**Ciclo Formativo de Grado Superior**

Desarrollo de Aplicaciones

Multiplataforma.

PROYECTO INTEGRADO

**[TheBubble-GAME]**

Presentado por:

**[SANTIAGO OJEDA ARENAS]**

Tutor:

**[Diego Linarez Ortiz]**

Curso académico [2022/2023]



INDICE

[INTRODUCCION 3](#_Toc135310971)

[DESCRIPCION 3](#_Toc135310972)

[INSTALACION Y PREPARACION 3](#_Toc135310973)

[COMO JUGAR 4](#_Toc135310974)

[Funcionamiento del juego 4](#_Toc135310975)

[Main Camera 5](#_Toc135310976)

[Directional Light 6](#_Toc135310977)

[GameManager 7](#_Toc135310978)

[Canvas 8](#_Toc135310979)

[Scenario 14](#_Toc135310980)

[Pool 15](#_Toc135310981)

[Burbujaexplotar 16](#_Toc135310982)

[Astilla 17](#_Toc135310983)

[Controlador 18](#_Toc135310984)

[Assets 19](#_Toc135310985)

[Internet 25](#_Toc135310986)

[0\_documentacion.txt 26](#_Toc135310987)

[Header.php 26](#_Toc135310988)

[Reg\_user.php 27](#_Toc135310989)

[Login.php 29](#_Toc135310990)

[Edit\_usuario.php 30](#_Toc135310991)

[Return\_score.php 31](#_Toc135310992)

[Scripts 32](#_Toc135310993)

[GameManager 33](#_Toc135310994)

[Pool 36](#_Toc135310995)

[Burbujas 40](#_Toc135310996)

[Astilla 43](#_Toc135310997)

[Web/Servidor 45](#_Toc135310998)

[Web/Register 47](#_Toc135310999)

[Web/Login 49](#_Toc135311000)

[Web/Upload 51](#_Toc135311001)

[VOCABULARIO 53](#_Toc135311002)

INTRODUCCION

En este proyecto realizare un videojuego puesto que es a lo que me gustaría dedicarme.

Existen pocos programadores de videojuegos profesionales que son de Cádiz y yo aspiro a ser uno de ellos.

Para el proyecto usare el motor de unity y el editor de texto visual studio 2019.

DESCRIPCION

El juego que he creado trata de explotar burbujas en un tiempo limitado.

Hay dos tipos de burbujas una azul que se explota con una sola pulsación y te da un punto y una burbuja amarilla que se explota con tres toques y te da 3 puntos.

Tienes 60 segundos para explotar el máximo de burbujas posibles.

INSTALACION Y PREPARACION

Windows:

Descarga el archivo TheBubble.zip del enlace que te dejamos a continuación [Pincha aquí](https://payoyo12.itch.io/the-bubbles).

Extraer su contenido.

Abre la carpeta TheBubble y ejecuta el archivo TheBubble.exe.

Disfruta del juego.

Android:

En esta versión se guardan los datos en local, una vez desinstalado se perderán los datos.

Descarga el archivo TheBubble.apk del enlace que te dejamos a continuación [Pincha aquí](https://payoyo12.itch.io/the-bubbles).

Ejecuta el archivo y sigue los pasos de instalación.

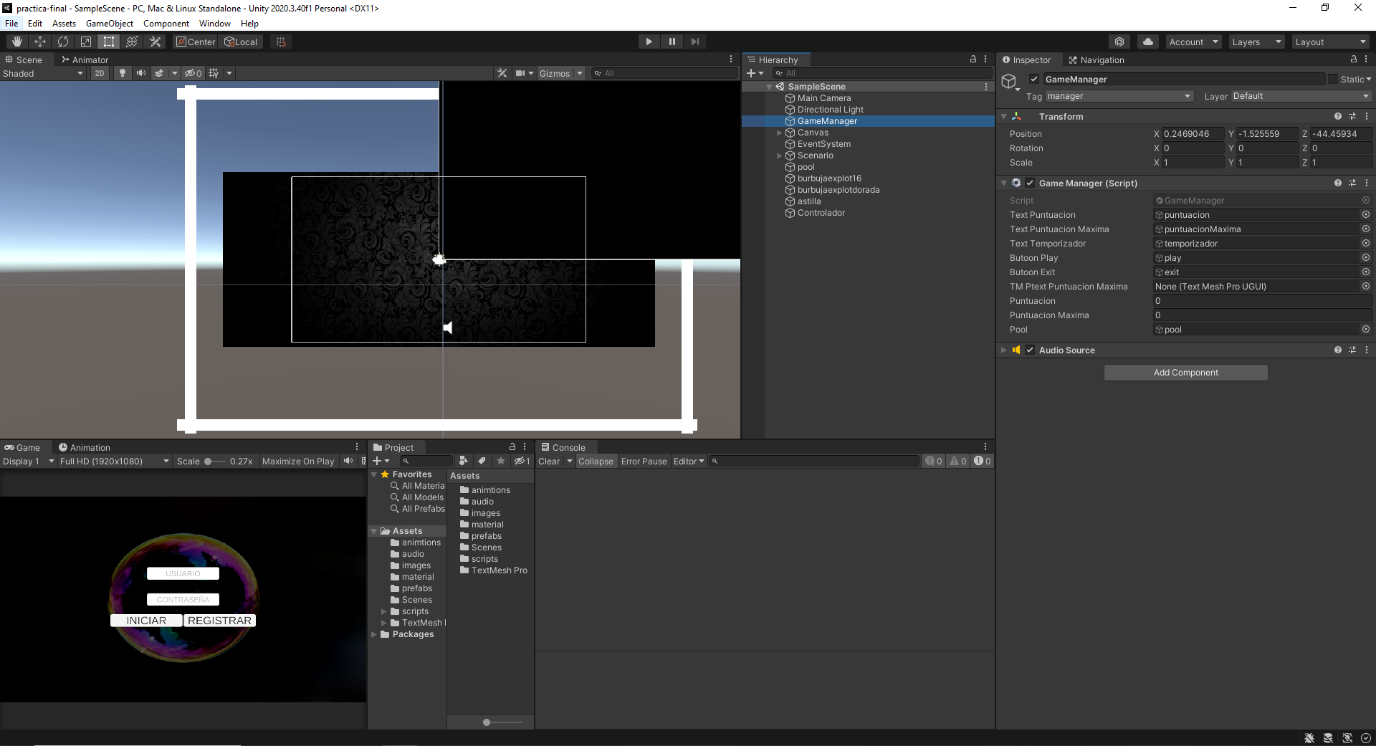
Disfruta del juego.

COMO JUGAR

El juego trata de hacer explotar burbujas que van apareciendo en el mapa, con el objetivo de conseguir la mejor puntuación posible en un tiempo de un minuto.



Funcionamiento del juego



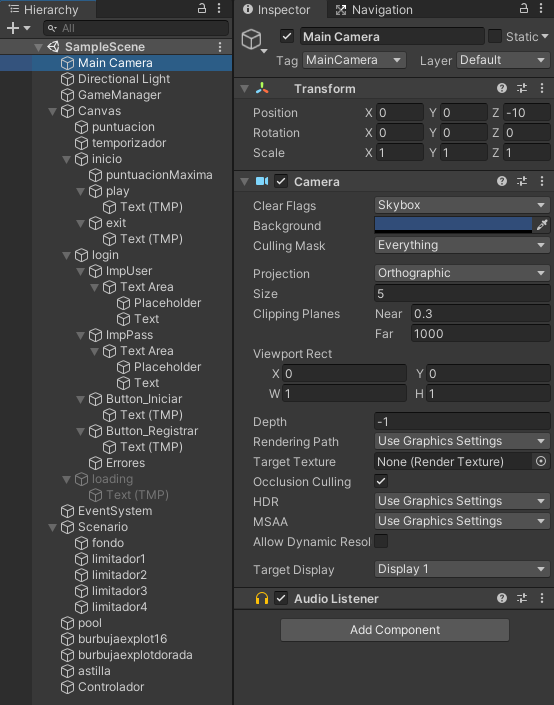
Para un mejor entendimiento del funcionamiento de unity acceda al apartado del Diccionario.

Para un mejor entendimiento de cómo están hechos cada script acceda al apartado de Scripts.

Aquí se explicará cómo funciona la aplicación y sus relaciones entre objetos y archivos.

Para una explicación más ordenada lo aremos siguiendo el orden de la jerarquía.

# Main Camera



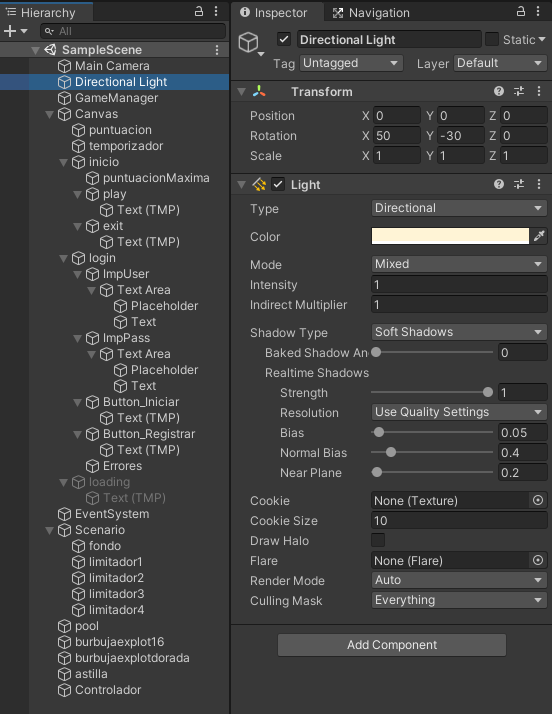
La "Main Camera" en Unity es un objeto esencial que define la vista principal del jugador dentro del juego. Es responsable de renderizar la escena y mostrarla en la pantalla.

El componente Transform se ajusta la posición de Z a –10 para que se puedan visualizar el resto de los objetos.

El componente Camera se deja todo por defecto y ponemos la proyección en Ortográfica puesto que estamos haciendo un juego en 2D.

El componente Audio Listener viene por defecto y sirve para poder escuchar los sonidos del juego.

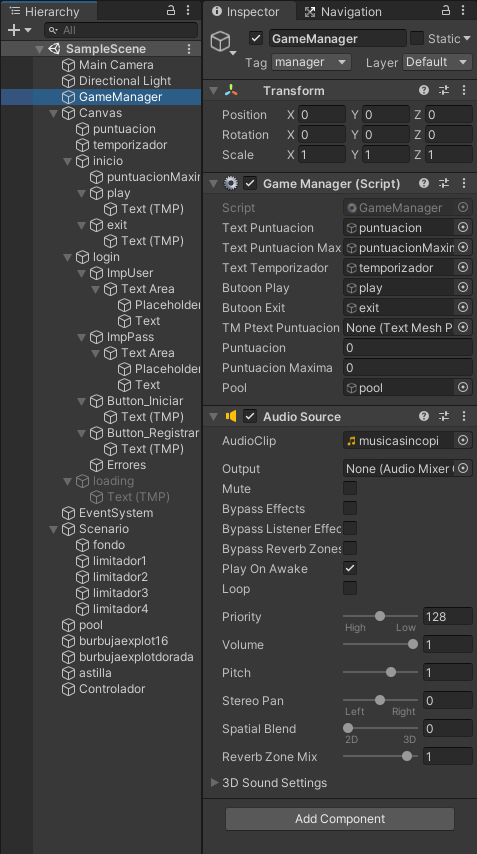
# Directional Light



La "Directional Light" en Unity es un tipo de luz que emite una iluminación uniforme y paralela en una dirección específica.

El componente Transform y Light se dejan por defecto puesto que para esta aplicación no realizaremos ningún cambio específico de iluminación.

# GameManager



El "GameManager" en Unity es un componente o script que se utiliza para gestionar y controlar el flujo del juego. Su función principal es coordinar y supervisar diversos aspectos del juego, como el inicio y finalización de niveles, la gestión de puntuaciones, la gestión de vidas, la carga y guardado de datos, la gestión de eventos y mucho más.

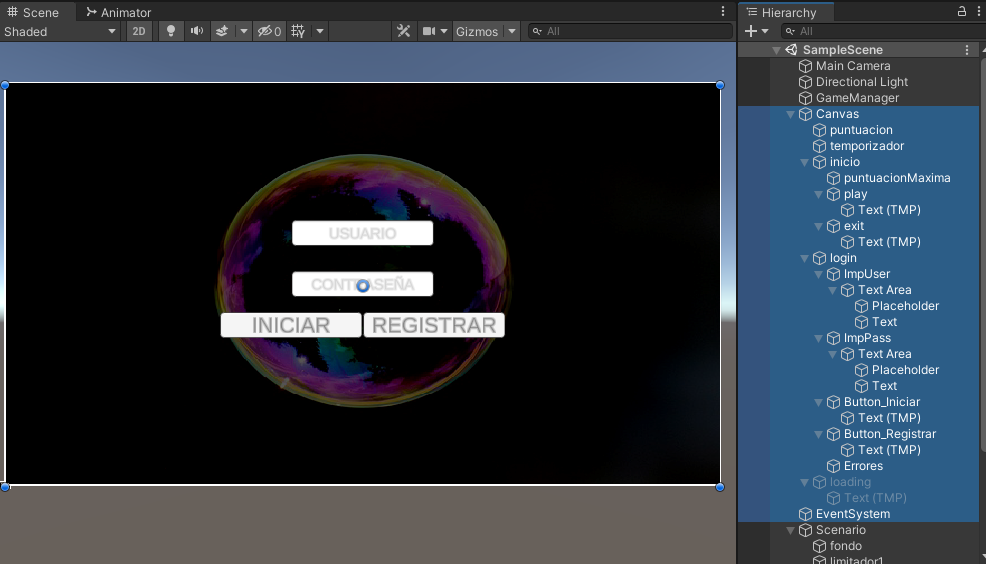
El componente Transform se deja por defecto.

