Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

IMPLEMENTACIÓ D'UN JOC ARPG AMB UNITY

José Antonio Cid Montero

Director: Oriol Pujol Vila
Realitzat: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 20 de juny de 2013

Contenido

Tabla de ilustraciones

Abstract

The gaming industry is getting bigger and the size of projects are constantly growing, requiring the work of many people in various fields to offer the player the best gaming experience possible. Among other things, the interest of the project is to understand the development stages of a game and all that this implies for increasing my knowledge on these and its operation, as has always interested me and I would like to work in the future.

As I do not have the resources of time and staff, I will work with an agile development standard known as scrum methodology which is based on sprints with fixed time periods in which they prioritize a number of tasks and you get a beta version stable and terminated. In this way we will not know the level of development reached at the end of work but it will be finished.

The aim of this work is to develop a demo that can be played fluently, where you control a knight and we will beat that is known in the game as an area boss (a dragon) which is stronger than regular enemies. To do this work I used a game engine called unity which provides a number of tools to facilitate development.

Introducción

El objetivo de este proyecto es realizar una pequeña demo funcional de juego del genero ARPG (Action Role Playing Game) que se caracterizan por el combate a tiempo real, por lo que el jugador debe aprender de los movimientos del enemigo, planificar estrategia y reaccionar a estos movimientos a medida que juega. Estos juegos se caracterizan por un gran nivel de fantasía tanto en personajes como en escenarios y por la estrategia ya sea a la hora de subir las estadísticas del personaje, en la historia o en el combate.

La idea es trabajar la lucha contra el que sería un jefe de zona, donde se establece un área de combate más limitada, suele haber solo un enemigo pero con una IA más desarrollada que junto a ataques más fuertes de lo habitual, pongan al límite las capacidades del jugador.

En este caso, tendré todos los roles del proyecto, aunque me centrare más en el diseño del juego y el game play, estás serán mis responsabilidades:

- Diseñador del juego: Definir todo lo que abarca el juego, los personajes (aspecto, motivaciones, movimientos, ataques), quienes serán los enemigos (aspecto, ia, estrategia para vencerlo), el escenario donde ocurrirá todo (texturas, iluminación, distribución), en definitiva todas las decisiones que afecten al juego. Hay que abstraerse para visualizar el producto final y ver como transmitirlo al usuario final.
- Programador: Desarrollar en el entorno Unity la lógica y la IA del juego, esto incluye organizar bien el proyecto y programar los componentes necesarios para el juego (colisiones, físicas, partículas, máquina de estados y otros).
- Artista: Encargarse del apartado visual del juego, se podrán utilizar modelos ya hechos, pero en caso de que sea necesario se crearan modelos y animaciones con 3d studio max u otros programas, para ambientar el juego según se precise.
- Tester: Comprobar que todas las situaciones están controladas, esto incluye que no haya errores, que el juego sea justo, que los modelos y la programación cumplan los requisitos y gestión de las versiones al final del sprint.

Entorno de trabajo

Para la realización del proyecto se va a utilizar Unity que es un motor de juego que nos permitirá organizar, desarrollar y testear nuestro proyecto de forma eficaz y por otro lado utilizare 3ds Studio max como herramienta para editar y animar los modelos.

Unity

Unity es un motor de juego muy potente que se utiliza actualmente en el desarrollo de juegos en casi cualquier plataforma: ordenadores, consolas y móviles, en este enlace podemos ver proyectos que se han realizado mediante este motor:

http://spanish.unity3d.com/gallery/made-with-unity/game-list

Conceptos importantes

A continuación se detallan algunos conceptos previos necesarios para entender el entorno de programación y que serán usados a lo largo del proyecto.

¿Qué es una escena? En la escena (Ilustración 1) se introducen una serie de objetos en posiciones determinadas con unos comportamientos que pueden ser cargados en cualquier momento.

Ilustración 1: Ejemplo de escena des de la vista "Scene" del entorno de desarrollo

Se podría pensar en una escena como una representación de un nivel de nuestro juego, pero también podría ser solo una parte, de esta manera podríamos de forma fácil cargar las partes del nivel a medida que nos hacen falta, si pensamos en proyectos grandes podemos ver apreciar que no se carga toda la información al iniciar el juego, de esta manera no sobrecargar los recursos y mantener una mayor fluidez. Pero no solo eso también sirven para cargar menús, escenas de juego en las que suceden acciones o diálogos.

¿Qué es un gameobject? Es un contenedor (Ilustración 2) que puede agrupar uno o más objetos y se caracteriza por que siempre contiene un componente tranform que nos permite situarlo en una posición determinada.

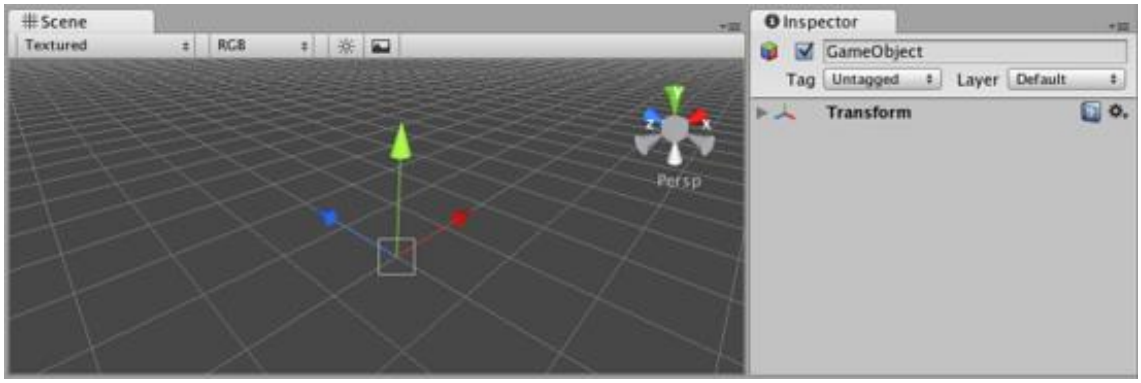

Ilustración 2: GameObject vacío colocado en la escena donde se aprecia el componente transform que determina su posición

Por tanto sirve para situar un componente o para agrupar otros componentes o incluso otros gameobjects, representando así una estructura de árbol que es donde se encuentra realmente su potencial. De esta manera podemos propagar acciones a los nodos hijos o simplemente agrupar componentes. En este ejemplo vemos un gameobject llamado caballero (Ilustración 3) que contiene el modelo del caballero, el esqueleto y un proyector.

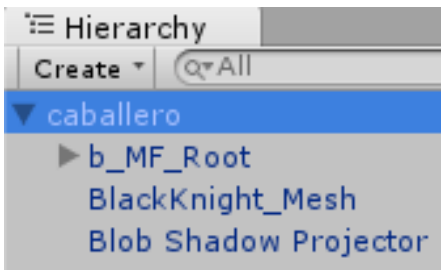

Ilustración 3: Vista del GameObject caballero en el panel de jerarquía del proyecto

¿Qué es un componente? Es la estructura básica, que siempre va asociada a un gameobject y que puede representar entidades visibles, como mallas, materiales, datos sobre el terreno o un sistema de partículas así como otros tipos más abstractos, tales como cámaras y luces.

¿Qué es un Asset? Son todos los archivos importados, pueden ser casi cualquier cosa: un material, textura, archivos de audio, modelos, animaciones o incluso un prefab.

¿Qué es un prefab? Un prefab es un asset que ha sido definido como una plantilla. Al arrastrar un prefab a la escena lo que haces realmente es crear un link a ese prefab o hacer una instancia, de manera que si haces un cambio en el prefab original automáticamente cambiara el estado de los demás. Por tanto es ideal para generar muchos elementos reutilizables, como balas, enemigos, etc, porque si falla algo no tienes que ir uno por uno rectificando.

UI

El entono de desarrollo (ilustración 4) tiene esto componentes aunque pueden estar en una distribución distinta:

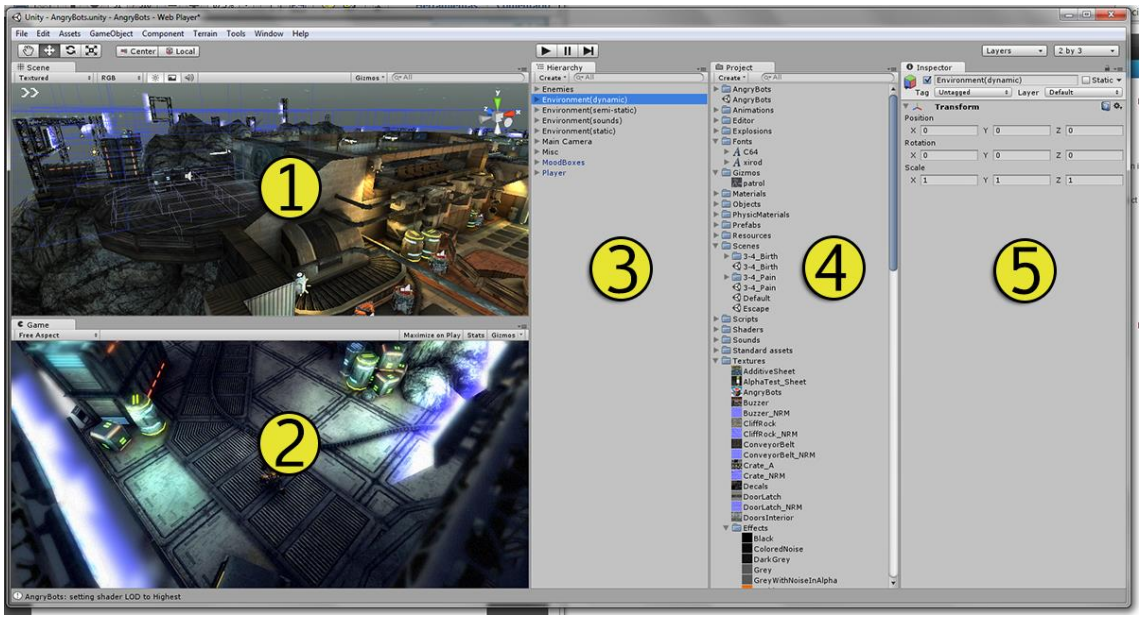

Ilustración 4: Vista de las diferentes partes que componen el entorno de desarrollo

1. Este punto hace referencia a la vista de escena, en la cual nos podemos desplazar y visualizar desde diferentes puntos de vista, pulsando sobre los diferentes ejes del cubo para cambiar de vista o pulsando clic derecho para rotar y la ruedecilla del raton para hacer zoom. Con el clic izquierdo podemos seleccionar, mover, rotar y escalar cualquier objeto de la escena.
   Además esta vista nos muestra información adicional como el tamaño de mallas y colliders, posición y características de la cámara, etc.

2. En la vista de juego, es lo que se vería el juego si estuviésemos jugando realmente, a través de las cámaras que hayamos colocado.

3. Este es el panel de jerarquía, donde se muestran todos los Gameobject, componentes y prefabs de nuestra escena. Los que tienen una flechita quiere decir que se pueden desplegar para ver el siguiente nivel de jerarquía de ese gameobject como habíamos comentado anteriormente(los objetos azules además indican que son también prefabs).

4. El panel de proyecto muestra todos los scripts y assets que tenemos actualmente en el proyecto. Para añadir más elementos podemos hacerlo a través de los submenús superiores o simplemente arrastrando los objetos al panel del proyecto.

5. Por último el panel inspector nos muestra los detalles de los componentes asociados al objeto que tengamos seleccionado. La gracia de este engine es la capacidad de añadir comportamiento a nuestro objeto simplemente añadiendo script y otros componentes para que se comporte como nosotros deseemos.

Funciones

Aquí comento algunas de las funciones de unity más destacadas, que podemos encontrar en la documentación de unity u otras páginas incluidas en la bibliografía.

Update: *function Update () : void*

Esta función es llamada cada frame, si el Monobehaviour (script que la invoca) está activo.

LateUpdate: *function LateUpdate () : void*

LateUpdate es llamado una vez todas las funciones Update han sido llamadas.

FixedUpdate: *function FixedUpdate () : void*

Esta función se ejecuta cada cierto número preestablecido y fijo de frames.

Awake: *function Awake () : void*

Awake es llamada cuando se inicia el script, osea, cuando se carga la escena, y justo después de que se carguen los objects (gameobjects y components) a que el script hace referencia.

Start: *function Start () : void*

Es llamada después de Awake y antes de Update. Al igual que awake, sólo es llamada una vez a lo largo de toda la vida del script.

Reset: *function Reset () : void*

Resetea a los valores por defecto, restaurando los valores originales. Esta función es llamada cuando se pulsa reset en el inspector o cuando se añade un componente por primera vez.

Montar ejecutable

Crear un ejecutable no es nada difícil con unity, lo que necesitamos es que no tengamos errores de compilación, vamos a ir a “File” -> “Build settings…” y se nos abrira la pantalla de compilación (Ilustración 5).

Editamos los ajustes de usuario, la plataforma y la arquitetura, elegimos las escenas que queremos incluir en el ejecutable mediante “add Current” o arrastrándolas y le damos a build, se nos generara una carpeta con los datos necesarios para el juego y el ejecutable.

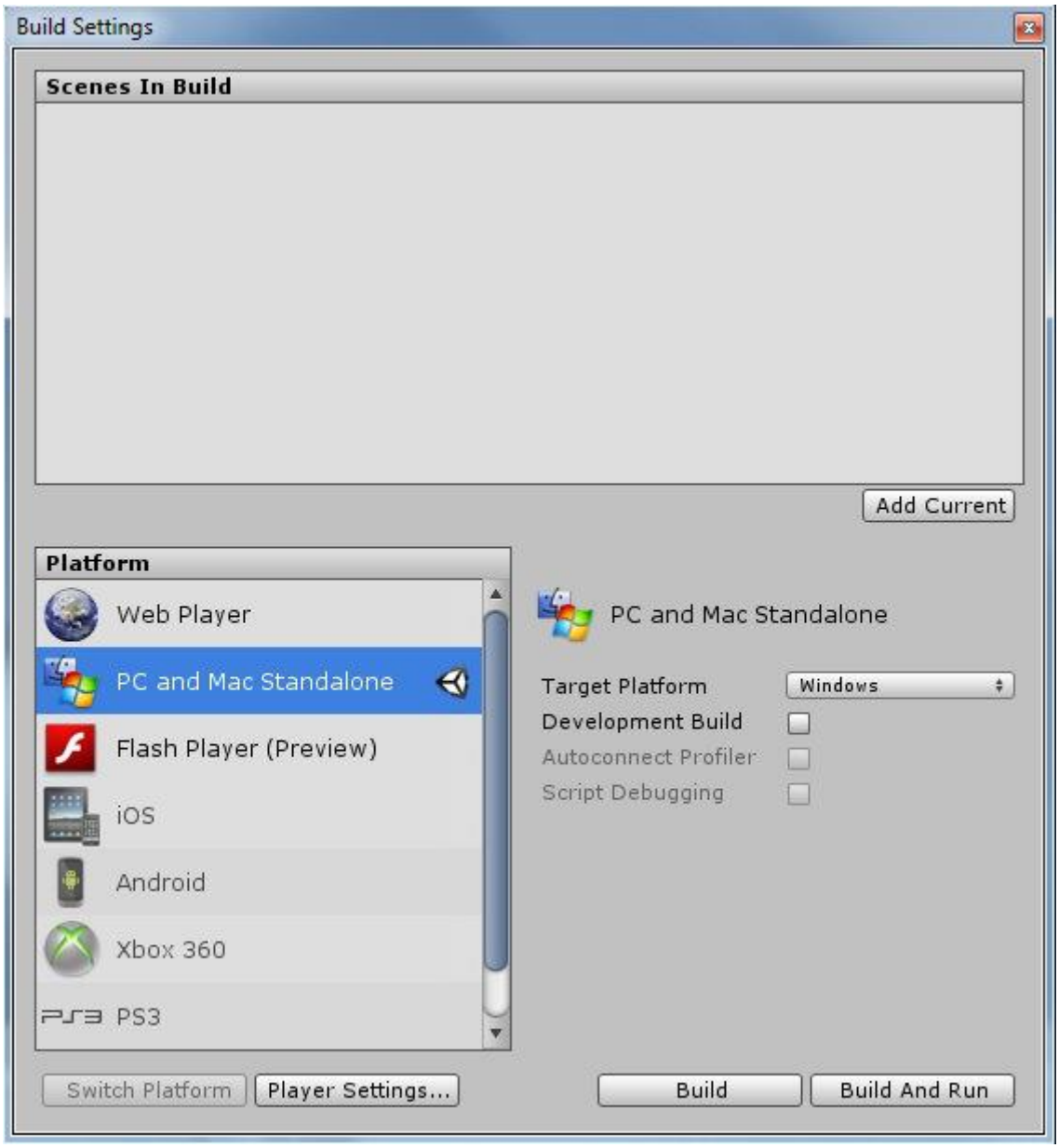

Ilustración 5: Vista de la pantalla que sirve para generar el ejecutable

3DS Studio Max

Este programa servirá para las labores de modelado y animación que se precisen para el desarrollo de nuestro videojuego.

