Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías de Telecomunicación

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

APLICACIÓN LÚDICA MULTIJUGADOR EN TIEMPO REAL USANDO REALIDAD AUMENTADA

Alumno: **Vanesa Moreno Pérez**

Tutor: Prof. D. Luis Ramón López López
Depto.: Ingeniería de Telecomunicación

Junio, 2021

Universidad de Jaén

Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías de Telecomunicación

Trabajo de Fin de Grado

Aplicación lúdica multijugador en tiempo real usando realidad aumentada

AGRADECIMIENTOS:

Hay que recordar que el empeño y el esfuerzo es principal para todo, y para seguir consiguiendo metas, esas metas que nunca podrías imaginar que ahora estas consiguiendo, y esto es gracias a los valores que mi familia me han inculcado desde muy pequeña, porque nadie te regala nada si tu no pones de tu parte y sacrificas miles de cosas para ello.

Gracias de corazón a todas las personas que me han apoyado, dado algún que otro consejo y sobre todo los ánimos recibidos por toda mi familia, compañeros y amigos, y por su puesto a mi tutor Luis Ramón López López que me ha apoyado en todo y seguido todo el transcurso de este proyecto para terminar esta etapa.

ÍNDICE

ÍNDICE DE FIGURAS

1. RESUMEN

1.1. Resumen

El objetivo principal de este TFG es la creación de una aplicación de Realidad Aumentada que sirva como distracción en cualquier restaurante o bar, con el fin de crear una experiencia al usuario y a la misma vez estar entretenidos mientras esperan su servicio, algo que al fin a cabo hace más ameno al cliente su espera.

El funcionamiento de esta aplicación se basa en la utilización de un marcador, en este caso será de un código QR. El marcador estará en aquellos establecimientos que soliciten este tipo de tecnología. Una vez que captemos el marcador con una Tablet o un teléfono móvil, nos saltara la aplicación de Realidad Aumentada. Previamente, antes de realizar estos pasos necesitaremos tener descargar nuestra aplicación en el teléfono móvil o Tablet, para ello el establecimiento tendrá otro código QR, que nos llevara directamente a descargar la aplicación.

Esta aplicación está pensada para dispositivos tanto Android como iOS.

1.2. Abstract

The main goal of this TFG is the creation of an Augmented Reality application that serves as a distraction in any restaurant or bar, in order to create an experience for the user and at the same time be entertained while they wait for their service, something that finally out makes waiting more nice for the customer.

The operation of this application is based on the use of a marker, in this case it will be a QR code. The marker will be in those establishments that request this type of technology. Once we capture the marker with a Tablet or a mobile phone, we will skip the Augmented Reality application. Previously, before performing these steps we will need to download our application on the mobile phone or Tablet for this the establishment will have another QR code which will take us directly to download the application.

This application is intended for both Android and iOS devices.

2. INTRODUCCIÓN

A modo de introducción, se define la Realidad Aumentada como una tecnología que nos permite combinar el mundo real con el mundo virtual. De esta forma, la imagen en su conjunto, tiene una mayor cantidad de información que ya tenia con anterioridad. Se consigue poder ver a través de algunos dispositivos tecnológicos (como los teléfonos móviles, tablet, etc…) con el fin de crear experiencias interactivas al usuario.

A través de esta tecnología, podemos incorporar contenido como texto, imagen, audio, figuras en 3D, video, así como información táctil u olfativa.

Cada vez es más usual encontrarse este tipo de tecnología en la vida diaria, incluso en muchas tiendas, a nivel nacional como Amazon, Ikea, entre otros mucho más empresas.

En los siguientes apartados, se detallará su procedencia, así como las primeras definiciones de esta tecnología, hasta la definición más actual. Se verá algunas empresas que hacen uso de esta tecnología. Se hará una comparación entre las plataformas Android y Apple. Además se hará una distinción con la realidad virtual y la realidad aumentada, ya que son dos definiciones que suele confundir a muchas personas. Por último, se detallará el motor de Unity donde se verás las características principal, como es su interfaz y algunos objetos de los más característicos.

2.1. Antecedentes

La Realidad Aumentada viene reiterada de hace mucho tiempo atrás, incluso desde mucho antes de que apareciera en la ciencia ficción por primera vez. En el año 1901 una novela escrita por L.Frank Baum, hacia mención a la Realidad Aumentada. La novela trataba de las aventuras de un niño que con unas gafas podía ver imágenes superpuestas en la frente de las gente. Pero su origen no remonta hasta el año 1950, en el que Morton Heiling escribió sobre un cine de experiencia en el que pretendía incorporar todos los sentidos. Esto no ocurrió hasta que en el año 1961 se saco un prototipo llamado “Sensorama”, donde proyectaba imágenes en 3D junto a un sonido envolvente además de hacer vibrar el asiento y crear viento lanzado al aire al espectador. Todo esto permitía crear una experiencia al espectador. Las imágenes que se veía en el Sensorama producían una sensación de estar montada en bici por las calles de Brooklyn.

Además el descubrimiento de la Realidad Aumentada esta ligado a la Realidad Virtual. Pasado un tiempo, estas dos tecnologías estaban más perfeccionadas y fueron separadas.

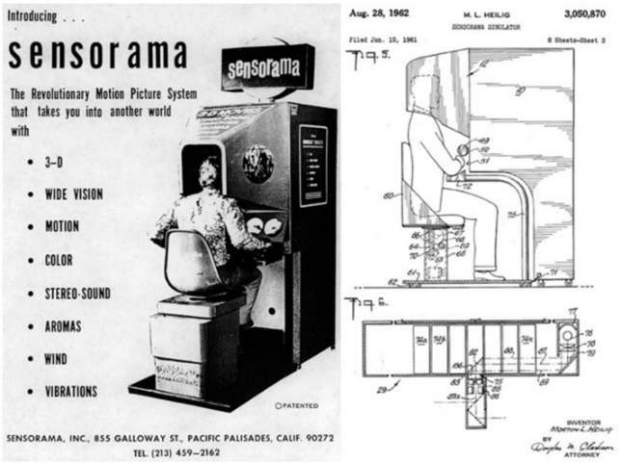


Figura 2.1: Sensorama

Entre el año 1994 y 1997, se dieron a conocer dos definiciones de esta tecnología. La primera definición fue constituida por Paul Milgram y Fumio Kishino en 1994, definiéndola como una escala continua del entorno real al entorno virtual, El área en el que se combina lo real y lo virtual la definieron como Mixed Reality.

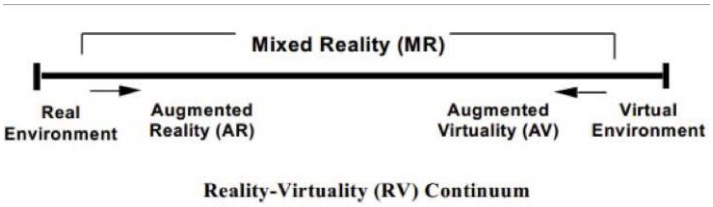


Figura 2.2: Diagrama de Milgram-Virtually

Con respecto a la figura, si se avanza de izquierda a derecha se va aumentando los elementos virtuales que se agregan al entorno real, pero si se avanza de derecha a izquierda aumenta los elementos reales que se agregan al entorno virtual.

La otra definición fue por Ronald Azuma en el año 1997, en lo que exponía que un sistema de Realidad Aumentada es aquel que tiene que cumplir las tres características nombradas a continuación:

- Combinar elementos virtuales y reales
- Interactiva en tiempo real
- Realiza en 3D

Por otro lado, cada vez es más común encontrarnos con este tipo de tecnología en la vida cotidiana, ya sea en comercios, educación, medicina como en ocio. Todas ellas tienen algo en común y es facilitar la gestión al ser humano. Se verá con más detalle aquellos sectores en lo que aparece esta tecnología:

- **Educación:** En este ámbito la RA puede ayudar a incrementar la capacidades del alumno con el simple hecho de interactuar con un dispositivos electrónico que haga el estudio una forma de entretenimiento.
- **Medicina:** Otro punto fuerte en la RA es en la medicina, donde resulta tener una previsión adelantada de los problemas que pueda haber en una intervención quirúrgica o incluso podría superponer al cuerpo de ese paciente unas guías que le indiquen por donde debe de intervenir gracias al uso de imágenes medicas con realidad aumentada, y ayudar al paciente en su rehabilitación.
- **Turismo:** La RA también se utiliza en el turismo como un método complementario que ayuda a mejorar la experiencia del turista, revelando todo tipo de información sobre los elementos del entorno y de realizar interpretaciones más completas de la visita. La Realidad Aumentada se puede utilizar en este sector como una herramienta para la reconstrucción de monumentos.
- **Información:** Muchos medios de comunicación están utilizando esta tecnología para complementar sus noticias y hacer que sea más cercana y atractiva al espectador, como por ejemplo la recreación de la llegada del hombre a la luna en el plato de Antena 3.
- **Videojuegos:** Sin duda alguna el punto fuerte de la Realidad Aumentada es en los videojuegos, como el famoso popular juego Pokemon Go que mediante una localización actual del usuario podía capturar mascotas que iba apareciendo en su pantalla de manera muy atractiva.
- **Moda:** En este sector la Realidad Aumentada no se queda atrás, grandes empresas como son Zara, Nike o Adidas, están probando experiencias diferentes para que el momento de comprar sea más fácil y rápido, con solo la cámara del teléfono móvil el cliente puede ver como le quedaría el vestido o las zapatillas.

2.2. Estado del Arte

A día de hoy, existen muchas técnicas para visualización de esta tecnología, así como diferentes tipos de realidad aumentada. A continuación se detallará las técnicas utilizadas a día de hoy:

- ***Gafas de AR***

  Son un tipo de pantalla que se instaladas en las gafas de realidad aumentada, dado lugar a que se puede ver elementos reales como elementos virtuales.

  Esta técnica también conocida como HDM (head-Mounted Display), utilizan dispositivos ópticos el cual permite que el usuario pueda ver el mundo físico a través de una lente y superponga la información gráfica que se refleja en los ojos del usuario.

Figura 2.3: Gafas de Realidad Aumentada

- ***Display de mano***

  Esta es otra técnica de las más usadas en este entorno, ya que se puede usar con un móvil o una Tablet que cuenta con un dispositivo informático que incorpora una pantalla pequeña que cabe en la mano.

  Se utiliza de igual manera que las gafas de realidad aumentada, ya que emplea técnicas de superposición sobre el video con la información gráfica.

  Tanto en el presente como en el futuro, será la técnica más aplicada por los usuarios, ya que estos tendrán presentes a su día a día un teléfono móvil.

Figura 2.4: Display de mano

- ***Display espacial***

Esta técnica también se conoce como SAR, utiliza proyectores digitales para mostrar información gráfica sobre los objetos físicos, con diferencia de las otras técnicas es que la pantalla esta separada de los usuarios, debido a que el display no está asociado a cada usuario, de forma que varios usuarios lo puedan utilizar a la vez y coordinarse entre ellos. Aquí la realidad aumentada no funciona donde el usuario este mirando, sino donde este enfocando la cámara.

Una ventaja que presenta con respecto a las otras técnicas es que no esta obligado a llevar ningún dispositivo encima de manera que no se agota tanto la vista y no lo daña.

Otra gran ventaja que presenta esta técnica es que no se limita la resolución de la pantalla, por lo que se puede incorporar más proyectores para así ampliar la visualización.

Figura 2.5: Display Espacial

De igual manera que existen diferentes tipos de visualización, también existe diferentes modos de realidad aumentada. Sin duda, la más conocida es mediante el uso de un marcador, cuando la aplicación detecte el marcador saltará su contenido encima, es decir, el marcador aquí actuará como un activador. El marcador puede ser imágenes, QR en 2D.

Figura 2.6: Marcador Específico

Existe una variante de realidad aumentada basada en el reconocimiento, se trata de la realidad aumentada basada en la superposición que reemplaza total o parcialmente, la visión real de un objeto para mostrar una visión aumentada, en este tipo de realidad aumentada lo más importante es la detección, ya que sin determinar de que tipo de objeto se trata seria imposible mostrar una visión aumentada de esta. Aquí se usa objetos reales en lugar de un marcador específico.

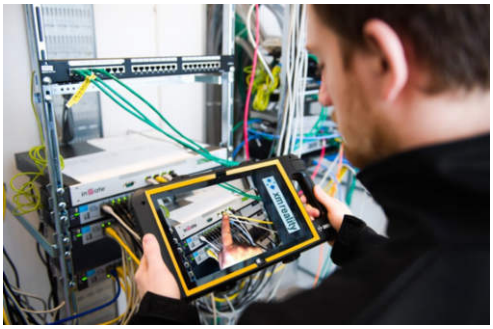


Figura 2.7: Marcador real

También se puede hacer uso de la realidad aumentada basada en la ubicación, este tipo de realidad aumentada lo realiza sin marcadores. Este tipo de realidad aumentada es más común utilizarla en exteriores, aquí el que actuará como activador será la ubicación del usuario en el mapa.

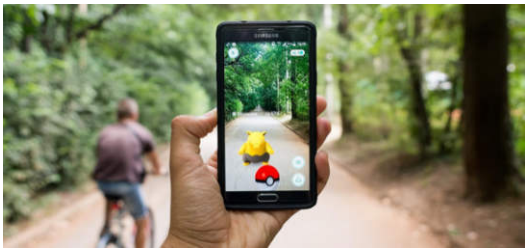


Figura 2.8: Marcador GPS

Otros tipos de realidad aumentada sería el uso de ella sin marcadores, aquí primero necesitamos crear el modelo en 3D detallado de una ubicación real, esta tecnología se llama SLAM (localización y mapeo simultaneo).

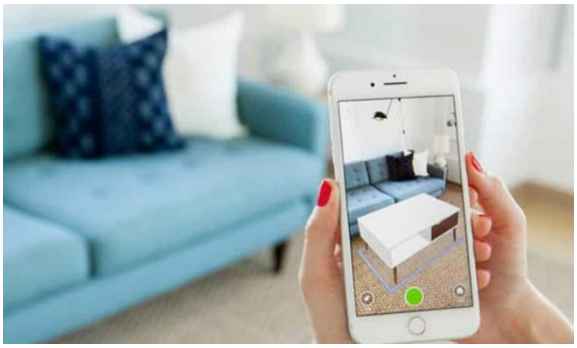


Figura 2.9: Realidad Aumentada sin marcador

Por último, existe la realidad aumentada basada en la proyección, este tipo de realidad aumentada proyecta una imagen en elementos y espacios físicos del mundo real. La realidad aumentada basada en la proyección se puede destacar por se la más atractivas, ya que es posible hacer que estas proyecciones sean interactivas.

Figura 2.10: Realidad Aumentada en proyección

2.3. Elementos que conforma la Realidad Aumentada

Para hacer uso de esta tecnología, se requerirá:

- Un elemento que capture las imágenes del mundo real, como por ejemplo la cámara de un teléfono móvil.
- Un elemento en el cual se proyecte la mezcla de las imágenes reales con las imágenes sintetizadas, como por ejemplo la pantalla de un portátil.
- Un procesador (hardware) que combine la imagen con la información que debe sobreponer.
- Un programa que gestione el proceso (software).

- Conexión a internet, debido a que la información del entorno real se envía al servidor y nos devuelve la información virtual asociada que se superpone a este.
- Se requerirá de un activador o marcador, este es un elemento del mundo real que el software los utiliza para reconocer el entorno físico y seleccionar la información virtual asociada que se debe añadir, en otras palabras es el que se encarga de reproducir las imágenes en 3D. El marcador puede ser un código QR, una imagen, etc…

2.4. Empresas que utilizan Realidad Aumentada

- ***Microsoft***

La empresa de Microsoft esta innovando en un nuevo dispositivo llamado “Hololens 2”. Es un dispositivo que utiliza la realidad mixta y que permite al usuario quedarse en el mundo real mientras el dispositivo va añadiendo elementos tridimensionales, de forma que interactúa de forma natural con el entorno real y hace que se pueda manejar y controlar. Este modelo de gafas son cómodas, ligeras y más ergonómicas que su anterior modelo, además funciona sin cables y nos da la posibilidad de tener la manos libres durante el tiempo que se este utilizando debido a que registra la información con la propia voz y mapea las manos para poder realizar todas las acciones sin necesidad de utilizar mandos o guantes. Este dispositivo esta pensando para la medicina, industria, construcción, recursos humanos, logística entre otros.

Figura 2.11: Ejemplo de uso del dispositivo Hololens 2 de Microsoft

El sistema de espejos que utiliza el dispositivo proyecta la imagen dentro del ojo, replicando la imagen miles de veces por segundo con el objetivo de tener un ancho de visión más amplia y una experiencia lo más inmersiva posible, también dispone de dos sensores debajo de los ojos que se encarga de reconocer a la persona y saber hacia donde se esta mirando.

Figura 2.12: Sistema de espejos del dispositivo Hololens 2 de Microsoft

- ***Apple***

La empresa de Apple tiene la plataforma de realidad aumentada más grande del mundo, con muchos dispositivos y aplicaciones compatibles. Además Apple diseña su hardware y software en conjunto, entre otros componentes como las cámaras, sensores y los procesadores gráficos se complementa con el aprendizaje autónomo personalizado para así poder disfrutar mejor la experiencia de la Realidad Aumentada.

Figura 2.13: Ejemplo de AR en dispositivos Apple

Las plataformas ARKit y RealityKit permite crear experiencias fascinantes de Realidad Aumentada en iPhone, iPad… Algunas aplicaciones de Apple:

- **Complete Anatomy:** Esta aplicación observa en todo detalle el cuerpo humano y se puede aprender de más de cerca cada una de sus partes, es compatible con Motion Capture del iPad Pro con el escáner LiDAR, donde los fisioterapeutas puede obtener más información sobre como mejorar la rehabilitación del paciente.

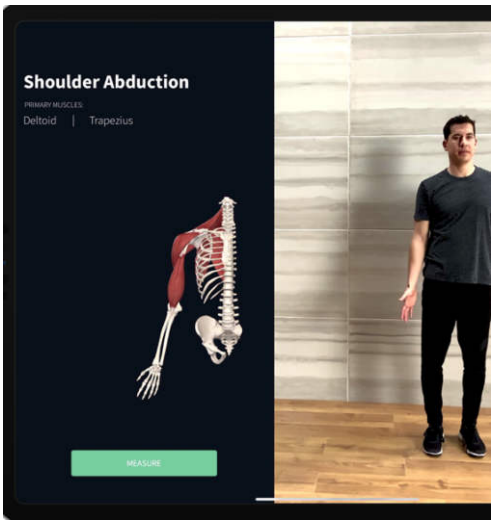


Figura 2.14: Aplicación Complete Anatomy

- **iScape:** Esta aplicación es ideal para visualizar el paisaje exterior de tu casa, en el cual nos da la posibilidad de elegir varias plantas y poder compartir la idea con tu familiares y amigos.

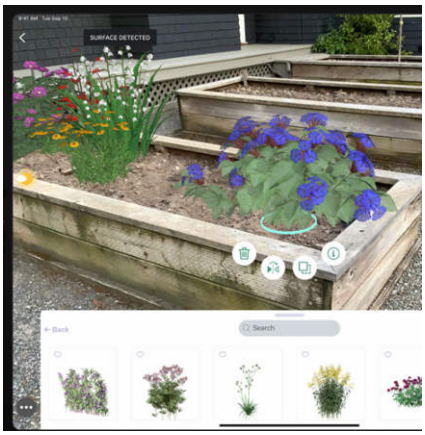


Figura 2.15: Aplicación iScape

- ***Ikea***

La empresa Ikea facilita a los usuarios el hecho de poder probar sus mueble en casa y decidir si le gusta como le queda o no, gracias a una aplicación en el móvil llamada “IKEA Place” que dispone de Realidad Aumentada, esto facilita al cliente si el mueble es ideal donde lo quiere poner o no, y no quedarse la duda en el tintero de una remota posibilidad que no le guste porque el color es distinto o por el simple hecho de que no le vale. La aplicación hace uso de ARKit, un framework de iOS, en el cual se puede instalar siempre y cuando dispongamos de iOS11, de momento para Android no esta disponible pero no lo descartan para un futuro.

Figura 2.16: Ejemplo de la aplicación Ikea

- ***Amazon***

Amazon otra empresa que apuesta por la tecnología de Realidad Aumentada, donde dispone de una aplicación llamada “AR View” permitiendo probar sus productos virtuales sobre las zonas que queramos antes de comprarlos y hacernos de una idea sobre como quedaría nuestro producto en casa, además dispone de miles de productos como material de oficina, juguetes, cocina, decoración entre otros, algo muy similar a la de Ikea. Esta aplicación esta disponible para dispositivos de iOS o Android usando el ARKit de Apple ó el AR Core para el caso de Android.

Figura 2.17: Aplicación de AR de Amazon

- ***Volkswagen***

La empresa de coches Vokswagen presenta una cabina digital de Realidad Aumentada a través del efecto 3D, lo que le da al conductor una mayor sensación de profundidad. La graficas se proyectan en el salpicadero del vehículo a dos niveles, en el primero de ellos se proyecta en la carretera a unos 8 metros de distancia aproximadamente la información sobre la posición en el carril, la navegación y las distancias que hay con respecto a otros vehículos; el segundo nivel se proyecta a unos 3 metros aproximadamente y da datos sobre el sistema de infoentretenimiento.

Figura 2.18: Cabida digital de AR de Volkswagen

Además la empresa realizo una aplicación móvil llamada“ Volkswagen AR” y además esta disponible tanto para Android como para Apple, mediante la aplicación se puede conocer el modelo de T-Cross y su equipamiento; el usuario podrá probar las diferentes funciones como la elección del color, la escala, rotar 360º, la apertura de la puerta y acceder al interior del coche con una visión de 360º.

Figura 2.19: Aplicación Volkswagen AR

2.5. Distintivas de Realidad Aumentada para Android y Apple

Para los sistemas de Android y Apple utilizan diferentes entornos de desarrollo, para Apple utilizan ARKit mientras que para Android utilizan ARCore, ambas plataformas son entornos de desarrollo que permite crear experiencias de Realidad Aumentada, básicamente se trata de librerías que se añaden, en el cual por un lado esta el software integrado en los teléfonos móviles y por otro lado como herramienta dentro de los entorno de desarrollo con lo que creamos las aplicaciones.

Diferencias entre ambas plataformas:

- AR-Kit, nos permiten trazar con precisión la posición del gráfico digital en el mundo físico. Esto es tan necesario para obtener una sensación de realidad, lo que único

a la capacidad que tiene el framework para identificar superficies y detectar iluminaciones, permite integrar con naturalidad un objeto visual en el cual encuadren con nuestra cámara. Este tipo de desarrollo esta pensando para usos educativos, ocio y entretenimiento, además ya hay comercios en el cual utiliza la realidad aumentada, como es el caso de IKEA, que ha desarrollado una aplicación para obtener una imagen real de la ubicación de un mueble en nuestra casa.

Para utilizar este entorno de desarrollo necesitamos tener instalado el software XCode.

- Google, ha versionado su propia plataforma, llamada ARCore, no requiere de varias cámaras ni de un hardware especifico, aunque necesitara tener instalada como mínimo Android 7, así como la potencia del terminal y la calidad de las lentes será muy determinantes para que ARCore funcione con fluidez.

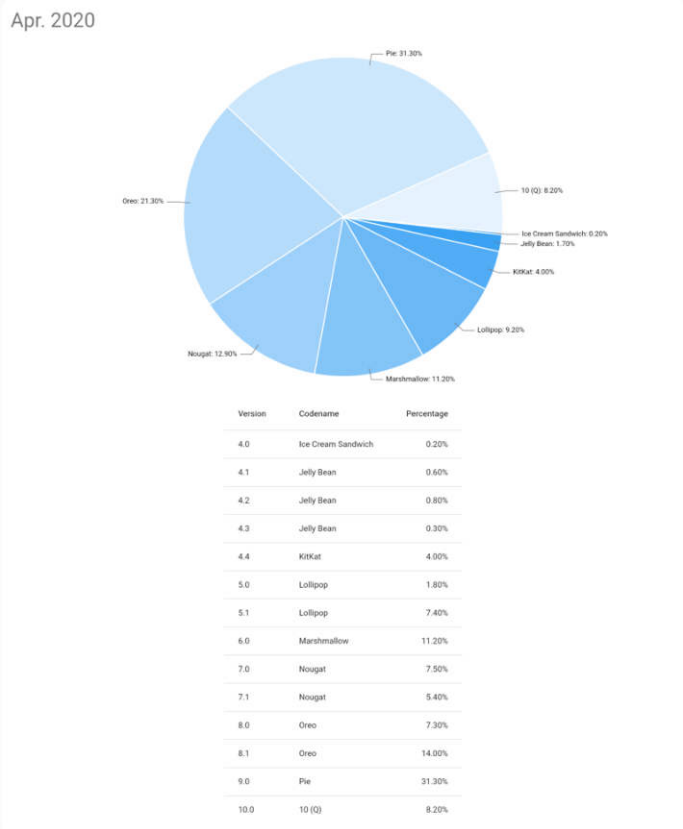
  Al igual que AR-Kit, los sensores de movimiento de ARCore permiten detectar nuestro desplazamiento y genera una imagen digital muy realista en tres dimensiones, sorprendiendo especialmente en su profundidad.

  Este tipo de desarrollo puede destacar por el anclaje de los objetos virtuales, en el cual nos permite tener un ajuste preciso de su ubicación o en la interacción con el usuario.

  Para utilizar este entorno de desarrollo necesitamos tener instalado el software Android Studio.

Debemos tener en cuenta que la ventajas de AR-Kit con respecto a ARCore es que el rendimiento de su framework será más inmediato, dado a que la gama de compatibilidad es mucho más reducida. Ambos desarrollos, pueden trabajar con software muy parecidos.

Pero el principal problemas que ha tenido Android con respecto Apple es la fragmentación, debido a la falta de actualizaciones de software y la gigantesca variedad de dispositivos compatibles hacen que sea una pesadilla para los desarrolladores. La ultima versión disponible que nos ofrece unos datos estadísticos son de abril del 2020, Android estima 633 millones de usuarios son capaces de ejecutar ARCore.



Version	Codename	Percentage
4.0	Ice Cream Sandwich	0.20%
4.1	Jelly Bean	0.60%
4.2	Jelly Bean	0.80%
4.3	Jelly Bean	0.30%
4.4	KitKat	4.00%
5.0	Lollipop	1.80%
5.1	Lollipop	7.40%
6.0	Marshmallow	11.20%
7.0	Nougat	7.50%
7.1	Nougat	5.40%
8.0	Oreo	7.30%
8.1	Oreo	14.00%
9.0	Pie	31.30%
10.0	10 (Q)	8.20%

Figura 2.20: Datos estimados sobre las versiones de Android

A diferencia con Android, Apple estiman que hay alrededor de 1.19 mil millones de dispositivos compatibles con AR-Kit, este datos nos da un visión más atractiva sobre que desarrollo de realidad aumentada prefieren los usuarios y esto es debido al gran potencial y a la gran calidad que Apple ofrecen a sus clientes, también comentar que Apple no tiene tantos modelos como Android tiene, en parte es una gran ventaja para Apple, ya que los desarrolladores tendrá menos problemas para adaptar este tipo de desarrollo a cada modelo.

2.6. Comparativa con Realidad Virtual

La Realidad Virtual y la Realidad Aumentada son dos términos diferentes y que mucha gente suele confundirlo, la Realidad Virtual como su propio nombre indica todo es virtual nos ofrece una realidad totalmente distinta y requiere de dispositivos específicos como por ejemplo el uso de cascos, va destinado a fines lúdicos o recreativos y tiene como principal objetivo la venta del propio producto y sus experiencias. A diferencia de la Realidad Aumentada transforma la realidad actual, en la cual se añaden elementos virtuales a la realidad, se puede visualizar a través de tablet o teléfonos móviles, se basa en una realidad tangible y tiene como principal objetivo la interacción.

2.7. Motor de juego Unity 3D

Unity es un motor para la creación y el desarrollo de videojuegos creada por la empresa de Unity Technologies, además tiene un gran potencial ya que permite crear juegos en 2D y en 3D, cuenta con una gran variedad de herramientas para la creación de cualquier tipo de videojuego.

El motor de Unity dio luz en el año 2005 en una conferencia mundial de desarrolladores de Apple, fue construido exclusivamente para funcionar y generar proyectos en los equipos de Apple, debido al éxito que tuvo continuaron con el desarrollo del motor y de las herramientas. En el año 2010, la empresa lanzo Unity 3 en el cual se centraron en introducir más herramientas con el fin de captar el interés de los desarrolladores más grandes. La última versión se dio en el año 2015 que lanzaron Unity 5, donde incluía Mecanim animation, soporte para DirectX 11 y soporte para juegos en Linux y arreglos en los bugs y texturas.

2.7.1. Principales características de Unity

- El motor gráfico de Unity para Windows,Mac o Linux es Open GL.
- Permite desarrollar juegos en muchas plataformas como Xbox 360, Playstation3, Wii, iPad, Android, iPhone, Mac, Windows, Linux, entre otros.
- Su desarrollo es fácil e intuitivo dentro de la complejidad de crear un videojuego.
- Tiene tres tipos de licencias (Unity Personal, Unity Plus, Unity Pro) que en función de un número de ingresos. En el caso de superar ese número de ingresos en Unity Personal se deberá de subscribir a Unity Plus, y en Unity Pro no tiene tope de ingresos.
- Presenta gráficos en 2D y en 3D, además de utilizar animaciones y sonidos.
- Unity nos permite utilizar el lenguaje de programación C Sharp (C#).
- Nos permite ordenar el contenido en carpetas, en el cual esto nos permite buscar los archivos y los objetos de manera más fácil. Además tenemos objetos predefinidos tanto en 3D como en 2D.
- Dispone de una tienda online llamada “Asset Store”.
- Presenta juegos en multijugador.

2.7.2. Interfaz de Unity 3D

En la imagen mostrada a continuación se puede ver una interfaz bastante intuitiva y fácil de localizar cada ventana.

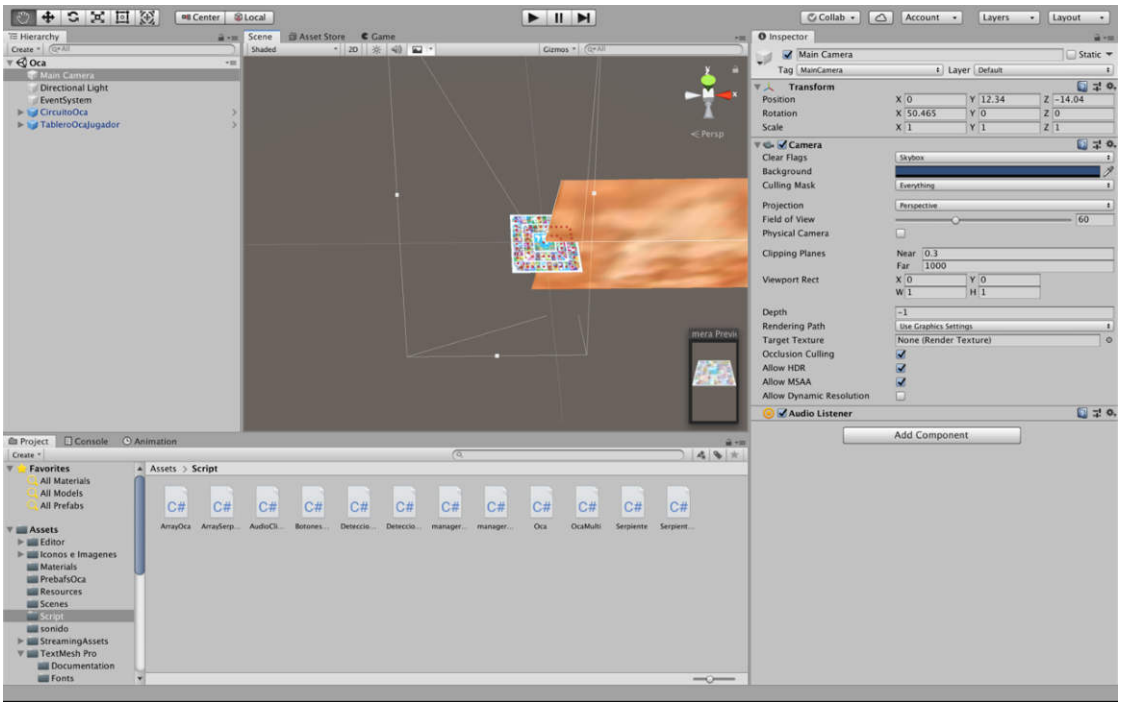


Figura 2.21: Interfaz de Unity 3D

A continuación se va a ir describiendo cada una de las ventanas que nos proporciona el programa:

- **Project Windows**

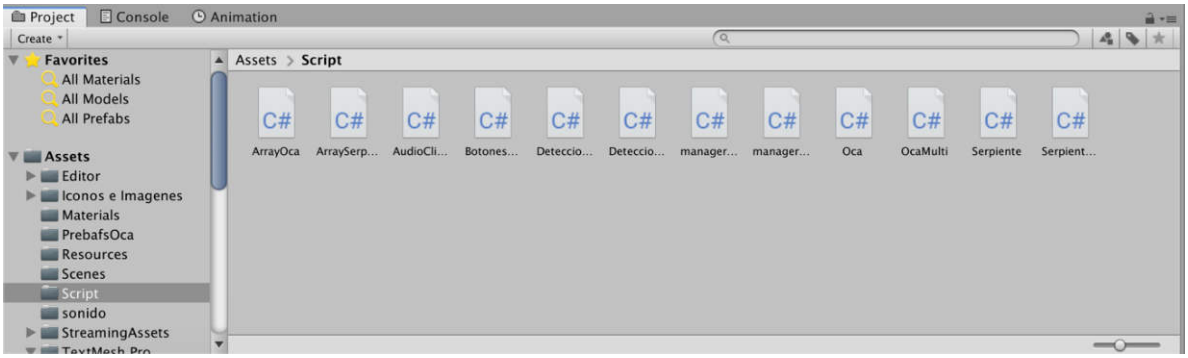


Figura 2.22: Ventanas del proyecto de Unity

En la parte izquierda aparece una jerarquía de las carpetas del proyecto en el cual están organizadas por secciones, si accedemos por ejemplo a la carpeta de “script”, nos aparecerá en la parte derecha todos los script que se haya creado en el proyecto. Además Unity nos ofrece diferentes iconos para saber si se está utilizando un material, un objeto u script, de esta manera nos resulta más fácil identificar los Assets.

- **Scene View**

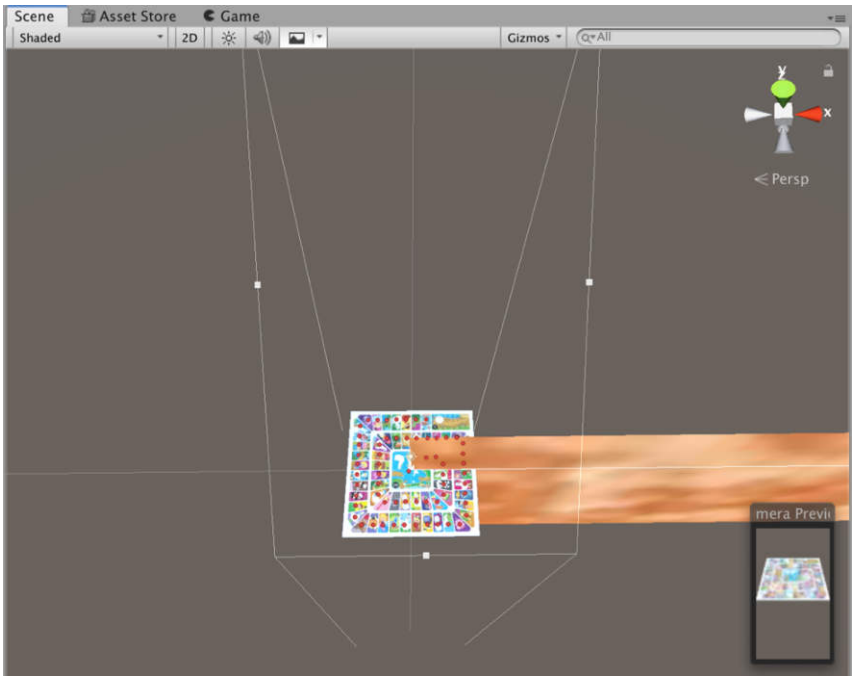


Figura 2.23: Vista de escena de Unity

La vista de escena nos permite ver una representación visual de la escena y además de editar la escena en cualquier momento como por ejemplo cambiar la posición del objeto, mover la cámara, etc.. También nos proporciona la opción de cambiar la vista a 2D y a 3D.

- **Hierachy Windows**

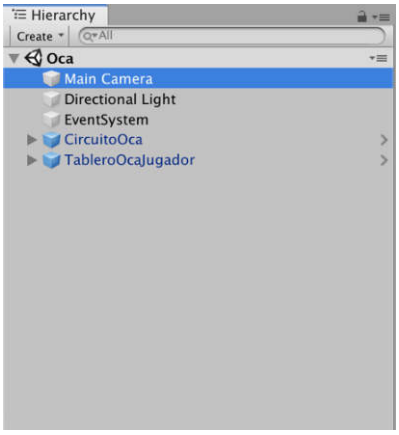


Figura 2.24: Ventana de jerarquía de Unity

La ventana de jerarquía nos muestra cada objeto que se ha añadido a la escena, alguno de estos son instancias directas de archivos de asset como modelos 3D, y otras como instancias de Prefabs (objetos personalizados). Cuando se añadan o se eliminen los objetos en la escena, estos van a aparecer o desaparece

en la jerarquía. Por defecto los objetos se mostrara en el orden que han sido creados, pero también es posible reordenar los objetos arrastrándolos hacia arriba o hacia abajo.

- **Inspector Windows**

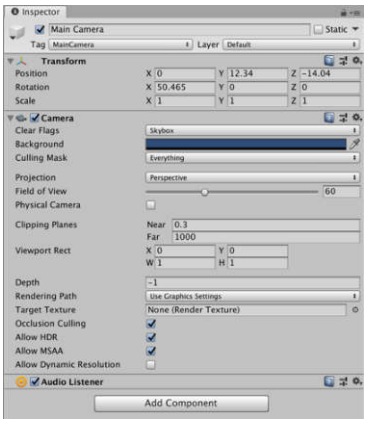


Figura 2.25: Ventana de inspector de Unity

En la ventana del inspector nos permite editar cualquier propiedad del objeto como por ejemplo la posición, rotación y escala, esto se puede hacer seleccionando un objeto en la escena y en la parte izquierda nos aparece una imagen como la mostrada con la posibilidad de cambiar cualquier acción.

- **Toolbar**

Figura 2.26: Barra de Herramientas de Unity

La barra de herramientas, se puede dividir en tres partes. En la parte izquierda contiene la herramientas de poder cambiar la vista de la escena y los objetos, en el centro se dispone de los botones de reproducción, pausa y ejecución paso a paso, y finalmente en la parte derecha se muestra los botones de acceso a los servicios de Unity Cloud, la cuenta de Unity, la visualización de las capas que se componen la escena (Layers) y algunos diseños alternativos para la ventana del editor (Layout).

2.7.3. GameObject de la escena

Los GameObject son objetos fundamentales en Unity en el cual representa personajes, accesorios, y el escenario. Estos no realizan nada por si mismos, simplemente funcionan como contenedoras para los “Components” que si realizan una funcionalidad.

A continuación se verán con más detalle los componentes más destacados de los GameObjects:

- **Transform**

  El Transform es el componente más importante de los GameObject, este componente va siempre acompañado al GameObject, y en el cual no es posible eliminarlo debido a que representa en que posición, rotación y escala el objeto.

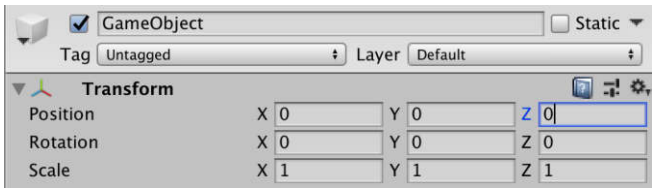


Figura 2.27: Componente Transform

- **Rigidbody**

  El componente Rigidbody modela el comportamiento físico de un objeto y ese comportamiento físico es la gravedad. Este componente tiene asociada una propiedad llamada “Kinematic” en el cual lo que hace es eliminar el control del motor de la física, y nos permite poder controlarlo mediante scripts, además nos permite poner la masa que tendrá el objeto, la resistencia del aire al mover el objeto, la resistencia del aire al girar el objeto, entre otros.

- **Colliders**

  Este componente define la forma de un objeto para colisiones físicas, en el cual expresa una aproximación del objeto. Los Colliders en 3D más utilizados son Box Collider, Sphere Collider y Capsule Collider. Todos estos tipos de Colliders tiene a su vez algunas propiedades como el tipo de material físico que utiliza, el tamaño, la posición central , y la opción “Triggers” que hace que un objeto atraviese a otro, en el caso de que se activara esta opción en nuestro software.

- **Particle System**

  El sistema de partículas es un sistema compuesto de pequeñas partículas, donde cada partícula representa (reproduce) una pequeña parte del fluido, y la acción de todas las partículas juntas produce la impresión de una entidad completa. Con este componente se podría crear humo, fuego, explosiones, etc…

  Cada partícula tiene un tiempo de vida predeterminado, esta empieza cuando es generada, y el sistema emite las partículas en posiciones aleatorias dentro de una región de espacio que puede ser la de un cubo, una esfera, por

ejemplo. La partícula se ve hasta que el tiempo predeterminado se termina y hace que se elimine el sistema.

Posee de muchos ajuste que se pueden cambiar como por ejemplo el color, el tamaño, la velocidad en la que se expande las partículas, la rotación, entre otros.

- **Audio**

  El softwate de Unity nos permite añadir audio a los objetos importando el archivo o grabar el audio desde el software. Algunos formatos de audio que Unity permite son WAV,MP3, entre otros.

3. OBJETIVOS

El objetivo principal de este proyecto, es diseñar una aplicación lúdica que permita interactuar a más de un jugador, a través de la tecnología Realidad Aumentada en tiempo real usando una conexión a Internet, así como la implementación de la aplicación en diferentes dispositivos móviles.