El componente Game Manager (Script) es el script encargado de la gestión del juego, ve al apartado de Scripts para verlo más en detalle, todos los atributos públicos se muestran en el inspector para poder arrastrar a estos lugares el objeto al que se hace referencia si es necesario puesto que algunos se ven porque es necesario acceder a ellos desde otro script y por eso pone None en su lugar.

Para un mejor entendimiento de cómo están hechos cada script acceda al apartado de Scripts.

El componente Audio Source se encarga de reproducir lo que será la música de fondo con el apartado Play On Awake activado para que se ejecute desde que se inicia la aplicación.

# Canvas



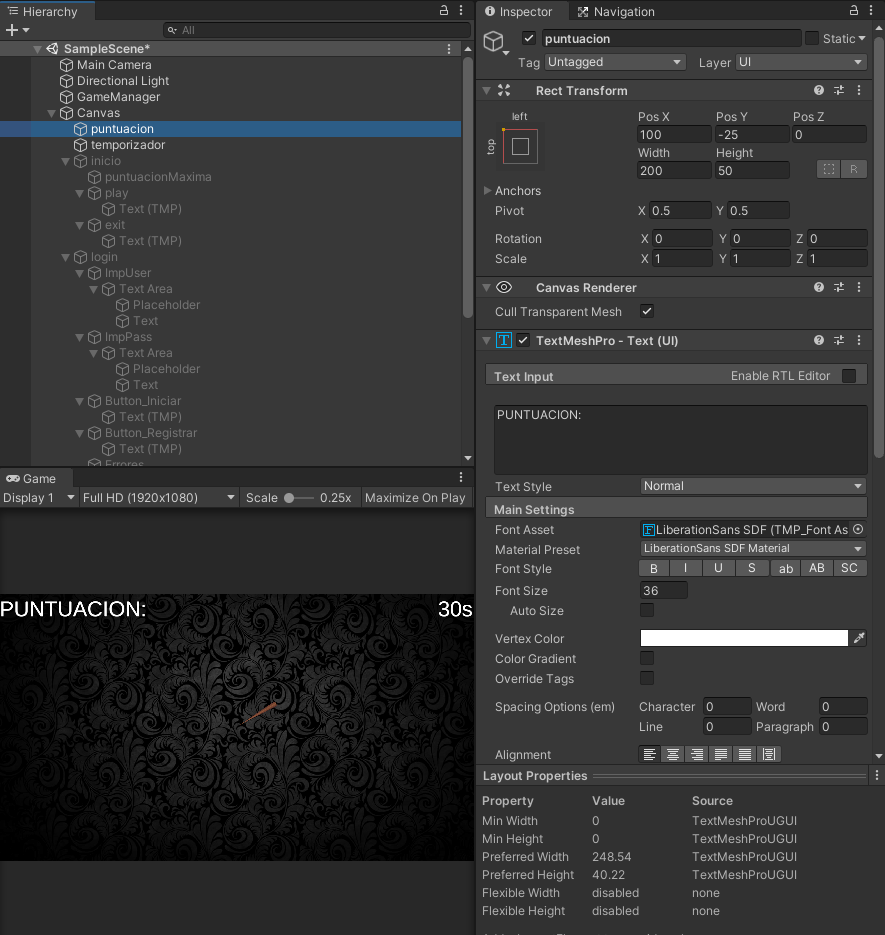
El "Canvas" en Unity es un componente que se utiliza para crear y administrar interfaces de usuario (UI) en un juego. Proporciona un espacio en el que se pueden colocar elementos visuales, como botones, texto, imágenes y paneles, para interactuar con el jugador.

Este viene siempre acompañado del EventSystem para funcionar.

El "EventSystem" en Unity es un componente fundamental que se utiliza para manejar la interacción y la detección de eventos en la interfaz de usuario. Proporciona una forma de capturar y procesar eventos de entrada, como clics de botones, movimientos del ratón, toques en pantalla, entre otros.

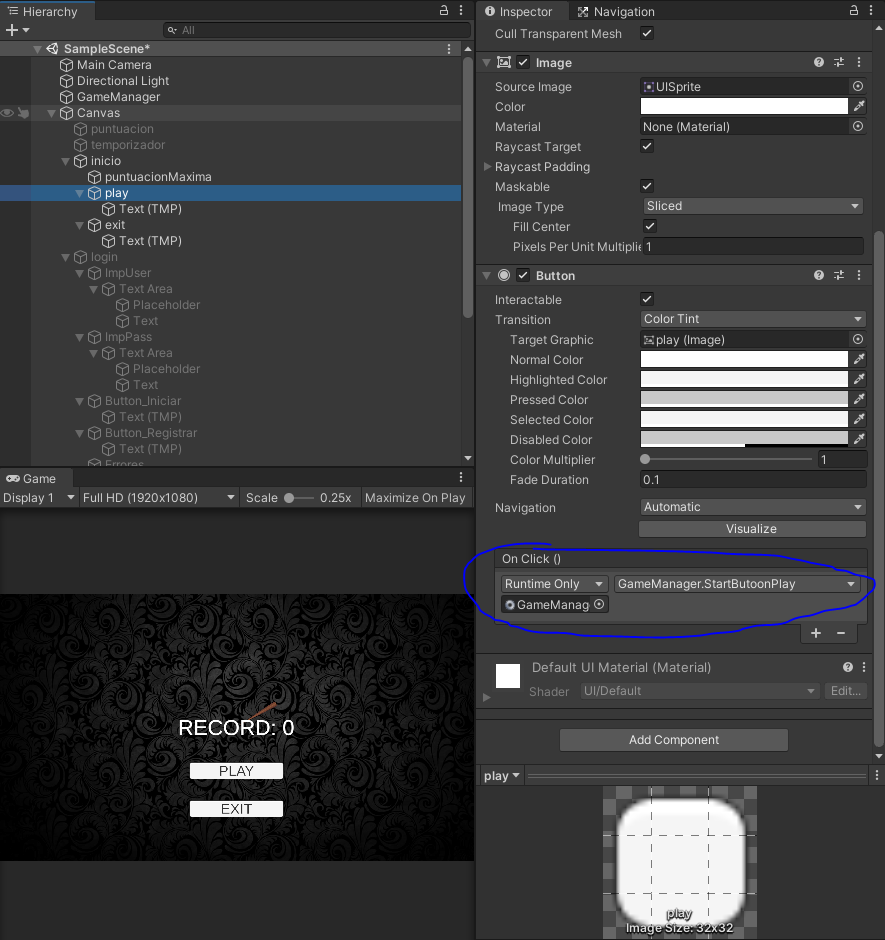
El orden del Canvas en Unity determina cómo se representan y se superponen los elementos visuales en una escena.

Los objetos de interfaz tienen un Rect Transform que sustituye al Transform de los objetos normales y es también un componente por defecto y sirve para posicionar cada interfaz en el sitio que queremos que se vea.



Como se muestra en la pantalla de Game nos encontramos con la puntuación y temporizador.

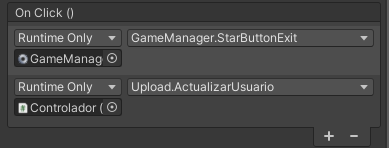
Estos dos objetos son Text-TextMeshPro que como se ve en el inspector con el cual se le da formato al texto con propiedades como alineación, tamaño y color de letra entre otros.



Como se muestra en la pantalla de Game nos encontramos en la pantalla inicial, esta consta de un Text-TextMeshPro para mostrar la puntuación máxima alcanzada y dos botones para iniciar la partida y salir del juego los cuales tienen su propio Text-TextMeshPro.

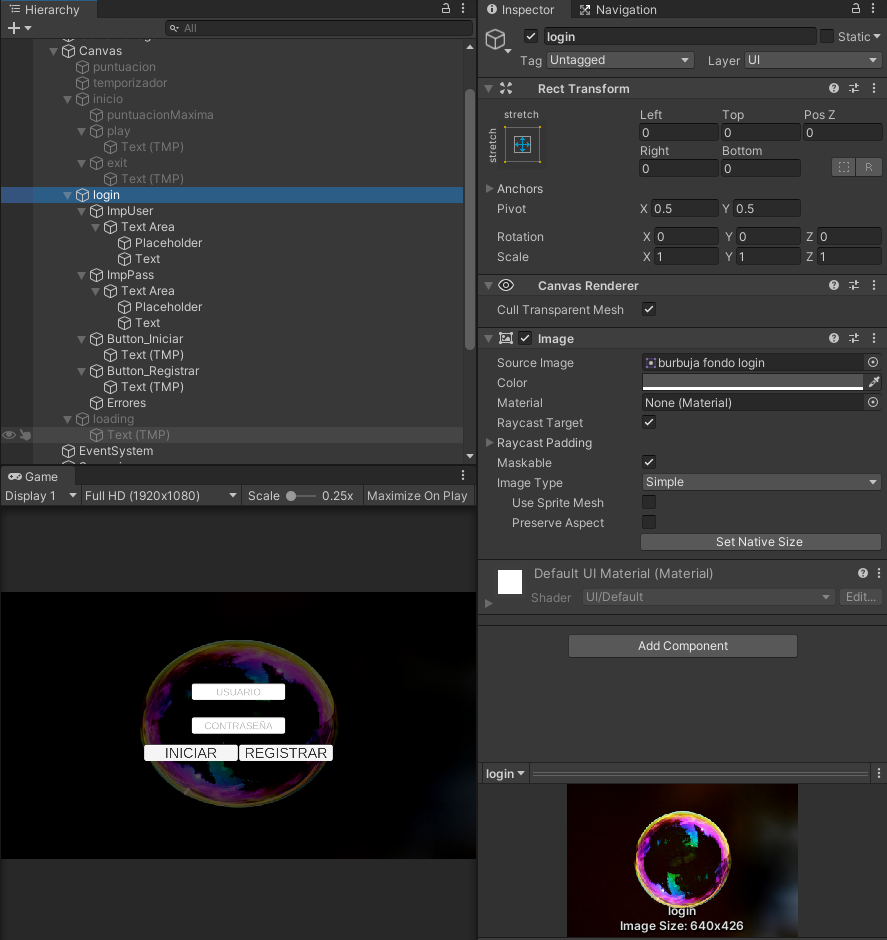
Los botones tienen el componente Button con el apartado On Click() en el cual se pone lo que este tiene que realizar cuando sea pulsado.

Para el botón Play queremos que nos ejecute la función StarButtonPlay que se encuentra en el script GameManager.cs que lo tiene asignado el objeto GameManager.



Para el botón Exit queremos que nos ejecute la función StarButtonExit que se encuentra en el script GameManager.cs que lo tiene el objeto GameManager.

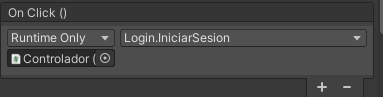
También queremos que nos ejecute la función ActualizarUsuario que se encuentra en el script Upload que lo tiene asignado el objeto Controlador.



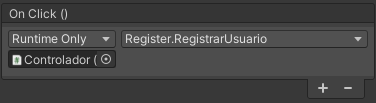
Como se muestra en la pantalla de Game nos encontramos en la pantalla encardada del login y registro del usuario, esta consta de una imagen de fondo, dos ImputField para el nombre y la contraseña los cuales tienen un Placeholder y un Text-TextMeshPro cada uno y dos botones para logearse y registrarse los cuales tienen su propio Text-TextMeshPro, por último tiene un Text-TextMeshPro en el que se mostraran los errores.

El Placeholder es un Text-TextMeshPro que muestra el texto de fondo para que el usuario sepa lo que tiene que escribir en ese apartado y el Text es donde escribe el usuario.

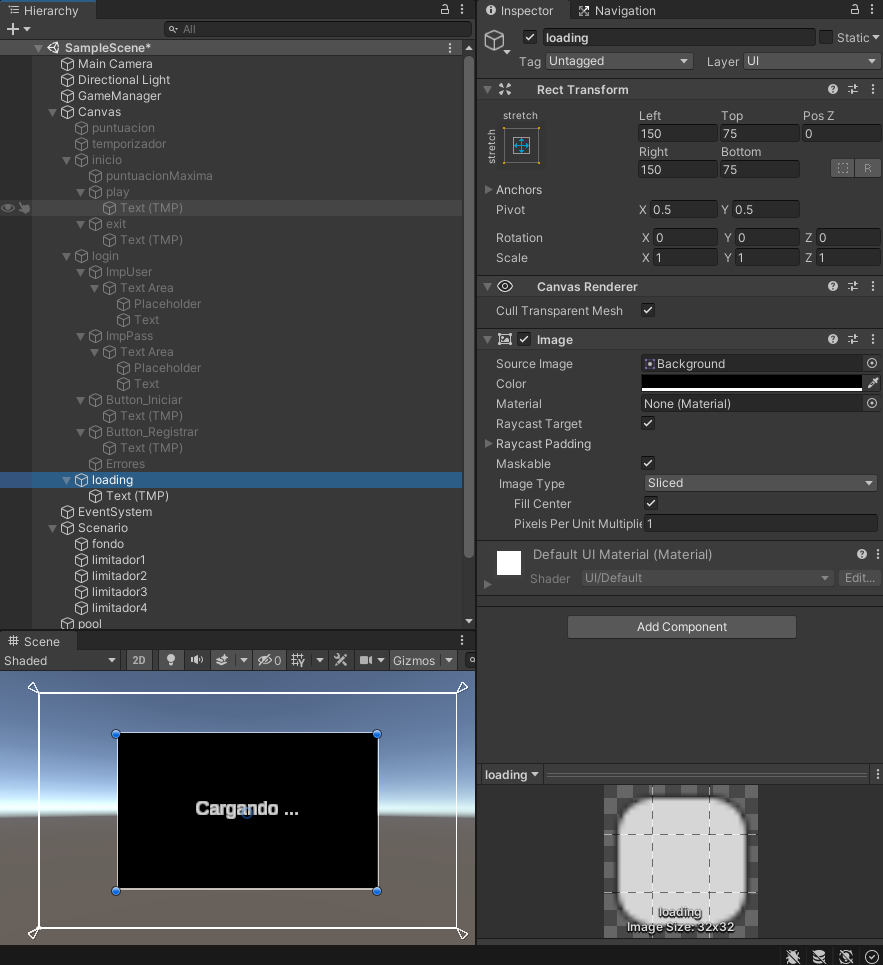
Los botones tienen el componente Button con el apartado On Click() en el cual se pone lo que este tiene que realizar cuando sea pulsado.



