

Trabajo de fin de grado: Desarrollo de un videojuego de lucha en Unity

Desenvolupament d’un videojoc de lluita en UnityDevelopment of fight video game in Unity



ENRIC bONET CORTÉS

UNIVERSITAT POLITÉCNICA DE VALENCIA

***Resumen***

Este proyecto consistirá en el desarrollo de un videojuego, mayoritariamente 2D, de lucha utilizando para ello la herramienta o motor de desarrollo Unity. El formato del videojuego será de navegador, pensado para que pueda ser accesible mediante el portal de videojuegos minijuegos.com.

El videojuego consistirá en un juego de peleas en el cual los personajes, manejados por los jugadores, lucharán en peleas de uno contra uno, haciendo uso de las diferentes habilidades especiales que tendrán los personajes. El objetivo será debilitar al otro jugador y dejarlo fuera de combate para así lograr la victoria.

Para el desarrollo del mismo, contaré con diferentes herramientas usadas frecuentemente en este campo.

Unity será la plataforma encargada del desarrollo mayoritario del videojuego, implementando así la lógica de los personajes y la interacción personaje-jugador con la ayuda de la plataforma de programación Mono. En la fase de modelado, tanto de personajes, se utilizará Blender, herramienta que además nos ayudara para detallar los escenarios, que mayoritariamente se desarrollaran en Unity.

Para la realización del videojuego, utilizaré el conocimiento adquirido en diferentes asignaturas de este curso, que se centran en este campo, como: Introducción a los sistemas gráficos interactivos (SGI) e Introducción a la programación de videojuegos (IPV). Además de estas asignaturas, también me ayudaré del curso de “Introducción al desarrollo de videojuegos con Unity” de la plataforma edX.

La redacción del proyecto consistirá en un seguimiento de las diferentes fases del videojuego, en las que se irán explicando detalladamente cada una de estas, haciendo referencia al videojuego final.

Finalmente, destacar que se consideran otras implementaciones al proyecto como un comportamiento inteligente para los personajes de manera que se pueda jugar en solitario contra la máquina, o en otro formato de peleas más amplio, como combates de cuatro jugadores.

***Palabras Clave***

Unity-Videojuego-Lucha-2D-Blender-Minijuegos-Multijugador-Programacion

# Explicación Primera reunión TFG

Unity 🡪WebGL

**Recomendaciones**

**Primero: Diseño de los elementos** que componen el juego (pjs, contexto, objetivos…)[Organizar un diagrama de flujo]:

* Elementos sonoros
* Especificación del funcionamiento general
* Diseño de programación (lenguaje, metodología, …)

**Planificación:**

Importante planificar hitos para, por ejemplo, los niveles: acabas el primero, lo pruebas, pasas al siguiente, etc.

**Recomendación**

Interfaz🡪 Escenario🡪 Personajes Principales🡪 Secundarios🡪 nivel

**Producción:** aquí ya con las tareas organizadas, empezar a realizarlas, arriba esta un buen orden para seguir. Poco a poco se van haciendo tareas y completando hitos.

**Pruebas:** corrección de los errores de programación y **MEJORA DE LA JUGABILIDAD**

* Pruebas Alpha: hechas por el programador
* Pruebas Beta: realizadas por futuros jugadores

**Memoria:** Ir escribiendo mientras haces, no hace falta que sea la redacción propia de la memoria, sino una plantilla en sucio que puedas pegar luego en la memoria final.

Estado del arte OBLIGATORIO.

Anexo->Manual de usuario.

# Borrador TFG - Victory Royale

(14-02-2017) Lo primero de todo, realizar el diseño del videojuego en general para definir las ideas base del proyecto. Para ello realice un StoryBoard a lápiz que me sirviera de apoyo para ir avanzando en el proyecto.

(21-02-2017) **[Menu principal]** Una vez ya en producción, diseñé el menú principal con la ayuda del curso de Introducción a la programación de videojuegos en Unity de edX y el video 1. Primero creo el fondo con elementos de Unity básicos, y a este escenario, le añado un script de Unity para que rote continuamente sobre el eje Y para dar una sensación de ambiente. Ahora paso a organizar los elementos del menú interactivo, básicamente el canvas, al cual le añado un panel que lo hacemos translucido de color negro, superpuesto al fondo, de modo que se vea centrado en todo momento. Añado una Image al canvas para poner una foto de prueba que posteriormente será sustituida por la imagen/logo del videojuego. Tras esto empecé a colocar los diferentes botones ajustándolos y colocando el texto idóneo para cada uno.

