

NULLOINTEREXCEPTION

PROJECTE M13

Pablo Sanjose, Victor González, Dori Ionita

Xavier Abilla, Roger Girbes, Montse Tallada, María José Uribe

2ºDAM

Índice

1- Introducción	4
1.1- Motivación Personal	4
1.2- Objetivo y Explicación del Proyecto	4
1.3- Metodología	5
2- Proyecto	7
2.1- Idea inicial	7
2.2- Temática	7
2.3- Historia	8
2.4- Creación del mapa	8
2.5- Modelo 3D jugador	13
2.6- Animaciones	15
2.7- Scripts	18
2.7.1- Cámara	18
2.7.2- Jugador	19
2.7.3- Diálogos	24
2.7.4- Movimiento de los Coches	31
2.7.4- Cambio de Escenas	33
3- Conclusión	34
3.1- Resultado Obtenido	34
3.2- Valoración	35
5- Webgrafía	36
6- Anexos	36

1- Introducción

1.1- Motivación Personal

Desde el momento que formamos el grupo sabíamos que queríamos hacer un proyecto en el que nos ayudará a desarrollar habilidades técnicas y creativas, como la programación, el diseño gráfico y el storytelling. A pesar de que sabíamos que este pequeño videojuego no estaría tan desarrollado ni perfeccionado como uno cualquiera del gran mercado de los Videojuegos, sí que tuvimos en cuenta que sería una forma divertida y entretenida de pasar el tiempo y de trabajar en equipo. En resumen, hacer un proyecto escolar sobre un videojuego basado en la realidad puede ser una experiencia enriquecedora y gratificante para cualquier estudiante interesado en los videojuegos y en la tecnología.

1.2- Objetivo y Explicación del Proyecto

Nuestro objetivo principal es crear un juego estilo [aventura grafica](#) en el que pudieras interactuar con varios [NPCs](#). Hemos querido basar parte de la dinámica del juego en los trabajos precarios que muchas veces nos vemos obligados a ejercer, ya que son un aspecto importante de la vida cotidiana. La dinámica principal del juego se basa en ayudar o interesarse por problemas que puedan tener personajes secundarios, de esta forma el videojuego te obliga a investigar y explorar por tu cuenta pero llevándote de la mano e indicando siempre por donde avanzar.

Un objetivo que teníamos en el videojuego era poder incluir personajes basados en personas que puedes encontrar en tu dia a dia e implementar mecánicas de juego que reflejaran situaciones de la vida cotidiana.

Como queríamos que todo el mundo se sintiera identificado con nuestro personaje lo hemos basado en el famoso meme de [NPC Wojack](#).

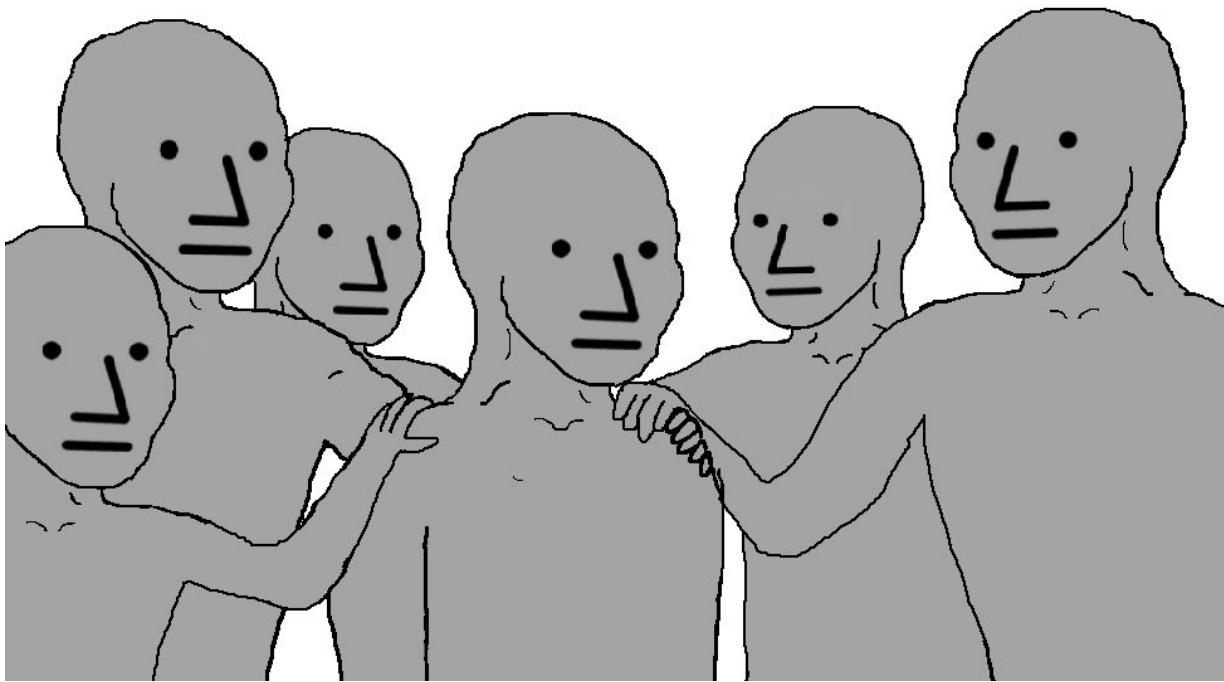


Imagen del cómic “*todos se convertirán*” creada por @SpookyStirnman

1.3- Metodología

Para organizar las diferentes tareas que hemos tenido en este proyecto hemos empleado la metodología SCRUM. Hemos dividido nuestra tabla en los diferentes apartados: Tareas por hacer, Tareas en curso, Tareas finalizadas, Parking y Tareas Validadas. Hemos considerado que las tareas más importantes eran las de hacer un menú, investigar cómo implementar diálogos de personajes y crear un jugador principal con animaciones propias , por eso las hemos puesto desde el primer día en la categoría de Tareas en curso.

NullPointerException		
Tasques per fer	Tasques en curs	Tasques finalitzades
Parquing	Validades	

Creación de la tabla para el método SCRUM

NullPointerException		
Tasques per fer	Tasques en curs	Tasques finalitzades
Parquing	Validades	
Plantilla menu	Salvare comunitat	Crear usuari
Integrar comunitat en la base de dades	Integrar usuari en la base de dades	Mostrar usuari en la base de dades

Imagen de la tabla a mitad de proyecto

Gracias a la metodología SCRUM hemos podido desde un principio definir las tareas en las que nos queríamos centrar, y nos ha sido útil para tener una idea general del proyecto y poder añadir nuevas ideas a lo largo de todo el proyecto.

Cabe añadir que gracias a la metodología SCRUM nos ha resultado muy sencillo realizar un seguimiento del proceso y orden de tareas. En este aspecto también ha contribuido otro tipo de metodología que hemos utilizado, cada dos semanas hemos realizado una reunión y un acta de la misma. Gracias a las reuniones programadas y las actas hemos sido capaces de dividir tareas y enfocarnos en los puntos en los que algún compañero necesitaba apoyo o en juntar nuevas ideas que aplicar al proyecto.

2- Proyecto

2.1- Idea inicial

La idea inicial del proyecto era crear una ciudad en la que al hablar con diferentes personajes cada uno te asignará una misión o una tarea que hacer por la ciudad. La idea inicial del proyecto era ambiciosa ya que queríamos tener un gran número de personajes, a su vez queríamos implementar un estilo de juego llamado aventura gráfica, en el que tienes todo tipo de libertad para moverte por el mapa y jugar la historia en el orden que decidas sin alterar la narrativa.

2.2- Temática

El videojuego está basado en el famoso meme de NPC Wojack, como se puede ver reflejado en el personaje principal y los personajes secundarios. La estética del proyecto es de estilo Low Poly, traducida al castellano por “bajo poligonaje”, es un estilo que afronta diseños con un bajo número de polígonos, es decir, elementos geométricos que tienen pocas caras o vértices, como las esferas, los cilindros o los cubos. Al adaptar este estilo a nuestro proyecto conseguimos

2.3- Historia

Una de las partes importantes del videojuego es la historia ya que se usa de hilo conductor para que el jugador vaya de una misión a otra. La historia ,aunque es algo corta, trata de cómo nuestro protagonista llega a un nuevo barrio y todas las cosas que tiene que hacer para poder instalarse en su nuevo hogar. Para ello tendrá que ayudar a algunos vecinos y ganar algo de dinero, además de tener la opción de conocer a algunos vecinos un poco peculiares.

