Fausto Sánchez Hoya

Índice

[1 Introducción 2](#_Toc87264910)

[2 Diseño y estética del juego 3](#_Toc87264911)

[3 Desarrollo 5](#_Toc87264912)

[3.1 Importación de assets 5](#_Toc87264913)

[3.2 Pruebas de animaciones en Mike 5](#_Toc87264914)

[3.3 El movimiento de Mike 10](#_Toc87264915)

[3.3.1 Animaciones 10](#_Toc87264916)

[3.3.2 Movimiento real 14](#_Toc87264917)

[3.4 La cámara 17](#_Toc87264918)

[3.5 Terreno de prueba 18](#_Toc87264919)

[3.6 Las armas 19](#_Toc87264920)

# 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 toda una serie de enemigos que la han conquistado.

Este videojuego será de tipo “Diablo” en el que se deberán de liberar una serie de escenarios de enemigos para poder ir avanzando.

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

Lo iremos viendo…

# 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]](#footnote-1) excepto los relacionados con las armas que son de Unity-Fantom [[2]](#footnote-2).

 Imagen que contiene luz, tabla, pequeño, grupo

Descripción generada automáticamente

Imagen que contiene barco, iluminado, viendo, oscuro

Descripción generada automáticamente

Por otro lado, las animaciones de los personajes serán extraídas de Mixamo[[3]](#footnote-3).

# Desarrollo

## Importación de assets

Lo primero que he hecho ha sido importar todos los assets que me había descargado de varias páginas a mi proyecto de Unity. Para ello he creado una carpeta con el nombre del proyecto y dentro de ella otra carpeta llamada assets.

Texto

Descripción generada automáticamente con confianza baja

## Pruebas de animaciones en Mike

Para empezar con el proyecto quería probar las animaciones del personaje principal por lo que el primer paso es elegir a nuestro personaje de “Mike el Todopoderoso”. Al final me he decantado por este modelo (TPose\_Character05):

Imagen que contiene lego

Descripción generada automáticamente

He creado una carpeta dentro de personajes llamada Mike que contendrá todos sus recursos:

Texto

Descripción generada automáticamente con confianza baja

A continuación, he buscado en Mixamo animaciones de movimiento como correr y andar. Para ello he importado primero su modelo a la página web:

Interfaz de usuario gráfica

Descripción generada automáticamente

Seguidamente he descargado las animaciones de “Idle”, “Walking” y “Running”. Como en Mixamo te puede generar el prefab con el modelo y el esqueleto para directamente importarlo a escena, en la primera animación (Idle) he descargado estos recursos y por tanto serán utilizados como el modelo a importar a la escena de forma predefinida. Todos estos recursos estarán en la carpeta de “Animación” dentro de la carpeta de Mike:

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Como Unity tiene problemas para reconocer las texturas del modelo 3D, he exportado las texturas y las he guardado en una carpeta llamada Materials dentro de la carpeta Mike:

Texto

Descripción generada automáticamente

Nuestro personaje ya está listo para ser importado por lo que sólo nos hace falta arrastrarlo a escena, colocarlo y aumentar su tamaño para que sea distinguible:

Imagen que contiene agua, vuelo, colgando, esquiando

Descripción generada automáticamente

A continuación, he creado un animator controller llamado “MikeAnimator” el cual he alojado dentro de la carpeta “Animations”, he hecho que el primer estado al que pasa Mike nada más empezar es al de reposo y se lo he añadido al GameObjet Idle de la escena.

Interfaz de usuario gráfica

Descripción generada automáticamente

Como la animación tiene un fin he activado la opción Loop Time en todas las animaciones para que se queden reproduciéndose, he activado también la opción Loop Pose para que haya menos problemas en las transiciones y las he añadido al animator controller.

Interfaz de usuario gráfica

Descripción generada automáticamente

Para controlar el paso de una animación a otra he creado un parámetro entero que representará los tres estados de velocidad que puede llevar:

* Si está en 0 está en reposo.
* Si está en 1 está andando.
* Si está en 2 está corriendo.

Teniendo este parámetro en cuenta he creado las transiciones de un estado a otro a las cuales les he desactivado la opción de “Has exit time” para asegurar que en cuanto el personaje comience a correr su animación también lo haga.