3.1. Objetivos específicos

- Facilitar el uso de la aplicación al usuario
- Instalación sencilla
- Soporte para varias plataformas

4. DESARROLLO DEL PROYECTO

4.1. Decisiones adoptadas

Para la realización de este trabajo, se ha hecho uso del motor de videojuegos Unity, debido a que este motor de videojuegos es muy potente gráficamente. Además presenta una amplia variedad de posibilidades como por ejemplo el uso de juegos en red, animaciones, realidad aumentada, realidad virtual, etc…

Una decisión importante es la elección de que juegos se iba a implementar. En un primer momento se iba a realizar solo el parchís, pero debido a la dificultad y una lógica más complicada, se optó por la selección del juego de la oca. También se iba a incorporar otro juego como es el de la Serpiente y Escalera, de manera que el usuario pueda elegir a cual quiere jugar.

La versión de Unity utilizada es 2018.4.14 con soporte a largo plazo (LTS), se ha utilizado esta versión con LTS debido a que se recibe las correcciones de Unity con un periodo de tiempo más largo de forma que no se tenga que estar actualizando las versiones de Unity constantemente.

La arquitectura cliente-servidor para multijugador está basado en llamadas que se realizan en el cliente y el servidor se encargar de distribuirlos al resto de clientes.

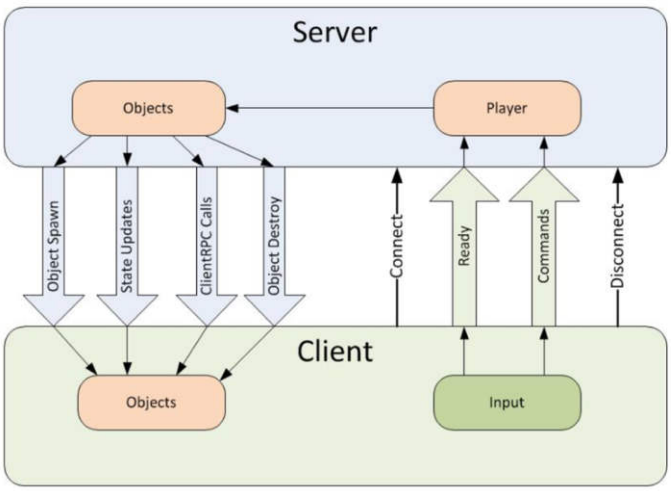


Figura 4.1: Modelo Cliente-Servidor

4.2. Comparación con otros motores

Se comentará otros motores de videojuegos muy conocidos en la actualidad como son Unreal Engine 4 y ViroReact.

4.2.1. Unreal Engine 4

Esta plataforma es muy similar a Unity, y en algunos aspectos supera a Unity como por ejemplo en los gráficos así como rendimiento.

Las diferencias que hay entre ellos son que Unreal Engine 4 trabaja con un tiempo real de 80 fotogramas por segundo y Unity trabaja a 60 fotogramas por segundo. Por otro lado, Unreal Engine 4 presenta una programación basadas en nodos, facilitando a las personas con poca experiencia con la programación utilizando el lenguaje C++.

De igual similitud, tanto Unreal Engine 4 como Unity, presenta un sistema de efectos de partículas para crear humo, niebla, explosiones, así como la creación de materiales basadas en físicas

4.2.2. ViroReact

La plataforma ViroReact permite la creación de aplicaciones tanto de realidad aumentada como realidad virtual utilizando React Native.

Esta plataforma para aplicaciones de realidad aumentada admite ARKit (iOS) y ARCore (Android), y para aplicaciones de realidad virtual permite Cardboard tanto para iOS como Android, así como Daydream y Gear VR.

Se compone de un motor de renderizado 3D nativo de alto rendimiento y una extensión personalizada de React para el desarrollo de AR y VR. El lenguaje de programación que utiliza es JavaScript.

A comparación con el motor que se ha utilizado en nuestro proyecto, Unity viene con un paquete mucho más completo que ViroReact, como por ejemplo el uso de juegos en red.

4.3. Elección de los juegos implementados

Los juegos elegidos para la implementación de este proyecto han sido el juego de la Oca y el juego de las Serpientes y Escaleras.

Se han elegido estos dos tipos de juegos porque tiene una lógica menor que otros como por ejemplo un juego de cartas o el parchís. Además, se ha preferido implementar dos juegos para que el usuario tenga la posibilidad de optar por aquel que más le guste, ya que son dos juegos muy típicos tanto en niños pequeños como en adultos.

Por otro lado, también se optó por estos juegos ya que las partidas no son muy largas de terminar e incluso de volver a jugar otra partida mientras que esperan los clientes la comida.

4.4. Aprendizaje en el diseño

En esta primera fase, se comentará todos los pasos previos a la implementación de este proyecto.

El proyecto constará de la creación de un tablero así como la ficha para cada usuario que tendrá un color diferente entre cada uno. De esta manera, cada usuario puede identificar su color. También será necesario de crear una ruta por donde nuestro objeto va a ir. Para todo lo comentado será necesario usar los “GameObject” de Unity. Por consiguiente, será de utilidad el manejo de los botones, y como añadir una cierta función especifica a un botón.

Una escena viene determinada por aquellos objetos que se utiliza en ellas. Por tanto, un proyecto puede estar compuesto por varias escenas diferentes. Cada escena es un nivel único, es decir, que si cambias de escena esos objetos no se trasladan a otra escena.

A su vez, será necesario saber como manejar el cambio de escenas, como por ejemplo el cambio de escena cuando pulsas un botón. Esto será necesario dado que nuestro proyecto tendrá más de una escena.

Se diseñará que cuando se pulse sobre el botón de tirar los dados, realice el sonido de tirar los dados. Además, mientras se mueve una ficha, nos muestra cual de los usuarios esta realizando el movimiento, así como perder turno o caer en alguna casilla especial a través del color que le ha tocado aleatoriamente.

Cuando se finalice la partida, el usuario tendrá diferentes opciones cuya principal opción es reiniciar el juego o salir completamente del juego.

Ya que nuestro proyecto está basado en un juego multijugador en tiempo real, se necesita crear un servidor y que el usuario se una a ella. Para ello, se facilitará al usuario

que no tenga que indicar la dirección IP así como el puerto al que se va a conectar, sino que se realizará mediante una selección de botones que facilitará su uso.

Por último se utilizará la tecnología Realidad Aumentada que, a través de un código QR, se puedan ver los objetos virtuales creados con anterioridad. Cada código QR estará enlazado con un puerto. Esto se podrá aplicar a cualquier teléfono móvil o Tablet.

4.5. Implementación

En este apartado hablaremos más detenidamente sobre lo comentado en el apartado anterior. Además, en los siguientes apartados de este capitulo se detallará el código empleado en cada función y explicado con detalle que es lo que realiza cada cosa.

Para la realización de este proyecto se hará a través de Visual Studio con el lenguaje de programación C#, además de las librerías que hay que usar como UnityEngine.UI,UnityEngine.SceneManagement, TMPro. Si queremos acceder a los métodos de botones, canvas, imágenes entre otros necesitamos tener la librería UnityEngine.UI. Para hacer un cambio de escena, necesitamos tener la librería UnityEngine.SceneManagement. En el caso de utilizar texto de alta calidad es necesario incorporar la librería TMPro.

Para el movimiento de la dicha, se va hacer uso de la función MoveTowards que dispone Unity, en el cual se le pasará tres parámetros a esta función (el objeto que se quiere mover, hacía donde se dirige y la velocidad que tendrá el objeto). Esta función, es un tipo de animación que hará que parezca que la ficha la este empujando una persona con el dedo.

Para que cada usuario tenga un color, se hará uso de una lista, en este caso del método “Color” que hará referencia a cada componente RGB de cada color. Esa lista de colores estará formado por trece colores, para ello se hará uso de la función UnityEngine.Random.Range que determinará de esa lista un color aleatorio.

Se incorporará el audio cuando se pulse sobre el botón de tirar los dados. Además en el menú, cuando se pulsa sobre cualquier botón sonará un audio de clicar sobre algún botón, así como un sonido ambiental cuando se inicia la aplicación. Para la realización de ello, será necesario incorporar estos sonido a Unity, y crear un objeto llamado “AudioSource” para cada audio y arrastrar los sonidos importados. Por otro lado, se debe de crear la función que realiza cada audio y añadiéndolo a la sección “OnClick” de cada botón. Para crear una función de sonido, se necesita crear dos variables una llamada “AudioSouce” y otra llamada “AudioClip”, y la referenciaremos con el objeto creado en Unity

“AudioSource” mencionado anteriormente y el sonido que se ha importado. Estas dos variables se utilizará en la función y para hacer que se escuche el sonido, se utilizará la función de Unity llamada “PlayOneShot” que hará que se reproduzca el sonido con las variables que se han referenciado.

Por otro lado y de manera paralela para saber por texto que color tira, hay que realizar una serie de comprobaciones en función de las componentes de color (RGB). Por ejemplo, si el objeto de la componente R es 1 y si el objeto de la componente G y B es 0, se usará una variable de tipo String llamada TextColor y se igualará al nombre del color entre paréntesis, es decir, en el caso expuesto “Rojo”.

Cuando un usuario tira los dados, usará una función que estará compuesto por una variable de tipo “TextMeshProUGUI” llamada textoJugador, en el cual, esta variable será igual al texto que se desea introducir entre comilla más la variable textColor. Con esto sabremos que ficha se esta movimiento en función del color.

Para realizar un cambio de escena, el script como se ha comentado antes, debe tener la mención de la librería “UnityEngine.SceneManagement”, en el cual se puede acceder al método LoadScene y entre comillas indicar el nombre de la escena tal y como aparece en el editor de Unity. Sino se pone bien el nombre saltará un pequeño error diciendo que la escena no ha sido cargada.

Una vez que se tenga una función terminada como el cambio de escena con el uso del botón, al seleccionar un botón, en la ventana inspector del editor se puede ver la sección llamada “OnClick”. Se le añade el objeto de la ventana de jerarquía que contenga dicho script con la función. Se busca la función que va a realizar el cambio de escena con el nombre que se le ha puesto a la función.

Ya que el juego va a ser un juego en red para poder jugar más de una persona en una misma partida, el script del objeto en lugar de ser MonoBehaviour tendrá que ser NetworkBehaviour, así como insertar la librería UnityEngine.Networking. También se necesita usar que funciones va a realizar el cliente y el servidor, para ello si se quiere sincronizar algo con los demás jugadores se usar las funciones ClientRpc y Commands, en el apartado 4.9 se detallará su uso así como la estructura que debe tener. Además, en el editor de Unity se necesitará hacer uso de algunos componentes para poder usar la parte de red de Unity. Para ello en el apartado 4.9 se indicará cuales se han usado así como la función especifica de cada uno de ellos.

Por último, para implementar la Realidad Aumentada, se necesita incorporar la “ARCamera”, en el cual en la configuración se accederá a la pagina web de Vuforia para que se cree una cuenta y se pueda crear una licencia, así como una base de datos para

incorporar las imágenes que hará de marcador para usar el juego, de forma que en el editor también se necesita incorporar la “ImageTarget” para poder usar esa base de datos que posteriormente importaremos en el editor de Unity. En el objeto incorporado “ImageTarget”, tiene un script que cuando detecta un marcador de la base de datos salte el objeto virtual y cuando no detecta el marcado es porque se ha perdido el objeto virtual. Este script que Unity lo trae por defecto será modificado para nuestro proyecto que estará detallado en el apartado 4.10.

4.6. Desarrollo de la Interfaz de la aplicación

En esta sección todo lo que se comentara estará creado en una escena llamada “Menu”.

Se explicará, como se ha creado la interfaz de la aplicación (botones, imágenes, audio, efectos en los iconos y botones). La explicación de esta sección se realizara mediante diferentes pantallas, es decir, cuando se pulse un determinado botón se ira a otra pantalla dejando de aparecer la anterior, solo se mostrara la pantalla correspondiente en función de que botón del menú se ha pulsado.

Todos las pantallas que formen en este punto estarán dentro de un “Canvas”. Para crear un “Canvas” en la escena del juego se puede hacer o bien pinchando botón derecho en la ventana de jerarquía, sección UI y se selecciona “Canvas”, tal y como aparece en la siguiente imagen, u otra opción en la sección de barras situada en la parte superior pinchamos sobre GameObject→UI→Canvas.

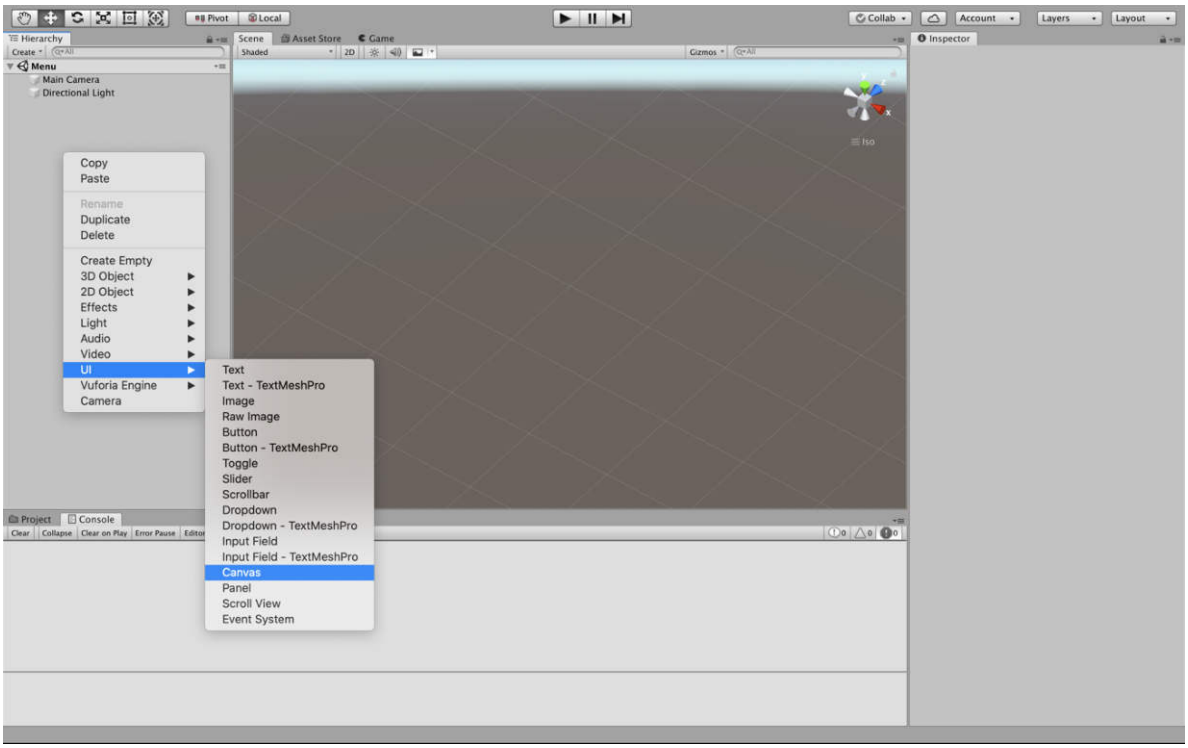


Figura 4.2: Añadir Canvas en una escena

Una vez que se ha creado el “Canvas” como aparece en la siguiente imagen, se va a seleccionar el tamaño de este.

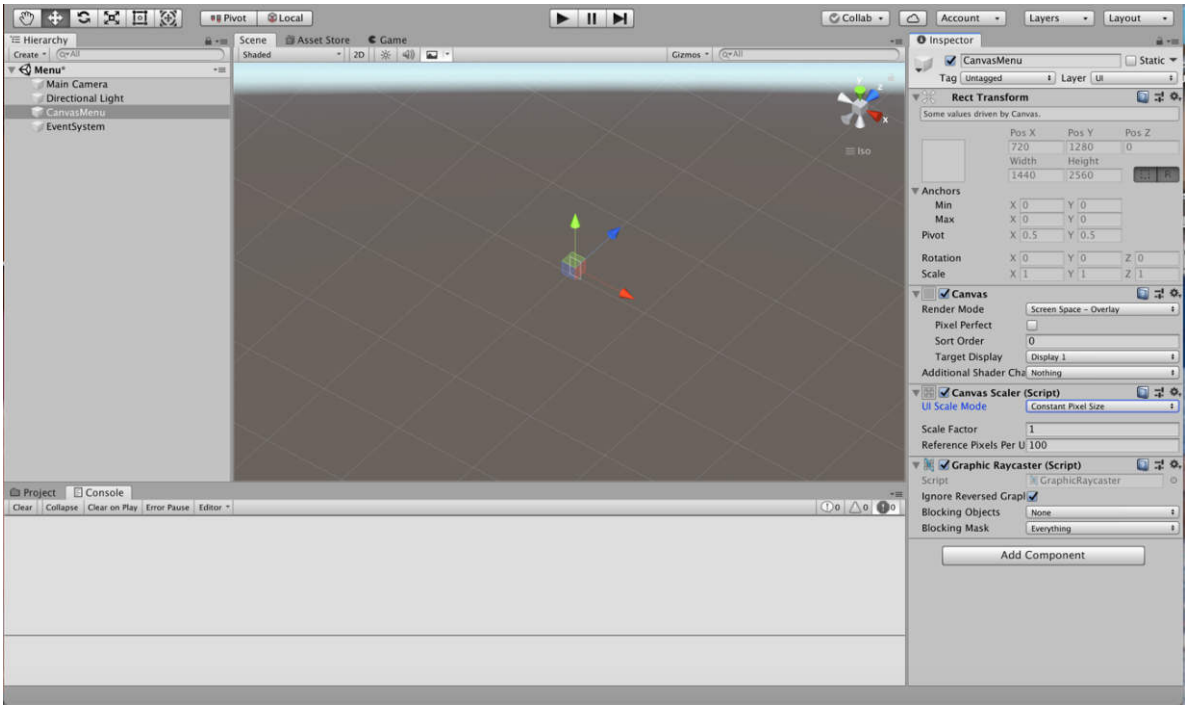


Figura 4.3: Vista del objeto Canvas en la escena

Para la selección del tamaño, en la sección de “Game” nos aparecerá varios tipos de resoluciones, en el cual se seleccionará el tamaño de 1080x1920 debido a que este es un tamaño para un dispositivo móvil, tal y como se muestra a continuación en la imagen.

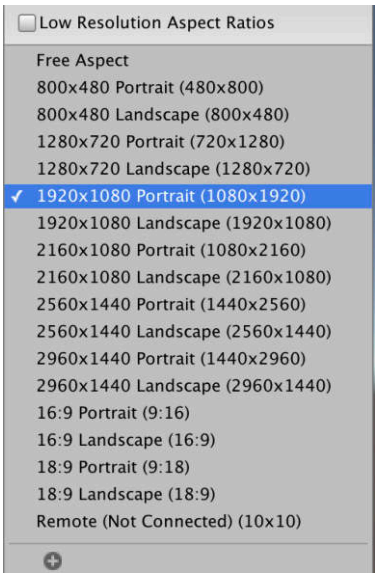


Figura 4.4: Selección del tamaño

Si nos fijamos en la siguiente imagen, se ve que ya se tiene el tamaño deseado, recordemos que se ha usado un tamaño para un teléfono móvil.

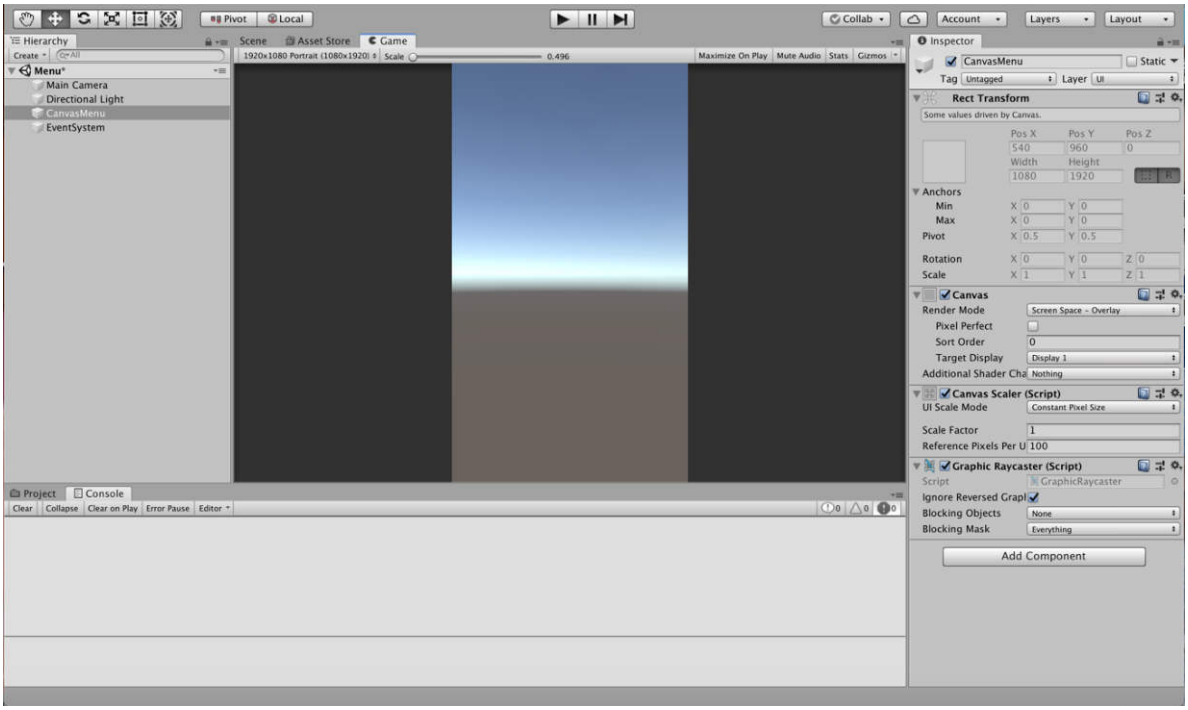


Figura 4.5: Vista de la selección del tamaño

Por otro lado en la ventana de “Inspector” del “Canvas”, se va a seleccionar que la escala del “Canvas” sea escalable con el tamaño de la pantalla, es decir, que se ajuste a cualquier pantalla.

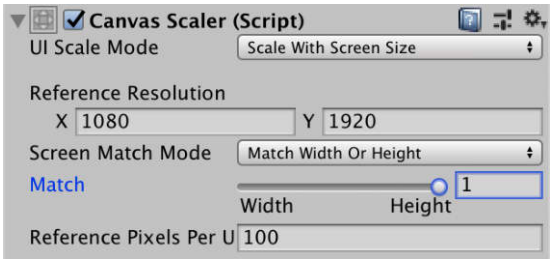


Figura 4.6: Interfaz de Usuario escalable con el tamaño de la ventana

Dentro del “CanvasMenu”, irá cada sección de pantallas que son creados mediante objetos vacíos (Create Empty) y en el cual cada sección de pantallas irán con sus correspondientes botones.

4.6.1. Menú Principal

En esta pantalla, aparecerán los botones Jugar, Opciones y Salir, además del fondo de pantalla.

Para crear los botones dentro de esta sección, se pincha sobre el objeto vacío “MenuPrincipal” seguidamente de UI, y se selecciona “Button”, tal y como se muestra en la siguiente imagen.

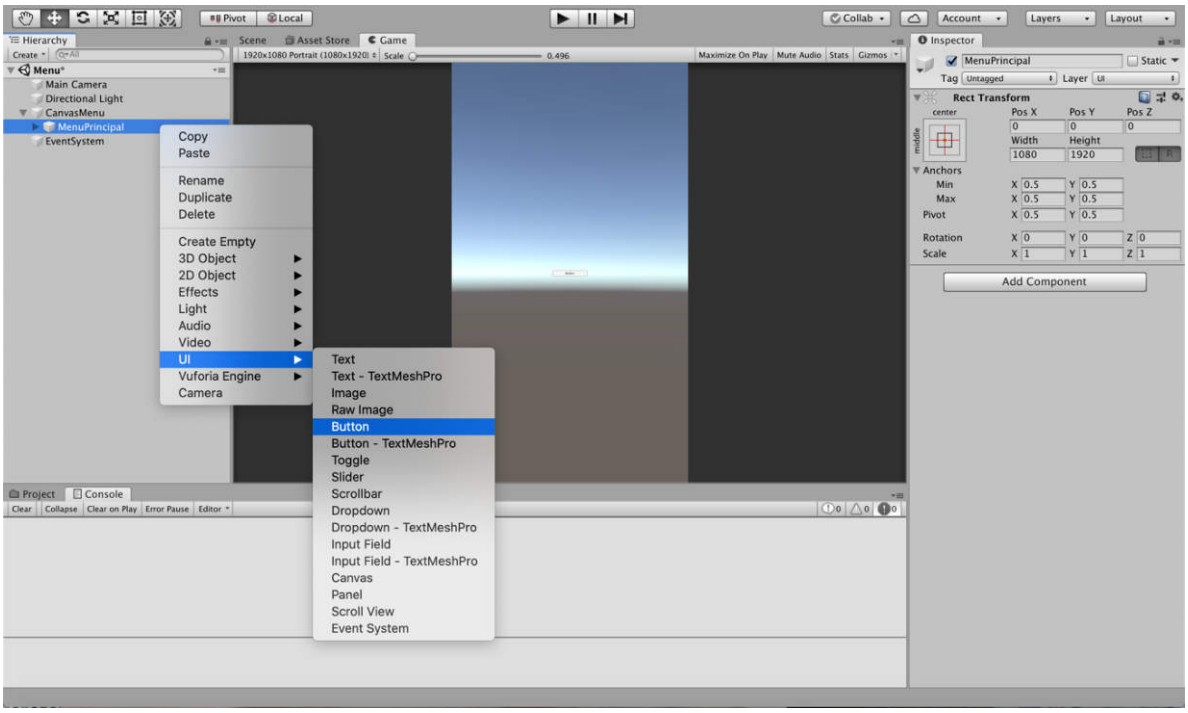


Figura 4.7: Añadir un botón en la escena

Cuando creamos los tres botones, le cambiamos la posición en el eje Y a cada uno de los botones para que no aparezca una encima de otro.

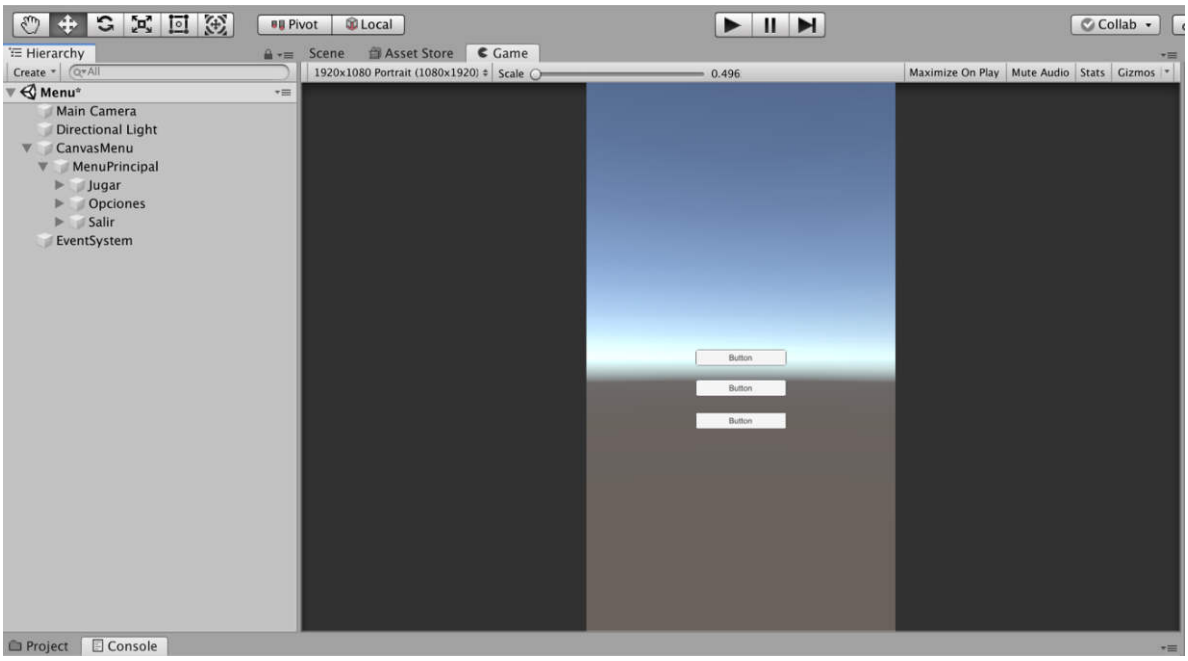


Figura 4.8: Posición de los botones en la vista del usuario

También tiene la posibilidad de cambiar el nombre del botón que por defecto, aparece como “Button”.

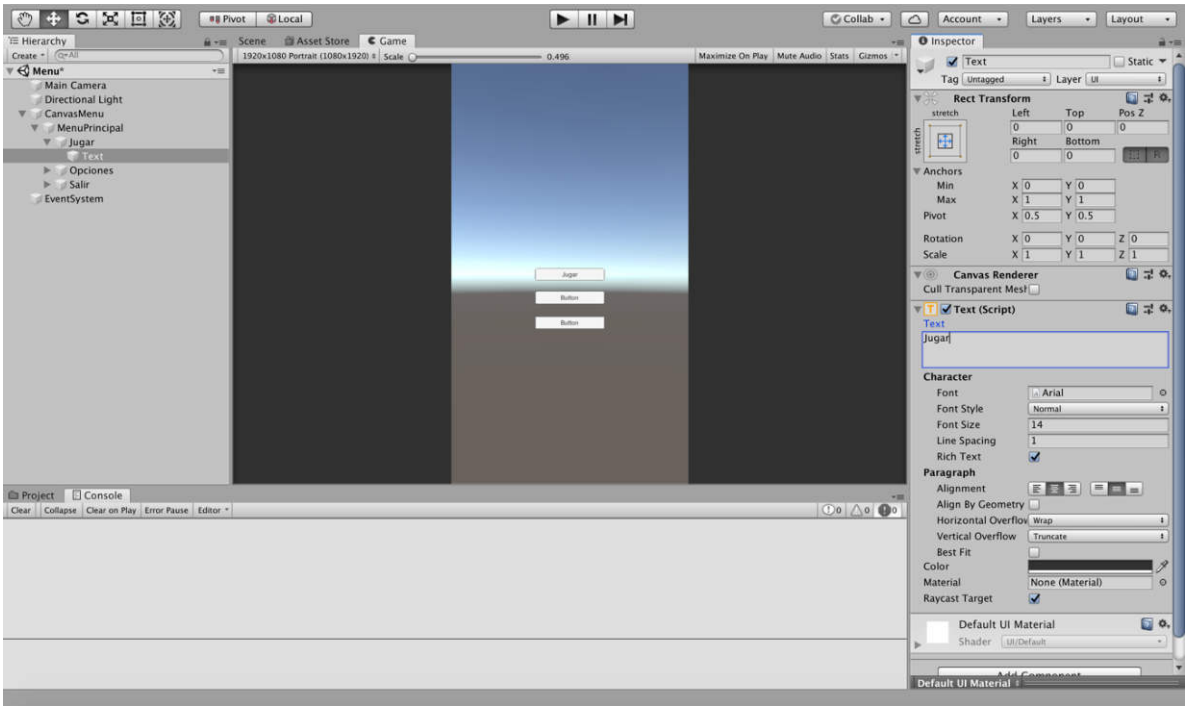


Figura 4.9: Cambiar el texto de los botones

Como se muestra en la imagen anterior, dentro de cada botón tiene incorporado el texto que va asociado a cada botón, además de poder cambiar la fuente, tamaño, líneas de espacio, entre otras más opciones.

Las características del texto para todos los botones serán las mostradas a continuación:

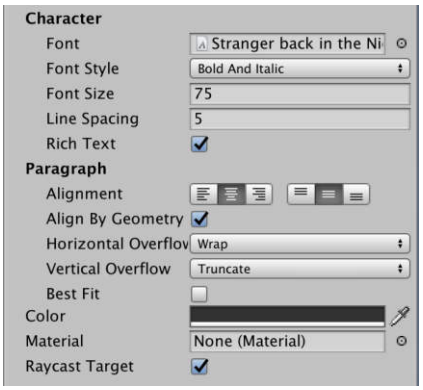


Figura 4.10: Componentes del texto de botones

El tamaño para todos los botones será de 195.1 de anchura y de 78.3 de altura. Además, se cambiara el icono de la imagen de cada botón, por uno que se ha descargado en internet, y para asociar nuestro icono descargado al botón se realiza de manera sencilla, simplemente arrastrando la imagen del icono a la sección de “Source Imagen” situado en la ventana “Inspector”. Para la utilización del icono se debe de cambiar la textura de dicho icono a Sprite.

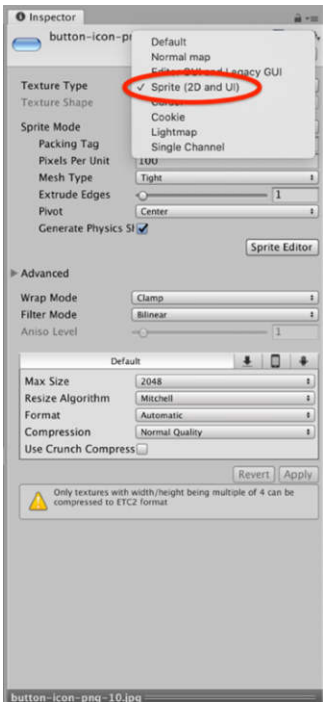


Figura 4.11: Tipo de textura del botón

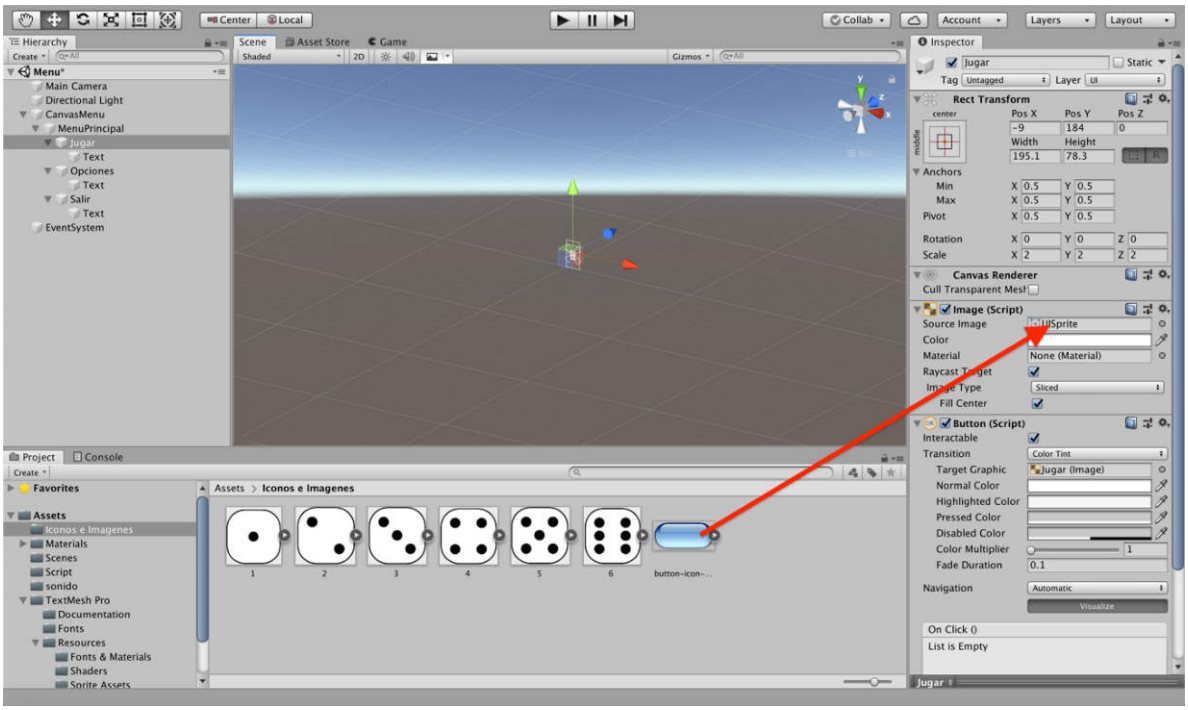


Figura 4.12: Añadir el icono del botón

Aplicando todas las características mostradas, aparece nuestro menú principal de la siguiente manera.

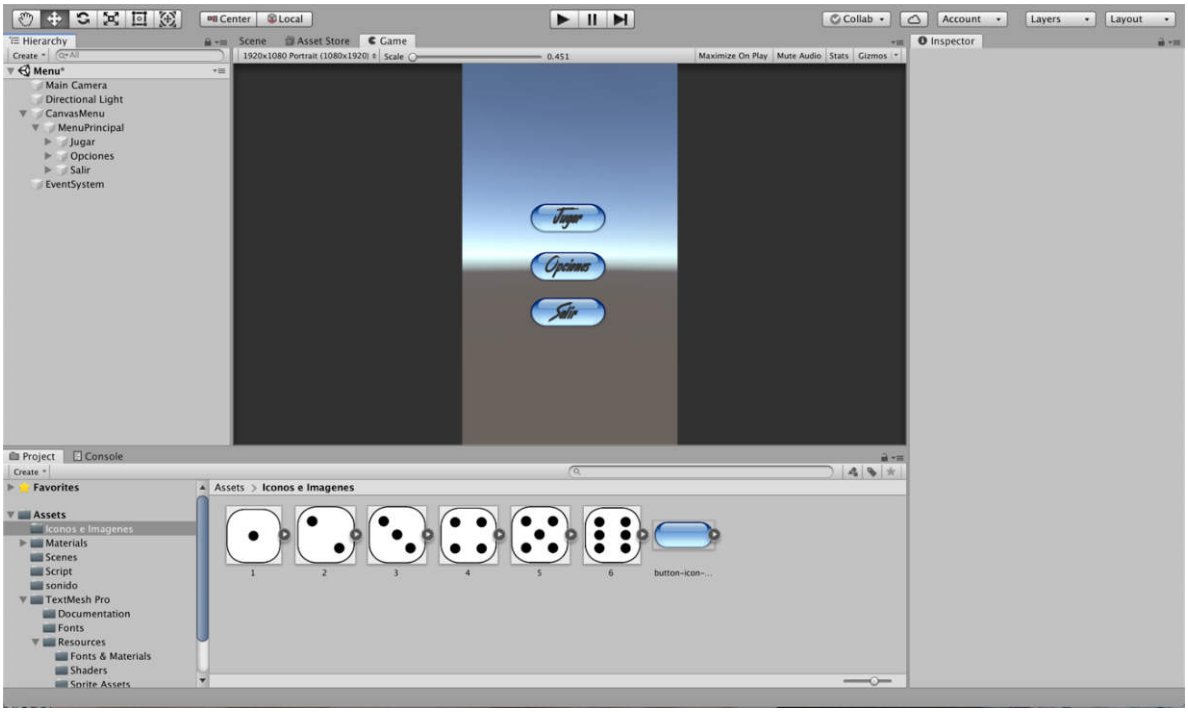


Figura 4.13: Vista final de los botones

Por último, lo que nos falta de la parte estética en esta sección es el fondo de pantalla, para incluirlo pinchamos sobre “MenuPrincipal”, seguidamente de UI, y seleccionamos “RawImage”.

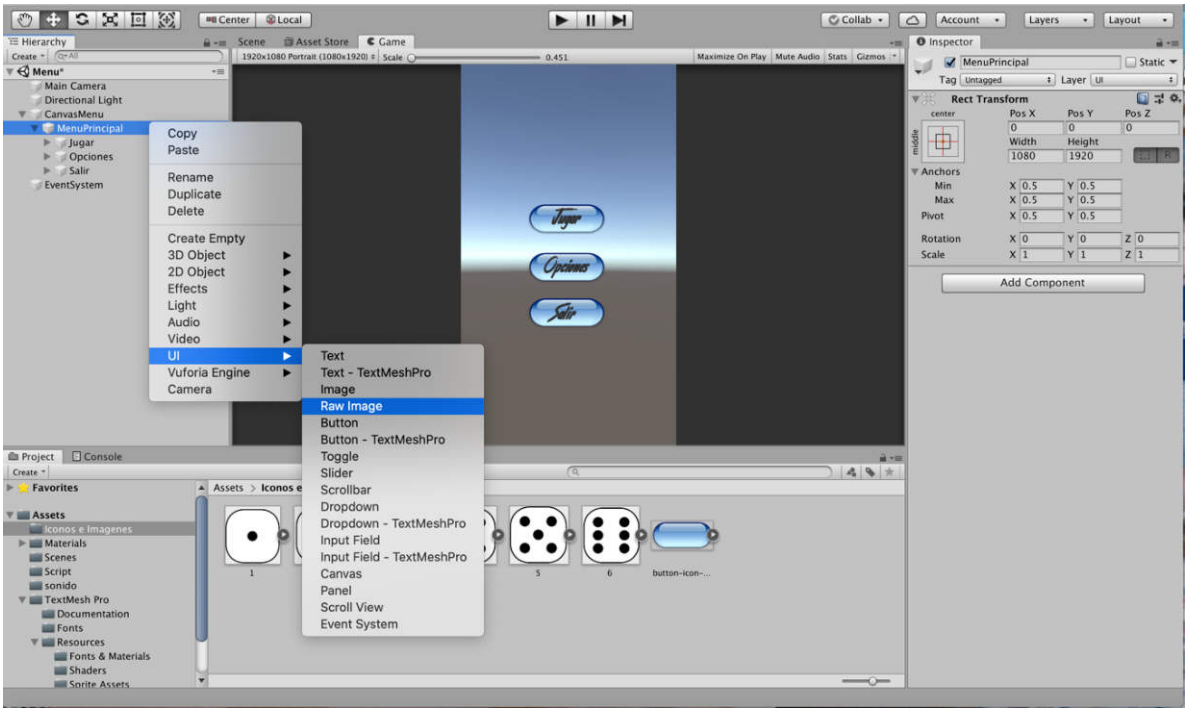


Figura 4.14: Añadir un fondo al Canvas en la escena

Una vez que ya aparece en la escena, le modificamos el tamaño para que sea el mismo que el del “Canvas”. Además, se le cambiara el fondo por uno que se ha descargado en internet, y se realizará de la misma manera que con el botón se le debe de cambiar el tipo de textura a “Sprite”, y arrastrar el fondo escogido a la sección de “Texture” en la ventana del Inspector de Fondo.