En este proyecto se ha utilizado para un par de animaciones del caballero y en el caso del dragón, se han juntado 2 mallas estaba generado a partir de espejo, añadido los “bones” o huesos que permiten el movimiento de todo su cuerpo y todas las animaciones de este.

UI

Este sería el aspecto de la UI (ilustración 6) donde podemos tener 1 o más vistas desde los diferentes ángulos con diferentes tipos de renderizado. Es importante tener más de una vista para garantizar que los cambios que estamos realizando son como esperamos y en un modelo 3d es complicado de mantener.

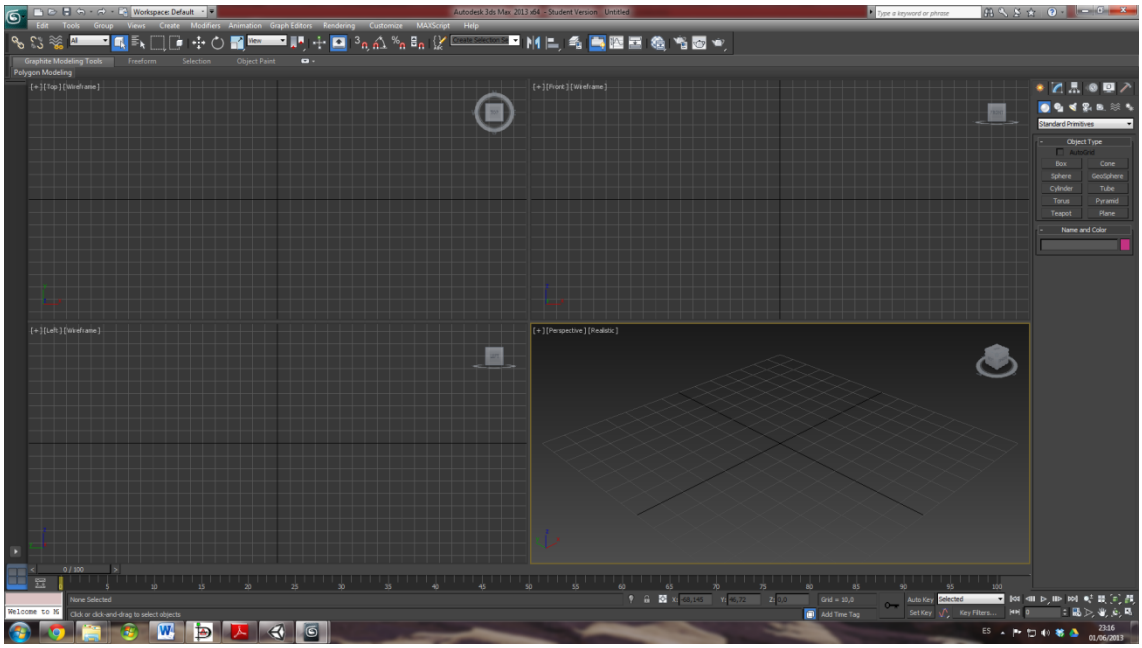

Ilustración 6: Vista del entorno de animación y modelado 3ds max

Exportación

Para el correcto funcionamiento entre ambos programas se debe seguir una serie de requisitos para una buena conexión entre ambos.

Es decir para que todos los materiales, texturas, modelo y animación funcionen en unity deberemos descargarnos el plugin de fbx para exportar los modelos en dicho formato con las siguientes opciones(ilustración 7).

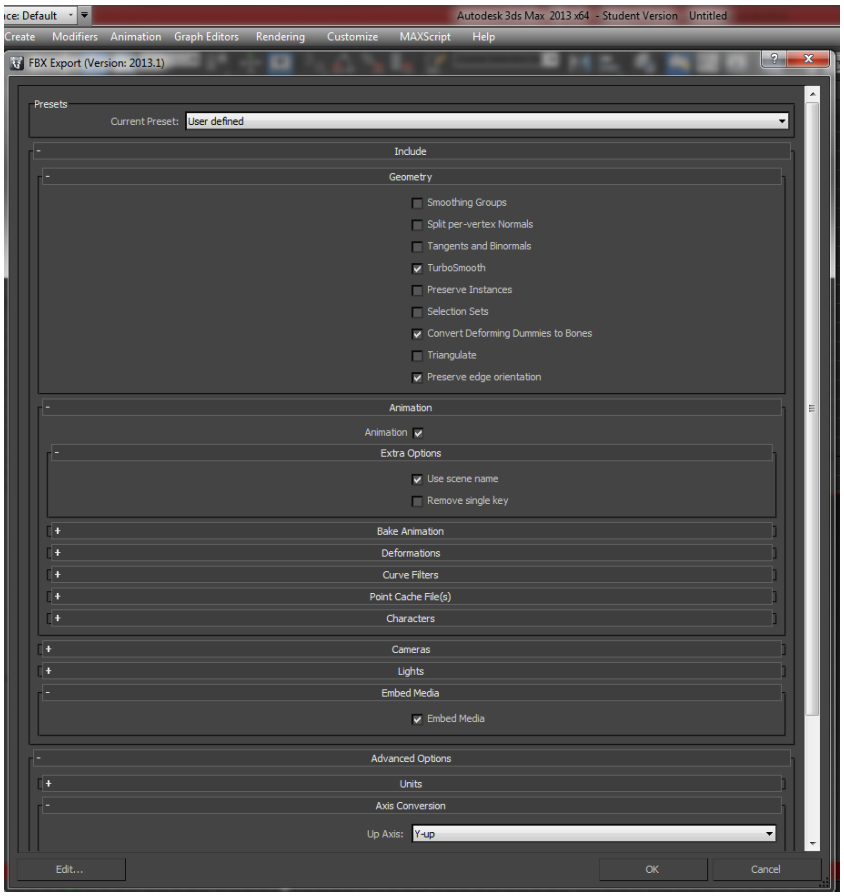

Ilustración 7: Opciones para la correcta exportacion

Una vez lo configuremos la primera vez ya no nos tendremos que preocupar y todo funcionara a la perfección.

Por parte de unity, en cuanto a las animaciones es importante (ilustración 8), que seleccionemos la animacion, y en la pestaña rig marquemos el tipo de animación como “Legacy”.

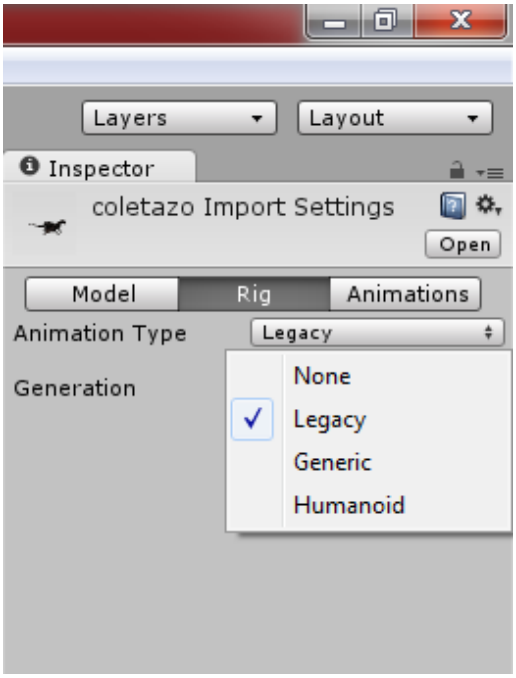

Ilustración 8: Configuración para que funcionen las aniamciones

Análisis

Personaje principal

Descripción

Nuestro personaje (ilustración 9), es un caballero oscuro, que no teme a nada y busca hacerse con las almas de las criaturas más feroces por tal de conseguir su poder, posee una armadura de peso medio, un espadón grande con el que hacer añicos a sus enemigos y un escudo con el que protegerse de los ataques. Por el peso y el tipo de equipo los movimientos y la reacción se realizaran a una velocidad. Dispone hacer ataques ligeros y pesados que podrá combinar para combatir a su rival.

Ilustración 9: Aspecto del personaje que controla el jugador

Movimiento

El movimiento del jugador, es el que nos permite desplazar al personaje por todo el escenario, es el que se encarga también de gestionar las animaciones que pertenecen al movimiento (correr, caminar, etc).

- Caminar.
- Correr.

Es la parte que se encarga de hacer de intermediaria entre el input de teclado del usuario y el juego, se suelen usar las teclas “W”, “S”, “A” y “D” que equivalen a adelante, atrás, izquierda y derecha, aunque también se producir desplazamientos diagonales con la combinación de dos de las teclas.

Además se ha de encargar de gestionar las animaciones pertinentes para aportar realismo al juego, estas animaciones suelen ser cíclicas para poder repetirlas un número indeterminado de veces.

Cámara

El caballero es el personaje principal, es el que va a ser controlado por el usuario y por ello requiere una atención especial. Para eso va a necesitar 3 scripts, uno que controla la cámara, otro el movimiento y el otro el ataque. Separamos el código por funcionalidad y modularidad a la hora de programar, de esta manera no se acumula demasiado código y es más fácil detectar un problema, además podemos llamar a funciones de otros scripts con lo cual no perdemos comunicación entre funcionalidades.

La cámara es un aspecto fundamental, para la experiencia de juego ha de seguir siempre a nuestro jugador para que podamos obtener información suficiente para tomar las decisiones que nos pueden llevar a la victoria o a la derrota. Ha de ser por tanto cómoda y eficiente si no el jugador enseguida se cansara, por ello no tiene que hacer movimientos bruscos porque podrían producir mareo o desorientación en el jugador. Además ha de estar bien posicionada para que no suponga para el usuario un esfuerzo observar la situación actual.

En la escena aparecerá como un pequeño icono blanco en forma de cámara o proyector (ilustración 10), donde se mostraran también unas líneas que representan la zona de alcance y además disponemos de una pre visualización en pequeñito de lo que capta.

Ilustración 10: Aspecto de la cámara al colocarla en la escena

Movimientos en combate

- Ataque con la espada vertical de arriba abajo
- Ataque con la espada vertical de abajo arriba
- Ataque con la espada horizontal de izquierda a derecha
- Ataque con la espada horizontal de derecha a izquierda
- Ataque con el escudo vertical de arriba abajo
- Patada

El sistema de combate funcionara mediante un sistema de combos, en los cuales se podrán encadenar diversos goles rápidos y pueden finalizar con un golpe fuerte. Mediante el clic izquierdo del ratón nuestro personaje realizara un ataque rápido en función del número de ataques realizados anteriormente en la secuencia del combo hasta un máximo de 4, y podremos volver a empezar la secuencia desde el principio. Por otro lado tenemos los golpes fuertes que son algo más lentos pero más poderosos y que se realizan con el clic derecho del ratón, al realizar uno de estos golpes el combo se dará por finalizado y tendremos que esperar un par de segundos para volver a atacar.

Este es el diagrama (ilustración 11), el color verde representa el estado idle, el azul los ataques rápidos y el rojo los fuertes:

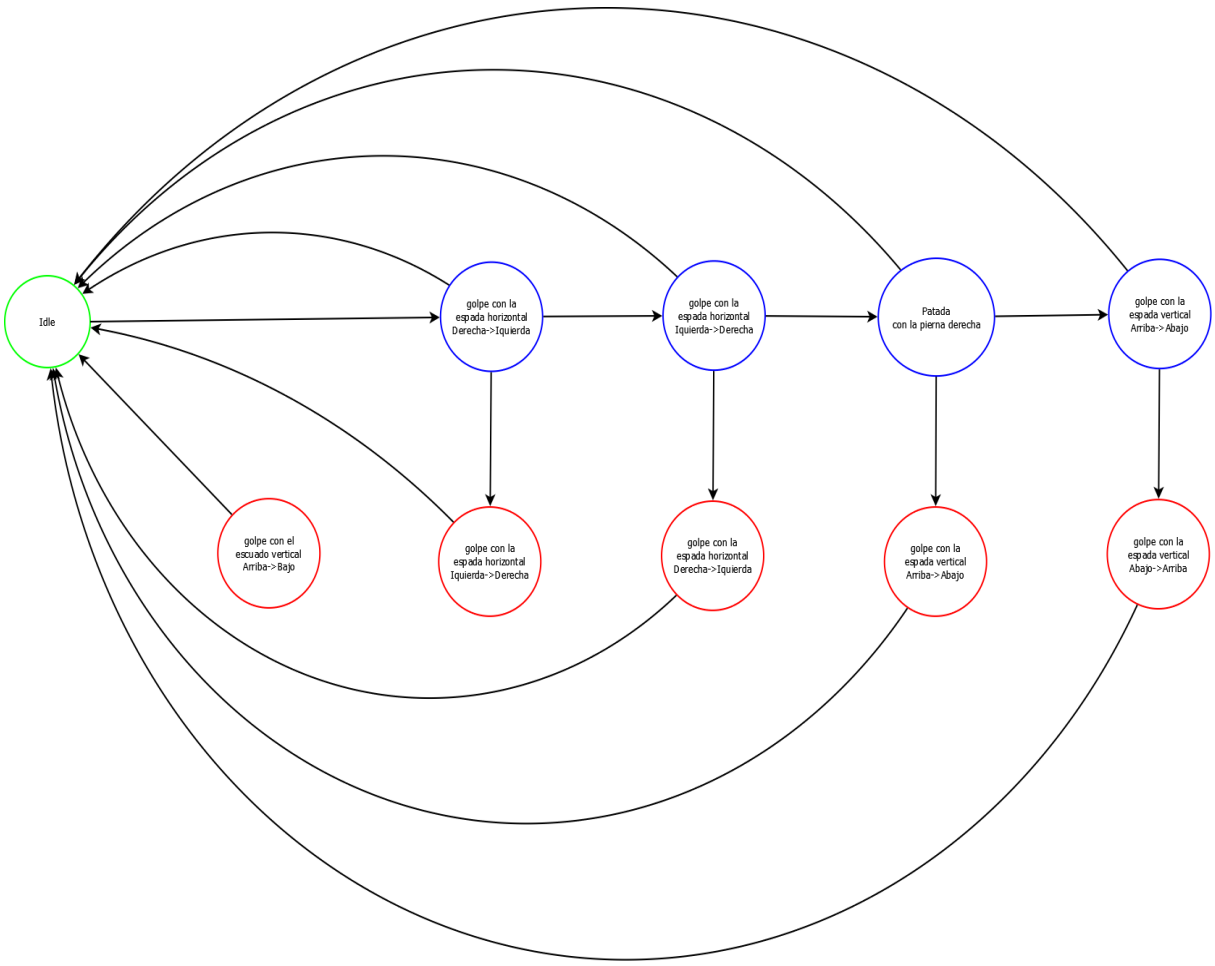

Ilustración 11: Diagrama de flujo de los combos de ataque del caballero

Para la reproducción de las animaciones en función del input del usuario seguiremos este diagrama de flujo (ilustración 12).

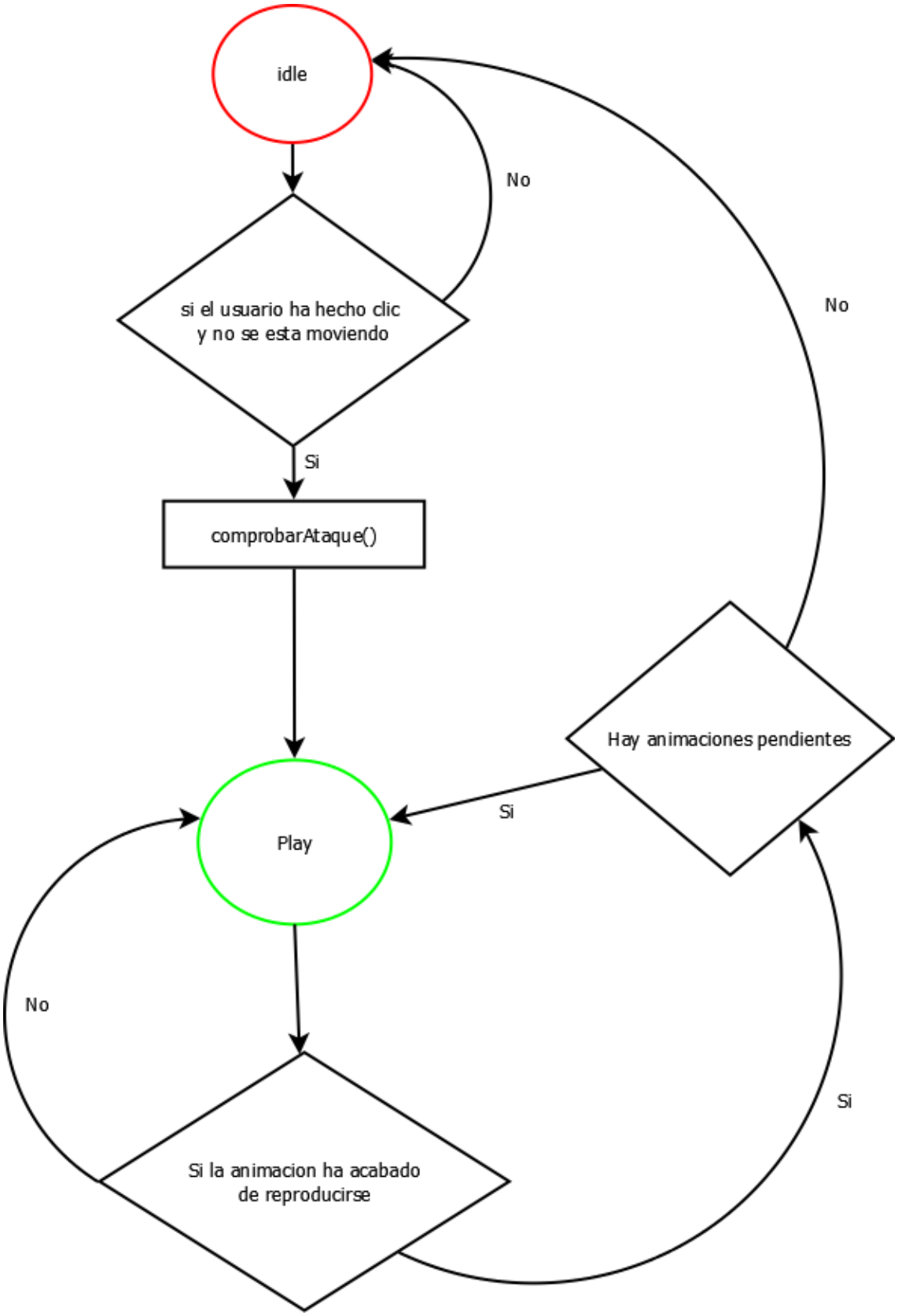

Ilustración 12: Diagrama de flujo para controlar las animaciones de ataque

NPC

Descripción

El enemigo al que tendremos que enfrentarnos no es ni más ni menos que el último dragón (ilustración 13) existente en esta era. Bestia muy territorial, de garras y dientes muy afilados y fuego capaz de fundir la piedra. El caballero entra en dicho territorio cosa que no puede permitir y ha de dejar las cosas claras y solo puede ser por las malas.