Para el botón Iniciar queremos que nos ejecute la función IniciarSesion que se encuentra en el script Login.cs que lo tiene asignado el objeto Controlador.

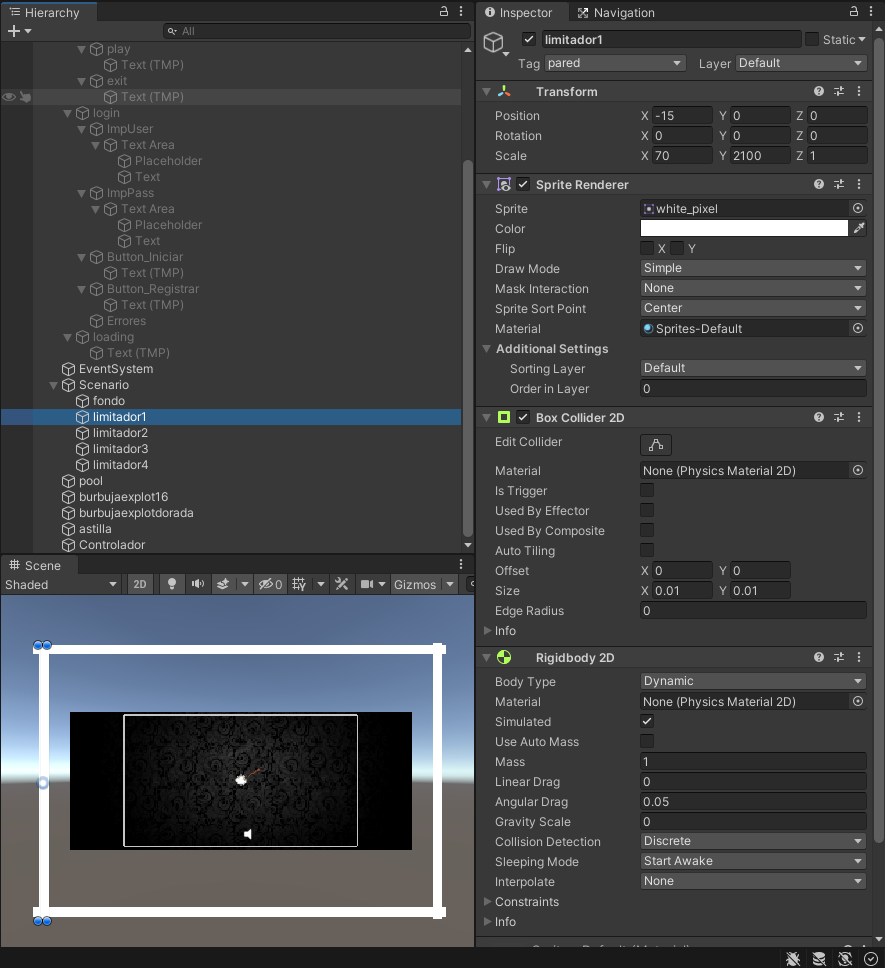


Para el botón Registrar queremos que nos ejecute la función RegistrarUsuario que se encuentra en el script Register.cs que lo tiene asignado el objeto Controlador.



Por último, tiene una pantalla de carga que contiene una imagen de fondo y un Text-TextMeshPro que te muestra un texto indicando al usuario que se están cargando los datos.

# Scenario



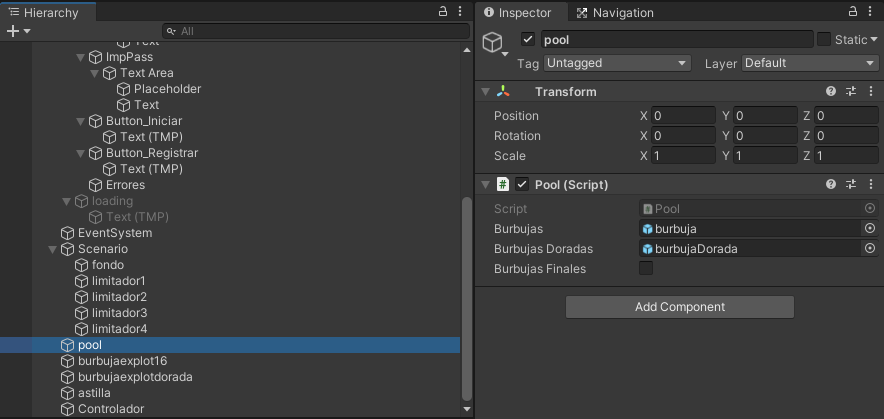
El objeto Scenario tal como indica su nombre contiene las piezas que conforman el scenario del juego. Esta consta de una imagen de fondo y cuatro pilares para limitar el mapa. Estos limitadores servirán para que las burbujas no se ballan al infinito haciendo que reboten sobre ellos y a sin vuelvan a aparecer en pantalla desde un punto distinto del que iniciaron y como las burbujas rebotan también entre si al final crea ese efecto de que las burbujas salgan desde cualquier punto creando la dificultad y magia del juego.

El "Box Collider 2D" en Unity es un componente que se utiliza para definir y controlar el área de colisión de un objeto en un entorno 2D. Proporciona una forma rectangular y bidimensional que rodea al objeto y permite detectar colisiones con otros objetos en el juego.

El "Rigidbody 2D" en Unity es un componente utilizado para simular la física de objetos en un entorno 2D. Proporciona comportamientos realistas, como la gravedad, la colisión y la respuesta a las fuerzas aplicadas, lo que permite que los objetos se muevan y se comporten de manera natural en el juego.

De este componente de deja todo por defecto excepto el Gravity Scale que se pone a 0 para que no tenga gravedad y así no se caiga al vacío no se mueva del sitio.

# Pool

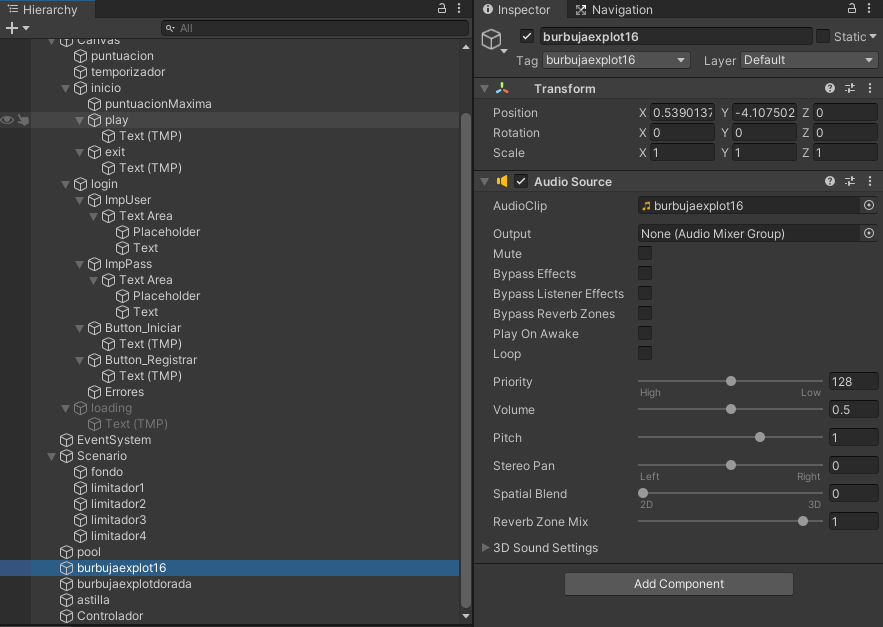


La pool (Piscina de Objetos) se encarga de gestionar la aparición de las burbujas gracias al script que lo compone en el cual se hace referencia a prefabs que serán las burbujas que el usuario tendrá que explotar.

Para un mejor entendimiento de cómo están hechos cada script acceda al apartado de Scripts.

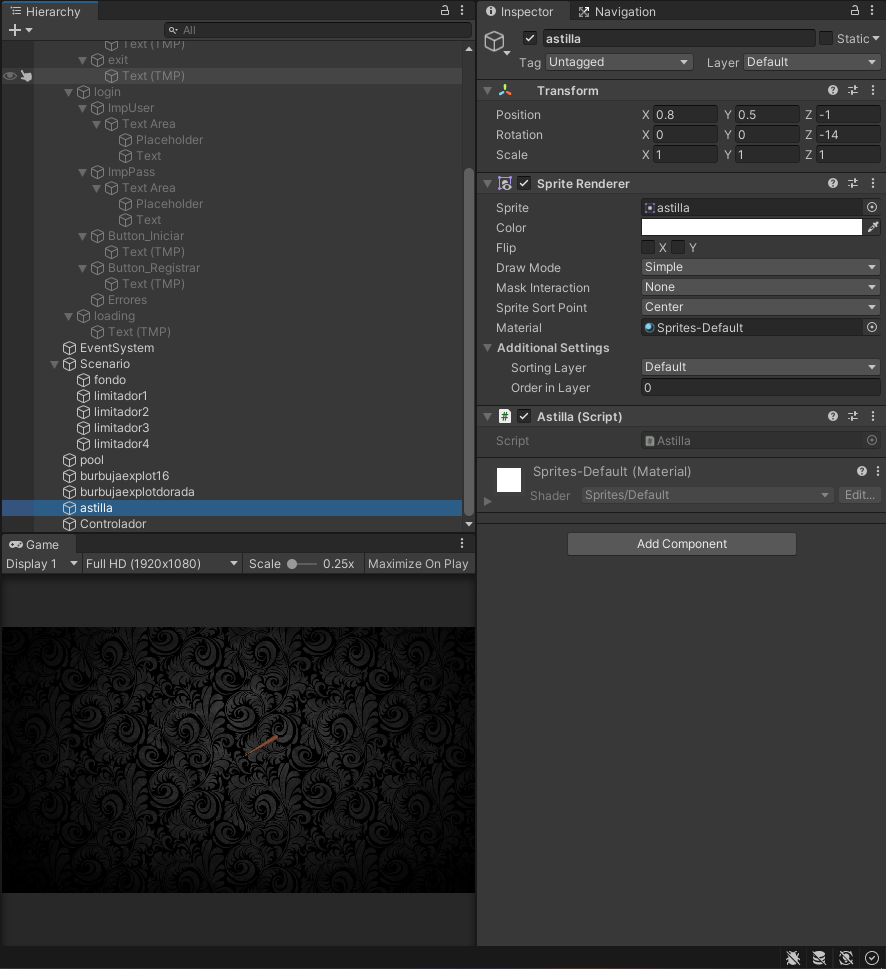
Un "Prefab" (abreviatura de "prefabricado") en Unity es un objeto o conjunto de objetos predefinidos y reutilizables que se pueden instanciar fácilmente en el juego. Básicamente, un Prefab es una plantilla que contiene toda la información y configuraciones necesarias para crear instancias de un objeto específico en el juego.

# Burbujaexplotar



El objeto Burbujaexplotar y Burbujaexplotardorada simplemente contienen es audio que se ejecutara cuando se reviente una burbuja.

# Astilla

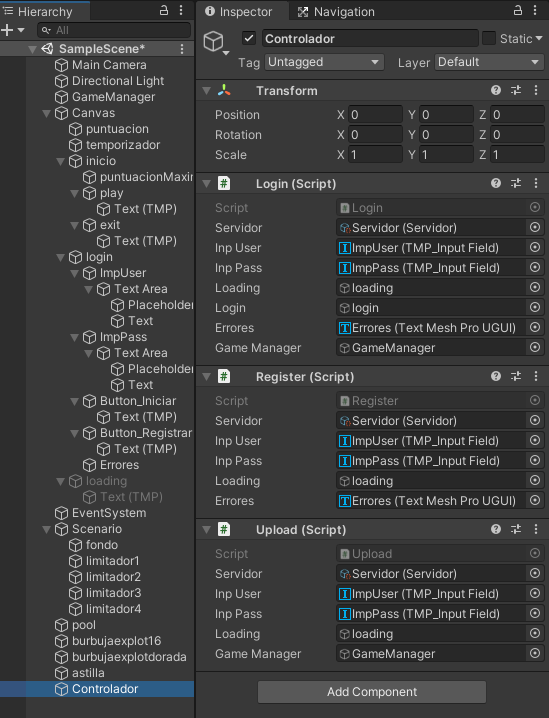


Como se muestra en la pantalla de Game el objeto astilla se encarga de mostrar una astilla como puntero para simular que las burbujas se revientan con una astilla de madera, esta consta de un Sprite Render y un script para que el sprite siga la ubicación a la que señalamos.

Para un mejor entendimiento de cómo están hechos cada script acceda al apartado de Scripts.

El "Sprite Renderer" en Unity es un componente que se utiliza para renderizar imágenes o sprites 2D en el juego. Es responsable de mostrar los gráficos visuales de un objeto en la escena y determinar cómo se verá ese objeto en la pantalla.

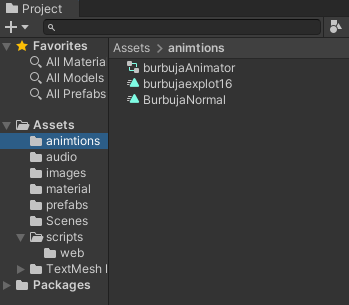
# Controlador



El objeto controlador se encarga del apartado de red, este contiene los scripts encargados del logeo registro y actualización del jugador en el servidor.

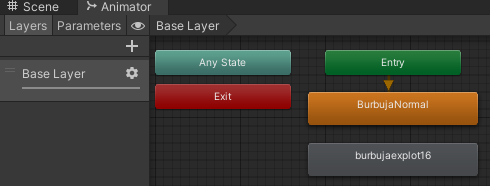
Para un mejor entendimiento de cómo están hechos cada script acceda al apartado de Scripts.