Imagen que contiene interior, tabla, computadora, juego

Descripción generada automáticamente

Ahora que sus animaciones básicas ya funcionan vamos a proceder con su activación a través de los controles del juego. Para el desplazamiento he decidido que el jugador podrá usar las teclas W A S D para moverse por el mapa y el L-Shift para aumentar su velocidad por lo que deberemos crear un script que recoja la activación de esas teclas y cambie el estado del integer “Velocidad” de la animación. Este script y todos los relacionados con Mike estarán alojados en una carpeta llamada Scripts dentro de la carpeta Mike:

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

De momento sólo puede estar en tres estados a la vez por lo que su controlador de movimiento es muy básico todavía:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## El movimiento de Mike

### Animaciones

Es hora de hacer que Mike pueda hacer movimientos más complejos como girar (todavía no me voy a centrar en los ataques). Para ello vamos a definir primero los movimientos que va a poder hacer nuestro personaje.

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.



Para ello me voy a descargar una serie de animaciones de giros y de cambios de velocidad de movimiento.

Seguidamente he completado el animator de Mike con los estados nuevos. Para tener en cuenta los giros he creado otro parámetro integer el cual he llamado “Dirección” el cual tendrá los siguientes valores:

* 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.

Diagrama

Descripción generada automáticamente

En cuanto a su controlador de movimiento he hecho que pueda cambiar a todos sus estados pulsando las teclas W A S D y L-Shift haciendo que por defecto haga que esté quieto y de ahí a cualquier estado modificando su dirección y velocidad.

A continuación, me enfrento a un problema: las animaciones mueven al personaje ligeramente y si se vuelven a ejecutar reinician su posición haciendo que experimente un movimiento brusco. Para solucionarlo lo primero que hago es en la parte de Rig de nuestro modelo “Idle” seleccionamos crear un avatar a partir de este modelo y seleccionamos como nodo raíz las caderas:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Haciendo esto se nos crea nuestro avatar que podremos ver dentro del modelo “Idle”.

Interfaz de usuario gráfica

Descripción generada automáticamente

A continuación, he configurado el avatar de avatar de cada animación que me daba problemas como el avatar que hemos generado:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Después he configurado la opción del “Root Motion” para que en aquellas animaciones que daban problemas se configure correctamente haciendo que su último estado se tome como inicio de la siguiente. Ejemplo de la animación de correr mientras se gira:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

En este caso como quiero que la animación controle la rotación del personaje mientras gira, pero no su avance porque eso quiero controlarlo yo mediante código, activo la opción “Back Into Pose” para los ejes x y z.

He realizado esta acción en todas las animaciones que lo necesitaban y finalmente, para que esto se aplique, en el animator de Mike he adjuntado su avatar y he activado la opción de “Root Motion”.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

### Movimiento real

Ahora toca hacer que Mike se pueda mover por un plano real. Para ello lo primero que he hecho ha sido crear un suelo expandiendo un cubo debajo de Mike. Como el cubo ya tiene de por si un box collider no he tenido que importarle ninguno.

Imagen que contiene mapa, nieve, esquiando, cubierto

Descripción generada automáticamente

A continuación, he tenido que seleccionar un collider para Mike. En este caso he seleccionado un collider de cápsula ya que es el que mejor se adecua al personaje. Al hacerlo no acababa de estar lo suficientemente ajustado a como quería así que lo he acabado ajustando a mano.

Imagen que contiene lego, juguete, calle, agua

Descripción generada automáticamente

Lo siguiente que he hecho ha sido asignarle a Mike un Rigidbody para darle gravedad. Nada más añadirlo y darle al play he visto como de repente empezaba a flotar hacia arriba. Esto se debía a que anteriormente no había activado la opción de controlar el eje Y mediante código en las animaciones por lo que un pequeño cambio en la animación del eje Y, de forma acumulativa hacia que pareciese que flotaba. Seguidamente he visto como había una interferencia entre las animaciones y el Rigidbody por lo que he activado la opción “Extrapolate” para que estas no se vean afectadas.



Ahora me encuentro otro problema y es que algunas animaciones a las que no puedo bloquear su acceso a la rotación modifican la del eje X o el eje Z por lo que Mike se acaba cayendo. Para solucionar esto, en el Rigidbody he activado el bloqueo de ambos ejes ya que el único que quiero que se vea afectado es el eje Y.

Interfaz de usuario gráfica

Descripción generada automáticamente

Ahora es momento de hacer que Mike se mueva hacia delante cuando se pulsa la W y gire mientras anda. Para lograrlo he utilizado los quaternions que hemos dado en la asignatura.