Ilustración 13: Ilustración del enemigo

Movimientos

- Caminar
- Volar

Ataques

- Mordisco
- Llamarada
- Golpe con la cola
- Lanzar fuego mientras vuela

Para la inteligencia artificial del enemigo vamos a utilizar un script que vamos a asociar al gameobject que representa nuestro enemigo y que gestionara su máquina de estados.

El primer concepto importante a tratar será la percepción del enemigo, tanto el campo de visión, como el auditivo o de proximidad. El campo visual contiene diversos aspectos particulares que hay que tener en cuenta, por ejemplo ¿El objetivo está cerca? , ¿Puedo verlo siempre o tengo limitaciones como la distancia u obstáculos? , ¿La zona de visión es lineal? , ¿Puedo ver en todas direcciones?

En primer lugar el enemigo solo puede ver cosas que estén delante de él (ilustración 14):

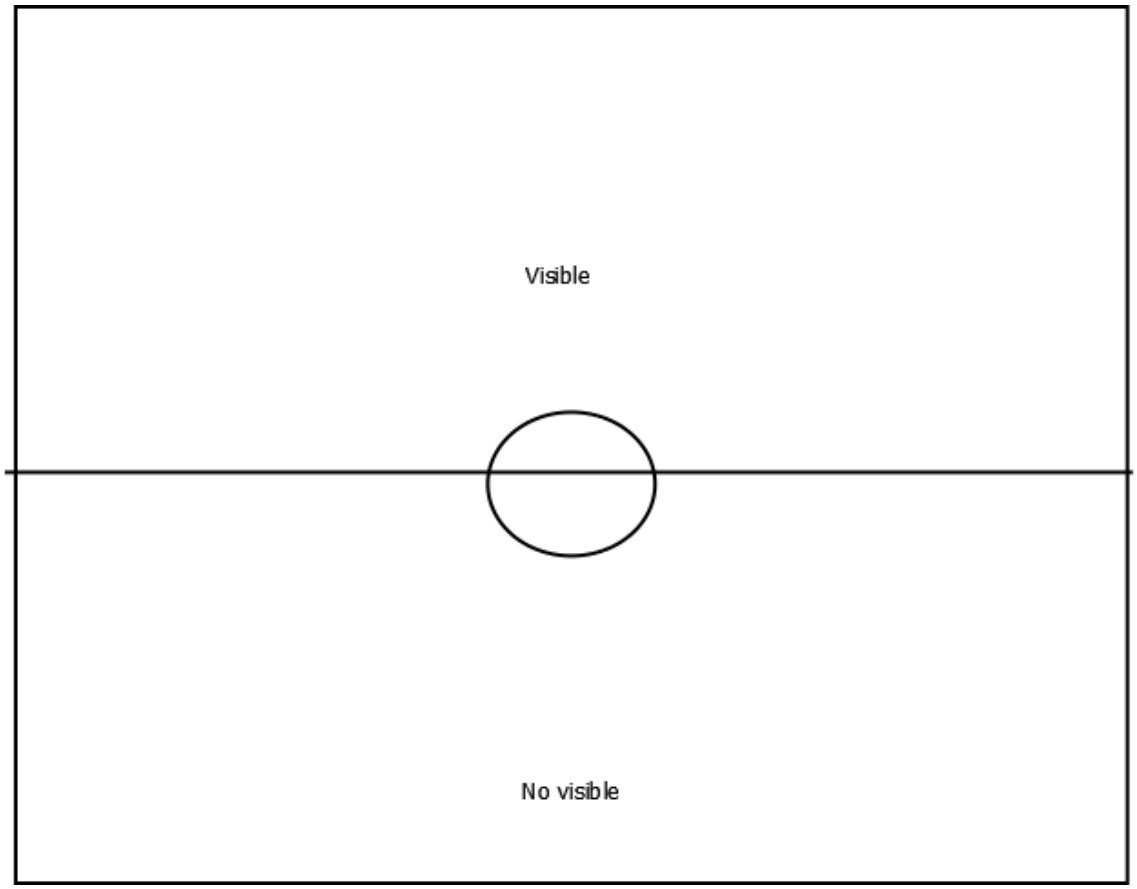

Ilustración 14: Campo de visión simple

Por otro lado el campo de visión no sigue una forma lineal, tendría forma de cono (ilustración 15), aunque podemos simplificarlo como un triángulo, de cara a nuestras necesidades, es por eso que las cosas más cercanas tienen que estar justo delante nuestro para poder ser percibidas, mientras que de las cosas más lejanas podemos tener una perspectiva más amplia de la situación, sin embargo las distancia también nos aporta otra serie de limitaciones.

Aquí podemos ver una pequeña representación visual de lo que intento explicar, sin embargo hay que tener en cuenta que podemos perder de vista al objetivo tanto si se acerca mucho y se esconde de nuestro campo de visión como si se aleja mucho y no podemos percibirlo, por no hablar de los obstáculos.

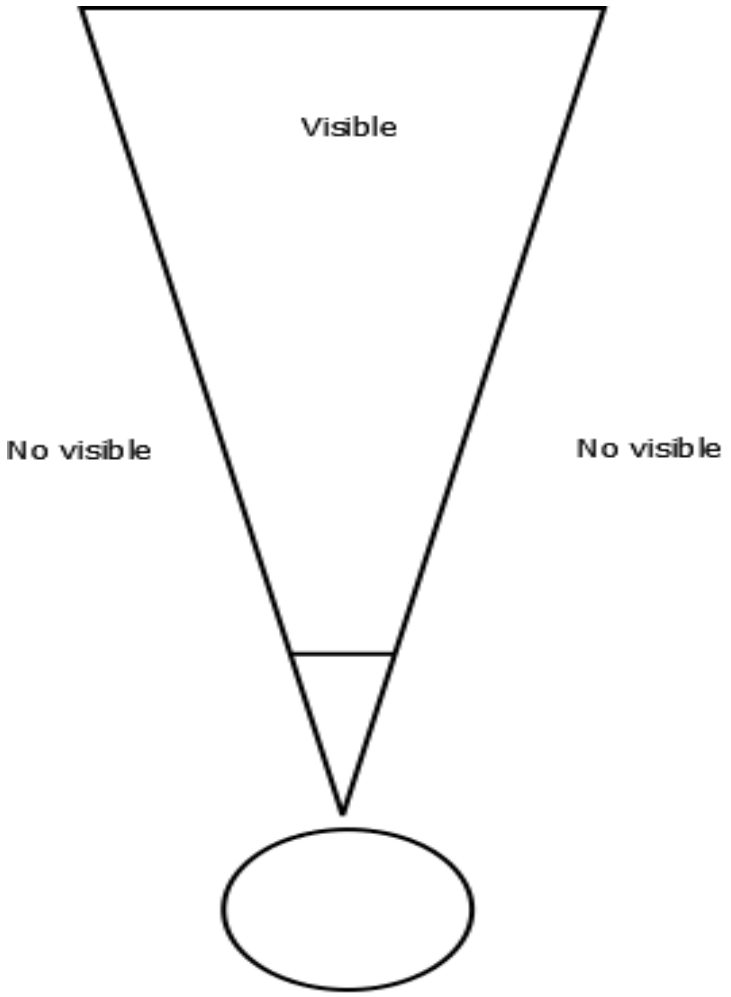

Ilustración 15: Aproximación de campos visual real

Qué pasa si el objetivo se pone detrás de algún obstáculo, o tú mismo te sitúas en una posición desventajosa, eso implica que perdemos de vista al objetivo y debemos tenerlo en cuenta para conservar el realismo.

Pero qué pasa si el enemigo está muy cerca o detrás, ¿eso implica que estamos indefensos a su entera disposición? Pues no para ello simularemos también la audición, tacto u olfato (ilustración 16) con percepción de corta distancia que representaremos de la misma manera mediante un raycasting más pequeño, de 360º.

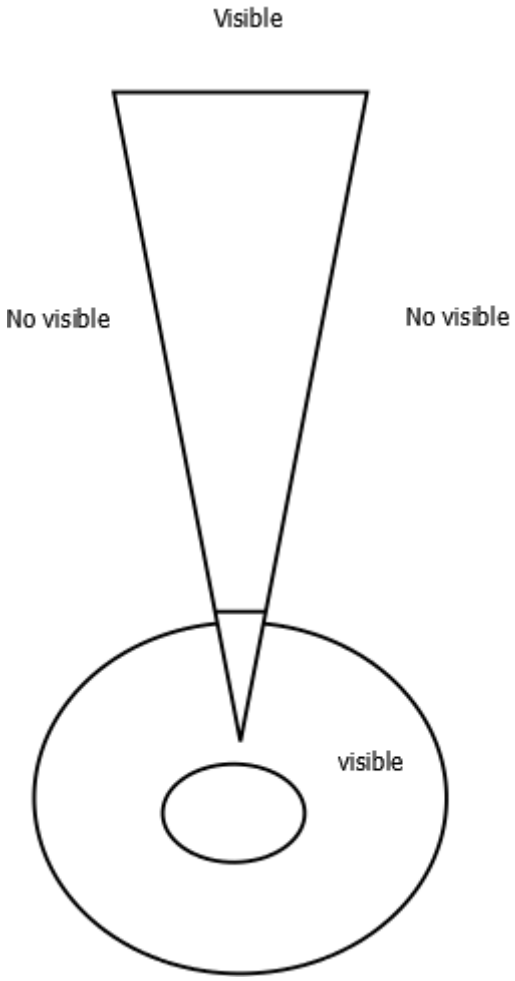

Ilustración 16: Campo de visión mas sentido auditivo

También podemos comprobar el ángulo dentro de la zona por ejemplo para saber si esta justo detrás y actuar de una manera determinada.

Pero no todo es percepción, en caso de no saber dónde está el objetivo podemos hacer algo más que quedarnos quietos esperando a ser atacados por sorpresa y dejar planear al usuario la estrategia con total parsimonia, para ello podemos simular una pequeña búsqueda, de forma aleatoria moveremos el gameobject mediante translaciones o rotaciones.

Por otro lado una vez detectado no basta, atacar hay que buscar la orientación correcta y acercase lo suficiente para intentar dar un ataque certero y realizar ataques de diferente alcance, además de una cierta aleatoriedad para que el usuario no controle la situación tan fácilmente, una IA debe ser equilibrada ni demasiado difícil como para ser invencible, ni demasiado fácil para como para permitir al usuario campar a sus anchas.

Se ha definido pues este diagrama (ilustración 17) que mediante test, garantiza un buen sistema de juego contra el usuario

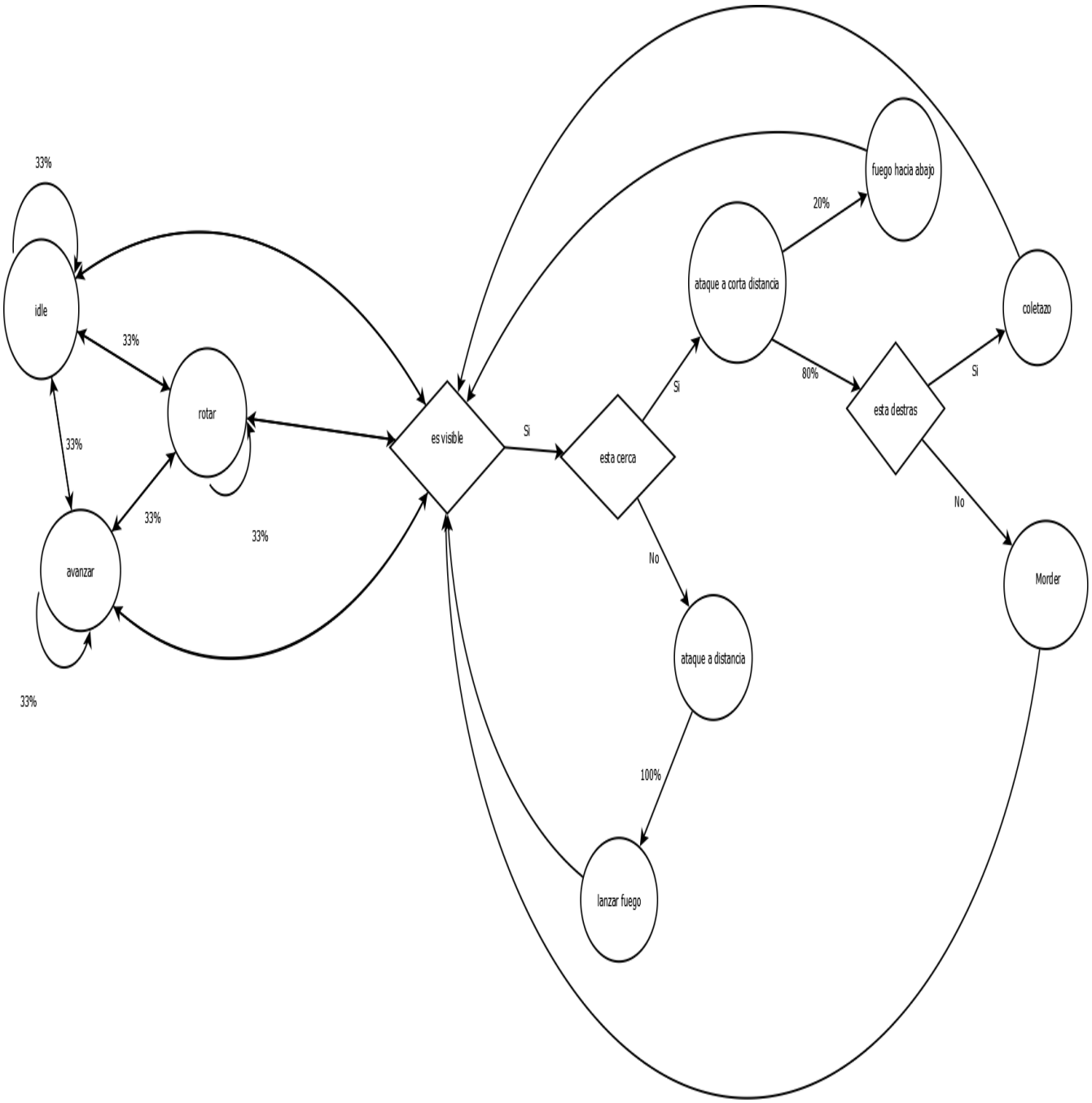

Ilustración 17: Diagrama de la inteligencia artificial del dragón

Es importante que antes de hacer un ataque el enemigo mire hacia el adversario, para que el ataque pueda tener éxito, pero no de forma continua porque si no podría ser esquivado y lo mismo a la hora de colocarse par aun ataque a corta distancia, debemos permitir que el enemigo se acerque pero no que le siga indefinidamente. Igual que con el caballero hay que asegurarse que la animación termina antes de empezar la siguiente y dar un margen entre ataque y ataque para que el jugador tenga tiempo a reaccionar o aprender a anticiparse para poder ganar.

