# DANGEROUS DUNGEONS

Un videojuego por Fausto Sánchez Hoya

# Índice

# 1 INTRODUCCIÓN

En este trabajo se va a desarrollar el videojuego Dangerous Dungeons. Un videojuego 3D desarrollado con Unity 2020.3.18f1 en el que encarnaremos a nuestro personaje "Mike el Todopoderoso" que intenta liberar a su tierra de hordas de zombies que la atacan.

Este videojuego ha pasado por varias fases de desarrollo: primeramente, iba a ser del estilo al videojuego Diablo, pero ha acabado siendo parecido al modo zombies del Call Of Duty con una estética diferente.

¿Conseguirá Mike liberar a su tierra? ¿Conseguirá Fausto programar adecuadamente el mundo de Mike?

Lo iremos viendo…

# 2 DISEÑO Y ESTÉTICA DEL JUEGO

Para empezar, he elegido que la estética del juego será parecida a la del videojuego "Minecraft Dungeons".

Por lo que he decidido utilizar assets creados con Voxel para dar esa apariencia de cubos o "píxel art". La mayoría de estos recursos proceden del autor Maxparta[1] excepto los relacionados con las armas que son de Unity-Fantom [2].

[1] Recursos del autor en https://maxparata.itch.io/
[2] Recursos del autor en https://unity-of-fantom.itch.io/voxel-character-pack-v1

En cuanto a las partículas[3] serán extraídas de assets de la Asset Store al igual que los cielos[4].

Por otro lado, las animaciones de los personajes serán extraídas de Mixamo[5].

[3] Partículas de:
https://assetstore.unity.com/packages/vfx/particles/fire-explosions/procedural-fire-141496
https://assetstore.unity.com/packages/vfx/particles/particle-collection-skj-2016-free-samples-72399#content

[4] Cielos de https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014

[5] Mixamo: https://www.mixamo.com/#/

# 3 DESARROLLO

El desarrollo del videojuego se ha partido en varias fases. El proceso de desarrollo se puede ver a través de un hilo de Twitter en el que he ido subiendo los diferentes avances (especial agradecimiento a mi compañero David Montagut Pamo por la idea).

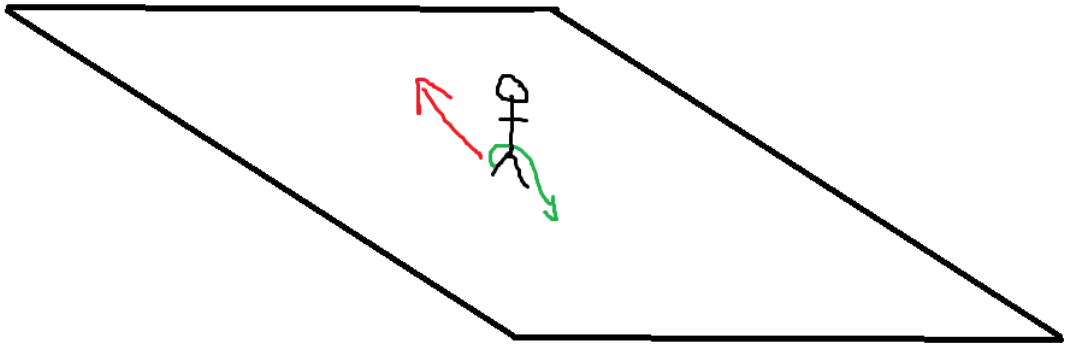
https://twitter.com/Fausto_1802/status/1450809325505728521

## 3.1 EL MOVIMIENTO DE MIKE

En la primera fase del proyecto me he encargado de desarrollar el movimiento que tendrá Mike dentro del videojuego. Para ello he definido las siguientes posibilidades de movimiento: Mike podrá caminar en todas las direcciones, pero siempre andando hacia delante, es decir, si Mike está mirando hacia la parte de arriba de la pantalla y quiere andar hacia la parte de abajo no podrá andar marcha atrás, sino que dará un giro de 180 grados y comenzará a andar hacia delante. No podrá agacharse ni saltar ya que no lo necesitará en el juego.

Rojo: dirección actual.

Verde: camino que seguiría para ir en dirección contraria.



Después de definir cómo va a ser su movimiento he de descargarme las animaciones correspondientes de Mixamo. Estas animaciones estarán en la ruta:

Assets\Personajes\Mike\Animacion\Movimiento

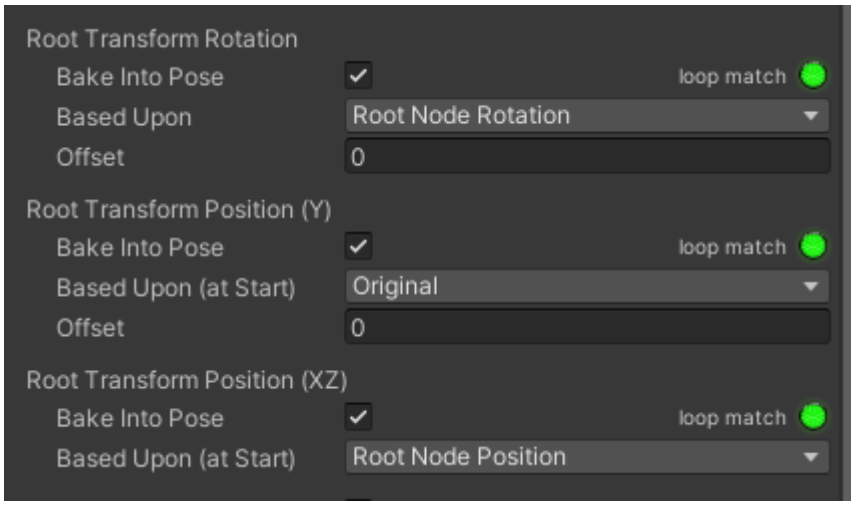
Para que estas animaciones puedan ser utilizadas he creado un Animator Controller el cual estará alojado en la misma ruta. En cuanto al movimiento Mike tendrá dos parámetros dentro del Animator:

- Velocidad: será un integer que indica si debe correr o estar quieto:
  - Si está en 0 está en reposo.
  - Si está en 1 está andando.

    - Si está en 2 está corriendo.
- Dirección:
    - Si está en 0 deberá hacer la animación de caminar/correr hacia delante
    - Si está en 1 deberá hacer la animación de caminar/correr hacia la izquierda si está moviéndose o simplemente se girará si está quieto
    - Si está en 2 deberá hacer la animación de caminar/correr hacia la derecha moviéndose o simplemente se girará si está quieto
    - Si está en 3 hará la animación de darse la vuelta y su estado cambiará automáticamente a 0.

(De momento el Animator no se va a mostrar ya que todavía no está en su estado final)

A la hora de construir el Animator hay que tener en cuenta que las animaciones deben tener un avatar (el cuál extraeré del modelo de la animación Idle) y habrá que decidir si estas animaciones afectan o no a su desplazamiento real. Esto último ser hará modificando los parámetros de root motion en la animación. Ejemplo con la animación de correr:




Como podemos ver esta configuración hace que todo el movimiento se haga mediante desplazamientos configurados en el código y no con la animación, pero sí que tendrá en cuenta la posición del avatar para comenzar la animación.

En cuanto a las animaciones queda hacer que, por código, se compruebe que se están pulsando las teclas correspondientes y se active la configuración de parámetros del Animator que lleve al estado correspondiente.

Este código estará dentro del script MikeController localizado en la siguiente ruta:

Assets\Personajes\Mike\Scripts

Podemos ver a través del hilo de Twitter cómo ha ido desarrollándose el proceso de las animaciones:

- Programación de animaciones básicas: https://twitter.com/Fausto_1802/status/1450809335152529409
- Movimiento completo: https://twitter.com/Fausto_1802/status/1453102601491320832

Finalmente queda hacer que Mike pueda moverse realmente. Para ello, dentro del código que controla la pulsación de teclas se han añadido vectores de posición que se editan usando el Time.deltaTime para el desplazamiento en los ejes y quaternions para la rotación. Primero fijamos la posición y el ángulo de giro que tiene Mike:

```
Vector3 position;
Quaternion quaternion;
float giro = 0;
position = new Vector3(0, 0, 0);
quaternion = transform.rotation;
```