Para las misiones nos basamos en escenas de la vida cotidiana, como hacer trabajos precarios, como repartidor, para pagar el alquiler. También existen escenas donde debes ayudar a gente que ha olvidado o perdido objetos, o incluso personas, como se puede ver reflejado en la misión de buscar las llaves o en la misión del laberinto, las cuales explicaremos más adelante.

2.4- Creación del mapa

Lo primero que hicimos fue decidir que tipo de mapa queríamos, si preferíamos un mapa único con todas las misiones en él, o un mapa principal y apartados diferentes para cada misión. Decidimos ir con la segunda opción ya que nos daba más libertad a la hora de crear misiones y modificar el terreno de juego.

Lo primero que hicimos fue el mapa principal donde estarían los NPCs y donde se le darían las misiones a nuestro personaje. Decidimos que sería un barrio cerrado con una sola calle, ya que para la historia que queríamos contar era la mejor opción.

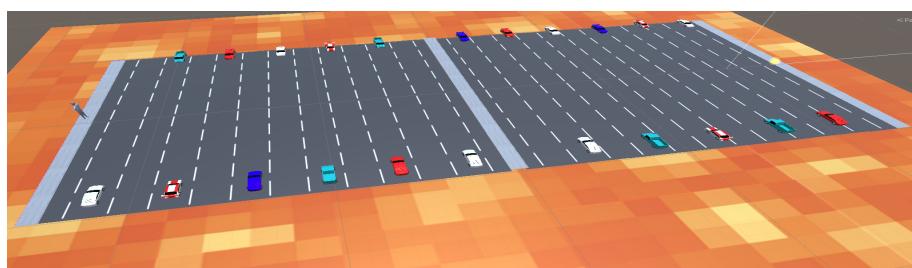
Encontramos unos [assets](#) gratuitos de diferentes edificios y con ellos desarrollamos el mapa principal.



Mapa principal del proyecto.

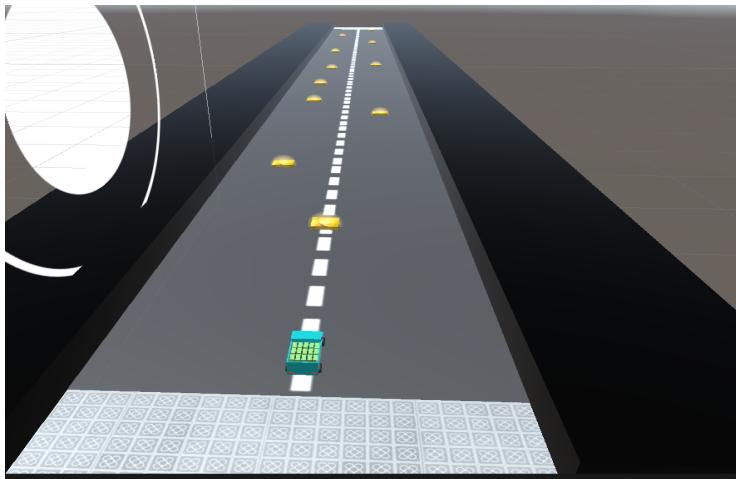
Con el mapa principal ya creado era momento de empezar a crear los mapas de cada misión.

Para la primera misión decidimos que fuera un minijuego con un objetivo único, cruzar la carretera para recuperar unas llaves, así que creamos un mapa con tres aceras, que servían como zona segura y diferentes carriles entre medio por los que pasarían coches a distintas velocidades. Estos coches al chocar con el jugador lo devuelven a la última acera en la que ha estado. También decidimos rodear el mapa con lava, simulando una zona límite del mapa al que no se puede ir más allá, la lava mata al jugador mandándolo al inicio del mapa.



Mapa de la misión de cruzar la carretera.

El siguiente mapa que creamos fue una misión simulando un reparto en furgoneta, creamos una carretera por la que teníamos que conducir pasando por los puntos de entrega hasta llegar al final, para que el coche no se cayera de la carretera levantamos dos muros y los ampliamos para cubrir parte de la pantalla y que no se viera el vacío.



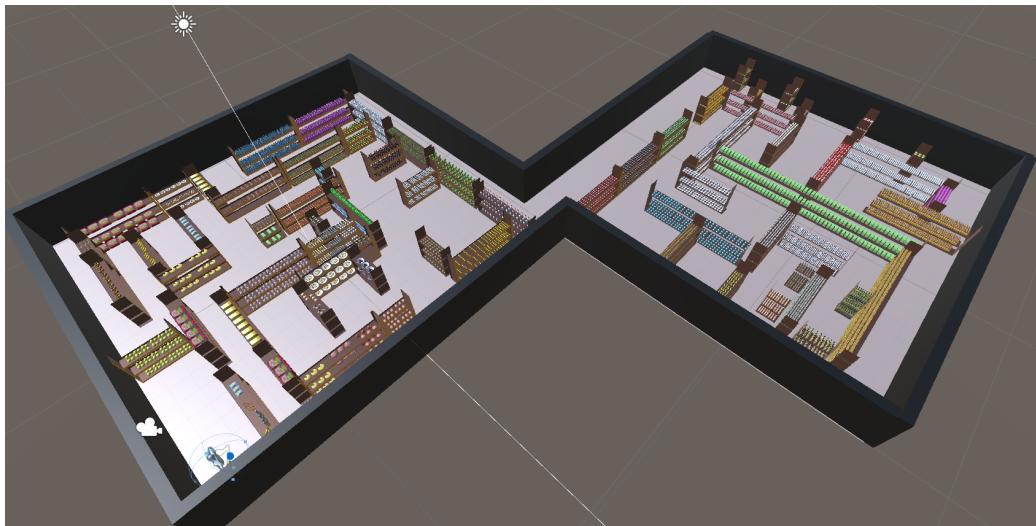
Mapa del minijuego de entregar comida

Y el último mapa que creamos fue el que llamamos “La misión del supermercado”, en la cual el jugador tiene que moverse por un supermercado, que resulta ser un laberinto. El objetivo de este nivel es encontrar a un niño perdido. Para ello creamos dos salas conectadas entre sí, creando una forma simétrica y las rellenamos de estanterías creando caminos por los que se moverá el personaje. Gracias a unos assets gratuitos pudimos llenar unas estanterías de comida para que simulara aún más un supermercado. El modelo de las estanterías está hecho por nosotros en el programa de modelado 3D [Blender](#).



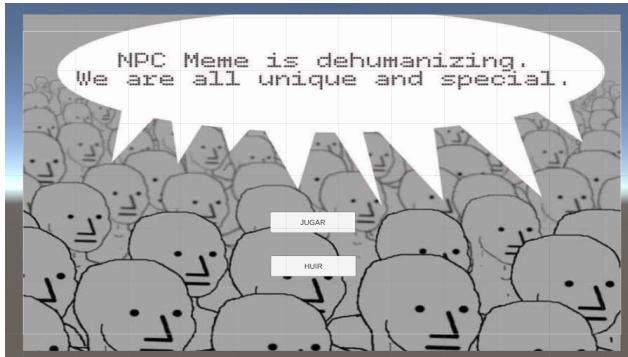
Estantería creada por nosotros en Blender

Juntando todos estos componentes quedó un laberinto con aspecto y temática de supermercado:

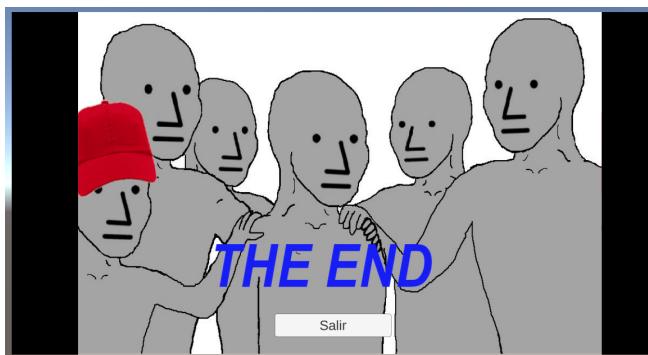


Mapa del juego “*La misión del supermercado*”

Con todo el mapa creado lo último que hicimos fue crear dos escenas extra, un menú principal mediante el cual entrar al juego y otro menú simulando unos créditos finales para salir de la aplicación al completar la historia.