Estrategia

1. Si estamos detrás del enemigo y este levanta la cola quiere decir que se está preparando para darnos un coletazo, debemos actuar lo antes posible para prevenir el ataque y esquivarlo.
2. Si estamos delante de él y levanta la cabeza es que se dispone a mordernos. Debemos salir del rango de alcance.
3. Si estamos cerca del dragón y mueve las alas es que se dispone a volar para lanzar una llamarada de fuego hacia abajo, por lo que deberemos correr para salir del rango de alcance (deberemos salir lo antes posible para minimizar los daños).
4. Si nos encontramos lejos y abre mucho la boca es que se dispone a lanzar una llamarada, igual que antes nuestra única opción será salir del área de ataque.

Entorno

Estamos en lo alto de una montaña, donde se sitúan unas ruinas derruidas, de lo que fue una gran civilización hasta que dicho dragón decidió hacer de este su hogar. En el suelo abundan las cenizas debido a que reclamo su territorio de la mejor forma que saber hacer, quemarlo todo!

La gran cantidad de cenizas que se desprendieron durante años de llamas y más llamas, han sobrecargado el ambiente, haciendo que se acumulasen en el cielo y provocando una luminosidad tenue (ilustración 18).

Ilustración 18: Ilustración para dar la idea de un paisaje similar al que se quiere transmitir

La única manera de llegar a este lugar a este peculiar lugar al que hace siglos que nadie osa pisar desde la catástrofe es a través de un inestable puente de madera (ilustración 19) que conecta con la zona “sin retorno” de la montaña denominada así por los pueblerinos atemorizados y las leyendas que se cuentan sobre los que lo han intentado y nunca han vuelto.

Ilustración 19: Ilustración del modelo de puente

Lo único que queda es algún ruinoso edificio en pie, el dragón y tú.

Esquema del escenario (ilustración 20):

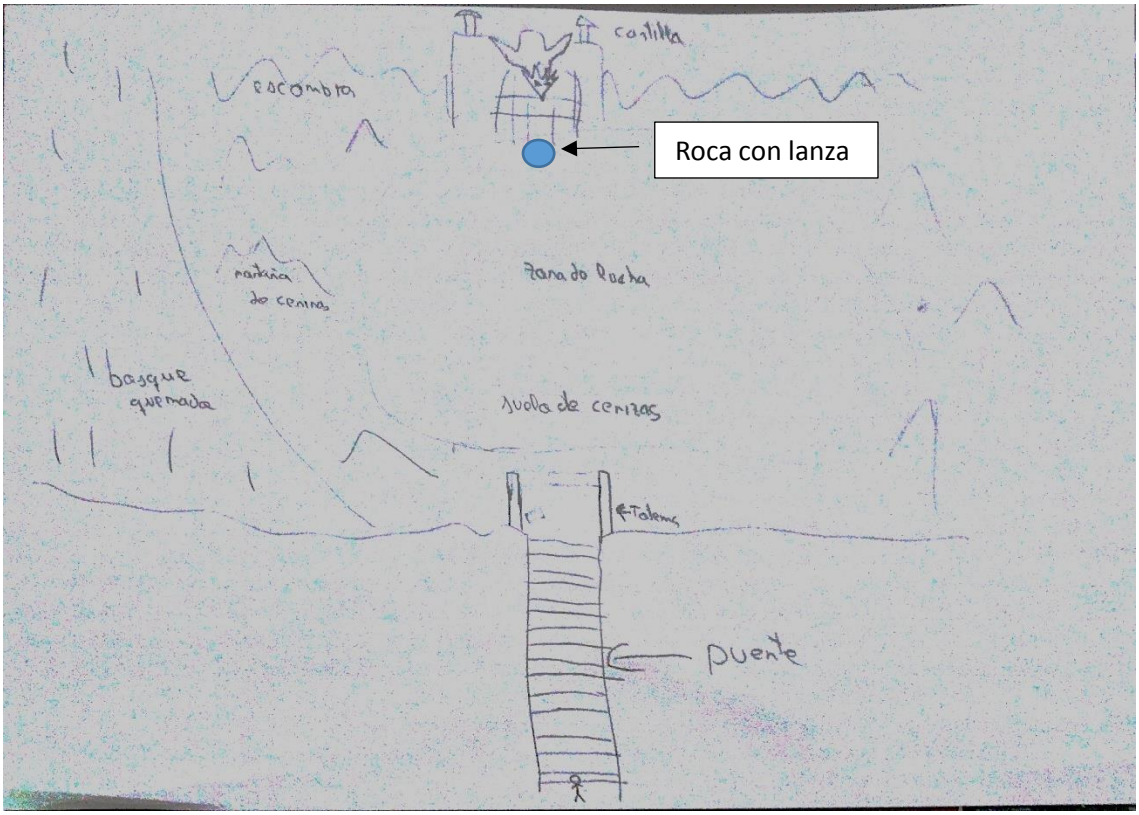

Ilustración 20: Mapa del escenario

HUD

En el hud se va a visualizar una barra de vida de color verde y una imagen del caballero en la parte inferior derecha de la pantalla. Hay varias opciones para mostrar la barra de salud, yo me he decantado por esta ya que permite de forma fácil hacer que la barra de vida adopte cualquier forma.

Las barras de vidas sirven para dar un feedback al jugador sobre el estado del juego, esto ayuda al usuario a aprender la mecánica del juego y saber cuál es su error o que es lo que le hace más vulnerable, así pues a medida que juega puede llegar a conclusiones como que ataques debe evitar a toda costa (los que le causan más daño o el que recibe mayor número de veces), así como ser más cauteloso al ser consciente de que le queda poca vida.

El dragón no contara con barra de vida para provocar en el usuario una sensación de presión constante, al no saber la vida del enemigo suele tener más cuidado de sus acciones, ya que no sabe cuánto tiempo más tiene que aguantar la batalla y solo sabe aproximadamente que daño puede haberle causado. Tiene la parte mala que no aumentara la intensidad del jugador al final de la batalla para terminar con el dragón y pierda una buena oportunidad o se agobie tras varios intentos sin saber qué hacer.

Sonido

El sonido nos ayuda a sumergirnos aún más dentro del juego, ya que cuantos más sentidos afectamos, más profunda es la experiencia.

Hay varios tipos de sonidos:

- Sonido ambiente: Es el sonido de la escena, puede ser producido por factores del entorno como el viento, objetos de alrededor próximos a nosotros o también una música de fondo que acompañe la situación, por ejemplo en una batalla hacerla más frenética.

- Efectos de sonido: producidos por alguna situación concreta y que dan más realismo al juego por ejemplo cuando se produce un golpe.

Los podemos reconocer en la vista de escena por este símbolo que representa un altavoz (ilustración 21) y podemos ajustar varios parámetros:

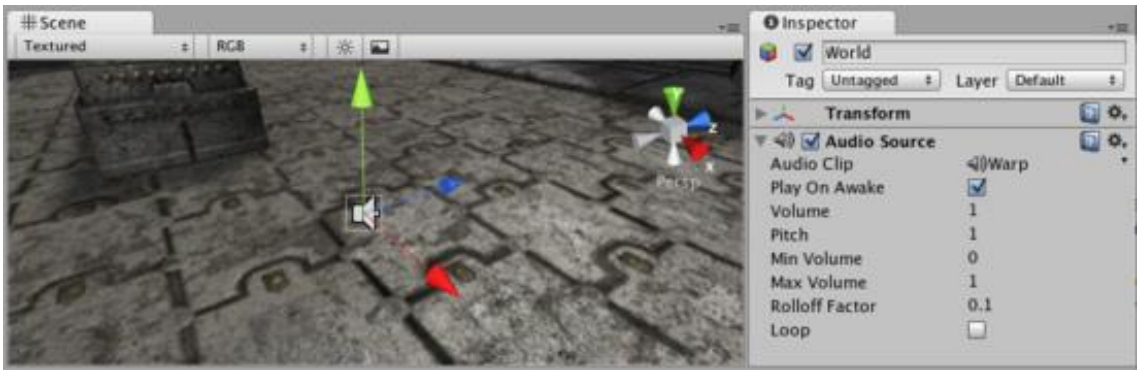

Ilustración 21: Componente AudioSource colocado en la escena

Colisiones

Como en todo juego es una parte fundamental ya que se basan en la interacción, en nuestro caso la más importante causar daños al enemigo y que el nos cause los menos posibles. Por ello debemos ser cuidadosos con este tema ya que el juego gira entorno a esto y es importante que solo se reciba daño en el momento adecuado, para una buena experiencia de juego y para hacerlo lo más realista posible vamos a trabajar con colliders.

Un collider (ilustración 22) es un espacio de la escena donde captamos que algo ha entrado en ella y en consecuencia actuamos de alguna manera, en nuestro caso lo vamos a utilizar para las colisiones colocando estos colliders de forma estratégica, representando el alcance de un ataque de manera que si algo entra en ese collider cuando hacemos el ataque quiere decir que el ataque habrá sido un éxito.

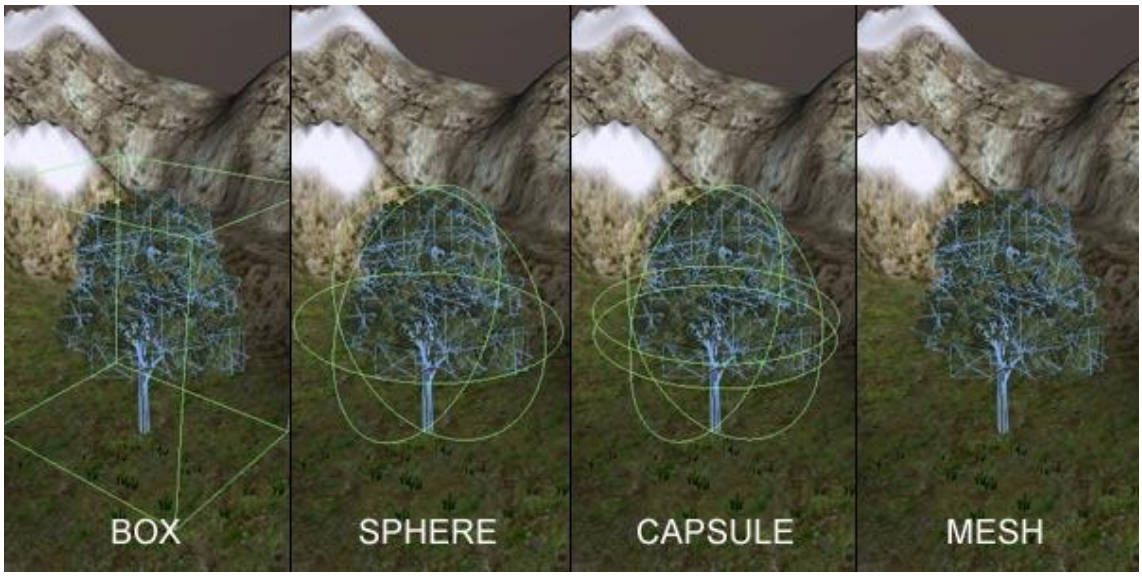

Ilustración 22: Tipos de colliders

Sistemas de partículas

Estos sistemas se utilizan para simular efectos como por ejemplo el fuego (ilustración 23), el humo, etc. Estos sistemas están formados por varios componentes.

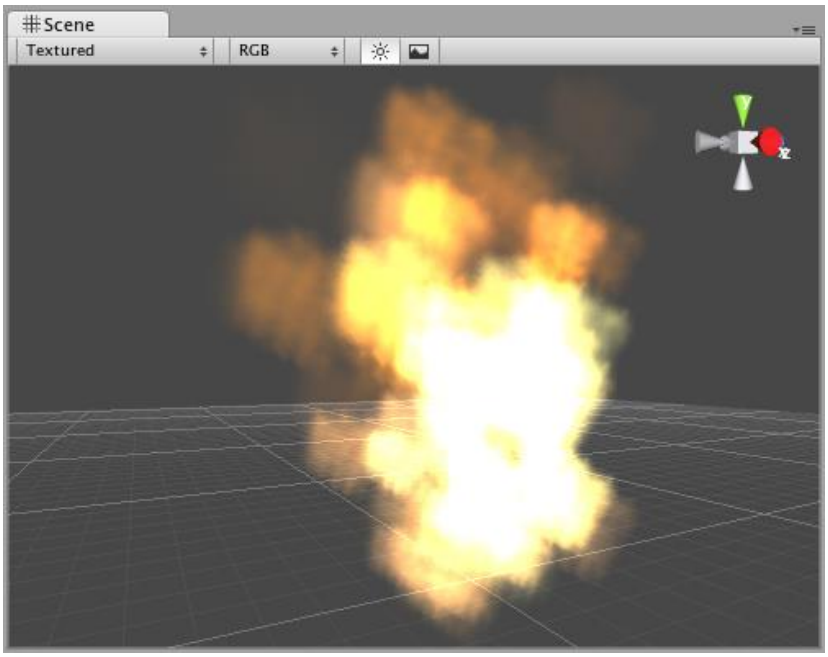

Ilustración 23: Ejemplo de fuego hecho con un sistema de particulas

Diseño e implementación

Personaje principal

Movimiento

Para el movimiento podemos aprovechar un script básico que nos proporciona unity con una adaptación, añadiremos un nuevo método que llamaremos desde otros scripts dando valor a un booleano para indicar cuando el jugador que puede moverse y controlando la condición en la función update, esto se usara para limitar el movimiento cuando el jugador ataca.

```
function isMovable(ocupado : boolean) {
        isControllable = ocupado;
}
```

Además de este script deberemos asociar al gameobject del caballero un componente gameController, después asignamos las animaciones correspondientes y editamos los parámetros a nuestro gusto (ilustración 24), de manera que la velocidad de la animación se ajuste al desplazamiento.

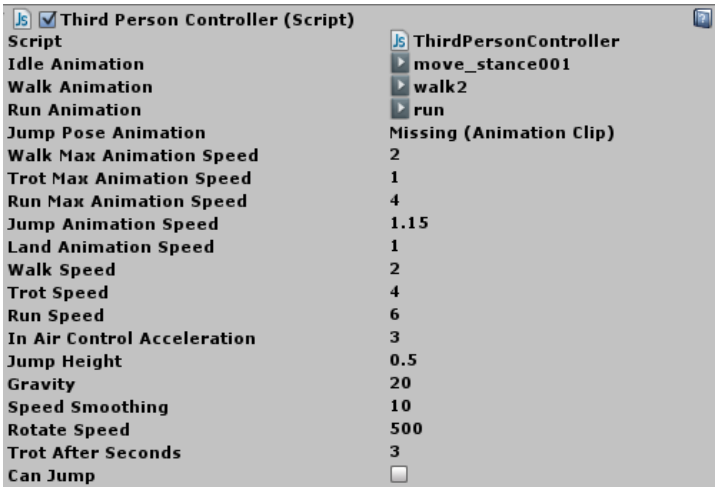

Ilustración 24: Parametrizacion del script que controla el movimiento

Cámara

En este caso vamos a utilizar una cámara en tercera persona que seguirá al usuario y que nos permite tener un rango de visión más amplio de la situación actual del personaje.

Para ello podemos utilizar el script thirdpersonCamera que nos proporciona unity y únicamente haremos una pequeña variación de manera que podamos dar la opción al usuario de elegir entre cámara libre o centrada sobre nuestro objetivo.

El uso es sencillo unimos el script al caballero y asociamos a este la cámara que debe seguirlo que en este caso es la principal, asociamos también por referencia al objetivo que queremos centrar, podríamos obtenerlo por código pero como solo hay un enemigo y siempre va a ser el mismo lo asociamos ya directamente y retocamos los parámetros de la posición de la cámara a nuestro gusto (ilustración 25).

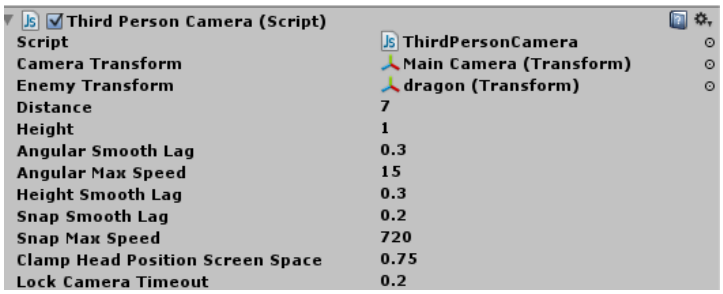

Ilustración 25: Parametrizacion de la camara

Para el ajuste del script cada vez que pulsemos el botón central del ratón se cambiara el modo de la cámara, añadiremos otra variable más debido a que si no generamos un pequeño delay de 1 segundo porque si no se ejecuta demasiadas veces las instrucción (Recuerda que el update se llama a cada frame) y no conseguimos el efecto deseado.

```
if (Input.GetButton("Fire3") && !busy) {
        fijarObjetivo = !fijarObjetivo;
        wait();
}
```

Ataque

Para programar esta utilidad en unity he creado un fichero javascript que se llama "ThirdPersonCharacterAttack.js" que se enlaza con nuestro prefab caballero. Dicho script controla la máquina de estados definida por el diagrama de arriba, consta de la función start que se llama una única vez antes que cualquier función y nos sirve para inicializar variables en este caso las animaciones. Después tenemos la función update que se llama en cada frame, esta función comprobara si el usuario ha pulsado el clic izquierdo o el derecho y se encargara de gestionar y reproducir las animaciones secuencialmente.