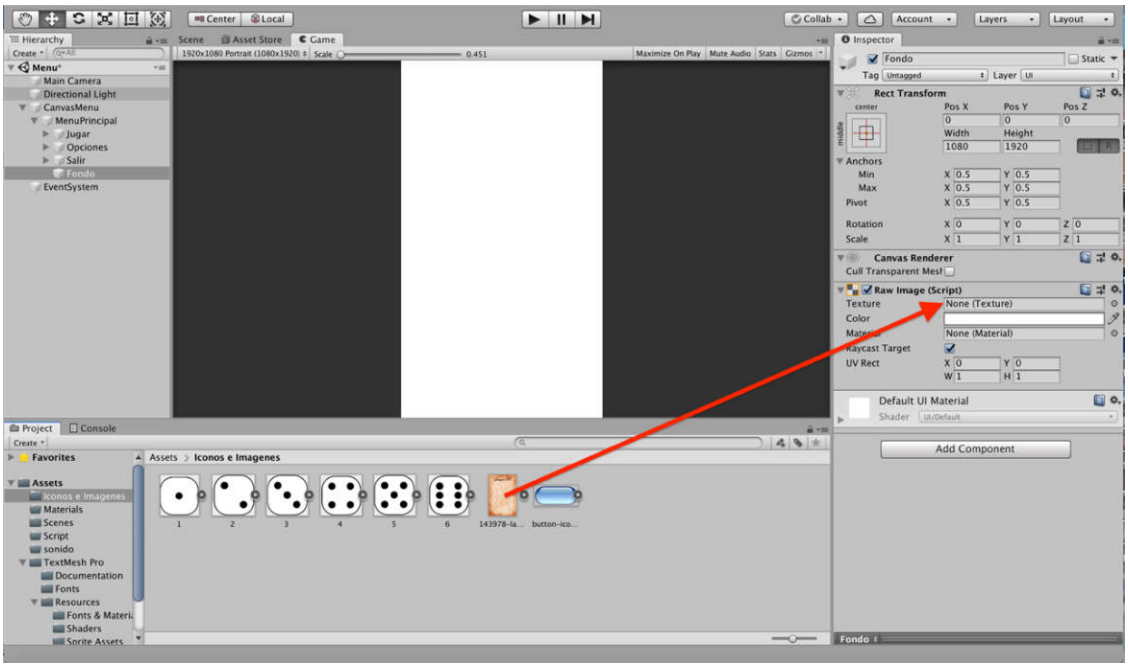


Figura 4.15: Añadir el icono del fondo

Por lo tanto, el fondo de nuestro menú aparecerá de la siguiente manera:

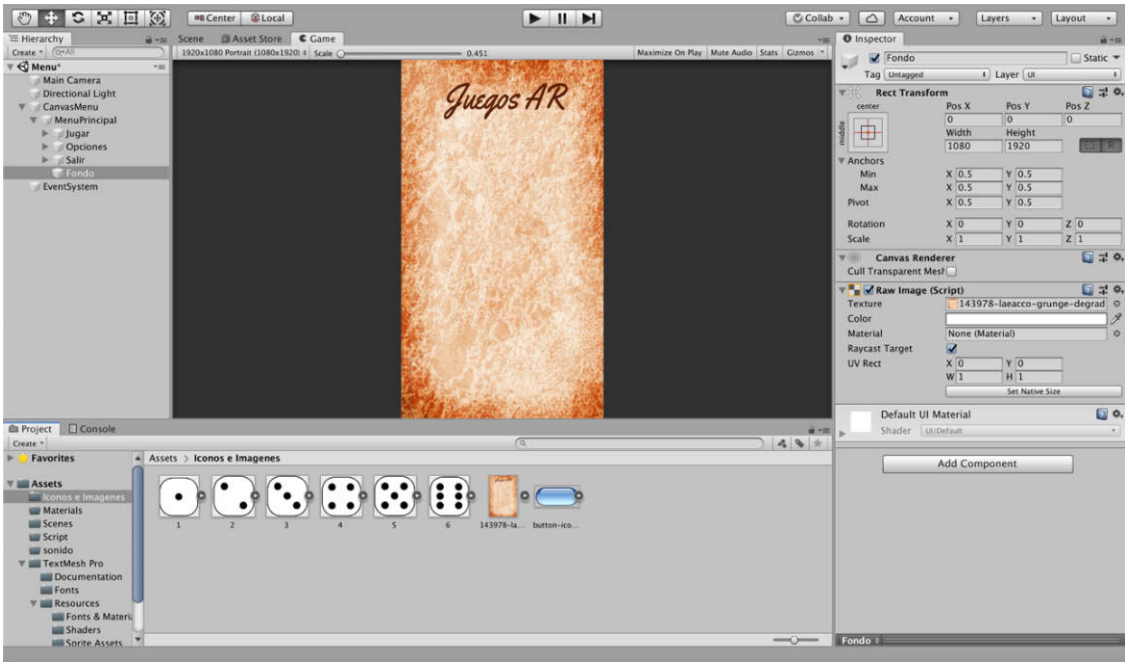


Figura 4.16: Vista del fondo

Como vemos en la imagen anterior, los botones no aparecen es debido a que los botones deben estar dentro del objeto llamado “Fondo”, de esta manera ya nos aparece como queríamos.

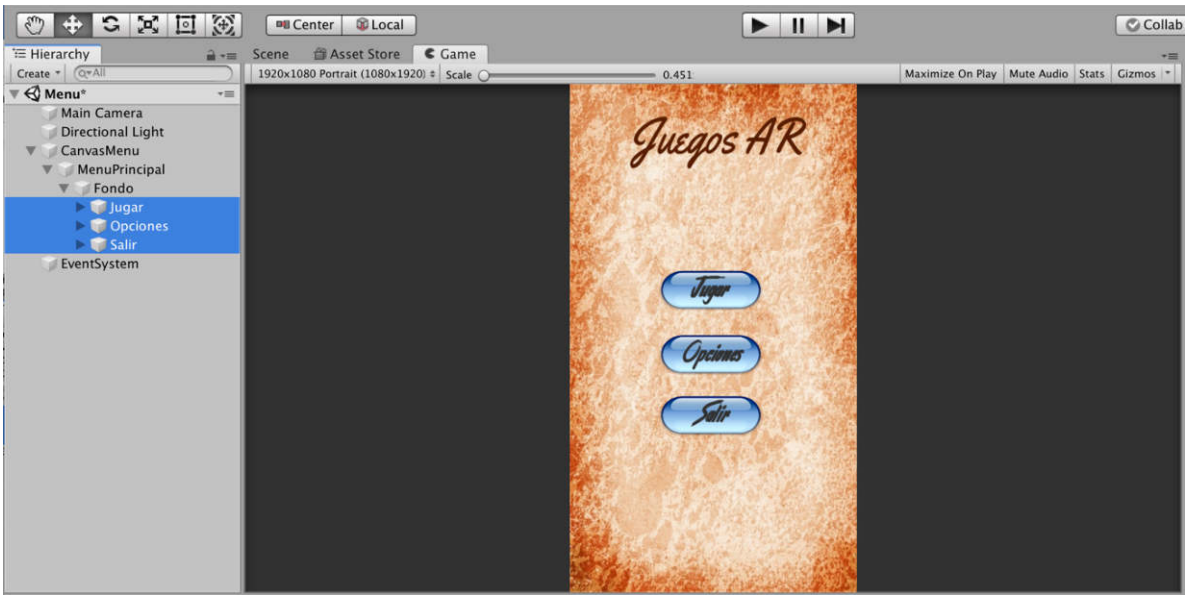


Figura 4.17: Vista final del Menú Principal

4.6.1.1. Funciones de los botones

En esta parte, se va a mostrar las funciones que componen cada botón, es decir, aquí lo que se muestra es que pase de una pantalla de menú a otra cuando se pulsa sobre cualquier botón.

Para incluir una función al botón, se va hacer uso de la sección “OnClick()” situado en la ventana de Inspector que dispone cada botón que se crea, en el cual se le puede atribuir múltiples funciones. Para asignarle alguna función, se tendrá que pulsar sobre el icono +, de esta forma se le asignara una función en concreto para cada objeto o para el mismo objeto.

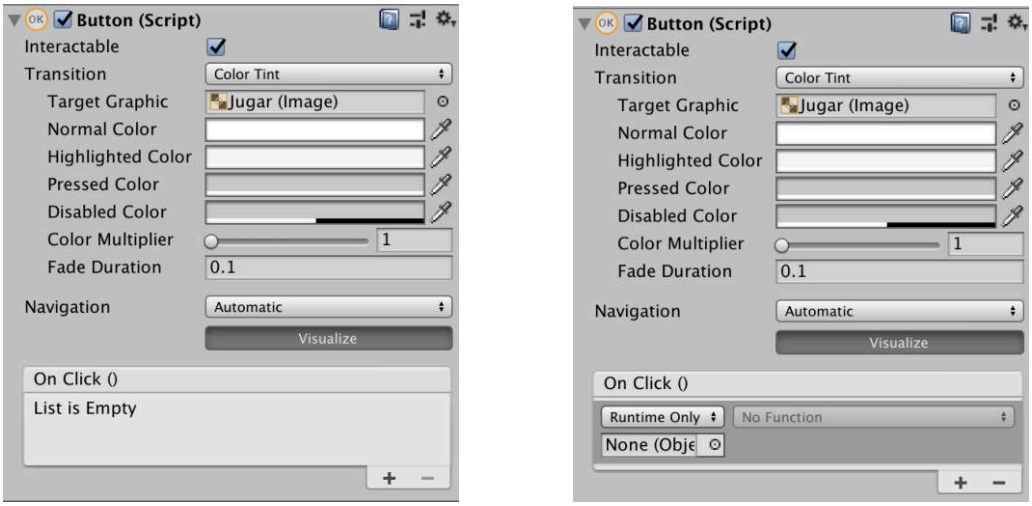


Figura 4.18: Componentes del botón

- ***Jugar***

A este botón, se le va atribuir tres funciones, el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo) y la pantalla del siguiente menú (para activarlo). Para que se pueda asociar la siguiente pantalla del menú, se debe de crear en la ventana de jerarquía otro objeto vacío con el nombre “MenuJugar” (para que se le pueda asociar este objeto a este botón), a la cual en el siguiente punto se detallará que botones y funciones realizan.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello se arrastra los dos objetos en la sección de “OnClick()” y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el botón “Jugar” cambie a la pantalla de “MenuJugar” y se desactive la pantalla “MenuPrincipal”.

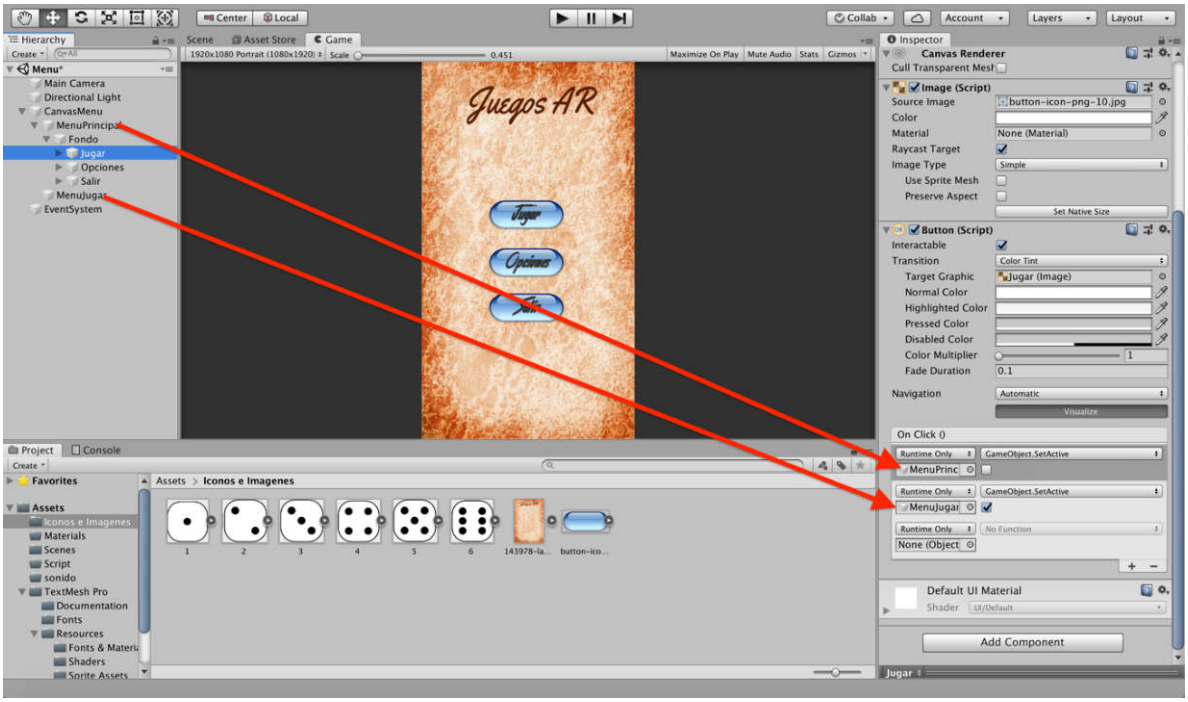


Figura 4.19: Cambiar de pantalla del botón Jugar

Por último, incorporamos el audio. Lo primero es importar el audio a la carpeta de sonido en la ventana de proyectos de Unity.

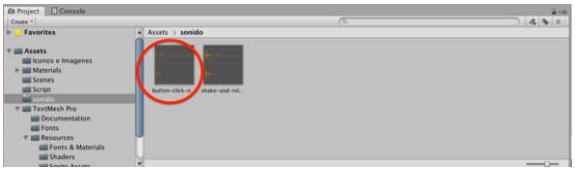


Figura 4.20: Audio al clicar un botón

Lo siguientes es crear dentro de “CanvasMenu” un objeto vacío al que llamaremos “AUDIO”, en el cual se va a contener un script llamado “Sonidos.cs”, para el audio del menú.

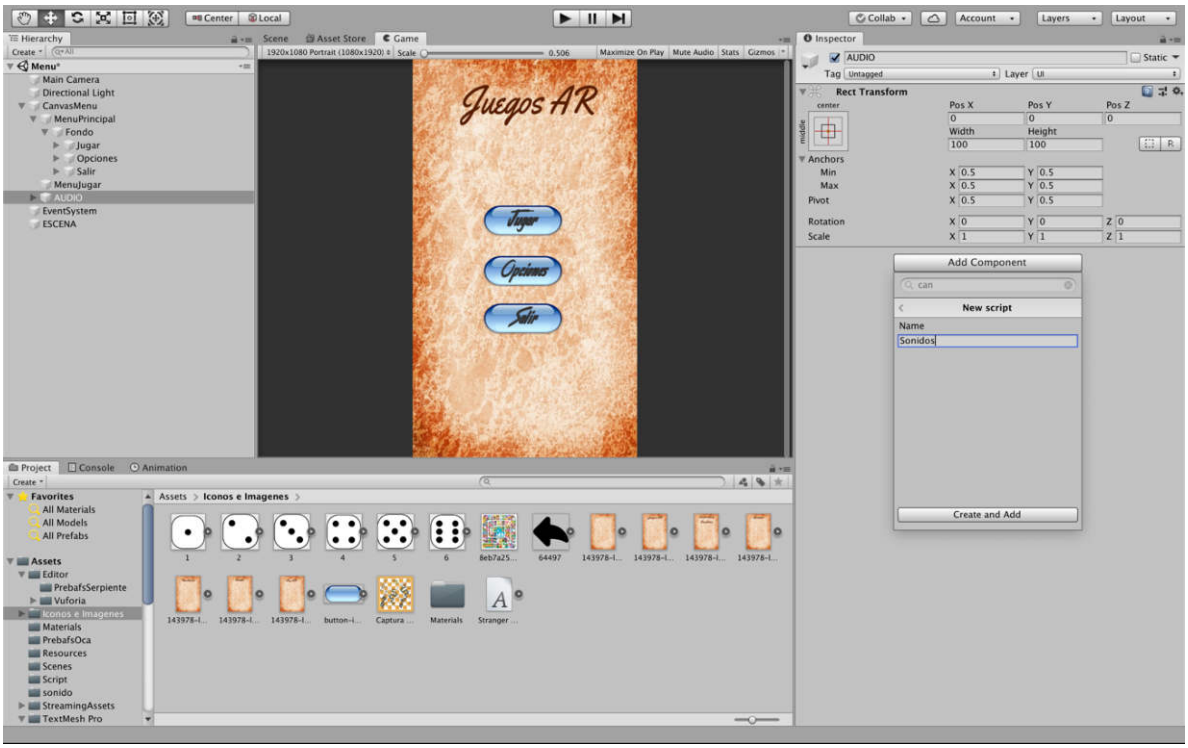


Figura 4.21: Nombre del script de los sonidos

En el script creado, se va a declarar dos variables de tipo públicas, una de ellas “AudioClip” que almacena el archivo de audio comprimido, y la otra variable “AudioSource” que hace referencia a los AudioClip y se utiliza para poder reproducir el sonido.

```
public AudioSource AudioClip;
public AudioClip ClipButton;
```

Figura 4.22: Declaración de variables del sonido al clicar un botón

Declaradas ambas variables, se procede a crear la función llamada “ClipButtonAudio()”, en la cual se va a utilizar una función que ya tiene creada por defecto Unity llamada “PlayOneShot”, que hará que se reproduzca el sonido cada vez que se pulse sobre algún botón del menú (es valido para todos los botones referidos en la escena del menú).

```
public void ClipButtonAudio()
{
    AudioClip.PlayOneShot(ClipButton);
}
```

Figura 4.23: Función que reproduce el sonido al clicar un botón

En Unity, dentro del objeto “AUDIO”, se crea un objeto de audio (AudioSource), se hará como se muestra a continuación en la imagen.

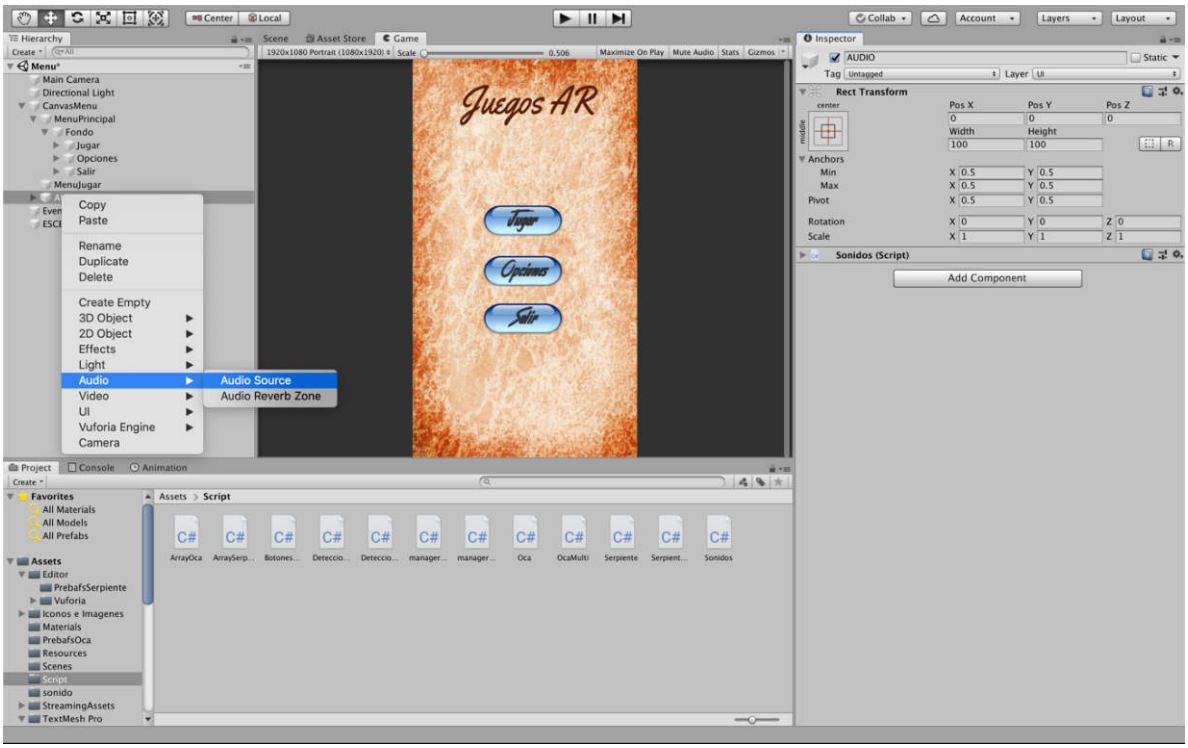


Figura 4.24: Añadir el componente AudioSource

El objeto creado “AudioSource”, le cambiamos el nombre a “Audio Source Clip”, para distinguirlo por si creamos diferentes sonidos, y le añadimos el audio que importamos a la carpeta de sonido a la sección “AudioClip” situado en la ventana de jerarquía de “Audio Source Clip”.

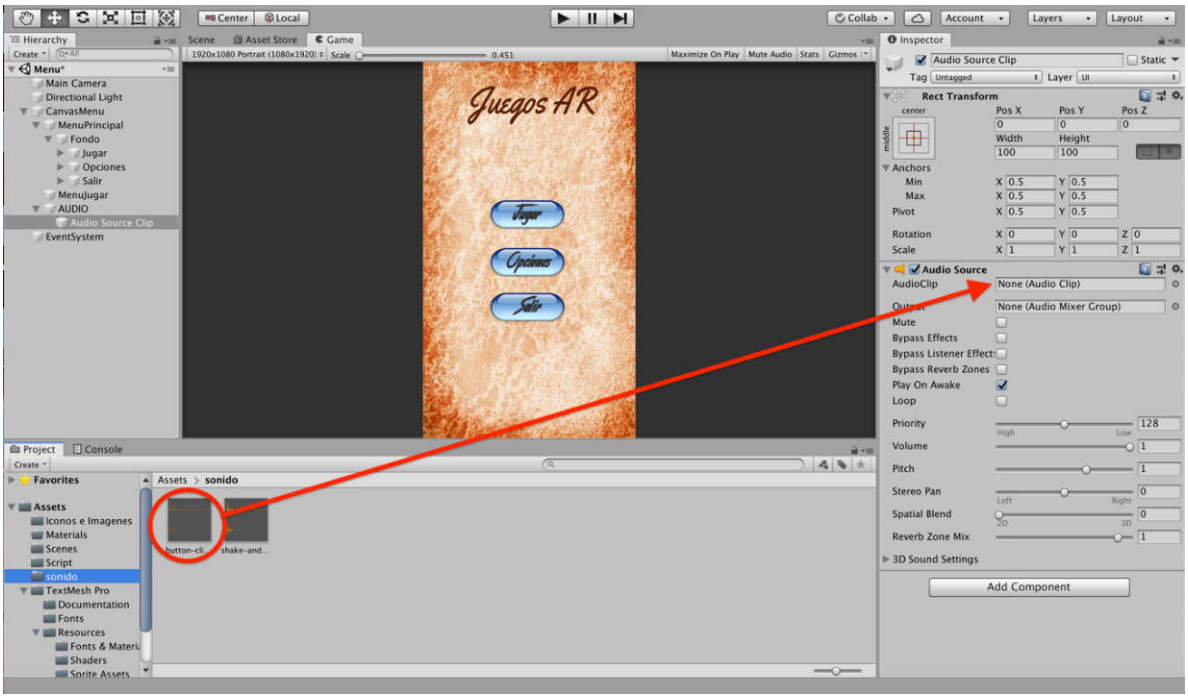


Figura 4.25: Añadir el sonido al componente AudioSource

Por tanto, ya solo queda asociar los objetos de audio a las variables publicas que se han creado en el script.

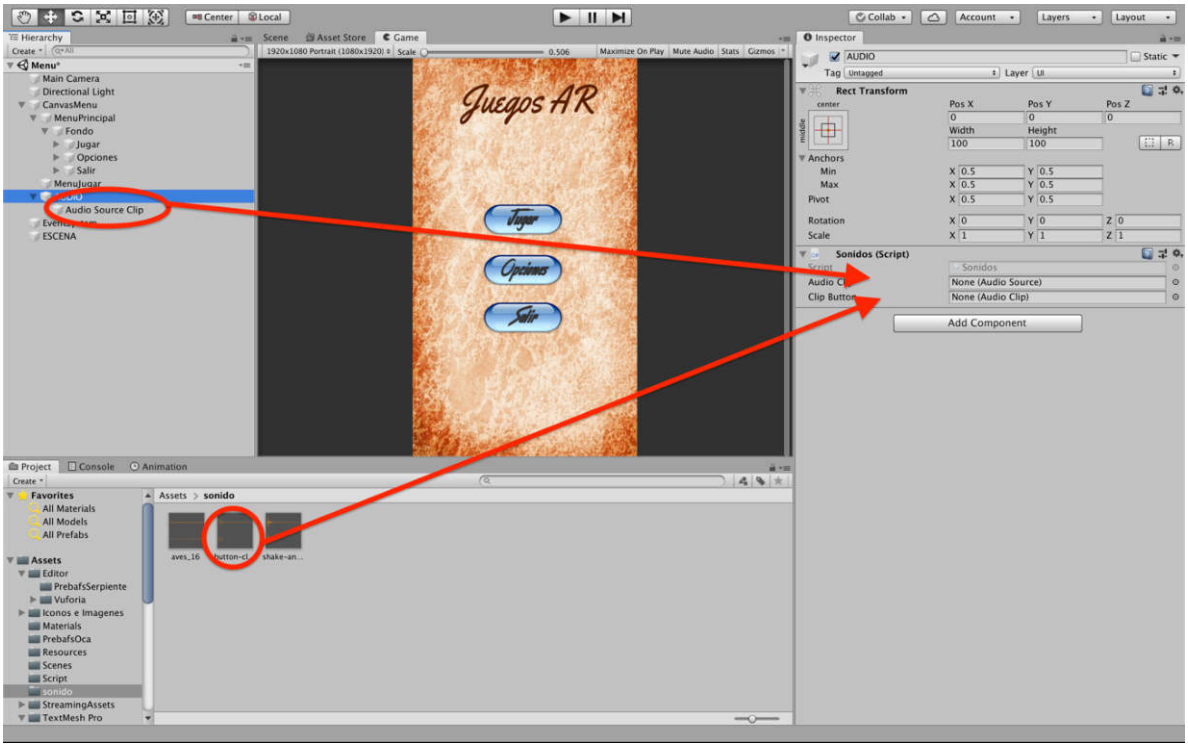


Figura 4.26: Asociar los componentes en las variables

Por último, es atribuirlo esta función de audio al botón “Jugar”, para ello se arrastra el objeto de AUDIO a la sección de “OnClick()” del botón “Jugar”, y ya asociarle la función en este caso la función “ClipButtonAudio”, tal y como aparece en la imagen siguiente.

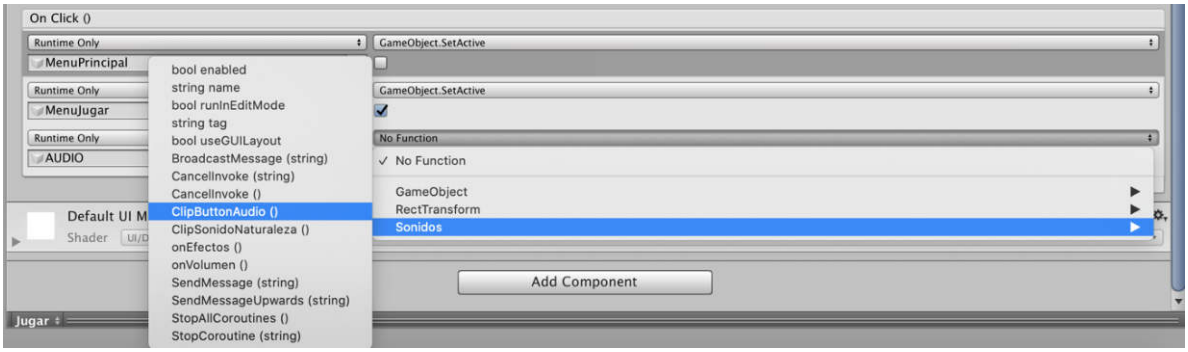


Figura 4.27: Seleccionar una función en el botón

- ***Opciones***

A este botón, se le va atribuir tres funciones, el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo), y la pantalla del siguiente menú (para activarlo). Para que se pueda asociar la siguiente pantalla del menú, se debe de crear en la ventana de jerarquía otro objeto vacío con el nombre” MenuOpciones” (para que se le pueda asociar este objeto a este botón), a la cual en el siguiente punto se detallará que botones y funciones realizan.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello se arrastra los dos objetos en la sección de “OnClick()” y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el botón Opciones cambie a la pantalla de “MenuOpciones” y se desactive la pantalla “MenuPrincipal”.

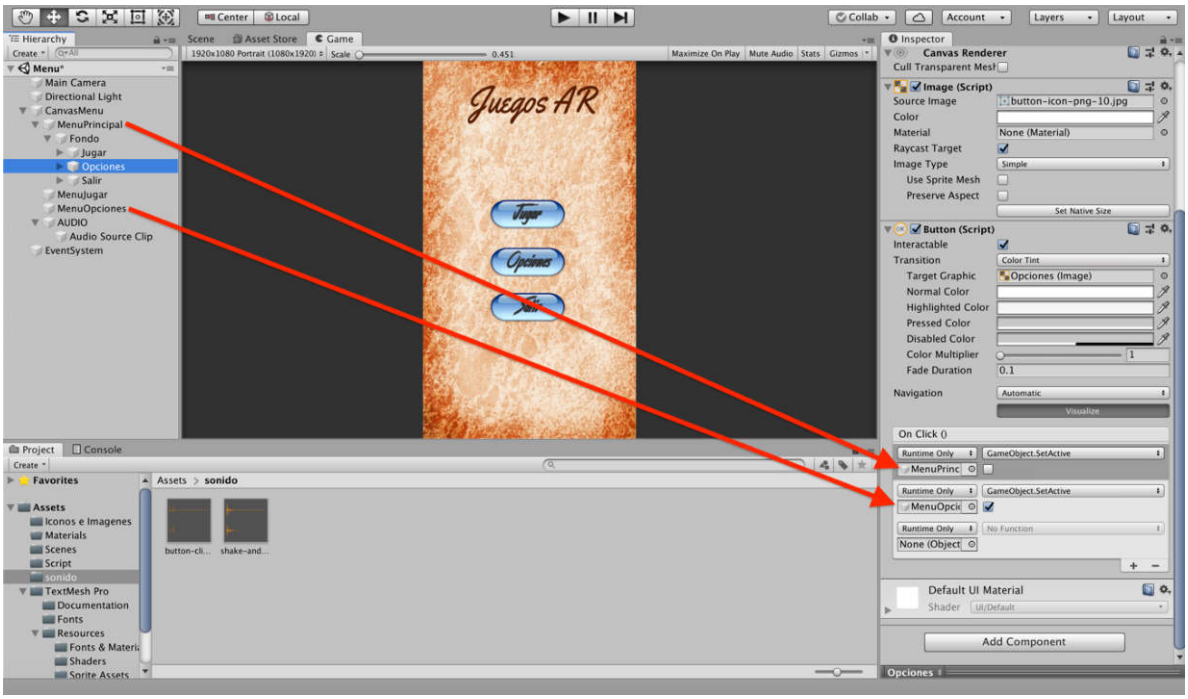


Figura 4.28: Cambiar de pantalla del botón Opciones

Por último, se le atribuye el audio al clicar en el botón. Como ya esta creado es realizar lo mismo que en el anterior botón.

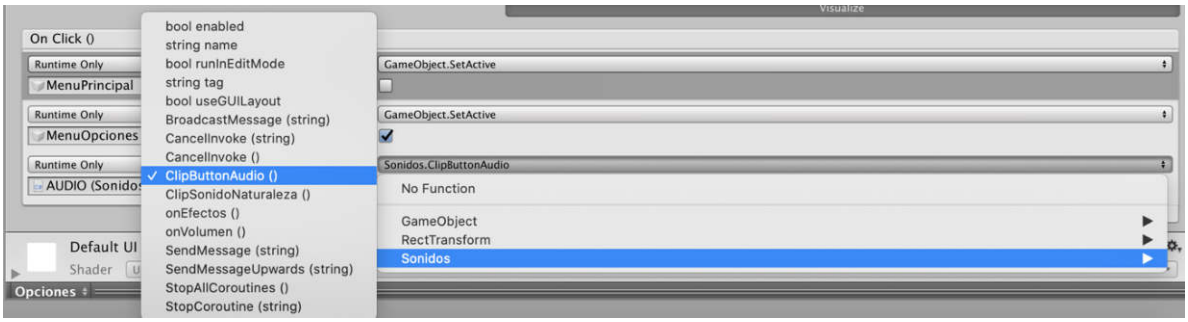


Figura 4.29: Seleccionar una función en el botón

- ***Salir***

En este último botón del Menú Principal, se le va atribuir dos funciones, el audio cuando se pulsa dicho botón, y la función de salir de la aplicación. Para la realización de salir de la aplicación, se creara un objeto vacío llamada “ESCENA”, en la cual se le atribuirá un Script nuevo llamada “BotonesMenu.cs”.

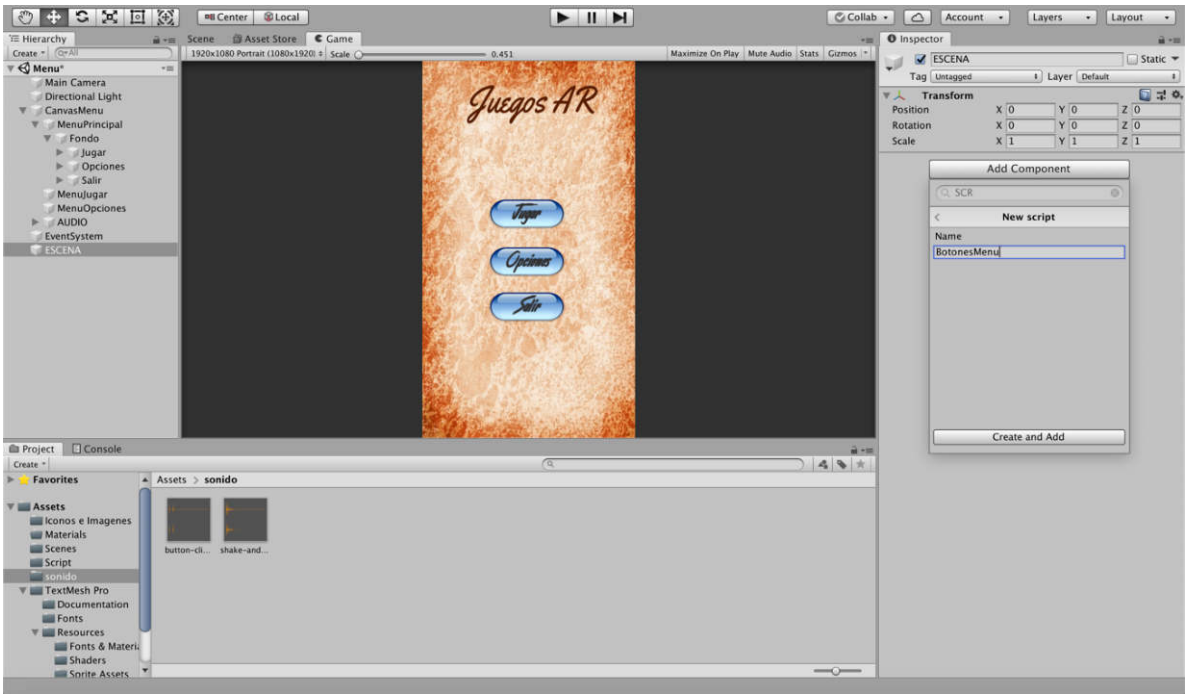


Figura 4.30: Nombre del script de las acciones de la escena Menú

En el script se crea la función, con la ayuda de las funciones que tiene Unity por defecto, para ellos accedemos a la clase “Application”, para así poder acceder al método “Quit”. Con la utilización de este método, nuestra aplicación se cerrará en cuanto se pulse este botón.

```
public void BotonSalir()
{
    Application.Quit();
}
```

Figura 4.31: Función para salir de la aplicación

Por tanto, en el botón Salir se le atribuye esta función, para ello se arrastra el objeto “Escena” a la sección “OnClick()” situado en la ventana de inspector del botón Salir.

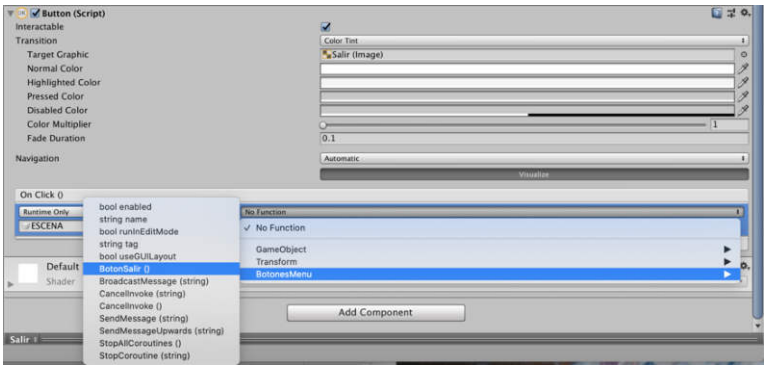


Figura 4.32: Seleccionar una función en el botón

Para asociarle el audio, se realiza de la misma manera que con el resto de botones. A continuación, en la imagen siguiente se muestra todas las acciones de este botón.

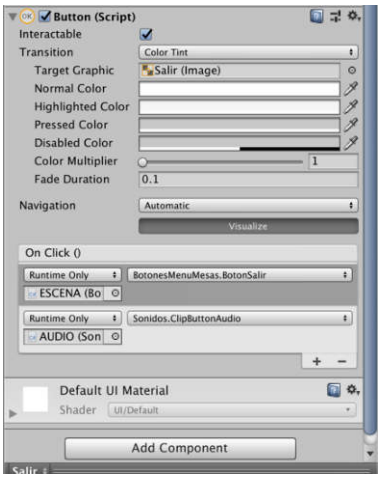


Figura 4.33: Funciones compuestas del botón Salir

4.6.2. Menú Jugar

Con respecto a la parte estética, se realiza de la misma forma que se ha hecho con el Menú Principal. Lo único que se va añadir a la parte estética es un icono para ir hacia atrás a una pantalla de menú, pero el procedimiento es el mismo para incorporar el icono.

A continuación, se muestra una imagen sobre la estética de la pantalla "MenuJugar":

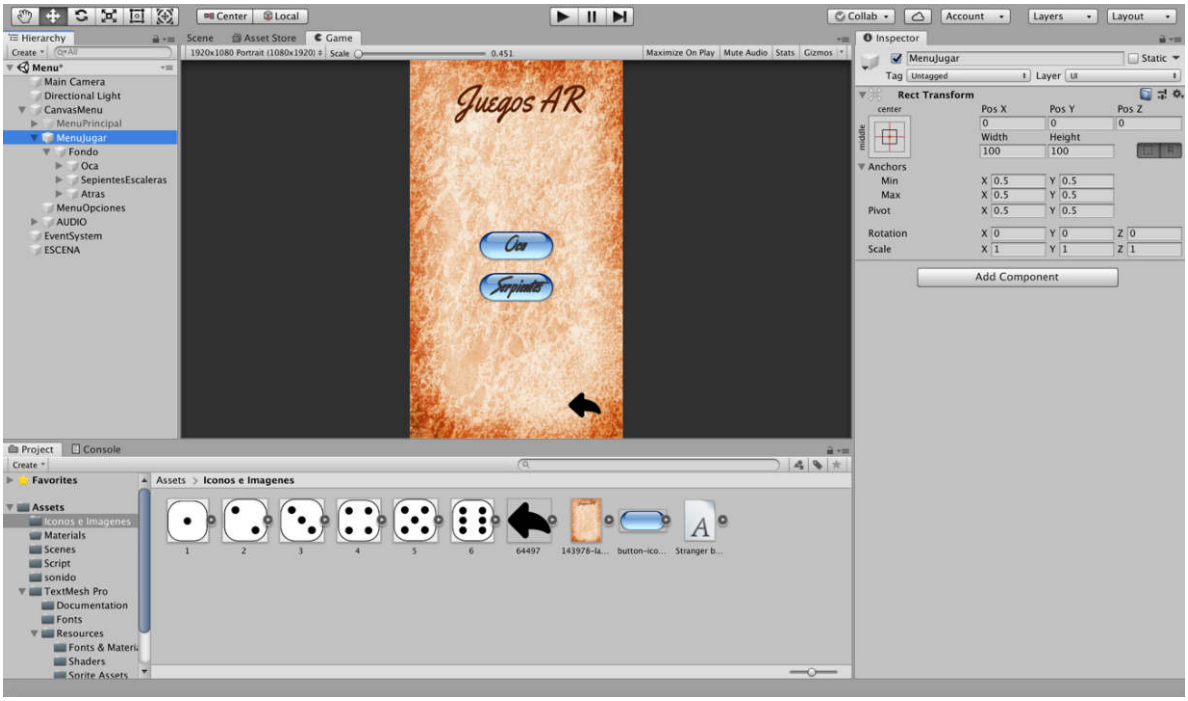


Figura 4.34: Vista final del Menú Jugar

4.6.2.1. Funciones de los botones

Se procederá en este apartado a las funciones que contiene los botones que presenta este menú.

- ***Oca***

A este botón, se le va atribuir tres funciones, el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo) y la pantalla del siguiente menú (para activarlo). Para que se pueda asociar la siguiente pantalla del menú, se debe de crear en la ventana de jerarquía otro objeto vacío con el nombre" MenuOca" (para que se le pueda asociar este objeto a este botón), a la cual en el siguiente punto se detallará que botones y funciones realizan.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello se arrastra los dos objetos en la sección de "OnClick()" y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el botón "Oca" cambie a la pantalla de "MenuOca" y se desactive la pantalla "MenuJugar".