Captura de pantalla de un celular

Descripción generada automáticamente

Además, mientras lo hacía he visto que las animaciones de girar mientras se corre o camina no estaban bien hechas del todo por lo que al final me he decantado por girar el personaje mediante código, pero manteniendo las animaciones excepto las de andar girando que estaban especialmente mal hechas y apenas se notaban; el resto están bien. Así que he bloqueado toda opción de que las animaciones afecten a la posición o rotación de Mike. Ejemplo de la animación de correr hacia la derecha.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Finalmente, he estado probando de nuevo las animaciones y arreglando bugs en cuanto el movimiento y finalmente Mike se mueve correctamente. Las animaciones con las que me quedo al final en cuanto a movimiento son:

Un dibujo con letras

Descripción generada automáticamente con confianza media

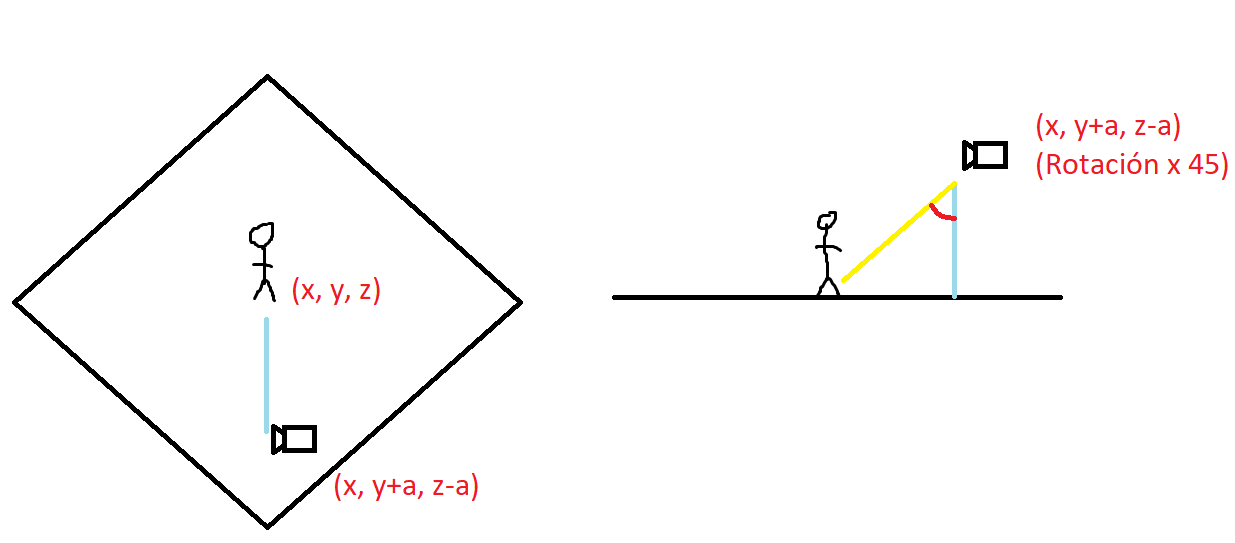
Y el animator controller queda tal que así:

Imagen que contiene interior, tabla, computadora, grande

Descripción generada automáticamente

## 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 el cual almacenaré en DangerousDangeons ->GeneralScripts y que se llamará CameraController:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Dentro de este he creado dos variables llamadas “mike” y “distancia” las cuales he hecho públicas para poder asignar sus valores a través de la interfaz cuando este scritp se añada a la cámara:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Finalmente el scritp está terminado, lo pruebo y funciona correctamente:

Gráfico

Descripción generada automáticamente con confianza baja

## Terreno de prueba

Al hacer las pruebas de la cámara me he dado cuenta que no se aprecia del todo bien el movimiento debido al suelo blanco, por ello he decidido crear un escenario de prueba usando los assets que ya me había descargado y asignándoles los colliders pertinentes para tanto suelo como obstáculos:

Imagen que contiene foto, pastel, tabla, grande

Descripción generada automáticamente

Este escenario está mal hecho y tiene muchos fallos pero me servirá de momento para poder apreciar mejor las pruebas que haga.

## 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:

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja

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 arco:

Imagen que contiene juguete, lego, tabla, luz

Descripción generada automáticamenteImagen que contiene juguete, lego, luz, tabla

Descripción generada automáticamente

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.

