THE QUANTUM CHALLENGE

Trabajo de fin de máster

Jorge Quintana LLitrá

Índice

1. Requisitos

PROYECTO 1: VIDEOJUEGO 3D EN PRIMERA PERSONA

Deberás realizar un videojuego en el que un personaje tenga una vista en primera persona y pueda moverse a lo largo de un escenario, que también crearás tú.

En este entorno debemos tener la posibilidad de encontrarnos con diferentes tipos de enemigos, los cuales deberán tener características diferenciadas entre ellos (por ejemplo, de movimiento, vida, ataque…).

Además del movimiento básico del personaje -andar, correr, saltar y agacharse-, este tendrá un sistema de vida y energía.

Será necesario que todos tengan un determinado valor de energía para poder correr y saltar.

También será imprescindible disponer de un sistema de armas a distancia y cuerpo a cuerpo. Para darle mayor realismo existirá el factor munición, recarga y varias armas disponibles.

El *gameplay* deberá tener una completa interfaz de usuario que en todo momento nos dé información útil del estado del juego.

2. Justificación del proyecto elegido

He elegido este proyecto (Proyecto 1) a desarrollar principalmente porque no posea las cualidades para dibujar, animar o realizar modelos artísticos para un videojuego. Por esa razón he querido platear una idea básica, la cual se pueda suplir con herramientas gratuitas de la tienda de Unity.

En base a la investigación previa, he concluido que hay una mayor cantidad de recursos en 3D que en 2D y hay mayor flexibilidad a la hora de añadir animaciones.

Por otro lado, la gran parte de los proyectos realizados durante el curso, han sido por mi parte en 2D y tenía ganas de enfrentarme a una experiencia nueva en 3D y así aumentar mis conocimientos en Unity.

3. Descripción del juego

La idea principal del juego ha sido crear un juego de acción en primera persona a contrarreloj en el que el jugador debe matar a todos los enemigos de cada nivel lo más rápido posible.

He querido añadir un ranking para añadir una experiencia un poco multijugador en la que los usuarios pueden desafiarse a si mismos para llegar a la posición número 1.

Mecánicas:

El jugador posee diferentes armas que irá desbloqueando con el paso de los niveles:

- Puños: poseen poco daño, pero son menos tiempo de enfriamiento de ataque.
- Hacha: gran capacidad de daño y un ataque lento.
- Arco: tiene un daño decente, pero posee munición finita y se tiene que recargar cuando se acaben las flechas cargadas.
- Bastón: la única arma con daño en área y gran poder ofensivo. También tiene una cantidad de bolas de fuego finitas y el tiempo de recarga es mayor.

Hay tres categorías de enemigos, cada una con una combinación única de salud y escudo. El escudo actúa como una barrera protectora que absorbe el daño antes de que se reduzca la salud del enemigo:

- Minotauro: posee un escudo y es rápido atacando cuerpo a cuerpo.
- Zombi: con poca vida y sin escudo, pero su punto fuerte es que son numerosos.
- Dragón: un formidable enemigo que posee escudo, mucha vida y tiene 3 fases. En la primera, atacara solamente cuerpo a cuerpo. En la siguiente fase, empezará a volar y escupirá fuego atacando desde las alturas. En la última fase, volverá a atacar desde el suelo, pero esta vez con otro ataque en su repertorio y escupirá fuero desde el suelo.

El jugador posee 2 movimientos por armas, acción de castear, apuntar o bloquear y la otra acción de ataque. También tiene una barra de vida y otra de estamina, los movimientos que gastan estamina son los de esprintar y saltar.

Existen 3 niveles en el juego, excluyendo el menú principal:

- Mazmorra: habitan 2 minotauros, realizando una patrulla.
- Cementerio: se encuentran los numerosos zombis moviéndose en un radio pequeño.
- Isla desierta: custodiada por el dragón.

Controles:

Los controles están resumidos en una sección del menú principal. Imagen

4. Detalles técnicos

En este apartado se detallará paso por paso los diferentes aspectos técnicos que son relevantes destacar de este proyecto:

Jugador

El jugador está construido de un script central ***PlayerController*** que se comunica con los diferentes componentes del jugador, dichos componentes son:

1. ***Character Controller***: se encarga de controlar las físicas del jugador, tiene un Collider que se comprobará las colisiones que se produzcan con el propio personaje. En el Animator, se ha construido un Blend Tree para controlar las animaciones de dirección en cualquier dirección, dando los valores en el eje X, Z y el factor de velocidad para manejar si el personaje está corriendo. Imagen

2. ***Input Manager:*** Al estar usando en New Input System de Unity, he querido simplificar el acceso a los controles para saber si están siendo presionados o si han sido pulsados mediante un script que tenga las funciones accesibles.