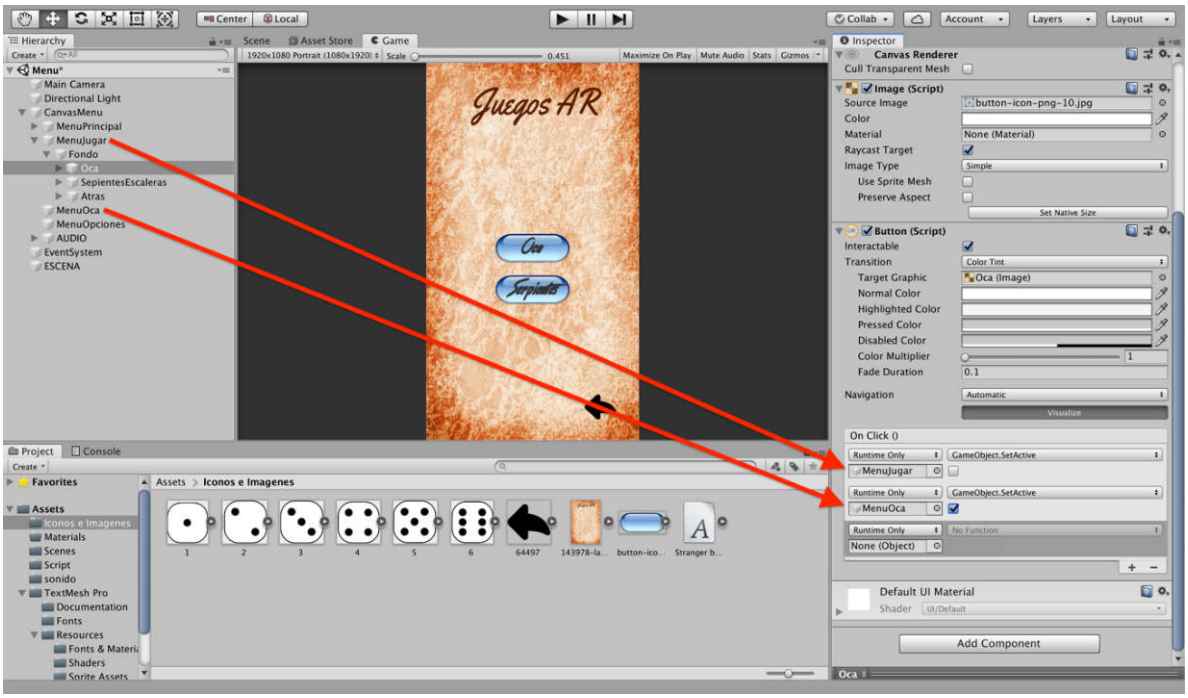


Figura 4.35: Cambiar de pantalla del botón Oca

Por último, se le atribuye el audio al clicar en el botón. Como ya esta creado es realizar lo mismo que en el anterior botón.

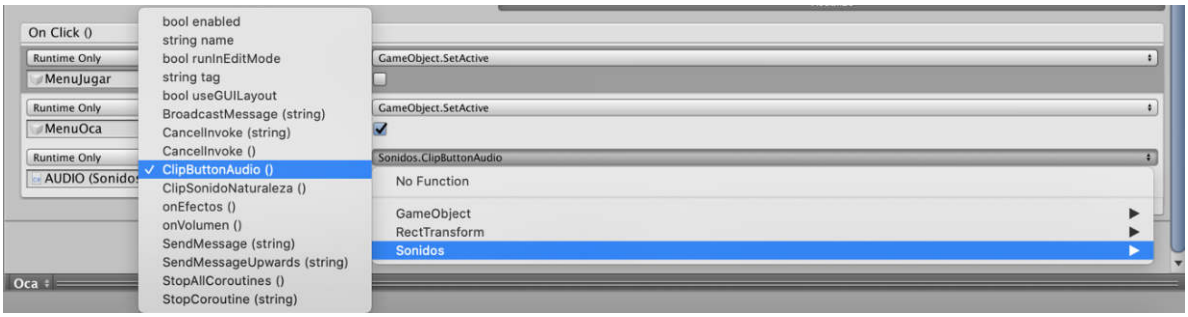


Figura 4.36: Selección de la función del audio

- ***Serpientes y Escaleras***

A este botón, se le va atribuir tres funciones, el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo) y la pantalla del siguiente menú (para activarlo). Para que se pueda asociar la siguiente pantalla del menú, se debe de crear en la ventana de jerarquía otro objeto vacío con el nombre" MenuSerpiente" (para que se le pueda asociar este objeto a este botón), a la cual en el siguiente punto se detallará que botones y funciones realizan.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello se arrastra los dos objetos en la sección de "OnClick()" y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el botón "Oca" cambie a la pantalla de "MenuSerpiente" y se desactive la pantalla "MenuJugar".

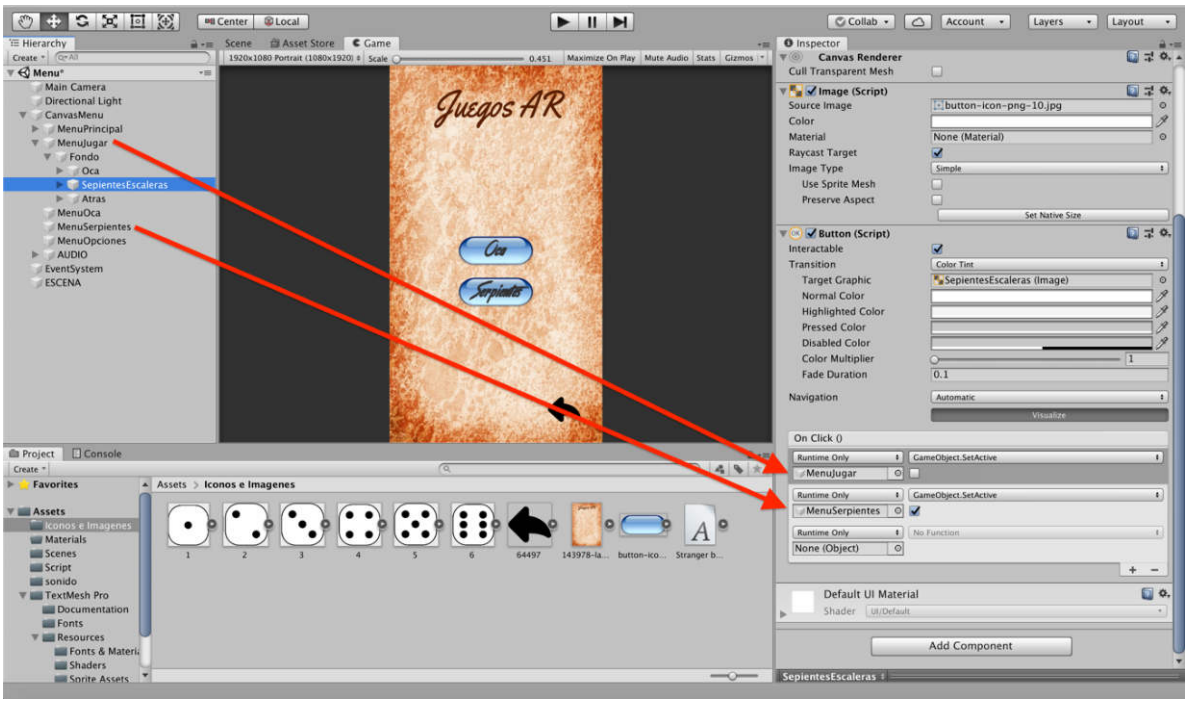


Figura 4.37: Cambiar de pantalla del Menú Serpientes

Por último, se le atribuye el audio al clicar en el botón, como ya esta creado es realizar lo mismo que en el anterior botón.

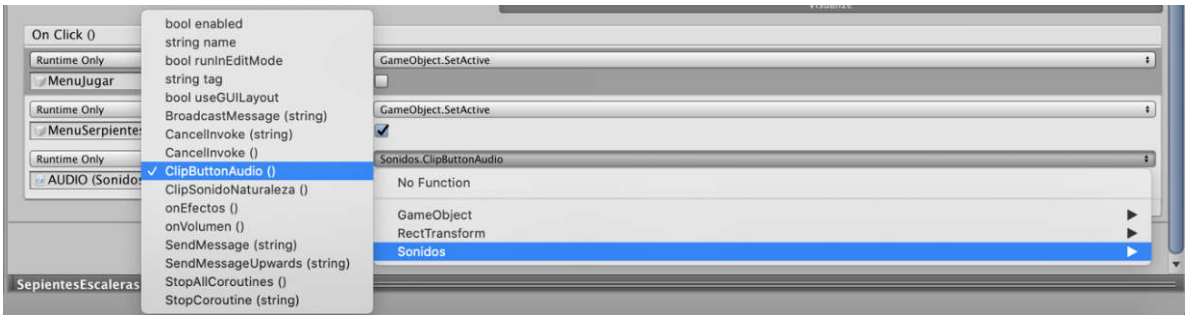


Figura 4.38: Selección de la función del audio

- ***Atrás***

Este botón estará constituido por tres funciones, el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo), y la pantalla del siguiente menú (para activarlo).

En este caso, se quiere que deje el “MenuJugar” por lo que habrá que desactivarlo, y activar el “MenuPrincipal” por lo que se tendrá que activar.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla. Para ello, se arrastra los dos objetos en la sección de “OnClick()” y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el icono de hacia atrás cambie a la pantalla de “MenuPrincipal” y se desactive la pantalla “MenuJugar”.

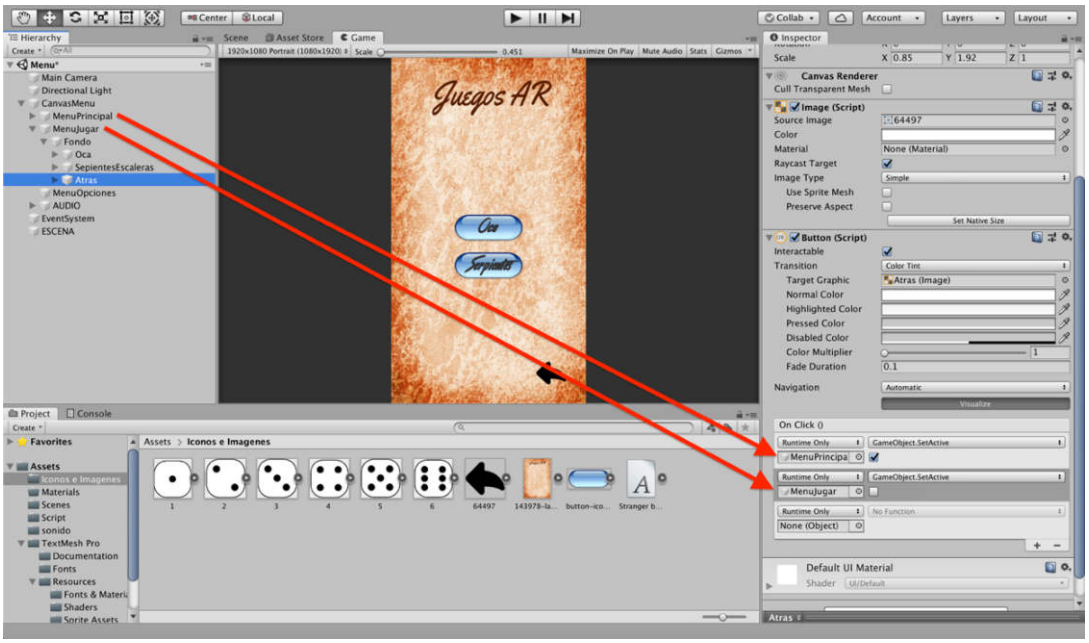


Figura 4.39: Cambiar de pantalla del botón Atrás

Por último para finalizar este botón, solo falta asignarle el audio para que se escuche cuando se pulsa sobre el botón. Simplemente, se arrastra el objeto de “AUDIO” a la sección “OnClick()”, a la cual se le atribuirá la función “ClipButtonAudio()”.

4.6.3. Menú Oca

Con respecto a la parte estética, se realiza de la misma forma que se ha hecho con el Menú Principal, con la diferencia de incorporar el icono de hacia atrás para que pueda volver a la pantalla anterior. A continuación, se muestra una imagen sobre la estética de la pantalla “MenuOca”:

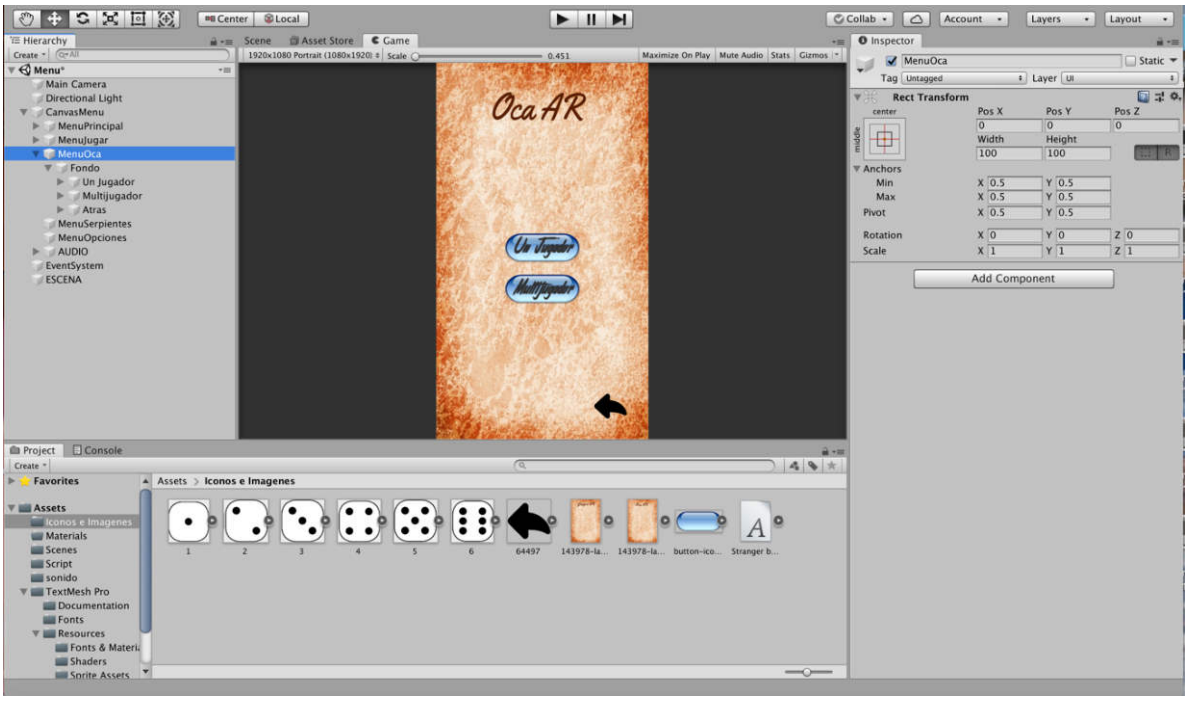


Figura 4.40: Vista final del Menú Oca

4.6.3.1. Funciones de los botones

Se procederá en este apartado a las funciones que contiene los botones que presenta este menú.

- ***Un Jugador***

En este botón se le asignará dos funciones, uno de las funciones que cuando pulse sobre dicho botón salte la escena “RestauranteMesaOca” que es la que el usuario indica el restaurante y la mesa donde se encuentra, y la otra función es incluirle el audio cuando se pulsa el botón.

Para asignarle al botón la función que salte a la escena “RestauranteMesaOca”, se debe de incluir dicha función en el script “BotonesMenu” que esta asignado al objeto

“ESCENA”. Para ello, en el script se le debe de incorporar la librería UnityEngine.SceneManagement. La función que realizará el salto a otra escena se va a llamar “BotonOca”, el cual se utilizará las funciones que Unity tiene por defecto, accediendo a la clase “SceneManager”, para así, poder acceder al método “LoadScene”, y se le dirá entre comillas que escena se quiere que se cargue, en nuestro caso indicara “RestauranteMesasOca”.

```
public void BotonOca()
{
    SceneManager.LoadScene("RestauranteMesaOca");
}
```

Figura 4.41: Función de cambio de escena del botón Un Jugador

Por tanto, para incluirle la función de irse a otra escena, arrastramos el objeto “ESCENA” a la sección “OnClick()”, lo que permite acceder al script y atribuirle la función “BotonOca()”, tal y como se muestra en la siguiente imagen:

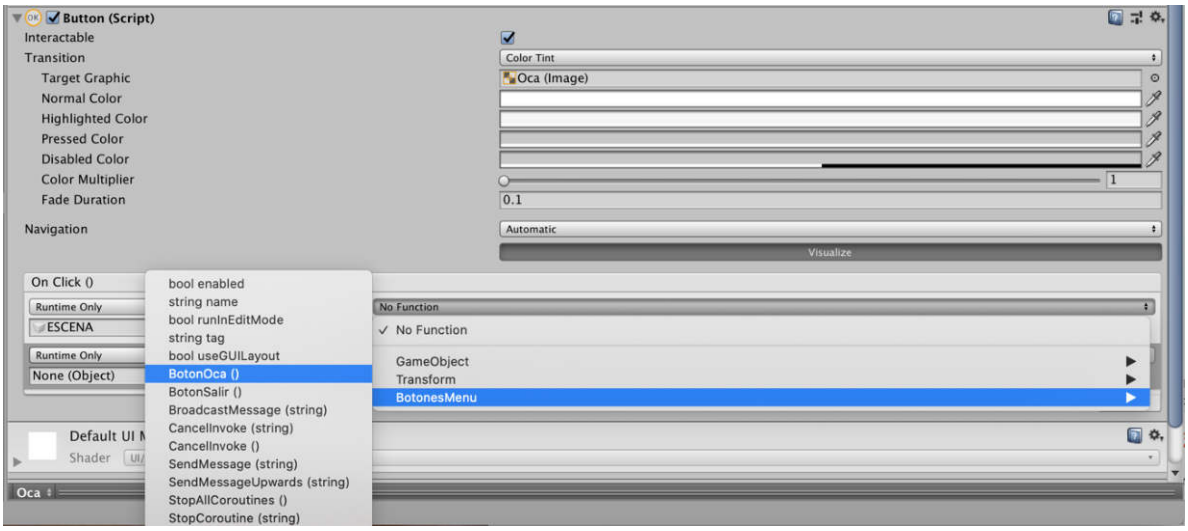


Figura 4.42: Selección de la función del cambio de escena

Para finalizar este botón, solo faltaría asignarle el audio para que se escuche cuando se pulsa sobre el botón. Simplemente, se arrastra el objeto de “AUDIO” a la sección “OnClick()”, a la cual se le atribuirá la función “ClipButtonAudio()”.

- ***Multijugador***

En este botón se le asignará dos funciones al igual que en el botón de “Un Jugador”, uno de las funciones que cuando pulse sobre dicho botón salte la escena “RestauranteMesaOcaMulti”, que es la que el usuario indica el restaurante y la mesa donde se encuentra, y la otra función es incluirle el audio cuando se pulsa el botón.

```
public void BotonOcaMulti()
{
    SceneManager.LoadScene("RestauranteMesaOcaMulti");
}
```

Figura 4.43: Función de cambio de escena del botón Multijugador

A modo resumen, se mostrará en una imagen las dos funciones que se le han incluido en este botón, ya que se procede de la misma manera que en botón “Un Jugador”.

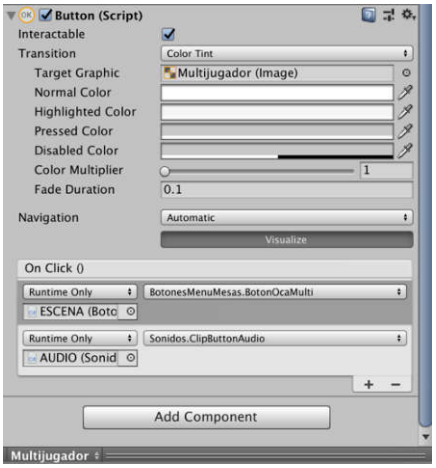


Figura 4.44: Funciones compuestas del botón Multijugador

- ***Atrás***

Este botón estará constituido por tres funciones, el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo), y la pantalla del siguiente menú (para activarlo).

En este caso, se quiere que deje el “MenuOca” por lo que habrá que desactivarlo, y activar el “MenuJugar” por lo que se tendrá que activar.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello se arrastra los dos objetos en la sección de “OnClick()” y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el icono de hacia atrás cambie a la pantalla de “MenuJugar” y se desactive la pantalla “MenuOca”.

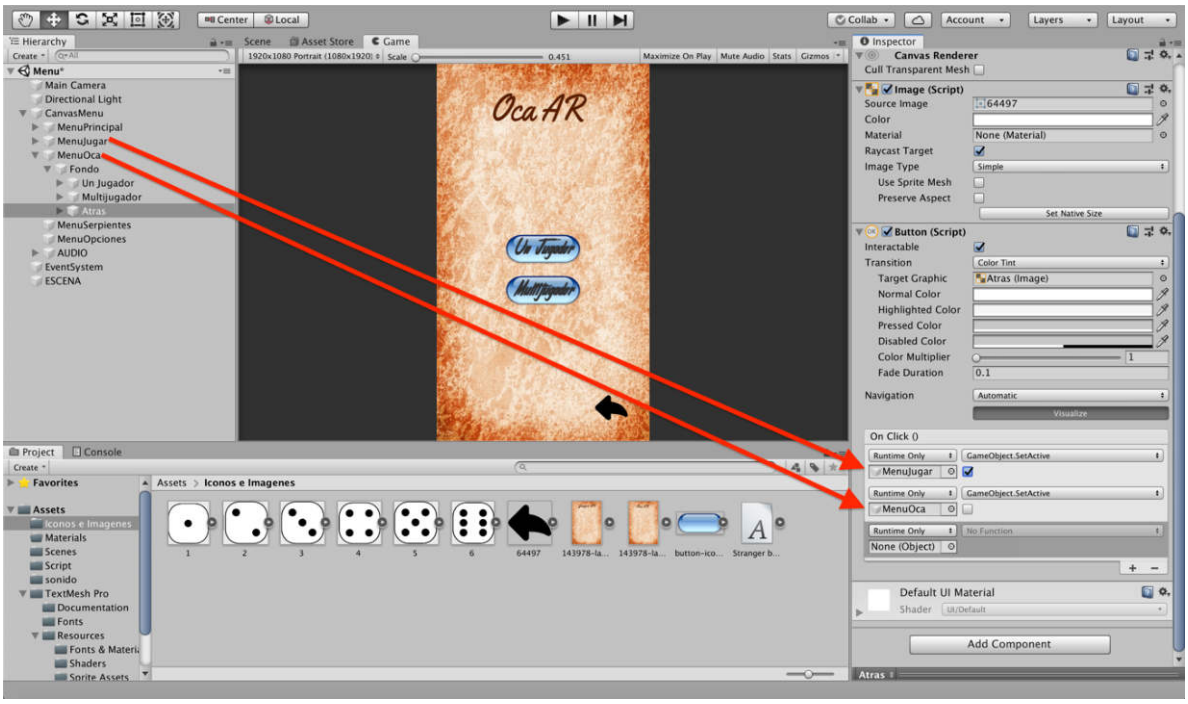


Figura 4.45: Cambiar de pantalla del botón Atrás

Por último para finalizar este botón, solo falta asignarle el audio para que se escuche cuando se pulsa sobre el botón. Simplemente, se arrastra el objeto de “AUDIO” a la sección “OnClick()”, a la cual se le atribuirá la función “ClipButtonAudio()”.

4.6.4. Menú Serpientes y Escaleras

Con respecto a la parte estética, se realiza de la misma forma que se ha hecho con el Menú Principal, con la diferencia de incorporar el icono de hacia atrás para que pueda volver a la pantalla anterior. A continuación, se muestra una imagen sobre la estética de la pantalla “MenuSerpientes”:

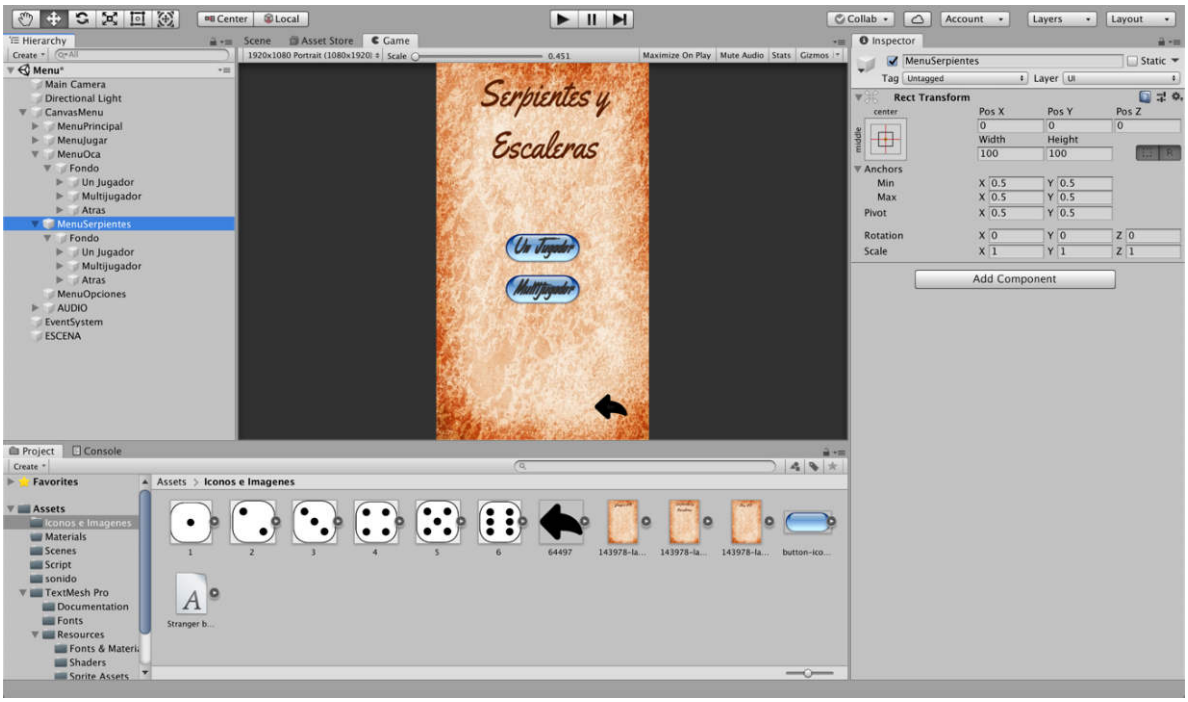


Figura 4.46: Vista final del Menú Serpientes

4.6.4.1. Funciones de los botones

Se procederá en este apartado a las funciones que contiene los botones que presenta este menú.

- ***Un Jugador***

En este botón se le asignará dos funciones, uno de las funciones que cuando pulse sobre dicho botón salte la escena “RestauranteMesaSerpiente” que es la que el usuario indica el restaurante y la mesa donde se encuentra, y la otra función es incluirle el audio cuando se pulsa el botón.

Para asignarle al botón la función que salte a la escena Inicio, se debe de incluir dicha función en el script “BotonesMenu” que esta asignado al objeto “ESCENA”. Para ello, en el script se le debe de incorporar la librería UnityEngine.SceneManagement. La función que realizará el salto a otra escena se va a llamar “BotonSerpiente”, en la cual se utilizará las funciones que Unity tiene por defecto, accediendo a la clase “SceneManager”, para así poder acceder al método “LoadScene”, y se le dirá entre comillas que escena se quiere que se cargue, en nuestro caso indicara “RestauranteMesaSerpiente”.

```
public void BotonSerpiente()
{
    SceneManager.LoadScene("RestauranteMesaSerpiente");
}
```

Figura 4.47: Función de cambio de escena del botón Un Jugador

Por tanto, para incluirle la función de irse a otra escena, se arrastra el objeto “ESCENA” a la sección “OnClick()”, lo que permite acceder al script y atribuirle la función “BotonSerpiente()”, además del audio para que se escuche cuando se pulsa sobre el botón, para ello se le atribuirá la función “ClipButtonAudio()” también a la sección “OnClick()”, tal y como se puede ver en la siguiente imagen:

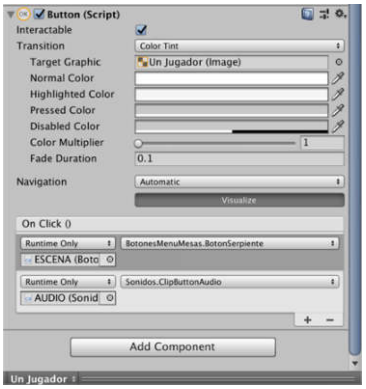


Figura 4.48: Funciones compuestas del botón Un Jugador

- ***Multijugador***

En este botón se le asignará dos funciones al igual que en el botón de “Un Jugador”, uno de las funciones que cuando pulse sobre dicho botón salte la escena “SerpienteMulti” que es la que el usuario indica el restaurante y la mesa donde se encuentra, y la otra función es incluirle el audio cuando se pulsa el botón.

```
public void BotonSerpienteMulti()
{
    SceneManager.LoadScene("RestauranteMesaSerpienteMulti");
}
```

Figura 4.49: Función del cambio de escena del botón Multijugador

A modo resumen, se mostrara en una imagen las dos funciones que se le han incluido en este botón, ya que se procede de la misma manera.

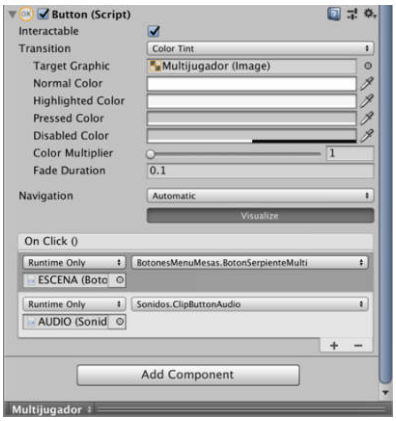


Figura 4.50: Funciones compuestas del botón Multijugador

- ***Atrás***

Este botón estará constituido por tres funciones, el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo), y la pantalla del siguiente menú (para activarlo).

En este caso, se quiere que deje el "MenuSerpiente" por lo que habrá que desactivarlo, y activar el "MenuJugar" por lo que se tendrá que activar.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello se arrastra los dos objetos en la sección de "OnClick()" y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el icono de hacia atrás cambie a la pantalla de "MenuJugar" y se desactive la pantalla "MenuSerpiente".

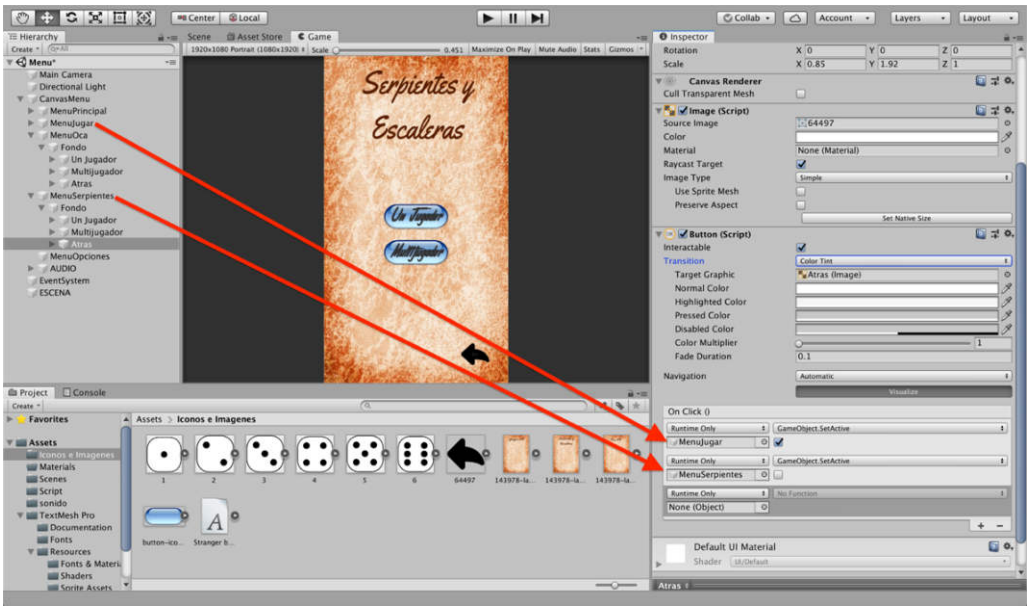


Figura 4.51: Cambiar de pantalla del botón Atrás

Por último para finalizar este último botón del "MenuSerpientes", solo falta asignarle el audio para que se escuche cuando se pulsa sobre el botón. Simplemente, se arrastra el objeto de "AUDIO" a la sección "OnClick()", a la cual se le atribuirá la función "ClipButtonAudio()".

4.6.5. Menú Opciones

Con respecto a la parte estética, el procedimiento a seguir es el mismo que se ha realizado con el Menú Principal, con la diferencia de incorporar el icono de hacia atrás para que pueda volver a la pantalla anterior. A continuación, se muestra una imagen sobre la estética de la pantalla "MenuOpciones":

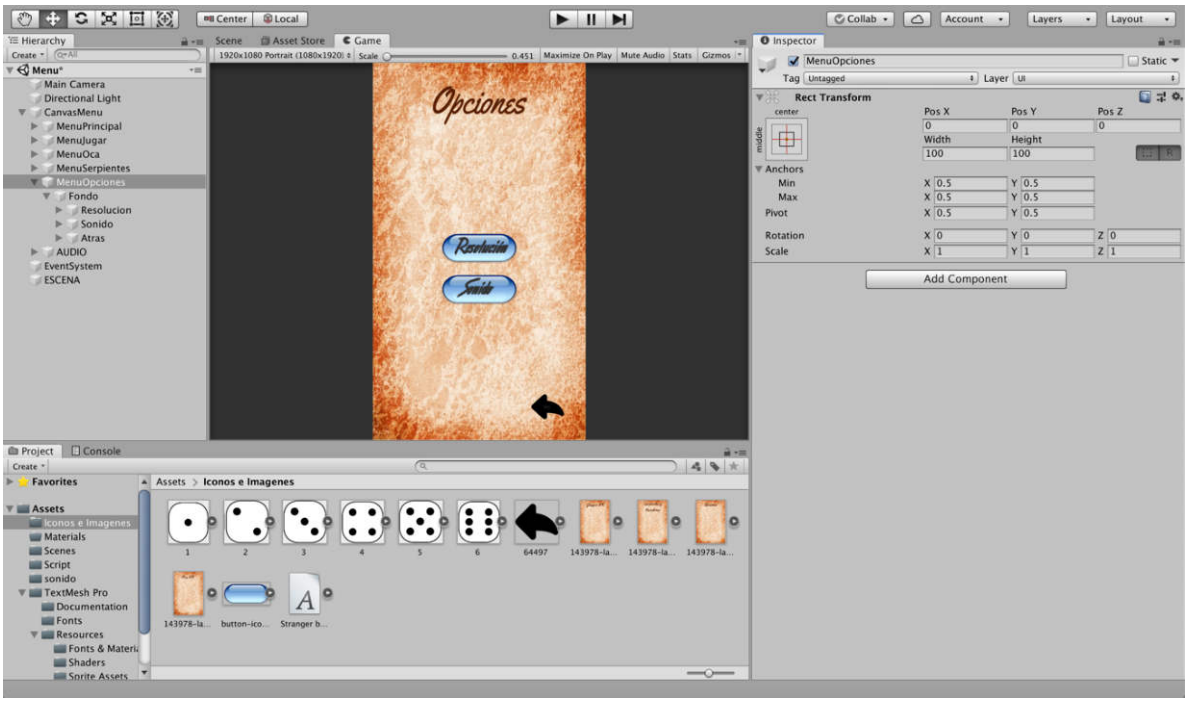


Figura 4.52: Vista final del Menú Opciones

4.6.5.1. Funciones de los botones

Se procederá en este apartado a las funciones que contiene los botones que presenta este menú.

- ***Resolución***

Con lo que respecta en este botón estará constituido de tres funciones**,** el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo) y la pantalla del siguiente menú (para activarlo). Para que se pueda asociar la siguiente pantalla del menú, se debe de crear en la ventana de jerarquía otro objeto vacío con el nombre "MenuResolucion" (para que se le pueda asociar este objeto a este botón), a la cual en el 4.6.6 punto se detallará que botones y funciones realizan.

En este caso, se quiere que deje el "MenuOpciones" por lo que habrá que desactivarlo, y activar el "MenuResolucion" por lo que se tendrá que activar.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello se arrastra los dos objetos en la sección de "OnClick()" y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el botón cambie a la pantalla de "MenuResolucion" y se desactive la pantalla "MenuOpciones".

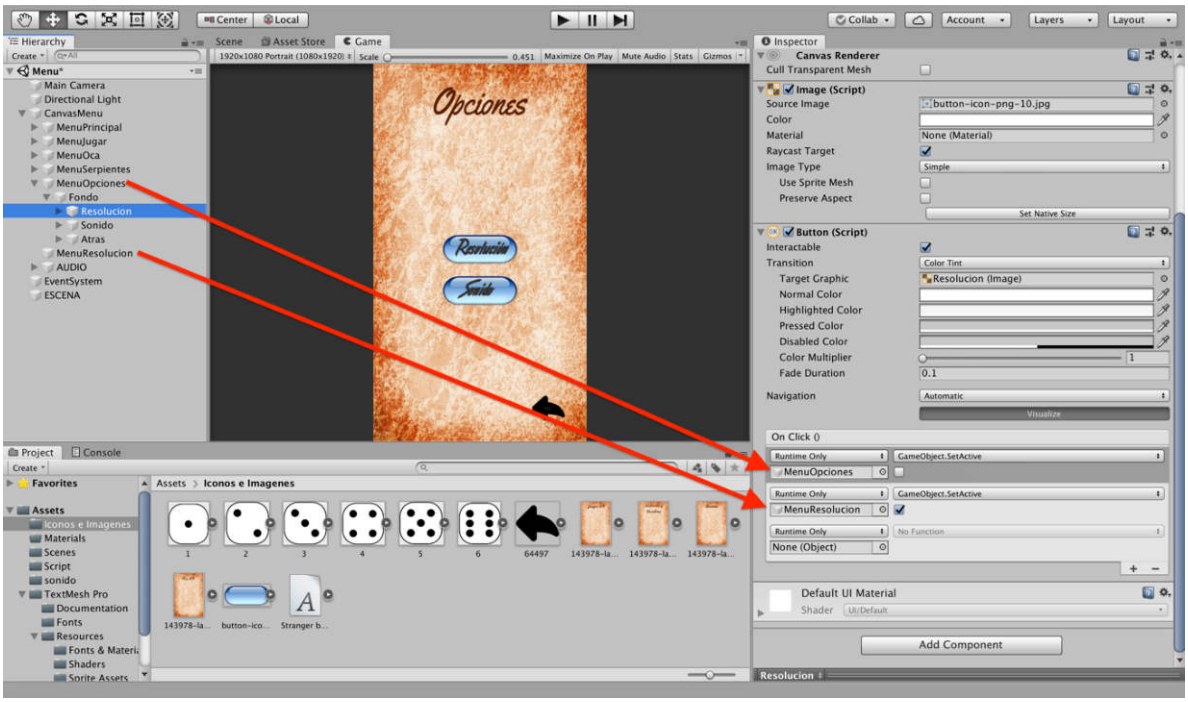


Figura 4.53: Cambiar de pantalla del botón Resolución

Además, se le debe de atribuir el sonido cuando se clica sobre el botón, para ello se arrastra el objeto AUDIO a la sección “OnClick()” situado en la ventana inspector del botón Resolución, y se selecciona la función “ClipButtonAudio”.

- ***Sonido***

Con lo que respecta en este botón estará constituido de tres funciones**,** el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo) y la pantalla del siguiente menú (para activarlo). Para que se pueda asociar la siguiente pantalla del menú, se debe de crear en la ventana de jerarquía otro objeto vacío con el nombre “MenuSonido” (para que se le pueda asociar este objeto a este botón), a la cual en punto 4.6.7 se detallará que botones y funciones realizan.

En este caso, se quiere que deje el “MenuOpciones” por lo que habrá que desactivarlo, y activar el “MenuSonido” por lo que se tendrá que activar.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello se arrastra los dos objetos en la sección de “OnClick()” y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el botón cambie a la pantalla de “MenuSonido” y se desactive la pantalla “MenuOpciones”.

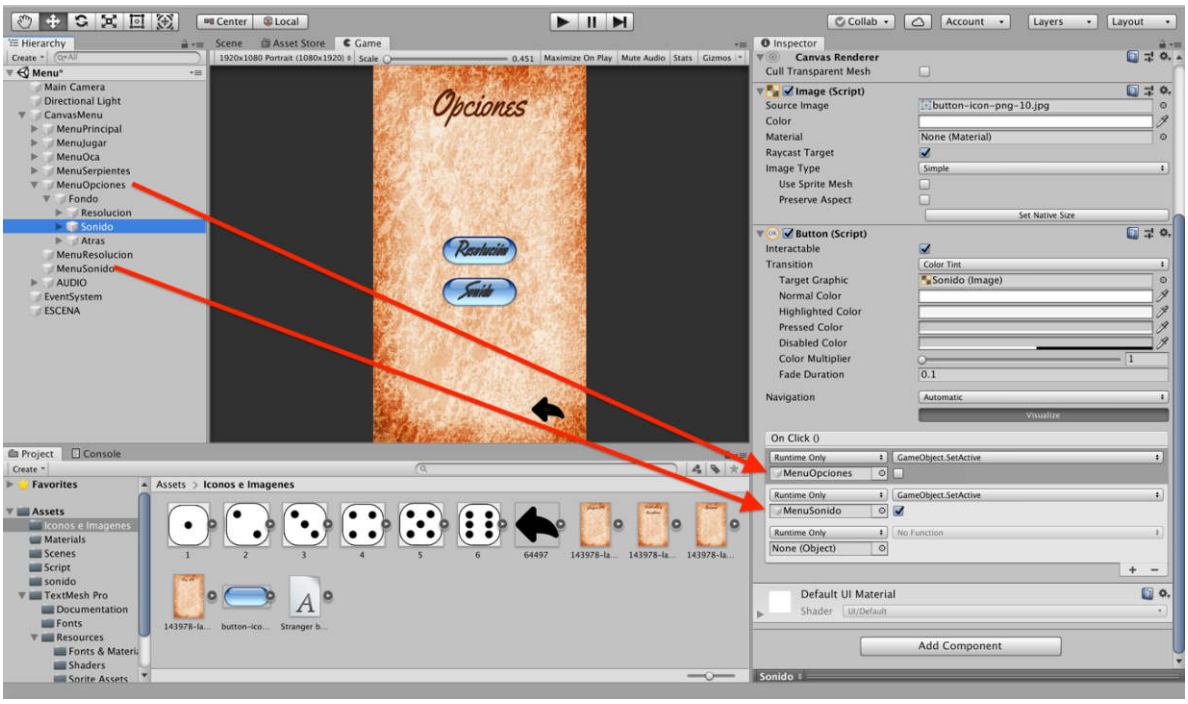


Figura 4.54: Cambiar de pantalla del botón Sonido

Además, se le debe de atribuir el sonido cuando se clica sobre el botón, para ello se arrastra el objeto AUDIO a la sección “OnClick()” situado en la ventana inspector del botón Resolución, y se selecciona la función “ClipButtonAudio”.