Menú de inicio del juego

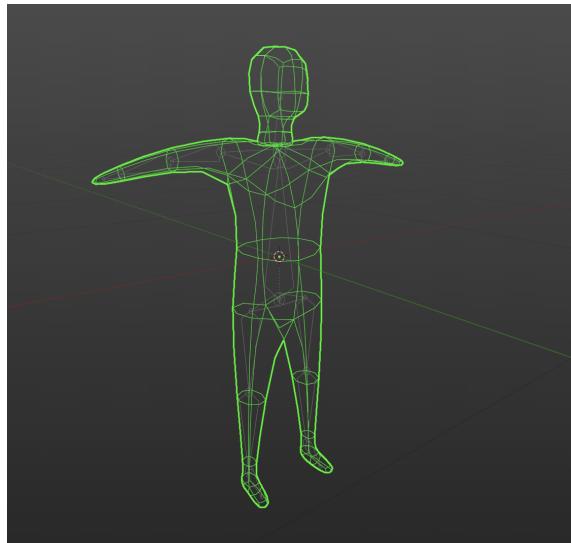


Menú fin del juego

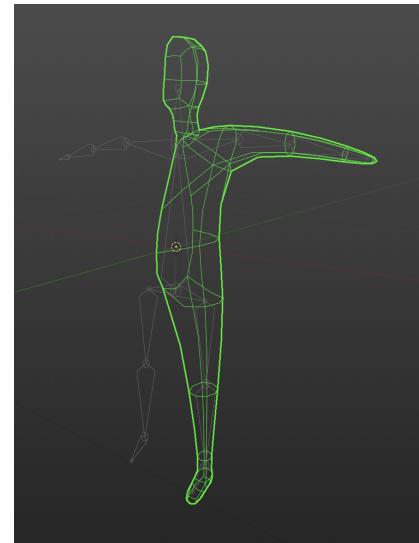
2.5- Modelo 3D jugador

Para crear el modelo 3D de nuestro personaje hemos decidido usar el programa Blender, ya que es un programa de código abierto y gratuito disponible para Windows, MacOS y Linux. Hemos escogido Blender básicamente porque es el único programa gratuito y que engloba todas las herramientas que necesitábamos para crear a nuestro personaje y exportarlo a Unity. Dentro del mismo Blender podemos a la vez crear un modelo, generar sus texturas y animarlo, entre muchas otras cosas.

Primero empezamos creando el modelo del personaje en sí, a partir de una forma básica como un cubo, usando las herramientas de extrude, transform y rotate. También le hemos agregado al personaje un modificador de simetría lo que nos permite modelar en espejo solo una mitad del personaje y no tener que hacer la otra mitad.



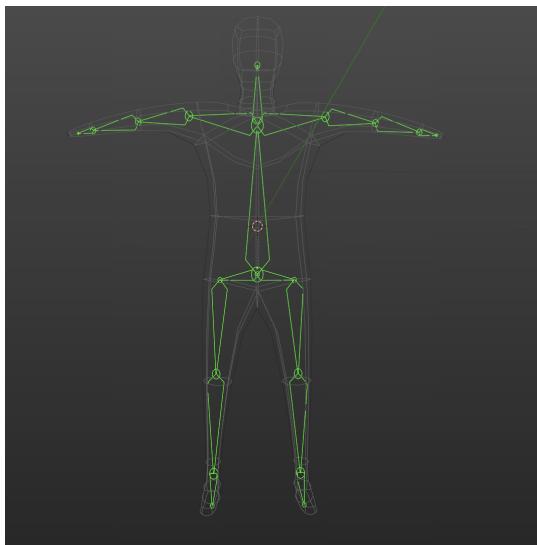
Modificador de simetría del personaje



Inicio creación del modelo

Para que Unity reconozca correctamente la forma humanoide del personaje el modelo tiene que estar formando una pose en T con los brazos estirados.

Una vez creado el modelo del personaje tendremos que crear un armature para que se pueda mover. El armature es básicamente una estructura de huesos que permite articular el modelo del personaje. Este procedimiento se conoce como [character rigging](#) en el desarrollo de videojuegos.



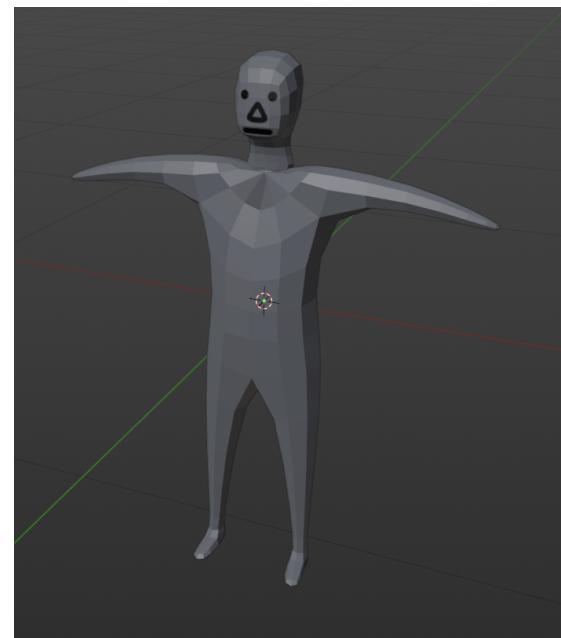
Armature del personaje

Es necesario que el armature también tenga una estructura parecida a la de un esqueleto humanoide, para que el mismo Unity la pueda reconocer automáticamente y pueda aplicarle bien las animaciones que crearemos para dicho personaje.

Una vez modelada la forma del personaje podemos crear sus texturas y previsualizar cómo se van a ver las mismas. Para dibujar las texturas crearemos un mapa de texturas que luego tendremos que agregar al personaje una vez lo hayamos exportado a Unity.