3. ***Equipment:*** Es el encargado de manejar el equipamiento del personaje, ya sea instanciar las armas o controlar las animaciones de las mismas.

4. ***Animation Controller:*** Ya que he separado la lógica del jugador, el Animator se encuentra en un gameobject hijo del prefab del gameobject del jugador y al querer reproducir los sonidos de las animaciones, le he puesto en el frame correspondiente un evento para que llame a la función determinada. Imagen

5. ***Player Controller:*** Se encarga de controlar las físicas, las interacciones con los enemigos, que pasará cuando se presionan los controles, el movimiento del jugador y así como algunas animaciones o valores de Animator para que las animaciones funcionen correctamente.

Enemigos

Los enemigos se componen en dos tipos, los enemigos estándar que comparten código, pero poseen diferentes atributos y el de tipo jefe.

El script EnemyController, se encarga del manejo de los atributos básicos de los enemigos (vida, escudo y daño), tantos como las acciones que ocurrirán al recibir daño y la actualización de las barras de vida de los enemigos.

He construido un script genérico al que le puedes pasar un enum, que serán los estados que posea esa máquina de estados, así se puede crear diferentes máquinas de estados sin necesidad de tener que crear la lógica de inicial desde 0. Es una clase abstracta que se compone de diferentes métodos que tendrán que ser sobrescritos por las diferentes máquinas de estados.

Todos los enemigos se componen de un Rigidbody, un Animator y un Nav Mesh Agent para controlar el movimiento.

Los enemigos estándar tienen una máquina de estados genérica, EnemyStateMachine que se encarga de controlar los estados que tendrán los enemigos básicos y que al heredar de StateManager, llamará a las funciones al cambiar de estados (EnterState y ExistState), a la comprobación de cuál es el siguiente estado al que ir (GetNextState) y lo que se hará en ese estado (UpdateState).

Por último, comprobará en todo momento la distancia con el jugador para saber si está a rango de perseguir o atacar y que cada estado posea esa información para saber a qué siguiente estado ir.

Los enemigos básicos poseen los siguientes estados:

1. Idle: mantiene la posición en un punto determinado y espera para ir al siguiente estado.
2. Patrol: calcula el siguiente punto de la ruta y se dirige hacia él.
3. Chase: ha detectado que el jugador está dentro del radio de persecución y comienza a perseguirlo
4. Attack: ha determinado que el jugador se encuentra en rango para poder atacar y hace la animación pertinente de ataque y habilita los colliders para hacer daño de sus armas.
5. Hit: se ha detectado que ha sido golpeado y ejecutará la animación pertinente.
6. Dead: la vida del enemigo a llegado a 0 y se ejecuta la animación de muerte y se llama al singleton EnemyLevelController para comprobar las condiciones de si se pasa al siguiente nivel.

El enemigo tipo jefe posee una estructura con menos estados, pero más compleja con diferentes fases y diferentes tipos de ataques dependiendo de la fase en la que se encuentre. Los estados de este tipo de enemigo son:

1. Idle: estado transitorio que mantiene la posición y comprueba si tiene que ir al estado de persecución o de ataque si está a rango.
2. Chase: persigue al jugador hasta que tenga la distancia para poder atacar
3. Attack: se ejecuta el ataque dentro de la fase actual
4. Dead: la vida del jefe llega a 0 y se considera que ha terminado el juego.

Armas

Se ha tenido en cuenta la posibilidad de generar armas de todo tipo y se ha creado un sistema modular para aumentar el repertorio de armas.

Existen dos tipos de entidades utilizando los Scriptables Objects, los objetos de tipo Weapon que contienen como propiedades importantes un identificador del arma autogenerado, los prefabs de cada mano del jugador, una animación de mantener y otra animación de action. La animación de action es la que hace el daño y la de mantener es la que te permite atacar o realizar otro tipo de animación.

Luego tiene el número de proyectiles y la cantidad en recámara y cuanto tarda en recargar.

Por último, tiene el Damageable Object, otro Scriptable Object que contiene la información de lo que va a hacer daño de esa arma.

Estos objetos poseen la cantidad de daño a transmitir del arma y se le puede indicar si es un proyectil, lo que te permite ponerle atributos como la velocidad del proyectil, el prefab del mismo, las particulas al empezar y al ser impactado o si es un ataque de área, con su respectiva fuerza de empuje y su área de impacto.