- ***Atrás***

Este botón estará constituido por tres funciones, el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo) y la pantalla del siguiente menú (para activarlo).

En este caso, se quiere que deje el “MenuOpciones” por lo que habrá que desactivarlo, y activar el “MenuPrincipal” por lo que se tendrá que activar.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello arrastramos los dos objetos en la sección de “OnClick()” y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el icono de hacia atrás cambie a la pantalla de “MenuPrincipal” y se desactive la pantalla “MenuOpciones”.

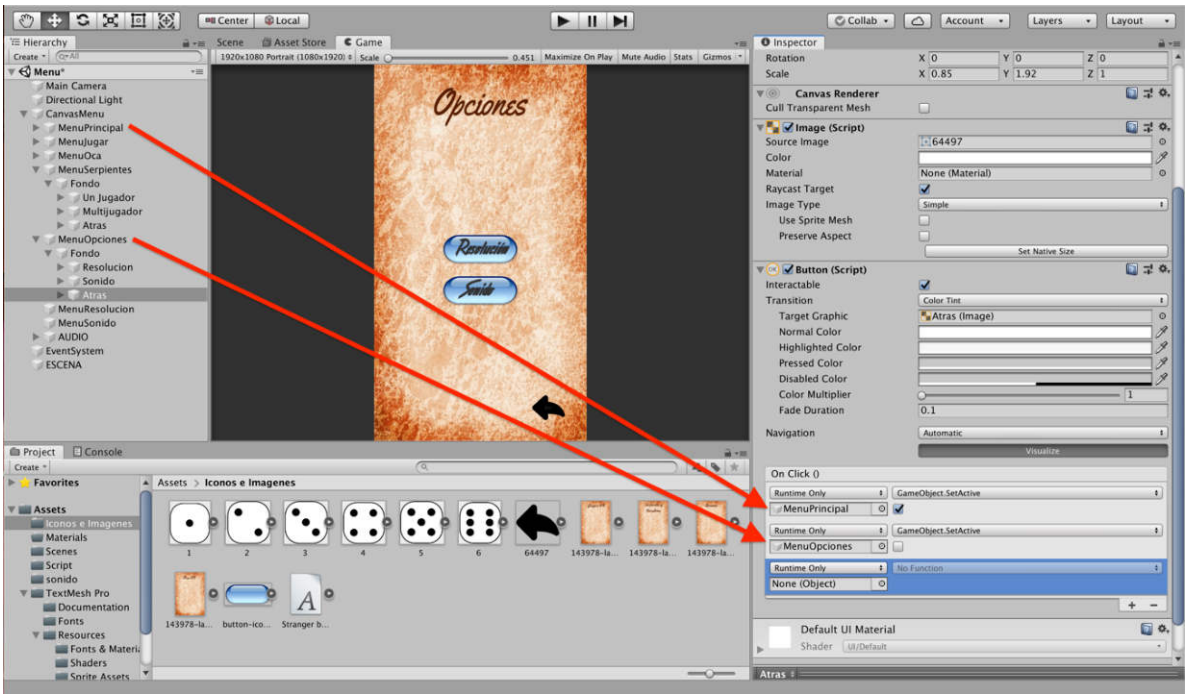


Figura 4.55: Cambiar de pantalla de botón Atrás

Por último para finalizar este botón, solo falta asignarle el audio para que se escuche cuando se pulsa sobre el botón. Simplemente, se arrastra el objeto de AUDIO a la sección "OnClick()", a la cual se le atribuirá la función "ClipButtonAudio()".

4.6.6. Menú Resolución

Con respecto a la parte estética, se procede de la misma forma que se ha hecho con el Menú Principal, con una diferencia a la cual ya se ha comentado en varios apartados de incorporar el icono de hacia atrás para que pueda volver a la pantalla anterior. A continuación, se muestra una imagen sobre la estética de la pantalla "MenuResolucion":

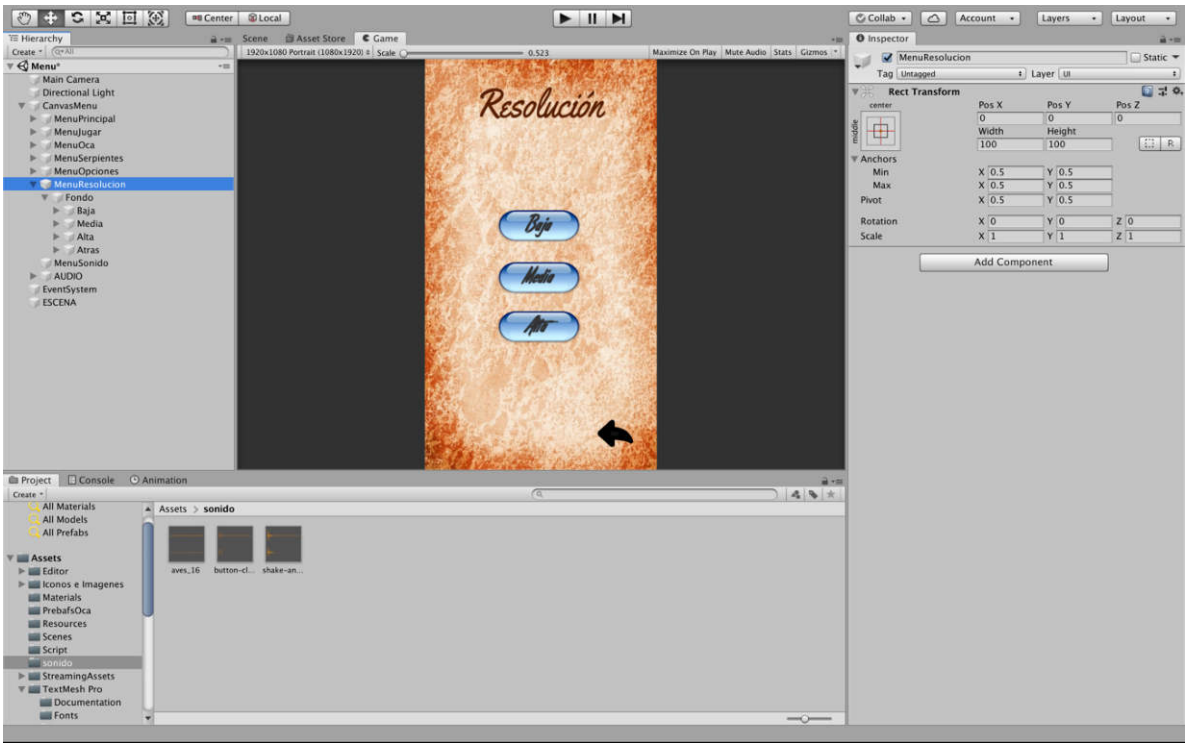


Figura 4.56: Vista final del Menú Resolución

4.6.6.1. Funciones de los botones

Se procederá en este apartado a las funciones que contiene los botones que presenta este menú.

- ***Baja***

En ese primer botón del Menú Resolución, se le asignará dos funciones, una de esas función es introducir el audio cuando pulsa sobre este botón, y la otra función para activar un tipo de calidad especificado, en este caso, una calidad baja, para que cuando se pulse sobre este botón tenga la calidad aplicada.

Para que a nuestro juego se le pueda aplicar a este botón una calidad baja, se debe de incluir en el script “BotonesMenu.cs” dicha función descripta, se recuerda que este script se encuentra ubicado en el objeto “Escena”. La función que realizará el cambio a una calidad gráfica baja se llamará ”BotonResolucionBaja()” que utilizará las funciones que Unity tiene por defecto, la clase que utilizará esta función se llama “QualitySetting”, a la cual se accederá al método “currentLevel” y se igualará a el nivel de calidad deseado, utilizando QualityLevel y especificando la calidad, en este caso “Fastest”. A continuación, se muestra la función:

```
public void BotonResolucionBaja()
{
    QualitySettings.currentLevel = QualityLevel.Fastest;
}
```

Figura 4.57: Función del cambio de calidad Baja

Una vez que se tiene la función realizada, ya se le puede asignar al botón, para ello, se arrastra el objeto “ESCENA” a la sección de “OnClick()” situado en la ventana de inspector del botón “Baja”, en la cual una vez ya contenido el objeto, se puede acceder a la función que se ha creado anteriormente, además de incluir la función de audio para que suene al clicar dicho botón, tal y como se muestra a continuación.

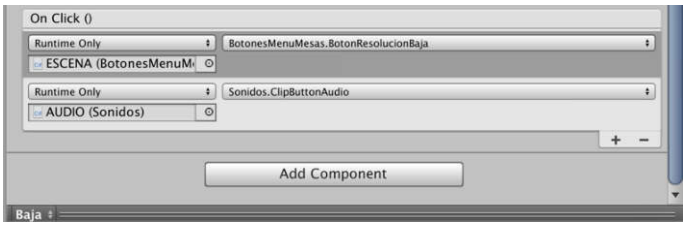


Figura 4.58: Selección de la función del cambio de calidad Baja

- ***Media***

En este botón se le asignará dos funciones, una de esas función es introducir el audio cuando pulsa sobre este botón, y la otra función para activar un tipo de calidad especificado, en este caso, una calidad media, para que cuando se pulse sobre este botón tenga la calidad aplicada.

Para que a nuestro juego se le pueda aplicar a este botón una calidad baja, se debe de incluir en el script “BotonesMenu.cs” dicha función descripta, recuerda que este script se encuentra ubicado en el objeto “Escena”. La función que realizará el cambio a una calidad gráfica baja se llamará ”BotonResolucionMedia()” que utilizará las funciones que Unity tiene por defecto, la clase que utilizará esta función se llama “QualitySetting”, a la cual se accederá al método “currentLevel” y se igualará a el nivel de calidad deseado, utilizando QualityLevel y especificando la calidad, en este caso “Good”. A continuación, se muestra la función:

```
public void BotonResolucionMedia()
{
    QualitySettings.currentLevel = QualityLevel.Good;
}
```

Figura 4.59: Función del cambio de calidad Media

Una vez que se tiene la función realizada, ya se le puede asignar al botón, para ello, se arrastra el objeto “Escena” a la sección de “OnClick()” situado en la ventana de inspector del botón “Media”, en la cual una vez ya contenido el objeto, se puede acceder a la función que se ha creado anteriormente, además de incluir la función de audio para que suene al clicar dicho botón, tal y como se muestra a continuación.

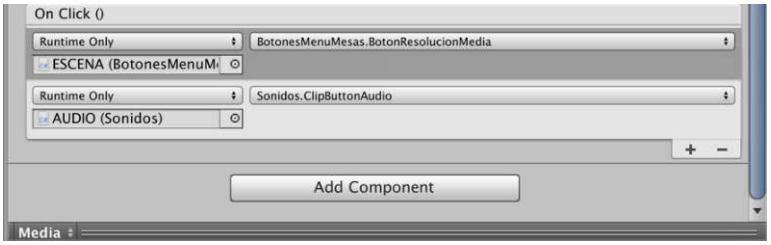


Figura 4.60: Selección de la función del cambio de calidad Media

- ***Alta***

En este botón se le asignará dos funciones, una de esas función es introducir el audio cuando pulsa sobre este botón, y la otra función para activar un tipo de calidad especificado, en este caso, una calidad alta, para que cuando se pulse sobre este botón tenga la calidad aplicada.

Para que a nuestro juego se le pueda aplicar a este botón una calidad baja, se debe de incluir en el script “BotonesMenu.cs” dicha función descripta, recuerda que este script se encuentra ubicado en el objeto “Escena”. La función que realizará el cambio a una calidad gráfica baja se llamará ”BotonResolucionAlta()” que utilizará las funciones que Unity tiene por defecto, la clase que utilizará esta función se llama “QualitySetting”, a la cual se accederá al método “currentLevel” y se igualará a el nivel de calidad deseado, utilizando QualityLevel y especificando la calidad, en este caso “Beautiful”. A continuación, se muestra la función:

```
public void BotonResolucionAlta()
{
    QualitySettings.currentLevel = QualityLevel.Beautiful;
}
```

Figura 4.61: Función del cambio de calidad Alta

Una vez que se tiene la función realizada, ya se le puede asignar al botón, para ello, se arrastra el objeto “Escena” a la sección de “OnClick()” situado en la ventana de inspector del botón “Media”, en la cual una vez ya contenido el objeto, se puede acceder a la función que se ha creado anteriormente, además de incluir la función de audio para que suene al clicar dicho botón, tal y como se muestra a continuación.

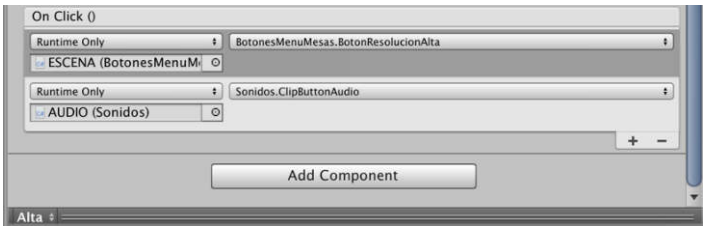


Figura 4.62: Selección de la función del cambio de calidad Alta

- ***Atrás***

Este botón estará constituido por tres funciones, el audio cuando se pulsa dicho botón, la pantalla propia de este menú (para desactivarlo) y la pantalla del siguiente menú (para activarlo).

En este caso, se quiere que deje el "MenuResolucion" por lo que habrá que desactivarlo, y activar el "MenuOpciones" por lo que se tendrá que activar.

En la siguiente imagen, se va añadir estos dos objetos para que se pueda cambiar de pantalla, para ello se arrastra los dos objetos en la sección de "OnClick()" y se le atribuye una función para cada objeto, uno activado y otro desactivado, con la finalidad de que cuando pulses sobre el icono de hacia atrás cambie a la pantalla de "MenuOpciones" y se desactive la pantalla "MenuResolucion".

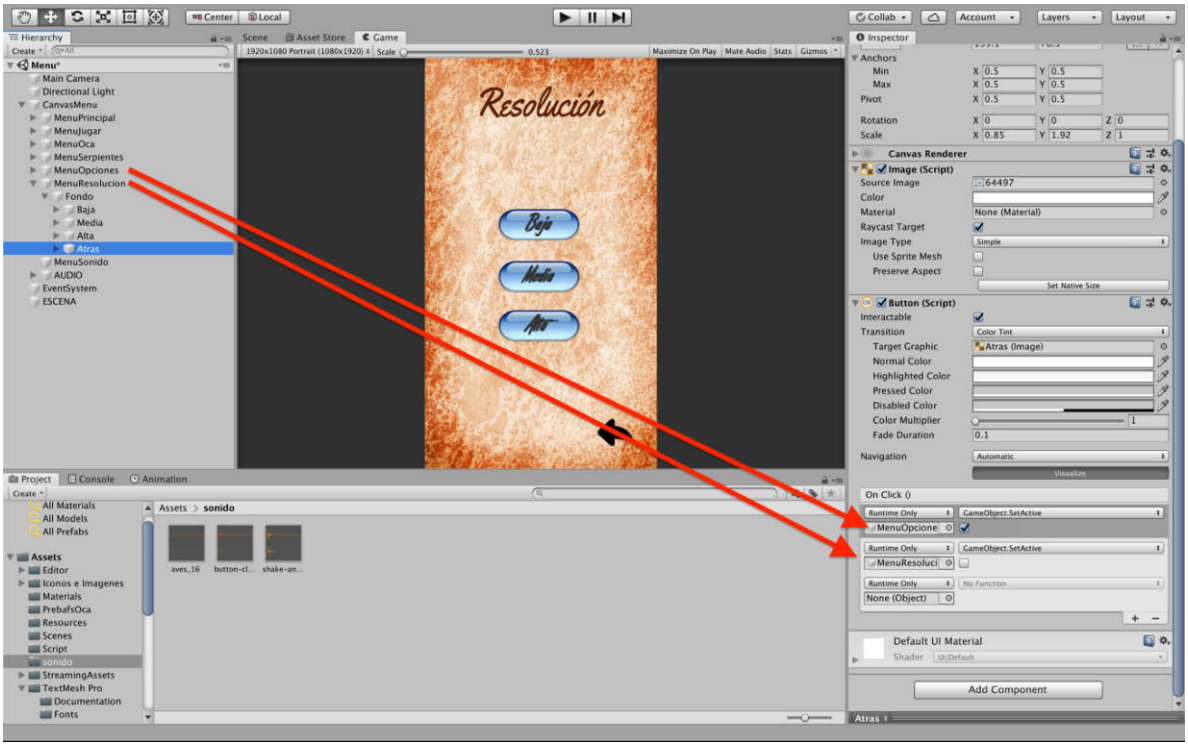


Figura 4.63: Cambiar de pantalla del botón Atrás

Por último para finalizar este botón, solo falta asignarle el audio para que se escuche el sonido cuando se pulsa sobre el botón. Simplemente, se arrastra el objeto de AUDIO a la sección “OnClick()”, a la cual se le atribuirá la función “ClipButtonAudio()”.

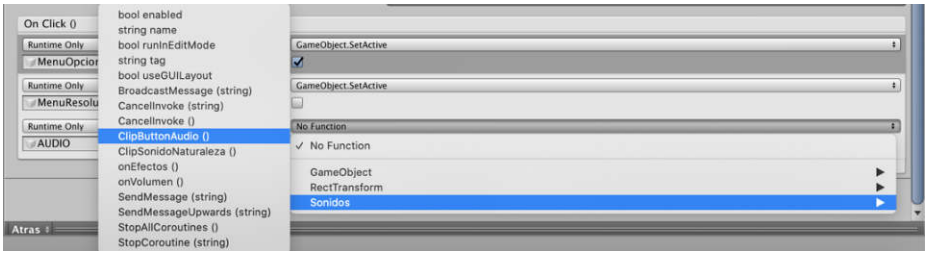


Figura 4.64: Selección de la función de sonido del botón

4.6.7. Menú Sonido

Con respecto a la parte estética, se va a cambiar los botones por barras de ajustes, que tendrá un valor entre 0 y 1, estas barras se llaman ”Slider”, se incluirán dos uno para los efectos (modificará el volumen del sonido cuando se pulse sobre algún botón) y otro de sonido (modificará el volumen de la música en la intro del menú).

Para la creación de este menú, se crea un objeto vacío y se le cambia el a “MenuSonido”, en el cual dentro de ella, va a estar formada por el mismo tipo de fondo que en el resto de menú, dos objetos de tipo Text y dentro de estos dos objetos, estará formada por dos “Sliders”, y por último el botón de ir hacia atrás para cambiar a la anterior pantalla. Para insertar los Slider, se puede ver en la siguiente foto como se inserta:

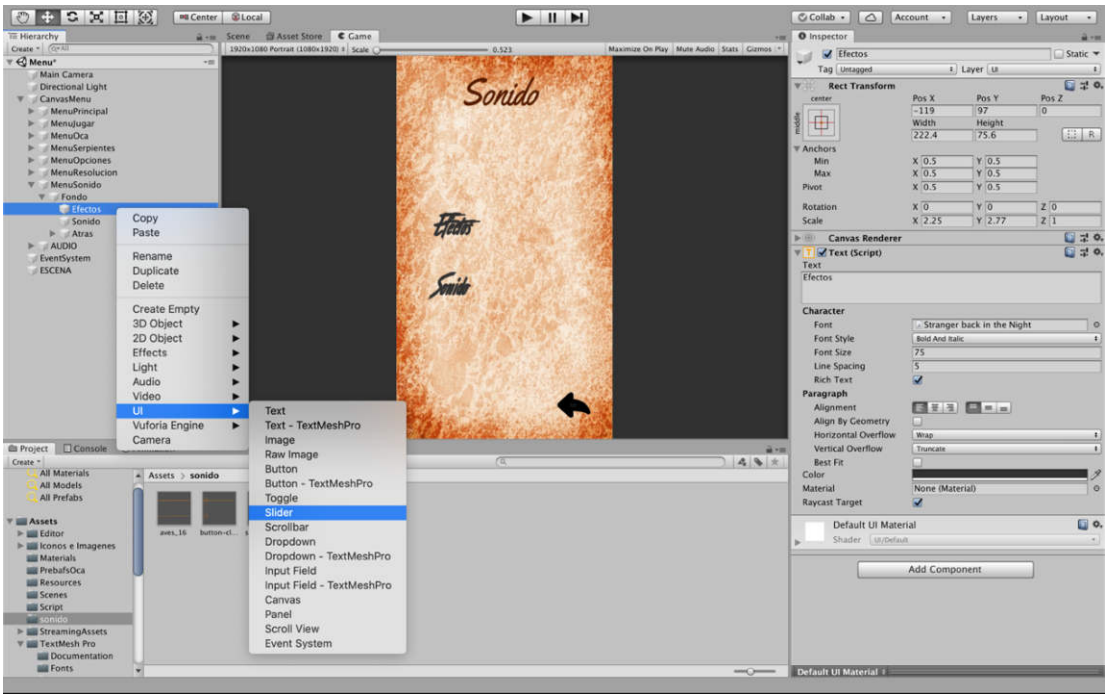


Figura 4.65: Añadir Slider a la escena

Se muestra a continuación, la parte estética con estos nuevos elementos comentado anteriormente.

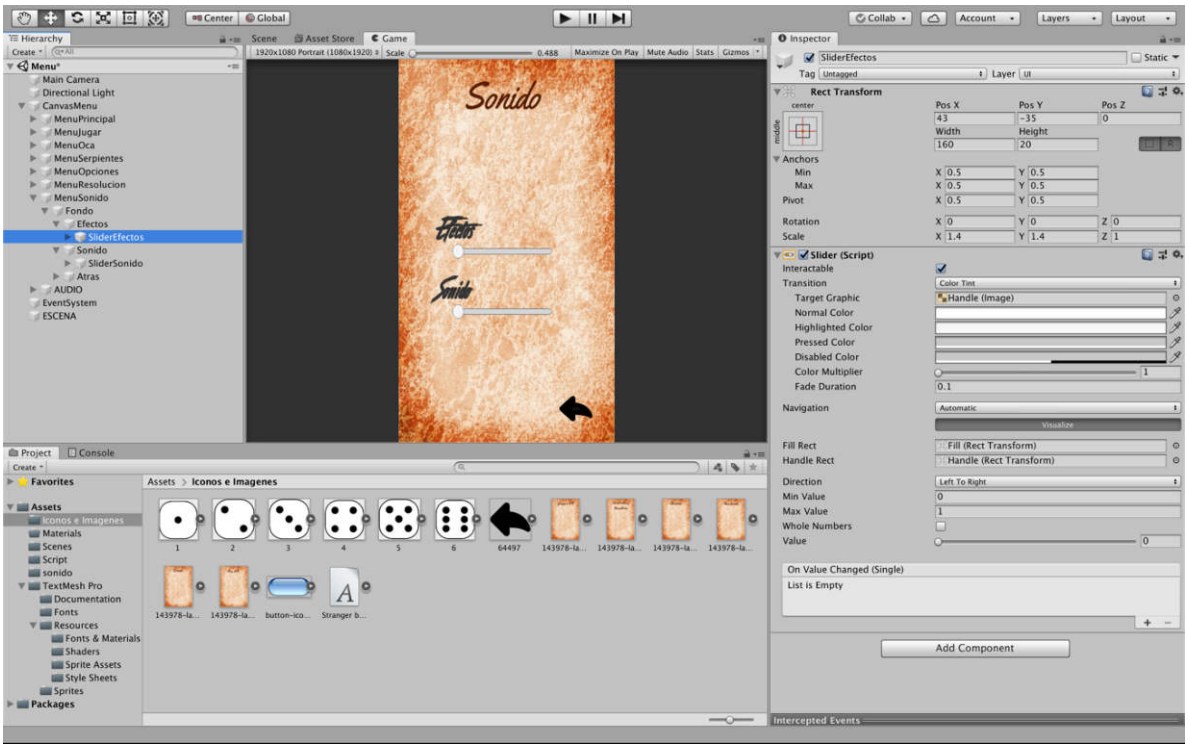


Figura 4.66: Vista preliminar del Menú Sonido

Como se ve en la imagen ambos Slider tiene valor de cero, se le cambiará ese valor a uno, para que cuando se inicie la aplicación el sonido que estará activado corresponda con el valor en realidad, para cambiarlo dentro de ambos sliders en la pestaña inspector buscar la opción con el nombre de “Value” y se pone ambos en el valor de uno, tal y como se muestra a continuación:

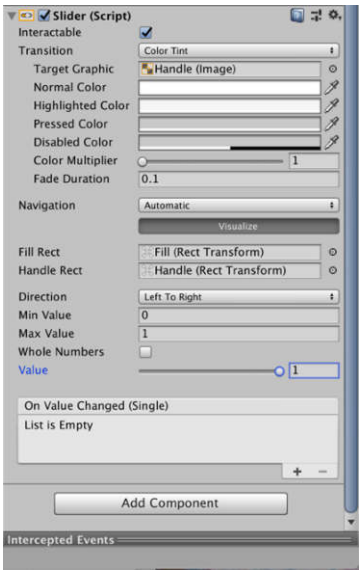


Figura 4.67: Componentes del objeto Slider

Una vez que se haya aplicado este cambio a los Sliders, se puede apreciar en la siguiente imagen como ambos tiene un valor de uno, es decir, que tiene el sonido y los efectos aplicados al 100%.

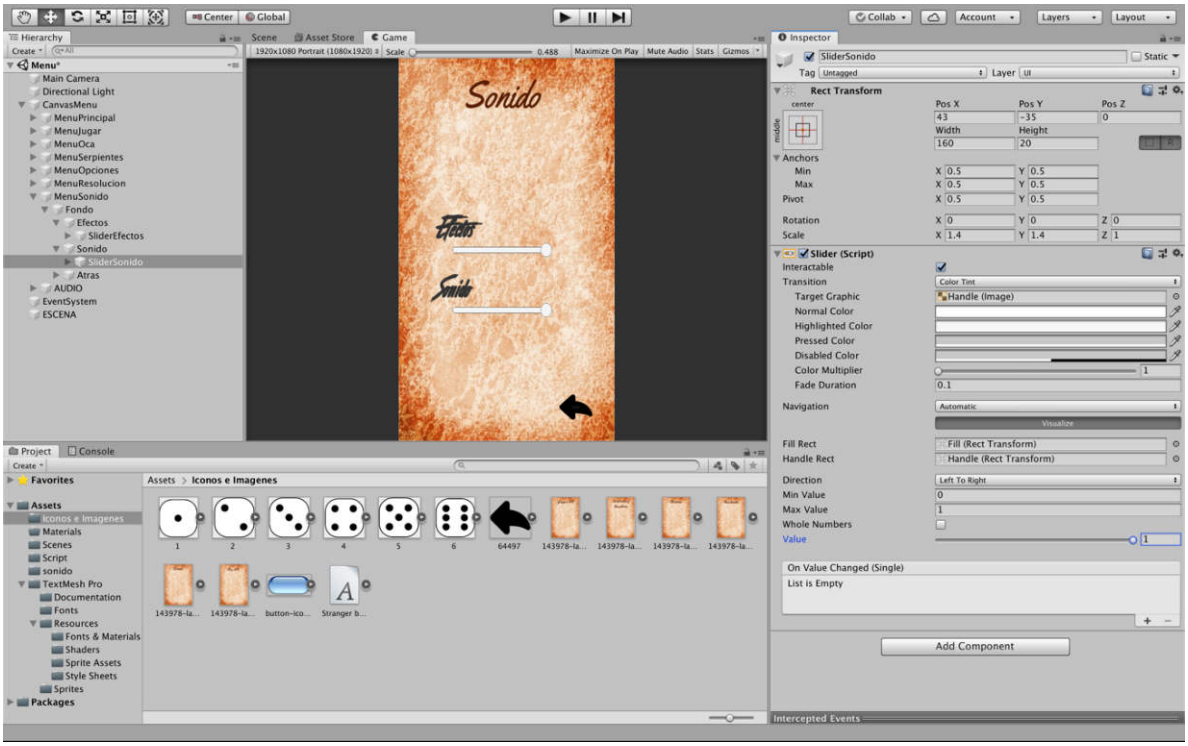


Figura 4.68: Vista final del Menú Sonido

4.6.7.1. Funciones de los Slider

Antes de realizar la configuración, se va añadir el audio que corresponda a la intro de este menú, es decir, que este sonido se va a escuchar en cuanto se inicie la aplicación será un sonido ambiental.

Para la incorporación de este nuevo audio, en el script "AudioClipButton" (que forma parte del objeto "AUDIO" situado en la ventana de jerarquía), se declarará las variables "AudioSource" (hace referencia a los AudioClip y se utiliza para la reproducción del sonido) y "AudioClip" (almacena el archivo de audio comprimido). A continuación se verá en la imagen estas dos variables declaradas:

```
public AudioSource audioNaturaleza;
public AudioClip ClipNaturaleza;
```

Figura 4.69: Declaración de las variables del sonido intro

Declaradas ambas variables, se procede a crear la función que reproduzca el sonido durante el tiempo que tiene ese audio, la función se llamará "ClipSonidoNaturaleza()", en el cual se va a utilizar una función que ya dispone el propio

Unity por defecto, esta función se llama “PlayOneShot”, por lo que se asociará el “AudioSource” a la función “PlayOneShot” y se le indicará que audio se quiere que se reproduzca.

```
public void ClipSonidoNaturaleza()
{
    audioNaturaleza.PlayOneShot(ClipNaturaleza);
}
```

Figura 4.70: Función que reproduce el sonido al iniciar la aplicación

Por otra parte, en Unity dentro del objeto “AUDIO”, se crea un objeto de audio de tipo “AudioSource”, se realizará tal y como aparece en la siguiente imagen:

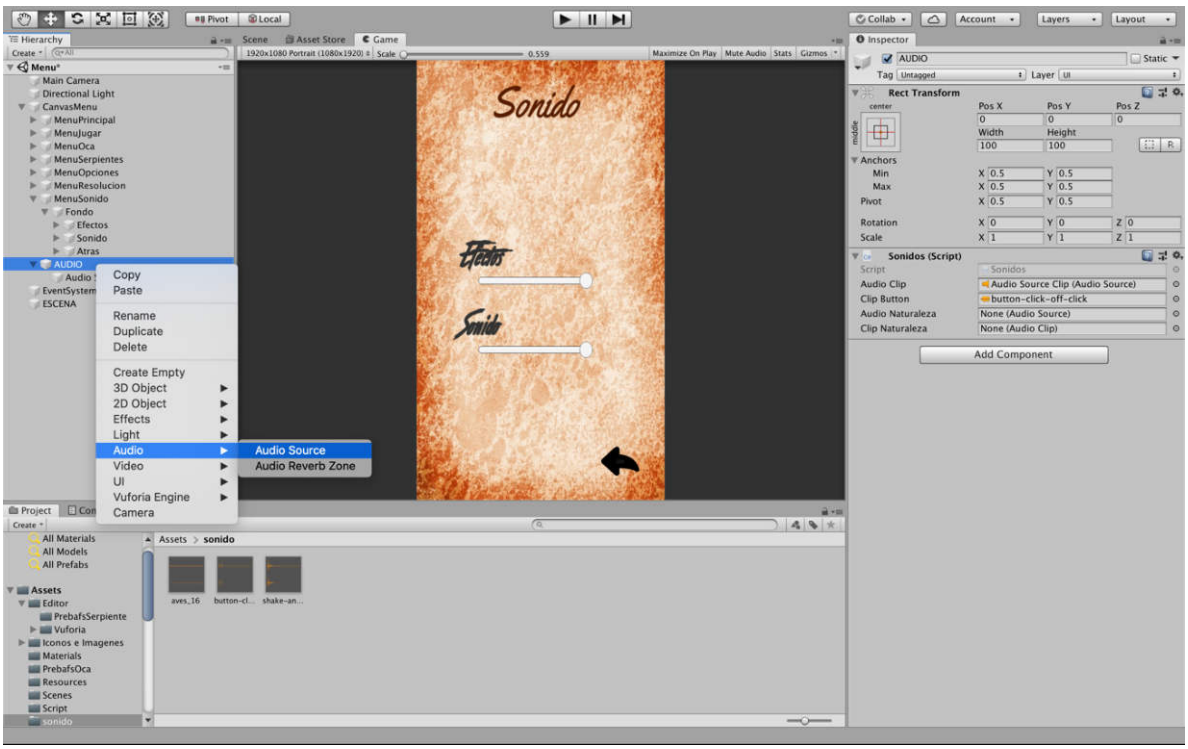


Figura 4.71: Añadir el componente AudioSource

Este objeto creado, se le cambiará el nombre a “Audio Source Sonido” para así poder diferenciar entre un audio y otro, en el cual se le importará el audio que se encuentra dentro de la carpeta “sonido” situado la ventana de proyectos a la sección “AudioClip” situado en la ventana de jerarquía de “Audio Source Sonido”. Además se deben de tener la opción “Play On Awake” activada para que simplemente cuando se pulse sobre el botón play se inicie el sonido, destacar que si esta opción no se activa cuando pulsemos sobre el play no se va a escuchar este sonido.

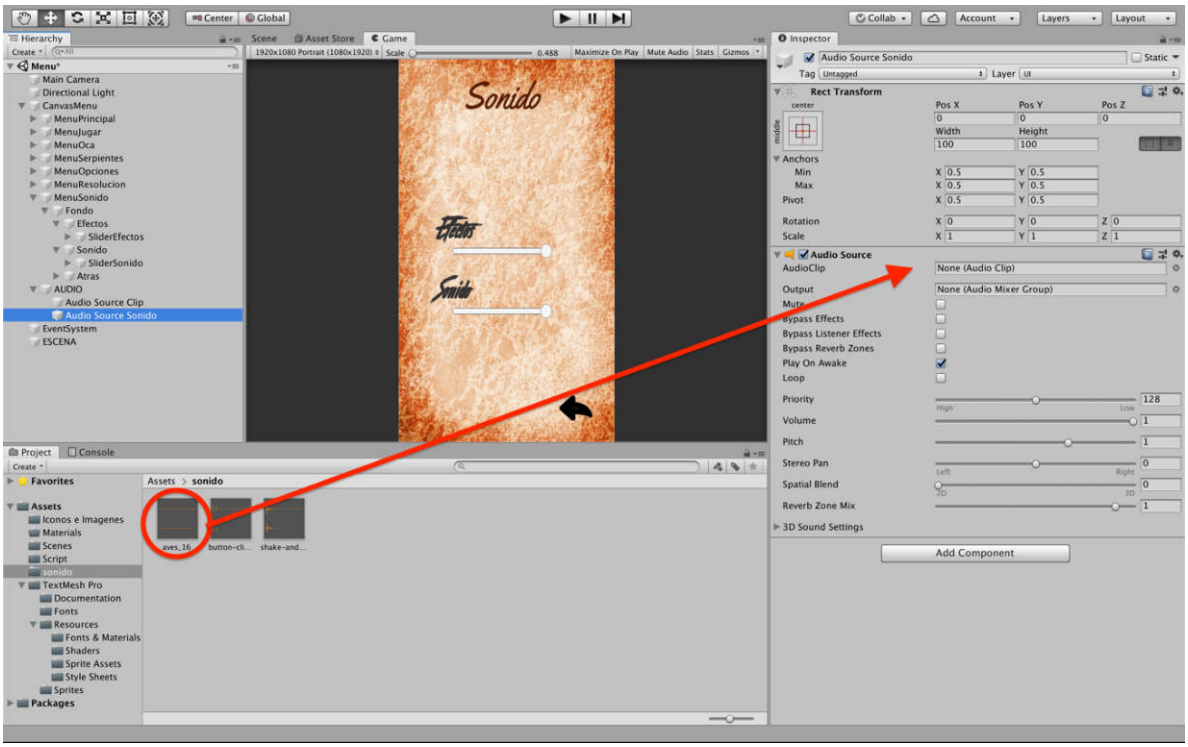


Figura 4.72: Añadir el sonido al componente AudioSource

Por tanto, solo queda asociar los objetos de audio a las variables públicas que se han declarado en el script anteriormente.

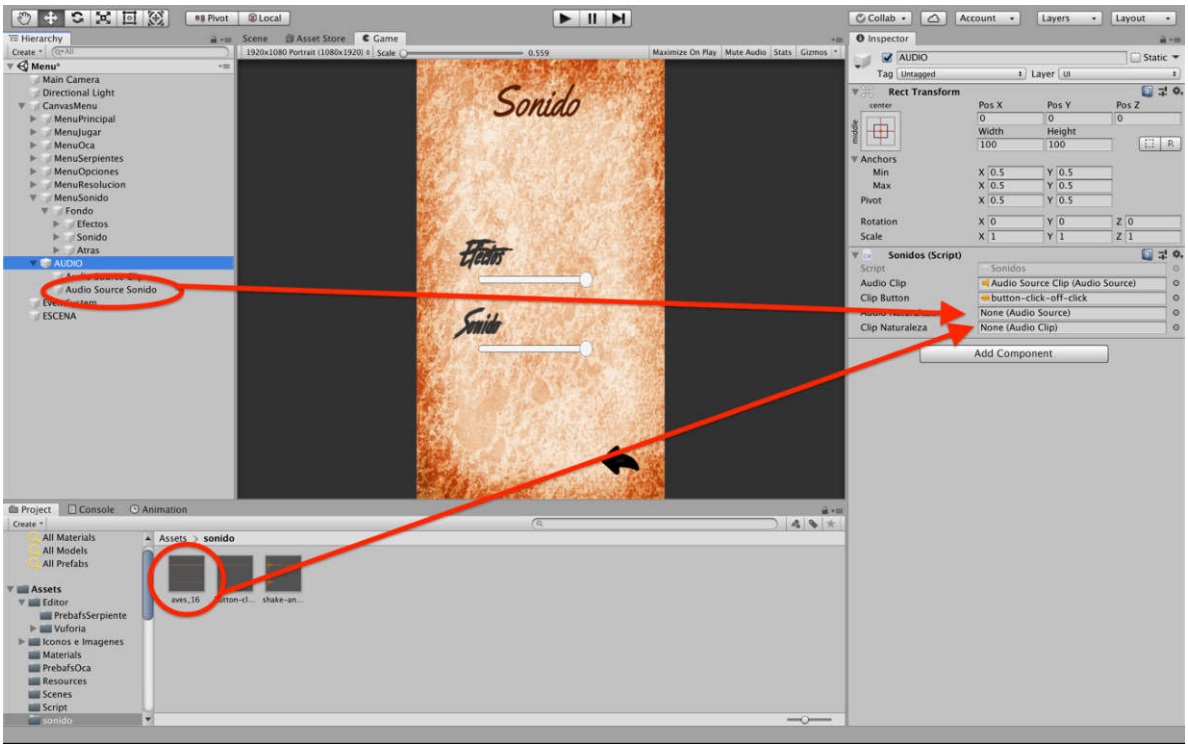


Figura 4.73: Asociar los componentes en las variables

Por consiguiente, se pasa a la configuración de estos sliders para que pueda el usuario bajar el volumen de ellas o directamente ponerlo en silencio. Para ello, en el mismo script donde se han creado ambos sonidos, se va a declarar dos variables de tipo “Sliders”, una de ellas llamada efectos y la otra sonido, para así poder asociarlo a los objetos Sliders que han sido creadas anteriormente en Unity.

```
public Slider efectos, volumen;
```

Figura 4.74: Declaración de las variables Slider

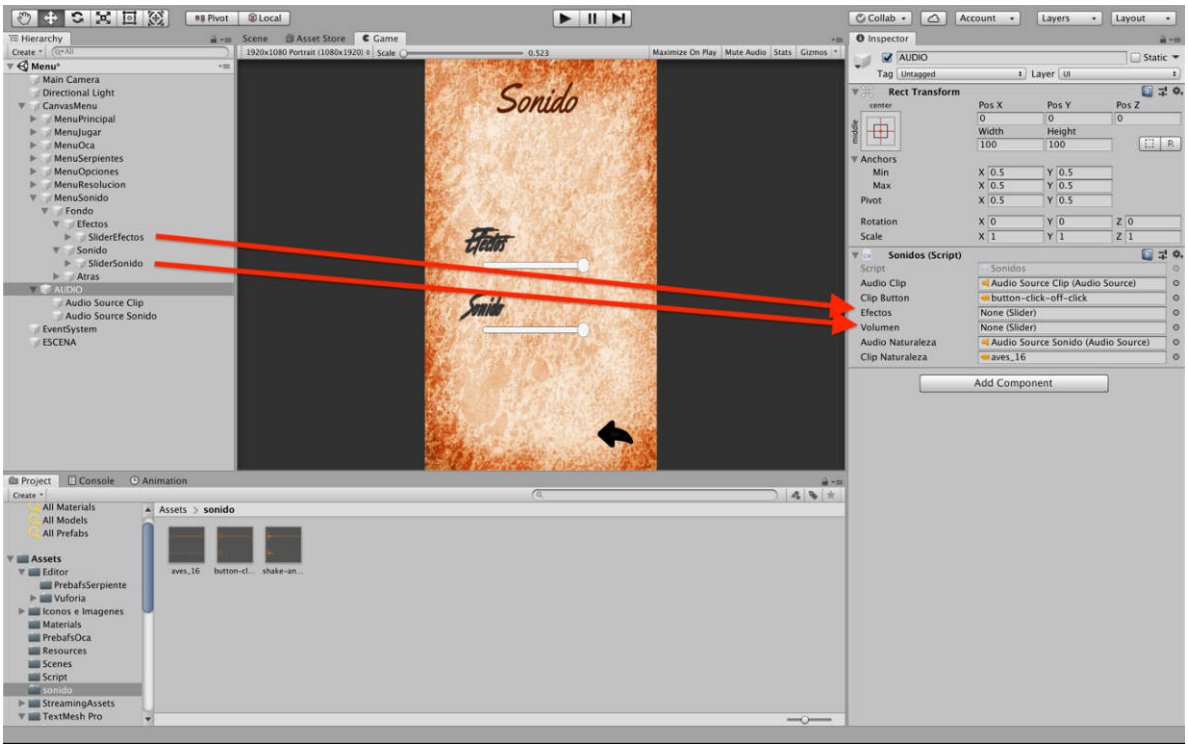


Figura 4.75: Asociar los componentes en las variables

De esta manera, las variables creadas por código van a estar referenciadas a estos dos objetos en Unity.