Mapas de texturas del personaje

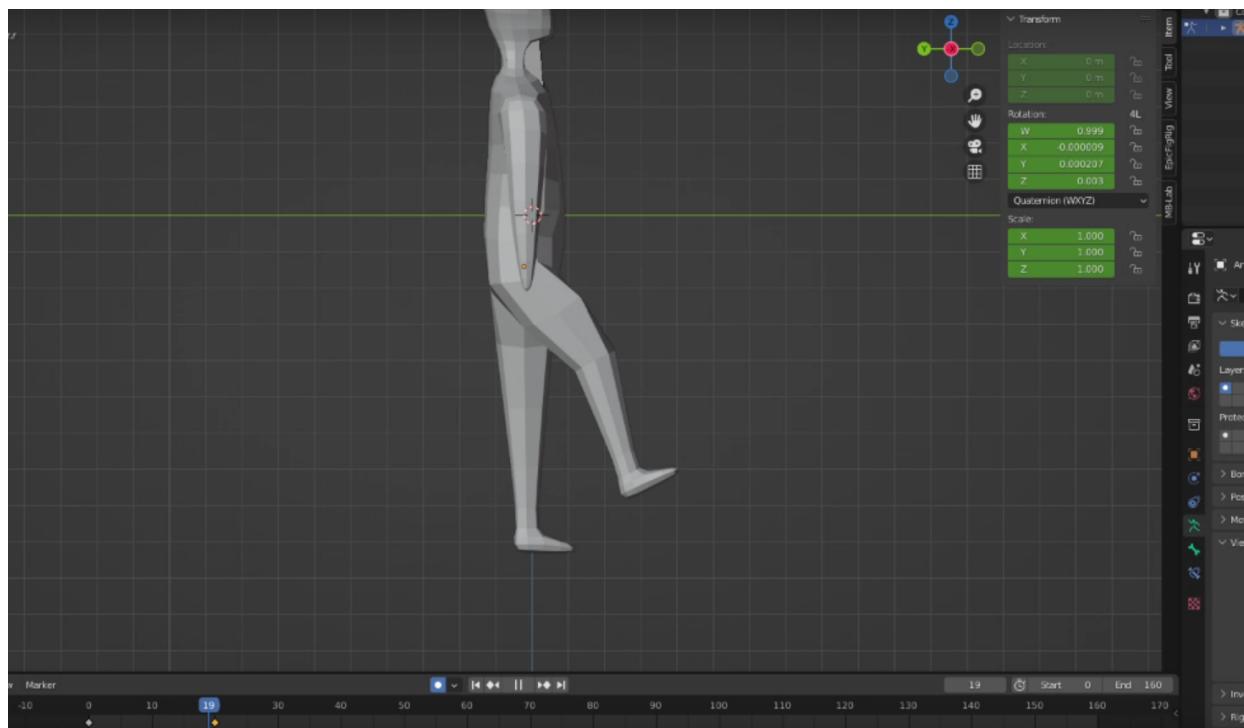


Personaje con texturas

2.6- Animaciones

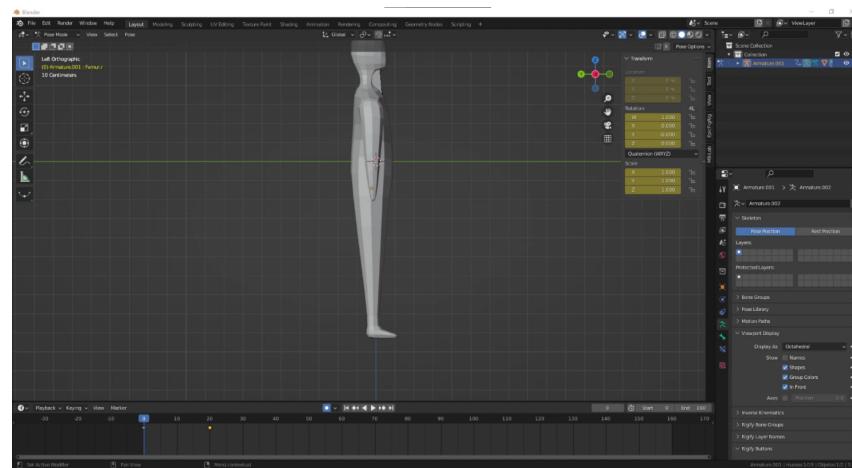
Usando el armature que hemos creado antes podemos generar animaciones para nuestro personaje. Las animaciones tendrán que durar 60 frames, ya que si no generarán problemas dentro del proyecto.

La primera animación que hemos creado es para que el personaje se mueva al andar por las diferentes escenas del proyecto.



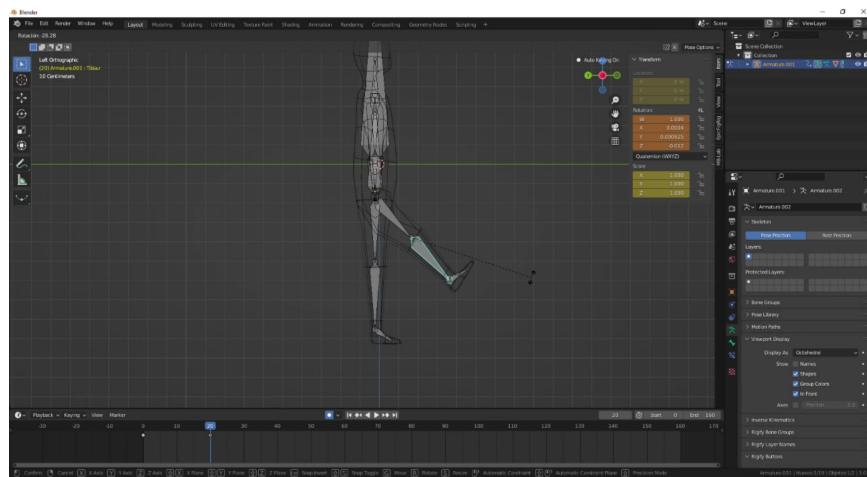
Animación del personaje andando

La segunda animación que hemos creado es para cuando el personaje esté quieto. Ya que si usaramos simplemente el modelo que hemos creado al principio el personaje estaría constantemente haciendo la misma pose en T estando quieto.

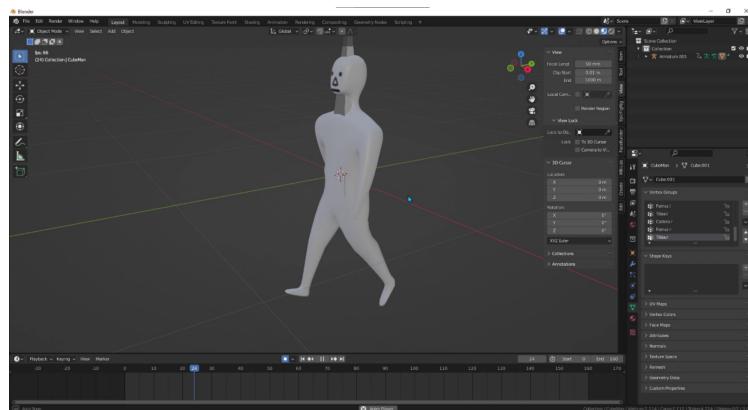


Animación del personaje quieto

Mediante el armature que hemos creado antes podremos rotar las diferentes partes de nuestro modelo. Cuando animamos en Blender usamos los keyframes para definir en donde se encontrará cada parte del cuerpo en cada frame de la animación, no es necesario que definamos los 60 keyframes de la animación. Si definimos los momentos más importantes de la animación como cuando las piernas están en la parte más alta, respectivamente, el programa generará el resto de keyframes automáticamente.



Definiendo los keyframes del personaje



Animación del Modelo 3D del personaje principal en Blender.

2.7- Scripts

Una vez hecho el mapa y el modelado 3D del personaje principal y secundarios, así como sus respectivas animaciones, el siguiente paso era encajar todas estas piezas a través de código y el propio Unity. Para eso hemos usado [scripts](#) en C# que es el lenguaje de programación que utiliza Unity.

2.7.1- Cámara

En Unity, cada escena dispone de su propia cámara e iluminación, para nuestro proyecto hemos usado un script específico para la cámara:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class CameraController : MonoBehaviour
6  {
7      public GameObject player;
8
9      private Vector3 offset;
10
11     //#[SerializeField]
12     //Transform[] Positions;
13
14     // Start is called before the first frame update
15     void Start()
16     {
17         offset = transform.position - player.transform.position;
18     }
19
20     // Update is called once per frame
21     void LateUpdate()
22     {
23         transform.position = player.transform.position + offset;
24     }
25 }
```

Script de Unity para el objeto cámara.

En este script se usa un GameObject y un Vector3. El GameObject en Unity sirve para referenciar un objeto dentro del juego, en este caso hace referencia al jugador. De este modo cuando se aplica el script a la cámara te pide que le references quien es el jugador principal, es decir el jugador “Player”. El Vector3 en Unity sirve para pasarle al código una posición 3D y una dirección. El script está ordenando en el módulo Start que el Vector3 sea igual a la posición del player, es decir, el jugador. En el módulo LateUpdate se le ordena a la cámara que siga desde su posición al jugador. De esta forma se crea una especie de seguimiento por parte de la cámara, en el que nunca estará ni más lejos ni más cerca del jugador, la cámara siempre estará a la misma distancia y ángulo.

2.7.2- Jugador

Una vez explicado cómo funciona la cámara en el videojuego es el momento de explicar cómo funciona el jugador principal, el que se controla durante gran parte del videojuego.

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.SceneManagement;

public class PlayerControllerMainScene : MonoBehaviour
{
    public float speed = 0;
    private Rigidbody rb;
    private float movementX;
    private float movementY;
    public GameObject jugador;
    [SerializeField]
    Transform[] Positions;
    Transform SpawnPos;
    Transform TextoPos;
    AudioSource audioData;
    private RigidbodyConstraints originalConstraints;
    public VariableJoystick variableJoystick;
    public Animator anim;
```

Encabezado del Script PlayerController, definición de objetos que usa el script.

Este script utiliza varios objetos y variables, todos ellos contribuyen a que el personaje principal funcione correctamente. En la captura anterior se aprecia la declaración de los objetos como el objeto jugador para referenciar al jugador mismo pero dentro del script y el objeto Rigidbody que sirve para aplicarle una fuerza y movimiento al personaje.

```
void Start()
{
    gameObject.tag = "Player";

    rb = GetComponent<Rigidbody>();
    originalConstraints = rb.constraints;

    SpawnPos = Positions[0];
    audioData = GetComponent< AudioSource >();

    anim = gameObject.GetComponent< Animator >();
}

void FixedUpdate()
{
    rb.velocity =
        new Vector3(variableJoystick.Vertical * speed,
                    rb.velocity.z,
                    variableJoystick.Horizontal * speed * -1);

    if (variableJoystick.Vertical != 0 || variableJoystick.Horizontal != 0)
    {
        transform.rotation = Quaternion.LookRotation(rb.velocity);
        anim.SetBool("Quieto", false);
    }
    else
        anim.SetBool("Quieto", true);
}
```

Script PlayerController, módulos Start() y FixedUpdate().