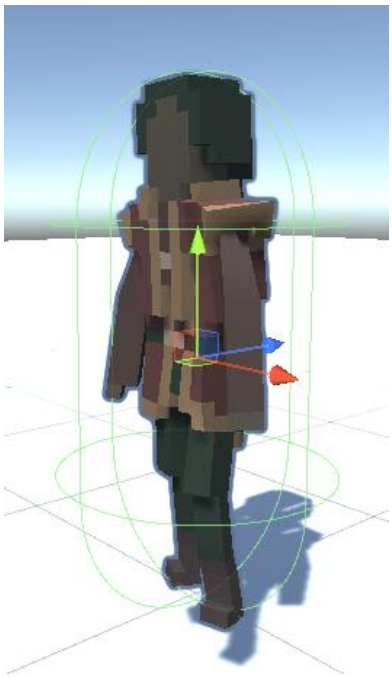
Lo modificamos (ejemplo de correr hacia la derecha):

```
position.z = 4 * Time.deltaTime;
animator.SetInteger("Velocidad", 2);
if (Input.GetKey(KeyCode.D))
{
    giro = (180) * Time.deltaTime;
    animator.SetInteger("Direccion", 2);
}
```

Y finalmente actualizamos su posición:

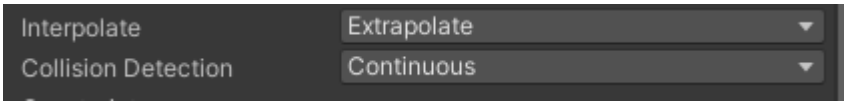
```
transform.position += quaternion * position;
quaternion *= Quaternion.Euler(0, giro, 0);
transform.rotation = quaternion;
```

Para poder ver esto dentro del juego he creado un plano con un cubo que he estirado y le he asignado un Collider al igual que a Mike. En el caso de Mike será un Capusle Collider.



Además, para que experimente gravedad le he añadido un RigidBody al que, para evitar que se caiga hacia los lados debido a posibles colisiones con defectos del terreno o por culpa de las animaciones he activado Freeze Rotation dentro de este:


Al igual que he activado la opción de extrapolate para evitar ciertos conflictos de las animaciones con el entorno y la detección de colisión continua para hacer que las colisiones con Mike sean perfectas ya que puede haber algunas que sean demasiado rápidas:



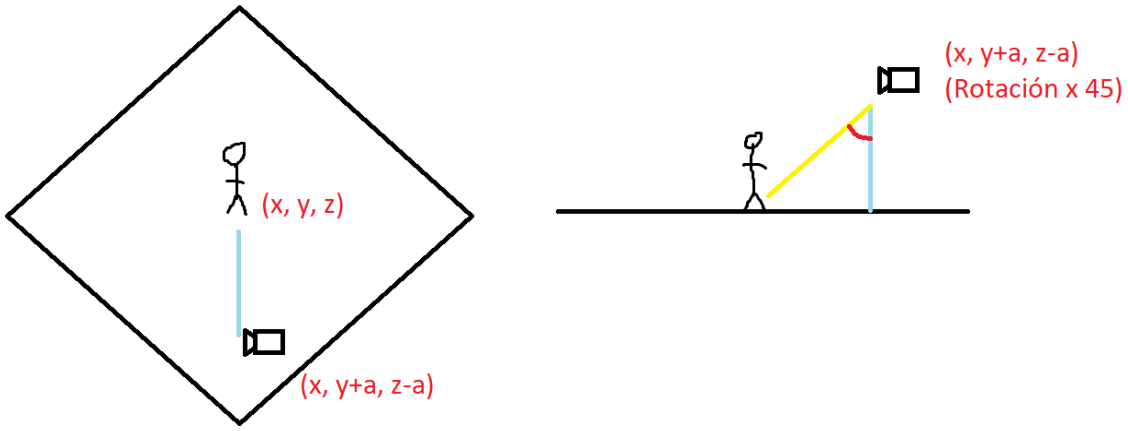

Podemos ver el desarrollo de esta parte en el siguiente tweet:

https://twitter.com/Fausto_1802/status/1453789704076341248

## 3.2 La cámara

En cuanto a la cámara he decidido que esta debe seguir a Mike en todo momento manteniéndolo siempre en el centro ya que si se acerca demasiado a alguno de los bordes puede estar en desventaja frente a enemigos que puedan aparecer por ese lado. Por ello, en vez de utilizar la técnica que hemos dado en clase de un rectángulo en el que si Mike se sale la cámara le enfoca, las

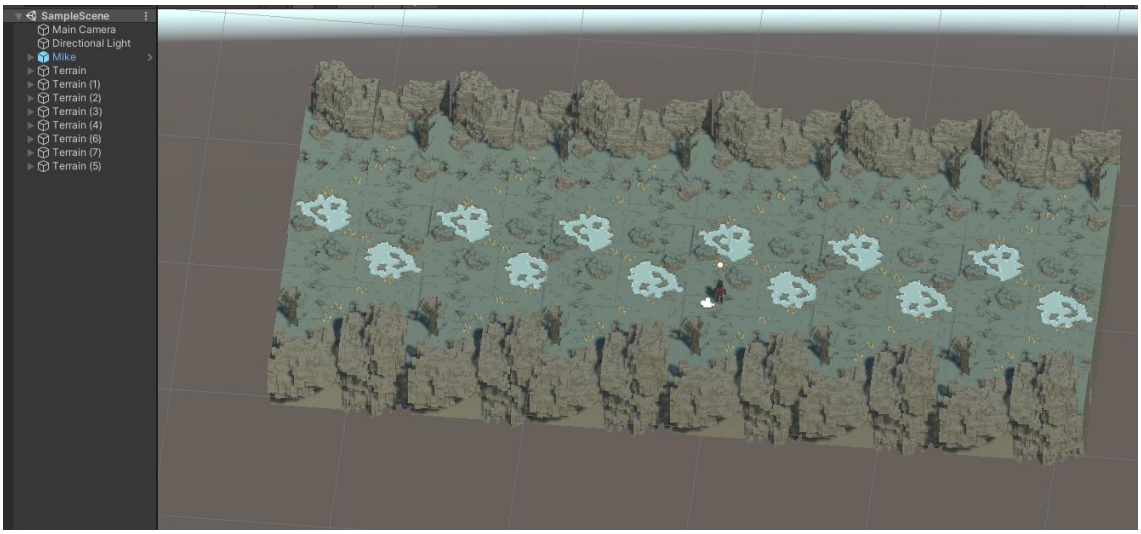
coordenadas de la cámara van a estar siempre en función de las de Mike y enfocándole en una especie de vista isométrica:




Para lograr esto he creado un script llamado CameraController el cual estará en la siguiente ruta:

Assets\GeneralScripts

A la hora de probar este script he creado un terreno de prueba con los assets descargados a los que les he asignado colliders:



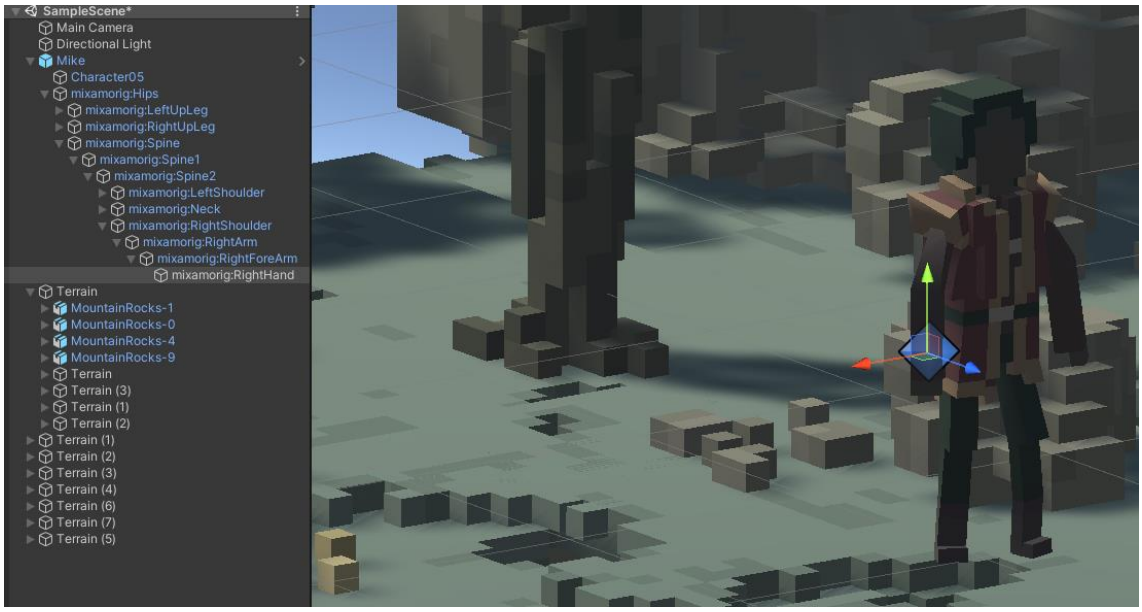

Podemos ver el funcionamiento de esta parte en el siguiente tweet:

https://twitter.com/Fausto_1802/status/1454053410844577793

## 3.3 LAS ARMAS

Las armas serán una parte importante del juego. Mike tendrá dos opciones, una espada y escudo o un hacha grande.