Hecho esto, el menú carece de funcionalidad por lo que paso a realizar nuevas escenas de Unity (vacias por el momento) para enlazarlas a los diferentes botones, y así, dejar el primer hito, el Main Menu, completado a falta de editar su estado del arte. Antes de realizar el script que enlace estos botones, añado estas escenas creadas desde Build Settings al proyecto, para que Unity sepa que debe de utilizarlas. Una vez hecho esto, en la escena del menú, nos creamos un objeto que se encargara de tener el script que selecciona las diferentes escenas en función al botón que se pulse, por lo que fabricamos este script y se lo añadimos a este objeto. En este es script simplemente encontramos un método que se encarga de lanzar la escena que se le pase como parámetro string, por lo que una vez creado el script y asignado al objeto, modificamos desde la interfaz de Unity el método onClick() de los botones, para que llamen a este método con la vista adecuada para cada botón. Y así se concluye la primer hito del proyecto, el Main Menu. (2h para realizarlo)

(05-03-2019) **[Selección de personajes]** Tras tener el menú principal, una de las cosas que más me hacía ilusión del juego es realizar el menú de selección de personajes. Dicho esto, apoyándonos en el Video 2 montamos una primera escena sencilla para el menú de selección. Este video muestra como montar la lógica del menú de selección de personajes, lo cual me sirvió de base para tener el primer prototipo de menú de selección de personajes, que al igual que con el menú principal, lo completaría más tarde con el estado del Arte que eligiese. La escena constaba de unas simples capsulas dispuestas en triangulo encima de un plano con la cámara enfocando a la que se situaba en el centro.

Tras organizar esta escena, empezamos a programar el script que irá asociado al objeto Selección, un objeto vacío que contiene a las otras capsulas. Su función será la de girar, girando de este modo también a las capsulas, reconociendo las flechas de dirección o el clásico A/D. Cuando se reconocen las pulsaciones de estas teclas, el ángulo del objeto Selección se modifica en 90 grados pues hemos dispuesto 3 capsulas inicialmente en triangulo. Hay que destacar de este script, el uso de Mathf.LerpAngle, una función que te permite transitar de un ángulo inicial a otro final, en un determinado tiempo. Con la introducción de los personajes reales del juego posiblemente el script se modifique para adaptarse al número de personajes que tenga el juego.

(06-03-2019) Para finalizar la escena, he introducido a las capsulas un material básico con un color para cada una para diferenciarlas.

Tras tener la escena operativa, me dispongo a colocar los elementos básicos de la interfaz de usuario. Esta interfaz gráfica, se compone de un botón que te permite volver atrás, un texto que varía según el personaje que estemos visualizando para seleccionar, y el botón que te permite pasar a la pantalla de juego, seleccionando el personaje escogido. Este botón solo se puede pulsar cuando el personaje se sitúa enfrente tuya, es decir, en transición se deshabilita.

**[Creación de GameScene y su IU]**Por último, para finalizar el menú de selección, creo una nueva escena vacía para enlazarla a través del botón de selección de personaje (anteriormente mencionado). De esta forma me dejo como base la escena de la partida. En esta escena, he colocado además una interfaz gráfica para poder volver atrás y realizar pruebas. Esta interfaz consiste en un botón que muestra un panel (inicialmente oculto) con una serie de botones, inicialmente: *“Resume”*, que te permite volver a la escena, *“Back to select”*, que te devuelve al menú de selección de personajes, y *“Back Title”*, que te devuelve al Main Menu. Para controlar esta funcionalidad, he creado un objeto vacío al que coloco un script que controla las diferentes funciones del menú.

(12-03-2019)**[Escenario y nombres de los pjs]** Con el menú de selección hecho, y enlazado con la escena del videojuego me dispongo a realizar el escenario del juego. Lo primero de todo, coloqué un plano que me serviría de suelo para el juego. Tras esto, me descargué un skybox para personalizar mi mapa del siguiente enlace: <http://www.custommapmakers.org/skyboxes.php>. Genere un nuevo material de tipo skybox y asigné cada imagen a su lugar en la skybox. También generé un material al que le asigné la imagen down del skybox para hacer el suelo a conjunto con el skybox.

A falta de implementar los personajes, empecé a añadir de nuevo elementos a la interfaz de usuario, esta vez, dos textos. Uno, el de la derecha, haría referencia al villano y el otro se modificaría dependiendo del personaje elegido por el usuario. Hecho esto, me dispuse a crear los métodos que permitiesen persistir dichos nombre a través de ejecuciones. Para esto, utilizo las funciones de PlayerPrefs, SetString y GetString para guardar el texto que aparece en el menú de selección, y mostrarlo en la interfaz del GameScene.

(15-03-2019)**[Ethan]** A partir de lo realizado anteriormente, una vez hecha la escena del juego, empecé a implementar personajes jugables en el juego. Como ya he explicado en el apartado del menú de selección, el menú lo he diseñado siguiendo el Video 2 de las referencias por lo que inicialmente tiene espacio para 3 personajes elegibles. El primero de todos es el más difícil de implementar, pues es el que debes incluir desde cero, pues no disponía de ningún script para el movimiento del personaje ni nada por el estilo. Por ello, elegí como modelo de mi primer personaje a Ethan, un modelo 3D que se encuentra en el Standard Assets de Unity y con el cual estoy familiarizado al trabajar con él en proyectos personales y en la asignatura Entornos de Desarrollo de videojuegos, asignatura que aprovechando su mención, añadiré que fue de gran utilidad para formar las animaciones y el control de tanto este primer personaje, como de los que le siguen.

Entrando ya en materia, lo primero que hice fue importar el modelo de Ethan y sus animaciones del paquete que ya he mencionado antes, Standard Assets.