Interfaz de Usuario

Se compone de 2 scripts básicos para determinar que objetos son paneles y como cambiar de uno a otro fácilmente.

- UIPanel: estas son las entidades básicas de cada ventana de la interfaz, se usa como una clase padre para gestionar como se abren o cierran las ventanas de manera general en todo el juego, también tiene eventos al entrar o cerrar para personalizar cada panel.

- UIController: contiene una lista de tipo Stack y es el script encargado de gestionar cual es el UIPanel actual y de volver al panel anterior.

1. Menu principal

Pantalla inicial de juego donde se encontrarán las diferentes acciones que se podrán hacer antes de iniciar el juego. Imagen

2. Controles

Panel que muestra de manera resumida los controles que se usaran para jugar. Imagen

3. Puntuaciones

Menú que mostrará las puntuaciones de la base de datos, ordenadas de menor a mayor, sabiendo quien es el jugador que ha completado el juego en menos tiempo. Imagen

4. Opciones

Pantalla donde se controlará el nivel de gráficos del juego, el volumen de los sonidos y la resolución. Imagen

5. Créditos

Panel donde se muestra el creador del juego y los assets utilizados para su desarrollo. Imagen

6. Interfaz del jugador (InGame)

En este menú se muestra la información básica del usuario; las barras de vida y estamina, las armas disponibles, la munición en caso de que el arma equipada tenga esa característica y el tiempo que lleva jugando esa partida. Imagen

7. Panel de resultado (InGame)

Pantalla donde se resume los datos de la partida jugada y la posición en la que puedes situarte si guardas tu nombre en la base de datos. Imagen

8. Interfaz Fin del Juego (InGame)

Panel que se muestra cuando el jugador se queda sin vida y te da la opción de ir al menú principal. Imagen

El script PlayerController tiene diferentes UnityAction, los métodos delegados de Unity, a los que el UIPanel encargado de la interfaz del jugador (UIPlayerController) se suscribirá para manejar los diferentes componentes de la interfaz relacionadas con el jugador.

Clases Singleton destacadas

1. Transition Manager

Script de un asset de la tienda de Unity encargado de cargar un nivel, contiene un sistema para realizar un fundido a negro para que no se vea el cambio brusco de la carga de nivel.

2. Level Manager

Es la entidad encargada de hacer uso de Transition Manager para gestionar que nivel ha de ser cargado al acabar el nivel actual, también hace uso de otras entidades para guardar información que se transmite de nivel a nivel y de contabilizar el tiempo de la partida.

3. Player Prefs Manager

Este script está compuesto de diferentes clases que ayudan al manejo de la carga de datos del jugador en la partida. Se ha hecho uso de la clase PlayerPrefs de Unity y se ha introducido como string un objeto de tipo PlayerData, creado en este script, para guardar información serializada a JSON y así acceder a un solo objeto que contiene toda la información necesaria de la partida actual.

4. Singleton

Clase padre que generaliza el patrón de diseño Singleton y crea la instancia de cada componente que desee hacer uso de este patrón de diseño.

5. Tween Manager

Es el encargado de gestionar el uso de los Tweens dentro del proyecto, centralizando y facilitando el manejo de dicho asset.

6. Rest Web Client

Script que gestiona las peticiones que se hacen a la API dentro del juego, hace uso de UnityWebRequest y crea un objeto de tipo Response para gestionar la respuesta de la entidad a la que se llama.

7. Leaderboard Request

Encargado de hacer uso del ScriptableObject que contiene la dirección URL de la API y de gestionar la respuesta mediante el uso del script RestWebClient. Contiene dos métodos delegados a los que se suscribirán las clases que necesiten hacer uso de esas llamadas.