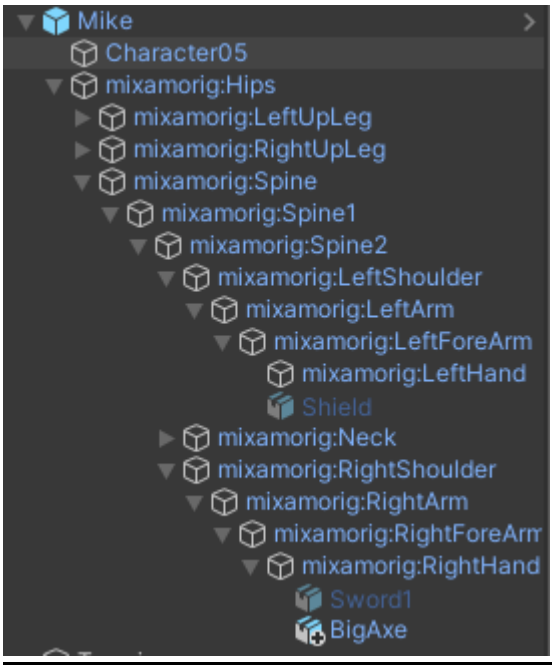
Empezando por la espada, esta estará equipada en la mano derecha de Mike. Para localizar su posición primero localizaremos la mano en su esqueleto:




A continuación, he añadido la espada a su mano y la he colocado como he pensado que quedaba mejor. He hecho lo mismo con el escudo y con el hacha:

Para hacerlos desaparecer y aparecer he pensado en activarlos o desactivarlos a medida que los tenga equipados o desequipados, pero siempre estarán

presentes en el juego. Por ejemplo, cuando tenga equipado el hacha estarán desactivados la espada y el escudo.




Para saber si Mike tiene disponible o no alguna de las armas utilizaré 3 Player Pref.

(Nota a posteriori: como en principio este videojuego tenía la idea de hacer varios mapas con una continuidad en la historia se usaron los Player Prefs para guardar el estado del inventario entre escenas. Esta idea finalmente fue descartada pero ya estaba implementado por lo que no se tocó. El funcionamiento podría ser igual usando variables normales.)

El primero se llamará Espada y será un entero en el que 1 significará que en algún momento Mike ha desbloqueado la espada y el escudo mientras que 0 es que todavía no los ha encontrado. El segundo será “Hacha” y seguirá la misma forma que el anterior. Finalmente, el tercero será “Inventario” un entero el cual si vale 0 Mike no llevará equipado nada, 1 será la espada y el escudo y 2 el hacha. Esta forma me ayudará a programar mejor el inventario ya que podré hacer que se pueda seleccionar el objeto simplemente usando los botones numéricos encima de QWERTY.

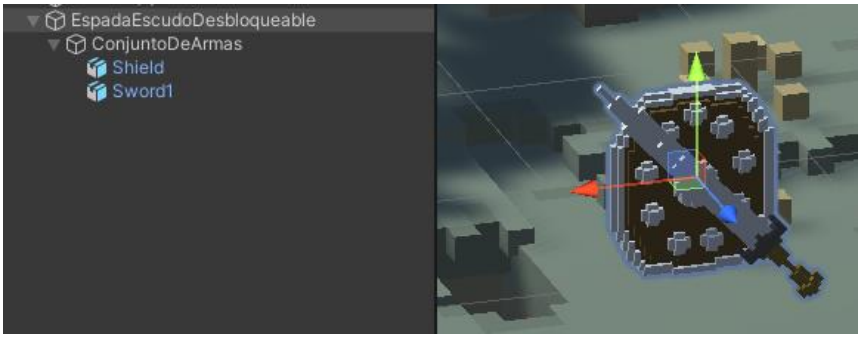
Para poder programar esto mejor he hecho públicas unas variables GameObject dentro del script Mike Controller a las que he asociado las armas:

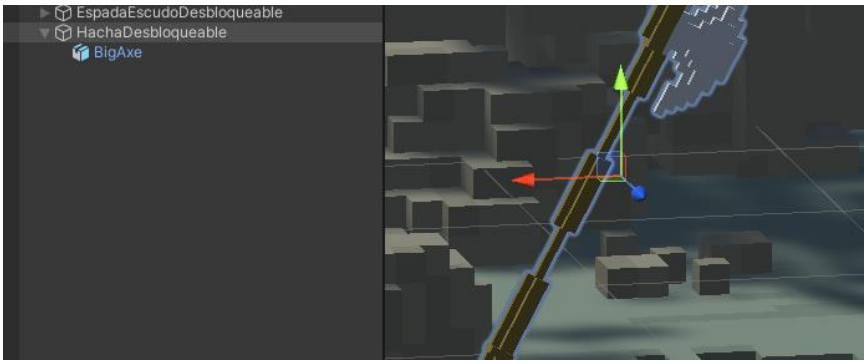

A continuación, he hecho que al iniciarse el personaje compruebe si los Player Pref están creados, en caso contrario los creará y se comprobará el estado del inventario. A parte, en el Update se comprobará si ha habido cambios, es decir, que si se han desbloqueado algún arma o se han pulsado los botones 1 o 2 para seleccionar el inventario.

Tweet relacionado con este avance:

https://twitter.com/Fausto_1802/status/1455969524751249417

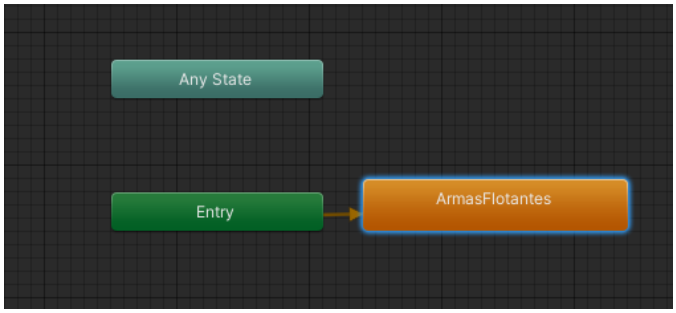
Para desbloquear alguna de las armas Mike simplemente deberá recogerlas del mapa. Al colisionar con ellas se borrarán del mapa y "pasarán a su inventario". Para poder realizar esto he creado un prefab de cada conjunto de armas que estará flotando en el mapa.

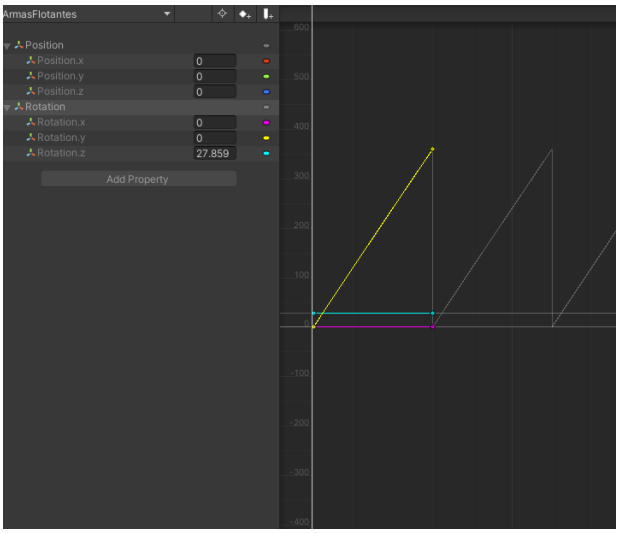



A continuación, para hacer el efecto de que estén flotando he creado una animación tanto para el hacha como para el escudo y la espada en la que moveré

su posición en el eje Y, y su rotación para dar ese efecto. Esta animación la aplicaré al hacha dentro de HachaDesbloqueable y a ConjuntoArmas dentro de EspadaEscudoDesbloqueable para así no afectar a su posición global en el mapa.




Estas animaciones y su respectivo Animator Controller estarán alojadas en la siguiente ruta:

Assets\Weapons\Weapons\Animaciones

Seguidamente, para poder detectar que Mike se ha chocado con las armas, he añadido un collider el cual será un trigger para así no afectar a las físicas. Cuando este trigger se active, las armas, modificarán los PlayerPref correspondientes y se desbloquearán las armas en el inventario de Mike.

Par realizar esto he creado un script llamado UnlockablesController alojado en la siguiente ruta:

Assets\Weapons\Weapons\Scripts

En el código he hecho que detecte si, primeramente, el objeto con el que colisiona es el jugador. Seguidamente comprueba si el objeto de ese trigger es o bien el hacha o el escudo y la espada y en función de uno u otro modifica el PlayerPref. Finalmente, el objeto se destruye.