En el módulo Start() se le otorga al jugador un tag “Player” que usaremos posteriormente para las colisiones con otros NPC, traducido del inglés, personaje no jugador, este tipo de personajes son personajes secundarios de la historia y que no puedes controlar.

En este módulo se le otorga a nuestro jugador un Rigidbody el cual también usaremos después para aplicar sobre él una fuerza para poder mover nuestro jugador.

Por otra parte y en un plano secundario quedan dos líneas de código que refieren a que nuestro jugador tiene una SpawnPos, es una posición inicial que le hemos asignado a nuestro jugador, de esta forma cada vez que vuelva a la escena o necesite usar posiciones dentro de una escena se almacenarán dentro de este objeto. También se puede observar un audioData, es un archivo de audio asociado, en este caso nuestro jugador es el que lleva la música según la escena en la que se encuentre.

Por último en el módulo Start() se crea un objeto llamado anim, es el encargado de referenciar y organizar las animaciones de nuestro personaje.

En el módulo FixedUpdate() es dónde se controla a través del Rigidbody el movimiento del jugador y la rotación del mismo. De esta forma logramos que junto al joystick que aparece en pantalla para moverlo el personaje se mueva y mire hacia la dirección que apunta el joystick. Se puede observar que en este módulo también se usan unas variables Booleanas, es un tipo de variable que solo puede ser True o False, Verdadero o Falso. Junto a estas variables booleanas y el movimiento del Rigidbody se detecta cuando el personaje está quieto o en movimiento. De esta forma el personaje hace la animación de andar cuando se toca el joystick y la animación de estar quieto cuando no se toca el joystick.

Más abajo en el script hay un apartado para detectar colisiones con otros personajes o con objetos:

```
private void OnTriggerEnter(Collider other)
{
    switch (other.tag)
    {
        case "Mision1":
            SceneManager.LoadScene("Scene Mision1");

            //Ahora lo hace el npc
            break;
        case "coches":
            jugador.transform.position = SpawnPos.position; //mover jugador posicion inicial
            break;
        case "SuSuelo":
            jugador.transform.position = SpawnPos.position; //mover jugador posicion iniciales
            audioData.Play(0);
            break;
        case "CheckPoint":
            //check = true;
            SpawnPos = Positions[1]; //posicion Inicial jugador
            break;
        case "Finish":
            SceneManager.LoadScene("Main Scene"); // El nombre de la escena que desea cambiar
            break;
        default:
            break;
    }
}
```

Script PlayerController, apartado de detección de tags.

En este apartado ocurre la detección de tags. Un tag es una palabra que va enlazada a un objeto del proyecto, en nuestro caso hemos usado varios tags, por ejemplo el tag de “coches”, con este tag cada vez que nuestro personaje colisiona con alguien con ese tag le envía a la posición inicial del nivel, este tag es usado en la misión de cruzar la carretera.

Por último en este script está el apartado de los diálogos:

```
public void velocidadtextos()
{
    rb.constraints = RigidbodyConstraints.FreezeAll;
}

public void velocidadNormal()
{
    rb.constraints = originalConstraints;
}

public void TpMisionUno()
{
    SceneManager.LoadScene("Scene Mision1");
}

public void TpMisionDos()
{
    SceneManager.LoadScene("juego2");
}

public void TpMisionTres()
{
    SceneManager.LoadScene("MiniGame");
}

public void TpCreditosFinales()
{
    SceneManager.LoadScene("Creditos");
}
```

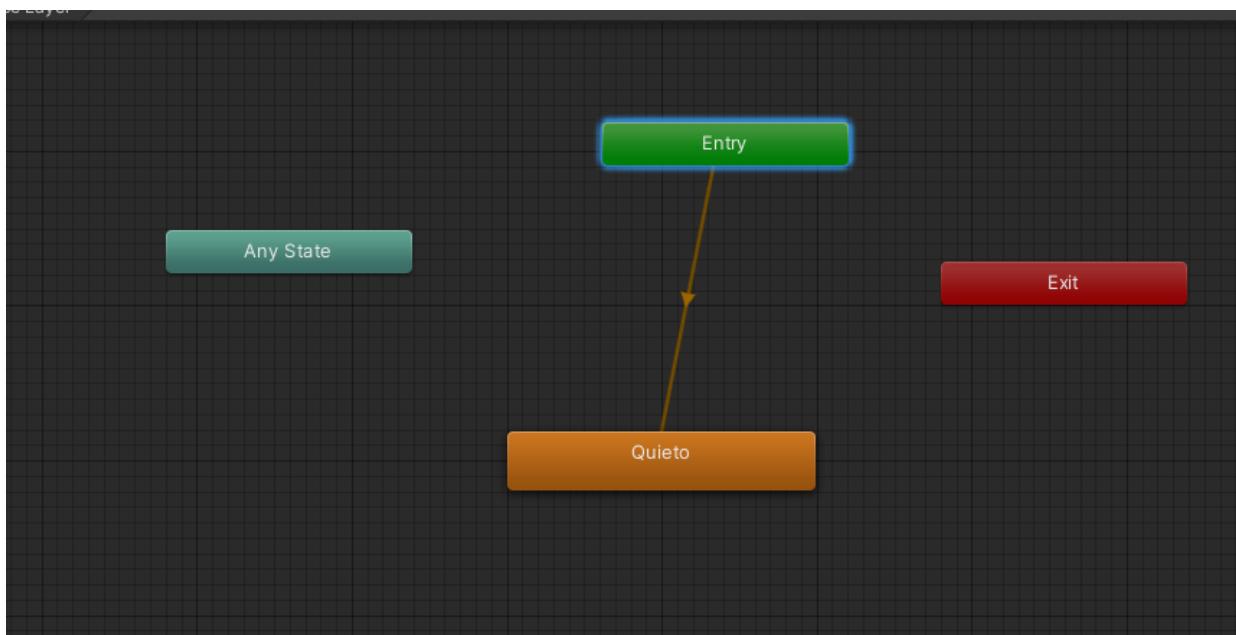
Script PlayerController, apartado de diálogos.

En esta parte final del script del jugador se utilizan módulos para realizar acciones como desactivar el movimiento del jugador, volverlo a activar o mover al jugador a una escena en específico. Este apartado va directamente conectado con los diálogos del videojuego.

