Memoria Práctica Shooter2D

Jorge Bárcena Lumbreras

Animación y Scripting

MEMORIA – PRÁCTICA SHOOTER 2D

1.- Introducción

El objetivo final de esta práctica era aplicar todo lo que hemos aprendido en Unity durante lo que llevamos de curso. En esta práctica se podían añadir co-rutinas, físicas2D, delegados…

La práctica consistía en crear un shooter 2D, siguiendo unas directrices muy determinadas. Dicho shooter se desarrollará de la manera siguiente: El jugador deberá apuntar con el ratón a una parte de la pantalla, cuando haga click izquierdo con el ratón, la flecha situada en la parte inferior de la pantalla lanzará un proyectil con el objetivo de impactar en los aviones que estarán pasando en la parte media de la pantalla, indefinidamente. Por lo tanto, el objetivo final del jugador será derrotar el número máximo de aviones en cada partida.

Antes de comenzar a desarrollar el pequeño mini-juego, atendí en clase, mientras el profesor explicaba la práctica, fui recopilando todas las funciones que iban a ser necesarias para desarrollar con éxito el mini-juego. Funciones como *"LookAt()"* o *"Camera.ScreenToWorldPoint()"*. Llegue a ellas mediante la explicación en clase. También, me dispuse a buscar una serie de arte 2D, que diera una estética distinta a la propuesta en el ejercicio.

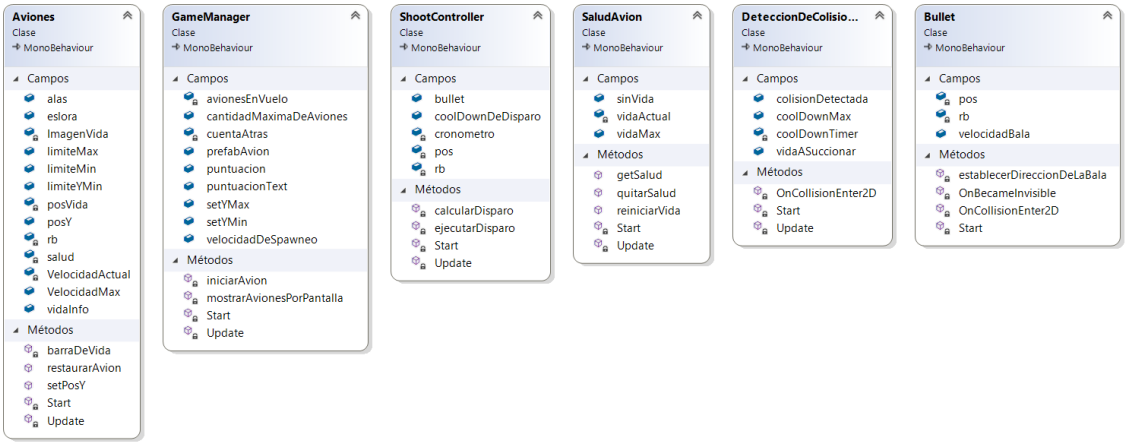
2.- Descripción Técnica

2.1 - Análisis Funcional.

La práctica consistía en crear un shooter 2D, siguiendo unas directrices muy determinadas. Dicho shooter se desarrollará de la manera siguiente: El jugador deberá apuntar con el ratón a una parte de la pantalla, cuando haga click izquierdo con el ratón, la flecha situada en la parte inferior de la pantalla lanzará un proyectil con el objetivo de impactar en los aviones que estarán pasando en la parte media de la pantalla, indefinidamente. Por lo tanto, el objetivo final del jugador será derrotar el número máximo de aviones en cada partida.

2.2 - Diagrama de clases: métodos y miembros.

A continuación, se adjuntan un diagrama de todas las clases, con sus correspondientes variables, que intervienen en el shooter 2D.



2.3 – Consideraciones especiales.

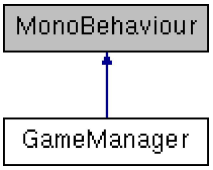
1. Los gameobjects de los aviones no se destruyen, sino que se resetean de posición y de vida.
2. Cuando se ha derribado un avión tarda unos segundos hasta que sube al marcador.
3. Hay un tiempo de cooldown en el cañón (El diseñador lo podrá modificar).
4. Hay dos zonas de impacto en el avión que determinan cuando daño hacen.