Podemos ver la implementación de esta parte en el siguiente tweet:

https://twitter.com/Fausto_1802/status/1457690584374726658

## 3.4 ANIMACIONES DE COMBATE

Habiendo creado ya el sistema por el cual puedo desbloquear armas ahora toca hacer que pueda atacar con ellas. Para ello primeramente he definido qué habilidades y qué sistema de combate debe tener Mike. El sistema de combate estará basado en estamina que Mike gastará para poder realizar los diferentes ataques o habilidades. El gasto de estamina variará en función del ataque o habilidad que sea y se podrán activar con las flechas del teclado:

- Hacha:
    - Flecha arriba: Mike pegará un grito al cielo invocando el poder del hacha y se curará una porción de su vida.
    - Flecha izquierda: Mike pegará un hachazo de derecha a izquierda.
    - Flecha derecha: barrido 360 grados con el hacha.
    - Flecha abajo: es el ataque más fuerte de Mike. Correrá hacia delante, pegará un salto en el aire y caerá clavando su hacha.
- Espada y escudo:
    - Flecha arriba: Mike se protegerá con el escudo.
    - Flecha izquierda: Mike anda hacia delante atacando con la espada.
    - Flecha derecha: Mike comienza a atacar 360 grados andando hacia delante.
    - Flecha abajo: es el ataque más fuerte de Mike con la espada. Invoca al poder de esta y la imbuye con él. A partir de ese momento el daño de la espada pasa a multiplicarse x2 durante los próximos 5 ataques.

Habiendo seleccionado las habilidades, he descargado las animaciones de Mixamo, las he importado al proyecto y las he añadido los avatares además de configurar el root motion.

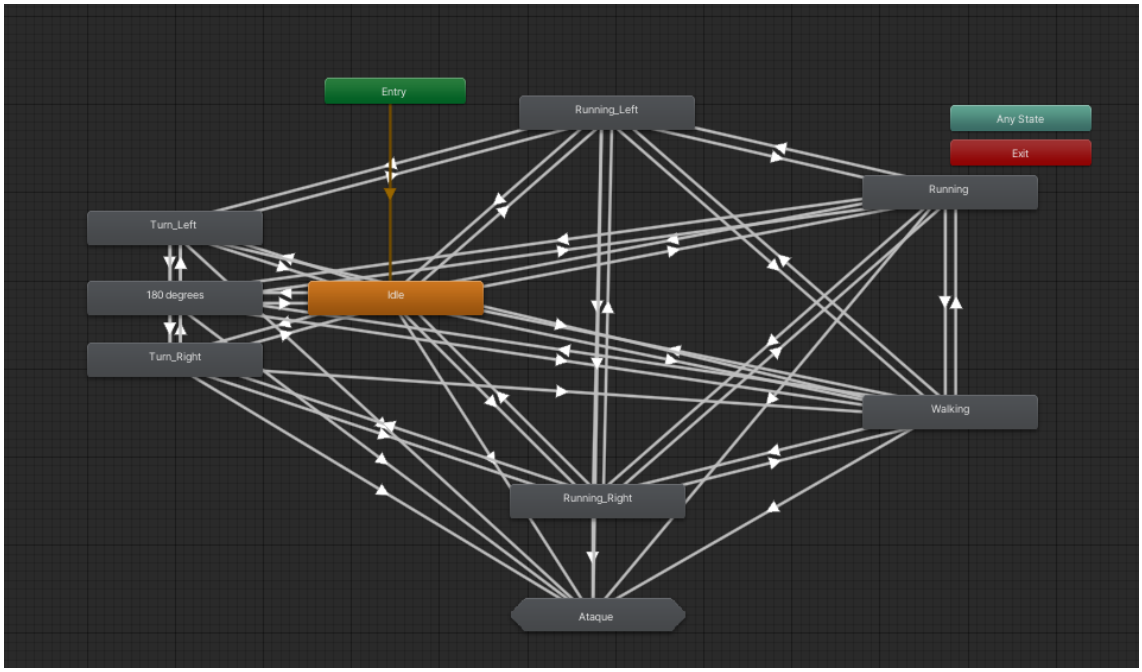
A continuación, he añadido las diferentes animaciones al animator. Estas se verán activadas en función de un integer llamado “Ataque” en el que cada valor significará:

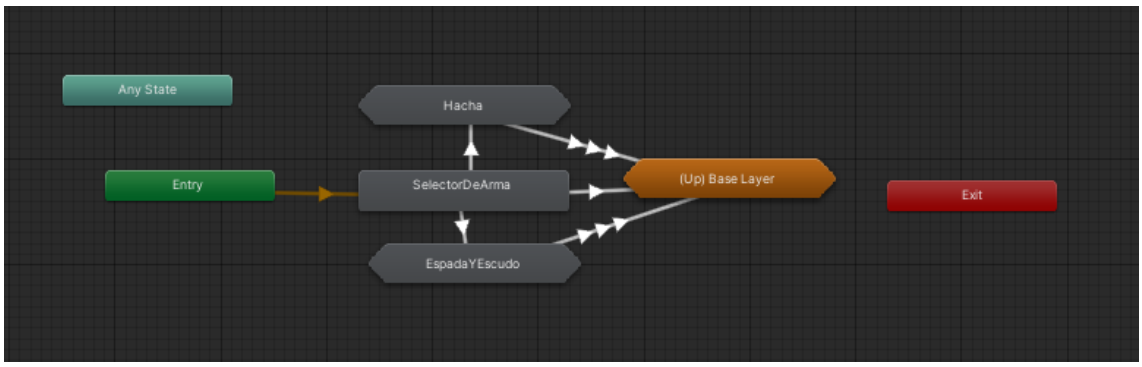
- -1: no está atacando.
- 0: ataque flecha arriba
- 1: ataque flecha abajo
- 2: ataque flecha izquierda
- 3: ataque flecha derecha

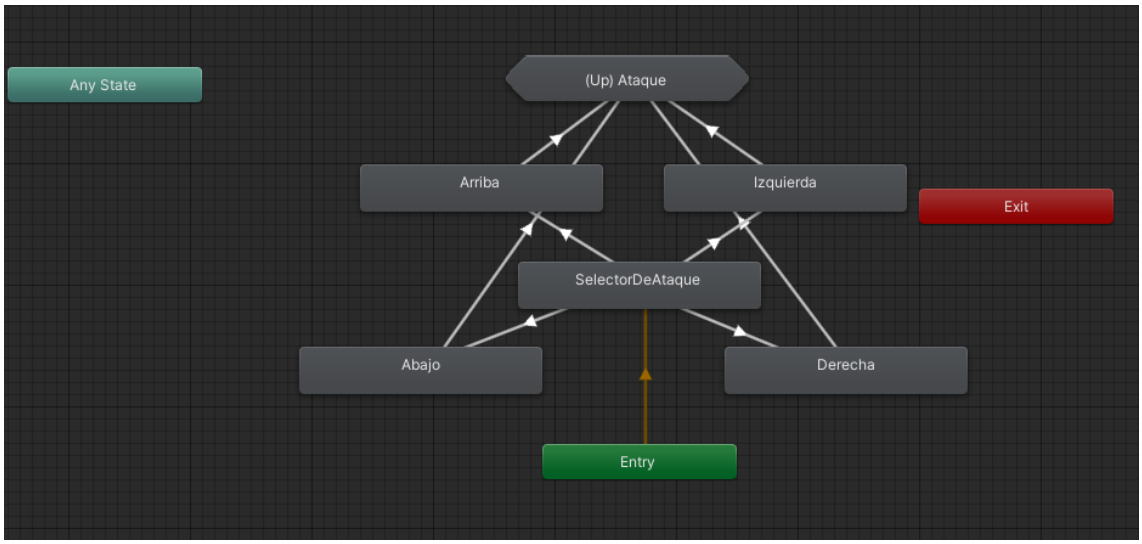
Se podrá acceder a estas animaciones en función de otro parámetro integer llamado “Arma”:

- 0: desarmado
- 1: espada y escudo
- 2: hacha

Si se detecta que Mike está armado y a parte está intentando realizar algún ataque entonces se pasará a una submáquina de estados que comprobará el arma que tiene y seguidamente realizará el ataque. Finalmente podemos ver cómo queda el Animator Controller de Mike:






Al realizar esto me doy cuenta de un pequeño fallo. Hay veces que, si pulso muy rápido una tecla de ataque, la animación no llega lo suficientemente rápido al estado de la animación del ataque y se queda atascada en el selector de ataque. Lo mismo puede suceder para el selector de arma. Para el primero la solución es sencilla, crear una transición que redirija de vuelta al Animator de movimiento si se detecta que no hay arma equipada. Para el segundo la solución se ha basado en incluir un script en las animaciones que vuelva a colocar el parámetro Ataque a -1 cuando la comprobación del parámetro al cambiar de estado ya se haya hecho, pero no esté a punto de salir de esta:

```
// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
Mensaje de Unity | 0 referencias
override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{
    animator.SetInteger("Ataque", -1);
}

// OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
Mensaje de Unity | 0 referencias
override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{
    animator.SetInteger("Ataque", -1);
}
```

Al ejecutar esto me he dado cuenta de otro error: cuando se está realizando la animación de ataque, Mike puede moverse hacia delante. Esto es debido a que no hay nada que bloquee la lectura de las teclas de movimiento mientras se realiza la animación. Para solucionarlo he decidido comprobar que el animator no se encuentre en ninguna de las 4 animaciones de ataque antes de leer el input de las teclas de movimiento. Para conseguirlo he etiquetado todas las animaciones de ataque con el Tag “Atacando” y justo antes de comprobar si se están pulsando las teclas de movimiento, compruebo si el estado del animator tiene ese tag.

```
if(!animator.GetCurrentAnimatorStateInfo(0).IsTag("Atacando"))
```

Finalmente podemos ver cómo ha quedado esto en el siguiente tweet:

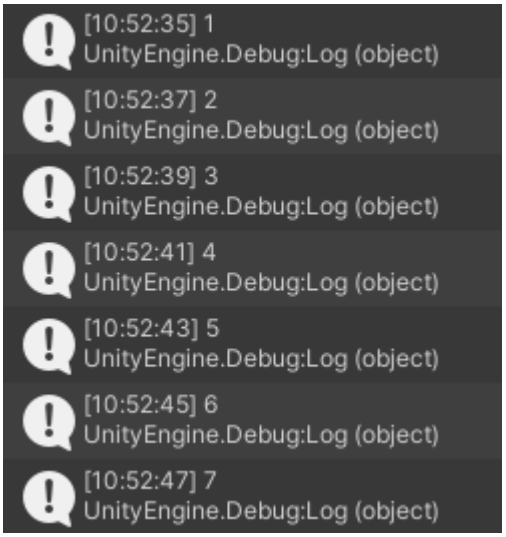
https://twitter.com/Fausto_1802/status/1458541357954383886

## 3.5 ESTAMINA

Para evitar que el combate sea muy aburrido y se estén haciendo los mismos ataques todo el rato, cada ataque costará ciertos puntos de estamina en función de su fuerza. Para controlar esta estamina, en cada escenario, Mike contará con 10 puntos de estamina al principio que podrá ir gastando y rellenando con el tiempo, ya que aproximadamente cada 3 segundos, se recuperará 1 punto de estamina.

El controlador de la estamina estará dentro del Mike Controller.

La estamina será representada por un integer que podrá ir de 0 a 10. Para hacer que este valor aumente cada dos segundos voy a utilizar el siguiente método[6]: en el Update, aumentaré el valor de la estamina en un valor que llamaré "cambioPorSegundoEstamina" el cual será 0.35 y lo multiplicaré por time.DeltaTime para que esto ocurra en un tiempo fijo. Como no se puede aumentar un integer con decimales entonces aumentaré una variable local de estamina la cual llamaré contadorEstamina que será un float. Cuando cotnadorEstamina sea mayor que 1 entonces aumentaré en una unidad la estamina comprobando antes que no vaya a superar 10.




[6] {How to GRADUALLY CHANGE a value in Unity} - Increase or decrease a variable with a rate of change. (2020, December 8). Youtube. https://www.youtube.com/watch?v=Wj_6BUIppAw

A continuación, simplemente tendré que comprobar que cuando se pulse una tecla de ataque, si se está seleccionando un arma y se tiene la suficiente estamina para lanzar dicha habilidad y si se tiene, se reste. Los costes de estamina serán:

- Flecha arriba: se trata de habilidades de protección. Se deben poder invocar frecuentemente pero no se puede abusar de ellas. Costarán 3 puntos.
- Flecha izquierda: es el ataque más débil y por tanto el que más se va a poder utilizar. Su coste será de 2.
- Flecha derecha: es un ataque más fuerte así que debe poder utilizarse menos. Su coste será 3.
- Flecha abajo: es la definitiva de Mike. Debe poder utilizarse muy poco ya que infringirá mucho daño. Su coste será 7.

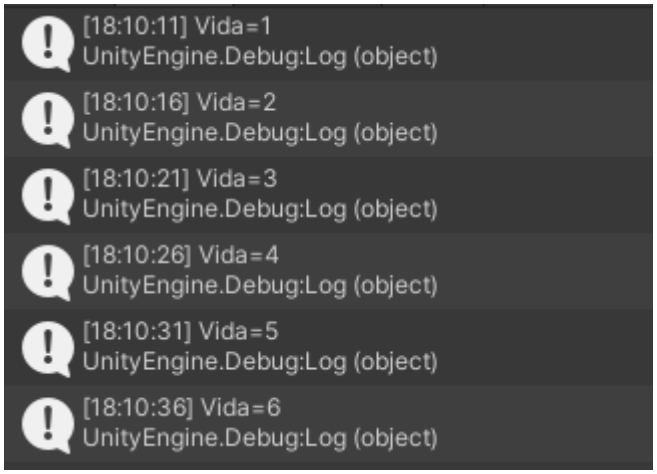
Además, he añadido otro contador llamado enfriamientoHabilidades el cual se asegurará que en el mismo segundo no se intentan lanzar dos habilidades. De esta forma se evita que si se pulsa muchas veces la misma tecla de habilidad se gaste más estamina de la que se debería.

## 3.6 LA VIDA

El sistema de vida de Mike será igual que el de estamina. También tendrá 10 puntos de vida, pero estos tardarán más en recuperarse. Cada 5 segundos se recuperará 1 punto de vida.

Para poder aplicar esto simplemente voy a seguir el mismo proceso que con la estamina.

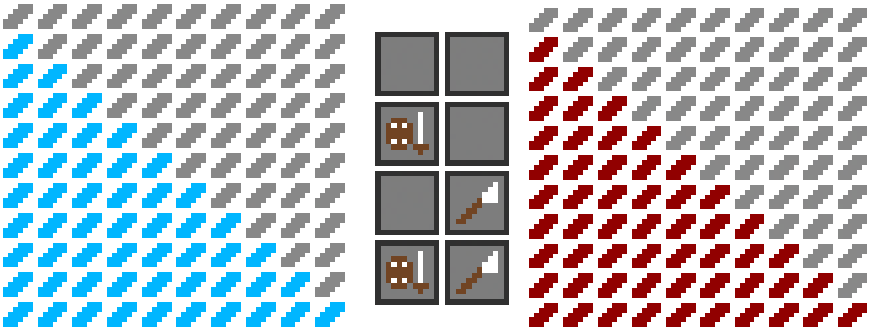
Como podemos ver, el proceso funciona bien y la vida aumenta 1 punto cada 5 segundos:



## 3.7 HUID

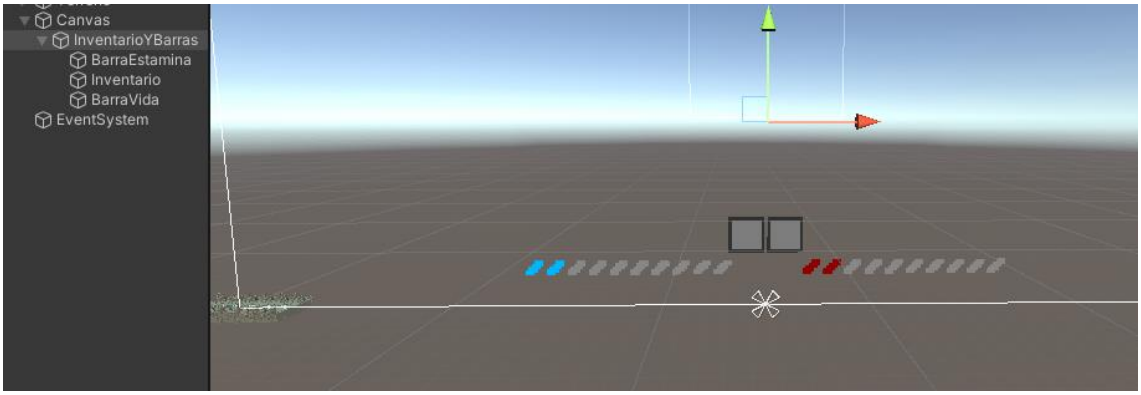
El próximo paso que voy a realizar será crear un HUID básico para poder mostrar los atributos de estamina, vida e inventario ya que ahora mismo los estoy representando por consola y para continuar trabajando será más cómodo tener una representación gráfica.