6. Anexo

Imagen de controles

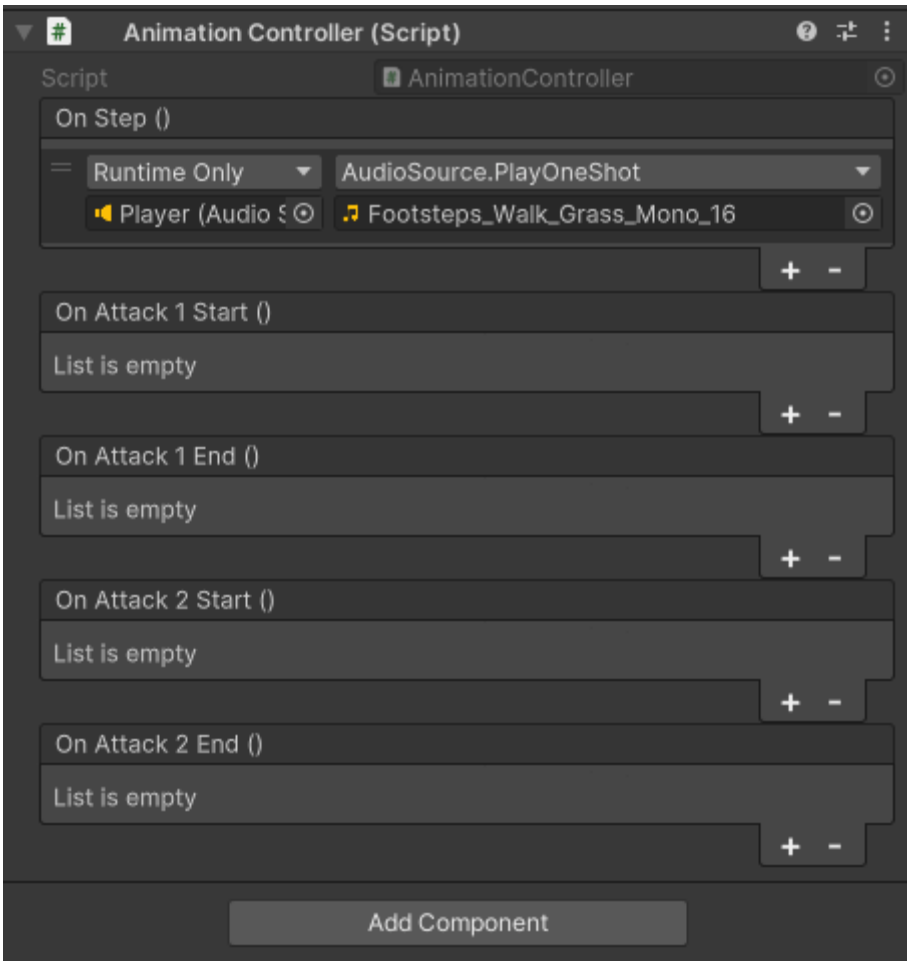

Imagen de ejemplo de Animation Controller

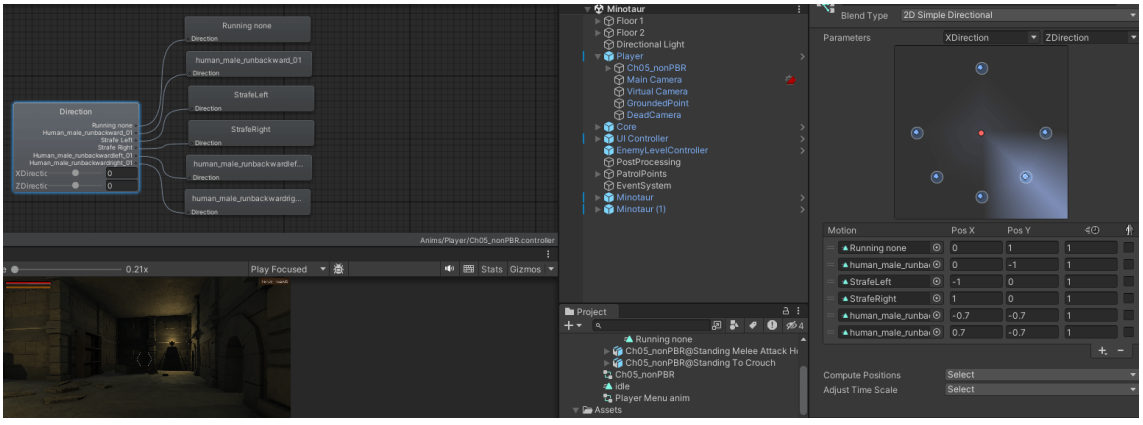

Imagen de Blend Tree Movimiento Jugador

Imagen Interfaz de Usuario – Menú Principal

Imagen Interfaz de Usuario – Controles

Imagen Interfaz de Usuario – Puntuación

Imagen Interfaz de Usuario – Opciones gráficas

Imagen Interfaz de Usuario – Créditos

Imagen Interfaz de Usuario – Interfaz del jugador

Imagen Interfaz de Usuario – Interfaz Final del juego

Imagen Interfaz de Usuario – Panel Resultado