2.4 – Comentarios en el código.

A continuación, se incluye un documento generado en “Doxygen”, sobre las clases y sus correspondientes métodos y atributos.

GameManager Class Reference

Inheritance diagram for GameManager:



Public Attributes

float	**setYMax** Maxima y minima "Y" de spawneo. More...
GameObject	**prefabAvion** Prefab de los aviones More...
int	**cantidadMaximaDeAviones** = 4 Cantidad maxima de aviones More...
float	**velocidadDeSpawneo** Frecuencia de Spawneo en segundos More...
Text	**puntuacionText** Texto que muestra la cantidad de aviones derribados More...

Static Public Attributes

static int	**puntuacion** = 0 Cantidad de aviones derribados More...

Member Data Documentation

◆ cantidadMaximaDeAviones

int GameManager.cantidadMaximaDeAviones = 4

Cantidad maxima de aviones

◆ prefabAvion

GameObject GameManager.prefabAvion

Prefab de los aviones

puntuacion

int GameManager.puntuacion = 0 static

Cantidad de aviones derribados

puntuacionText

Text GameManager.puntuacionText

Texto que muestra la cantidad de aviones derribados

setYMax

float GameManager.setYMax

Maxima y minima "Y" de spawneo.

velocidadDeSpawneo

float GameManager.velocidadDeSpawneo

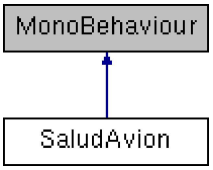
Frecuencia de Spawneo en segundos

The documentation for this class was generated from the following file:

- D:/Documentos/Universidad/Scripting - Practica final/PracticaFinal/Shooter/Assets/Scripts/s1/GameManager.cs

SaludAvion Class Reference

Inheritance diagram for SaludAvion:



Public Member Functions

void **reiniciarVida** ()
Establece la vida normal More...

void **quitarSalud** (int a)
Quita x vida al avion More...

int **getSalud** ()
Devuelve los puntos de vida actuales More...

Public Attributes

int **vidaMax** = 500
Puntos de vida del avion More...

bool **sinVida** = false
Si el avion esta derribado o no More...

Member Function Documentation

◆ getSalud()

int SaludAvion.getSalud ( )

Devuelve los puntos de vida actuales

Returns

◆ quitarSalud()

void SaludAvion.quitarSalud (int a)

Quita x vida al avion

Parameters

a Cantida de vida a succionar

◆ reiniciarVida()

void SaludAvion.reiniciarVida ( )

Establece la vida normal

Member Data Documentation

◆ sinVida

bool SaludAvion.sinVida = false

Si el avion esta derribado o no

◆ vidaMax

int SaludAvion.vidaMax = 500

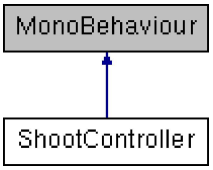
Puntos de vida del avion

The documentation for this class was generated from the following file:

- D:/Documentos/Universidad/Scripting - Practica final/PracticaFinal/Shooter/Assets/Scripts/s1/SaludAvion.cs

ShootController Class Reference

Inheritance diagram for ShootController:



Public Attributes

GameObject **bullet**
Prefab de la bala More...

float **coolDownDeDisparo** = 1
Tiempo de recarga de coolDown More...

Member Data Documentation

◆ bullet

GameObject ShootController.bullet

Prefab de la bala

◆ coolDownDeDisparo

float ShootController.coolDownDeDisparo = 1

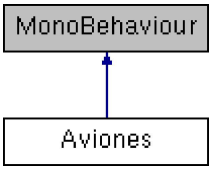
Tiempo de recarga de coolDown

The documentation for this class was generated from the following file:

- D:/Documentos/Universidad/Scripting - Practica final/PracticaFinal/Shooter/Assets/Scripts/s1/ShootController.cs

Aviones Class Reference

Inheritance diagram for Aviones:



Public Member Functions

void	**setPosY** (float a)
void	**restaurarAvion** () Cada vez que un avion vuelve a aparecer por pantalla More...