Añado a Ethan a la escena y le añado un componente Character Controller de Unity, el cual ajusto adecuadamente a las medidas del personaje. Lo siguiente es, tal vez, lo más complicado de la implementación del personaje: añadir el script que permite controlar el movimiento del personaje. Lo primero del movimiento, es controlar la dirección y el movimiento en tierra, lo cual el jugador podrá realizarlo por medio de A/D (izquierda/derecha) o con las flechas para moverse a través del eje x. Como el juego se realiza en 2D, no hay movimiento en el eje z. Luego de captar el movimiento, me centro en la rotación del personaje según la dirección que toma el jugador. Esto lo realizo mediante el método LookRotation de la clase Quaternion de Unity, al cual le paso el vector de movimiento del personaje como parámetro. Por último, añado para el personaje la acción de saltar. Esta acción es controlada por el usuario por medio de las teclas “W”, “Espace” o la flecha de dirección UP. Para implementarla, utilizo diferentes variables que pueden influenciar en dicha acción, estas son: la velocidad de salto, que determinará la potencia de salto del jugador, y otros parámetros que ayudan a dar una sensación de realidad al salto (como son la gravedad, la velocidad terminal o la caída mínima).

Una vez el personaje ya tiene movimiento, el siguiente paso es incluirle las animaciones. Para ello creo un nuevo Animator Controller, y en la ventana de Animator le añado las tres animaciones necesarias que va a tener de momento Ethan, el salto, correr y estar de pie. Estas animaciones transitaran de unas a otras con la ayuda de dos parámetros que he definido, Speed (Float) y Jumping (Bool). Estos parámetros son modificados en el script de movimiento del personaje, por ejemplo, cuando el jugador pulsa espacio Jumping pasa a ser True, pasando del estado que estuviese, al de salto.

De momento, el persoanje solo consta de tres animaciones para el movimiento. Posteriormente, se le introducirán las animaciones de golpes.

Por último, y para finalizar la inclusión de Ethan, me dispongo a enlazar el personaje en el menú de selección. Para ello, primero convierto a Ethan en un prefab, y lo elimino de la escena. Creo un objeto vacío GameController al cual añadiré un script nuevo con el mismo nombre para que se ocupe de instanciar el personaje según ha sido elegido desde su prefab.

En el menú de selección, sustituyo una de las capsulas iniciales para que aparezca Ethan, el cual lo introduzco en la escena a partir de su prefab(deshabilitando el Character Controller y el script de movimiento). También modifico el script que controla la interfaz de usuario para que aparezca el nombre de Ethan sobre su cabeza, de esta forma, el texto que guardamos cuando seleccionamos a Ethan es el nombre de este, y luego el GameController con un if con un if, controla que sea Ethan y los instancia.

(20-03-2019)**[Animaciones de golpes para el personaje]** Para esta primera implementación de los golpes del personaje, he decidido utilizar las animaciones del package FigthingMotinosVolume1, que se encuentra en la Asset Store de Unity. De momento solo he implementado dos animaciones, una para una patada alta y otra para un puñetazo.

Para incluirlas en el personaje, arrastro las animaciones al animator del personaje para introducirlos como nuevos estados, Kick y Hoop. Estos estados se acceden tanto por el estado de correr HumanWalk, como por el de estar quieto HumanIdle. Para transitar a los estados Kick y Hoop, utilizo dos parámetros booleanos con los mismos nombres que, a través del mismo script del movimiento del personaje, cambio a True o a False según me convenga.

En el caso de ponerlo a True, se hace mediante el reconocimiento de la tecla “J” para el caso de Kick, y la tecla “K” para el caso de Hoop.

También utilizo dos variables booleanas con el mismo nombre, esta vez declaradas en el mismo script, para impedir que se pueda avanzar en el eje X o pulsar la otra acción en el caso de que la acción anterior este aun en funcionamiento. Para impedir el movimiento del personaje mientras estamos dando un gancho o una patada, utilizo los booleanos del script que he mencionado antes para poner horInput, variable que venía dada por Input.GetAxis(“Horizonal”) a 0, para así aplicarle un movimiento nulo al personaje.

Para el caso de saltar y utilizar alguna de las teclas de golpes, si saltase y pulsase una tecla como “J” o “K”, el personaje se mantendría en la misma posición hasta que las variables del script, volviesen a ser False. Para ello, utilizando de nuevo otra variable booleana llamada jump, que se pone a False, cuando se detecta que el personaje está tocando el suelo, y se pone True, cuando se pulsa una de las teclas de salto. De esta forma, si jump es True, no se debe accionar ninguno de los golpes. De la misma forma, y utilizando los booleanos kick y hoop creado en este script, los golpes no se deben solapar, así que mientras uno este activo, el otro no debe accionarse.

Para volver a colocar los parámetros del animator a False, cada vez que realizo una acción, lanzo una corrutina que espera un segundo para que termine la animación y cambiar el parámetro correspondiente (Kick, en caso de la patada y Hoop, en caso del puñetazo) a False. Tras esto, espero de nuevo medio segundo, y pongo las variables que impedían el movimiento a False también, para qué de nuevo, el jugador se pueda mover.

