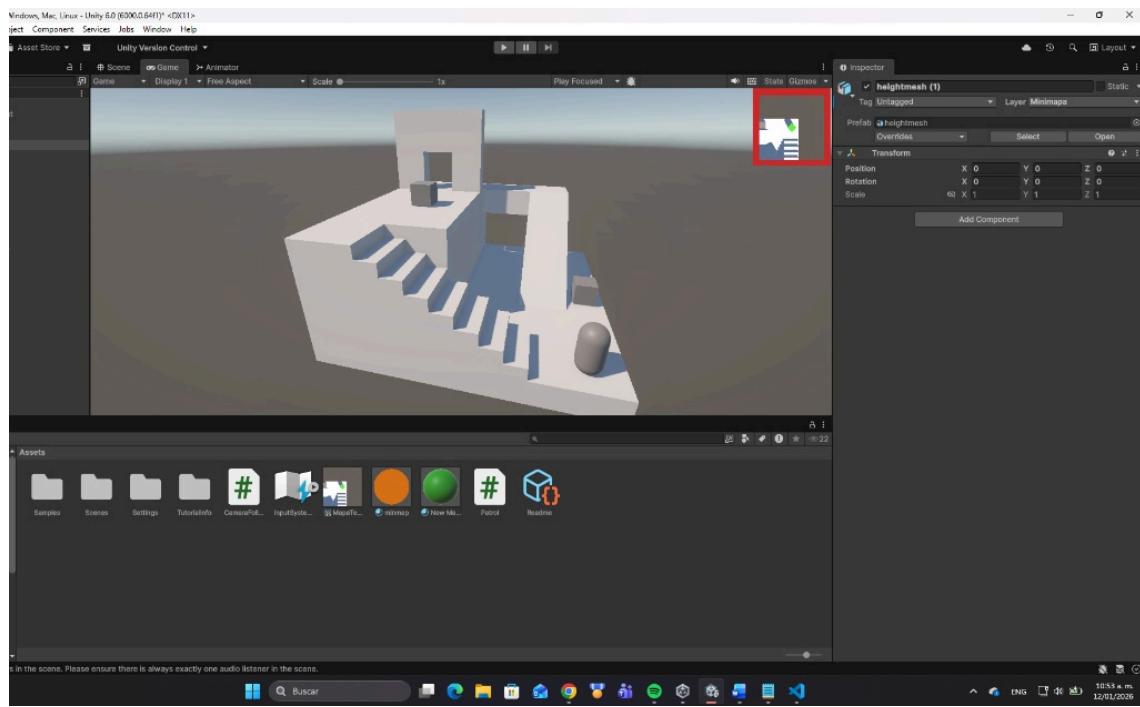
Mapeo topológico

Sistema de patrullaje entre varios puntos



```
C#  
  
using UnityEngine;  
using UnityEngine.AI;  
  
public class Patrol : MonoBehaviour  
{  
    [Header("Arrastra aquí los Cubos (Destinos)")]  
    public Transform[] puntosDeRuta; // Aquí arrastras tus objetos (Cube 1, Cube  
    private NavMeshAgent agent;  
    private int indiceActual = 0;  
  
    private void Awake()  
    {  
        agent = GetComponent<NavMeshAgent>();  
    }  
  
    private void Start()  
    {  
        // Desactivamos el frenado automático para que no se detenga en seco antes  
        agent.autoBraking = false;  
  
        // Manda al agente al primer punto nada más empezar el juego  
        IrAlSiguientePunto();  
    }  
  
    private void Update()  
    {  
        // Verifica si el agente ya llegó (o está muy cerca) del destino  
        // "!agent.pathPending" asegura que Unity ya terminó de calcular el camino  
        // "remainingDistance < 0.5f" es la distancia a la que cambia de objetivo  
        if (!agent.pathPending && agent.remainingDistance < 0.5f)  
        {  
            IrAlSiguientePunto();  
        }  
    }  
  
    void IrAlSiguientePunto()  
    {  
        // Si la lista de puntos está vacía, no hace nada para evitar errores  
        if (puntosDeRuta.Length == 0)  
            return;  
  
        // Le dice al agente que se mueva a la posición del punto actual  
        agent.destination = puntosDeRuta[indiceActual].position;  
  
        // Calcula cuál es el siguiente índice.  
        // El símbolo '%' (módulo) hace que si llega al final de la lista (ej: 2)  
        indiceActual = (indiceActual + 1) % puntosDeRuta.Length;  
    }  
}
```