Una vez que se tiene declaradas ambas variables, se pasará a la creación de las dos funciones para que modifiquen el sonido del audio con respecto a la barra de los propios Sliders. Ambas funciones serán declaras en script “AudioClipButton()”.

```
public void onEfectos()
{
    AudioClip.volume = efectos.value;
}
```

```
public void onVolumen()
{
    audioNaturaleza.volume = volumen.value;
}
```

Figura 4.76: Funciones que modifica el sonido

La función “onEfectos()” hace referencia al slider “Efectos”, y la función “onVolumen()” hace referencia al slider “Sonido”.

Por último, para finalizar se referenciará cada una de estas dos funciones a su respectivo Slider, ya que nos da la opción de asignarle una acción al igual que se nos permite hacer en los botones.

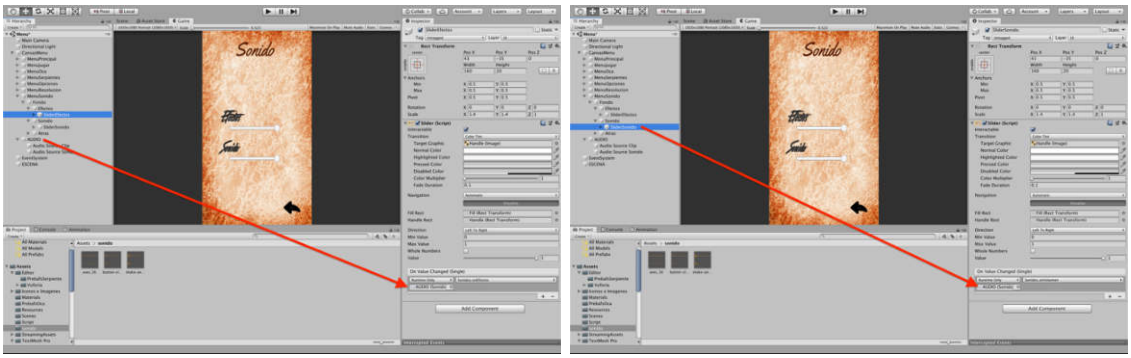


Figura 4.77: Añadir las funciones a cada componente Slider

4.6.7.2. Función del botón atrás

Con respecto a este apartado, se realiza de la misma manera que en el resto de menús, con la diferencia de que se quiere dejar el “MenuSonido” y se quiere activar el “MenuOpciones”, además de incorporar el sonido cuando se pulsa sobre este botón. Como el procedimiento es el mismo se muestra solo la imagen con las funciones de este botón.

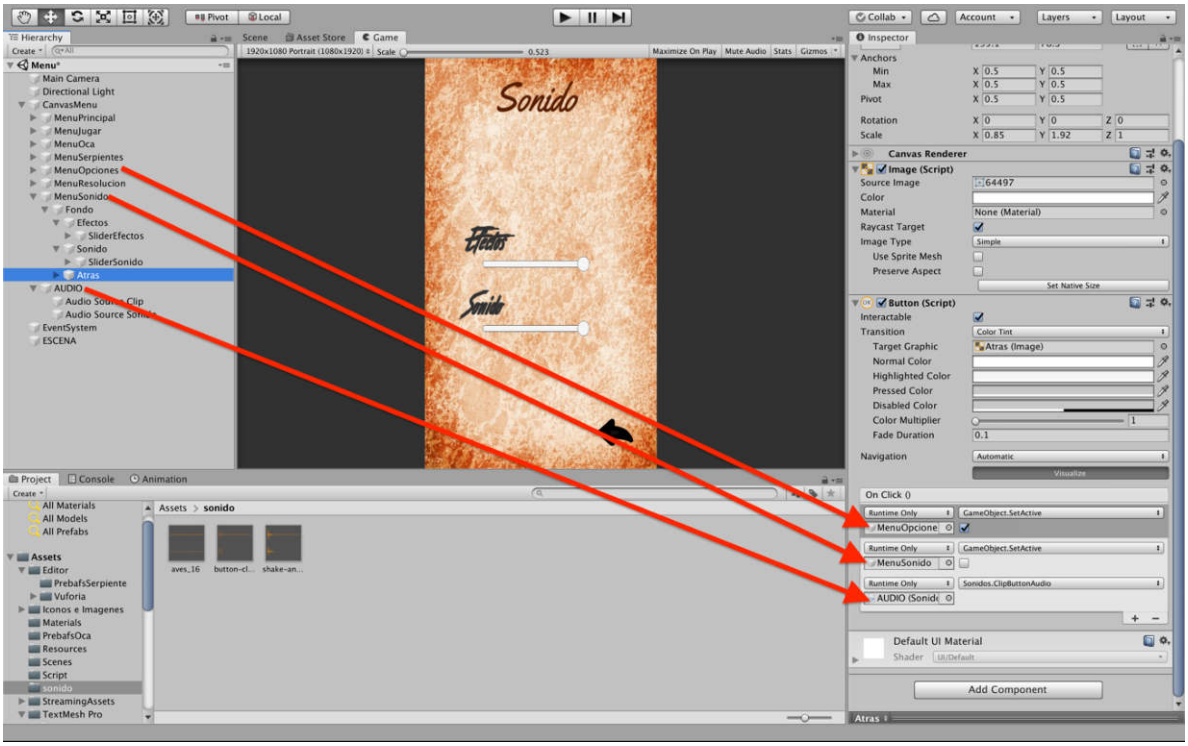


Figura 4.78: Cambiar de pantalla del botón Atrás

4.7. Desarrollo del Juego de la Oca

En esta sección todo lo que se comentara estará creado en una escena llamada “RestauranteMesaOca” y otra escena llamada “Oca”.

En la primera escena mencionada anteriormente, estará formada por dos “Canvas” uno es la selección del restaurante y otro la selección de mesas que cargará la escena “Oca”, ambos “Canvas” irán acompañados de una animación de texto. A continuación se muestra dos imágenes de lo comentado:

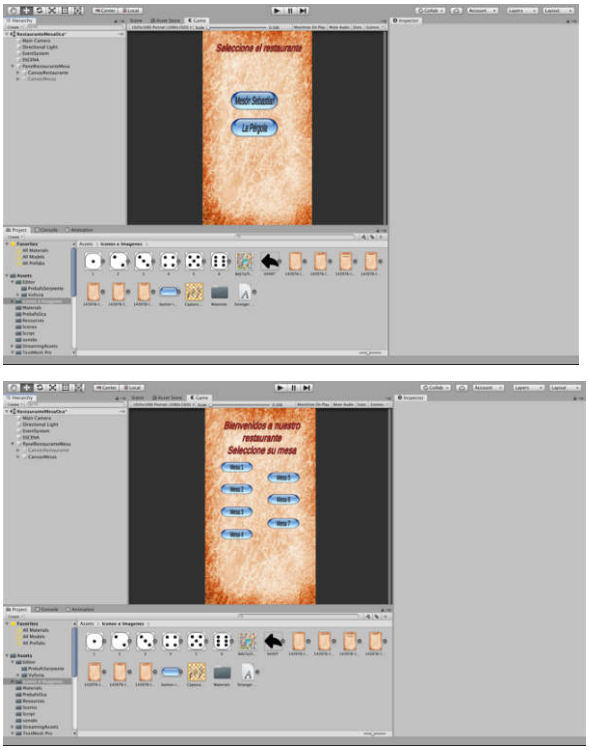


Figura 4.79: CanvasRestaurante y CanvasMesas

Para el desarrollo del juego de la oca, primero se empieza con la creación del tablero, seguidamente de las fichas y por último del movimiento y de efectos incluidos.

4.7.1. Creación del tablero

Para la creación del tablero, primero se va a crear un objeto de tipo “Plane” a una escala (1,1,1), a la cual llamaremos “TableroOcaJugador”. Seguidamente se le arrastrará la imagen del tablero descargada anteriormente, quedando de la siguiente manera:

Figura 4.80: Tablero de la Oca

A continuación, se crea la ruta que va a seguir nuestras fichas, en este caso se va a crear la ruta mediante “GameObject” vacíos, donde se creará cada objeto vacío en cada casilla y se enumerará en orden (servirá de utilidad para así poder localizarlos cuando se tenga cualquier problema en la consola de Unity, y además que seguirá cambiando la posición de la ficha de uno en uno). Todos los objetos estará dentro de un “Create Empty” (GameObject vacío) dentro de la ventana jerarquía, y se le pondrá de nombre “CircuitoOca”.

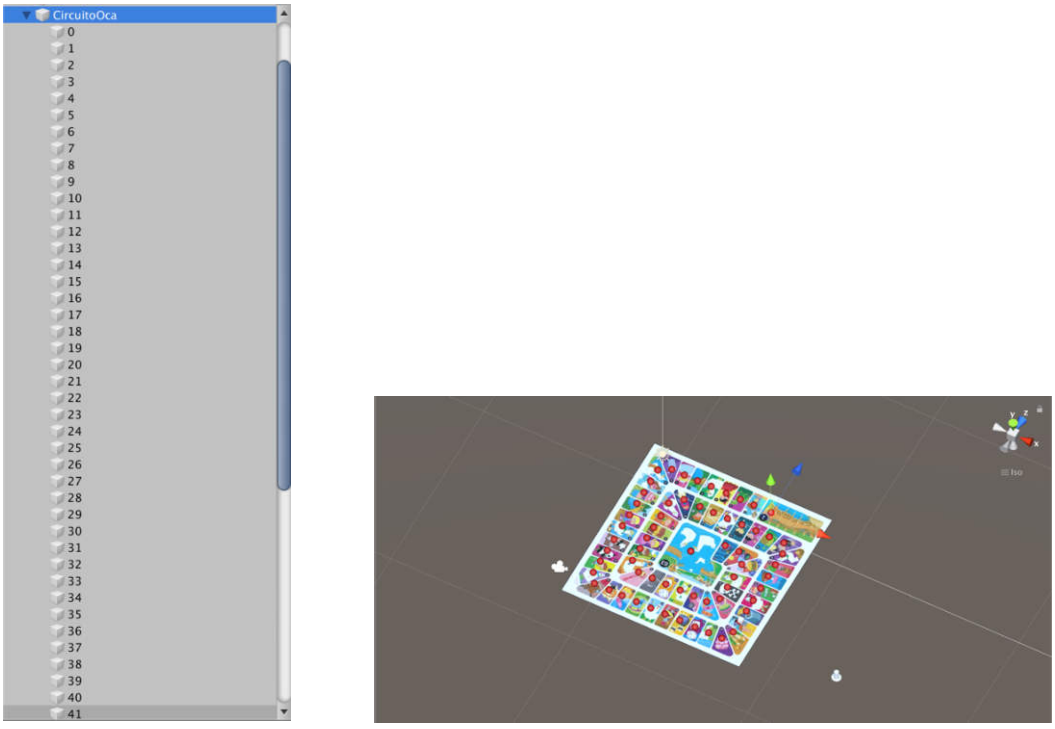


Figura 4.81: Ruta del tablero del juego Oca

Una vez, colocados todos los objetos como en la imagen anterior, lo siguiente que se va a realizar es la creación de una lista que conlleve estos “GameObject”, para ello se hará con este pequeño código, en el cual se puede añadir todos los objetos que se quiera a esta lista.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ArrayOca : MonoBehaviour
{
    Transform[] posicion;
    public List<Transform> listaPosicion = new List<Transform>();

    void Update()
    {

        listaPosicion.Clear();
        posicion = GetComponentsInChildren<Transform>();
        foreach (Transform child in posicion)
        {


            if (child != this.transform)
            {
                listaPosicion.Add(child);
            }
        }


    }
}
```

Figura 4.82: Script “ArrayOca.cs”

Cuando se tenga el script listo, se le asignará al “CircuitoOca”, para asignárselo simplemente se arrastra el script llamado “ArrayOca”, de la forma en la que aparece en la siguiente imagen:

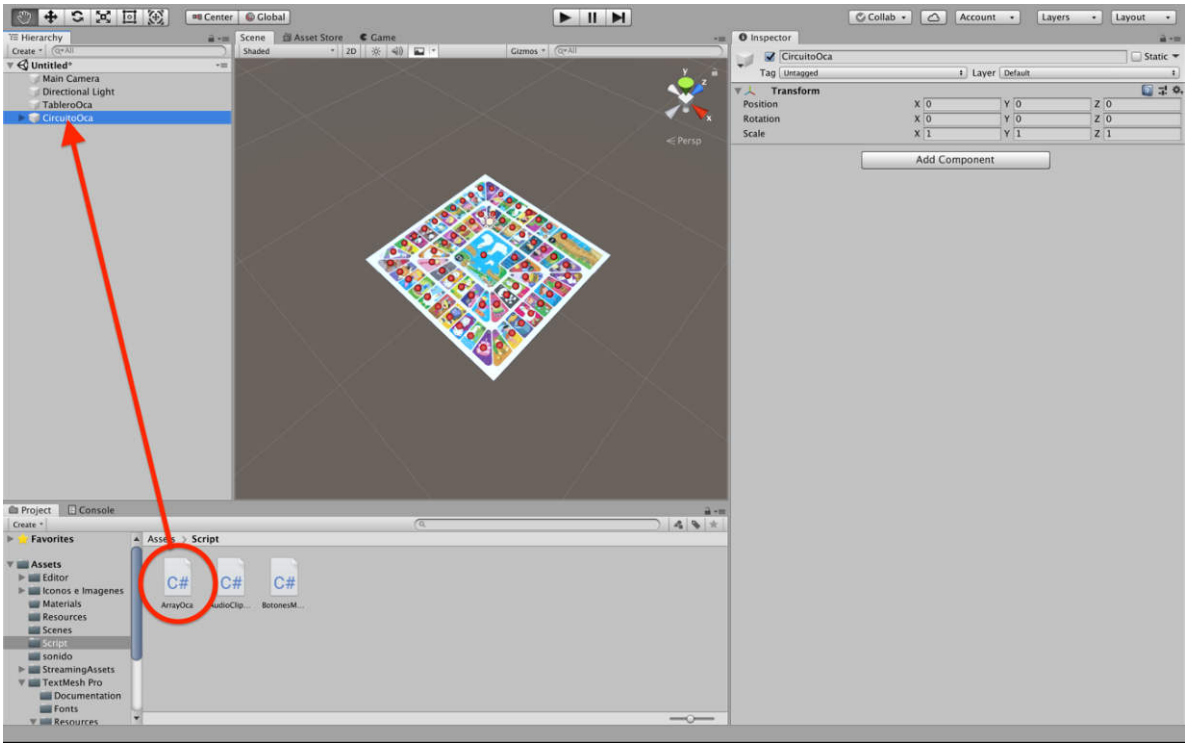


Figura 4.83: Añadir el Script “ArrayOca.cs” al objeto CircuitoOca

Como ya se tiene el script asignado, ahora lo que se hace es arrastrar todos los “GameObject” dentro del circuito, de esta manera ya se tiene asignados los “GameObject” a la lista creada por código llamada “listaPosicion”, como aparece en este imagen que vemos a continuación.

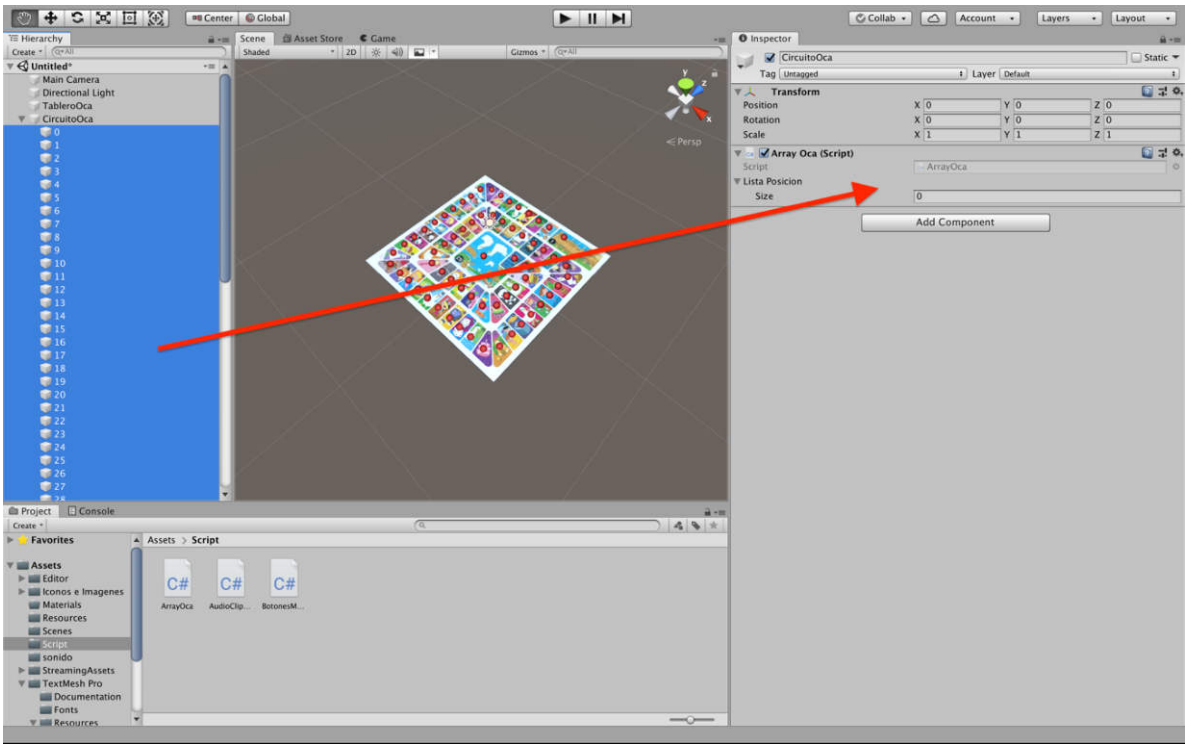


Figura 4.84: Asociar los objetos de cada posición al ArrayOca.cs

De esta forma, ya tendríamos nuestra ruta por la que la ficha va a pasar. Por lo tanto, en el script llamado “Oca.cs”, vamos a llamar al script “ArrayOca.cs” de tipo publico para asociarle el script, y se le llamará de nombre “ruta".

```
public ArrayOca ruta;
```

Figura 4.85: Declaración de la variable ruta

Por consiguiente se va a comentar como añadir el objeto “CircuitoOca”, situado en la ventana de jerarquía, a la cual ya tiene asignado el script “ArrayOca.cs”, y los objetos que forma el circuito, al script “Oca.cs”, donde todas las variables de tipo públicas aparecerán, y por lo cual es donde se va asociar el objeto “CircuitoOca”. Para poder asociar ambos script se necesita tener un objeto de referencia, y a la cual va destinado todo el proceso de movimiento y lógica de nuestro juego, en el cual se explicara con más detalle en el siguiente punto.

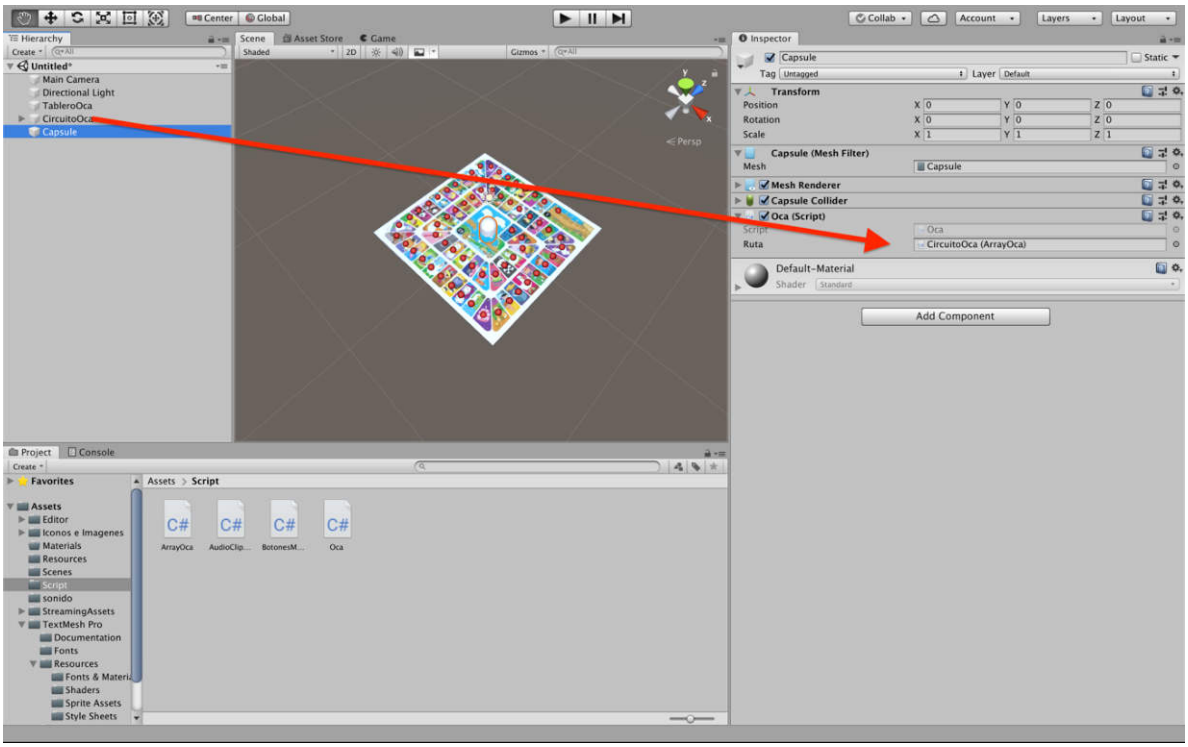


Figura 4.86: Añadir la ruta al objeto

De esta forma en el script “Oca”, cuando se quiera acceder alguna en posición en concreto se hará de la siguiente manera:

ruta.listaPosicion[3].position;

4.7.2. Creación de las fichas

En esta sección, se va a explicar como se crea las fichas del tablero, en este caso se va a realizar a través de la ventana de jerarquía, accediendo a la sección “GameObject” y seleccionado el tipo de objeto que se necesite, ya que Unity tiene varios objetos predefinidos como el cubo, cilindro, capsula, esfera, entre otros, pero en nuestro caso, se usará una capsula, a la que se llamará “FichaOcaJugador”. También comentar que estos objetos predefinidos de Unity se pueden crear mediante código con la función que aparece a continuación:

```
GameObject.CreatePrimitive(PrimitiveType.Capsule);
```

Figura 4.87: Función de crear objetos predefinidos

Además también se va a cambiar la posición de la ficha para que aparezca en la casilla inicial, así como la escala, para ello en la ventana de inspector en la sección “Transform” se cambiará estos valores.

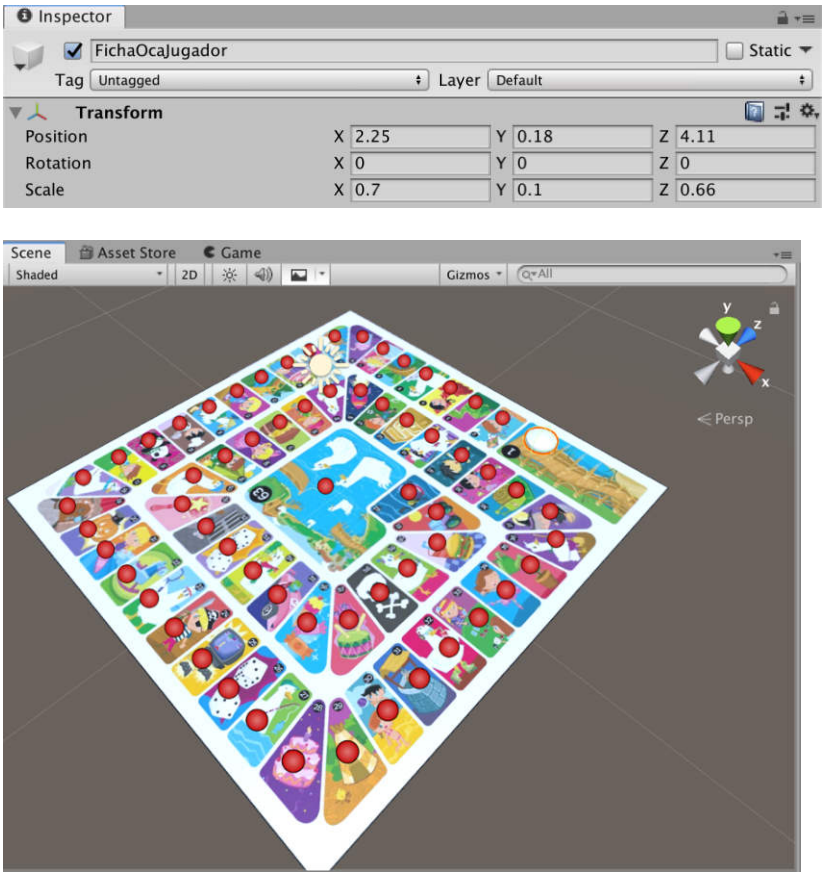


Figura 4.88: Posición inicial de la ficha

También se puede realizar estas opciones en el código mediante comandos, para cambiar cualquier posición, se puede realizar mediante el siguiente comando, donde "objeto", hará referencia al objeto que se quiera cambiar de posición:

```
objeto.transform.position = new Vector3(1, 1, 1);
```

Figura 4.89: Cambio de la posición mediante código

Además, para cambiar la escala de algún objeto, se puede hacer a través del siguiente comando, donde "objeto", hará referencia al objeto que se quiera modificar su tamaño:

```
objeto.transform.localScale = new Vector3(1, 1, 1);
```

Figura 4.90: Cambio del tamaño mediante código

A continuación, en nuestro script "Oca.cs", se creará una variable de tipo publica llamada "ficha", a la cual se le asociará nuestra ficha creada desde la ventana de jerarquía.

```
public GameObject ficha;
```

Figura 4.91: Declaración de la variable objeto

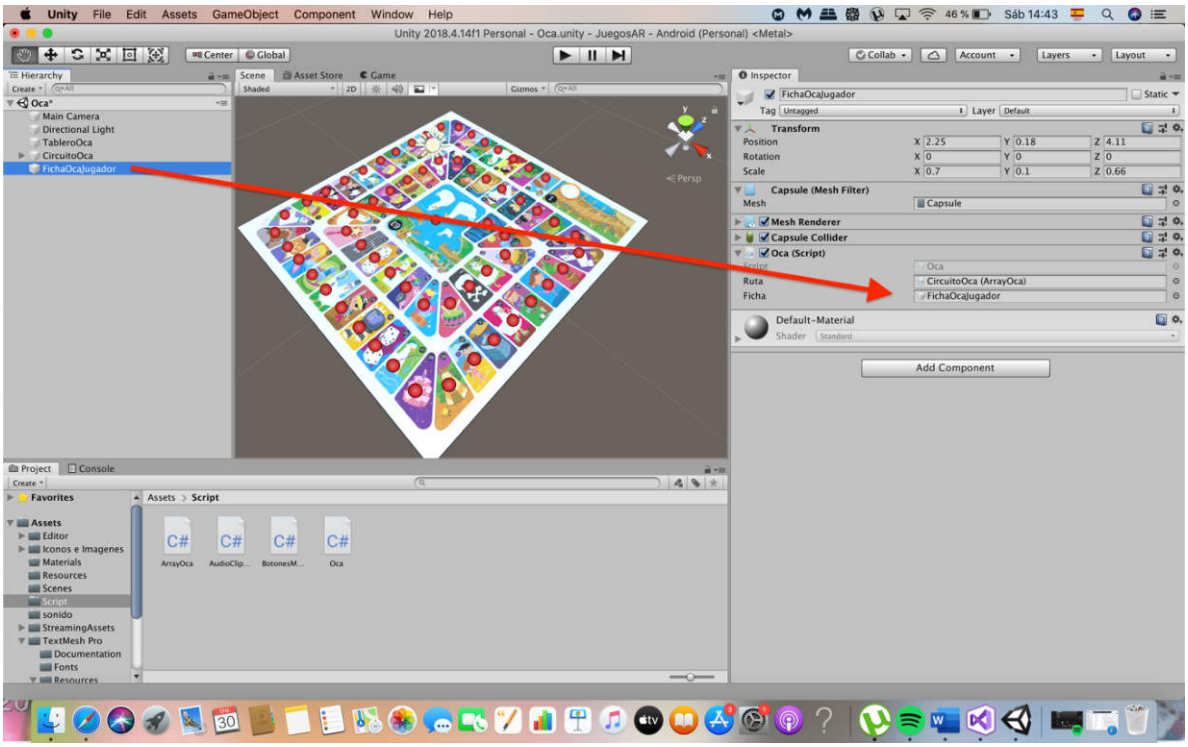


Figura 4.92: Asociar el objeto en la variable declarada

Además, también se creará una lista de tipo “GameObject”, a la que llamaremos “listaPlayer” en el cual va a contener todas nuestras fichas en el caso de que se tenga muchas fichas por manejar y sea más complicado de acceder a una en concreto, pero en este caso aunque solo se tenga una única ficha también se va a realizar de esta manera.

```
List<GameObject> listaPlayer = new List<GameObject>();
```

Figura 4.93: Crear una lista de objetos

Para añadir la ficha a esta lista, se accederá al método “Add” y, se le indicará el nombre de ese objeto, tal y como se puede mostrar a continuación:

```
listaPlayer.Add(ficha);
```

Figura 4.94: Añadir un objeto a la lista creada

Por consiguiente, se va a explicar como se ha creado una lista de colores, además de añadir a esa lista más colores, por otro lado, en función de los valores RBG, se le asociara por texto el nombre del color, así como crear los objetos con colores aleatorios.

A continuación, se muestra la creación de la lista, esta lista tiene que ser tipo “Color”, a la cual se le asociará los colores que viene por defecto, además de añadir otros colores como se ha mencionado antes, para añadir los nuevos colores se realizará a través de los

valores de RGB, para ello se le dará un valor entre 0-1 a cada componente RGB, en el cual se formará un color en concreto..

```
public List<Color> listaColores = new List<Color> { Color.red, Color.green, Color.grey, Color.blue, Color.cyan, Color.magenta, Color.yellow, Color.black,

new Color(1f, 0.44f, 0.16f, 1f), new Color(0.3f, 0.4f, 0.6f, 0.3f), new Color(0.55f, 0f, 1f, 1f), new Color(0.63f, 0.33f, 0.11f, 1f), new Color(0.60f, 0.07f, 0.07f, 1f) };
```

Figura 4.95: Declaración de lista colores

Para poder asignarle un nombre a todos los colores, se necesita una variable de tipo “Color” a la que va a referencia los valores de RGB y una variable de tipo “String” que referenciará al nombre del color.

```
public Color randomColor;
public String TextColor;
```

Figura 4.96: Declaración de la variable Color y de la variable String

A continuación, se muestra dicha función llamada “NameColores()”:

```
public void NameColores()
{
    if (randomColor.r == 1 && randomColor.g == 0 && randomColor.b == 0)
    {
        TextColor = "Rojo";
    }

    if (randomColor.r == 0 && randomColor.g == 1 && randomColor.b == 0)
    {
        TextColor = "Verde";
    }

    if (randomColor.r == 0 && randomColor.g == 0 && randomColor.b == 1)
    {
        TextColor = "Azul";
    }

    if (randomColor.r == 0.5 && randomColor.g == 0.5 && randomColor.b == 0.5)
    {
        TextColor = "Gris";
    }

    if (randomColor.r == 0 && randomColor.g == 1 && randomColor.b == 1)
    {
        TextColor = "Cyan";
    }

    if (randomColor.r == 1 && randomColor.g == 0 && randomColor.b == 1)
    {
        TextColor = "Rosa";
    }

    if (randomColor == Color.yellow)
    {
        TextColor = "Amarillo";
    }

    if (randomColor.r == 0 && randomColor.g == 0 && randomColor.b == 0 && randomColor.a == 1f)
    {
        TextColor = "Negro";
    }

    if (randomColor.r == 0.3f && randomColor.g == 0.4f && randomColor.b == 0.6f && randomColor.a == 0.3f)
    {
        TextColor = "Azul Agua";
    }
```

```
    if (randomColor.r == 1f && randomColor.g == 0.44f && randomColor.b == 0.16f && randomColor.a == 1f)
    {
        TextColor = "Naranja";
    }

    if (randomColor.r == 0.55f && randomColor.g == 0f && randomColor.b == 1f && randomColor.a == 1f)
    {
        TextColor = "Morado";
    }

    if (randomColor.r == 0.63f && randomColor.g == 0.33f && randomColor.b == 0.11f && randomColor.a == 1f)
    {
        TextColor = "Marrón";
    }

    if (randomColor.r == 0.6f && randomColor.g == 0.07f && randomColor.b == 0.07f && randomColor.a == 1f)
    {
        TextColor = "Granate";
    }
}
```

Figura 4.97: Función NameColores()

Ya solo quedaría, crear una función de tipo “Color” y que nos devuelve un color aleatorio de la “listaColores”.

```
public Color crearOb()

{
    int indice = UnityEngine.Random.Range(0, listaColores.Count);

    randomColor = listaColores[indice];

    ficha.GetComponent<Renderer>().material.color = randomColor;//asignamos un color al objeto de la lista seleccionado

    return randomColor;//obtenemos de la lista un color aleatorio
}
```

Figura 4.98: Función crearOb()

Por último, esta última función, y la función “NameColores()”, la declaremos en la función “Start()”, tal y como se puede ver en la siguiente imagen.

```
void Start()
{

    listaCambioPosiciones.Add(0);

    listaNuevaPosicion.Add(0);

    listaPerderTurno.Add(0);

    listaPlayer.Add(ficha);

    crearOb();

    NameColores();

    CurrentPlayer = 0;

    textoOcaPuente.gameObject.SetActive(false);

    textoJugador.gameObject.SetActive(false);

    textoCasillaEspecial.gameObject.SetActive(false);

    textoGanador.gameObject.SetActive(false);
}
```

Figura 4.99: Función Start()

4.7.3. Creación del movimiento

En esta sección se va a dividir en tres partes. Una de ellas es la función “TirarDados()”, que cuando pulsas se activa la función “MoverFicha()”, y la última es la función “movimientoOca()”, esta función se va activar cuando un jugador caiga en una oca, puente o cuando tenga que retroceder en la última casilla, porque el número del dado es mayor que las posiciones que le quedan.

4.7.3.1. Función TirarDados()

En esta función, va a realizar la suma de los dos dados, así como actualizar por texto a que jugador le toca tirar al que le pasaremos la función “TextoJugador()”, y se le pasará la función “MoverFicha()” y, “ClipButtonDados()”.

Lo primero de todo será crear los dos dados (el que contendrá las imágenes de los dados) y el botón de Tirar (que accionará el movimiento). Para ello dentro del objeto “FichaOcaJugador” situado en la ventana de jerarquía, se creará un objeto de tipo “Canvas” al cual le llamaremos “CanvasPanel” conteniendo un objeto de interfaz de usuario de tipo “RawImagen” al que llamaremos “FondoPanel”, y a su misma vez, estará compuesto por estos botones mencionados anteriormente.

El objeto “FichaOcaJugador”, será hijo del objeto “TableroOcaJugador”, así como otro objeto de tipo “Canvas” al que se llamará “CanvasTextoUnJugador”, que estará compuesto por un objeto de interfaz de usuario “RawImagen” llamado “FondoTexto”, y dentro de este estará todos los objetos de texto que contempla este juego.

Con respecto a los objetos de tipo texto hay dos modelos, se puede utilizar el que más guste, uno de ellos es “Text” y el otro objeto es “TextMeshPro”, la principal diferencia entre estos dos modelos es la alta calidad del texto, por esta razón se ha utilizado “TextMeshPro”.

A continuación, se mostrará dicha jerarquía comentada anteriormente, aunque en la jerarquía no aparece el objeto de audio se añadirá más adelante dentro del objeto “TableroOcaJugador”. Los objetos que aparecen en color azul es porque son objetos prefabricados, es decir, que se han añadido a la sección “Project Windows” en una carpeta llamada “PrebabsOca”.

Figura 4.100: Jerarquía del objeto "TableroOcaJugador"

El script "Oca.cs" a la cual esta asociada al objeto "FichaOcaJugador" se le asociara todos estos elementos que se han creado en la ventana de jerarquía, de manera que estén referenciados todas las variables creadas en el script de tipo "Public", a las que algunas se necesitará añadir imágenes como por ejemplo las de los dados. Para realizar esto, simplemente es arrastrar los objetos creados en la ventana de jerarquía a la ventana inspector, de manera que se tenga asociado los objetos con las variables públicas. Se puede apreciar en la siguiente imagen, la parte derecha aparece las variables públicas que se han creado en el script "Oca.cs".

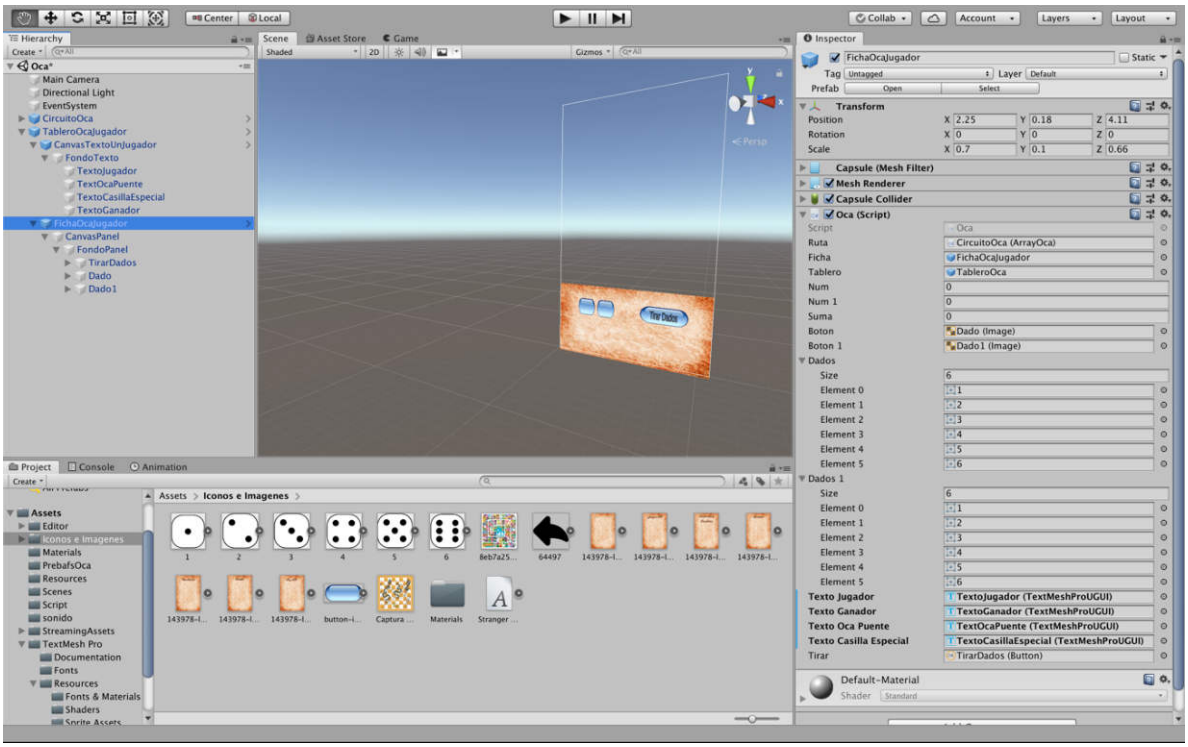


Figura 4.101: Vista de las variables del objeto "FichaOcaJugador"

Como he comentado antes, los botones de los dados serán de tipo imagen para ello se hará mediante código de la siguiente manera como aparece en la imagen.

```
public Image Boton;
public Image Boton1;
```

Figura 4.102: Declaración de las variables Image

Por otro lado, el botón para que tire los dados, se declara en el script de tipo "Button", tal como aparece en la imagen.

```
public Button Tirar;
```

Figura 4.103: Declaración de la variable Button

De igual manera, se realizara la declaración de variables de tipo "TextMeshProUGU", pero para ello se debe de incluir la librería "using TMPro".

