**Computación Gráfica**

Juego II: Shooter Modificado

Máster Universitario en Ingeniería Informática

 Curso 2020/2021



Luis Blázquez Miñambres

Contenido

[Tabla de ilustraciones 3](#_Toc59694708)

[1. Introducción 5](#_Toc59694709)

[2. Modificaciones realizadas 6](#_Toc59694710)

[2.1. Contador de munición y acción de recarga 6](#_Toc59694711)

[2.2. Mejora del movimiento: correr y saltar 13](#_Toc59694712)

[2.3. Vida de los enemigos 15](#_Toc59694713)

[2.4. Barra de energía 19](#_Toc59694714)

[Bibliografía 23](#_Toc59694715)

# Tabla de ilustraciones

[Ilustración 1: Imágen del contador de munición en el Canvas 6](#_Toc59694716)

[Ilustración 2: Imagen a transformar en sprite 7](#_Toc59694717)

[Ilustración 3: Inspector de la imagen de munición 7](#_Toc59694718)

[Ilustración 4: Componente "Text" deshabilitado en el GameObject UI.Text del canvas 8](#_Toc59694719)

[Ilustración 5: Elementos dispuestos en el Canvas para la modificación de la munición 9](#_Toc59694720)

[Ilustración 6: Elementos en la jerarquía de la escena 9](#_Toc59694721)

[Ilustración 7: Código de JugadorDisparo.cs para la modificación de munición (1) 10](#_Toc59694722)

[Ilustración 8: Código de JugadorDisparo.cs para la modificación de munición (2) 10](#_Toc59694723)

[Ilustración 9: Código de JugadorDisparo.cs para la modificación de munición (3) 11](#_Toc59694724)

[Ilustración 10: Resultado de la modificación de munición y recarga 12](#_Toc59694725)

[Ilustración 11: Código de movimiento de correr del jugador principal en JugadorMovimiento.cs 13](#_Toc59694726)

[Ilustración 12: Código de movimiento de salto del jugador principal en JugadorMovimiento.cs (1) 13](#_Toc59694727)

[Ilustración 13: Código de movimiento de salto del jugador principal en JugadorMovimiento.cs (2) 14](#_Toc59694728)

[Ilustración 14: Código de movimiento de salto del jugador principal en JugadorMovimiento.cs (3) 14](#_Toc59694729)

[Ilustración 15: Modo de ajuste del Canvas de la barra de salud en la escena 15](#_Toc59694730)

[Ilustración 16: Adición del componente Canvas al prefab del enemigo 15](#_Toc59694731)

[Ilustración 17: Inspector de la referencia del modelo del enemigo 16](#_Toc59694732)

[Ilustración 18: Código de barra de salud del enemigo en EnemigoModelo.cs 16](#_Toc59694733)

[Ilustración 19: Aplicación de cambios en los prefabs del enemigo 17](#_Toc59694734)

[Ilustración 20: Resultado de barra de salud del enemigo 17](#_Toc59694735)

[Ilustración 21: Representación del Canvas principal con la barra de energía añadida 19](#_Toc59694736)

[Ilustración 22: Código de la barra de energía del jugador principal en JugadorVida.cs (1) 20](#_Toc59694737)

[Ilustración 23: Código de la barra de energía del jugador principal en JugadorVida.cs (2) 20](#_Toc59694738)

[Ilustración 24: : Código de consumición de la barra de energía del jugador principal al saltar en JugadorMovimiento.cs 21](#_Toc59694739)

[Ilustración 25: Resultado de la barra de energía del jugador principal 21](#_Toc59694740)

# Introducción

En este documento se recoge el informe acerca de las modificaciones realizadas al trabajo II sobre un *shooter* en Unity de la asignatura “Computación Gráfica” del Máster en Ingeniería Informática por la Universidad de Salamanca en el Curso 2020-2021, consistente en la explicación de las modificaciones realizadas paso por paso.