Por un lado tenemos la función comprobarAtaque que se llama solo si en el update detectamos un clic y no se está moviendo y gestiona la máquina de estados, añadiendo a la variable ataquesiguiente el nombre de la siguiente animación a reproducir y creando un delay de espera después de un ataque fuerte.

Después comprábamos en cada frame si la animación ha acabado de reproducirse, en ese caso asignamos la animación siguiente como la actual, en el que caso que tengamos una preparada, sino volveremos al estado idle.

Para que funcione además hay que añadir en nuestro caballero un componente Animation, donde podemos apreciar la animación actual y una lista con el resto de animaciones (ilustración 26), para añadir una animación a la lista basta con aumentar el tamaño de la lista (size) y arrastrar la animación hasta el elemento que queramos o podemos hacer clic en el círculo que hay al lado de cada elemento y elegirlo a través del selector.

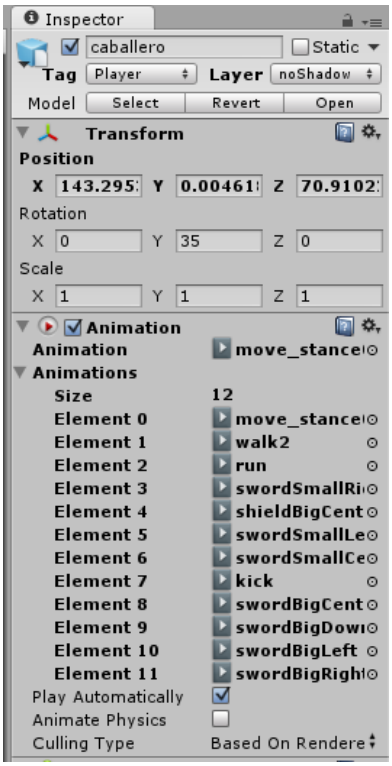

Ilustración 26: Lista de animaciones en el componente Aniamtion

Por otro lado en script podemos inicializar estas animaciones con algunos valores como la velocidad a la que se reproducen (ilustración 27).

```
function Start ()
{
    animation["swordSmallRight"].speed = swordsmallSpeed;
    animation["swordSmallLeft"].speed = swordsmallSpeed;
    animation["kick"].speed = kickSpeed;
    animation["swordSmallCenter"].speed = swordsmallSpeed;
    animation["shieldBigCenter"].speed = shieldSpeed;
    animation["swordBigLeft"].speed = swordbigSpeed;
    animation["swordBigRight"].speed = swordbigSpeed;
    animation["swordBigCenter"].speed = swordbigSpeed;
    animation["swordBigDown"].speed = swordbigSpeed;
}
```

Ilustración 27: Inicialización de la velocidad de las animaciones

Para encadenar animaciones deberemos estar al tanto de si una animación actual ha terminado para pasar a reproducir la siguiente, esto lo haremos mediante la función animation.IsPlaying(attackAnimation) donde recibe por parámetro la animación actual, una vez terminada asignaremos a attackAnimation la siguiente animación.

Para reproducir una animación, podemos usar diversos métodos de unity según la situación:

animation.Play(attackAnimation) o animation.CrossFade(attackAnimation)

Lo importante entonces es construir una máquina de estados sólida en la función update para que solo reproduzca animación cuando nosotros lo deseemos y las reproduzca una después de la otra sin entorpecerse. Es más complicado de lo que parece ya que dentro del update no podemos hacer utilizar la función WaitForSeconds por lo que si queremos generar una espera deberemos llamar a una función nuestra que modifique una variable y establecer una condición en el update, por ejemplo la espera al terminar un combo (ilustración 28).

```
function finalizarCombo ()
{
    fin = true;
    yield WaitForSeconds(3.5);
    fin = false;
}
```

Ilustración 28: Función que finaliza el combo

Por ultimo podemos comunicarnos con otros scripts y ejecutar sus funciones, en este caso enviaremos un mensaje al script que controla el movimiento del jugador para modificar una variable y que por condición el jugador no se pueda mover mientras estemos realizando un ataque.

SendMessage("isMovable", true);

Esta función los que hace es llamar a la función isMovable con el parámetro true, una vez hayamos acabado de atacar volveremos a llamarla con valor false para poder volver a movernos.

```
SendMessage("isMovable", false);
```

Pero esta función no solo sirve para scripts del mismo objeto, si tenemos una referencia de otro gameobject, podemos enviarle un mensaje.

NPC

Creación de huesos

Para la creación de los huesos, vamos a ver seguir estos pasos para hacer un breve ejemplo sobre un cilindro (ilustración 29) , en el cuadro de opciones de la derecha nos mantenemos en la pestaña “Creation” que tiene forma de estrella y luego vamos a la última opción que es Geometry y seleccionamos cylinder, mantenemos clic izquierda para el radio del círculo, soltamos movemos el ratón para elegir la altura y hacemos clic izquierdo para acabar.

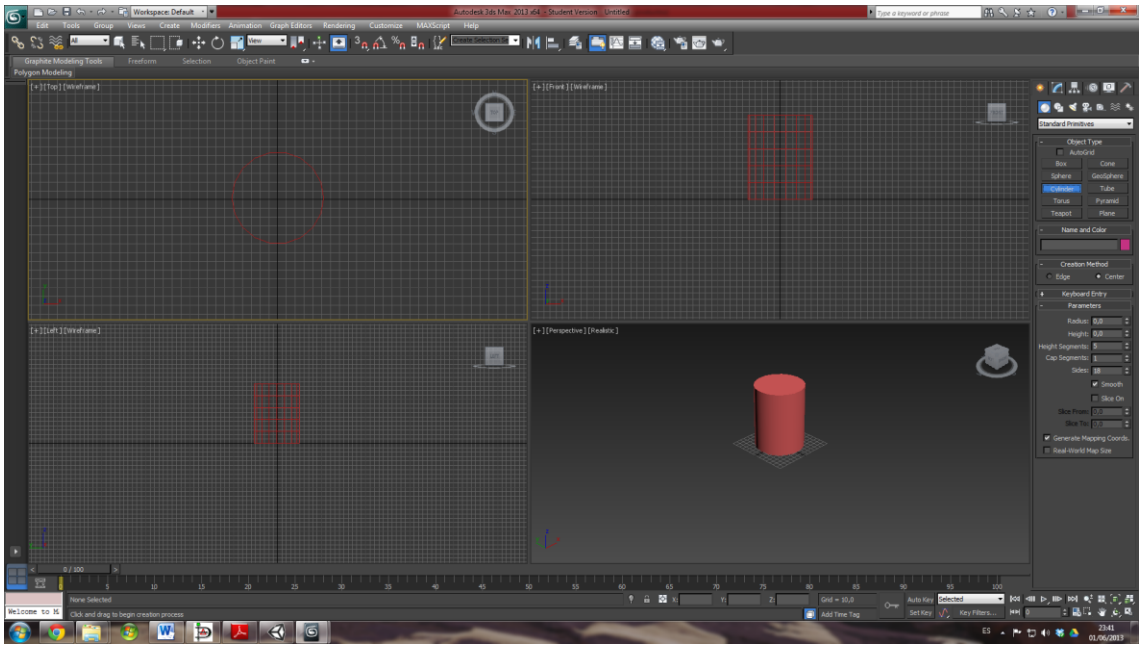

Ilustración 29: Cilindro creado en 3ds max

Para ello en el cuadro de opciones de la derecha nos mantenemos en la pestaña “Creation” que tiene forma de estrella y luego vamos a la última opción que es System (ilustración 30).

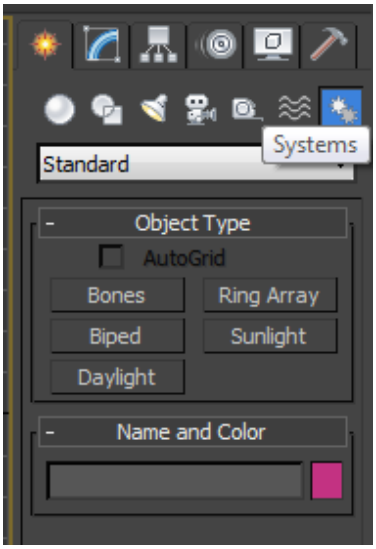

Ilustración 30: Pestaña Systems con opciones para añadir huesos

Como podemos ver aparece la opción Bones, que es la que vamos a utilizar así que pulsamos y hacemos clic izquierdo sobre la vista que queramos para empezar a situar el hueso allá donde pulsemos, veremos que ya aparece el primer vértice del hueso y podemos mover el otro vértice del hueso a la posición deseada, podemos concatenar tantos huesos como queramos, para acabar hacemos clic derecho (ilustración 31).

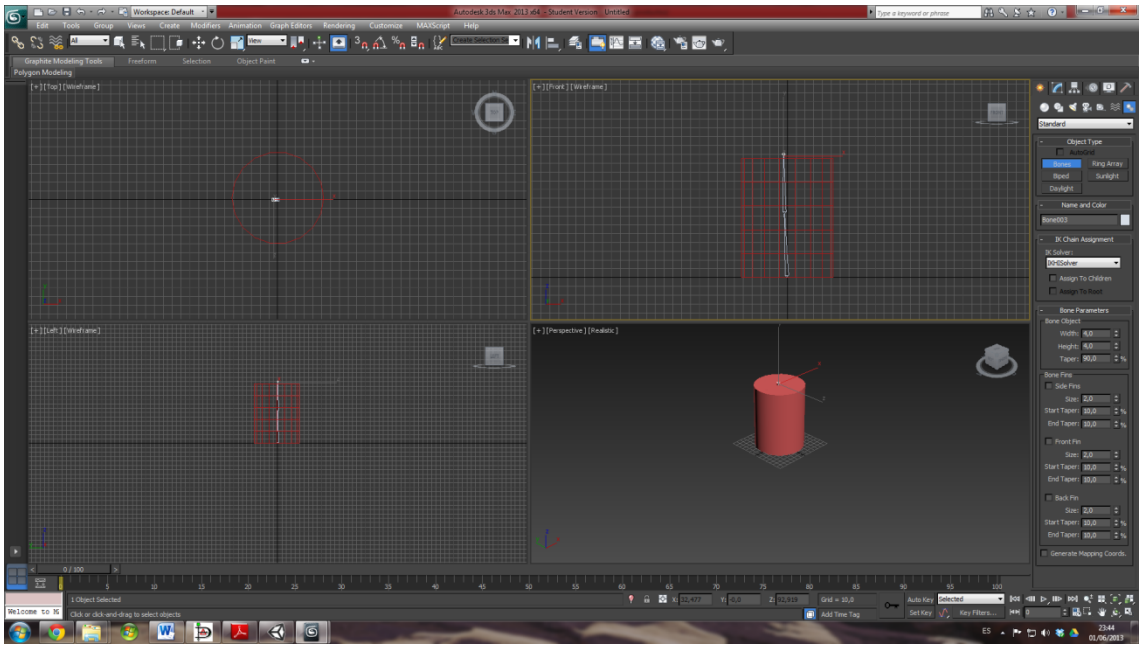

Ilustración 31: Cilindro con los huesos incorporados

Finalmente hay que unir los huesos al objeto, para eso vamos a la pestaña modify, justo la de al lado que parece un arco azul, pulsamos sobre el combobox Modifier List con el cilindro seleccionado y elegimos skin (ilustración 32).

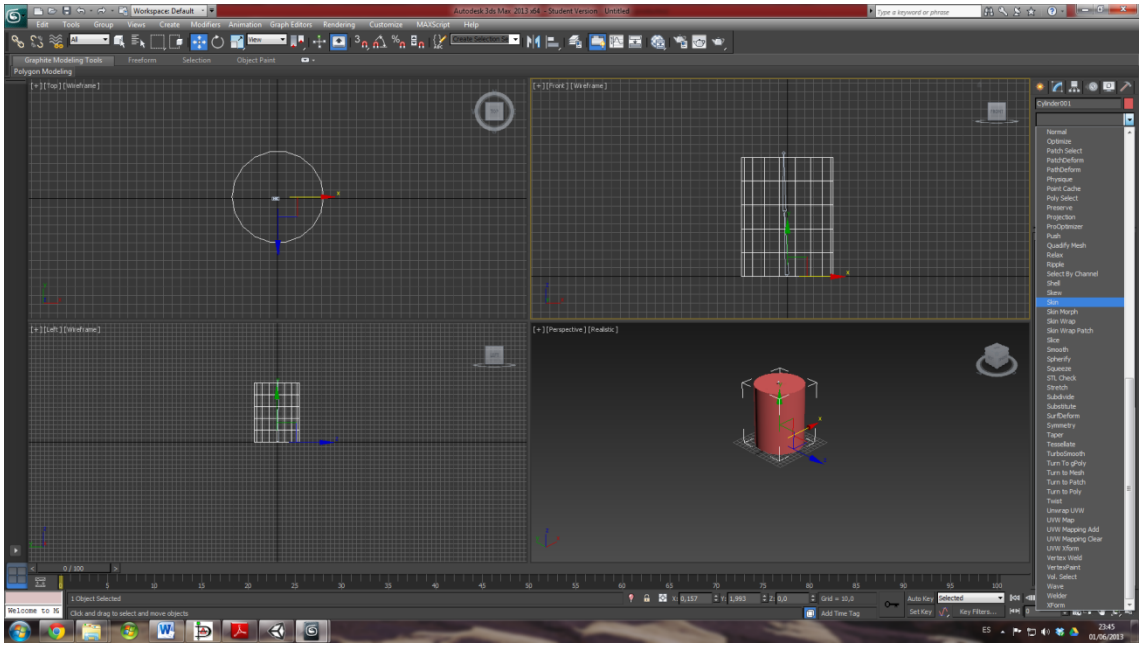

Ilustración 32: Menú para añadir el modificador skin

Para acabar pulsamos Add en la misma pestaña y elegimos los huesos que queremos añadir (ilustración 33)

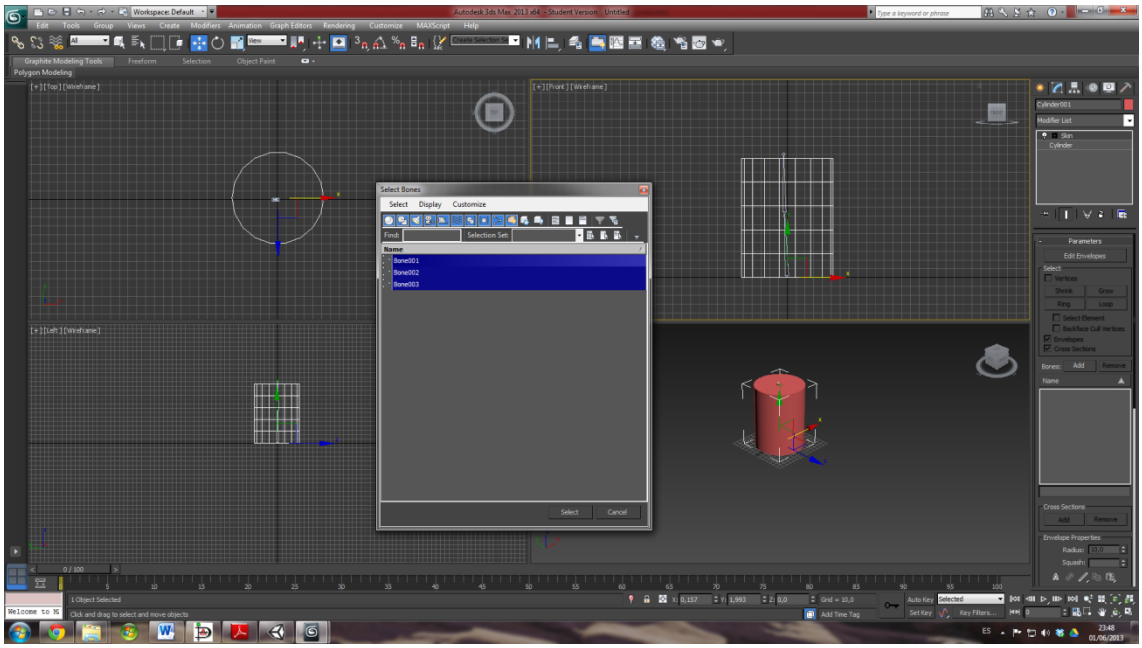

Ilustración 33: Menú añadir huesos

Si todo ha ido bien podemos seleccionar un hueso y al aplicarle una rotación el objeto se deformara (ilustración 34).