(22-03-2019)**[Gancho y patada baja]** Tras incluir los primero golpes del personaje, considere aplicar otro golpes al personaje para agrandar el abanico de movimientos del personaje. En este caso, he considerado introducirle dos movimientos más, el gancho y una patada baja. Ambas animaciones las he sacada del mismo package que utilice para los dos primero golpes, y he seguido el mismo procedimiento para implementarlos.

Algo que destacar aquí, puesto que el procedimiento es el mismo que anteriormente ya he explicado, es que he cambiado el juego de teclas que inicialmente tenían los dos primeros golpes, así, los dos puñetazos, el puñetazo simple y el gancho, quedan en las teclas “J” y “K” respectivamente, y las dos patadas, la baja y la alta, quedan en la “U” y la “I”.

Por último, he corregido algunos problemas que me surgían en las animaciones, reduciendo tanto los tiempos de transición, como los de duración de las animaciones a 1e-5, haciéndolas mas fluidas. Gracias a esto, también he podido reducir los tiempos de las corutinas que devuelven el control tras la animación de un golpe a .5f en puñetazos y .65f en patadas (pues la animación de estas es mas larga) para volver a la animación idle y a .05f luego para devolver el control de movimiento.

También, una de las cosas se ha aplicado al código para optimizarlo, ha sido eliminar todos los bools que se habían colocado en el script, para implementar un solo bool a modo de semáforo llamado “golpe”, para controlar tanto las acciones como el movimiento del personaje.

(23-03-2019)**[Atenea]** Se comienza ahora la inclusión de un nuevo personaje al juego, esta vez la guerrera Atenea. Este modelo ha sido descargado de la página web Mixamo. El nombre original de es personaje no es Atenea, pero se le ha cambiado el nombre por que me ha dado la gana **(CAMBIAR!!)**.

Lo primero que se ha realizado para incluir este personaje, ha sido importar su modelado. Ajustado su posición por defecto igual que a la del prefab de Ethan, lo siguiente es incluirle el Character Controller, ajustándolo a las dimensiones del modelo, y el script CharMovement, el cual permite controlar el movimiento del personaje (es el mismo script que tiene Ethan).

El Animator que tiene este personaje, es el mismo que el de Ethan, pero como en un futuro se cambiaran las animaciones con respecto a la de Ethan, se ha hecho una copia del Animator para poder modificar sin variar a Ethan. A partir de aquí, se colocan las animaciones correspondientes a Atenea en cada uno de los estados del Animator.

Este personaje, debido al tiempo de duración de sus animaciones en los golpes, ha forzado convertir el script inicial de CharMovement en dos diferentes, uno para Ethan y otro para Atenea, pues se modificado los parámetros de las corutinas que controlan la desactivación de las animaciones.

Luego, para incluirla en el resto del juego, se ha sustituido una de las capsulas del menú de selección, por el modelo de Atenea, modificando los scripts para que aparezca su nombre en el menú de selección y en el juego. Se ha seguido para este paso, el mismo procedimiento que se utilizó con Ethan.

(02-05-2019)**[Modificación del script de los personajes]** Como se ha comentado con anterioridad, se ha utilizado un Script para el control del personaje diferente para cada uno de los personajes que hay ahora mismo en el juego. Para favorecer la reutilización, se va a pasar a modificar los Scripts para que los personajes respondan a un mismo Script. Para esto, se coge uno de los Scripts del movimiento de los personajes y donde esta los parámetros de espera de las corutinas de las acciones (que son lo único que difieren los Scripts), se van a parametrizar como variables públicas, de forma que al asignarse a cada personaje solo se tenga que poner el valor a dichas variables desde la interfaz que ofrece Unity para las variables públicas.

(02-05-2019)**[Camara suplementaria]** Para poder implementar el enemigo, primero se implementará un script para la cámara. En primera instancia, esta cámara simplemente seguirá al jugador con un pequeño smooth que ayudará a suavizar su movimiento. Este Script buscara el GameObject de la escena que contenga la etiqueta (tag) “Player” para utilizarla como objetivo. Estas etiquetas se aplican a los personajes en su invocación con ‘Instantiate(characterPrefab).gameObject.tag = “Player”’.

(02-05-2019)**[Enemigo Tipo Zombie]** El enemigo contra el que vamos a luchar va a ser un zombie, para ello, se seguirán las explicaciones del video 3 que aparece en la bibliografía.

Como se puede observar en el video, el lenguaje que se utiliza para realizar el Script del enemigo es JavaScript, por lo que primero se debe “traducir” el Script a C#.

Tras esto, y ya siguiendo las explicaciones del video, se importa el modelado a la escena, y se le aplican componentes como el RigidBody o el Character Controller, y se ajustan al modelado.

Lo siguiente que se realizará, será el Animator conforme se explica en el video. Consta de 4 animaciones y 3 parámetros que se ajusta por el Script.

Por último, se modificarán determinados parámetros del NavMeshAgent, como la velocidad o la distancia de parado, para ajustarlas al comportamiento que queremos que tenga el zombie.