# Modificaciones realizadas

En este apartado, se explicará brevemente un ejemplo sencillo de un videojuego en 2D con una nave que dispara a naves enemigas, desarrollado y enfocado principalmente para ser jugado en multijugador o en línea a partir de los aspectos desarrollados anteriormente.

## Contador de munición y acción de recarga

El primer cambio que se ha realizado en el juego ha sido el de crear un contador de munición del arma que utiliza el jugador principal y que tenga la posibilidad de recargar.

En primer lugar, se creará la interfaz del contador de munición en el Canvas principal de la escena, donde se encuentran también la vida del personaje y el contador de puntuación. Principalmente se tratará de un GameObject UI.Text con el contador de la munición y otro UI.Image que representará la imagen de la munición, tal y como se puede ver en la Ilustración 1.

![Interfaz de usuario gráfica

Descripción generada automáticamente]()

Ilustración 1: Imágen del contador de munición en el Canvas

Para cargar la imagen en el UI.Image es necesario convertirlo a un *sprite*, que es un formato que Unity puede reconocer a partir de la imagen. Este proceso consistirá en seleccionar la imagen dentro del proyecto (en este caso en la carpeta Assets>Textures tal y como se ve en la Ilustración 2) , marcar el tipo de textura como “Sprite (2D and UI)” y aplicar los cambios para que Unity recorte y aplique las modificaciones de *sprite*. El inspector deberá quedar como sigue en la Ilustración 3.

![Interfaz de usuario gráfica, Aplicación, Patrón de fondo

Descripción generada automáticamente]()

Ilustración 2: Imagen a transformar en sprite

![Interfaz de usuario gráfica

Descripción generada automáticamente]()

Ilustración 3: Inspector de la imagen de munición

Además de estos dos elementos se incluirá también un GameObject UI.Text en el Canvas principal que tendrá deshabilitado el componente “Text” como se puede ver en la Ilustración 4, y que representará el texto indicativo que aparecerá en la pantalla cuando el jugador se quede sin munición y tenga que recargar el arma. De esta manera, con el componente “Text” deshabilitado el GameObject estará activo dentro de la escena, pero , inicialmente, no se mostrará en pantalla y permitirá que sea , a través del código de un *script,* que aparezca solo cuando el contador de munición del jugador llega a 0.

![Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente]()

Ilustración 4: Componente "Text" deshabilitado en el GameObject UI.Text del canvas

Todo los elementos añadidos al Canvas principal quedarán dispuestos en pantalla como se muestra en la Ilustración 5 y jerarquizados en la escena tal y como se observar en la Ilustración 6.

![Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media]()

Ilustración 5: Elementos dispuestos en el Canvas para la modificación de la munición

![Texto

Descripción generada automáticamente]()

Ilustración 6: Elementos en la jerarquía de la escena

A continuación, se modificará el script “JugadorDisparo.cs” que contiene todas las acciones referentes al RayCast y disparo del jugador principal. Tal y como se muestran los cambios en las Ilustraciones 7-9, principalmente se tendrán varios componentes públicos para referenciar los elementos de la interfaz de forma que, inicialmente se tendrá un valor de munición de 50 balas y , por medio del método FixUpdate, se irá decrementando el contador cada vez que el jugador dispara y se crea un RayCast.

En el momento en el que el contador del jugador llega a 0, la variable “municionActual” hará que no se cumpla la condición para entrar al código de disparo imposibilitando disparar al jugador si no hay munición. Es entonces cuando se habilitará el componente “Text” del GameObject referente al mensaje de recarga, que indica que pulsando la tecla “R” el jugador recargará la munición, para que se muestre en pantalla. Y , si el jugador pulsa esta tecla , a continuación, se recargará el contador y el valor de la munición y se deshabilitará el componente “Text” del GameObject del mensaje de recarga para que desaparezca el mensaje.