# Assets



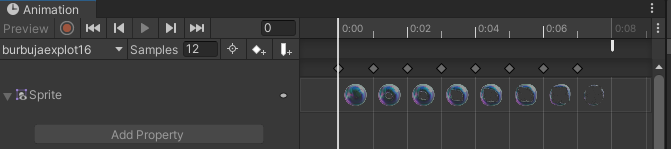
Los "Assets" en Unity son los recursos utilizados en el desarrollo de juegos o aplicaciones, como modelos 3D, texturas, materiales, scripts, audio, animaciones y más.

**Animations**:

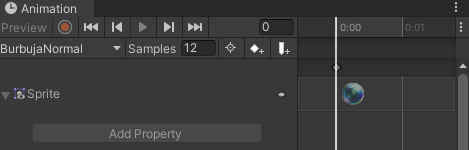


BurbujaAnimator es el encargado de controlar y gestionar las animaciones.

Animacion burbujaexplot16:

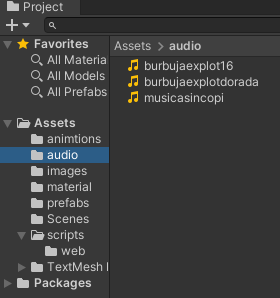


Animacion BurbujaNormal:



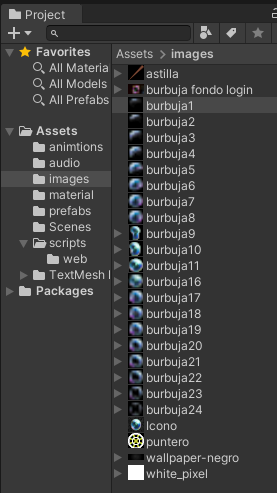
Las animaciones se generan con una colección de imágenes puestas en un orden específico.

**Audio**:



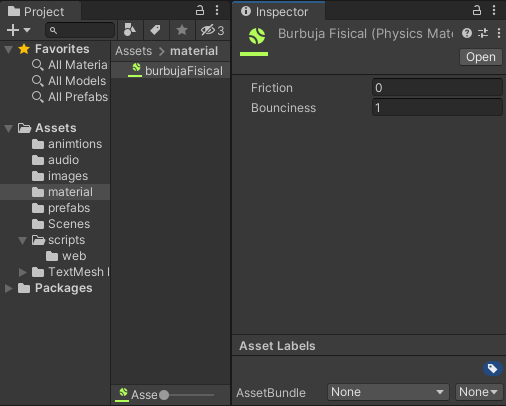
Aquí se guardan los archivos de audio.

**Images**:



Aquí se guardan las imágenes y sprites.

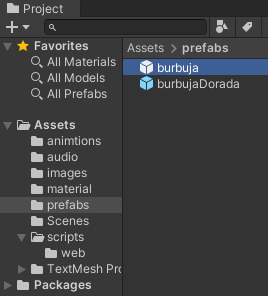
**Material**:



Aquí se guardan los materiales y materiales de fisicas.

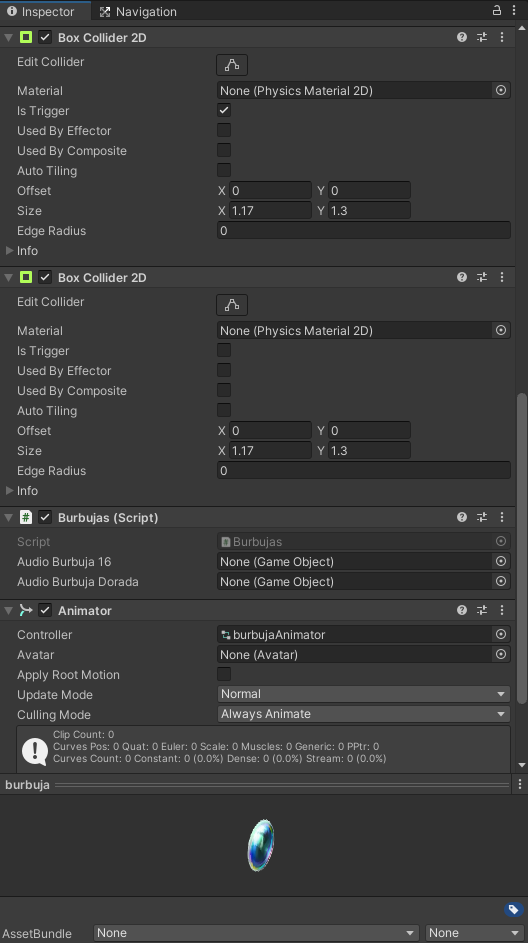
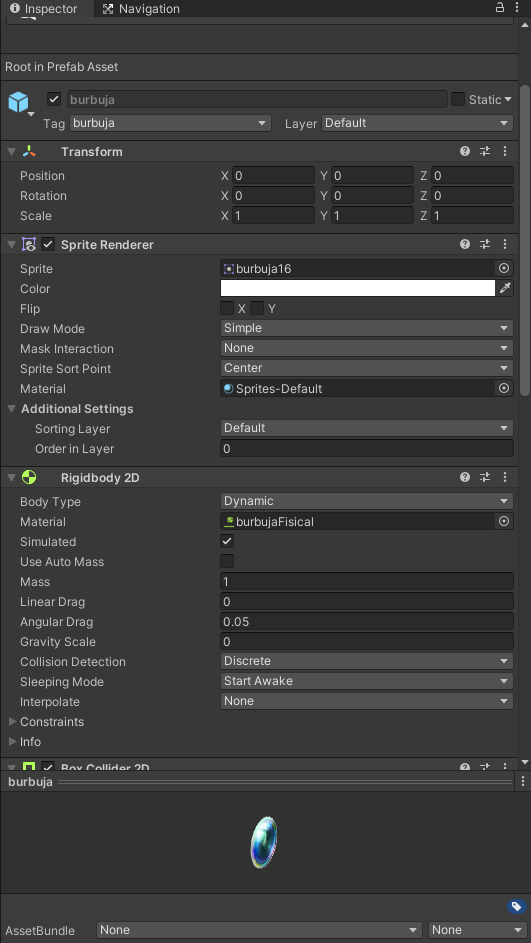
El “Physical Material” en Unity es un tipo de material utilizado para simular la apariencia y el comportamiento físico de los objetos en el juego, en este caso se ponemos que no tenga fricción y que rebote.

**Prefabs**:



Aquí se guardan los prefabs de los objetos.

Un "Prefab" (abreviatura de "prefabricado") en Unity es un objeto o conjunto de objetos predefinidos y reutilizables que se pueden instanciar fácilmente en el juego. Básicamente, un Prefab es una plantilla que contiene toda la información y configuraciones necesarias para crear instancias de un objeto específico en el juego.



El prefab de burbuja tiene un sprite render el cual contiene un sprite de la burbuja que se ve como en la parte inferior. La burbujaDorada tiene su propio sprite pero el color se cambia a amarillo.

Tienen un Rigidbody 2D con la gravity Scale en cero para que no se caigan al vacío y luego se le pueda aplicar una fuerza atreves del script.

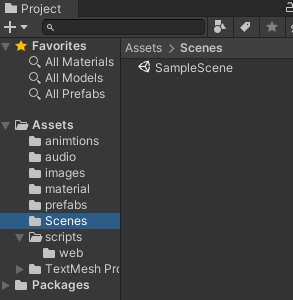
Los prefab tiene dos colisionadores: un colisionador normal (Collider) para colisiones físicas con otros objetos en el juego y un colisionador trigger (Trigger) para desencadenar eventos o acciones sin generar una respuesta física de colisión. Estos colisionadores permiten interactuar con el ratón para que pueda ser pulsado y que puedan colisionar con los limitadores y las otras burbujas.

El script Burbujas gestiona lo relacionado con la burbuja tales como la vida, sonidos, movimiento, etc.

Para un mejor entendimiento de cómo están hechos cada script acceda al apartado de Scripts.

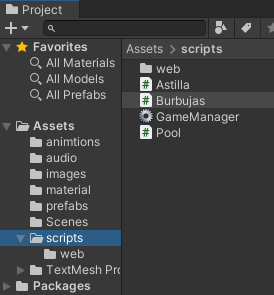
También tiene un Animator para manejar las animaciones de las burbujas.

**Scenes**:

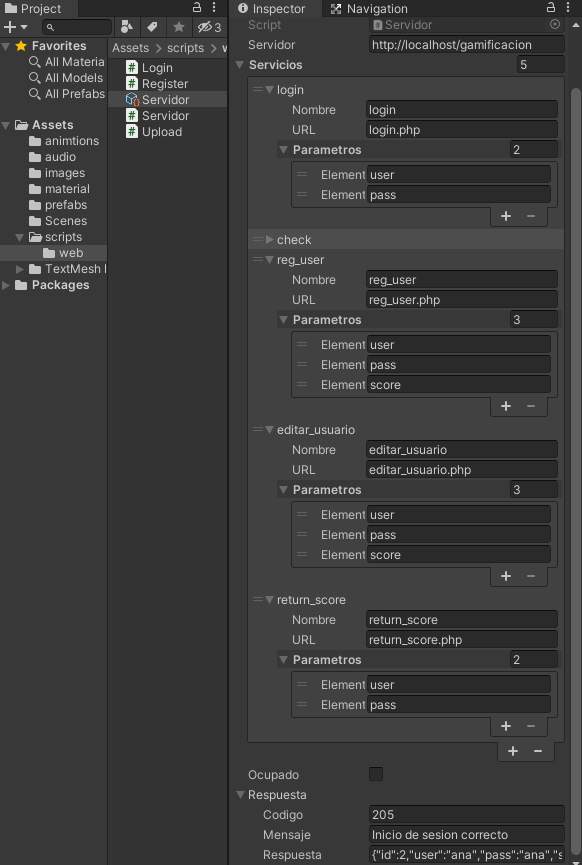


Aquí se guardan las escenas que se usaran en el proyecto, en este caso estamos trabajando sobre una solamente.

**Scripts**:



Aquí se guardan los scripts y los ScriptableObject.



Un “ScriptableObject” permite crear objetos personalizados con propiedades y comportamientos específicos. Puedes definir tus propias clases heredando de ScriptableObject y agregarles variables, métodos y funcionalidades personalizadas.

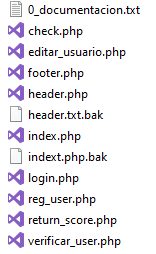
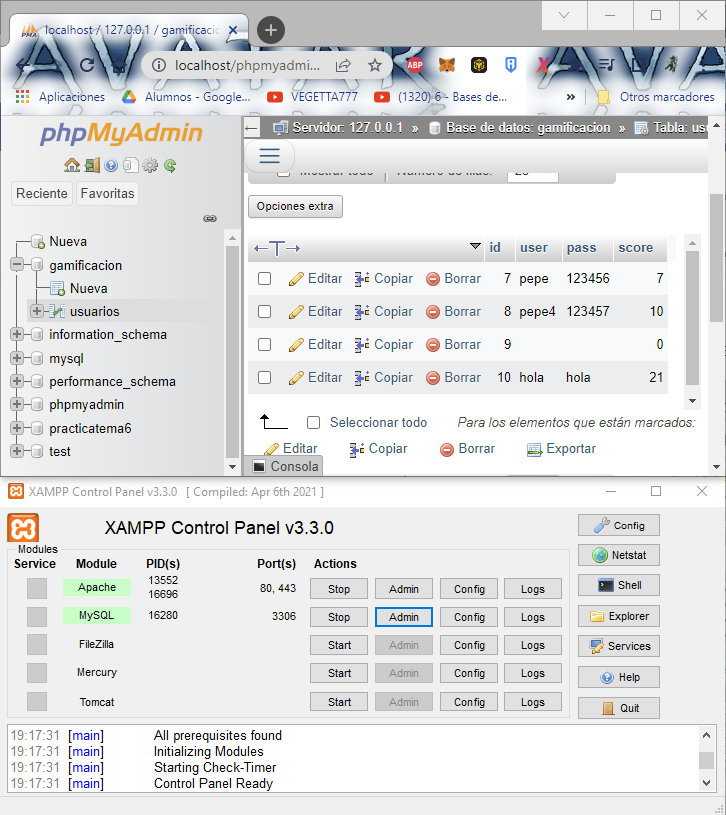
Este ScriptableObject surge del script Servidor y como se puede observar contiene las rutas del servidor de apache y los diferentes php a los que tiene que acceder con sus parámetros correspondientes mostrándote también las respuestas que devuelven estos.

Para un mejor entendimiento de cómo están hechos cada script acceda al apartado de Scripts.

También puedes echar un vistazo al servidor en el apartado Internet.

Internet

Para la conexión a internet la he realizado en MySQL y Apache atraves de XAMPP.





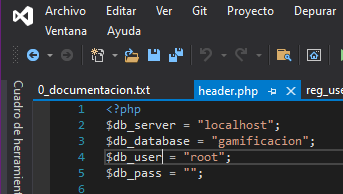
En la imagen superior se puede ver la base de datos gamificacion con la tabla de los usuarios y la carpeta gamificacion de apache donde se encuentran los archivos php que se usan para el login, registro, y modificación de usuario. Explicamos los archivos utilizados a continuación.

# 0\_documentacion.txt

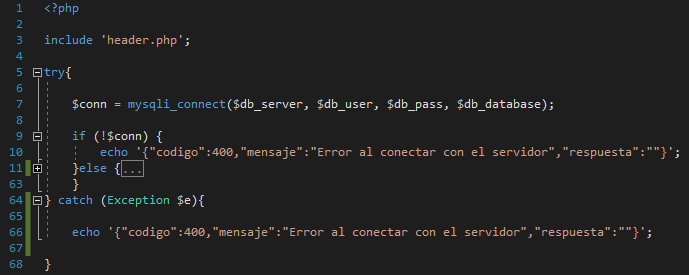


Aquí se encuentran los códigos de respuesta y el uso de las funcionalidades.