Para finalizar, tanto en el CharController como en el EnemyController (Script que lleva el zombie), se ha colocado el método HurtLife, que permite restar vida a los dos personajes. La vida es una variable de tipo entero que se puede modificar en la interfaz de Unity (es decir, es una variable pública en los Scripts). El daño que se quita con cada golpe del jugador también son variables que se pueden modificar en la Interfaz.

(04-05-2019)**[Enemigo Tipo Zombie]** Para solucionar una los problemas que habían conque el jugador atravesase al personaje del zombie, se ha tenido que calcular la distancia del jugador al zombie y la dirección que llevaba el jugador respecto a la del zombie para poder evitar que el jugador atravesase el zombie. Este es el código que se utilizado para evitar este problema.

distancia = Vector3.Distance(transform.position,enemy.transform.position);

Vector3 heading = enemy.transform.position - transform.position;

float distance = heading.magnitude;

Vector3 direction = heading / distance;

...

if (horInput != 0 && !(Mathf.Sign(horInput) == Mathf.Sign(direction.x) && distancia <= 1f))//Para que se pueda mover el personaje

Aparte de lo explicado anteriormente, se le ha aplicado al zombi un Quaternion.LookRotation(direction), con el mismo cálculo de “direction” que hemos aplicado al jugador, para que gire de forma similar a la del jugador y gane tiempo respecto al giro natural.

Finalmente, se le aplican a los personajes una animación de muerte para el caso de que el zombie gane, y una animación de victoria tanto a ellos como al zombie.

(04-05-2019)**[UI Vida]** Tras introducir el primero de los personajes enemigos, queda introducir un sistema que haga visible la vida tanto del jugador como la del enemigo. Para ello se utilizará dos ScrollBar y mediante el Script que controla la interfaz del usuario, configuraremos las barras de vida obteniendo los parámetros de vida de los Scripts CharController y EnemyController de los GameObjects que tenga los tag “Player” y “Enemy” en la escena.

La configuración de las barras de vida simplemente son dos fracciones de la forma: Vida Restante / Vida total.

(05-05-2019)**[Tercer personaje: Eve]** El siguiente personaje que se va a introducir es Eve. Como en el resto de personajes, primero importamos su modelado desde Mixamo, y ajustamos este al escenario. A continuación, se le colocan los dos componentes esenciales, el Character Controller, el cual ajustamos al personaje, y el Script CharController. Tras esto, se ha de ajustar el Animator partiendo de alguno de los anteriores personajes con las nuevas animaciones. Una vez se ha realizado esto, en el personaje de Eve se ajustan los parámetros de tiempo de las animaciones. Estos parámetros los modificamos desde el editor de Unity, anteriormente se pusieron como públicos y nos permite modificarlos desde ahí.

Para finalizar, se debe enlazar el nuevo personaje con la escena del menú de selección para que se pueda instanciar por medio de su prefab. Para ello, se coloca el prefab en la escena SelectMenu, y se desactiva tanto el CharController.cs como el componente CharacterController, y por medio del Script de UISeleccion, se coloca el nombre “Eve” para que en la escena GameScene se instancie Eve cuando sea seleccionada.

(06-05-2019)**[Implementación de rondas]**  Este juego de lucha está ambientado en el clásico Street Figther. En este juego, la victoria se conseguía cuando se gana 2 rondas de un combate similar, por lo que lo siguiente que se va a implementar será este sistema de rondas.

Lo primero que se va a hacer, va a ser colocar 4 imágenes similares en la interfaz gráfica. Estas imágenes, servirán para que en cada ronda, el jugador sepa quién va ganando.Esta será la imagen que indicara cuantas rondas lleva ganadas cada personaje. Se situa debajo de la barra de vida del jugador.



Para implementar el sistema descrito anteriormente, se utilizará el PlayerPrefs, que nos permite guardar información entre escenas, con 4 strings que serán iniciadas a “No” cuando se seleccione un personaje en el menú de selección. Los strings del PlayerPrefs se pondrán a “Yes” por medio de los métodos de victoria de CharController y EnemyController.

Estos strings, serán utilizados por el método Start de UIGameScene de la siguiente forma:

WinPlayer1.gameObject.SetActive(PlayerPrefs.GetString("WinPlayer1")=="Yes");

Esta línea corresponde a la primera de las imágenes para el jugador.