Texto

Descripción generada automáticamente

Para saber si Mike tiene disponible o no alguna de las armas utilizaré 3 Player Pref. 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 haré públicos unas variables GameObject a las que asociaré las armas:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

A continuación haré 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 han habido cambios.

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 podre realizar esto voy a crear un prefab de cada conjunto de armas que estará flotando en el mapa.

Interfaz de usuario gráfica

Descripción generada automáticamente

Una captura de pantalla de un videojuego

Descripción generada automáticamente con confianza baja

A continuación, para hacer el efecto de que estén flotando crearé 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.

Interfaz de usuario gráfica

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

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

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

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:

Texto

Descripción generada automáticamente con confianza media

Habiendo creado ya estos desbloqueables, los guardaré como prefabs en la carpeta weapons.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## 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 parta está intentando realizar algún ataque entonces se pasará a una submáquina de estados que comprobará el arma que tiene y seguidamente realizará el ataque:

Imagen que contiene interior, paraguas, tabla, grande

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

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 segundo 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:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

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.



## 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, o bien pociones de estamina o bien con el tiempo, ya que cada 2 segundos, se recuperará 1 punto de estamina.

La estamina será representada por un integer que podrá ir de 0 a 10:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Para hacer que este valor aumente cada dos segundos voy a utilizar el siguiente método[[4]](#footnote-4): en el update, aumentaré el valor de la estamina en un valor que llamaré “cambioPorSegundo” el cual será 0.5 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 estaminaVariable que será un float. Cuando estaminaVariable corresponda a un entero entonces se lo aumentaré a estamina comprobando antes que no vaya a superar 10.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteTexto

Descripción generada automáticamente

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.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

## La vida

El sistema de vida de Mike será igual que el de estamina. Tambíen 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:

Texto

Descripción generada automáticamente

## HUID básico

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:

Imagen en blanco y negro

Descripción generada automáticamente con confianza baja

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

Captura de pantalla de un celular

Descripción generada automáticamente

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:

Imagen que contiene Diagrama

Descripción generada automáticamente

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.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente

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.

A continuación, y para pulir un poco el diseño, el objeto vacío padre lo he cambiado por un Sprite de fondo y finalmente queda así:

Mapa

Descripción generada automáticamente

## 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.

Una captura de pantalla de un videojuego

Descripción generada automáticamente con confianza baja

Seguidamente he creado un script que controle al enemigo zombie al cual he llamado EnemyContorller y lo he almacenado en una carpeta llamada enemigos dentro de los scripts generales. 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 que 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.

Finalmente tengo que hacer que me puedan quitar vida. Para poder hacerlo cuando me ataquen simplemente me bajarán el contador de vida que ya había implementado.

## Mapeo del entorno

Para poder comenzar haciendo que los enemigos sigan automáticamente a Mike primero deberé crear un mapa que esté mejor desarrollado que el que hay actualmente. Para ello he importado más assets de entorno del mismo creador citado al principio.

Imagen que contiene edificio

Descripción generada automáticamente

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

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Y a continuación he creado el nav mesh del terreno:

Imagen que contiene pastel, tabla, tren, cumpleaños

Descripción generada automáticamente

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

## Enemigos atacando

Los enemigos también deben atacar en el juego. En el caso de los zombies (que servirán de ejemplo para ilusitrar el proceso) 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 esta. Para ello he creado un script llamado EnemyAtack:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

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, cada 9 segundos se elegirá un hijo aleatorio y se spawneará de esa posición un enemigo.

Texto

Descripción generada automáticamente

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

Imagen que contiene viejo, tabla, cubierto, colgando

Descripción generada automáticamente

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.

## Escena de muerte

Cuando Mike muerte debe haber una escena de muerte. Par ello habrá un script asignado a un objeto vacío llamado LaParca que controlará que cuando la vida de Mike llegue a 0 ocurra esta escena. En cuanto al botón de regresar al menú, tendrá el script equipado en cuanto a cambios de escena llamado EscenasController.

1. Recursos del autor en <https://maxparata.itch.io/> [↑](#footnote-ref-1)
2. Recursos del autor en <https://unity-of-fantom.itch.io/voxel-character-pack-v1> [↑](#footnote-ref-2)
3. Mixamo: <https://www.mixamo.com/#/> [↑](#footnote-ref-3)
4. { 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> [↑](#footnote-ref-4)