```
public TextMeshProUGUI textoJugador;
public TextMeshProUGUI textoGanador;
public TextMeshProUGUI textoOcaPuente;
public TextMeshProUGUI textoCasillaEspecial;
```

Figura 4.104: Declaración de las variables TextMeshProUGUI

De esta manera, ya aparece en la ventana inspector y a la cual se le va asociar los dos botones que se crearon para los dados, el botón de tirar los dados, así como los diferentes textos. Simplemente para asociarlo con arrastrarlo es suficiente como ya se ha comentado.

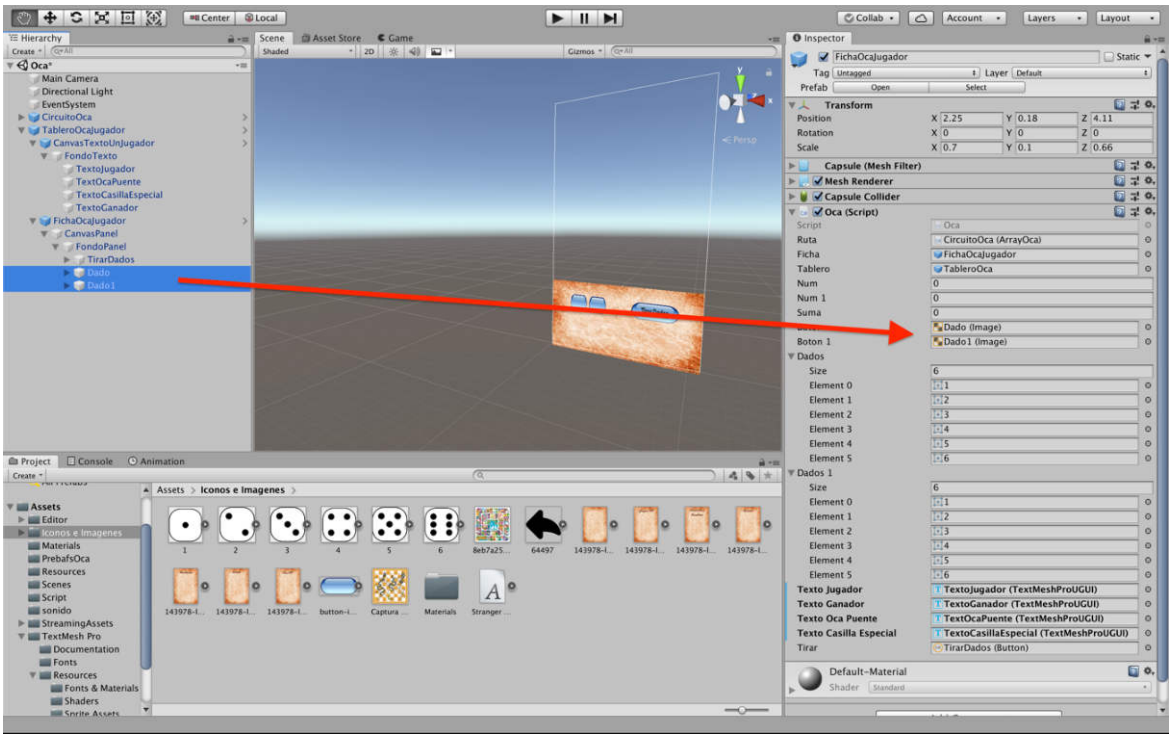


Figura 4.105: Asociar los dados en las variables declaradas

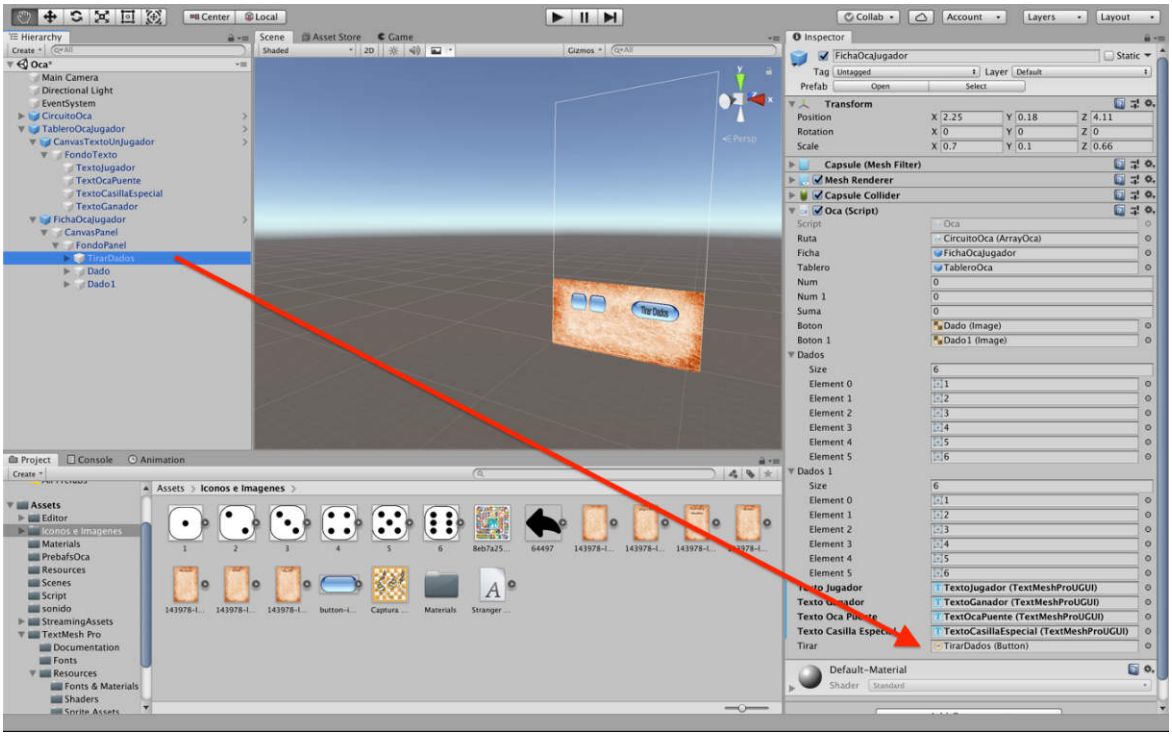


Figura 4.106: Asociar el botón Tirar en la variable declarada

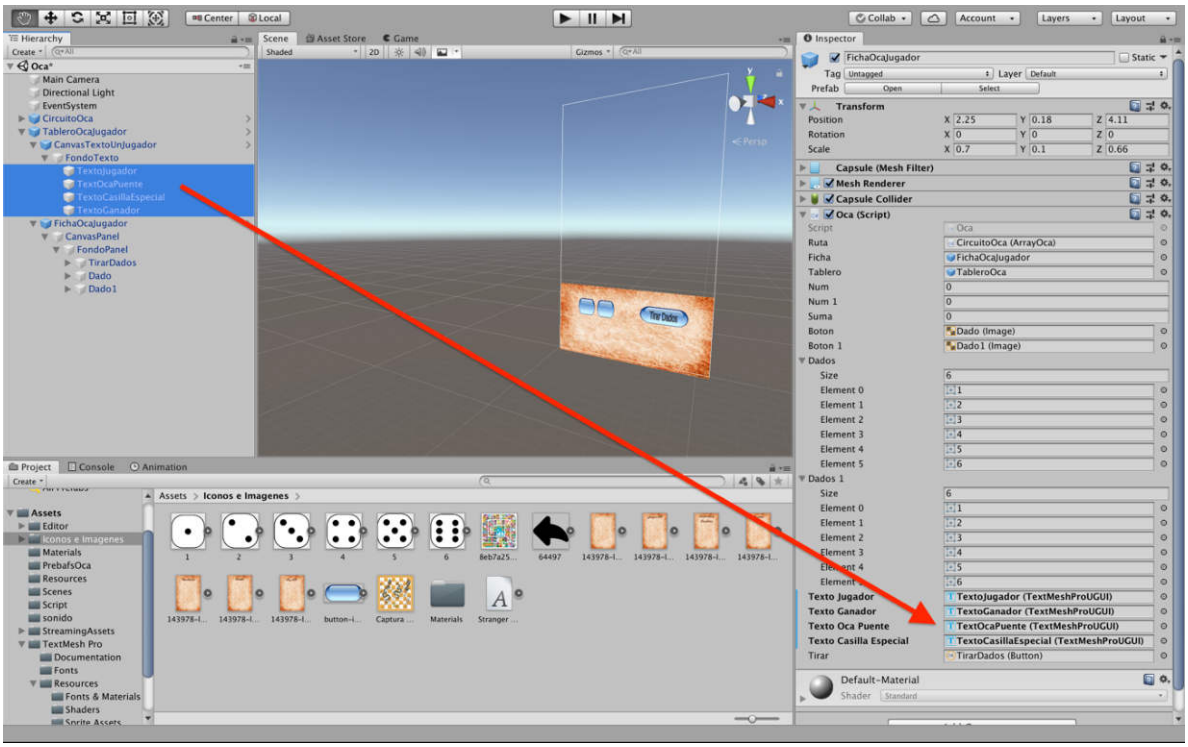


Figura 4.107: Asociar los objetos tipo texto en las variables declaradas

De esta manera, al tener referenciado el botón creado por código, al botón creado en la ventana de jerarquía, se puede hacer por ejemplo, que cuando la ficha este en movimiento este botón lo bloquee y que cuando no tenga movimiento que el botón este activada para poder volver a tirar.

Una vez que ya se tenga asociado el botón a la imagen, lo que se va hacer mediante código, es un array para introducir las imágenes de los dados.

```
public Sprite[] dados;

public Sprite[] dados1;
```

Figura 4.108: Declaración de las variables Sprite

Se asocia el array con las imágenes de las caras de los dados, arrastrando esta última a la ventana de inspector.

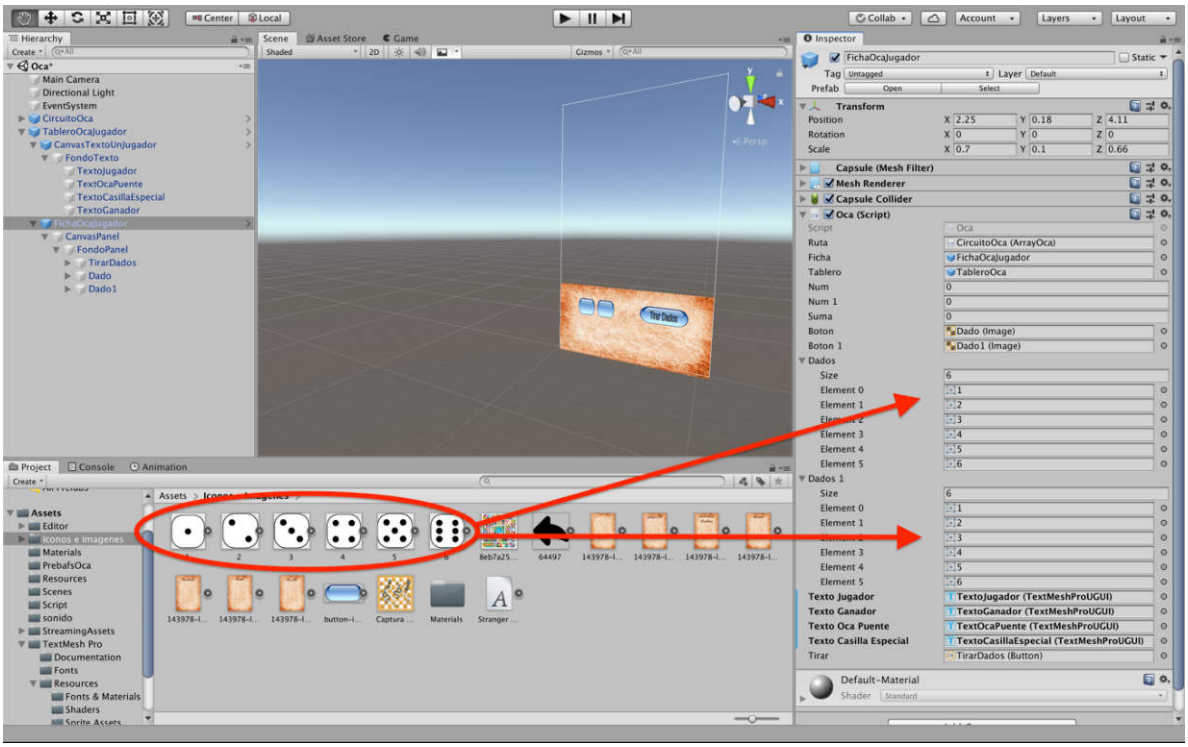


Figura 4.109: Asociar las imágenes de los dados en las variables declaradas

Por último, falta enlazar ambas cosas, es decir, asociar los botones con las caras de los dados y que de un valor aleatorio. Primero se realiza que la variable de tipo entero “num” y “num1” de un valor aleatorio, seguidamente se le asociara la imagen del botón con el array de las caras de los dados.

```
else
{
    Boton.enabled = true;//Para activar el boton que se desactivo cuando estaba en la posicion 60
    num = UnityEngine.Random.Range(0, dados.Length);//numero aleatorio del primer dado

    Boton.sprite = dados[num];//para asociar el boton que es una imagen a las imagenes del dado


    num1 = UnityEngine.Random.Range(0, dados1.Length);//numero aleatorio del segundo dado


    Boton1.sprite = dados1[num1];//para asociar el boton1 que es una imagen a las imagenes del dado

    Boton1.rectTransform.anchoredPosition = new Vector3(-54.73f, 24.09f, 0f);// le damos la posicion que tenia con anterioridad al
    //desactivar el boton

    suma = num + num1; //suma de ambos dados
    suma = suma + 2;

    Debug.Log("la suma es " + suma);

}
```

Figura 4.110: Parte de la función TirarDados()

Se tiene que destacar en el código que en la penúltima línea se le suma un +2 porque en el array de las caras de los dados empieza por el elemento 0, entonces para el elemento 0 esta asociado el valor 1 del dado y así sucesivamente, como se puede ver en la imagen que se ve a continuación.

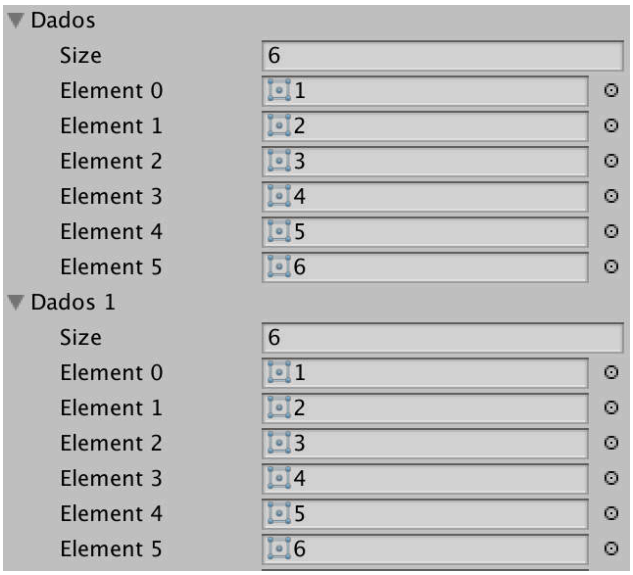


Figura 4.111: Índices de los array

Para que esto funcione, se le debe de asociar al botón Tirar esta función, para que simplemente cuando se pulse se active esta función, tal como aparece en la siguiente imagen.

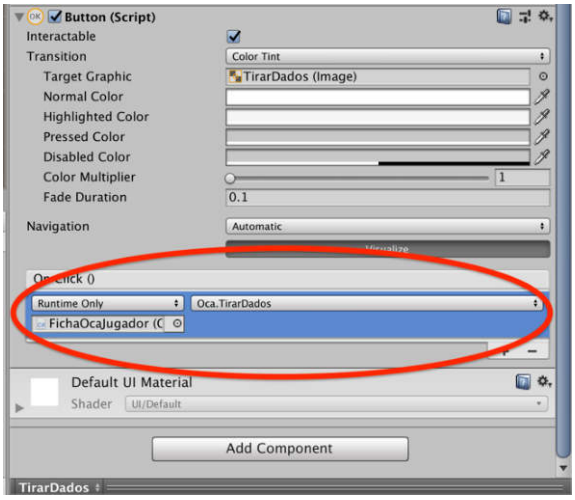


Figura 4.112: Asociar la función TirarDados en el botón Tirar

Por otro lado, se mostrará la funcion “TextoJugador()”, que dirá que objeto con dicho color le toca tirar.

```
public void TextoJugador()
{
    textoJugador.gameObject.SetActive(true);
    textoCasillaEspecial.gameObject.SetActive(false);
    textoJugador.text = "\n Tira el jugador " + TextColor;


}
```

Figura 4.113: Función TextoJugador()

A continuación se realiza una prueba, para así comprobar que funciona el dado:

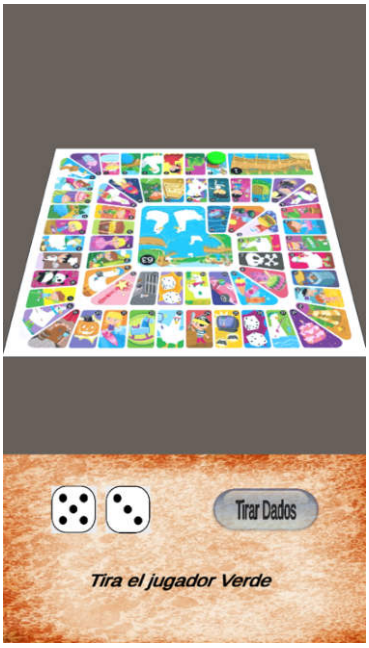


Figura 4.114: Prueba de funcionamiento del botón Tirar Dados

Con esto, esta función estaría terminada. Solo faltaría que cuando el jugador este en la casilla 60, eliminar uno de los dados. Hay que recordar que esta es una de las reglas del juego, que cuando se este a partir de la casilla 60 solo se juega con un dado. Simplemente, se le tiene que decir que cuando esté en esa casilla, se elimina un dado y que haga el proceso de obtener un número aleatorio del dado y asociarlo al botón, y además se centrará la posición del dado.

También se añade que cuando el jugador no tiene que perder ningún turno se ejecute todo lo anterior, y que si el jugador cae en una casilla que pierde turno, esa lista va a disminuir uno a uno hasta completar las veces que tiene que perder el turno, y que cambie de jugador. Pero esto último, se comentara en la función MoverFicha().

A continuación se muestran el código completo de esta función.

```
public void TirarDados()
{
    if (listaPerderTurno[CurrentPlayer] == 0)
    {

        if (listaCambioPosiciones[CurrentPlayer] >= 59)
        {
            Boton.gameObject.SetActive(false);//Desactivamos el boton para que no aparezca

            Boton.sprite = null;//Eliminamos las imaganes porque sino se queda con la ultima imagen guardada

            num1 = UnityEngine.Random.Range(0, dados1.Length);//Numero aleatorio del segundo dado

            Boton1.sprite = dados1[num1];//Para asociar el boton1 que es una imagen a las imagenes del dado

            float Posx = -36.7f;
            float Posy = -9f;
            float Posz = 0f;

            Boton1.rectTransform.anchoredPosition = new Vector3(Posx, Posy, Posz);//Nueva posicion cuando solo hay un dado

            suma = num1;
            suma = suma + 1;

        }
        else
        {

            Boton.gameObject.SetActive(true);//Para activar el boton que se desactivo cuando estaba en la posicion 60

            num = UnityEngine.Random.Range(0, dados.Length);//Numero aleatorio del primer dado

            Boton.sprite = dados[num];//Para asociar el boton que es una imagen a las imagenes del dado

            num1 = UnityEngine.Random.Range(0, dados1.Length);//Numero aleatorio del segundo dado

            Boton1.sprite = dados1[num1];//Para asociar el boton1 que es una imagen a las imagenes del dado

            Boton1.rectTransform.anchoredPosition = new Vector3(-53f, -9f, 0f);// Le damos la posicion que tenia con anterioridad al
            //desactivar el boton

            suma = num + num1; //Suma de ambos dados
            suma = suma + 2;


        }

        TextoJugador();//Indica el jugador que tira

        PosicionPorRecorer = 62 - listaCambioPosiciones[CurrentPlayer]; //Averiguamos cuantas posiciones le queda por recorrer

        resta = suma - PosicionPorRecorer; //Averiguamos la resta, para cuando la ficha vaya hacia atras

        MoverFicha(suma);//Activa la funcion y recorre las posiciones de la suma de ambos dados

        ClipButtonDados();//Activa el sonido de los dados
    }
    else
    {
        CambiarPlayer();

        listaPerderTurno[CurrentPlayer]--;

        TextoPerderTurnos();
    }
}
```

Figura 4.115: Vista completa de la función TirarDados()

4.7.3.2. Función MoverFicha

En esta función se va a comentar el movimiento de la ficha cuando en la posición inicial, el dado sale un 5 y 4 o 4 y 5, y cuando la ficha sale un 6 y 3 o 3 y 6, haciendo un cambio de posición mediante el uso de “transform.position” a la cual se accede mediante el array de posiciones y se le asignará la posición en concreto. Se recuerda que esta es otra regla que presenta el juego de la oca y el movimiento de la ficha en función de la suma de los dados.

Para lo primero que he comentado, se realiza mediante una condición de tipo “if”, diciéndole que si esta la posición “0” de la lista (se recuerda que nuestra listaCambioPosicion empieza por 0 en la posición original) y que si el numero del primer dado es 5 y el numero del segundo dado es un 4, o viceversa, que cambie la posición a la casilla 53 usando transform.position (y se le pasará del array de posiciones, la posición a la que se quiera ir para cada una de las coordenada (x,y,z),) a la lista de jugadores del

jugador que le tocaba tirar. Una vez cambiado la posición se tiene que guardar el valor, para ello simplemente se le tiene que decir que listaCambioPosicion[CurrentPlayer]=52), esto se hace con el fin de que cuando vuelva a tirar no empiece a moverse desde el inicio sino de la casilla 53.

Para el otro caso es lo mismo, solo cambiando el valor del número del dado, y guardar la listaCambioPosicion[CurrentPlayer]=25 (casilla 26).

```
public void MoverFicha(int suma)
{

    if (listaCambioPosiciones[CurrentPlayer] == 0 && ((num == 5 && num1 == 2) || (num == 2 && num1 == 5)))//Al sacar un 9 en el inicio  puede ir a la casilla 26
    {
        movimiento = true;

        TextoAvanza26();

        Vector3 lista25 = ruta.listaPosicion[25].position;

        listaPlayer[CurrentPlayer].transform.position = new Vector3(lista25.x, lista25.y, lista25.z);

        listaCambioPosiciones[CurrentPlayer] = 25;

        TerminaCasillaEspecial();

        CambiarPlayer();

    }
    else if (listaCambioPosiciones[CurrentPlayer] == 0 && ((num == 4 && num1 == 3) || (num == 3 && num1 == 4)))//Al sacar un 9 en el inicio puede ir a la casilla 53
    {
        movimiento = true;

        TextoAvanza53();

        Vector3 lista52 = ruta.listaPosicion[52].position;

        listaPlayer[CurrentPlayer].transform.position = new Vector3(lista52.x, lista52.y, lista52.z);

        listaCambioPosiciones[CurrentPlayer] = 52;

        TerminaCasillaEspecial();

        CambiarPlayer();

    }
```

Figura 4.116: Primera parte de la función MoverFicha(int suma)

Simplemente, añadir a lo comentado que cuando caiga en algunos de los casos que salte por pantalla el mensaje de que el jugador avanza a la casilla 53 o 26, con las funciones “TextoAvanza26()” y “TextoAvanza53()” , a su vez también estos mensajes se quitaran cuando pase un determinado tiempo a través de una corrutina, esta función se llama “TerminaCasillaEspecial()”, y por último, que cuando caiga en estas casillas que cambie de jugador con la función CambiarPlayer(), esta función se comentara más adelante en otra sección de este apartado.

```
public void TextoAvanza26()
{
    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(true);
    textoCasillaEspecial.text = "\n Avanza a la casilla 26 el Jugador " + TextColor;


}
```

```
public void TextoAvanza53()
{

    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(true);
    textoCasillaEspecial.text = "\n Avanza a la casilla 53 el Jugador " + TextColor;


}
```

```
IEnumerator terminaCasillaEspecial()
{
    yield return new WaitForSeconds(1.5f);
    textoCasillaEspecial.gameObject.SetActive(false);
}


public void TerminaCasillaEspecial()
{

    StartCoroutine(terminaCasillaEspecial());

}
```

Figura 4.117: Funciones de texto

También añadir que en ambas condiciones aparece una variable booleana, esto se ha hecho con la intención de que también puede suceder que moviendo la ficha llegue a las casilla 26 y tenga que cambiar a la casilla 53 (esta es otra regla). Este último caso esta dentro de un switch que se verá a continuación, por tanto simplemente se ha hecho para que dos condiciones que son las mismas pero en diferente situación no se unan, por lo que en la imagen de arriba se tiene que esa variable es verdadera y en el caso del switch este en falso.

Con respecto, al movimiento en función de la suma de ambos dados, se va a realizar mediante un bucle “for” que recorra desde 0 hasta la suma, y dentro de ese bucle se incluirá la corrutina de mover la ficha (esta última es la que realiza el movimiento). A continuación se muestra, la imagen que falta de la función “MoverFicha()”.

```
else
{

    for (int i = 0; i < suma; i++)

    {

        StartCoroutine(move(i, suma));

    }


}
```

Figura 4.118: Segunda parte de la función MoverFicha(int suma)

A continuación, se comentará como funciona la corrutina del movimiento. La estructura que forma ese tipo de función es private o public +IEnumerator +” nombre de la función ()”. En este caso será de tipo privado y se llamara “move” y además, se le pasará dos variables de tipo entero (ambas variables son las mismas que constituye el bucle “for”).

En este tipo de funciones, se le puede introducir un tiempo de espera, en nuestro caso se ha utilizado para que haga una parada en cada cambio de posición a través de un tiempo. El comando utilizado es el siguiente:

```
yield return new WaitForSeconds(0.6f * i);//tiempo de parada en cada movimiento
```

Figura 4.119: Tiempo de parada en cada movimiento

Por consiguiente, se utilizará una condición que nos realice que si nuestra listaCambioPosiciones + 1 es mayor o igual al array de posiciones,es decir, si se sale del rango además de si la suma es mayor que las posiciones por recorrer se active la función movimientoOca() para realizar hacia atrás el movimiento así como de esperar el mismo tiempo para que empiece a moverse, y por lo contrario que realice el movimiento de la ficha uno a uno, además para que se realice este movimiento se debe de incluir una variable pública que se activará en este caso para que realice dos funciones (movimientoFicha() y Bloqueo()), es decir, en función de esta variable se llamara a estas dos funciones, como se ve en la siguiente imagen.

```
private IEnumerator move(int i, int suma)
{
    yield return new WaitForSeconds(0.6f * i);//tiempo de parada en cada movimiento

    if (listaCambioPosiciones[CurrentPlayer] + 1 >= ruta.listaPosicion.Count)//si sale del rango y la suma es mayor que realice el movimiento hacia atras
    {
        yield return new WaitForSeconds(0.6f);
        if (suma > PosicionPorRecorer)
        {

            caso = true;

            PosicionAtras = 62 - resta;

            listaNuevaPosicion[CurrentPlayer] = PosicionAtras;

        }

    }

    else // si esta dentro del rango que realice el movimiento
    {

        listaCambioPosiciones[CurrentPlayer] = listaCambioPosiciones[CurrentPlayer] + 1;

        moveficha = true;

        movimientoFicha();

        Bloqueo();

    }
```

Figura 4.120: Corrutina move

Se incluye otra condición, en el cual comprueba si es el último movimiento, si esto es así, se le da un tiempo de espera para que realice el último movimiento y se cambia la variable booleana “moveficha” a falso, además de llamar a la función “Bloqueo()”, para que el botón se vuelva activar.

Dentro de esa condición, se utilizará otra nueva condición que diga que si la variable booleana llamada “movimiento” sea falso, se ejecutará un “switch case” para el movimiento de las casillas especiales como la oca, el puente, las casillas de perder turno, entre otras (esta se explicara en el “movimientoOca” porque están enlazadas a través de una variable booleana llamada “caso”). Se recuerda que la variable “movimiento” se ha utilizado anteriormente, para que cuando al inicio de la partida si sale un 9 se fuera a la casilla 26 o 53 y esa variable era true. Como comenté se ha utilizado la misma variable para dos condiciones que son las mismas pero para situaciones diferentes y que no se junten ambas condiciones.

Además cuando se finaliza la condición de si es el último movimiento que realiza la ficha, se debe de poner la variable booleana “movimiento” a falso.

Por consiguiente, dentro de la condición si es el último movimiento, se añade otra condición que si la variable booleana llamada “VolverTirar” es falsa y la variable booleana llamada “ultimo” es falsa, que cambie el jugador. Esta última variable se explicará más adelante para que se utiliza. A su vez, dentro de esa condiciones, si listaCambioPosiciones es distinta a la casilla 62 así como si es distinta de 0, 29, 51,18 y 30 (se quiere que en estas posiciones no se pare todas las corrutinas porque en ellas tiene otra corrutinas que deben de realizar) que pare todas las corrutinas para evitar un exceso de sobrecargas en la última comprobación, en la función “movimientoOca()”, se explica con más detalle esto último . Esto se realiza de esta manera: debido a que dentro del “switch case” si se cae en alguna oca o puente de la posibilidad de que vuelva a tirar el mismo jugador, haciendo que esta variable será verdadera, y recordar que se tiene que volver a poner la variable boolena “VolverTirar” a falso fuera de la condición.

A continuación, se muestra el código explicado anteriormente. El switch case se incluirá en el apartado de función “movimientoOca”, como he dicho esta enlazado por una variable boolena llamada “caso”.

Figura 4.121: Código del último movimiento de la corrutina move

- ***movimientoFicha***

En esta función, se aplicará un tipo de animación para el movimiento y hará que la ficha se mueva a la siguiente posición. Gracias a que en la corrutina se le pasará la lista uno a uno y se irá guardando. El tipo de animación será utilizando la implementación de la función MoveTowards, es una función ya implementada en unity, en el cual se le tiene que incluir tres parámetros:

- Posición inicial
- Posición final

- Velocidad

La estructura de esta función será la siguiente:

```
Vector3 MoveTowards(Vector3 current, Vector3 target, float maxDistanceDelta);
```

Figura 4.122: Estructura de la función MoveTowards

A la cual, esta función se va a modelar para nuestra estructura. En nuestro caso, el "Vector3 current" será el jugador que le toque en ese momento, el "Vector3 target" será nuestro array de posiciones a la cual va a acceder a la "listaCambioPosiciones" (se recuerda que en la corrutina esa "listaCambioPosicion" será igual a listaCambioPosicion +1, para así poder movernos uno a uno en la ruta), y por último "float maxDistanceDelta", será la velocidad a la cual vaya la ficha multiplicado por "Time.deltaTime".

```
public void movimientoFicha()
{
    if (moveficha)
    {
        Vector3 lista = ruta.listaPosicion[listaCambioPosiciones[CurrentPlayer]].position;//no se añade un +1 porque ya lo tiene asignado en la coroutina

        listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista, 2.45f * Time.deltaTime);
    }
}
```

Figura 4.123: Función movimientoFicha()

- ***Bloqueo***

En esta función, lo que hará es que mientras las ficha se este movimiento el botón de pulsar para tirar lo bloquee, y cuando no haya movimiento que el botón de pulsar para tirar este activo. Para bloquear o activar el botón se usara la función de unity "interactable" y se igualará a true o false en función de la condición mencionada anteriormente.

```
public void Bloqueo()
{
    if (!moveficha)
    {
        Tirar.interactable = true;
    }
    else
    {
        Tirar.interactable = false;
    }
}
```

Figura 4.124: Función Bloqueo()

- ***CambiarPlayer***

Lo que esta función realiza es que si ambos dados no son iguales, pase a tirar el otro jugador, y si ambos dados son iguales y no hay ningún tipo de penalización de perder el turno, el mismo jugador vuelve a tirar, pero si ambos dados son iguales y hay penalización pasa a tirar el siguiente jugador, por otro lado en la condición de que si ambos dados son iguales y ha caído en la calavera o en el laberinto que pase el turno, en estos dos casos en el switch case se utilizará una variable booleana llamada “retroceder” y se pondrá en true para que en esta función pueda entrar, así como si da la casualidad de que cae en la casilla 29 y 0 con los dados dobles no pase el turno por lo que la variable “retroceder” este a false.

Además, se añade otra condición tanto si los dados son iguales como si son distintos, que compruebe que si la suma es mayor que las posiciones por recorrer, para que no desactive el botón durante ese movimiento hacia atrás, se le dice que solo lo bloquee, y ya cuando compruebe que ha llegado se desactivará (esto último lo va a realizar en la función movimientoOca()). Para realizar esta función, se va a crear en el editor de Unity, un “Canvas”, al que se llamará “CanvasTurnoOca” y este objeto se encontrará dentro del objeto “TableroOcaJugador”, en el que estará compuesto por un botón llamado “FinalizaTurnoOca”, por tanto la metodología a seguir es, que cuando un jugador pulse el botón de tirar los dados al llegar el último movimiento, esto botón se desactivará y se desbloqueará el botón “FinalizaTurnoOca”, una vez que se pulse sobre este último botón activará el botón de tirar los dados y bloqueará el botón “FinalizaTurnoOca”, todo siempre y cuando ambos dados no sean iguales o exista alguna de penalización tanto con dados iguales como si son distintos. De lo contrario, si ambos dados son iguales y no hay penalización, el botón de tirar los dados estará activo y el botón “FinalizaTurnoOca” bloqueado.

A continuación, se muestra la función “CambiarPlayer()”:

```
public void PasarTurno()
{
    Tirar.gameObject.SetActive(false);

    GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().interactable = true;

    GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().onClick.AddListener(() => BloqueoTurno());

}


public void SeguirTurno()
{
    Tirar.gameObject.SetActive(true);

    GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().interactable = false;
}


public void CambiarPlayer()
{

    if (Boton.sprite != Boton1.sprite)
    {
        if (suma > PosicionPorRecorer)
        {
            Tirar.interactable = false;
        }
        else
        {
            PasarTurno();
        }


    }
    else if (Boton.sprite == Boton1.sprite)
    {

        SeguirTurno();

        if (listaPerderTurno[CurrentPlayer] > 0) //Si los botones son iguales y hay penalizacion cambia el jugador
        {
            if (suma > PosicionPorRecorer)
            {
                Tirar.interactable = false;

            }
            else
            {
                PasarTurno();
            }

        }
        if ((listaCambioPosiciones[CurrentPlayer] == 0||listaCambioPosiciones[CurrentPlayer]==29)&&retroceder)
        {
            PasarTurno();
        }

    }
}
```

Figura 4.125: Función CambiarPlayer()

Por último, se muestra cual es la función que realiza cuando se pulsa sobre el botón de “FinalizaTurnoOca”, y detallar que en la función Start(), este botón estará bloqueado.

```
public void BloqueoTurno()
{
    GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().interactable = false;

    Tirar.gameObject.SetActive(true);
}
```

Figura 4.126: Función BloqueoTurno()

4.7.3.3. Función movimientoOca

En esta función, se va a ejecutar cuando la variable booleana “caso” sea verdadero, y en este caso va a entrar a cada oca o puente del switch case de la corrutina “move”, a la cual se le aplicará otra corrutina que realizará el movimiento de una oca a otra o de un puente a otro puente. Además de los movimiento cuando tenga que retroceder la ficha hacia atrás, estos movimiento se aplicará siempre y cuando ambas listas sean distintas.

```
public void movimientoOca()
{

    if (caso)
    {

        if (listaNuevaPosicion[CurrentPlayer] != listaCambioPosiciones[CurrentPlayer])
        {

            Tirar.interactable = false;
            StartCoroutine(PasoOca());

        }
    }

}
```

Figura 4.127: Función movimientoOca()

La “listaNuevaPosicion”, se ha creado con la intención de que en la corrutina “PasoOca()” vaya desde la “listaCambioPosiciones” hasta la “listaNuevaPosicion”. Para ello en el “switch case” (corroutina move) de cada caso se le pondrá la “listaNuevaPosicion” a la casilla a la que tiene que ir, además de un mensaje de texto que salte cuando caiga en alguna oca o puente, y poner que la variable booleana “VolverTirar” a true, para que de la posibilidad de volver a tirar el mismo jugador. Se verá a continuación el “switch case” de la corrutina “move”:

```
case 4: //De oca a oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 8;
    VolverTirar = true;

    break;
case 8://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 13;
    VolverTirar = true;
    break;
case 13://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 17;
    VolverTirar = true;
    break;
case 17://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 22;
    VolverTirar = true;
    break;
```

```
case 22://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 26;
    VolverTirar = true;
    break;
case 26://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 31;
    VolverTirar = true;
    break;
case 31://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 35;
    VolverTirar = true;
    break;
case 35://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 40;
    VolverTirar = true;
    break;
```

```
case 40://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 44;
    VolverTirar = true;
    break;
case 44: //De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 49;
    VolverTirar = true;
    break;
case 49://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 53;
    VolverTirar = true;
    break;
case 53://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 58;
    VolverTirar = true;
    break;
```

```
case 58://De oca en oca y tiro porque me toca
    TextoOca();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 62;
    ultimo = true;
    break;

case 5://De puente en puente y tiro porque me lleva la corriente
    TextoPuente();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 11;
    VolverTirar = true;
    break;
case 11://De puente en puente y tiro porque me lleva la corriente
    TextoPuente();
    caso = true;
    listaNuevaPosicion[CurrentPlayer] = 5;
    VolverTirar = true;
    break;
```

```
case 57://Calavera, hay que volver a la casilla inicial
    TextoInicial();
    Vector3 lista0 = ruta.listaPosicion[0].position;
    listaPlayer[CurrentPlayer].transform.position = new Vector3(lista0.x, lista0.y, lista0.z);
    listaCambioPosiciones[CurrentPlayer] = 0;
    TerminaCasillaEspecial();
    retroceder = true;
    break;

case 41: //Laberinto, si cae aqui hay que volver a la casilla 30
    TextoRetroceder();
    Vector3 lista30 = ruta.listaPosicion[29].position;
    listaPlayer[CurrentPlayer].transform.position = new Vector3(lista30.x, lista30.y, lista30.z);
    listaCambioPosiciones[CurrentPlayer] = 29;
    TerminaCasillaEspecial();
    retroceder = true;
    break;
case 51://Cárcel, si cae en esta casilla se pierde 3 turnos
    listaPerderTurno[CurrentPlayer] = 3;
    TextoPerderTurnos();
    TerminaCasillaEspecial();
    break;
case 18://Posada, si cae en esta casilla se pierde 2 turnos
    listaPerderTurno[CurrentPlayer] = 2;
    TextoPerderTurnos();
    TerminaCasillaEspecial();
    break;
case 30://Pozo, si cae en esta casilla se pierde 2 turnos
    listaPerderTurno[CurrentPlayer] = 2;
    TextoPerderTurnos();
    TerminaCasillaEspecial();
    break;
```

```
case 25: //si cae esta casilla se ira a la casilla 53
    TextoDados();
    Vector3 lista52 = ruta.listaPosicion[52].position;
    listaPlayer[CurrentPlayer].transform.position = new Vector3(lista52.x, lista52.y, lista52.z);
    listaCambioPosiciones[CurrentPlayer] = 52;
    TerminaCasillaEspecial();
    VolverTirar = true;
    break;
case 52: //si cae esta casilla se ira a la casilla 26
    TextoDados();
    Vector3 lista25 = ruta.listaPosicion[25].position;
    listaPlayer[CurrentPlayer].transform.position = new Vector3(lista25.x, lista25.y, lista25.z);
    listaCambioPosiciones[CurrentPlayer] = 25;
    TerminaCasillaEspecial();
    VolverTirar = true;
    break;
case 62:
    if (suma == PosicionPorRecorer)
    {
        ultimo = true;
        Bloqueo();
        TextoGanador();
        yield return new WaitForSeconds(3f);
        Espera();
    }
    break;
}
```

Figura 4.128: Switch case de la corrutina move

Por último, para terminar de mostrar todas las función del “switch case”, queda las distintas funciones de texto que se usa para cada casilla especial.

```
public void TextoOca()
{
    textoOcaPuente.gameObject.SetActive(true);
    textoJugador.gameObject.SetActive(false);
    textoOcaPuente.text = "De Oca en Oca y tiro porque me toca";

}
public void TextoPuente()
{
    textoOcaPuente.gameObject.SetActive(true);
    textoJugador.gameObject.SetActive(false);
    textoOcaPuente.text = "De Puente en Puente y tiro porque me lleva la corriente";
}

public void TerminaOcaPuente()
{
    textoOcaPuente.gameObject.SetActive(false);

}
```

```
public void TextoDados()
{
    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(true);
    textoCasillaEspecial.text = "De Dado a Dado y tiro porque son cuadrados";
}


public void TextoInicial()
{
    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(true);
    textoCasillaEspecial.text = "Vuelve a la casilla de salida el Jugador " + TextColor;

}


public void TextoRetroceder()
{
    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(true);
    textoCasillaEspecial.text = "Retrocede a la casilla 30 el \n Jugador" + TextColor;

}
```

```
public void TextoPierde3Turno()
{
    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(true);
    textoCasillaEspecial.text = "Pierde 3 turnos el Jugador " + TextColor;

}

public void TextoPierde2Turno()
{
    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(true);
    textoCasillaEspecial.text = "Pierde " + listaPerderTurno[CurrentPlayer] + " turnos el Jugador" + TextColor;
    textoCasillaEspecial.text = "Pierde 2 turnos el Jugador " + TextColor;

}

public void TextoPierde1Turno()
{
    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(true);
    textoCasillaEspecial.text = "Pierde 1 turno el Jugador " + TextColor;

}

public void TextoPierde0Turno()
{
    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(true);
    textoCasillaEspecial.text = "Vuelve a jugar a la proxima el jugador " + TextColor;

}
```

```
public void TextoPerderTurnos()
{
    if (listaPerderTurno[CurrentPlayer] == 3)
    {
        TextoPierde3Turno();
        TerminaCasillaEspecial();
    }
    if (listaPerderTurno[CurrentPlayer] == 2)
    {
        TextoPierde2Turno();
        TerminaCasillaEspecial();
    }
    if (listaPerderTurno[CurrentPlayer] == 1)
    {
        TextoPierde1Turno();
        TerminaCasillaEspecial();
    }
    else if (listaPerderTurno[CurrentPlayer] == 0)
    {
        TextoPierde0Turno();
        TerminaCasillaEspecial();
    }
}
```

```
public void TextoGanador()
{
    textoJugador.gameObject.SetActive(false);
    textoGanador.gameObject.SetActive(true);
    textoGanador.text = "Ha ganado el Jugador " + TextColor;

}
```

Figura 4.129: Funciones de texto

Se pasa a ver a continuación la corrutina “PasoOca()”, y por último se hará un breve resumen para entender el proceso de esta función, ya que esta compuesto de varias corrutinas.

En primer lugar, el funcionamiento sobre el movimiento será lo mismo con respecto al “movimientoFicha()”, por lo que se utilizará la animación “MoveTowards”.

Todo el funcionamiento de la animación, está incluida dentro de un bucle “for” que recorrerá desde “listaCambioPosiciones” hasta “listaNuevaPosicion”, y en función de si la “listaCambioPosiciones” es menor que “listaNuevaPosicion” sumará una posición ó si la “listaCambioPosiciones” es mayor que “listaNuevaPosicion” restará una posición. Por tanto, el cambio de posición hacia el objetivo final se llama “lista”, y la función “moveTowards” estará dentro de este bucle, de forma que se pueda contabilizar el cambio de posición de uno en uno, y en el cual habrá que ponerle un tiempo de espera para que se aprecie el cambio entre posición y posición.

Además como se quiere que durante el movimiento el botón este bloqueado para de esta maneta no hacer trampas, se pone el botón “Tirar” en false.

```
private IEnumerator PasoOca()
{
    float step = 2.45f * Time.deltaTime;

    int incremento = listaCambioPosiciones[CurrentPlayer] < listaNuevaPosicion[CurrentPlayer] ? 1 : -1;

    for (int j = listaCambioPosiciones[CurrentPlayer]; j != listaNuevaPosicion[CurrentPlayer]; j += incremento)//realizar el array dependiendo si lista Cambio Posicones es menor lista Nueva posocion suma 1 de lo contrario resta -1
    {
        Vector3 lista = ruta.listaPosicion[j + incremento].position;// se le añade el incremento para que pase a la siguiente posicion en caso de que caiga en una oca,
        //por ejemplo si cae en la oca 4 que empieze a contar desde 5 y no desde 4 porque sino un movimiento no lo va hacer porque el primero lo cuenta
        //desde la casilla 4 (oca)

        Tirar.interactable = false;

        GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().interactable = false;

        moveficha = false;

        listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista, step);

        yield return new WaitForSeconds(0.7f);

        caso = false;
    }
```

Figura 4.130: Corrutina PasoOca()

Por otro lado, una vez que la ficha llegue a su posición final deseada, se tiene que comprobar que la ficha a llegado. Además hay que guardar la posición para la siguiente tirada, para guardar la posición se tiene que igualar ambas listas (listaCambioPosiciones y listaNuevaPosicion), así como pasado un determinado tiempo el texto se desactive.