# Header.php

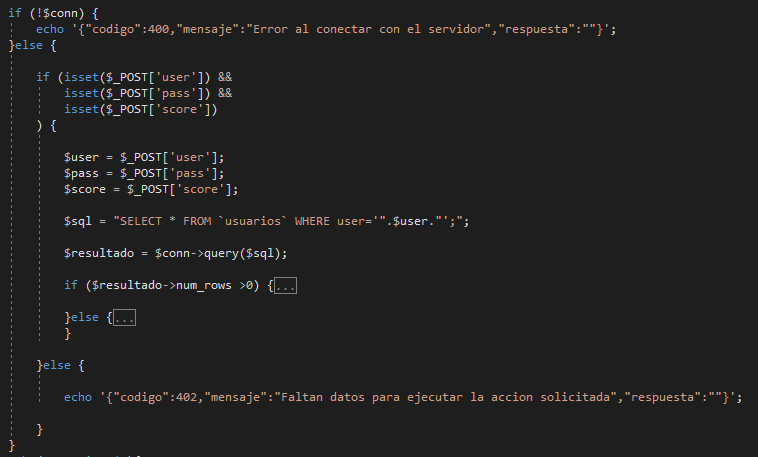


El header contiene los datos necesarios para conectarse con el servido SQL.



El resto de archivo empiezan de la misma manera, añadiendo lo de la cabecera y comprobando que la conexión a la base de datos funciona.

# Reg\_user.php



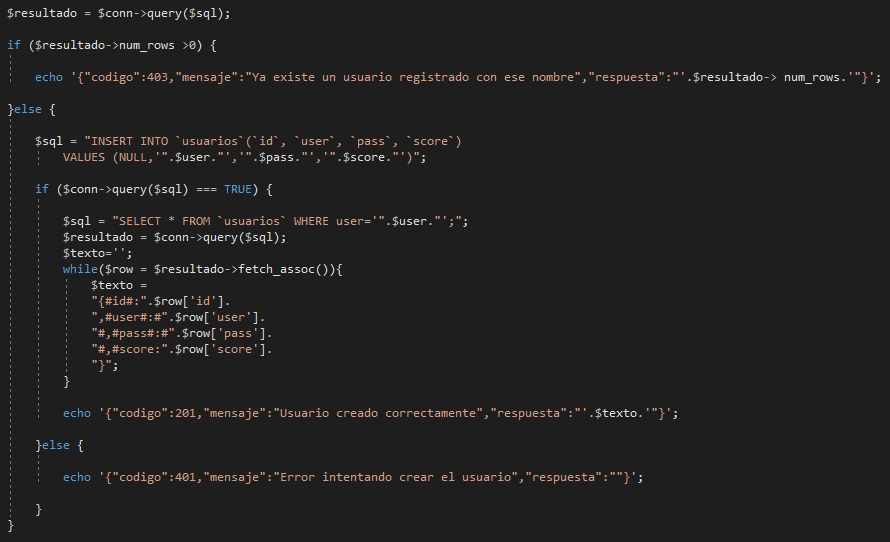
Este archivo se encarga del registro del usuario.

Verificamos si se han enviado tres variables específicas ('user', 'pass' y 'score') a través de una solicitud POST.

La solicitud POST es una forma de enviar datos desde un formulario HTML u otra fuente al servidor web.

En caso negativo muestra el error correspondiente.

En caso afirmativo los guardamos en variables y creamos una sentencia SQL para buscar el usuario que queremos registrar y guardamos el resultado de la consulta.



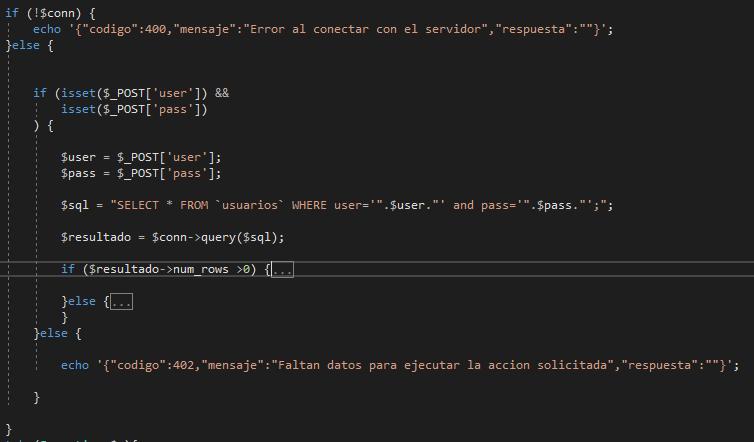
Si el número de tuplas devueltas es superior a 0 mostramos el error correspondiente puesto que nos interesa que no exista el usuario para crearlo.

Si el número de tuplas devueltas es 0 registramos al usuario con su insert correspondiente.

Luego realizamos la conexión y si esta se realiza con éxito guardamos el resultado en un texto formateado específicamente para mostrarlo en la respuesta que usaremos luego en unity.

Si la conexión no se realiza con éxito mostramos el error correspondiente.

# Login.php



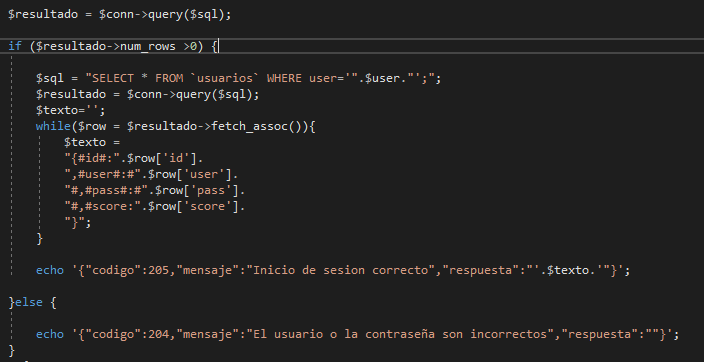
Este archivo se encarga del logeo del usuario.

Verificamos si se han enviado dos variables específicas ('user', 'pass') a través de una solicitud POST.

La solicitud POST es una forma de enviar datos desde un formulario HTML u otra fuente al servidor web.

En caso negativo muestra el error correspondiente.

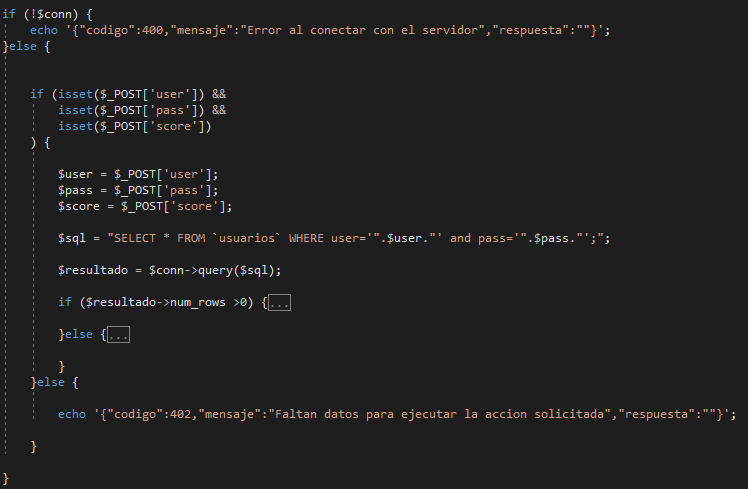
En caso afirmativo los guardamos en variables y creamos una sentencia SQL para buscar el usuario y guardamos el resultado de la consulta.



Si el número de tuplas devueltas es superior a 0 creamos una sentencia SQL para buscar el usuario y guardamos el resultado en un texto formateado específicamente para mostrarlo en la respuesta que usaremos luego en unity.

Si el número de tuplas devueltas es 0 mostramos el error correspondiente.

# Edit\_usuario.php



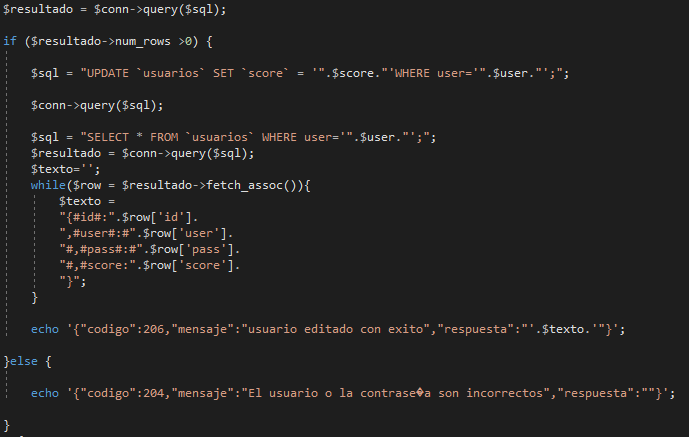
Este archivo se encarga de modificar el usuario para guardar la puntuación.

Verificamos si se han enviado tres variables específicas ('user', 'pass' y 'score') a través de una solicitud POST.

La solicitud POST es una forma de enviar datos desde un formulario HTML u otra fuente al servidor web.

En caso negativo muestra el error correspondiente.

En caso afirmativo los guardamos en variables y creamos una sentencia SQL para buscar el usuario que queremos modificar y guardamos el resultado de la consulta.

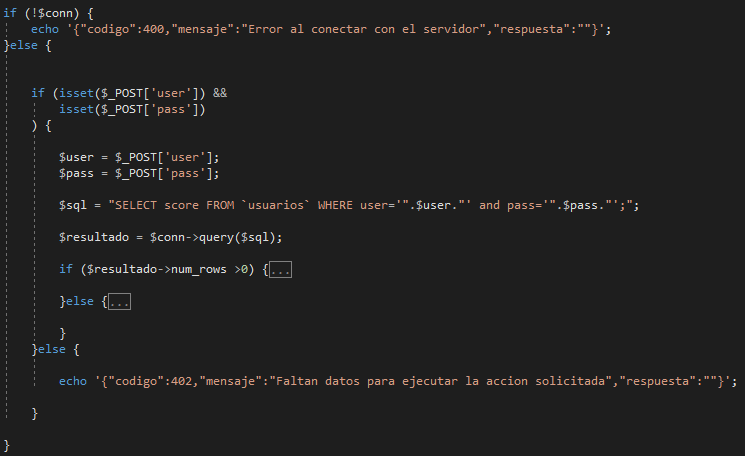


Si el número de tuplas devueltas es superior a 0 actualizamos al usuario con su update correspondiente.

Luego realizamos la conexión y creamos una sentencia SQL para buscar el usuario y guardamos el resultado en un texto formateado específicamente para mostrarlo en la respuesta que usaremos luego en unity.

Si el número de tuplas devueltas es 0 mostramos el error correspondiente.

# Return\_score.php



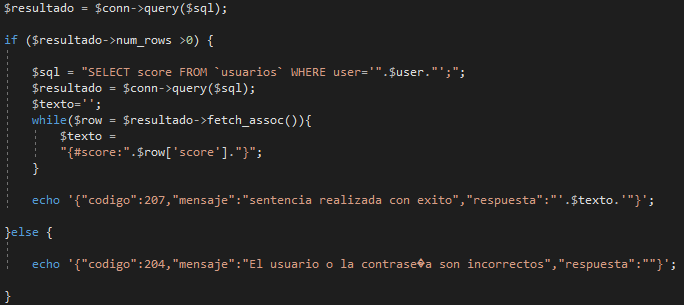
Este archivo se encarga de devolver la puntuación del usuario.

Verificamos si se han enviado dos variables específicas ('user', 'pass') a través de una solicitud POST.

La solicitud POST es una forma de enviar datos desde un formulario HTML u otra fuente al servidor web.

En caso negativo muestra el error correspondiente.

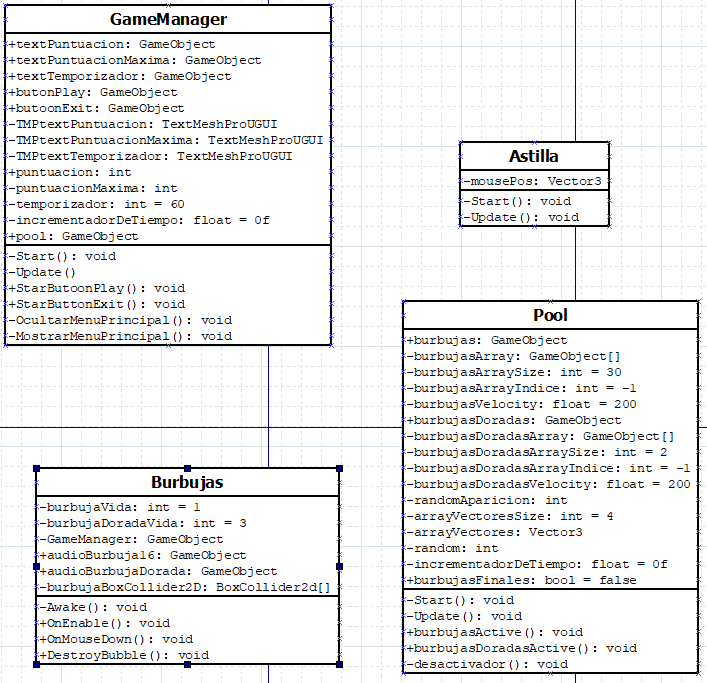
En caso afirmativo los guardamos en variables y creamos una sentencia SQL para buscar el usuario y guardamos el resultado de la consulta.



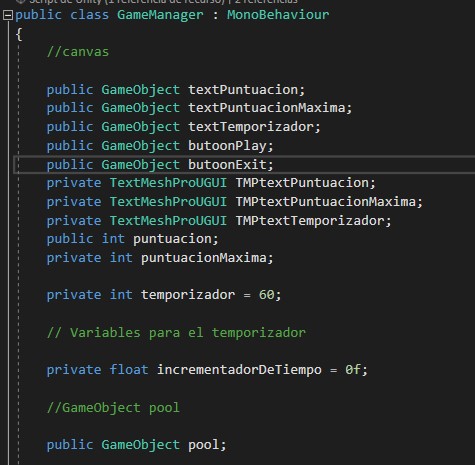
Si el número de tuplas devueltas es superior a 0 creamos una sentencia SQL para buscar el usuario y guardamos el resultado en un texto formateado específicamente para mostrarlo en la respuesta que usaremos luego en unity.