Este script se realizó para el movimiento de los npc o cápsulas y poder ver el desplazamiento en el minimapa



RayCasting

Sistema Disparo:

```

    Script de Unity (1 referencia de recurso) | 0 referencias
public class SistemaDisparo : MonoBehaviour
{
    [Header("Configuración")]
    public float rango = 100f; // Distancia del disparo
    public Transform camaraJugador; // Referencia a la cámara (Main Camera)

    // Mensaje de Unity | 0 referencias
    void Start()
    {
        // Si se te olvida poner la cámara, intenta buscarla automáticamente
        if (camaraJugador == null) camaraJugador = Camera.main.transform;
    }

    // Mensaje de Unity | 0 referencias
    void Update()
    {
        // --- DETECTAR CLICK IZQUIERDO (NUEVO INPUT SYSTEM) ---
        // Usamos .wasPressedThisFrame para que dispare una sola vez por clic
        if (Mouse.current != null & Mouse.current.leftButton.wasPressedThisFrame)
        {
            Disparar();
        }
    }

    1 referencia
    void Disparar()
    {
        RaycastHit hit; // Aquí guardamos la info de lo que golpeamos

        // Lanzamos el rayo desde la posición de la cámara hacia adelante
        if (Physics.Raycast(camaraJugador.position, camaraJugador.forward, out hit, rango))
        {
            // Dibuja una linea roja en la escena (solo visible en la pestaña "Scene", no en "Game")
            Debug.DrawLine(camaraJugador.position, hit.point, Color.red, 2f);
            // Debug para ver en consola qué golpeamos
            Debug.Log("Disparé a: " + hit.transform.name);

            // Intentamos obtener el script del enemigo en el objeto que golpeamos
            ObjetivoEnemigo enemigo = hit.transform.GetComponent<ObjetivoEnemigo>();

            // Si el objeto TIENE el script, es un enemigo
            if (enemigo != null)
            {
                enemigo.RecibirDaño();
            }
        }
    }
}

```

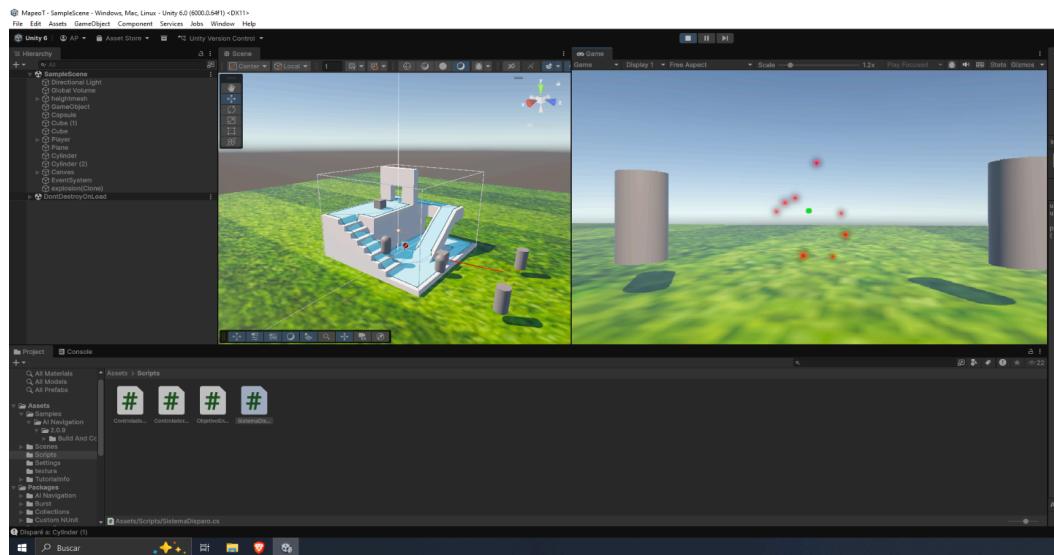
Objetivo Enemigo

```
using UnityEngine;

public class ObjetivoEnemigo : MonoBehaviour
{
    [Header("Efectos")]
    public GameObject particulasExplosion; // Arrastra aquí tu prefab de partículas

    // Esta función será llamada por el jugador
    public void RecibirDaño()
    {
        // 1. Crear la explosión antes de morir
        if (particulasExplosion != null)
        {
            Instantiate(particulasExplosion, transform.position, Quaternion.identity);
        }

        // 2. Destruir al enemigo
        Destroy(gameObject);
    }
}
```