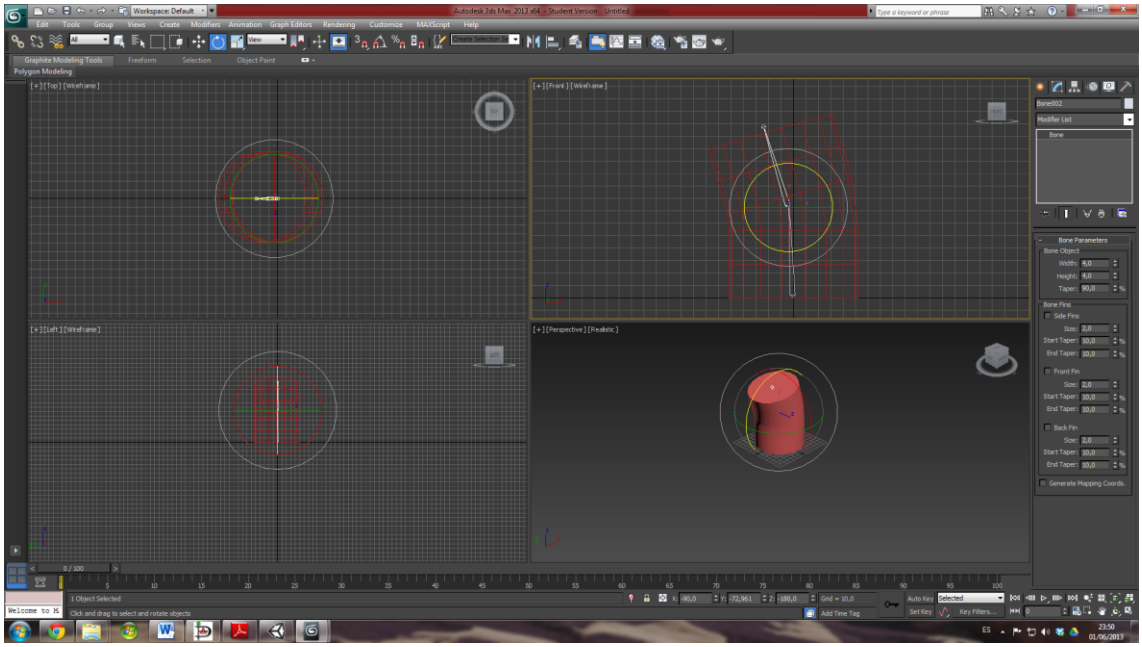

Ilustración 34: Geometría del cilindro modificada por la rotación de un hueso

Como extra comentar que se puede editar el nivel de afectación de un hueso sobre la malla (ilustración 35) si editamos los pesos, para ello bajamos hasta q veamos la opción weight table o tabla de pesos.

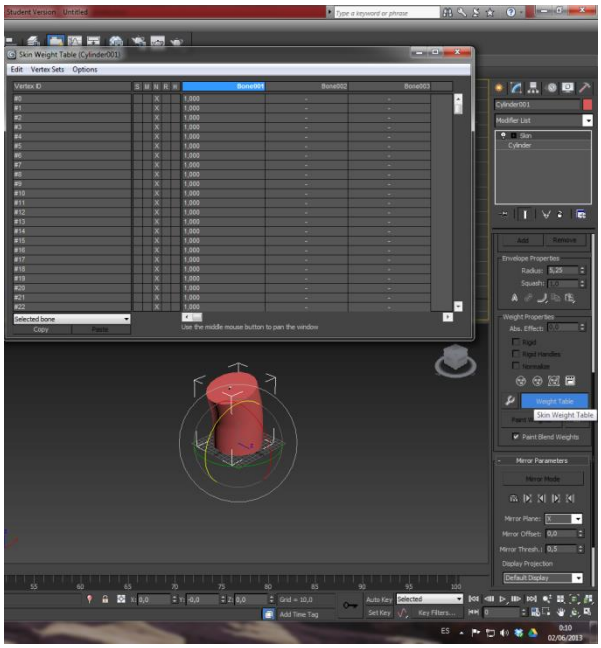

Ilustración 35: Tabla para modificar el alcance de un hueso sobre la geometría

Animaciones

Aunque parezca sencillo a primera vista, es proceso un proceso tedioso, ya que aunque el movimiento sea simple si queremos un resultado realista hay muchos movimientos involucrados de forma implícita y de los cuales a veces no somos conscientes.

Por ejemplo el dragón a la hora de caminar no solo debería de mover las piernas, sino también la cola, el torso y la cabeza, sino obtendríamos un movimiento robótico. Por suerte 3ds max con su herramienta para animaciones nos facilita la tarea interpolando movimiento y dando la posibilidad de duplicar key frames, para no ir frame a frame y hacer el proceso un poco menso laborioso.

Esta es la parte de la ui que nos interesa para animar (ilustración 36), a parte de las vistas para rotar los huesos y generar el movimiento. Aquí vemos lo que representaria la linea del tiempo a lo largo de la animacion. Y disponemos de controles para cambiar la longitud, reproducir la animacion o despalzarnos por los frames.

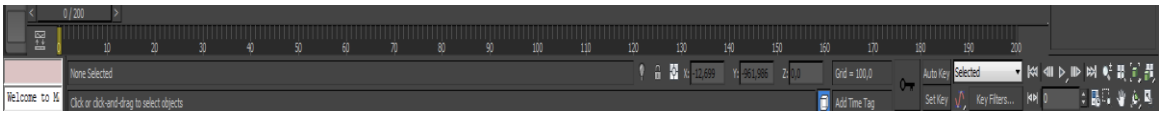

Ilustración 36: Barra de frames

Lo primero que debemos hacer es pulsar Auto key (ilustración 37), después nos desplazamos con la barra hasta el frame que deseemos y aplicamos las rotaciones a los huesos, no hace falta ir frame a frame como he comentado antes si nos vamos al frame 15 y rotamos un hueso 3ds max interpolara el movimiento entre los frames 0 y 14 y no será necesario editar a no ser que haya que aplicar alguna corrección.

Así pues al seleccionar un hueso o un conjunto de huesos veremos todos los cambios que le hemos aplicado, si elegimos un key frame podemos eliminar con suprimir o hacer una copia pulsando shift clic sobre el frame y arrastrándolo hasta el frame donde queremos copiar el movimiento, función interesante para repetir ciertos ciclos en una animación.

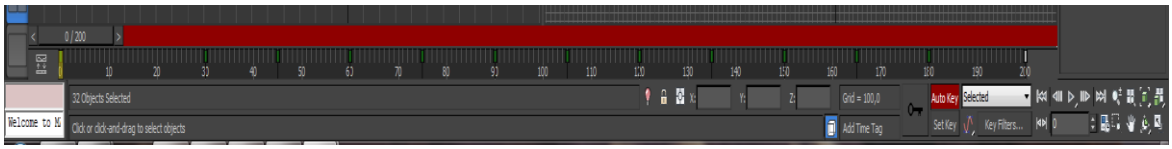

Ilustración 37: Ajuste automático de frames

Tal como se ve en esta imagen, tenemos en verde lo que podemos ver y en rojo lo que no, la forma más sencilla de implementar este quebradero de cabeza, es utilizar el raycasting, en este caso solo es necesario lanzar un rayo hacia nuestro objetivo, lo primero es comprobar que este en el ángulo adecuado y después lanzamos el rayo:

Mediante una sencilla resta de vectores obtenemos la dirección hacia donde hemos de lanzar el rayo, posición del jugador menos las del enemigo:

rayDirection = playerObject.transform.position - transform.position;

Aquí verificamos el angulo:

Vector3.Angle(rayDirection, transform.forward) < fieldOfViewRange);

Finalmente lanzamos un rayo en esa dirección a una distancia límite determinada (hasta que distancia queremos que vea) y guardamos el resultado en la variable hit

Physics.Raycast (transform.position+Vector3.up*0.8, rayDirection, hit, rayRange)

Una vez verificado el ángulo podemos lanzar el rayo y saber fácilmente si ha impactado contra algo que es nuestro jugador o algo que no lo es (ilustración 38), mediante hit.transform.tag == "Player", no necesitamos concretar más.

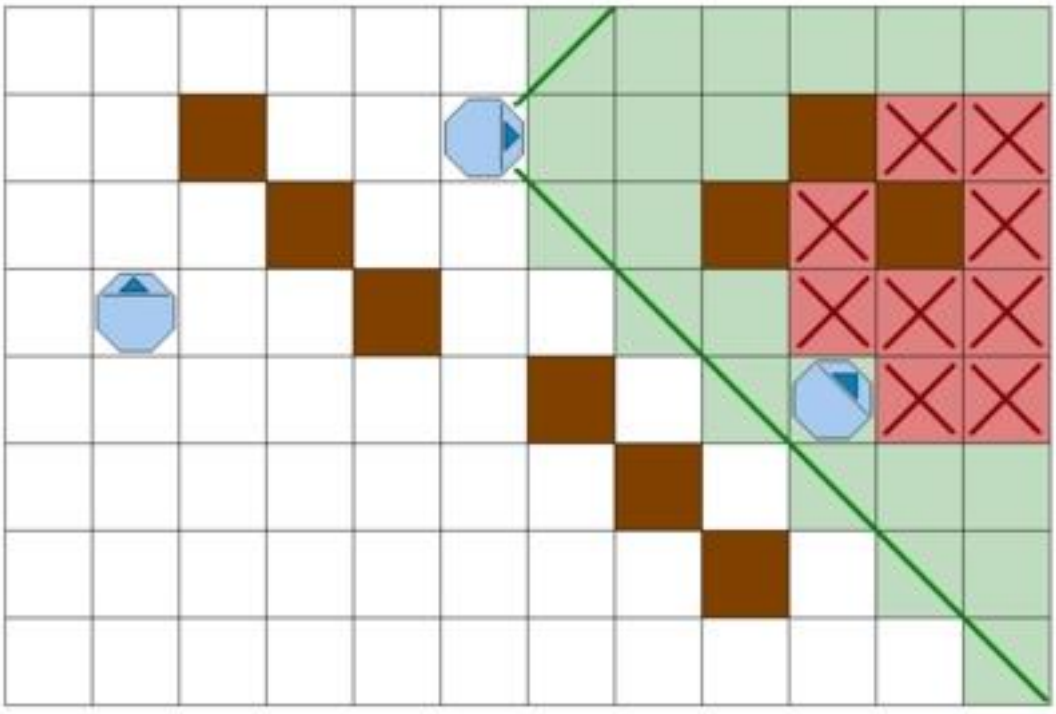

Ilustración 38: Campo de visión final

Escenario

Para desarrollar el escenario, se utilizara la herramienta terrain de unity que permite editar el terreno y las texturas de forma fácil y potente.

En el menú de arriba seleccionamos la opción para crear un terreno y después a set resolution para elegir las medidas (ilustración 39):

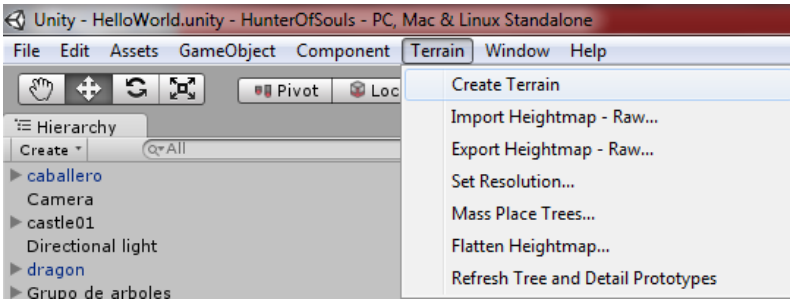

Ilustración 39: opción crear terreno y cambiar el tamaño del terreno en el menú de unity

Después seleccionamos el terreno a través del script podremos hacer todo tipo de cosas sobre el terreno (ilustración 40), entre las que se incluyen modificar el terreno para hacer montañas o desniveles más pequeños así como la colocación de texturas, árboles u otros detalles. Después elegimos el tipo de pincel para aplicar el efecto de formas y tamaños.

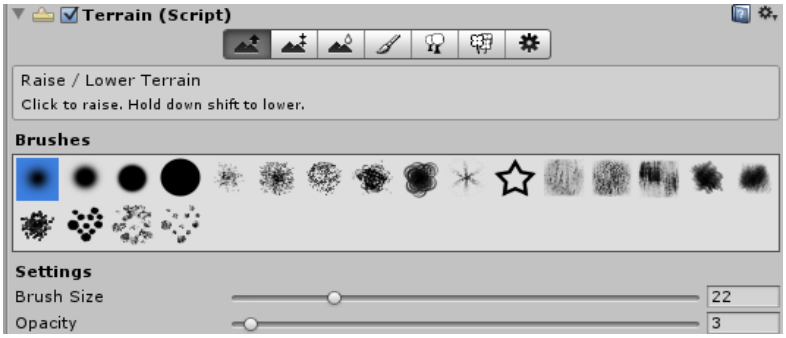

Ilustración 40: Opciones y tipos de pinceles par aplicar cambios al terreno

En la escena veremos una zona azul donde se aplicaría el efecto, solo tendremos que hacer clic izquierdo para aplicar o shift + clic izquierdo para deshacer.

Tambien deberemos asociarle un skybox (ilustración 41)que es como una cupula circular alrededor del escenario para representar cielos o paisages sin necesitad de generar montones y montones de Assets y que aportan un buen aspecto estetico al juego, para ello en la camara elegimos la opcion skybox en clear flags y asociamos un componente skybox donde seleccionamos nuestro cielo, podemos utilizar algunos que nos proporciona unity, en este caso se utiliza un cielo de aspecto nuboso.

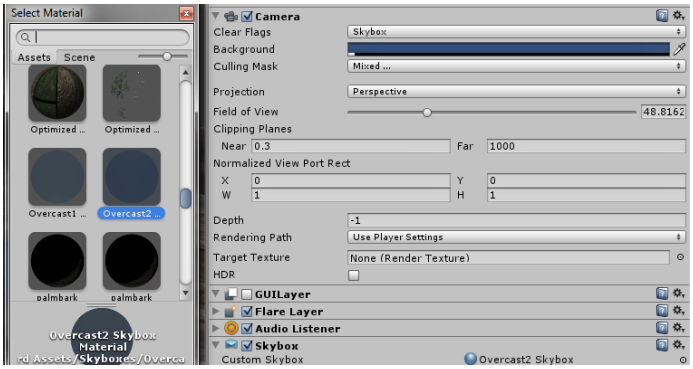

Ilustración 41: Selección de una skybox para la cámara

Para llevar a cabo esta opción lo que necesitamos es primero crear un plano al cual le vamos a asociar la textura con la barra de vida vacía (ilustración 42).

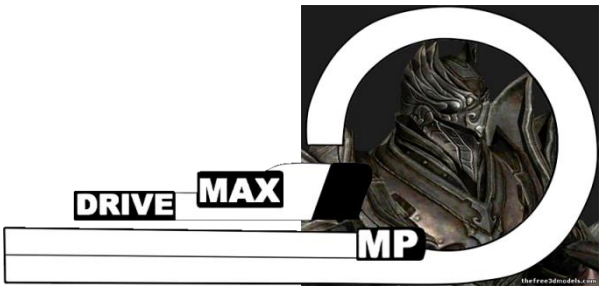

Ilustración 42: Aspecto de la barra de vida vacía

El segundo paso es rellenar la barra de vida del jugador pero siguiendo un gradiente de transparencia (ilustración 43), para esto podemos usar herramientas como photoshop. Las otras barras y el círculo donde se ve el personaje han de ser transparentes para que al ponerlo delante se siga viendo la textura anterior y asignamos esta textura en un plano diferente que colocaremos ligeramente más adelante.

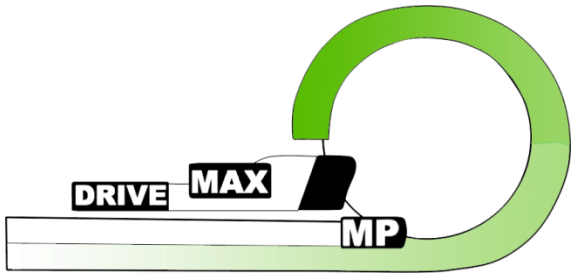

Ilustración 43: Aspecto de la barra de vida rellenada siguiendo un gradiente

También es importante una vez hayamos añadido el componente al plano indicar que el shader es transparent cutout. Podemos apreciar una slide bar llamada Alpha cutoff, esta es la parte importante que nos va a permitir regular la transparencia (ilustración 44) para que se vea uno u otro (ilustración 45).