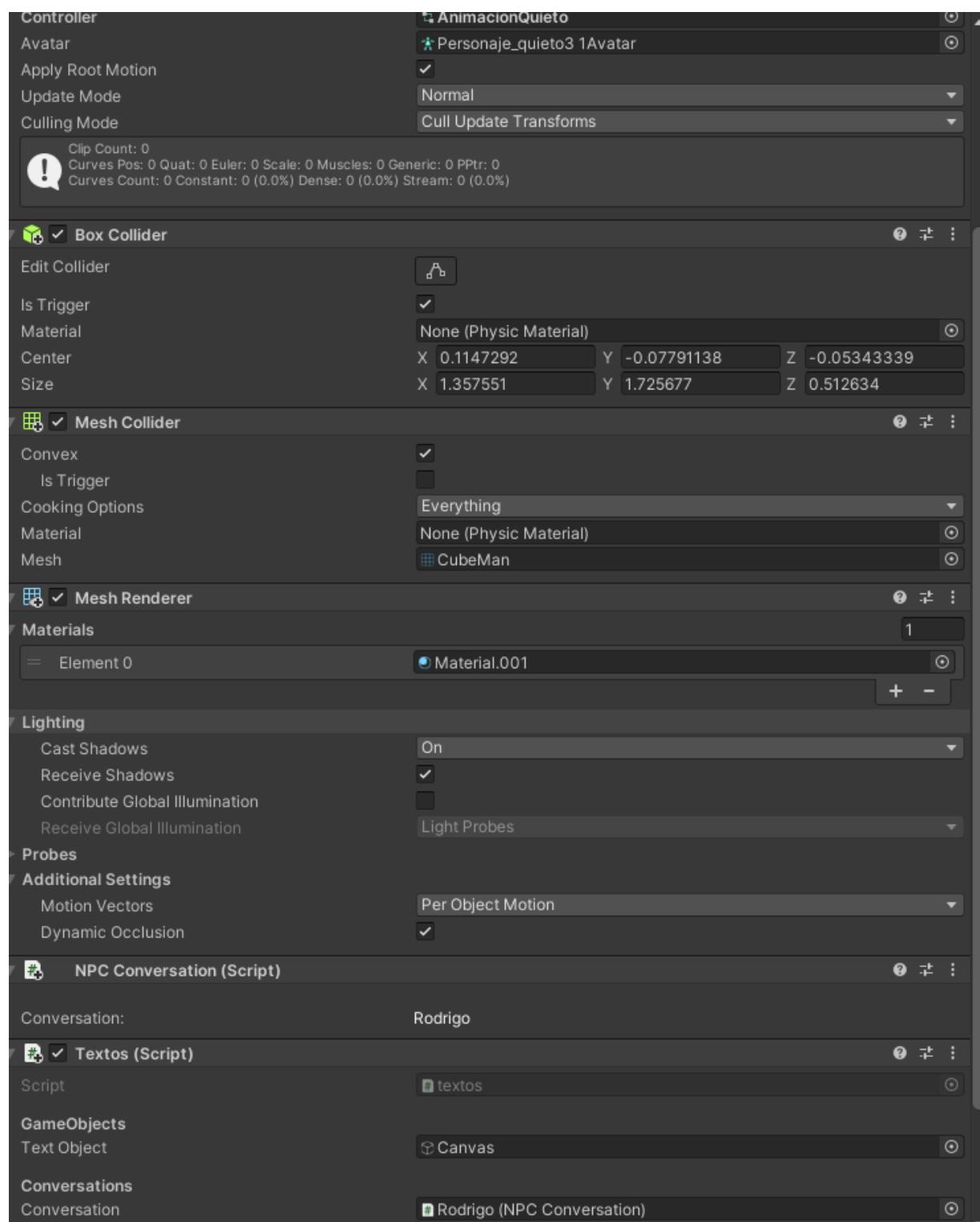
2.7.3- Diálogos

Con el jugador ya explicado ahora toca hablar de la creación de diálogos y los personajes que los usan.

Lo primero que hicimos fue poner los NPCs por el mapa, al usar el mismo modelo que el jugador principal lo único que se tuvo que cambiar fue la animación, para que solo hiciera la animación de estar quieto, y cambiar el rigidbody por un mesh collider y un mesh render.

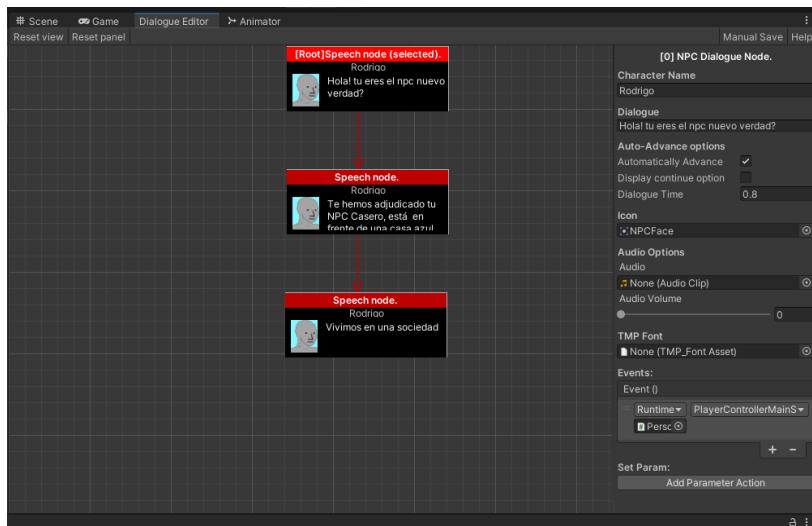


Animación de los NPCs

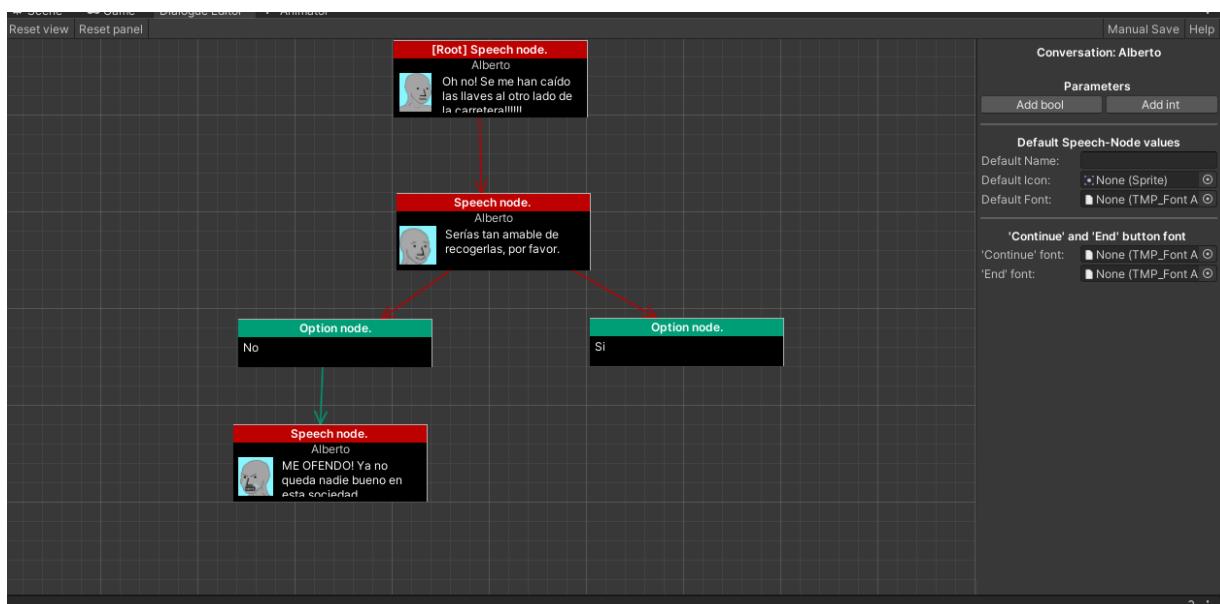


Componentes de los NPCs

Con los NPCs creados era momento de crear los diálogos, para ello usamos un asset de la tienda de Unity store con el cual se simplifica la creación de diálogos añadiendo una nueva ventana para crear las conversaciones, y unos fondos para los diálogos, todo con un código para controlar la creación de los cuadros.



Ventana de creación de diálogos



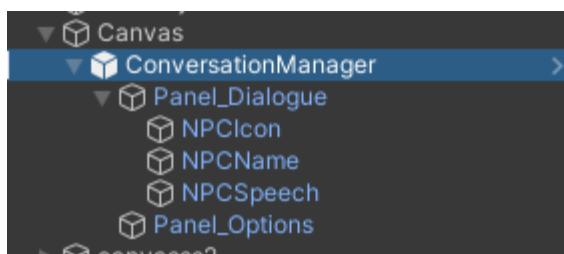
Ventana de creación de diálogos

Gracias al menú nos fue más fácil escribir los diálogos y sus opciones, y poder poner cuadros de texto en el canvas. El canvas es un espacio abstracto que sirve para poner todos los componentes del UI.

El cuadro de texto es un prefab con fondo, un par de cuadros de texto, uno para el nombre y otro para el diálogo del NPC, y una rawimage para poner la foto de perfil de cada NPC. Además hay un apartado para que salgan las opciones de respuesta.