Billboarding

```
using UnityEngine;

public class NameBillboard : MonoBehaviour
// Define una clase llamada NameBillboard que hereda de MonoBehaviour
// Esto permite que el script se pueda usar como componente en un GameObject
{
    void LateUpdate()
    // LateUpdate se ejecuta una vez por frame, después de Update
    // Es ideal para billboards porque asegura que la cámara ya se movió
    {
        Camera cam = Camera.main;
        // Obtiene la cámara principal (la que tiene el tag "MainCamera")

        if (cam == null) return;
        // Si no hay una cámara principal, sale del método para evitar errores

        Vector3 camPos = cam.transform.position;
        // Guarda la posición de la cámara en un Vector3

        camPos.y = transform.position.y;
        // Igualamos la altura (eje Y) del texto con la del objeto
        // Así el billboard solo rota en el eje Y y no se inclina hacia arriba o abajo

        transform.LookAt(camPos);
        // Rota el objeto para que mire hacia la posición de la cámara

        transform.Rotate(0f, 180f, 0f);
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine; // Librería principal de Unity para que funcionen los comandos como GameObject y Transform

public class Billboard : MonoBehaviour // Define el nombre de la clase y permite que se asigne a objetos en Unity
{
    private Camera theCam; // Variable privada para guardar la referencia a la cámara

    public bool useStaticBillboard; // Variable pública (aparece en el Inspector) para elegir el tipo de rotación

    // Se ejecuta una sola vez al iniciar el juego o al aparecer el objeto
    void Start()
    {
        // Busca automáticamente la cámara que tenga el tag "MainCamera" y la guarda en la variable
        theCam = Camera.main;
    }

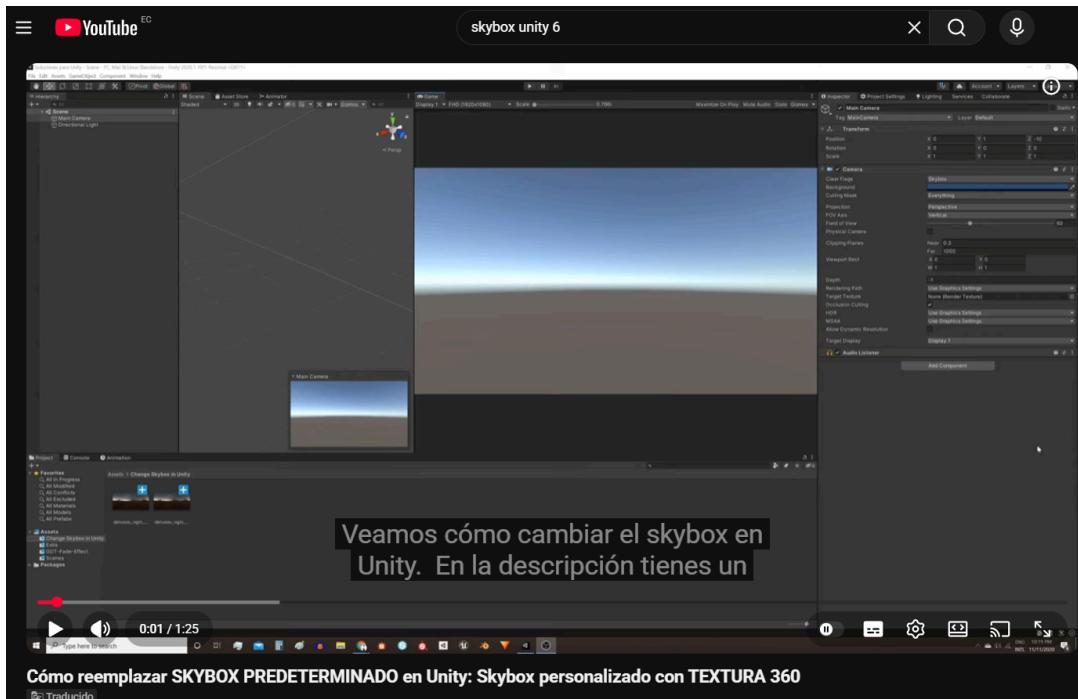
    // Se ejecuta en cada frame, pero después de que todos los objetos hayan procesado su Update normal
    void LateUpdate()
    {
        // Si la opción useStaticBillboard NO está marcada (está en falso)
        if (!useStaticBillboard)
        {
            // Hace que el objeto rote para mirar directamente hacia la posición de la cámara
            transform.LookAt(theCam.transform);
        }
        else // Si la opción está marcada como verdadero
        {
            // Copia exactamente la rotación que tiene la cámara (útil para efectos 2D en mundos 3D)
            transform.rotation = theCam.transform.rotation;
        }

        // Esta línea bloquea la rotación en los ejes X y Z para que el objeto siempre esté "vertical"
        // Solo conserva la rotación en el eje Y (vertical)
        transform.rotation = Quaternion.Euler(0f, transform.rotation.eulerAngles.y, 0f);
    }
}
```