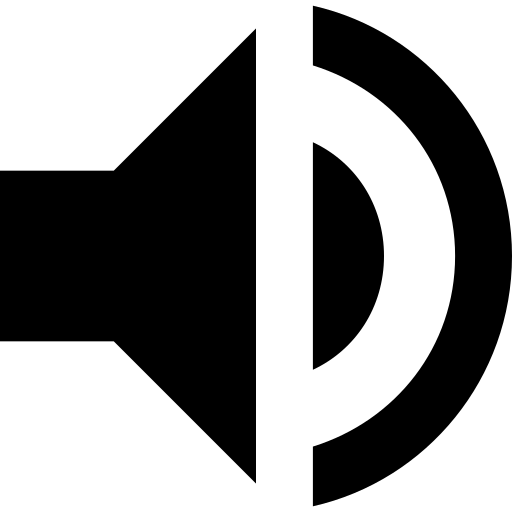
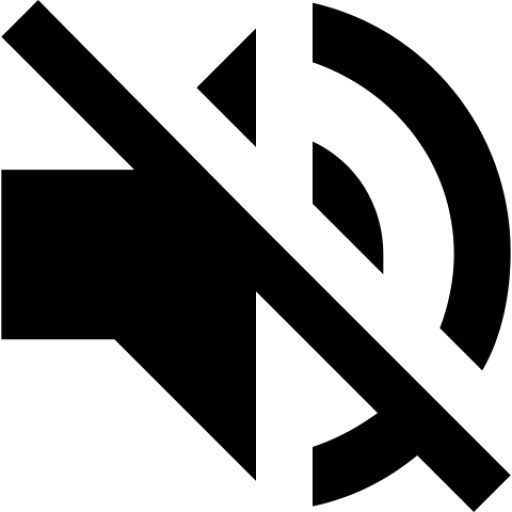
Por último, se crea un panel que se mostrara con un texto y un botón cuando alguien gane la partida. El texto cambia por Script para mostrar el nombre del vencedor y el botón lanza la escena de seleccionar un personaje. El panel se muestra gracia a otros dos Strings de PlayerPrefs, PlayerVictory y EnemyVictory. Estos strings funcionan de la misma forma que los anteriores ya mencionados (“Yes” o “No”) y se modifican en los métodos de victoria (“EnemyDied” o “PlayerDied”) de CharController y EnemyController, cuando se vuelven a llamar después de a ver puesto WinPlayer2 o WinEnemy2 (respectivamente) a “Yes”, es decir, cuando el jugador o el enemigo ganan la tercera ronda.

(07-05-2019)**[Sonido y musica]** A continuación se va a implementar el sonido en el juego. Para ello, se creará un GameObject Empty que se llamará Music al cual le pondremos un tag con el mismo nombre para poder buscarlo. Este GameObject contendrá un componente AudioSource que reproducirá una canción que sonará tanto en el menú de inicio como en los posteriores hasta la GameScene. En la GameScene, habrá un GameObject similar con otra canción para el combate.

También los personajes tendrán un AudioSource para reproducir sonidos para los golpes, y Derrick, el Zombie tendrá un AudioSource también que le permita reproducir efectos de zombie mientras esté vivo.

Una vez configurados los sonidos y los componentes, queda realizar la interfaz gráfica para subir y baja el volumen en partida o silenciarlo.

Para silenciarlo, se pondrá un botón que será visible en el menú principal y alternara entre las siguiente dos imágenes:.



Este botón pondrá AudioListener.pause a true o false según convenga, y junto a un Slider en el menú de pausa de la GameScene, se encargarán de regular el volumen de la partida.

(07-05-2019)**[Selección de mapas]** El selector de mapas que vamos a implementar, será una nueva escena (MapSelector) en la que unos botones permitan cambiar el mapa elegido. Esta nueva escena se encontrará entre la escena SelectCharacter y GameScene.

Para ello, se ha creado 2 nuevos Scripts: MapSelect y MapSelectorController.

La escena cuenta con una serie de 6 botones dispuestos en forma de cuadricula que permiten al usuario pulsarlos y cambiar el fondo de la escena actual. Esto es debido a que los botones contienen el Sript MapSelect, que cuando se pulsa sobre ellos, el método OnClickMap() ejecuta el código que permite cambiar la skybox de la escena actual: RenderSettings.skybox = mat;, siendo “mat”, una referencia al Material que guarda cada botón dentro de su MapSelect.

Otro elemento importante que se ha dispuesto en esta escena, es el GameObject empty MapSelectorController, el cual contiene el Script mencionado anteriormente con su mismo nombre. Este Script, contiene un método SetName(String), que permite modificar una variable privada “name” cuando es llamado. Así pues, la llamada a este método se realiza en el OnClickMap() tras cambiar la skybox de la escena, pasándole al MapSelectorController el nombre del mapa que se ha seleccionado, para que cuando se pulse “Select” en la Interfaz de Usuario, se almacene en PlayerPrefs, una variable “map” de tipo string, que contenga el nombre del último mapa seleccionado.

Con el mapa guardado en las PlayerPrefs, en la escena GameScene, el GameController consulta dicha variable, y conmuta entre los diferentes materiales de los distintos mapas (que contiene en un array de materiales) según el nombre que posea la variable “map”.

El botón de back de la escena MapSelector, permite al usuario volver a la escena SelectCharacter, para volver a escoger el personaje con el que jugar.

\*SOLUCIONADO: El personaje ya no daña al otro cuando esta de espaldas a este (signo(rotación.y) == signo(dirección.x))\*

(07-05-2019)**[Reacción del personaje del jugador a golpes]** Actualmente, si el jugador recibe un golpe no reacciona a él. Con el fin de dotarlo de mayor realismo, se ha modificado el Animator de los personajes jugables para incluir un nuevo estado “Reached”, con una animación que permite aparentar una reacción a un golpe. A este nuevo estado se transitará desde cualquiera de los otros estados (estado Any State), gracias a un nuevo parámetro booleano en los Animator llamado “Reached” también. Este parámetro se pone a true en el método HurtLife(int) del CharController, junto a un semáforo (una variable booleana dentro de CharController), llamado reached, que impide cualquier acción del jugador. A la vez, se inicia una corutina que devuelve el semáforo a false y pone el parámetro “Reached” a false. El semáforo solo se devuelve a false, en caso de que “golpe” sea false, de esta forma, se evita que haya descuadre entre animaciones.