Primero he creado una serie de Sprites que representarán todos los posibles estados de la estamina, la vida y el inventario:



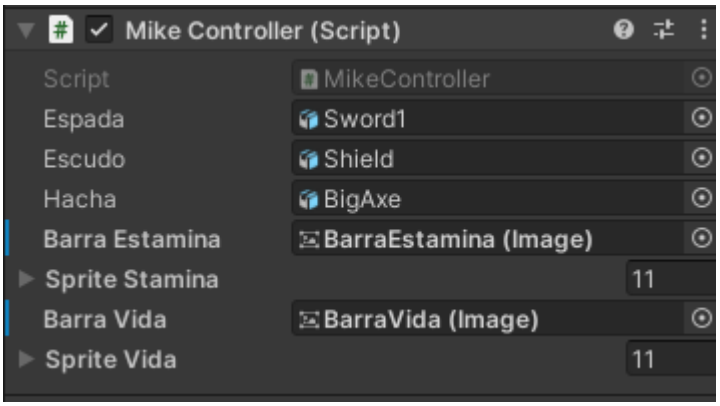
Seguidamente los he importado a Unity, los he catalogado como Sprite y los he recortado:

A continuación, los he colocado dentro de un objeto padre llamado “InventarioYBarras” en el canvas, el cual sus anchors los he anclado abajo mientras que los de las barras y el inventario los he anclado al centro. De esta forma no se moverán cuando se cambie el tamaño de la pantalla:




Seguidamente he hecho que en el script de Mike se le pueda pasar la imagen que contiene a la barra de estamina y la de la vida y un array con los diferentes

sprites y a continuación, he hecho que en función del indicador de estamina o el de vida, se cambie de Sprite.




```
barraEstamina.sprite = spriteStamina[estamina];
if(vida>=0&&vida<11)
    barraVida.sprite = spriteVida[vida];
```

Para el inventario he seguido un proceso parecido, pero en vez de programarlo en el MikeController, lo he hecho en el script que controla los objetos desbloqueables.

El HUID ha pasado por varias fases como podemos observar:

https://twitter.com/Fausto_1802/status/1459151840835248147

https://twitter.com/Fausto_1802/status/1459583132731064322

Pero al final ha quedado de la siguiente forma:

## 3.8 HACIENDO DAÑO CON LAS ARMAS

Ahora toca hacer que las armas interaccionen con el resto de los enemigos. Para ello he colocado primero en el mapa un modelo de enemigo con un Capsule Collider etiquetado como “Enemigo” para poder pegarle.


Seguidamente he creado un script que controle al enemigo zombie al cual he llamado EnemyController el cual estará almacenado en la siguiente ruta:

Assets\Personajes\Zombie1\Scripts

En este script ahora lo que voy a hacer es controlar su vida con una variable pública que pueda modificar desde otro objeto.

A continuación, le he añadido unos colliders a las armas y los he marcado como trigger para que así no afecten a las animaciones y seguidamente he hecho que si se detecta una colisión y Mike está atacando, se le quite la vida correspondiente al atacado. Para ello he creado un script llamado WeaponsController localizado en la siguiente ruta:

Assets\Weapons\Weapons\Scripts

Lo he añadido a las armas en las que, cuando hay una colisión, comprueba primero que se trata de un enemigo y después de que Mike está atacando para finalmente quitarle la vida correspondiente al enemigo. Esta vida estará controlada desde un script que se les añadirá a los enemigos llamado EnemyController localizado en la siguiente ruta:

Assets\Personajes\Zombie1\Scripts

## 3.9 MOVIMIENTO DEL ENEMIGO

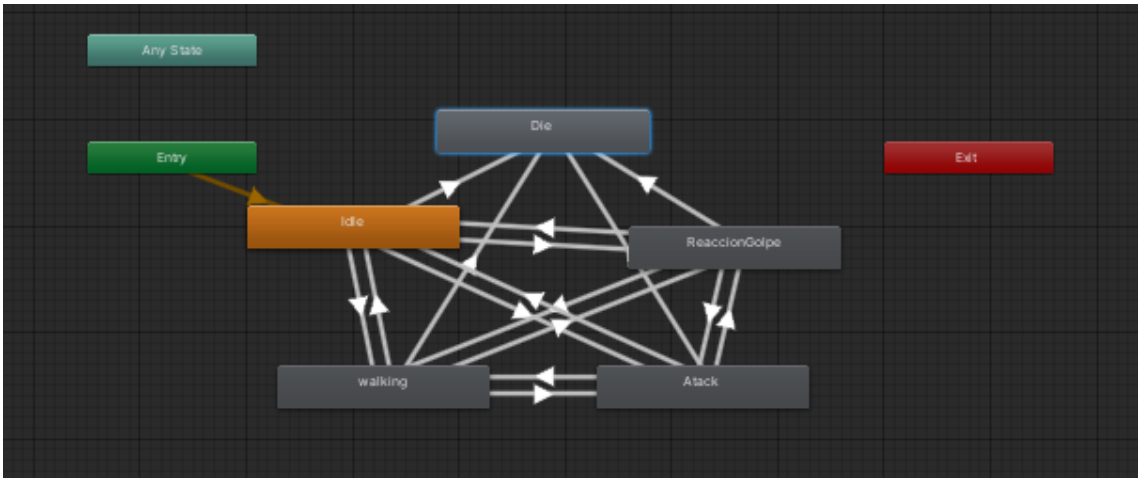
En el juego sólo estará un enemigo: el zombie y este podrá realizar las siguientes acciones: estar quieto, caminar, atacar y morir. Me he bajado sus correspondientes animaciones y he creado un Animator Controller los cuales he alojado en la siguiente ruta:

Assets\Personajes\Zombie1\Animations

El Animator Controller del zombie estará controlado por una serie de parámetros:

- Vida: integer que equivale a la vida del zombie

- Golpe: bool que se activa cuando recibe un golpe
- Andar: bool que está activo cuando debe estar andando
- Ataque: bool que se activa cuando debe estar atacando



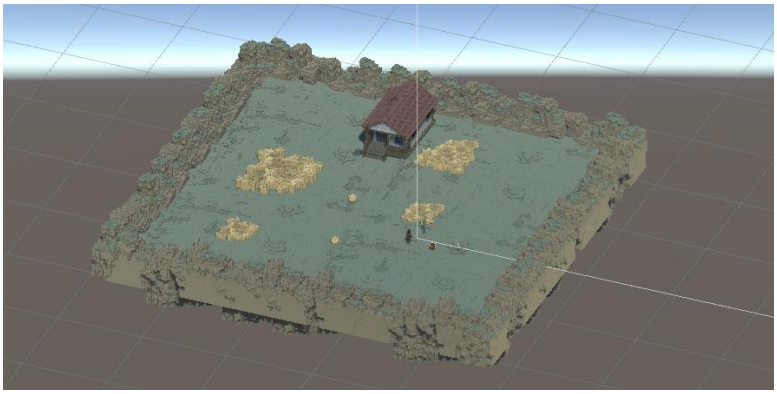

Estos parámetros serán controlados dentro del EnemyController.

Podemos ver una primera aproximación a interacción con los enemigos dentro de este tweet:

https://twitter.com/Fausto_1802/status/1462526675946819587

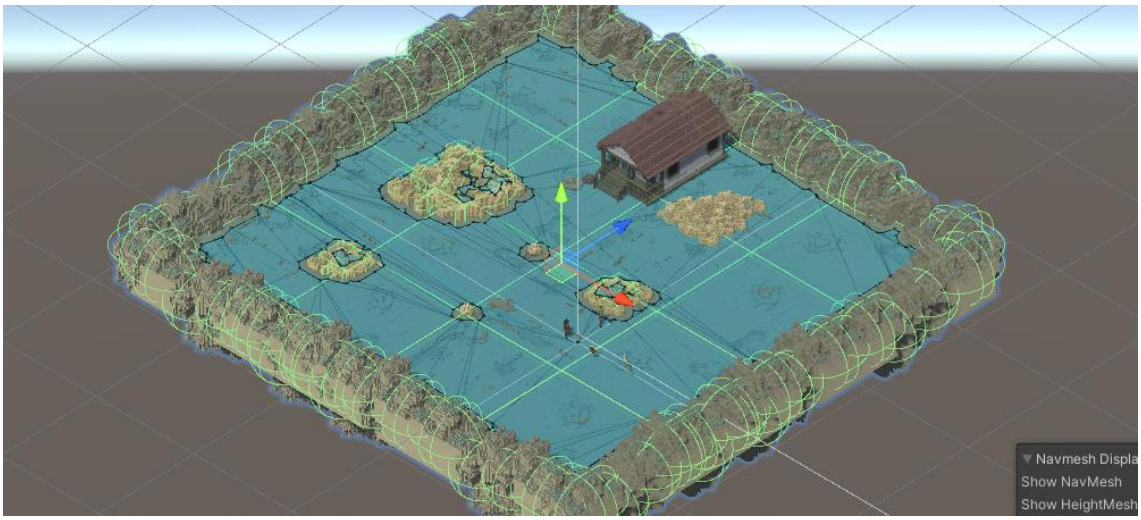
## 3.10 MAPEO DEL ENTORNO

Para poder comenzar haciendo que los enemigos sigan automáticamente a Mike primero he creado uno de los mapas finales. El del campo. Este mapa estará guardado en el GameObject TerrenoCampo dentro de la escena.