2. Repositorios y recursos

2.1 Videos de YouTube.

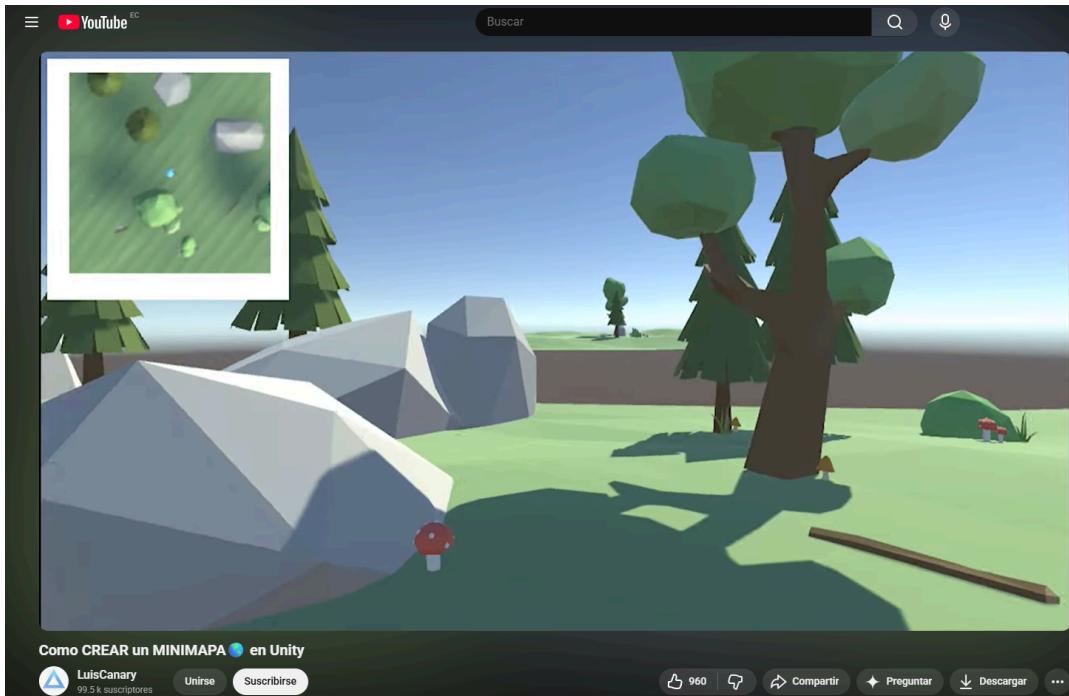
Videos de referencia



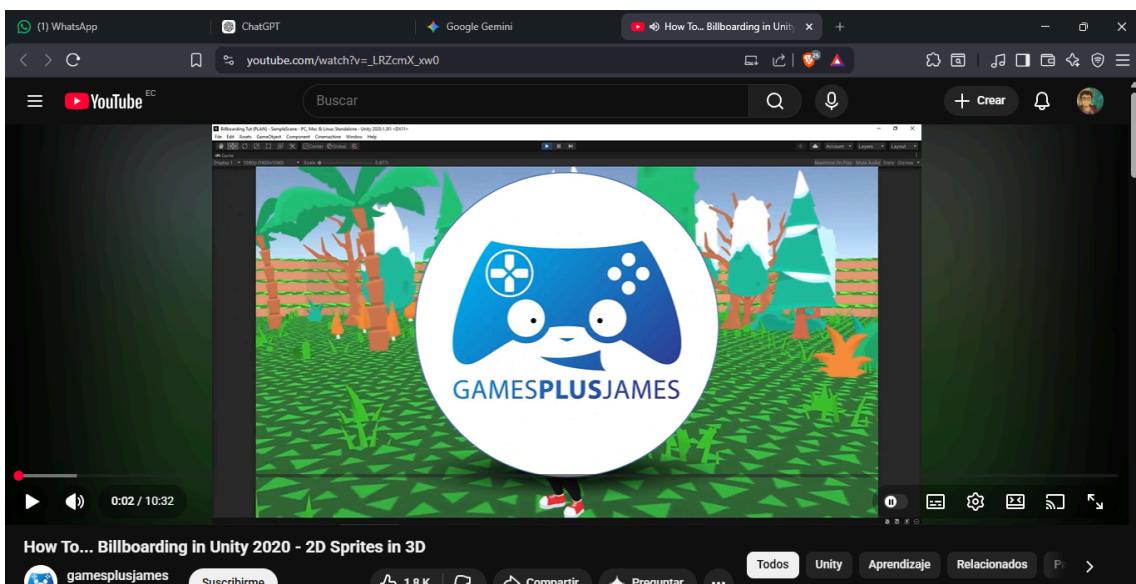
"Cómo reemplazar SKYBOX PREDETERMINADO en Unity: Skybox personalizado con TEXTURA 360"

Enlace: <https://youtu.be/ckgfQdaEdwk?si=gSQBAkOSRv-52vU8>

Cómo crear un minimapa en Unity



Enlace: <https://www.youtube.com/watch?v=HgRGsnxhqGc>



How To... Billboardng in Unity 2020 - 2D Sprites in 3D

Enlace: https://www.youtube.com/watch?v=_LRZcmX_xw0

3. Prompts

IA utilizadas: ChatGPT, Google Gemini

Durante el desarrollo del proyecto se utilizaron herramientas de inteligencia artificial como apoyo conceptual y técnico. Algunos de los prompts utilizados fueron:

- "Quiero crear un proyecto en Unity Windows, Mac, Linux 6.0 con Billboarding, Mapeo topológico, Skyboxes y skydomes, Ray casting. Indícame los pasos detallados."
- "Tengo muchos errores en la consola: InvalidOperationException: You are trying to read Input using the UnityEngine.Input class, but you have switched active Input handling to Input System package..."la animación la descargue de mixamo, pero solo la animación
- "Quiero importar un personaje y que tenga animaciones y colisiones, quiero que el Quad tenga textura de árboles."
- "Para la parte del raycasting quiero que se vea el rayo que traza el jugador y el objetivo."

Enlace de los prompts:

<https://gemini.google.com/share/de3b14340e4f>

4. Actividades por integrante

El proyecto fue desarrollado por un equipo de **cuatro integrantes**, cuyas responsabilidades se distribuyeron de la siguiente manera:

ANGELO FABRICIO PUJOTA PINEDA – Diseño y lógica inicial del proyecto

- **Implementación de Mecánica de Disparo mediante Raycasting** Desarrollo de un script en C# para gestionar la detección de impactos utilizando la física del motor (`Physics.Raycast`). Se configuró la proyección de rayos desde el vector delantero (*forward*) de la cámara principal para interactuar con objetos a distancia, validando colisiones mediante la obtención de componentes específicos (`GetComponent`) en los objetivos impactados.
- **Gestión de Feedback Visual y Ciclo de Vida de Objetos** Programación de la lógica de destrucción de enemigos vinculada a la instancia (`Instantiate`) de efectos de partículas (explosiones). Se aseguró la correcta gestión de memoria configurando la destrucción automática tanto del objeto objetivo como del *Prefab* de partículas tras finalizar su animación, evitando fugas de rendimiento en la jerarquía de la escena.
- **Integración de Retícula (Crosshair) y Alineación UI** Configuración de la interfaz de usuario (UI) para la asistencia de apuntado. Se implementó una imagen de retícula anclada al centro absoluto del *Canvas* mediante `RectTransform`, garantizando que la guía visual del jugador coincida perfectamente con la trayectoria física del rayo (Raycast) independientemente del movimiento o resolución de la pantalla.

JOFFRE DAVID ARIAS BASANTES – Mapeo Topológico

- Implementación de Navegación Autónoma en Entorno 3D configurando un componente NavMeshAgent sobre un controlador de personaje 3D (Cápsula) y desarrollo de un script en C# para gestionar el patrullaje cíclico entre múltiples puntos de destino
- Configuración de Captura Ortográfica (Render Texture) Implementación de una cámara secundaria en modo Orthographic posicionada cenitalmente sobre el escenario 3D. Configurando esta cámara para que no renderize directamente en pantalla, sino que vuelque su salida visual en un activo de Render Texture dedicado, creando la base de la imagen del mapa topológico.
- Integración de Visualización Topológica en la Interfaz (UI) Creación de la representación visual del mapa en el Canvas del juego utilizando un componente Raw Image. Asignando el *Render Texture* generada previamente a este componente y utilizando las herramientas de *Rect Transform* para anclar y dimensionar correctamente el minimapa en una esquina de la pantalla.
- Optimización Visual mediante Capas y Culling Masks Se mejoró la legibilidad del minimapa utilizando el sistema de Layers de Unity. Se creó una capa específica ("Minimap") para elementos simplificados (como un ícono plano para el jugador y una versión sin sombras del terreno usando materiales *Unlit*) y configurando las Culling Masks de ambas cámaras para que la cámara principal renderice la escena 3D realista y la cámara del mapa renderice solo la vista topológica limpia.

PABLO FERNANDO SIMBAÑA PULUPA – Implementación en Unreal Engine

- Configuración del sistema de iluminación ambiental, creación de materiales procedurales para el cielo y ajuste de la atmósfera global.
- Estudio de la diferencia técnica entre un **Skybox** (mapeo de 6 caras en un cubo infinito) y un **Skydome** (malla semiesférica con coordenadas polares).
- Decisión de utilizar un **Skybox Procedural** en Unity 6 para permitir cambios dinámicos en el color del horizonte y la posición del sol.
- Creación de un nuevo material optimizado para el motor de renderizado **URP (Universal Render Pipeline)**
- Asignación de la *Directional Light* de la escena como la fuente de sol principal para que el disco visual del Skybox coincida con la dirección de las sombras proyectadas en el plano.

ANDRES SEBASTIAN LARA VIANA – Documentación y pruebas

- **Elaboración de la documentación técnica del proyecto**

Se realizó la documentación integral del desarrollo del proyecto, describiendo de manera estructurada los objetivos, la idea general, las tecnologías empleadas y los principales sistemas implementados en Unity 6. Esta documentación permitió consolidar el conocimiento del equipo y facilitar la comprensión del funcionamiento interno del proyecto, tanto a nivel conceptual como técnico.

- **Aseguramiento de calidad y estabilidad:** Liderazgo en la fase de pruebas finales para corregir errores críticos de integración, como los conflictos de migración al nuevo Input System de Unity y la validación de la lógica de disparo por Raycasting.
- **Soporte pedagógico y técnico:** Creación de la documentación técnica detallada para la presentación en clase, facilitando la explicación de conceptos complejos como el mapeo topológico mediante Render Textures y el sistema de iluminación procedural del Skybox.
- **Validación visual del proyecto:** Generación de material audiovisual y capturas de pantalla que demuestran la integración fluida de las mecánicas de Billboard y el comportamiento dinámico de los NPCs en el entorno inmersivo.