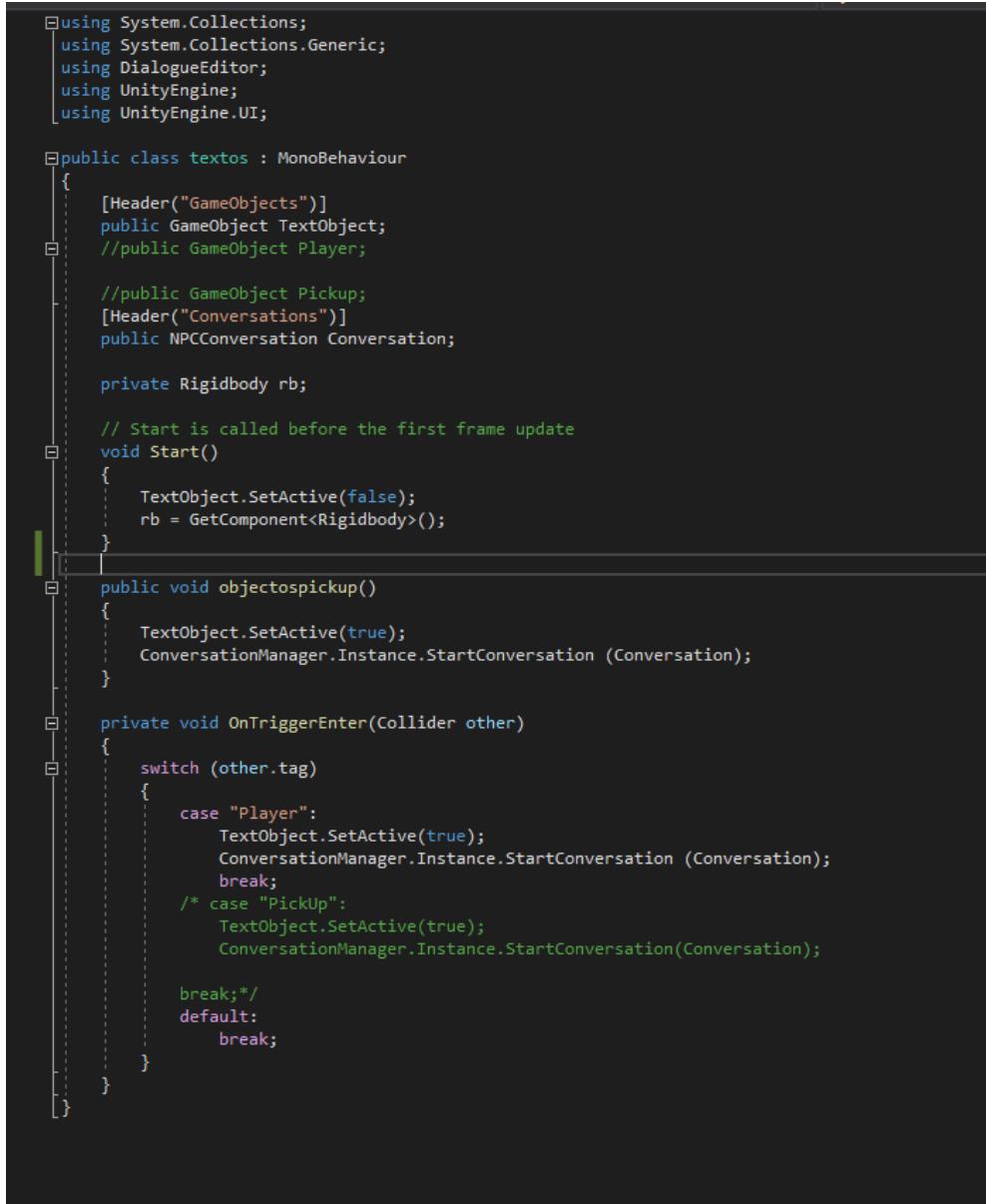


Canvas con el modelo de texto añadido



Partes del prefab de textos

Con los cuadros de diálogo hechos solo quedaba crear un script para controlar los diálogos, este script es corto, lo único que hace es definir en qué canvas quieres que aparezca el diálogo, que conversación ha de coger, y cuando se ha de activar dicha conversación, esta última parte está hecha con un OnTriggerEnter que detecta al jugador para activar el canvas en el que está el cuadro de texto y iniciar el diálogo de ese NPC.



```
using System.Collections;
using System.Collections.Generic;
using DialogueEditor;
using UnityEngine;
using UnityEngine.UI;

public class textos : MonoBehaviour
{
    [Header("GameObjects")]
    public GameObject TextObject;
    //public GameObject Player;

    //public GameObject Pickup;
    [Header("Conversations")]
    public NPCConversation Conversation;

    private Rigidbody rb;

    // Start is called before the first frame update
    void Start()
    {
        TextObject.SetActive(false);
        rb = GetComponent<Rigidbody>();
    }

    public void objectospickup()
    {
        TextObject.SetActive(true);
        ConversationManager.Instance.StartConversation (Conversation);
    }

    private void OnTriggerEnter(Collider other)
    {
        switch (other.tag)
        {
            case "Player":
                TextObject.SetActive(true);
                ConversationManager.Instance.StartConversation (Conversation);
                break;
                /* case "PickUp":
                    TextObject.SetActive(true);
                    ConversationManager.Instance.StartConversation(Conversation);

                    break;*/
            default:
                break;
        }
    }
}
```

Código para controlar los diálogos de los NPC

Con las conversaciones hechas solo quedaba usar el menú de los diálogos para iniciar otra escena y un par de códigos que paraban el movimiento del personaje mientras habla con un NPC y le devuelven el movimiento al jugador al finalizar la conversación.

```
public void velocidadtextos()
{
    rb.constraints = RigidbodyConstraints.FreezeAll;
}

public void velocidadNormal()
{
    rb.constraints = originalConstraints;
}

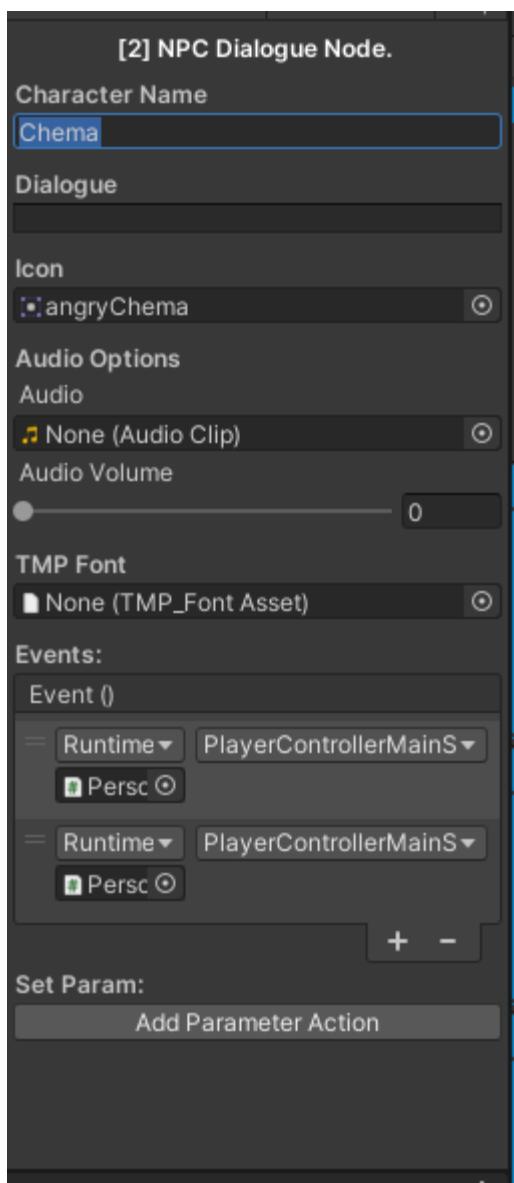
public void TpMisionUno()
{
    SceneManager.LoadScene("Scene Mision1");
}

public void TpMisionDos()
{
    SceneManager.LoadScene("juego2");
}

public void TpMisionTres()
{
    SceneManager.LoadScene("MiniGame");
}

public void TpCreditosFinales()
{
    SceneManager.LoadScene("Creditos");
}
```

voids almacenados en el jugador para cambiar de escena y para controlar el movimiento del personaje



Menú de diálogos para controlar el movimiento y
controlar pasar a otra escena

2.7.4- Movimiento de los Coches

En la primera misión, la de cruzar la carretera, los coches cruzan la carretera por los carriles en cambio el jugador lo hace atravesando los carriles, lo que provoca que a veces el coche colisione con los coches. También los coches al llegar al final del asfalto son teletransportados al inicio de su carril, todo ello se controla a través del siguiente script:

```
// Start is called before the first frame update
void Start()
{
    // Instantiate(coches, new Vector3(InPosIndexX,0,InPosIndexZ), Quaternion.identity);
    //createGameObject();
    nextPos = Positions[0];//posicion final coche
    inPos = Positions[1];//posicion inicial coche
    audioData = GetComponent< AudioSource >();
}

// Update is called once per frame
void Update()
{
    MoveGameObject();
}

//mover coches
void MoveGameObject()
{
    if (transform.position == nextPos.position)
    {
        nextPosIndex++;

        if (nextPosIndex >= Positions.Length)//coche llega posicion final
        {
            nextPosIndex = 0;
            coches.transform.position = inPos.position;//mover coche posicion inicial
            //Destroy(coches.gameObject);//eliminar coches
        }

        nextPos = Positions[0];
    }
    else
    {
        //movimiento coche
        transform.position = Vector3.MoveTowards(transform.position, nextPos.position, objectSpeed * Time.deltaTime);
    }
}
```

Script movimiento de los coches, misión cruzar la carretera.