Seguidamente hay que hacer que el terreno sea navegable, para ello he añadido un mesh renderer a TerrenoCampo y luego en la pestaña de navigation mesh, he hecho que TerrenoCampo y todos sus hijos sean static.

Y a continuación he creado el navMesh pulsando en la tecla Bake:




Seguidamente he añadido un navMesh agent al enemigo y he hecho que, mediante código en el EnemyController, cuando se detecte que se ha llegado a la posición de Mike, se cambie la animación a la de ataque.

Podemos ver la implementación de esto en el siguiente Tweet:

https://twitter.com/Fausto_1802/status/1462836186138001415

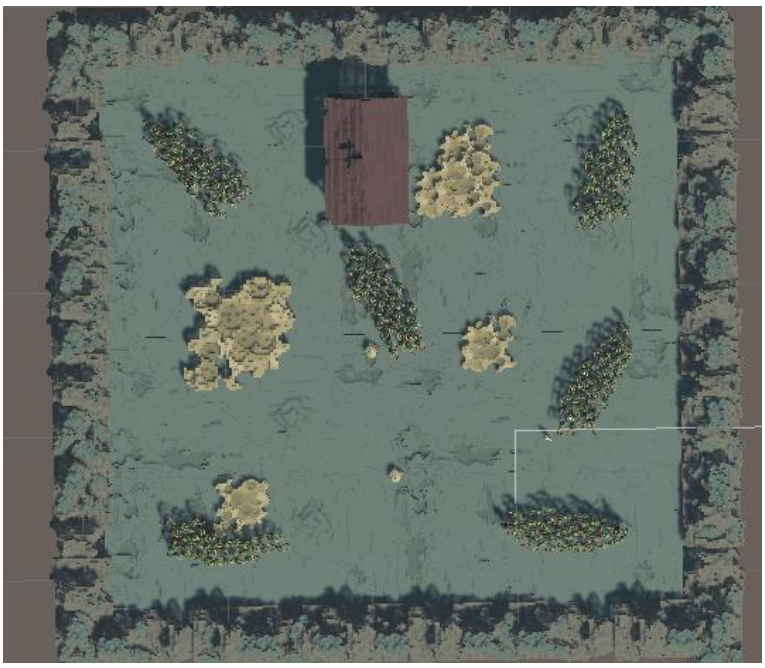
## 3.11 Enemigos atacando

Los enemigos también deben atacar en el juego. He añadido un GameObject vacío a sus manos y he hecho que este contenga un collider en modo trigger. Si se detecta que ha colisionado con Mike se le restará vida a este. Para ello he creado un script llamado EnemyAtack el cual estará en la siguiente ruta:

Assets\Personajes\Zombie1\Scripts

Ahora se presenta el requerimiento de que los enemigos deben ir spawneando en el mapa para que Mike los vaya matando. Para ello crearé un objeto vacío en medio del mapa y dentro de él crearé varios elementos hijos. A continuación, he creado un Script llamado EnemySpawner localizado en la anterior ruta que hará que cada 9 segundos se elegirá un hijo aleatorio y se spawneará de esa posición un enemigo.


Podemos ver que los zombies empiezan a salir de esos puntos.



Ahora hay que controlar que a medida que se vayan matando más zombies comiencen a salir más deprisa para así aumentar la dificultad. Para ello en el script de spawner de zombies he hecho que cuando se llega a una determinada cantidad de zombies muertos se disminuya la duración de aparición 1 segundo hasta llegar a 1 segundo por lo que, en esencia, habrá 9 rondas.

Aquí me he dado cuenta de un error que se debía a que los zombies no pueden estar en el mismo lugar y por tanto no podían atacar a Mike todos a la vez. Para solucionarlo he puesto los diferentes elementos del juego en diferentes capas y he desactivado las interacciones de físicas de algunas capas en el menú de Project Settings.

## 3.12 ESCENA DE MUERTE

Cuando Mike muerte debe haber una escena de muerte. Par ello habrá un script llamado MuerteController localizado en la siguiente ruta:

Assets\Canvas\Scripts

Este script lo he asignado a un objeto vacío llamado LaParca que controlará que cuando la vida de Mike llegue a 0 se activen una serie de elementos en el canvas que representarán la pantalla de muerte y se parará el tiempo usando Time.timeScale. En cuanto al botón de regresar al menú, tendrá el script llamado equipado en cuanto a cambios de escena llamado EscenasController y alojado en la misma ruta anterior que contendrá funciones para los botones de cambio de escena.

Podemos ver la implementación de esto en el siguiente tweet:

https://twitter.com/Fausto_1802/status/1466455830803402758

## 3.13 Menú de pausa

Para el menú de pausa he creado otro objeto hijo que he agregado al canvas. Este objeto se volverá activo cuando un script que añadiré al canvas llamado PausaController y alojado en la misma ruta anterior, detecte que el jugador pulsa la tecla ESC. En ese momento el juego se parará con timeScale=0 y se podrá o bien reanudar volviendo a pulsar ESC o bien volver al menú principal.


## 3.14 Menú principal

Para crear el menú principal he creado una nueva escena llamada menú principal y en ella he creado un canvas al que le he añadido los elementos pertinentes. De fondo he colocado el mapa del campo con Mike y un zombie bailando con unas animaciones que me he descargado de Mixamo. Estas animaciones y sus animatorController estarán en:

Assets\Bailes

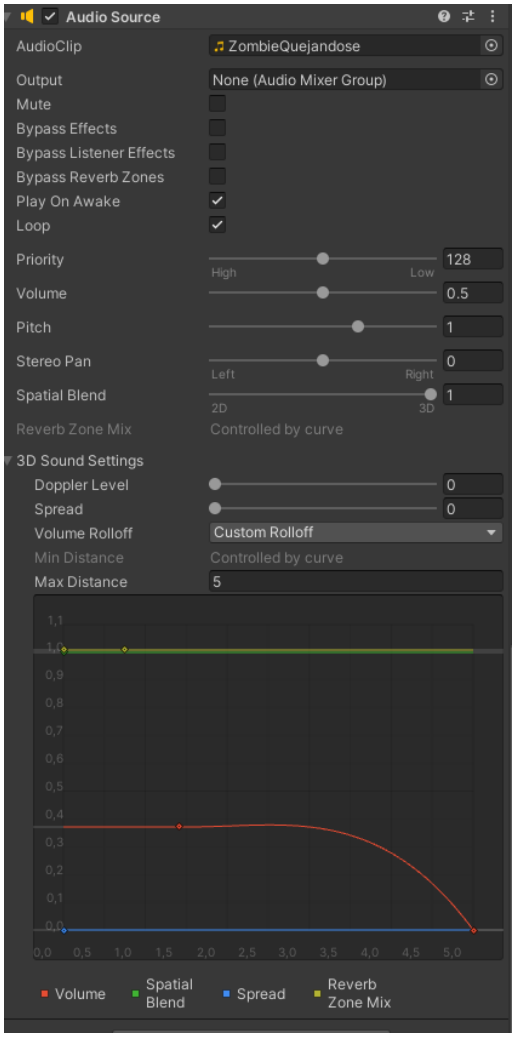
Si se quieren hacer cambios de escena todo está controlado por los botones a través del script EscenaController.

## 3.15 EFECTOS DE SONIDO

Dentro del juego habrá dos tipos de sonido diferentes:

- Música de fondo: la cual estará puesta de fondo gracias a un gameObject llamado altavoz el cual tendrá asociado un audioSource con una música en bucle.
- Efectos de sonido: Mike y los zombies tendrán asociado un audioSource el cual reproducirá ciertos clips de audio asociados con ataques, muertes, etc. Estos clips de audio serán controlados dentro de los controladores de tanto enemigos como Mike en el caso de sonidos como andar o los quejidos de los zombies y en el caso de los sonidos de ataques el audioSource estará en el arma o en el gameObject del puño y se controlará por el script de los controladores de ataques.

  Además, Mike llevará el audio listener y los sonidos de quejidos de los zombies tendrán activado el audio en 3d para así dar la sensación de que vienen a por ti.