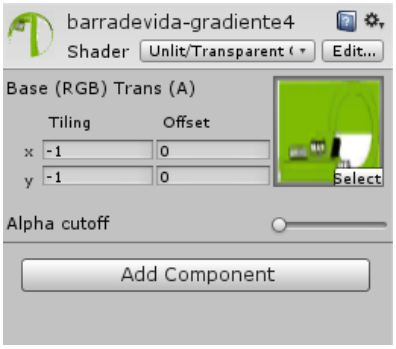

Ilustración 44: Opciones de transparencia sobre la textura

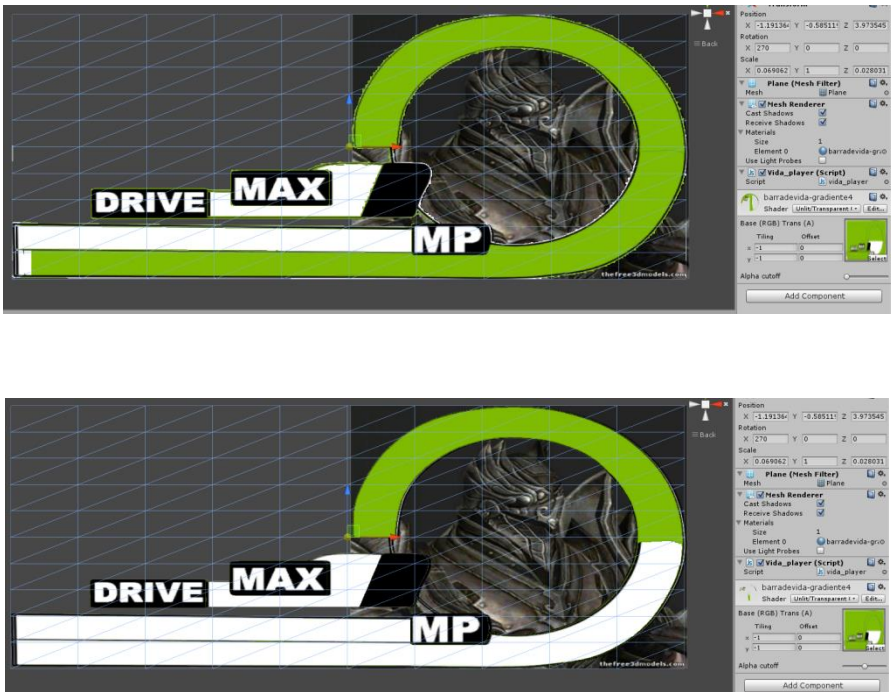

Ilustración 45: Barra de vida en funcionamiento

Esto lo podemos controlar por código añadiendo un script y eso es lo que vamos a hacer, en la imagen aparece llamado como vida.js. Básicamente lo que hace es recoger la vida de nuestro personaje y modificar este valor de forma relativa.

```
function mostrarvida (vida: int) {

        renderer.material.SetFloat("_Cutoff", -Mathf.InverseLerp(0,500,vida));

}
```

Cuando recibamos daño el script que controla la vida llamara a esta función para refrescar la vida que debemos mostrar, si el personaje tiene vida en un rango distinto de 0 a 100, solo tendremos que realizar una interpolación como se muestra en la función.

Otro punto importante es colocar una cámara nueva que única y exclusivamente debe renderizar estos dos planos y debe hacerlo siempre después de la cámara principal, así que creamos un nuevo layer al que llamaremos GUI y se lo asignaremos a los planos (ilustración 46) y a la cámara.

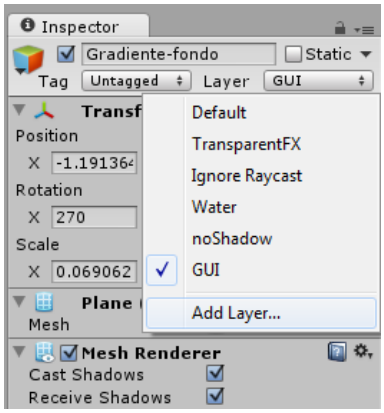

Ilustración 46: Ajustando el renderizado del plano

En la cámara cambiamos el renderizado y la profundidad que sea mayor a la de la cámara principal (ilustración 47).

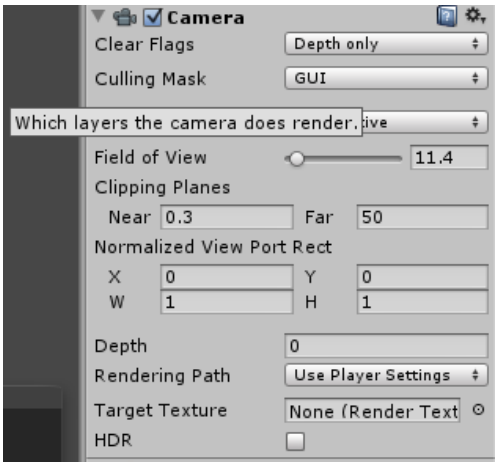

Ilustración 47: Ajustando renderizado en la cámara

Lógicamente debemos añadir tanto al jugado como al caballero un fichero que contenga información sobre su salud (ilustración 48), así como una referencia al plano para que le informe de los cambios en la vida.

Ilustración 48: Script que gestiona la vida del personaje

Sonido

Para el sonido declararemos en el script variables de tipo AudioClip y desde el inspector seleccionaremos que sonido contiene esa variable.

Para hacer que suene un sonido podemos utilizar este breve código:

```
if (struckSound) {
    AudioSource.PlayClipAtPoint(struckSound, transform.position);
}
```

Esto lo que hace es comprobar que hayamos asignado un sonido y en caso positivo, hacemos que suene en una posición determinada. Esto en caso que queramos añadir un sonido determinado por código, cuando se produce a causa de un evento.

También podemos arrastrar sonidos que sean estáticos por la escena o asociados a un gameobject determinado por ejemplo a la cámara en caso de la música que acompaña la batalla.

Colisiones

Para empezar vamos a ver como controlaríamos la interacción en que el usuario golpea al dragón, así que creamos un gameobject vació al que llamamos espada y le añadimos un componente box collider al que debemos márcale la casilla Is Trigger y muy importante un rigibody ya que si no se van a detectar bien las colisiones, podemos desmarcar la gravedad que no nos interesa y muy importante marcar el Is kinematic (ilustración 49).

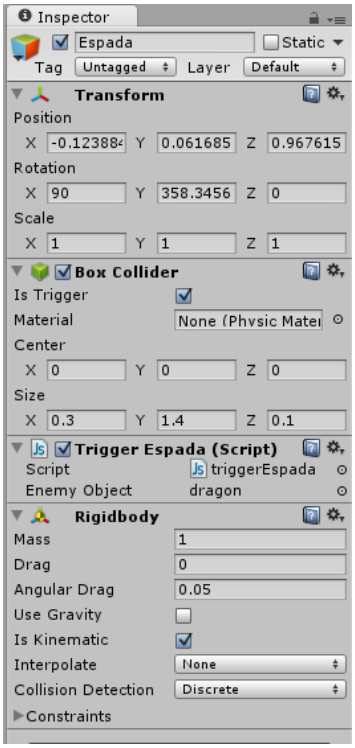

Ilustración 49: Ajustes necesarios para detectar bien las colisiones

Después ajustaremos la posición y medidas del collider para que se adapten a las de nuestra arma (ilustración 50) y saber exactamente cuando hemos tocado al enemigo con el filo de nuestra arma.

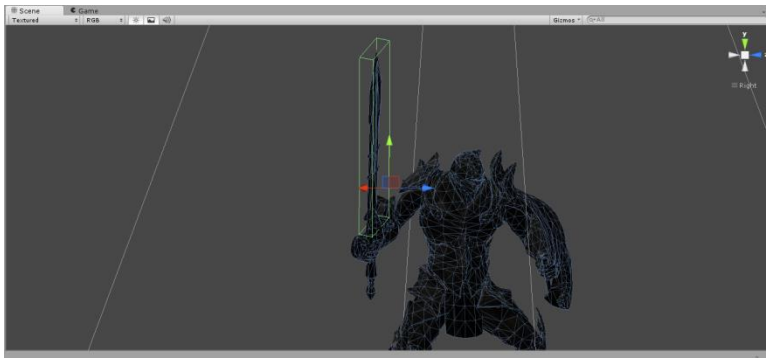

Ilustración 50: Collider colocado a medida para al espada

Ahora hay que añadir un script al gameobject de la espada para tratar la colisión, con las medidas oportunas, para eso usaremos la función onTriggerEnter que se llamara automáticamente cuando al entre en nuestro collider, es importante comprobar contra que hemos entrado en contacto para asegurarnos que estamos tratando la situación adecuada, por eso mediante el tag o el nombre del gameobject debemos asegurarnos que lo que ha entrado en el collider es el dragón y no nosotros mismos o el escenario.

```
function OnTriggerEnter (myTrigger : Collider) {
    if(myTrigger.gameObject.name == "dragon"){
        //Debug.Log("dragon went through!");
        enemyObject.SendMessage("ApplyDamage", 5);
    }
}
```

Una vez sabemos a ciencia cierta que hemos entrado en contacto con el dragón es cuando le enviamos un mensaje al dragón para decirle que le hemos hecho daño y que debe restarse vida, también podemos aprovechar para enviar un mensaje para reproducir una animación en reacción al golpe, para reforzar el realismo.

SendMessage("impacto");

Y en el script de ataque del caballero, definimos la función y reproduciremos la animación con el método play que para todas las animaciones que este reproduciendo el objeto y comenzara a reproducir la animación que le hemos indicado.

```
function impacto ()

{

        animation.Play("react");

}
```

Otra cosa interesante que podemos hacer es definir un sistema de partículas que simule el sangrado (ilustración 51) al recibir un golpe y crear una instancia donde se produzca la colisión. Con los colliders no tenemos tanta precisión así que en lugar del punto exacto de la colisión daremos la posición en la que se encuentra nuestro collider.

Instantiate(blood, pos, Quaternion.identity);

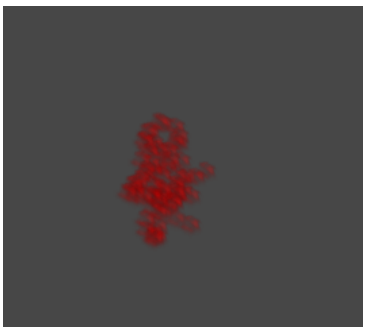

Ilustración 51: Sistemas de partículas de sangre

También vamos a hacer el daño que recibe el dragón por parte del caballero sea variable, de manera que se basara en la longitud del combo, siendo el más largo el que resultara más potente.

Cada vez que apliquemos daño obtendremos el número de ataques que llevamos hasta el momento que será el daño de ese golpe, de manera que el primer golpe causara 1 de daño , el siguiente 2 y así consecutivamente, pero a partir del 3 golpe premiaremos con dos puntos extra de daño a los golpe siguiente de manera que el cuarto golpe causara 6 de daño en lugar de 4.

```
damage = caballero.GetComponent("ThirdPersonCharacterAttack").numAttacks;
Debug.Log(damage+" pierde vida");
if(damage > 3) {
    damage += 2;
}
health -= damage;
```

De esta manera premiamos los combos largos y el jugador debe aprender una cierta estrategia, decidiendo cuando es el mejor momento para iniciar el combo, pero sin quedarse muy expuesto.

Pero no hemos terminado aquí, puede ser que algún collider entre en contacto con el adversario sin que les estemos atacando y no sería justo que el adversario percibiera daño. Tenemos dos opciones o bien activar los colliders solo cuando se produzca un ataque o bien avisar con un mensaje al adversario cuando se está atacando para que sepa realmente que el daño ha sido valido.

He usado ambas opciones, con el usuario los mensajes ya que el tipo de ataque suele ser un combo y además por las dimensiones es fácil que entre en contacto con el dragón o que permanezca el collider en contacto más tiempo. El dragón en cambio usa ataques más amplios y hay zonas como la boca que por su posición son inaccesibles si no se produce el ataque y también cuanta con ataques de zona donde el daño depende del tiempo que permanezca el usuario dentro del collider.

Para los ataques en zona usaremos la función onTriggerStay en el script que a diferencia del OnTriggerEnter se llama constantemente mientras haya algo dentro del collider.

```
var playerObject : GameObject;

function OnTriggerStay (myTrigger : Collider) {
    if(myTrigger.gameObject.name == "caballero"){
        //Debug.Log("caballero me quemo");
        playerObject.SendMessage("ApplyDamage", 1);
    }
}
```

Ahora solo falta situar el gameobject, en la jerarquía de objetos al mismo nivel que está situado el hueso que controla los movimientos de la espada (ilustración 52) de esta manera recibirá las mismas transformaciones y rotaciones y seguirá perfectamente la trayectoria de la espada incluso durante las animaciones.

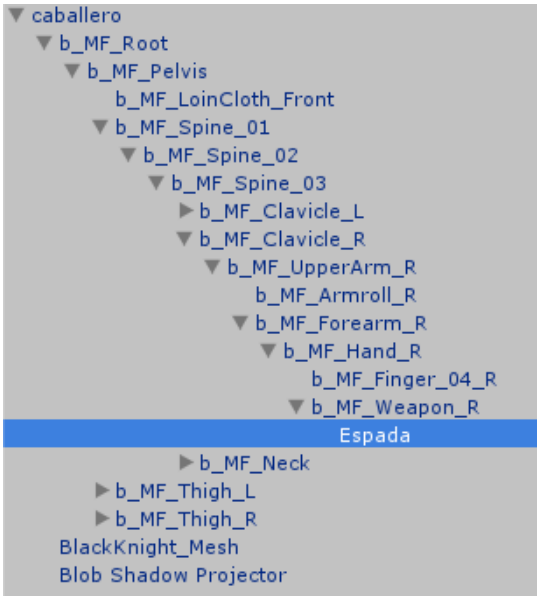

Ilustración 52: Collider de la espada colocado en el hueso que controla el movimiento de la espada

El primero de todos es el emisor (ilustración 52) que se encarga de producir las partículas con un tamaño, energía, cantidad de partículas que se emiten y velocidad determinada entre otros ajustes. En definitiva estos ajustes son un poco subjetivos dependiendo la cantidad de fuego que queramos generar y su dirección, así podremos crear desde una hoguera hasta una feroz llamarada.

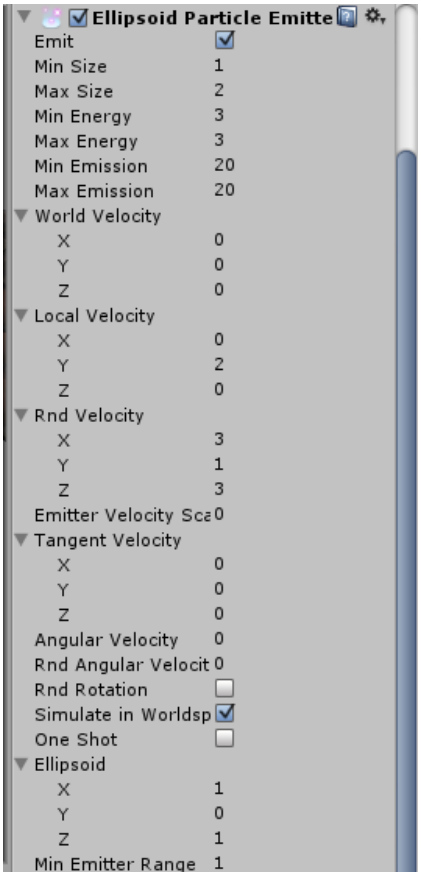

Ilustración 53: Componente que emite partículas

Otro componente importante es el Particle animator (ilustración 54), que como su nombre indica hace que las partículas no sean estáticas sino a qué medida entre el momento que se crean y se destruyen hay una breve interacción en subida en la que pueden cambiar de color o cambiar su desplazamiento.

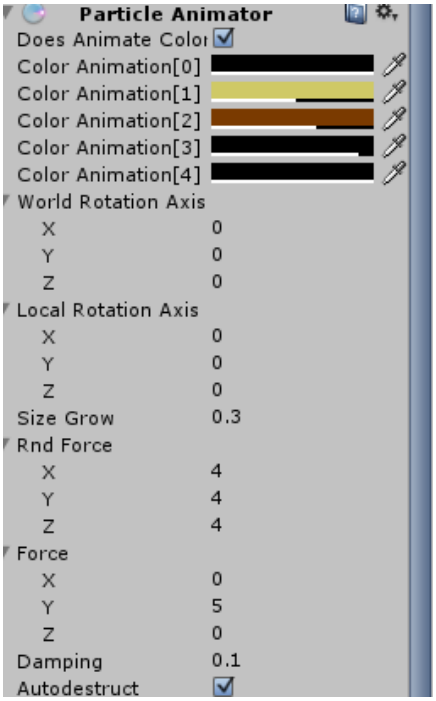

Ilustración 54: Componente que modifica las partículas

Después añadiremos el Particle Renderer para poder renderizar las partículas donde podemos definir cómo interactúan con las luces, a veces suelen ir acompañados también de una luz para caracterizar su color o visibilidad.

Por ultimo podemos añadir una textura (ilustración 55), quizás el punto más difícil de encontrar o crear para obtener un buen resultado.

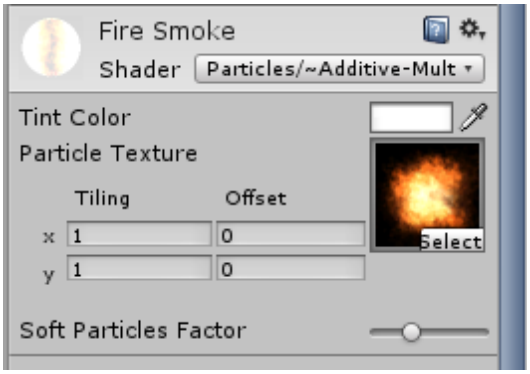

Ilustración 55: Textura que da más realismo al fuego

Resultados

Estructura del proyecto

Aquí muestro como se ha organizado el proyecto (ilustración 56) de forma fácil he intuitiva, podemos agrupar el contenido relacionado en carpetas, simplemente hacemos clic derecho y pulsamos en “create” -> “New Folder” y después arrastramos el contenido que queramos dentro.