Si el número de tuplas devueltas es 0 mostramos el error correspondiente.

Scripts



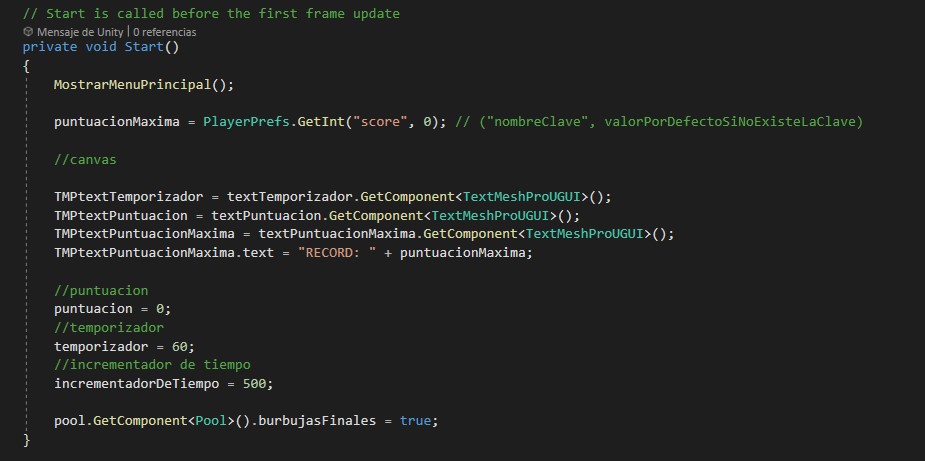
# GameManager



El GameManager se ocupa de la gestión general de la aplicación.

En este Script se maneja el funcionamiento del canvas que es el encargado del menú de inicio, la puntuación del jugador, el temporizador, menú de fin, etc.…, prácticamente se ocupa de la interfaz del juego.

En el GameManager también nos ocuparemos de eliminar todas las burbujas restantes al finalizar la partida.

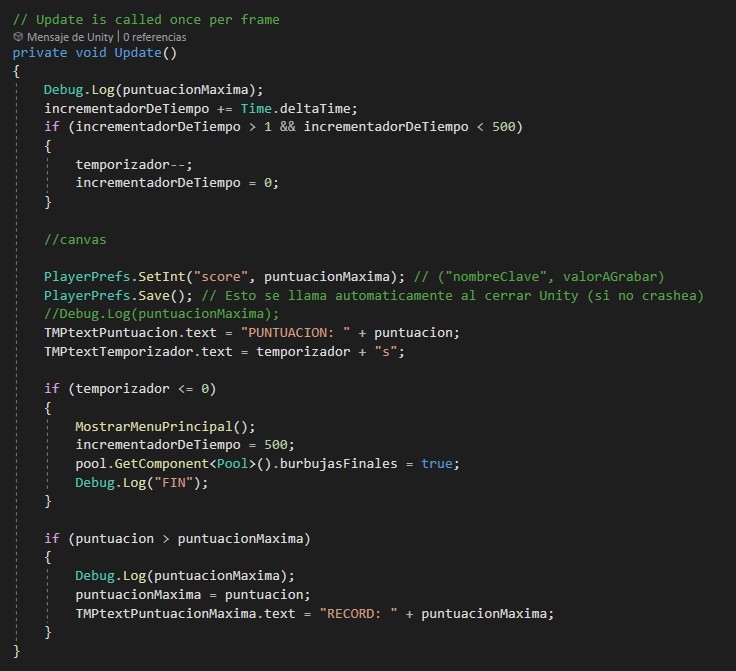


**private void Start(){}**

El método Start() es un método exclusivo de unity que se ejecuta al iniciar la partida.

En este método inicializamos los temporizadores, le asignaremos valor a las variables del canvas para poder mostrarlas más adelanté o modificarlas y llamaremos al atributo burbujasFinales que se encuentra en el componente Pool(que es el Script) del GameObject que se asigna en la variable pool de nuestro código y le asignaremos el valor true para asegurarnos de que no hay ninguna burbuja en pantalla al empezar el juego.

También se llamará al método MostrarMenuPrincipal() para mostrar el menú principal.



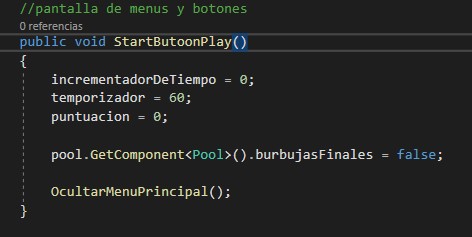
**private void Update(){}**

El método Update() es un método exclusivo de unity que se ejecuta cada frame.

En este método se realizará el control del temporizador, este temporizador es el tiempo que tienes en cada partida y se ira mostrando en pantalla.

También controla que cuando se te acabe el tiempo se guarde la puntuación si esta supera a tu puntuación máxima y te muestra el menú de fin.

Una vez termine el temporizador también llama al atributo burbujasFinales y le asigna el valor de true para que se eliminen las burbujas restantes de la partida.



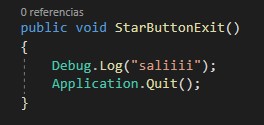
**public void StartButoonPlay()**

El método StartButoonPlay() se ejecutará cuando en el menú de inicio se le dé a PLAY.

Este método inicializara los valores del temporizador y puntuación adecuadamente para que comience la partida con todo correctamente.

También pone el atributo burbujasFinales en false para que empiecen a aparecer las burbujas y las puedas reventar.

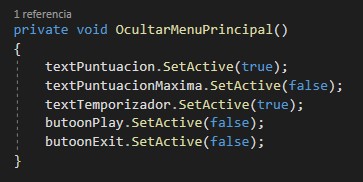
También se ocultará el menú llamando al método OcultarMenuPrincipal().



**public void StarButtonExit()**

El método StarButtonExit () se ejecutará cuando en el menú de inicio se le dé a EXIT.

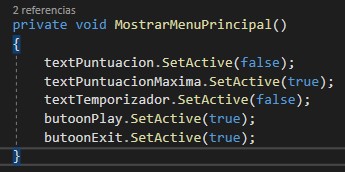
Este método lo único que hace es cerrar el programa.



**Private void OcultarMenuPrincipal()**

El método OcultarMenuPrincipal() oculta o muestra componentes del canvas.

Solo se mostrará la puntuación actual y el temporizador par el jugador pueda verlo mientras realiza la partida.



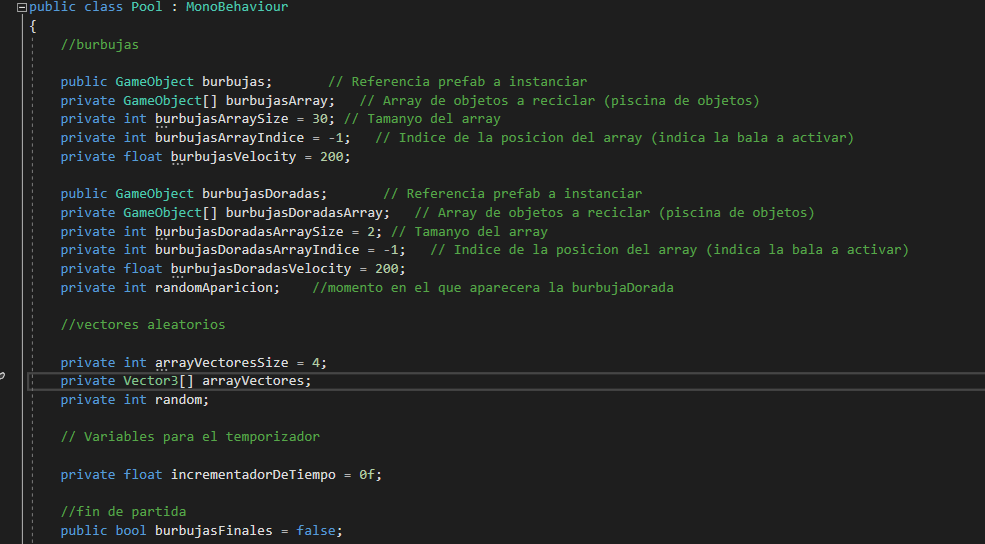
**Private void MostrarMenuPrincipal()**

El método MostrarMenuPrincipal () oculta o muestra componentes del canvas.

Solo se mostrará la puntuación máxima y los botones de PLAY y EXIT para que el jugador pueda verlo al finalizar la partida y jugar de nuevo para superar su máxima puntuación.

También se mostrará al abrir el juego.

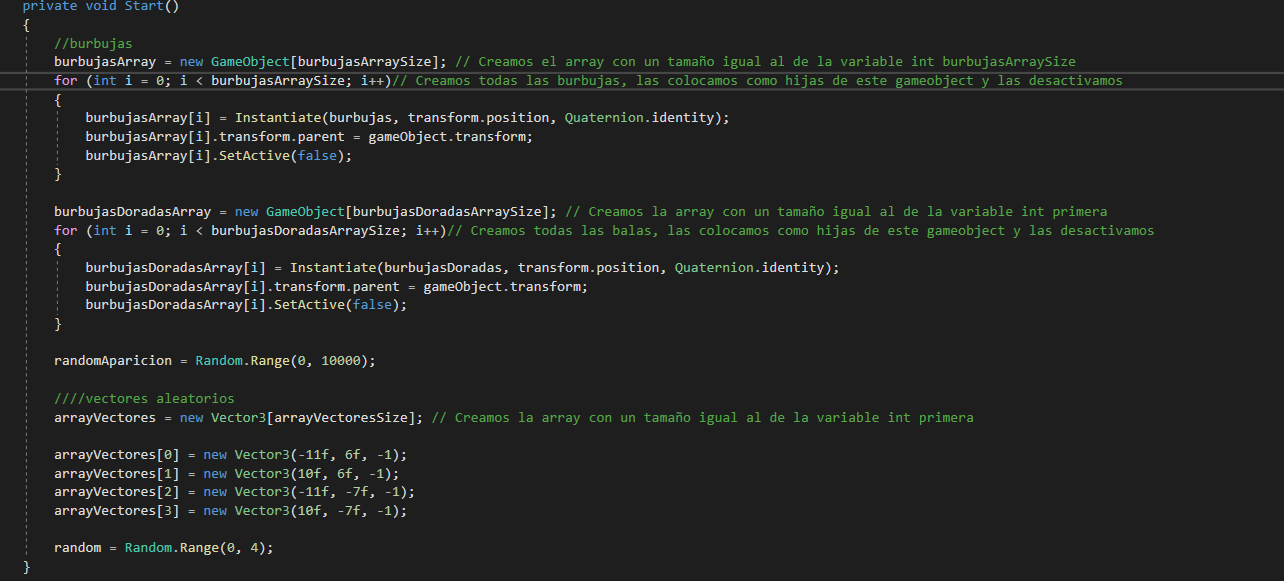
# Pool



El script Pool.cs se encarga de gestionar las apariciones de las burbujas.

Las piscinas de objetos o pools se encargan de generar prefabs de un objeto con ciertas características los cuales estos prefabs se generan un límite de veces para que no vaguen por el mundo infinitamente y así poder reutilizarlos una vez ya no sean necesario ahorrando espacio en memoria y carga de recursos innecesarios.

En este script se generarán las burbujas en 4 puntos distintos del mapa de forma aleatoria y con un número distinto de burbujas normales y doradas.



**private void Start(){}**

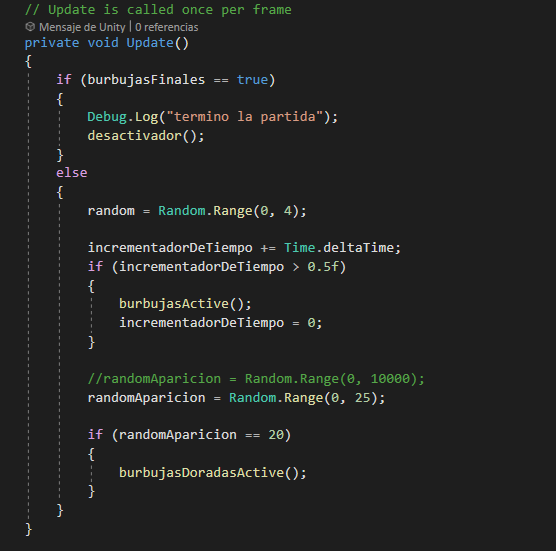
El método Start() es un método exclusivo de unity que se ejecuta al iniciar la partida.

En este método se rellenan los arrays de los dos tipos de burbujas y un tercer array que contiene los 4 vectores que serán los 4 puntos de aparición de las burbujas.

En los arrays de las burbujas se instancian el prefab de la burbuja en la posición del objeto que contiene este script y se desactiva puesto que todavía no se le ha dado a jugar.

RamdomAparicion se le da un valor inicial entre 0 y 10000. Esto servirá para que sea difícil que aparezcan burbujas doradas puesto de dan más puntos.

Ramdon se le da un valor inicial entre 0 y 4. Esto servirá para escoger el punto de aparición de la burbuja.