Public Attributes

float	**VelocidadMax** = 200 Velocidad del avion More...
float	**limiteMin** Limites de Spawn More...
float	**posY** Posicion Y del avion More...
DeteccionDeColisiones	**eslora** Detecta donde se ha recibido el impacto More...
GameObject	**vidaInfo** GameObject del cambas que almacena la vida More...
float	**limiteYMin** Limite Y minimo More...

Member Function Documentation

◆ restaurarAvion()

void Aviones.restaurarAvion ( )

Cada vez que un avion vuelve a aparecer por pantalla

Member Data Documentation

◆ eslora

DeteccionDeColisiones Aviones.eslora

Detecta donde se ha recibido el impacto

◆ limiteMin

float Aviones.limiteMin

Limites de Spawn

◆ limiteYMin

float Aviones.limiteYMin

Limite Y minimo

◆ posY

float Aviones.posY

Posicion Y del avion

◆ VelocidadMax

float Aviones.VelocidadMax = 200

Velocidad del avion

◆ vidaInfo

GameObject Aviones.vidaInfo

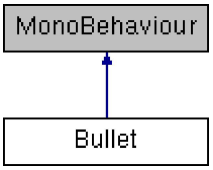
GameObject del cambas que almacena la vida

The documentation for this class was generated from the following file:

- D:/Documentos/Universidad/Scripting - Practica final/PracticaFinal/Shooter/Assets/Scripts/s1/Aviones.cs

Bullet Class Reference

Inheritance diagram for Bullet:



Public Attributes

float **velocidadBala** = 300
Velocidad de la bala More...

Member Data Documentation

◆ velocidadBala

float Bullet.velocidadBala = 300

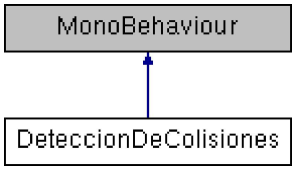
Velocidad de la bala

The documentation for this class was generated from the following file:

- D:/Documentos/Universidad/Scripting - Practica final/PracticaFinal/Shooter/Assets/Scripts/s1/Bullet.cs

DeteccionDeColisiones Class Reference

Inheritance diagram for DeteccionDeColisiones:



Public Attributes

bool **colisionDetectada** = false
Determina si hay o no algun tipo de colision More...

int **vidaASuccionar** = 100
Cantidad de vida que se va a supcionar More...

float **coolDownMax** = 2
Persiodo de cooldown More...

Member Data Documentation

◆ colisionDetectada

bool DeteccionDeColisiones.colisionDetectada = false

Determina si hay o no algun tipo de colision

◆ coolDownMax

float DeteccionDeColisiones.coolDownMax = 2

Persiodo de cooldown

◆ vidaASuccionar

int DeteccionDeColisiones.vidaASuccionar = 100

Cantidad de vida que se va a supcionar

The documentation for this class was generated from the following file:

- D:/Documentos/Universidad/Scripting - Practica final/PracticaFinal/Shooter/Assets/Scripts/s1/DeteccionDeColisiones.cs

---

3.- Diario de Desarrollo

Durante el desarrollo del mini-juego no encontré muchas complicaciones, quizás la complicación más importante que encontré sería, la función de LookAt(). No conseguí utilizar correctamente la función para orientar la flecha que lanza los proyectiles, debido a esto, me dispuse a buscar otra forma de rotación de objetos, tras buscar en internet como hacer esto, encontré otra manera de rotar el objeto con el Rigidbody2D.rotation. La función que se encarga de determinar el ángulo de dirección se obtiene a través de la tangente entre la posición X e Y del ratón en la pantalla, y más tarde haciendo una conversión de radianes a ángulos.

Mas allá de este problema, no he tenido muchos más problemas durante el desarrollo.

En mi opinión el desarrollo de esta práctica ha sido relativamente fácil con otras prácticas, pero, aun así, creo que ha sido muy útil debido a que he aprendido mejor como manejar el comportamiento de las físicas 2D, además de utilizar una técnica de reciclado de gameobjects en la escena.