De esta manera en jugador tendremos todo lo relacionado con los modelos, animaciones y scripts, etc y lo mismo con los otros contenidos.

Ilustración 56: Aspecto de cómo está organizado el proyecto

También se podría haber organizado por contenido del mismo tipo, es decir una carpeta para scripts otra para modelos, etc. Todo depende de las dimensiones del proyecto y la cantidad de elementos que contenga.

Jugador y colisiones

Aquí podemos ver todos los componentes del caballero(ilustración 57), entre los que se encuentran 3 collider para detectar los ataques, un carácter controller para mover al jugador y recibir colisiones por parte del dragón, una position light y un proyector para hacer una sombra:

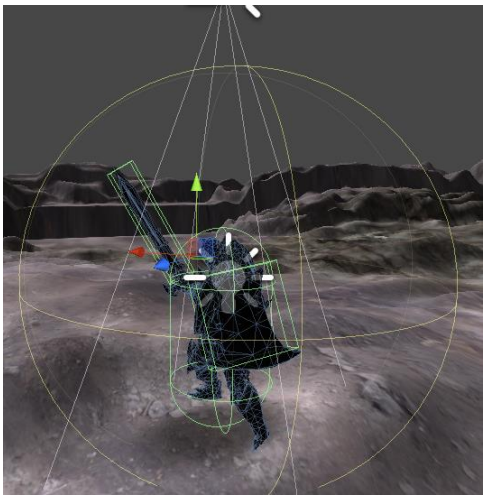

Ilustración 57: Personaje principal con todos los componentes asociados

Con una cámara (ilustración 58) que nos sigue comodamente por el escenario o que no pierde al enemigo de vista:

Ilustración 58: Campo de visión desde los dos tipos de cámara disponible

Ademas es capaz de realizar combos de hasta 5 ataques (ilustración 59), aquí veremos un breve ejemplo del combo mas largo, que consta de 4 ataques rapidos y un golpe fuerte:

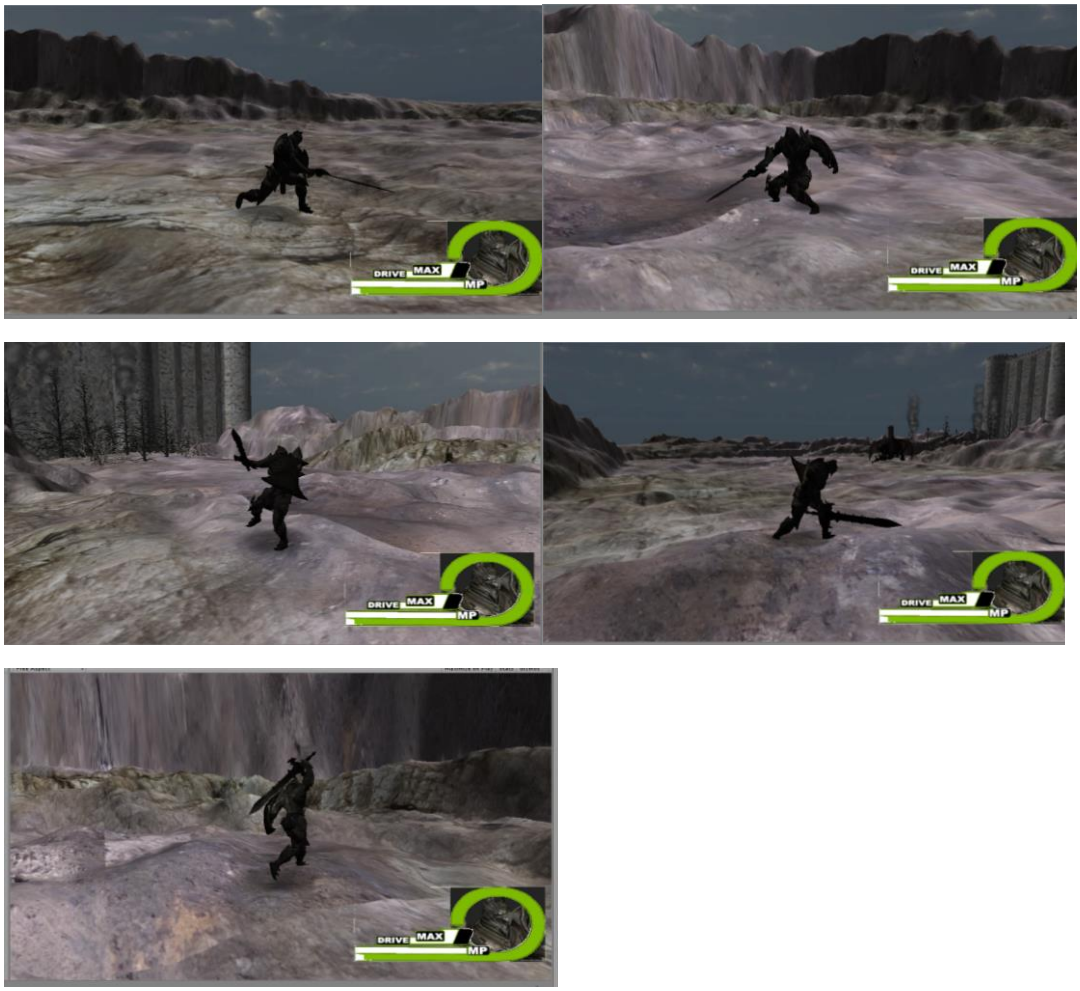

Ilustración 59: Secuencia de un combo del jugador

Además cuando el jugador hace daño al dragón se crea un sistema de partículas para simular que pierde sangre (ilustración 60) al hacerle una herida.

Ilustración 60: Imagen donde se percibe sangre en la colisión de la espada con el enemigo

Y al recibir el jugador el golpe reacciona, con una animación (ilustración 61).

Ilustración 61: El caballero reaccionando ante el ataque del dragón

Huesos en el dragón

Con un total de 131 huesos distribuidos por el modelo del dragón (ilustración 62) he conseguido que pueda realizar gran variedad de movimientos de forma más o menos realista, que abarcan las piernas , la cola, el torso, las alas, el cuello y la boca, como podemos apreciar en la imagen. Ya podemos pasar al siguiente paso que es realizar las animaciones.

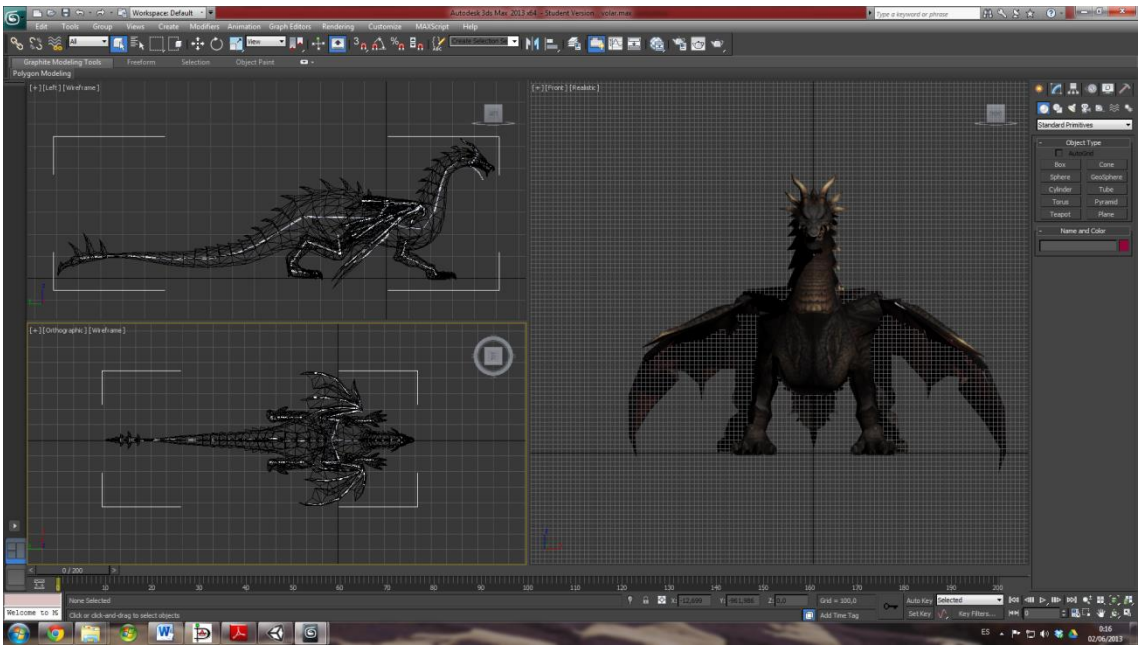

Ilustración 62: huesos que permiten mover al dragón

Animaciones en el dragón

Como ya he comentado antes podemos aplicar casi cualquier movimiento del dragón, aquí vemos un ejemplo de animación (ilustración 63).

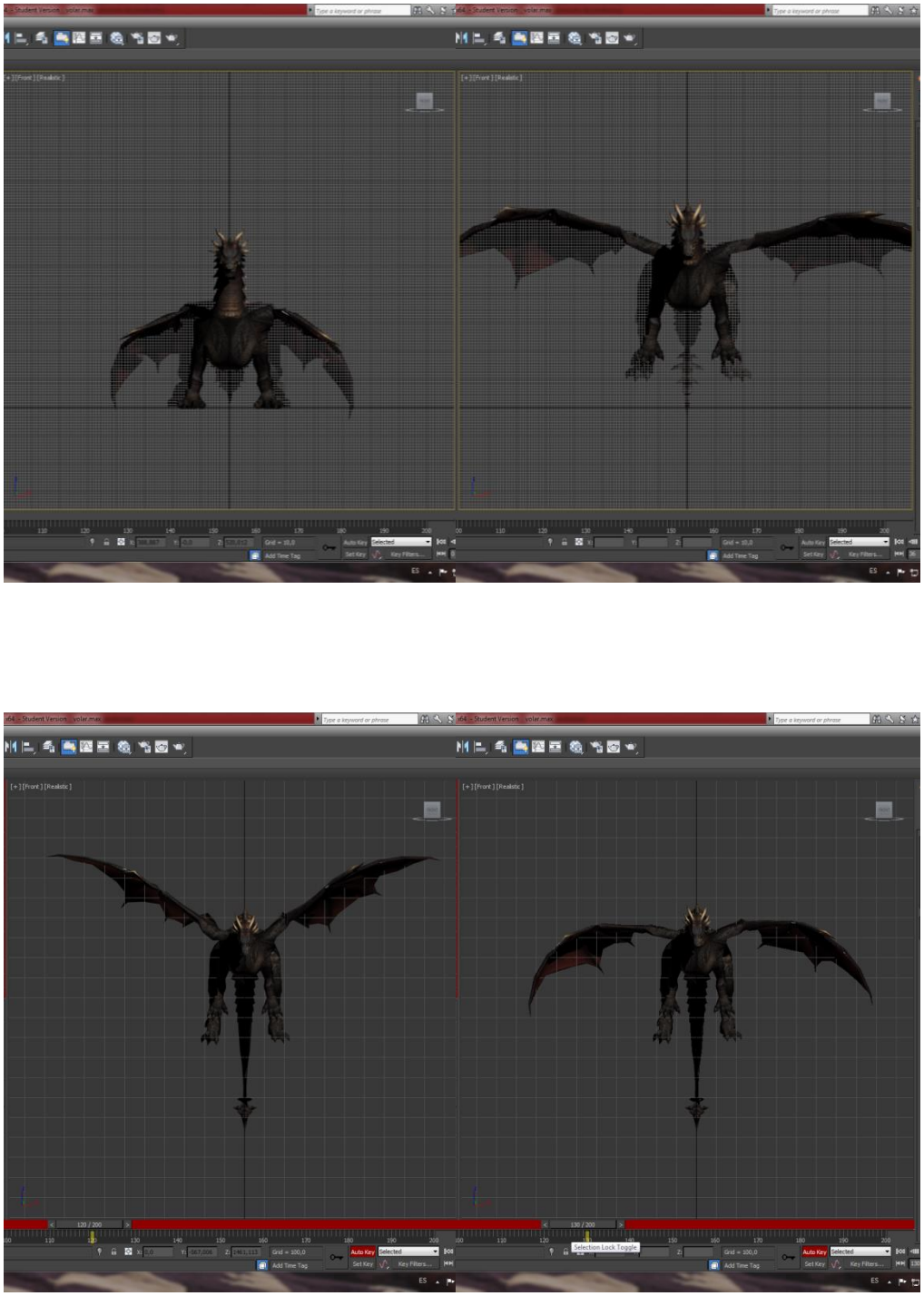

Ilustración 63: Algunos frames de la animación en que el dragón vuela

Dentro del juego podemos apreciar como se eleva, mientras activamos un efecto de particulas para simular que el dragon nos lanza fuego hacia abajo (ilustración 64).

Ilustración 64: Dragón volando y lanzando fuego

Escenario

Este es el aspecto final del escenario compuesto por un castillo deshabitado por la destruccion del dragon sobre la zona, donde solo quedan arboles quemandose y monstañas de escombros y cenizas.

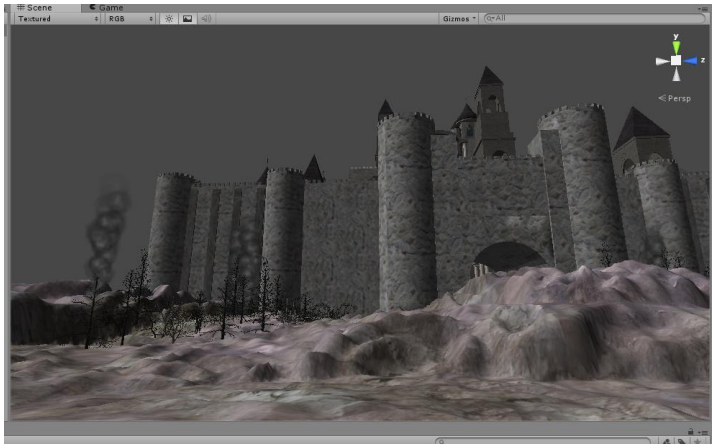

Ilustración 65: Aspecto final del escenario, suelo con cenizas, árboles quemados …

El escenario se caracteriza por una rugosa explanada donde se produce la batalla (ilustración 66), una zona relativamente amplia para que le jugador tenga espacio para esquivar los ataques de fuego

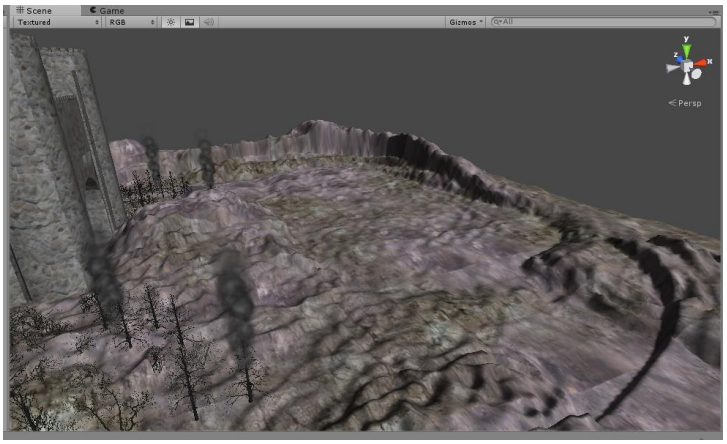

Ilustración 66: Esplanada donde se desarrolla el combate

Conclusiones

Objetivos realizados

- ✓ Se ha conseguido que el combate transcurra de forma fluida
- ✓ El combate contra el dragón es difícil pero equilibrado, animando al usuario a intentar superarse.
- ✓ Se han conseguido animaciones más o menos realistas que hacen el juego más impactante.
- ✓ El diseño del escenario y sonidos, acompañan la batalla.
- ✓ He aprendido a trabajar con un "game engine" bastante potente.
- ✓ Se han abordado todas las facetas de la creación de un videojuego.

Como continuar

Pienso que es un trabajo bastante creativo y casi siempre se pueden hacer cosas, el único límite son los recursos.

- Se podrían desarrollar niveles, habilidades del jugador y una historia con múltiples decisiones propias de los rpg en los que el jugador sigue un desarrollo incremental para no perder el interés del jugador.
- Se podrían mejorar los efectos gráficos y de sonido para despertar la fantasía del jugador.
- Mejorar la Inteligencia artificial e incluir niveles de dificultad.
- Crear un apartado multijugador.

Bibliografía

http://unity3d.com/learn/documentation

http://trinit.es/unity/doc/lecturas/Tutorial%20Unity3D%20-%20Franz%20Huanay.pdf

http://unity3d-es.blogspot.com.es/p/recursos.html

http://www.burgzergarcade.com/hack-slash-rpg-unity3d-game-engine-tutorial

http://unityscripts.blogspot.com.es/p/api-espanol.html

http://thefree3dmodels.com/

http://www.youtube.com/user/MrJocyf/videos

http://www.freesound.org/browse/

http://www.youtube.com/watch?v=qPqITdvLRWs