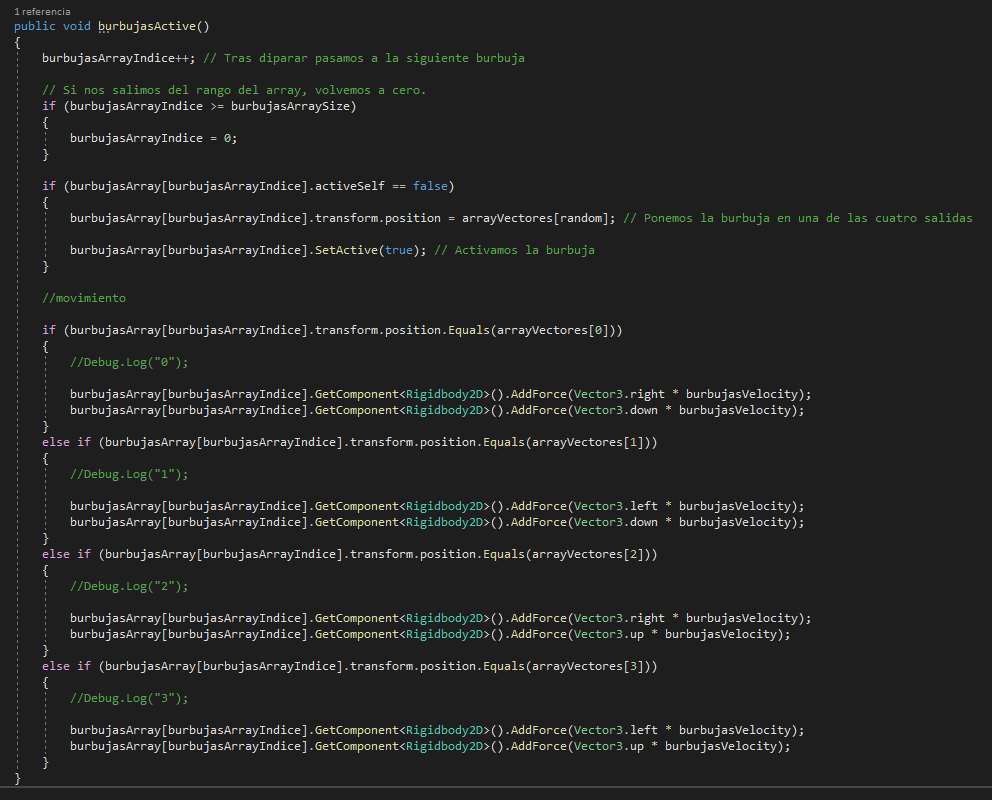
**private void Update(){}**

El método Update() es un método exclusivo de unity que se ejecuta cada frame.

En este método se controla la aparicion de las burbujas.

Si la variable burbujasFinales es verdadera de llamara al método desactivador() que se explicara más abajo.

Si la variable burbujasFinales es falso empezaran a activarse las burbujas normales cada 0.5 segundos para que no salgan muy de seguido llamando al método burbujasActive() y las burbujas doradas saldrán cada vez que la variable randomAparicion sea 20 llamando al método burbujasDoradasActive().



**public void burbujasActive(){}**

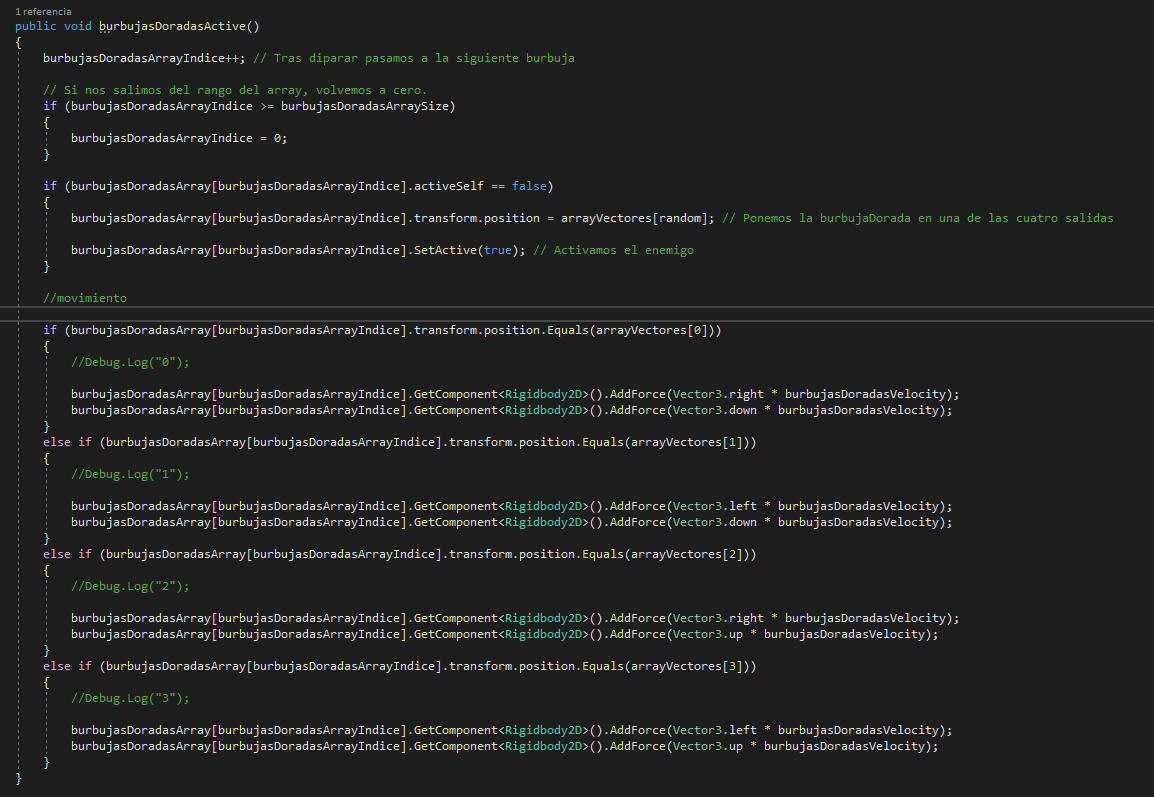
El método burbujasActive() activa las burbujas y las impulsa para que se muevan y así poder empezar a reventarlas.

Lo primero que se hace es sumar uno al índice del array para que cada vez que se llame a este método se apliquen los cambios a la siguiente burbuja.

Comprobamos que si el índice supera el tamaño del array este valga 0 para que vuelva a empezar de nuevo.

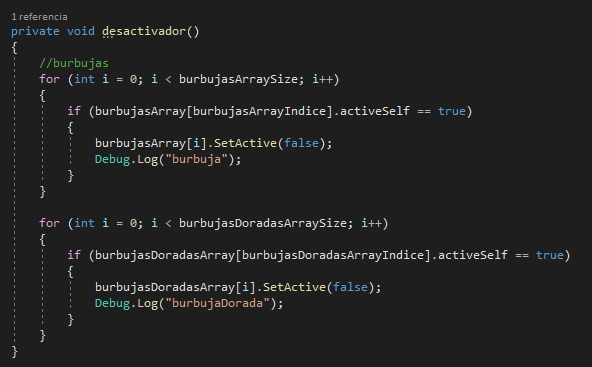
Luego comprobamos si la burbuja esta activada actualmente y si no lo está le damos una posición de los 4 puntos de aparición random y la activamos.

A continuación, según la posición en la que haya salido le aplicamos una fuerza al Rigidbody del objeto en una dirección que apunta al centro de la pantalla.



**public void burbujasDoradasActive(){}**

El método burbujasDoradasActive() hace exactamente lo mismo que burbujasActive() pero es necesario crear este método puesto que estas burbujas se generan en momentos distintos que las otras.



**Private void desactivador(){}**

El método desactivador() se encarga de desactivar todas las burbujas que hay activadas para que no se puedan seguir reventando las burbujas.

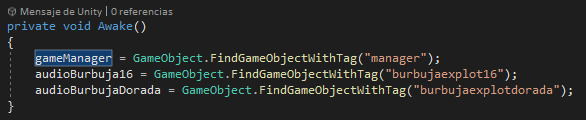
A este método se le llamara al finalizar la partida.

# Burbujas



El script Burbujas.cs se encarga de administrar lo que les ocurre a las burbujas.

Se les da un atributo de vida, sonido y colisión, también llamamos al gameManager para controlar la puntuación.



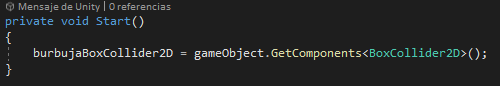
**private void Awake (){}**

El método Awake () es un método exclusivo de unity que se ejecuta antes del Start()

Y sirve para inicializar elementos de los que dependa la inicialización de otros objetos.

En este caso lo usamos para inicializar los objetos que necesitaremos buscándolo mediante su tag

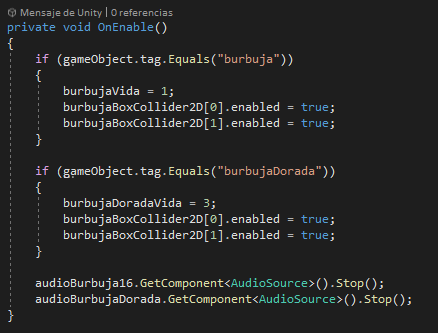
Usando una función predeterminada de GameObject.



**private void Start(){}**

El método Start() es un método exclusivo de unity que se ejecuta al iniciar la partida.

En este método le indicamos el componente de BoxCollider2D del objeto para poder utilizarlo.



**private void OnEnable (){}**

El método OnEnable () es un método exclusivo de unity que se ejecuta cuando un objeto es activado.

En este método lo que hacemos es que a cada burbuja que se activaran al aparecer en el juego se le asigna la vida y se les activan los BoxCollider2D.

Cada burbuja tiene dos BoxCollider2D porque una sirve para que estas puedan chocar con otros objetos y el otro BoxCollider2D es de tipo Trigger que sirve para que se pueda interactuar con él y así ejecutar eventos como el que se pueda pulsar con el ratón o con los dedos lo cual usaremos para reventarlo.

También se para la música que tenga el objeto para que no suene nada al aparecer la burbuja.



**private void OnMouseDown (){}**

El método OnMouseDown () es un método exclusivo de unity que se ejecuta cuando se pulsa el botón izquierdo del ratón o se pulsa sobre el objeto.

Al realizar esta acción lo primero que comprobamos es que tipo de burbuja emos pulsado y se le resta 1 punto de vida.

Luego de restarle la vida comprobamos si esta se pone a cero en cullo caso realizamos las siguientes acciones.

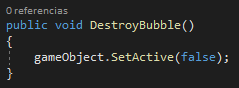
Activamos el componente AudioSource encargado del sonido de la explosión.

Iniciamos la animación de la explosión.

Desactivamos los dos BoxCollider2D para que dejen de chocarse y no se pueda seguir interactuando con ellos.

Del componente Rigigbody2D le ponemos una velocidad a cero para que se quede parada la burbuja en el sitio donde fue la explosión y así no se mueva de sitio mientras se muestra la animación de explosión.

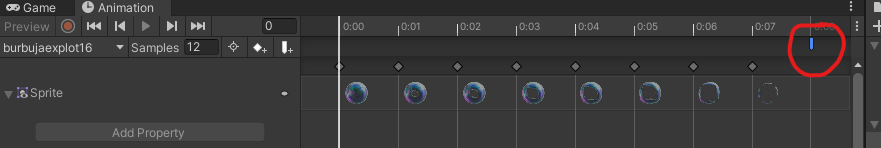
Por último, a la variable puntuación del GameManager se le suman los puntos correspondientes por reventar la burbuja.

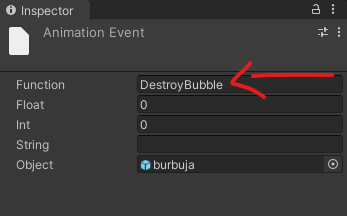


**Public void DestroyBubble (){}**

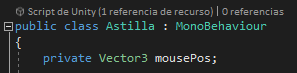
El método DestroyBubble () de lo único que se encarga es de desactivar el objeto.

Este método es usado por el Animation atreves de un evento para que se desactiven una vez terminen la animación.

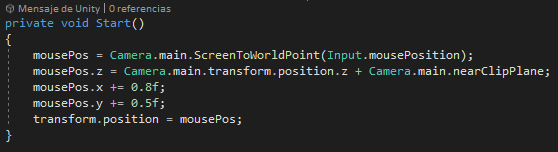




# Astilla



El script Astilla.cs se encarga de la hacer que el sprite de la astilla siga el puntero de manera que parezca que las explotamos con este.



**private void Start(){}**

El método Start() es un método exclusivo de unity que se ejecuta al iniciar la partida.

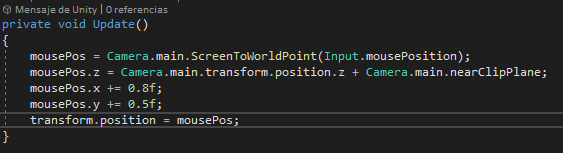
La variable mousePos le ponemos como vector3 el sitio en el que se encuentra el puntero.

Luego en el eje z que es el de profundidad utilizamos un método para que este objeto este siempre por encima del resto para que se vea la astilla en todo momento.

A los ejes X e Y se le suma cierta cantidad a la posición para que la punta de la astilla coincida con el punto exacto en el que se pulsa y así explotar las burbujas con la punta de la astilla.

Por último, se le asigna este vector3 al objeto que lleve este script que es la astilla.

Esto se pone en el Start para que la posición inicial sea el sitio en donde estemos apuntando.

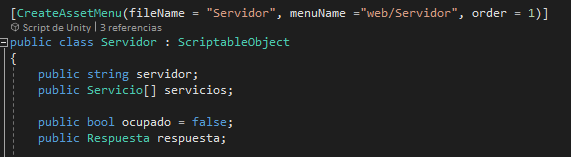


**private void Update(){}**

El método Update() es un método exclusivo de unity que se ejecuta cada frame.

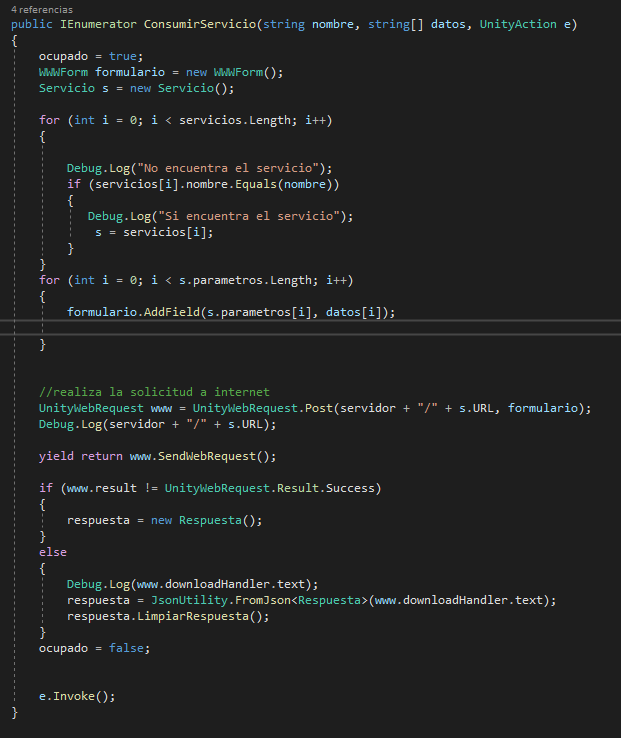
En este método realizamos lo mismo que en el Start para que la astilla este siempre en la posición donde apuntamos y así es como hacemos que la astilla nos siga.