(08-05-2019)**[Panel de Controles]** Tanto en el MainMenu como en la GameScene, se pueden consultar los controles del juego. Para ello, se crea un panel, que se activa y desactiva según la interacción del usuario. Esto se puede realizar gracias al Script ControlsPopUp, cuyos métodos Open() y Close() se han asignado a los botones correspondientes.

(08-05-2019)**[Escena Characters y visualización de parámetros de los personajes]** Para ayudar al usuario a elegir entre los tres personajes jugables, tanto en el menú de selección como en la escena Characters( copia de SelectCharacters con el botón de Select inhabilitado), accesible desde el menú principal, cuando se está visualizando en primer plano un personaje, se verán 3 parámetros del personajes: Vida, Fuerza y Velocidad. La vida se obtendrá directamente de Script CharController del personaje, mientras que la fuerza, será la media de los 4 ataques que posee el personaje, y la velocidad, corresponde a la siguiente formula: **100-(Jtime+Ktime+Utime+Itime)\*10**; es decir, se le resta a 100 (para tener un valor valorado como máximo en 100), la suma de los tiempos de cada ataque, multiplicado por 10.

(08-05-2019)**[Modo multijugador]** A partir de ahora, cuando se le clique sobre Play, desaparecerá el panel actual y aparecerá un panel similar, que contendrá dos botones, Solo y Versus. El modo “Solo”, llevará al flujo que se ha venido mencionando anteriormente. El modo “Versus” será el modo multijugador. Para ello, se reutilizaran las escenas SelectCharacter y la MapSelector, añadiendo además una copia de SelectCharacter, SelectSecondCharacter y otra copia de GameScene, MultiplayerScene. Además, utilizaremos PlayerPrefs para almacenar una variable “Modo”, que permita indicar al SelectCharacter a que escena pasar. Si “Modo” es “Solo”, el flujo que seguirá será el mismo que antes, en cambio, si es “Versus”, tras SelectCharacter se pasara a SelectSecondCharacter, de ahí a MapSelector y finalmente a MultiplayerScene. El parámetro “Modo” se guardará cuando se seleccione uno de los dos botones que hemos mencionado anteriormente, Versus o Solo.

Como hemos mencionado antes, el MapSelector irá tras SelectSeconCharacter en este nuevo flujo, por lo que, para respetarlo, se ha modificado el método que ejecutaba el botón Back de esta escena, para que consulte el parámetro de PlayerPrefs “Modo”. Si modo es “Solo”, cargará la escena SelectCharacter, en cambio, si “Modo” es “Versus”, cargara SelectSecondCharacter. Ademas, en UISeleccion, el Script de la interfaz de usuario de SelectManager y SelectSecondManager, se han creado dos nuevos métodos para que los botones Back y Choose a second de SelectSecondManager, carguen la escena de SelectManager (en el caso de Back) y guarden en PlayerPrefs una variable “secondCharacter” con el nombre del personaje elegido y carguen MapSelector (en el caso de Choose a second).

Por último, queda lo más importante: hacer que MultiplayerScene funcione como es debido. Primero se realizan 2 Scripts duplicados de CharController y se renombran como FirstController y SecondController, además de un duplicado de GameController que se nombrado MultiplayerController.

La función de los duplicados de CharController, es que sean asignados los controles correctos tanto al movimiento como a las acciones de los personajes. Así, FirstCharController será para los personajes escogido en SelectCharacter y SecondCharController será para los personajes escogidos en SelectSecondCharacter. Para que esto funcione como es debido, también hay que crear duplicados de los prefab de los tres personajes (Atenea, Ethan y Eve). Se han creado 2 duplicados de cada prefab, con los mismos nombres, pero con First y Second delante. Los duplicados First, llevaran el FirstCharController y los Second, el SecondCharController, y además se les cambiará la posición inicial para que aparezcan enfrente de los First, como “Derrick”. También se deben eliminar toda referencia al Script EnemigoController de estos Scripts, pues en la escena en la que estos Script existen, el enemigo para FirstCharController será SecondCharController y viciversa.

El MultiplayerController será el Script que llevará el GameObject GameController en esta escena, el cual se ha renombrado para que tenga el mismo nombre que su Script. En este Script, se pasarán como referencia 6 GameObject, que serán los seis prefabs anteriores (3 First y 3 Second). Los First, se instanciarán de igual forma a GameController según el parámetro “carácter” de PlayerPrefs, y de forma similar lo harán los Second con el parámetro “secondCharacter”, eso sí, con el tag “Enemy” en vez de “Player”.