![Texto

Descripción generada automáticamente]()

Ilustración 7: Código de JugadorDisparo.cs para la modificación de munición (1)

![Texto

Descripción generada automáticamente]()

Ilustración 8: Código de JugadorDisparo.cs para la modificación de munición (2)

![Texto

Descripción generada automáticamente]()

Ilustración 9: Código de JugadorDisparo.cs para la modificación de munición (3)

Finalmente, la representación de las acciones descritas anteriormente en este apartado se puede visualizar en el juego en la Ilustración 10.

![Imagen que contiene interior, cama, cuarto, oscuro

Descripción generada automáticamente]()

Ilustración 10: Resultado de la modificación de munición y recarga

## Mejora del movimiento: correr y saltar

En esta modificación del juego se permitirá al jugado realizar dos acciones de movimientos adicionales, la de correr y la de saltar. Para ambas acciones se recurrirá únicamente a la modificación de un script (JugadorMovimiento.cs) ya que las animaciones serán las mismas de movimiento que se tienen.

Por un lado, para permitir al jugador principal correr, se utilizará la tecla “Mayus Izquierdo” y, mientras esta esté pulsada mientras se mueve el jugador principal se incrementará la velocidad del jugador principal a un valor constante de 10. De cualquier otra forma la velocidad actual será la velocidad base en ese momento. Esta modificación se puede observar en la Ilustración 11.

![Texto

Descripción generada automáticamente]()

Ilustración 11: Código de movimiento de correr del jugador principal en JugadorMovimiento.cs

Por otro lado, para permitir al jugador saltar , en el mismo script “JugadorMovimiento.cs”, se utilizará un *flag* “enSalto” que indica cuando el jugador principal está saltando y principalmente se usará para no permitir al jugador correr mientras está en el aire o realizar cualquier tipo de movimiento que puede resultar extraño en la experiencia de juego.

El propio salto en el juego se utilizará mediante una propiedad “AddForce” del componente “RigidBody” del GameObject del jugador principal que le añadirá una fuerza de impulso hacia arriba con un valor constante “impulso” modificable en el editor del juego.

Tal y como se muestra en las Ilustraciones 12-14, el jugador saltará si no se encuentra en el aire (*flag* desactivado) y pulsa la barra espaciadora del teclado.

![Texto

Descripción generada automáticamente]()

Ilustración 12: Código de movimiento de salto del jugador principal en JugadorMovimiento.cs (1)

![Texto

Descripción generada automáticamente]()

Ilustración 13: Código de movimiento de salto del jugador principal en JugadorMovimiento.cs (2)

![Texto

Descripción generada automáticamente]()

Ilustración 14: Código de movimiento de salto del jugador principal en JugadorMovimiento.cs (3)

## Vida de los enemigos

Otra modificación realizada sobre el juego base del *shooter* ha sido añadir una barra de vida a los enemigos desplegados en escena.

El primer paso para llevarlo a cabo ha sido crear un nuevo Canvas en la escena que contenga un GameObject UI.Image dentro , similar al de la barra de vida del jugador principal pero coloreado en rojo y anclado a la parte inferior del Canvas. Se reduce el tamaño y se ajusta.

Para conseguir que la barra de salud siga al enemigo y se ajuste a su movimiento dentro de la escena y no se mantenga estática como el Canvas de la interfaz del jugador principal es necesario indicar en la propiedades del Canvas que el modo de renderización o “Render Mode” es “World Space” para que se puede mover libremente por el conjunto de la escena. Esto se puede observar en la Ilustración 15.

![Captura de pantalla de un celular

Descripción generada automáticamente]()

Ilustración 15: Modo de ajuste del Canvas de la barra de salud en la escena

A continuación, se seleccionará el *prefab* , anteriormente creado, del enemigo (Zombunny) y se adherirá como componentes hijos el Canvas con la imagen, tal y como se muestra en la Ilustración 16. De esta manera, se ajusta la posición adecuadamente encima del enemigo y quedaría correctamente ajustado , siguiendo al enemigo según se mueve.

![Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente]()

Ilustración 16: Adición del componente Canvas al prefab del enemigo

A continuación, a través del código del script “EnemigoModelo.cs” se hará efectivo que la barra de salud del enemigo decrezca según la salud del enemigo disminuye. Para esto se referencia la barra de salud del *prefab* dentro del script anteriormente mencionado como se ve en la Ilustración 17.

![Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente]()

Ilustración 17: Inspector de la referencia del modelo del enemigo

En cuanto al código del script tal y como se ve en la Ilustración 18, teniendo la referencia de la barra de salud de cada *prefab* del enemigo , será necesario calcular y ajustar el tamaño de la barra de salud al valor de la salud del enemigo (tal y como se hace con la barra de salud del jugador principal) y actualizando este tamaño según se decrementa la vida del enemigo.

![Texto

Descripción generada automáticamente]()

Ilustración 18: Código de barra de salud del enemigo en EnemigoModelo.cs

Sin embargo, para que los cambios aplicados se muestren inherentes a todos los clones del *prebab* del enemigo se irá al inspector y en la opción “Overrides” se dará a “Apply all” para que se apliquen los cambios. Estos se pueden ver en la Ilustración 19.

![Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente]()

Ilustración 19: Aplicación de cambios en los prefabs del enemigo

Finalmente, como resultado se obtiene en escena lo mostrado en la Ilustración 20.

![Imagen que contiene interior, verde, cama, computadora

Descripción generada automáticamente]()

Ilustración 20: Resultado de barra de salud del enemigo

## Barra de energía

El último cambio realizado en el juego base es el de añadir una barra de energía o , conocido en los juegos , *stamina* que se decrementará cada vez que el jugador salte.

Primero se añadirá un nuevo GameObject UI.Image similar al de la barra de salud del jugador principal, pero en color azul dentro del Canvas de la interfaz principal del jugador tal y como se representa en la Ilustración 21.

![Interfaz de usuario gráfica, Gráfico

Descripción generada automáticamente]()

Ilustración 21: Representación del Canvas principal con la barra de energía añadida

Posteriormente, para hacer efectivo el uso de la energía del jugador, se modificarán dos scripts: “JugadorVida.cs” y “JugadorMovimiento.cs”.

Por un lado, en el script “JugadorVida.cs”, como se ve en las Ilustraciones 22-23 , se añadirán los componentes análogos a los de la barra de salud del jugador para tener la referencia de la barra de energía, así como su valor. Además, se añadirá una función “restarEnergia” que restará una cantidad de energía determinada a la del jugador cada vez que se realice una acción determinada y haciendo que se baje la barra de energía. Para recuperar esta energía, será necesario que la energía llegue a 0, y esta se recuperará automáticamente como ocurre en otros videojuegos similares.

Por otro lado, en el script “JugadorMovimiento.cs”, tal y como se observa en la Ilustración 24, se llamará a la función “restarVida” de la clase “JugadorVida” que disminuirá la energía del jugador cada vez que este realiza la acción de saltar. La modularidad de esta función permite que se aplique a cualquier acción de movimiento que realice el jugador y que le permita consumir energía. De esta forma, cada vez que el jugador realice la acción de saltar se consumirá un parte de su energía.

![Texto

Descripción generada automáticamente]()

Ilustración 22: Código de la barra de energía del jugador principal en JugadorVida.cs (1)

![Texto

Descripción generada automáticamente]()

Ilustración 23: Código de la barra de energía del jugador principal en JugadorVida.cs (2)

![Texto

Descripción generada automáticamente]()

Ilustración 24: : Código de consumición de la barra de energía del jugador principal al saltar en JugadorMovimiento.cs

Finalmente, como resultado de esta modificación, como se observa en la Ilustración 25, el jugador tiene una barra de energía que consumirá al saltar.

![Imagen que contiene camino, pequeño, hombre, cuarto

Descripción generada automáticamente]()

Ilustración 25: Resultado de la barra de energía del jugador principal

# Bibliografía

|  |  |
| --- | --- |
| [1] | «Webopedia,» [En línea]. Available: https://www.webopedia.com/definitions/multiplayer/. |
| [2] | «Pogo,» [En línea]. Available: https://www.pogo.com/. |