# Web/Servidor



El script Servidor.cs se encarga de administrar la conexión y servicios con la base de datos.

Esta clase es un ScriptableObject que sirve para crear un objeto en unity que puedan usar los diseñadores sin tener que programar nada.



**public IEnumerator ConsumirServicio(string nombre, string[] datos, UnityAction e) {}**

El método ConsumirServicio() es un método que se ejecuta para realizar un servicio de apache

Logeo, registro, actualizar, etc.

El método es un IEnumerator lo que significa que este método se ejecutara a la misma vez que se ejecutan otros procesos. Es lo mismo que un hilo en java.

En este método indicamos con el atributó ocupado que se está ejecutando un servicio, creamos un formulario para registrar los parámetros que necesita cada servicio y un servicio para acceder y controlar los servicios del servidor.

Buscamos si el servicio que se busca por los parámetros de la función existe a través de un bucle y luego a través de otro guardamos los datos necesarios en el formulario.

Una vez tenemos el servicio configurado realizamos la conexión al servidor junto al formulario.

Luego comprobamos si la conexión es correcta guardando los errores en una respuesta.

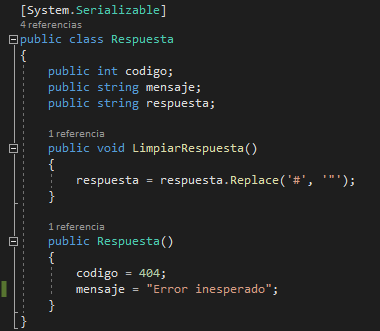
Al finalizar invocamos el siguiente método a ejecutar que nos mandaron por parámetros.



**public class Servicio {}**

La clase Servicio de tipo Serializable sirve para guardar los servicios que se ejecutaran los cuales tienen que coincidir con los del servidor al igual que los parámetros necesarios.

A la hora de crear el objeto ScriptableObject del Servidor se podrán rellenas estos atributos.



**public class Respuesta{}**

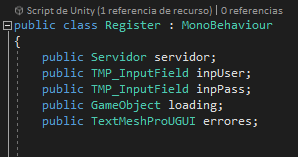
La clase Respuesta de tipo Serializable sirve para guardar las respuesta que serán los mensajes que se ejecutaran los cuales tienen que coincidir con los del servidor.

Esta clase tiene el método LimpiarRespuesta() que reemplaza caracteres del php para que aquí se puedan leer correctamente y poder acceder a los datos de forma eficiente.

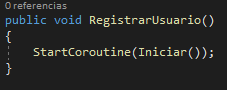
El método Respuesta(){} es un constructor por defecto para el caso de que no se sepa por que falla la conexión.

En los php vienen todos los tipos de errores que pueden surgir.

# Web/Register

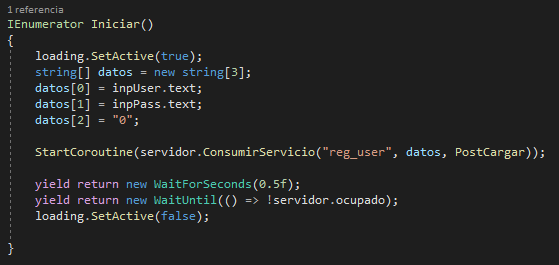


El script Register.cs se encarga de mandar los datos necesarios para realizar el registro del usuario.



**public void RegistrarUsuario() {}**

El método **RegistrarUsuario** () iniciara una corrutina que es una función específica de unity que puede ejecutar tareas en segundo plano o en intervalos de tiempo específicos.



**IEnumerator Iniciar() {}**

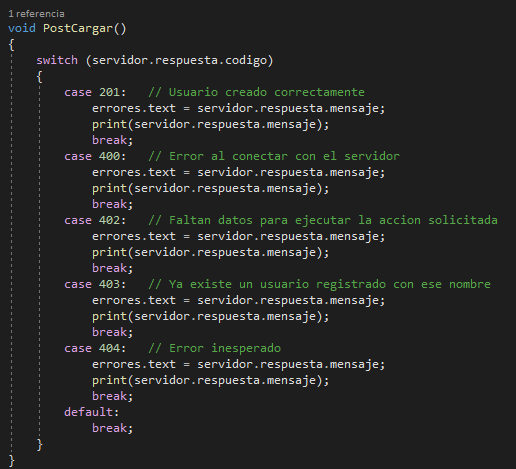
El método es un IEnumerator lo que significa que este método se ejecutara a la misma vez que se ejecutan otros procesos. Es lo mismo que un hilo en java.

El método **Iniciar** () activa una pantalla de carga y guarda los datos necesarios para el registro

Iniciando luego en una corrutina el servicio al cual se le manda el nombre del servicio, los datos que necesita ese servicio y lo siguiente a ejecutar una vez termine de ejecutarse ese método.

Esta corrutina se ejecutará cada 5 segundos o cuando el servidor deje de estar ocupado.

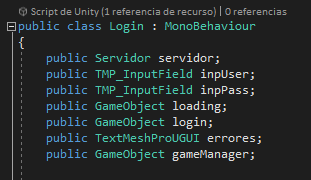
Luego dejara de mostrar la pantalla de carga.



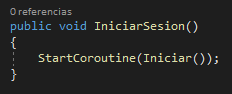
**void PostCargar() {}**

El método **PostCargar**() muestra al usuario los errores correspondientes de lo sucedido según su código de error.

# Web/Login

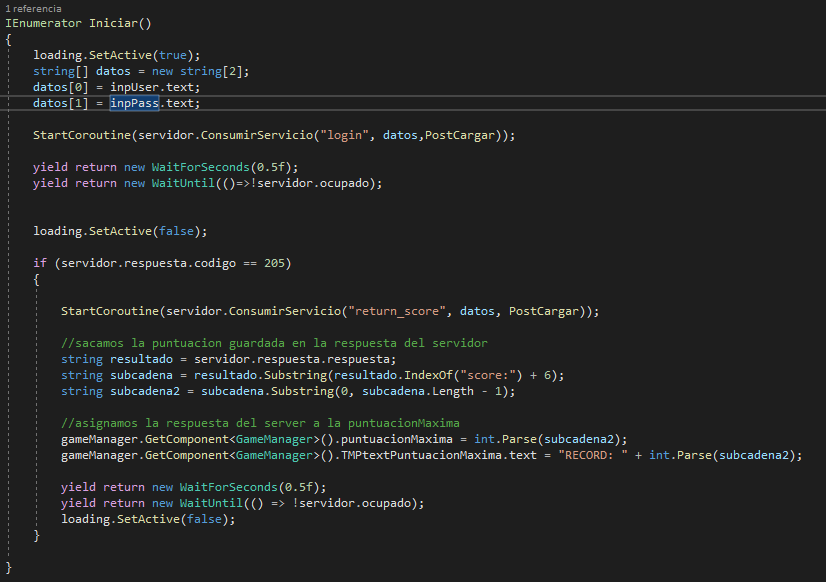


El script Login.cs se encarga de mandar los datos necesarios para realizar el logeo del usuario.



**public void IniciarSesion() {}**

El método **IniciarSesion** () iniciara una corrutina que es una función específica de unity que puede ejecutar tareas en segundo plano o en intervalos de tiempo específicos.



**IEnumerator Iniciar() {}**

El método es un IEnumerator lo que significa que este método se ejecutara a la misma vez que se ejecutan otros procesos. Es lo mismo que un hilo en java.

El método **Iniciar** () activa una pantalla de carga y guarda los datos necesarios para el logeo

Iniciando luego en una corrutina el servicio al cual se le manda el nombre del servicio, los datos que necesita ese servicio y lo siguiente a ejecutar una vez termine de ejecutarse ese método.

Esta corrutina se ejecutará cada 5 segundos o cuando el servidor deje de estar ocupado.

Luego dejara de mostrar la pantalla de carga.

Cuando el logeo sea correcto se ejecutará en una corrutina el servicio al cual se le manda el nombre del servicio, los datos que necesita ese servicio y lo siguiente a ejecutar una vez termine de ejecutarse ese método.

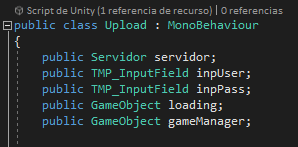
De la respuesta de este servicio sacamos la puntuación guardada en el servidor y se la asignamos a la puntuaciónMáxima del juego para que cada jugador continúe con su propia puntuación.



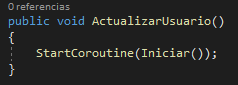
**void PostCargar() {}**

El método **PostCargar**() muestra al usuario los errores correspondientes de lo sucedido según su código de error.

# Web/Upload

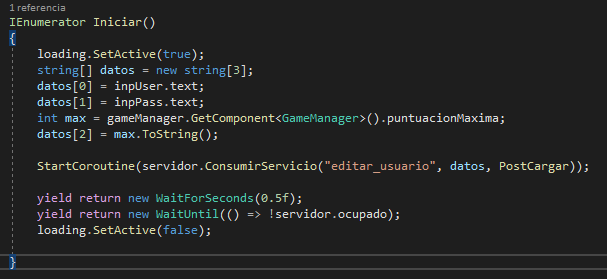


El script Upload.cs se encarga de mandar los datos necesarios para realizar el update del usuario.



**public void ActualizarUsuario() {}**

El método **ActualizarUsuario** () iniciara una corrutina que es una función específica de unity que puede ejecutar tareas en segundo plano o en intervalos de tiempo específicos.



**IEnumerator Iniciar() {}**

El método es un IEnumerator lo que significa que este método se ejecutara a la misma vez que se ejecutan otros procesos. Es lo mismo que un hilo en java.

El método **Iniciar** () activa una pantalla de carga y guarda los datos necesarios para la actualización Iniciando luego en una corrutina el servicio al cual se le manda el nombre del servicio, los datos que necesita ese servicio y lo siguiente a ejecutar una vez termine de ejecutarse ese método.

Esta corrutina se ejecutará cada 5 segundos o cuando el servidor deje de estar ocupado.

Luego dejara de mostrar la pantalla de carga.



**void PostCargar() {}**

El método **PostCargar**() muestra al usuario los errores correspondientes de lo sucedido según su código de error.

VOCABULARIO

Escena (Scene): La escena es el espacio de trabajo principal en Unity. Aquí es donde los desarrolladores construyen y diseñan los niveles, entornos y elementos del juego. Puedes arrastrar y soltar objetos, ajustar sus propiedades y organizar la disposición de los elementos en la escena.

Inspector: El Inspector muestra las propiedades y componentes de un objeto seleccionado en la escena o en la jerarquía (Hierarchy). Permite editar y ajustar los valores de los componentes, como transformaciones, materiales, colisiones, luces, scripts y más.

Jerarquía (Hierarchy): La Jerarquía es una estructura en forma de árbol que muestra todos los objetos presentes en la escena. Permite organizar los objetos en una estructura jerárquica, lo que facilita la manipulación y la relación entre ellos. Los objetos también se pueden activar o desactivar en la jerarquía.

Proyecto (Project): La pestaña Proyecto muestra todos los archivos y recursos que se encuentran en el proyecto de Unity. Aquí es donde importas y gestionas tus activos, como modelos 3D, texturas, scripts, audio y más. Puedes crear carpetas para organizar tus archivos y buscar elementos utilizando filtros y etiquetas.

Consola (Console): La Consola es una ventana que muestra los mensajes, advertencias y errores generados por el proyecto de Unity. Proporciona información útil para depurar y solucionar problemas en el juego y en los scripts. También puedes imprimir mensajes personalizados desde tus scripts a la Consola para realizar un seguimiento del comportamiento del juego.

Animator: El Animator es un editor visual para crear y administrar animaciones en Unity. Permite crear transiciones, establecer eventos, ajustar el tiempo y manipular las propiedades de los animadores de los personajes u objetos.

Animator Controller: El Animator Controller es una ventana que muestra los controladores de animación. Aquí puedes definir cómo se combinan y controlan diferentes animaciones en función de los estados y las transiciones. Puedes establecer condiciones y configuraciones para controlar las animaciones durante el juego.

Transform: El componente Transform define la posición, rotación y escala de un objeto en el espacio 3D. Permite mover, rotar y escalar el objeto en la escena.

Collider: El componente Collider permite que los objetos interactúen entre sí en términos de detección de colisiones. Los colliders definen volúmenes de colisión que se utilizan para detectar si un objeto ha chocado con otros objetos.

Rigidbody: El componente Rigidbody agrega simulación física a un objeto. Permite que el objeto responda a fuerzas externas, gravedad, colisiones y aplicaciones de fuerza y torque.

Script: El componente Script permite adjuntar scripts escritos en lenguajes de programación como C# o UnityScript (JavaScript) a un objeto. Estos scripts contienen código personalizado que define el comportamiento y la lógica del objeto.

Audio Source: El componente Audio Source se utiliza para reproducir sonidos en el juego. Puede emitir música de fondo, efectos de sonido y diálogos, y permite ajustar parámetros como el volumen, la posición y la reproducción en bucle.

Animator: El componente Animator se utiliza para controlar la animación de un objeto. Permite definir y gestionar una serie de animaciones y transiciones entre ellas, basadas en diferentes estados y condiciones.

Camera: El componente Camera define una cámara en la escena que representa la vista del jugador. Controla los parámetros de la cámara, como la posición, la rotación, el campo de visión y los efectos de pos procesamiento.

Light: El componente Light se utiliza para agregar iluminación a la escena. Puede ser una luz direccional, puntual o de área, y permite ajustar el color, la intensidad y otros parámetros de la luz.