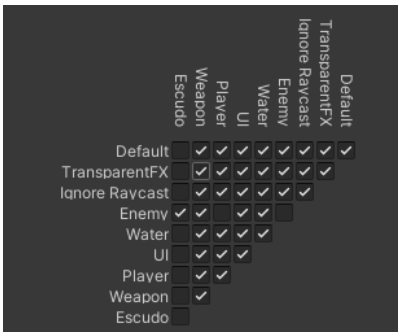


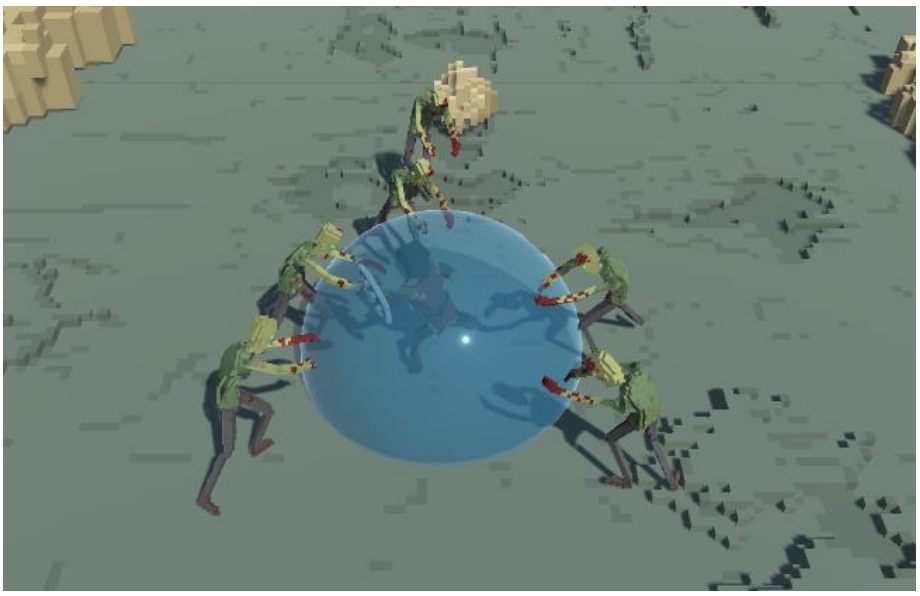
## 3.16 PARTÍCULAS Y EFECTOS

Para mejorar el impacto visual en el juego deberemos añadir unas partículas para que se vea mejor. Sobre todo, cuando por ejemplo se ha lanzado una habilidad como la de curarse o la de aumentar el daño de los ataques.

Antes de empezar con las partículas voy a hacer el efecto de que se abra una especie de escudo mágico cuando se active el escudo. Este será una esfera a la que le he añadido un material transparente y una animación de expansión cuando se activa. A continuación, la he posicionado dentro de Mike y cuando se lance la animación de escudo se activará.

Además, le he añadido un collider que sólo interactuará con los zombies para alejarlos. Finalmente, las interacciones entre los colliders quedan así:



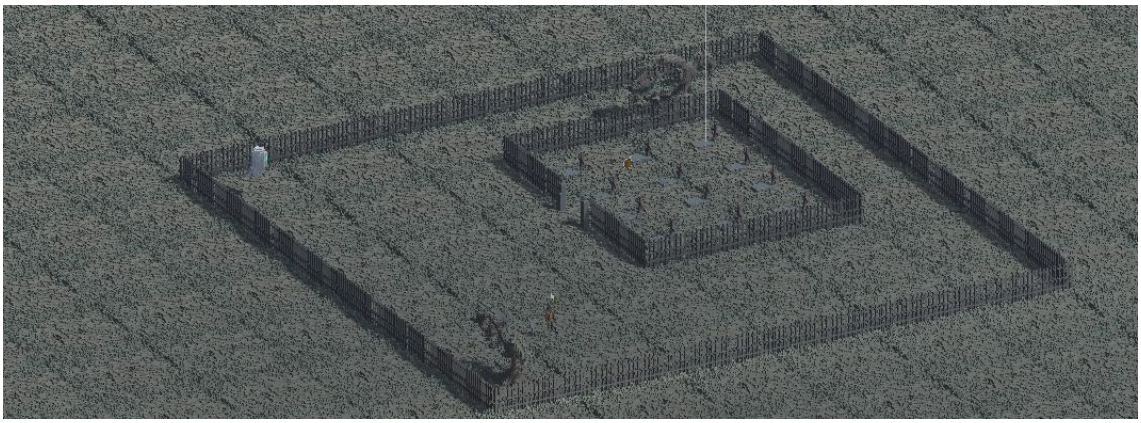
Para el efecto de la espada muy poderosa (habilidad flecha hacia abajo) voy a utilizar el asset de procedural fire que incluye partículas ya creadas.

A continuación, he añadido estas llamas como hijas de la espada y se activarán cuando el multiplicador de WeaponsController no esté a 1.

Finalmente le voy a añadir las partículas de curación. Estas partículas se activarán cuando se presione la tecla arriba con el hacha. El proceso será equivalente al de las llamas solo que aparecerán exclusivamente cuando se haga la animación.

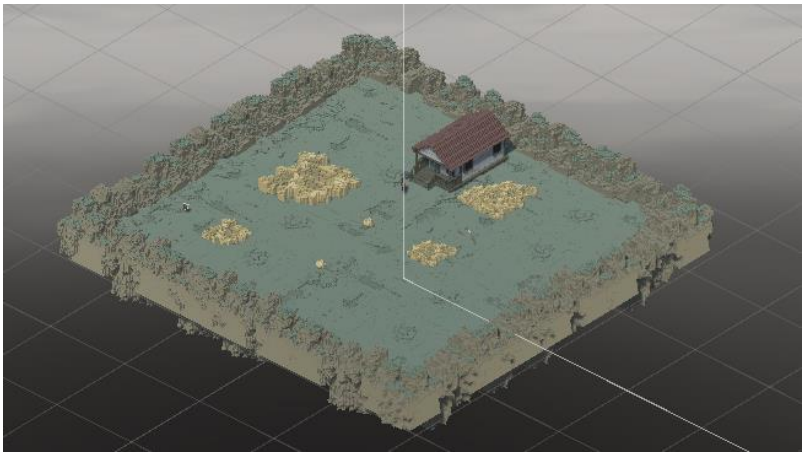
## 3.17 Último mapa

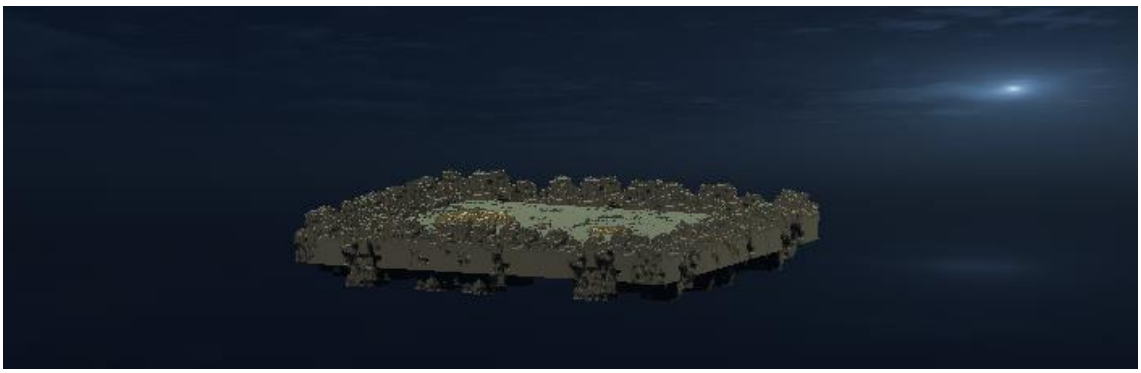
Finalmente he creado un segundo mapa, el del cementerio, siguiendo todos los procesos que hemos estado siguiendo durante esta práctica y reciclando algunos elementos del mapa anterior.



## 3.18 ÚLTIMOS DETALLES

Como detalle final he cambiado el cielo de cada mapa y el menú para que se vean mejor. Para ello he usado los assets importados.





# 4 FINAL Y AGRADECIMIENTOS

Este trabajo ha llevado bastantes horas de desarrollo y se puede ver cómo tanto la idea como el propio avance del juego han ido cambiando a lo largo del tiempo.

Para poder ver un tutorial de Dangerous Dungeons acuda a la siguiente dirección:

https://www.youtube.com/watch?v=-Z3azPjWJag

Especiales agradecimientos a:

David Montagut Pamo

Eli Moure López

Jordi Vidal Sola

Luis Martín Maíllo