Para finalizar, queda modificar el Script UISelección, el cual se encargaba de regular toda la interfaz del usuario. Anteriormente en el Update, solo se realizaba el cálculo de la barra de vida para un objeto “player” de tipo CharController, y un objeto “enemigo” de tipo EnemigoController. Ahora, esto se hace si el parámetro “Modo” de PlayerPrefs es “Solo”, si no, si “Modo” es “Versus”, el cálculo de la vida o, mejor dicho, el tamaño de ambas Scrollbars que visualizan la vida de los jugadores, se calcula en función de dos objetos: first, de tipo FirstCharController, que hace referencia al GameObject con el tag “Player”, y second, de tipo SecondCharController y que hace referencia al objeto con el tag “Enemy”.

(10-05-2019)**[Cámara Multijugador]** A continuación, se ha realizado un nuevo Script para la cámara del modo multijugador. La cámara que había actualmente en la escena es la misma que había en la GameScene original, y la cual solo seguía los movimientos del jugador con el tag “Player”. Con el fin de tener una cámara más equitativa en el juego, se ha creado es nuevo Script al que se ha nombrado MultiplayerCamara.

En este Script, se busca en primera instancia (en el método Start()), los GameObjects con el tag “Player” y “Enemy”, y se guardan en dos variable privada de tipo Transform (llamada firstPlayer y secondPlayer), su componente transform. Una vez hecho esto, en el método Update() del Script, se calcula la distancia entre los transform.position.x (posición en x), de ambos GameObjects. Esto se realiza mediante el método Distance(float, float), de la clase Vector3. Tras esto, se calcula la distancia media de los dos objetos, simplemente realizando una división entre 2.0f. Esta nueva medida, será la que sirva para cambiar la posición de la cámara tanto en el eje X, como en el Z.

Para cambiar la posición en X de la cámara, se consultan de nuevo ambas componentes transforma, para asegurar cual de ambas es la menor. En caso de que la posición en X de firstPlayer sea menor que la de secondPlayer, la nueva posición en X de la cámara será firstPlayer.position.x + midDistance, siendo midDistance la variable en la que hemos guardado la mitad de la distancia entre ambos objetos. En caso de ser menor la posición en X de secondPLayer, seria secondPlayer.position.x + midDistance.

Una vez calculada la posición en X de la cámara, resta calcular la posición en Z puesto que, si no se modificase la posición en Z de la cámara, los personajes podrían salirse del rango de visión de la cámara. Para hacer el cálculo de la posición en Z, utilizaremos la misma variable midDistance que hemos usado en el cálculo en Z. En este caso, la posición en Z se calculará de la siguiente manera: “posición original de la cámara en Z” - midDistance (pues en este caso la cámara se mueve en -Z) + 2.0f. La posición original de la cámara, se calculara teniendo en cuenta la posición en Z de firstPlayer (o seconPlayer también valdría, es indiferente), y la diferencia entre esta posición y la de la cámara, lo que en el código se llama offsetZ y es una variable pública a la que se le da valor desde la interfaz de Unity. Se calcula de esta forma, porque al estar en el Update(), si cogiésemos una referencia estatica en todo momento, como el transform.position.z de la cámara al principio de la escena, la posición en Z seguiría aumentando en cada paso de frame sin ningún tipo de control. Volviendo a la formula anterior, el + 2.0f del final, indica que inicialmente no se debe cambiar la posición en Z de la cámara, pues se desea que la cámara conserve su posición en Z como mínimo para no realizar un enfoque demasiado cercano sobre ambos personajes. Es + 2.0f porque la distancia inicial entre ambos personajes es de 4. Además de sumarle este valor al calculó de la posición en Z, para que la Z mínima sea la inicial, se debe comprobar en todo momento que la nueva posición en Z de la cámara, no sobrepase (pues recordemos que la cámara se mueve en –Z) la posición inicial, la cual guardamos en el método Start() en una variable privada llamada minZ.

Finalmente, en el método LateUpdate(), se configura la nueva posición de la cámara de forma similar a la cámara anterior, utilizando de nuevo el método SmoothDamp(Vector3, Vector3, Vector3, float) de la clase vector3. La posición en Y no se modifica en ningún momento.

\*Queda poner cosas tanto en Controls como en Credits. Luego ya meter el tema del arte de los huevos hostia puta\*

# Bibliografía borrador

**Cursos**

Introducción al desarrollo de videojuegos con Unity 🡪 curso de la plataforma edX impartido por profesores de la Universidad Politécnica de Valencia.

**Asignaturas**

Introducción a los sistemas gráficos e interactivos (4º A rama computación)

Introducción a la programación de videojuegos (4º A optativa)

Entornos de desarrollo de videojuegos (4º B optativa)

**Libros**

**Videos de referencia**

Video 1: Aprende Unity 3D – Crea tu Main Menu – UI, <https://youtu.be/Ygb9j9b4gQU>

# Video 2: Oye, ¿ Tienes 10 minutos ? - Selector de personajes rotatorio - FÁCIL -, <https://youtu.be/kjRc9bq7_AU>

Video 3: Unity5 - Crear enemigo tipo zombie, <https://youtu.be/MAj1Tp4umdI>