En el módulo Start() del script se asigna una posición inicial al coche y una posición final, también se le asigna un audioData, un archivo de sonido que sonará al colisionar con el tag “Player”.

En el módulo MoveGameObject() es dónde ocurre el movimiento del coche. Al iniciar la escena el coche aparece en su posición inicial y se moverá él sólo hacia adelante, cuando colisione con su posición final será movido a su posición inicial de vuelta. De esta forma se crea una ilusión de que en ese carril siempre hay un coche cruzando.

En la misión de conducir una furgoneta para realizar entregas es necesario aplicarle al coche un script, con él se puede controlar su movimiento:

```
void OnMove(InputValue movementValue)
{
    Vector2 movementVector = movementValue.Get<Vector2>();
    movementX = movementVector.x;
    movementY = movementVector.y;
}

void SetCountText()
{
    countText.text = "Entregas: " + count.ToString();
}

void FixedUpdate()
{
    Vector3 direction =
        Vector3.forward * variableJoystick.Horizontal +
        Vector3.left * variableJoystick.Vertical; //cambio joystick
    rb
        .AddForce(direction * speed * Time.fixedDeltaTime,
        ForceMode.VelocityChange); //cambio joystick
}

private void OnTriggerEnter(Collider other)
{
    switch (other.tag)
    {
        case "PickUp":
            other.gameObject.SetActive(false);
            count = count + 1;
            SetCountText();

            break;
        case "Finish":
            SceneManager.LoadScene("Main Scene");
            break;
        default:
            break;
    }
}
```

Script movimientoCoche, que actúa sobre la furgoneta de la misión de entregas.

En el módulo OnMove() se almacena en un Vector2 el movimiento horizontal y vertical del coche, que luego es usado para mover junto al joystick y el Vector3 en el módulo FixedUpdate(). De la misma forma que con el jugador todas las fuerzas son aplicadas al Rigidbody, dependiendo de la dirección del joystick.

Por otra parte en el script hay dos módulos más, el módulo SetCountText() es el encargado de mostrar por pantalla el número de entregas hechas durante la misión, el número de entregas saldrá a través del Canvas. El último módulo, OnTriggerEnter() hace la misma función que en el script del jugador, cuando el coche colisiona con un objeto con ese tag, en este caso cuando colisiona con el objeto que tiene el tag "Finish", manda al jugador a la escena principal, es una forma de finalizar la misión.

2.7.4- Cambio de Escenas

Por último en el apartado de scripts está el script encargado de cambiar de escenas y manejar las escenas en el proyecto:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneChange : MonoBehaviour
{
    public void MoveToScene(int sceneID)
    {
        SceneManager.LoadScene (sceneID);
    }

    public void Salidura()
    {
        Debug.Log("He salido");
        Application.Quit();
    }
}
```

Script SceneChange encargado del control de las escenas del proyecto.

Este script se encarga de obligar al proyecto a cargar la escena del menú justo al iniciar la aplicación, también se encarga del botón de Salir de la aplicación que se encuentre en el menú principal y en los créditos finales.

3- Conclusión

3.1- Resultado Obtenido

Comparando la idea inicial con el proyecto final que hemos obtenido podemos decir que hemos superado las expectativas iniciales con creces. Quisimos hacer una ciudad en la que poder interactuar con distintos personajes y que cada uno tuviese su respectiva misión. Como hemos explicado anteriormente hemos conseguido tener una ciudad por la que poder moverte y explorar, personajes con los que poder interactuar con un mecanismo de diálogo, y una animación propia para nuestro personaje.

Aunque nos ha costado hemos conseguido que todo el videojuego funcione correctamente, el único problema que nos hemos encontrado es un bug que puede ocurrir al exportar el trabajo para plataformas android, en el cual el juego deja de usar las tags de los objetos, con lo que nuestro personaje no puede interactuar con el resto de NPCs y no puede acceder a las misiones. Si el juego se exporta en .exe dichos problemas no suceden. Tras investigar por internet hemos descubierto que es algo que lleva pasando en Unity desde 2015, o al menos esa es la referencia más antigua que hemos encontrado a nuestro problema. Y tras probar los métodos que a algunos usuarios sugieren, los cuales a ellos sí les había solucionado el problema, no hemos sido capaces de solventar este error.

3.2- Valoración

A pesar de las dificultades que hemos encontrado durante el desarrollo del proyecto hemos sido capaces de en un periodo de 3 meses aprender y crear un videojuego desde cero.

Este proyecto nos ha mostrado por dentro un mundo tan complejo como es el de los videojuegos. Desde pequeños hemos jugado a videojuegos y nunca nos hubiésemos imaginado la dificultad que conlleva crear uno. Gracias a este proyecto hemos podido ver cuales son las fases de creación de uno mismo y todo su desarrollo, desde ser una simple prueba individual, a juntar escenas de otros compañeros del grupo. A pesar de habernos repartido el trabajo en ocasiones siempre hemos logrado mantener un equilibrio y no crear una sobrecarga de trabajo a algún miembro del grupo, gracias a la planificación y organización del proyecto que nos ha ayudado a mantener siempre un orden de prioridades y control sobre en qué tarea debíamos centrarnos como grupo. De esta forma ha acabado saliendo un gran proyecto del que estamos muy orgullosos, hemos aprendido mucho sobre el proceso de creación de un videojuego, gestión del tiempo así como organización de un grupo de trabajo. Creemos que para futuros proyectos podríamos mejorar algunos aspectos, la optimización del tiempo es algo que no tuvimos en cuenta y nos ha afectado a pesar de que creímos que lo teníamos bajo control. Nos ha ocurrido durante el proyecto que en múltiples ocasiones podríamos haber reutilizado parte del código que ya teníamos creado o podríamos haberlo adaptado al aspecto nuevo que intentábamos implementar. A pesar de estas dificultades siempre hemos sabido desenvolvernos bien juntos.

Respecto al uso de Unity hemos demostrado que no ha sido un trabajo muy complicado, es un motor de videojuegos resolutivo ya que dispone de foros en los que poder consultar cualquier duda, a pesar de algunas dificultades durante el proceso hemos sabido salir hacia adelante y encontrar una solución.

Como alumnos la valoración que podemos hacer sobre el proyecto de Unity es positiva, así como la experiencia que hemos adquirido aprendiendo en el entorno de Unity y adaptandonos a un nuevo lenguaje como es C#.

5- Webgrafía

- [Unity Documentation](#)
- [Unity Asset Store](#)
- [Blender](#)
- [playonloop](#)
- [AccuRig](#)
- [NPC Wojack](#)

6- Anexos

- **Aventura gráfica:** Es un género de videojuegos de mundo interactivo, es el género sucesor de las aventuras conversacionales.
- **NPC Wojack:** es una versión del personaje Wojack el cual tiene la mirada fija y ninguna expresión facial.
- **NPC:** Son las siglas en inglés de Non Playable Character, o en español personaje no jugable, en un juego son aquellos personajes que no son controlados por el jugador.
- **Script:** Un script es un documento de texto donde colocamos instrucciones u órdenes que luego serán ejecutadas por el programa. Estas instrucciones están escritas en algún lenguaje de programación. El conjunto de scripts de un programa conforma el código fuente de la aplicación.
- **Asset:** Un asset es un archivo externo a Unity, como un modelo 3D, un archivo de audio o una imagen. Es una referencia a los recursos que utiliza un videojuego y que forman parte de él al momento de su creación.

- **Blender:** es un programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, la animación y creación de gráficos tridimensionales.
- **Character rigging:** Es una técnica usada en la animación de esqueletos donde añades controles al modelo.
- **Keyframes:** Son los fotogramas que se toman como referencia con el fin de ahorrar espacio de almacenaje.
- **Scripts:** es una secuencia de comandos u órdenes que se van ejecutando de manera secuencial.