```
if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[listaNuevaPosicion[CurrentPlayer]].position)//comprobamos la posicion de la ficha
{

    moveficha = false;

    TerminaOcaPuente();

    listaCambioPosiciones[CurrentPlayer] = listaNuevaPosicion[CurrentPlayer];//guardamos la posicion de la ficha

    if (suma > PosicionPorRecorer && listaCambioPosiciones[CurrentPlayer] != 58 && listaCambioPosiciones[CurrentPlayer] != 62)
    {
        if (Boton.sprite != Boton1.sprite)
        {
            PasarTurno();

        }
        else
        {
            SeguirTurno();
        }
    }


    if (listaCambioPosiciones[CurrentPlayer] == 57 && suma>PosicionPorRecorer)
    {
        TextoInicial();

        Vector3 lista0 = ruta.listaPosicion[0].position;

        listaPlayer[CurrentPlayer].transform.position = new Vector3(lista0.x, lista0.y, lista0.z);

        listaCambioPosiciones[CurrentPlayer] = 0;

        PasarTurno();

    }

    if (listaCambioPosiciones[CurrentPlayer] == 58 && suma > PosicionPorRecorer)
    {
        TextoOca();

        caso = true;

        Tirar.interactable = false;

        GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().interactable = false;

        listaNuevaPosicion[CurrentPlayer] = 62;

    }
```

```
if (listaCambioPosiciones[CurrentPlayer] == 62)
{

    TerminaOcaPuente();

    ultimo = true;

    Bloqueo();

    TextoGanador();

    yield return new WaitForSeconds(3f);

    Espera();
}

else
{

    Bloqueo();
    StopAllCoroutines();

}
}
```

Figura 4.131: Comprobación final de la corrutina PasoOca()

Por consiguiente, puede suceder que la ficha tenga que recorrer posiciones hacia atrás en caso de que sobre posiciones con respecto a la suma o que caiga tanto en la calavera o en penúltima oca, por lo que se realizará lo mismo que el "switch case" de la corrutina "move", como se puede apreciar en la anterior imagen.

También puede suceder, que en el movimiento hacia delante la ficha caiga en la penúltima oca, por lo que entrará en el "switch case" de la corrutina "move", y realice el movimiento desde la penúltima oca hacia la última posición. Por lo que cuando llegue a la última posición se compruebe que ha llegado y se finalice el juego (última condición de la anterior imagen). Cuando realiza esa comprobación se ve en la imagen que se tiene una variable boolena llamada "ultimo" a true. Esto se utiliza para que cuando haya finalizado, el juego el botón de tirar los dados y el botón "FinalizaTurnoOca" se quede bloqueado. Por ello en la función "Bloqueo()", se le añade otra condición y es que si esta variable es true, bloquee el botón "Tirar". A continuación, se vuelve a mostrar la función con este añadido.

```
public void Bloqueo()
{
    if (ultimo)
    {
        Tirar.interactable = false;
        GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().interactable = false;

    }
    else if (!moveficha)
    {
        Tirar.interactable = true;

    }
    else
    {
        Tirar.interactable = false;

    }

}
```

Figura 4.132: Función Actualizada Bloqueo()

Como se puede ver en la imagen anterior, se ha utilizado la función "StopAllCoroutine" para evitar que se produzca sobrecargas en la última comprobación, ya que el método "Update()" actualiza 60 cuadros por segundos, por lo que cuando finaliza una oca y realiza la comprobación final, comprobará 60 veces. Si se utiliza un "Debug.Log",

se puede ver este efecto en la consola. Con esta función, se evita que esa última comprobación final, en vez de 60 true sea de un solo true. Esto se realizará en todos los casos, excepto si cae las casillas de retroceder así como perder turno ya que tiene una corrutina por realizar, y no se puede suspender.

La función “movimientoOca()” se ejecutará en caso de que en el “switch case” de la corrutina “move”, se tenga la variable “caso” a true. Una vez que ha entrado en alguna oca o puente ya sabe la ficha hacia donde tiene que ir porque se le indica la posición a la que tiene que ir la ficha a través de la “listaNuevaPosicion”.

Ahora se esta en la condición de que ambas listas son distintas, por lo que se ejecutará la corrutina “PasoOca()” que recorrerá a través de un bucle de tipo “for” desde la “listaCambioPosiciones” hasta la “listaNuevaPosicion”, y según si la “listaCambioPosiciones” es menor que “listaNuevaPosicion” se incrementará una posición, y si “listaCambioPosiciones” es mayor que “listaNuevaPosicion” se restará una posición.

Por ejemplo si cae en el “case 4”(listaCambioPosiciones), cuando entra en este caso ya sabe que se tiene que ir a la casilla 8 (listaNuevaPosicion), por lo tanto ambas son distintas, y recorrerá el bucle desde la casilla 4 hasta la 8 con un incremento de una posición en cada movimiento.

4.7.3.4. Otras funciones

- ***ClipButtonDados()***

Esta función, realizará el sonido de los dados cuando se pulse el botón de tirar los dados. Además, esta función tiene que ser llamada en la función “TirarDados()”, para así cada vez que se tire se escuche el sonido.

El procedimiento para incluirlo en Unity, primero de todo es descargar el sonido en internet, y importar el audio a la carpeta de sonido en Unity, como aparece en la siguiente imagen.

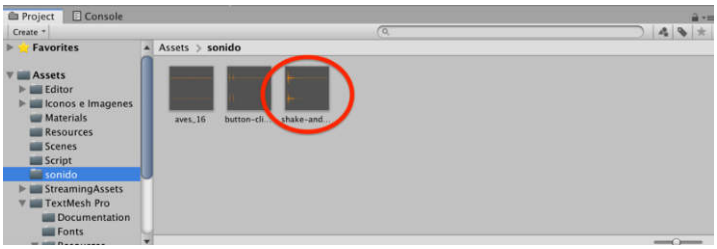


Figura 4.133: Audio al tirar los dados

Después, en el script “Oca.cs”, se declara dos variables públicas, una de tipo “AudioClip” que almacena el archivo de audio comprimido y la otra variable de tipo

“AudioSource” que hace referencia a los AudioClip y se utiliza para poder reproducir el sonido.

```
public AudioSource audioTirar;

public AudioClip audioDados;
```

Figura 4.134: Declaración de las variables del sonido al tirar los dados

Una vez declaradas la variables públicas, se procederá a crear la función llamada “ClipButtonDados()”, utilizando la función ya creada en unity llamada “PlayOneShot”, que hará que se reproduzca el sonido cada vez que se pulse, en este caso sobre el botón para tirar los dados.

```
public void ClipButtonDados()
{
    audioTirar.PlayOneShot(audioDados);//audio a tirar los dados
}
```

Figura 4.135: Función que reproduce el sonido al tirar los dados

En Unity, se crea un objeto vacío llamado “TirarDados” dentro de la jerarquía “TableroOcaJugador”, en el cual se le añadirá un objeto de tipo “AudioSouce”, en el cual se arrastrará el audio de los dados, ubicado en la carpeta de sonido.

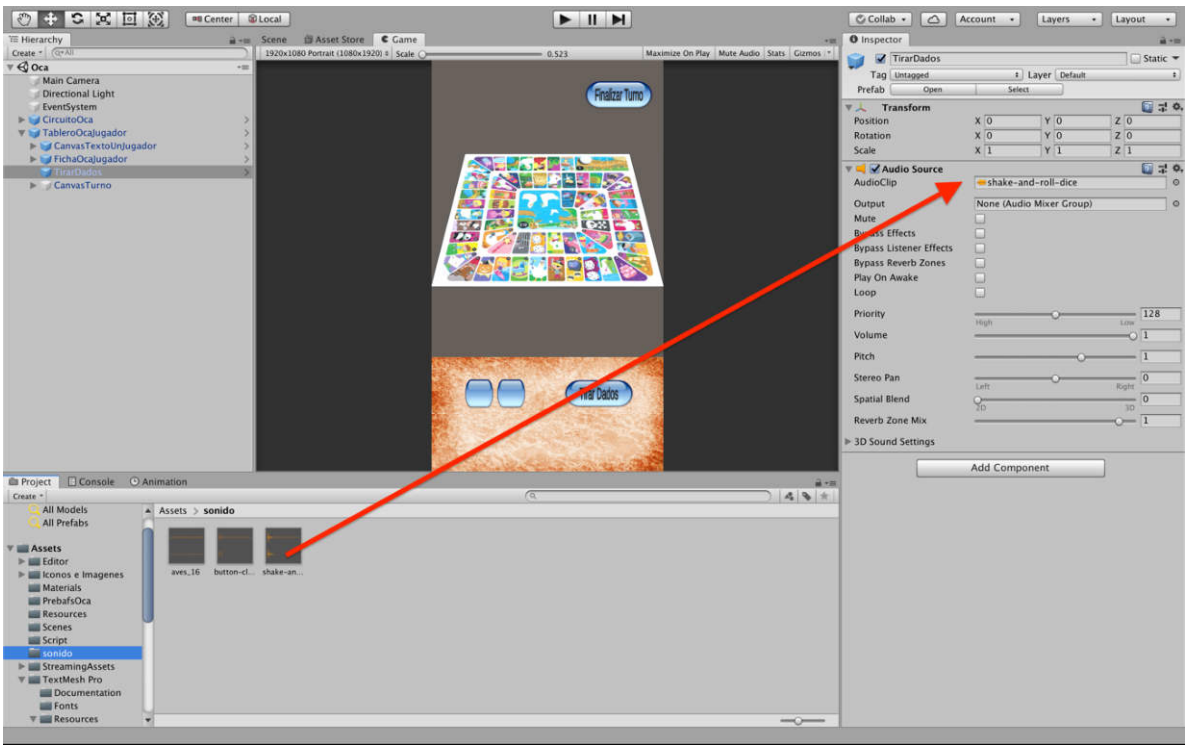


Figura 4.136: Añadir el sonido al componente AudioSource

Por último, en el gameObject “FichaOcaJugador”, donde esta asociado el script “Oca.cs”, se le asociará tanto el gameObject “TirarDados” que contiene el “AudioSource” y el “AudioCip” de la carpeta sonido, tal y como aparece en la siguiente imagen.

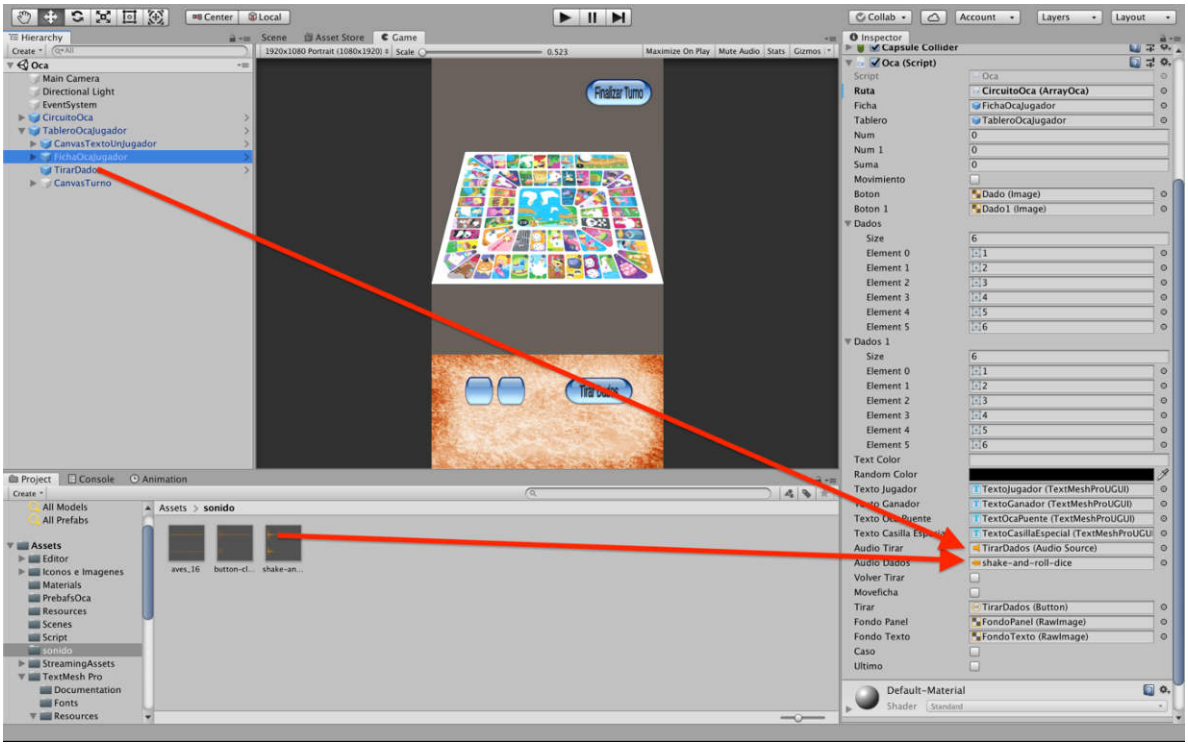


Figura 4.137: Asociar los componentes en las variables declaradas

- ***ReiniciarJuego() y SalirJuego()***

Estas funciones, se utilizará una vez que el juego haya acabado, saltará a una nueva escena a la que se llamará “FinalOca”, dando la posibilidad de reiniciar el juego o salir de la app.

Además, esta escena estará incluida dentro de una función llamada “Espera()”, que hará que cambie de escena mientras que el resto de los objetos que se han utilizado estén desactivados. Por tanto, esta función va a estar llamada en dos sitios, una de ellas cuando cae en la última oca y cuando la suma de los dados llegue a la última casilla con los dados justo, es decir, que no tenga que retroceder la ficha hacia atrás.

A continuación, se explicará el proceso de incluir los botones en Unity en la nueva escena y como enlazarlos con la función especifica para cada caso. Los botones, se creará desde la ventana de jerarquía, en el cual se procederá de la misma manera que se ha realizado en otras ocasiones.

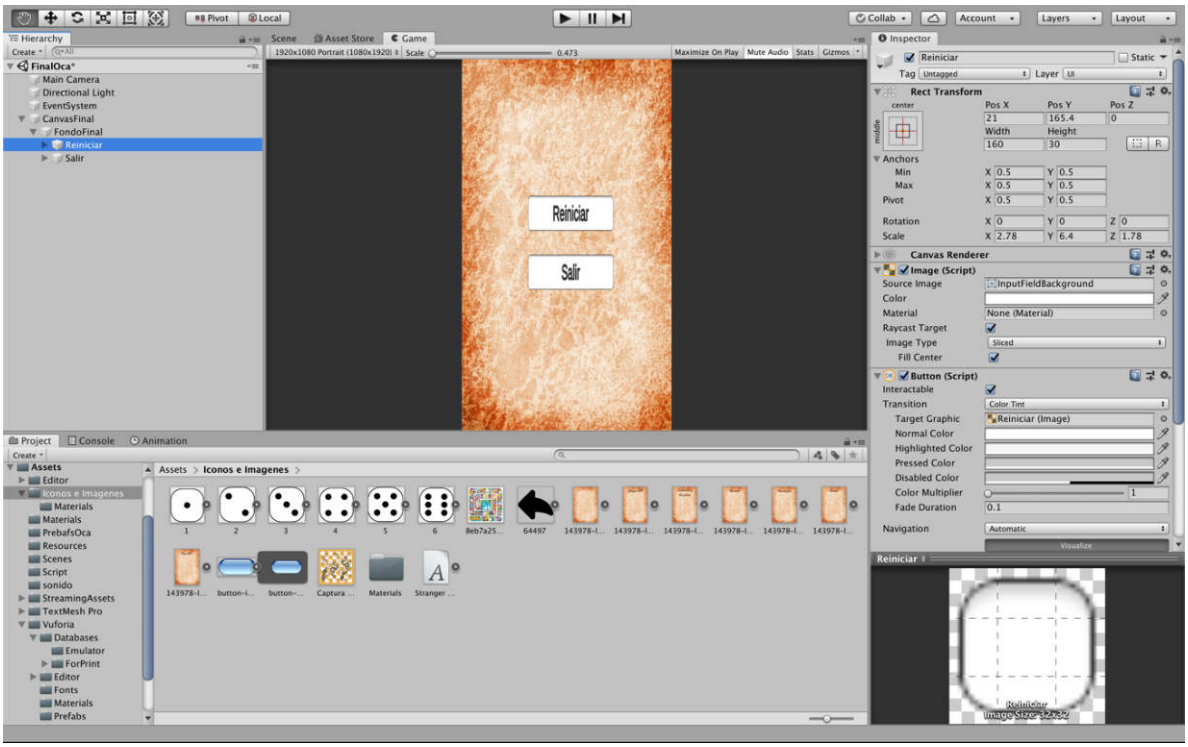


Figura 4.138: Vista preliminar de la escena "FinalOca"

Una vez creado, se centra los botones para que aparezca justo en el centro de la ventana "Game", y se le cambiará el nombre, hay que tener en cuanta que se ha utilizado el tipo de texto "TextMeshPro", debido a que presenta más calidad. Además, también se utilizará un icono en particular, que se ha descargado por internet, y se le ha añadido a la carpeta de "Iconos e Imagenes", a la cual hay que cambiarle el tipo de textura a "Sprite", como se muestra a continuación en la siguiente imagen.

Figura 4.139: Tipo de textura del botón

Una vez cambiado el tipo de textura, ya se le puede incluir en ambos botones arrastrando la imagen del botón a “Source Imagen” situado en la ventana inspector.

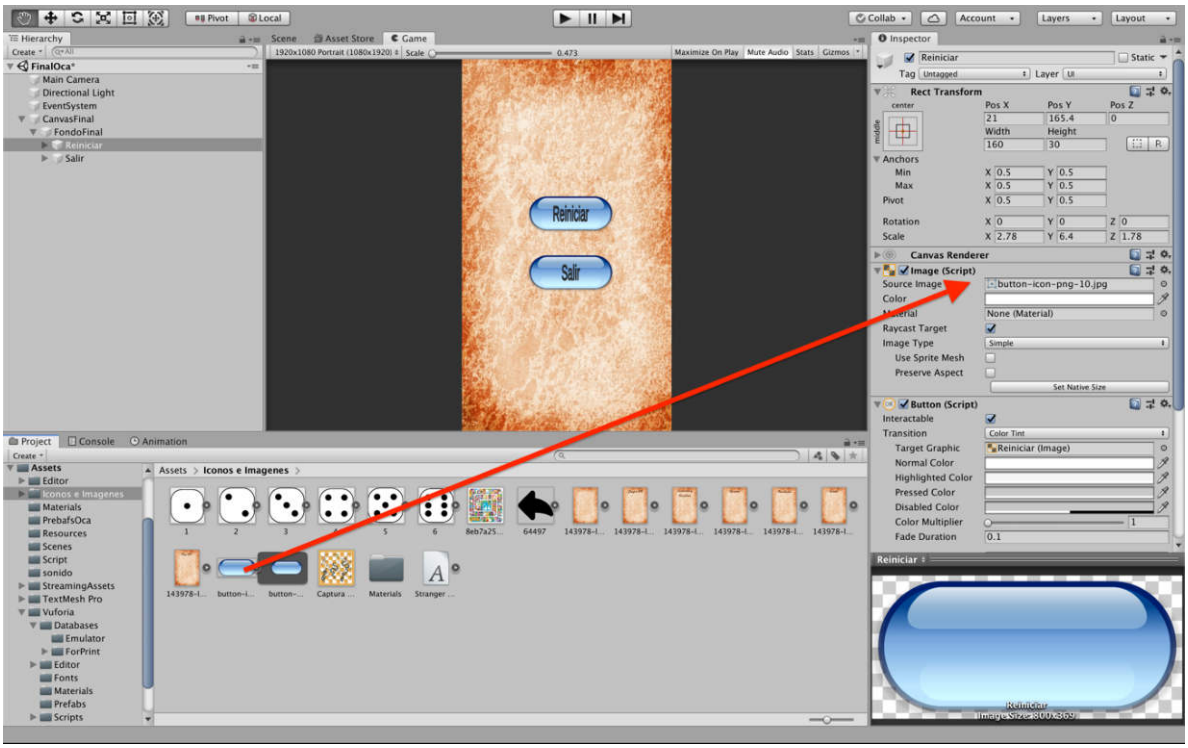


Figura 4.140: Añadir el icono del botón

A continuación, se verá primeramente las funciones de los botones, y por último la función en sí.

Para la realización de la función “ReiniciarJuego()”, se necesita importar la librería “UnityEngine.SceneManagement”.

Una vez que ya se tiene importada la librería, se puede hace uso de ella para acceder al método “LoadScene”, y se le indicará entre comillas a que escena quiere cargar, tal y como se muestra a continuación en la función “ReiniciarJuego()”.

```
public void ReiniciarJuego()
{
    SceneManager.LoadScene("Oca");

}
```

Figura 4.141: Función ReiniciarJuego()

Realizada la función, se irá a la ventana de jerarquía de Unity donde esta ubicado el botón “Reiniciar” para poder asociarle la función “ReiniciarJuego” en la ventana de inspector, de esta manera cuando se pulse sobre este botón, se volverá a recargar la misma escena. Recordar que al cambiar de escena, no se tiene el objeto, por lo que el objeto de la escena “Oca” se pondrá como un “prebafs” para así poder utilizarlo.

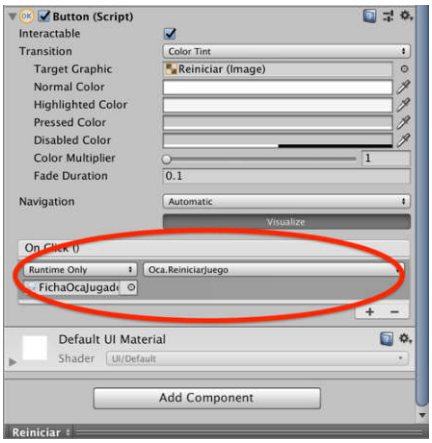


Figura 4.142: Añadir la función al botón Reiniciar

Ya solo quedaría realizar de forma similar el botón de salir del juego, lo único que cambiaria con respecto al de reiniciar el juego es la creación de la función.

```
public void SalirJuego()
{
    Application.Quit();
}
```

Figura 4.143: Función SalirJuego()

Simplemente, para la realización de esta función se accede a la clase “Application” para así acceder a su vez al método “Quit”, en el cual con este método va a cerrar la aplicación en cuanto se pulse sobre el botón.

Se procede de manera igual para asociarle el botón “Salir” ubicado en la ventana de jerarquía, la función “SalirJuego” en la ventana de inspector.

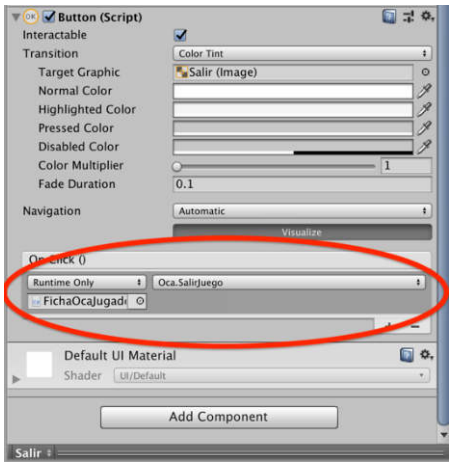


Figura 4.144: Añadir la función al botón Salir

Por último, se procederá a crear la función llamada “Espera()”, simplemente aquí se va ha activar o desactivar los objetos que se quiere utilizar y los que no. Para ello se utilizará

el método “SetActive” a la cual se le pasara un valor que en este caso es true o false, tal y como se muestra a continuación.

```
public void  Espera()
{

    ficha.gameObject.SetActive(false);
    Boton.gameObject.SetActive(false);
    Boton1.gameObject.SetActive(false);
    Tirar.gameObject.SetActive(false);
    Tablero.gameObject.SetActive(false);
    textoGanador.gameObject.SetActive(false);
    textoJugador.gameObject.SetActive(false);
    textoCasillaEspecial.gameObject.SetActive(false);
    textoOcaPuente.gameObject.SetActive(false);
    FondoPanel.gameObject.SetActive(false);
    FondoTexto.gameObject.SetActive(false);
    SceneManager.LoadScene("FinalOca");
}
```

Figura 4.145: Función Espera()

Como se comento antes, esta función será llamada en dos ocasiones, una de ella cuando cae en la última oca, y cuando la posiciones que le falta por recorrer sea la misma que la suma de ambos dados.

4.8. Desarrollo del Juego de la Serpiente y Escalera

En esta sección todo lo que se comentará, estará creado en una escena llamada “RestauranteMesaSerpiente” y la escena llamada “Serpiente”.

Con lo que respecta a la primera escena se realiza de la misma manera que en la oca, la única diferencia es que en la selección de mesas en vez de cargar la escena “Oca”, se cargará la escena “Serpiente”.

Para el desarrollo del juego de la serpiente y escalera, primero se empezará con la creación del tablero, seguidamente de las fichas y por último del movimiento y de efectos incluidos. Algunas de estas tareas son las misma que en el juego de la oca, por lo que no se explicará el proceso de crearlas.

4.8.1. Creación del tablero

Esta es una de las tareas idénticas a la de la oca, por lo que se mostrara solo la imagen del tablero con la ruta ya marcada.

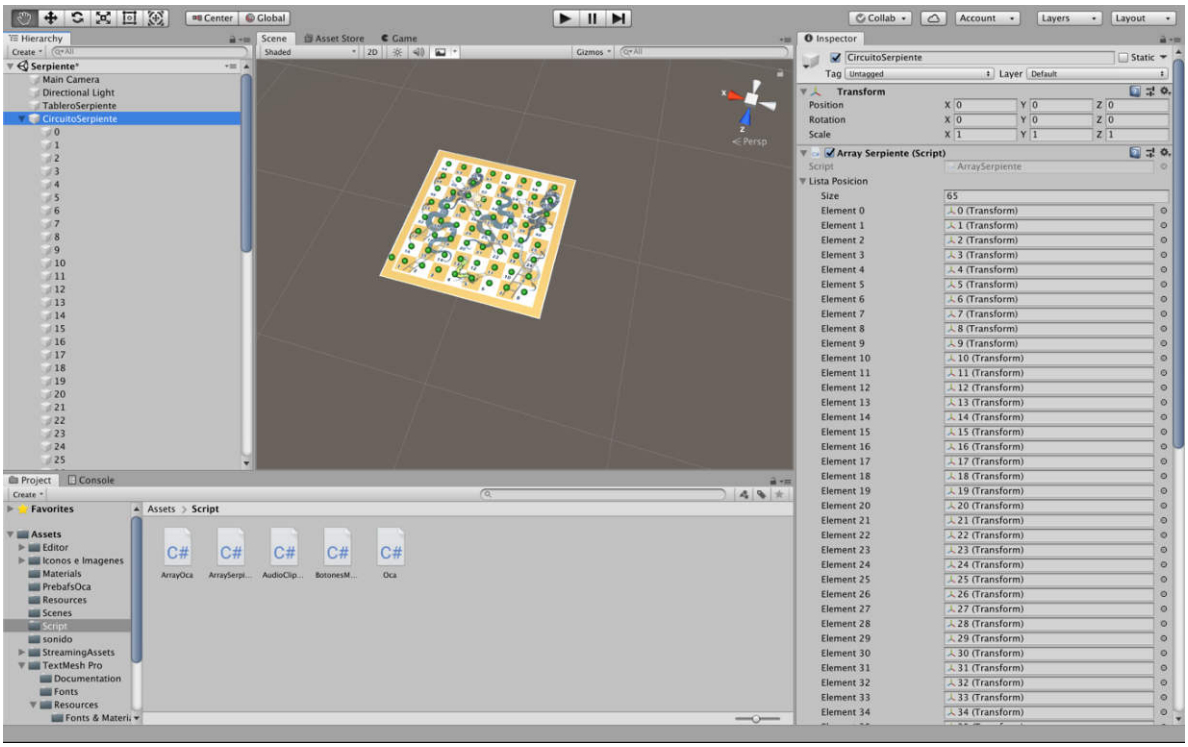


Figura 4.146: Ruta del tablero Serpientes y Escaleras

Se realiza el mismo procedimiento que se uso para el juego de la oca, se asigna el objeto llamado “CircuitoSerpiente”, al script “Serpiente”, de forma que tengamos referenciada el objeto con la ruta que debe de seguir.

```
public ArraySerpiente ruta;
```

Figura 4.147: Declaración de la variable ruta

A continuación, se muestra la asignación comentada anteriormente:

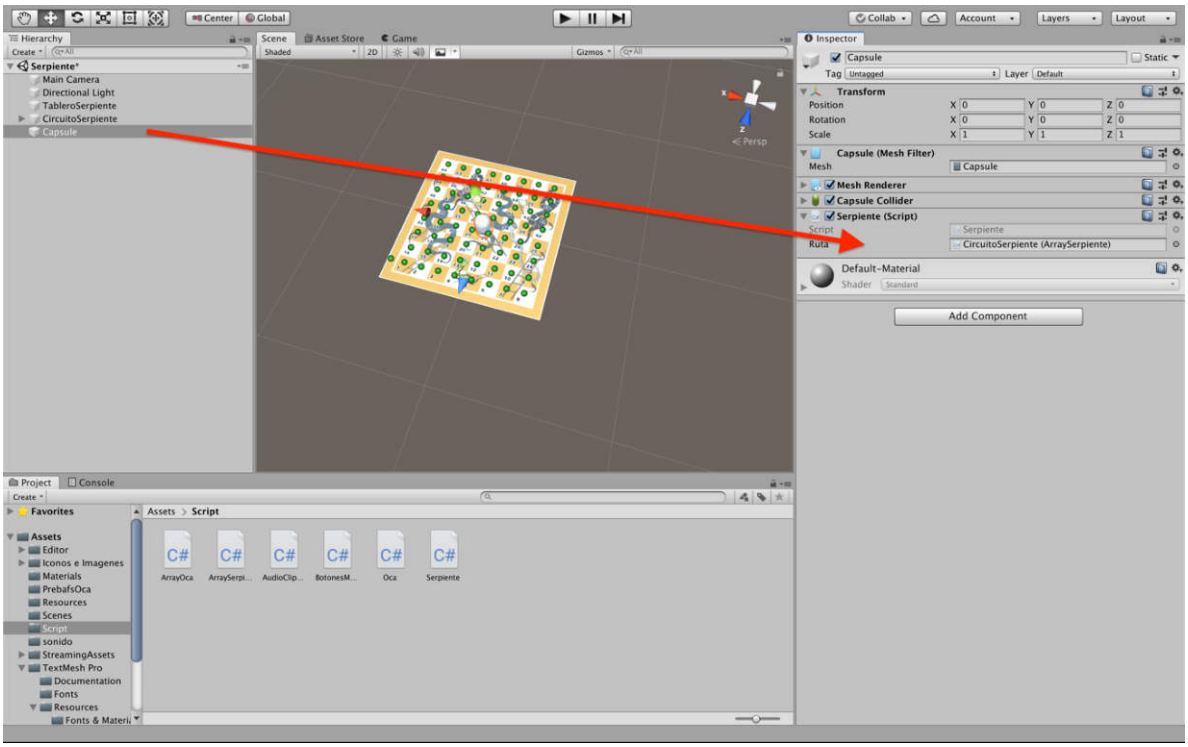


Figura 4.148: Añadir la ruta al objeto

4.8.2. Creación de las fichas

Esta sección, es la misma que la del juego Oca, lo que se modifica será la posición y la escala, ya que las posiciones por la que va a recorrer nuestra ficha es de ancho más pequeña que la Oca, a continuación, se muestra la imagen con dicha posición y escala de nuestra ficha.

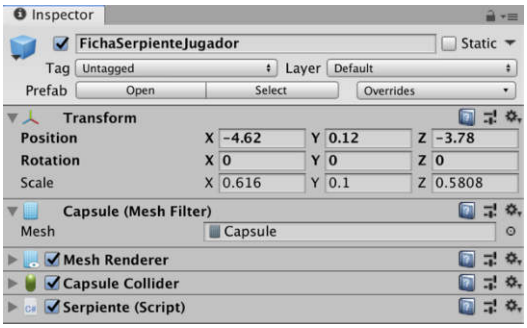


Figura 4.149: Modificación de los parámetros del objeto ficha

4.8.3. Creación del movimiento

En esta sección se va a dividir en tres partes. Una de ellas es la función tirar el dado, que cuando pulsas se activa la función MoverFicha(), y la última es la función Escala(), esta función se va activar cuando un jugador caiga en una escalera o en una serpiente.

4.8.3.1. Función TirarDado()

En esta función, lo que va a realizar es elegir un número de 1 al 6 de forma aleatoria, y se le va a asignar al botón del dado. Además en el código la suma va a ser la del dado más uno porque cuando se le asigna el array de las caras del dado el primero elemento es 0, por lo que el elemento 0 corresponde al dado numero 1. Además también se actualiza por texto a que jugador le corresponde jugador, llamada TextoJugador(), por otro lado se le pasará la función “MoverFicha()”, y la función “ClipButtonDados()”, que corresponde con el sonido de tirar los dados.

En la ventana de jerarquía, en el objeto “FichaSerpienteJugador”, estará formado por un objeto de tipo “Canvas”, al que se llamará “CanvasPanel”, en el cual contendrá un objeto de UI de tipo “RawImagen”, al que se llamará “FondoPanel”, y esta estará compuesta por el botón de “TirarDado”y el botón “Dado”.

El objeto “FichaSerpienteJugador”, será hijo del objeto “TableroSerpienteJugador”, asi como otro objeto de tipo “Canvas” al que se llamará “CanvasTextoUnJugador”, que estará compuesto por un objeto de interfaz de usuario “RawImagen” llamado “FondoTexto”, y dentro de este estará todos los objetos de texto que contempla este juego, que son varios, uno para el jugador que tira, otro por si un jugador cae en la escalera o serpiente, y el último para indicar que jugador a ganado.

A continuación se mostrará dicha jerarquía comentada, también decir que el objeto de tipo audio se añadirá más adelante, pero formará parte del objeto “TableroSerpienteJugador”.

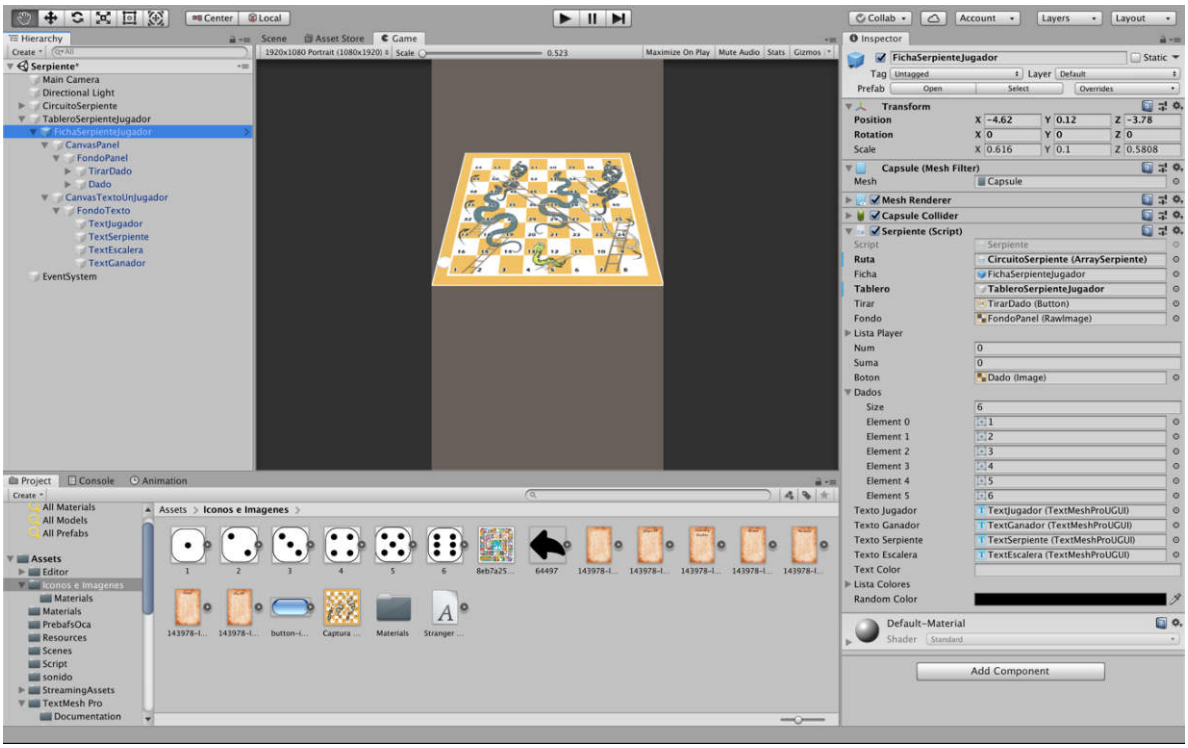


Figura 4.150: Jerarquía del objeto FichaSerpienteJugador

El script “Serpiente.cs”, se le asociará al objeto “FichaSerpienteJugador”, de forma que este todos los elementos referenciados con las variables que se han creado en dicho script. Por último, se mostrará la función de TirarDado (), y la función TextoJugador():

```
public void TirarDado()
{
    num = UnityEngine.Random.Range(0, dados.Length);//numero aleatorio del primer dado

    Boton.sprite = dados[num];//para asociar el boton que es una imagen a las imagenes del dado

    suma = num;

    suma = suma + 1;

    MoverFicha(suma);

    TextoJugador();

    ClipButtonDados();

    PosicionPorRecorrer = 64 - listaCambioPosiciones[CurrentPlayer];

    Debug.Log("Posicion por recorrer" + PosicionPorRecorrer);

    resta = suma - PosicionPorRecorrer;

    Debug.Log("la resta es :" + resta);


}


public void TextoJugador()
{
    textoJugador.gameObject.SetActive(true);
    textoJugador.text = "Tira el jugador " + TextColor;
}
```

Figura 4.151: Función TirarDado() y función TextoJugador()

4.8.3.2. Función MoverFicha

En esta función, va a realizar el movimiento en función del número que de el dado a través de un bucle de tipo “for”, en el que va a recorrer desde el valor 0 hasta el número del dado, en el que se ha guardado su valor en la variable llamada “suma”, el bucle for va a estar compuesto por una corrutina llamada “move”. A continuación se muestra una imagen de esta función, y se explicará esta corrutina.

```
public void MoverFicha(int suma)
{
    for (int i = 0; i < suma; i++)
    {
        StartCoroutine(move(i, suma));
    }
}
```

Figura 4.152: Función MoverFicha()

A continuación, se mostrará la corrutina “move”, esta es la misma que en el juego de la oca.

```
private IEnumerator move(int i, int suma)
{
    yield return new WaitForSeconds(0.6f * i);//tiempo de parada en cada movimiento

    if (listaCambioPosiciones[CurrentPlayer] + 1 >= ruta.listaPosicion.Count)//si sale del rango y la suma es mayor que realice el movimiento hacia atras
    {
        yield return new WaitForSeconds(0.6f);

        if (suma > PosicionPorRecorrer)
        {
            atras = true;

            PosicionAtras = 64 - resta;

            listaNuevaPosicion[CurrentPlayer] = PosicionAtras;


        }
    }

    else   // si esta dentro del rango que realice el movimiento
    {

        listaCambioPosiciones[CurrentPlayer] = listaCambioPosiciones[CurrentPlayer] + 1;

        moveficha = true;

        movimientoFicha();

        Bloqueo();


    }
```

Figura 4.153: Corroutina move

En la condición, que comprueba si es el último movimiento, se da un tiempo de espera para que pueda realizar ese último movimiento y poner la variable boolena “moveficha” a false. Además, se le pasará la función “Bloqueo()”, para que active el botón de tirar, así como desactivar el texto. En la condición de si es el último movimiento se hará otra comprobación que compruebe que si la “listaCambioPosiciones” no es la última casilla, que cambie de jugador, así como parar la corrutina, tal y como se hizo con la oca y por el mismo motivo que se explicó.

Dentro de la condición de si es el último movimiento, se va crear un “switch case” y se le indicará si la ficha esta en la posición 64 y la suma sea igual a las posiciones por recorrer indica quien ha ganado y salte la función en el que desaparecerá todos los objetos como botones, tablero, texto, ficha y nos cambie a una nueva escena en el que indique si quiere reiniciar el juego o bien salir de la aplicación.

```
if (i == suma - 1)//comprueba si es el ultimo movimiento
{

    yield return new WaitForSeconds(0.6f);//le damos este tiempo de espera para que finalice su ultimo movimiento y cuando pase ese tiempo se desactiva la variable booleana

    moveficha = false;

    Bloqueo();

    TextoJugadorTermina();


        switch (listaCambioPosiciones[CurrentPlayer])
        {
            case 64:

                if (suma == PosicionPorRecorrer)
                {

                    ultimo = true;
                    Bloqueo();
                    TextoGanador();
                    yield return new WaitForSeconds(3f);
                    Espera();
                }

                break;
        }

    if (listaCambioPosiciones[CurrentPlayer] != 64)
    {

        CambiarPlayer();
        StopAllCoroutines();

    }

}
    Debug.Log("ultimo" + i);

}
```

Figura 4.154: Código del último movimiento de la corroutina move

- ***movimientoFicha***

En esta función, es la misma que en la oca, como se comento en ese apartado lo que realizar es aplicar un tipo de animación al movimiento, la animación es como si la ficha la estuvieras empujando, esto dará algo más realidad a nuestro juego.

- ***Bloqueo***

Al igual que la función anterior, es la misma, lo que realizará es que cuando la ficha se este movimiento el botón no este activo, para así evitar que haya trampas en el juego, y cuando acabe el movimiento el botón se activará para poder tirar otra vez.

- ***CambiarPlayer***

Esta función, lo que va a realizar es el cambio del jugador cuando el dado sea distinto a seis, debido a que es un regla, que si un jugador saca un seis tiene derecho a otra tirada de dados. Para realizar esta función, se va a crear en el editor de Unity, un “Canvas”, al que se llamara “CanvasTurnoSerpiente” y se encontrará dentro del objeto “TableroSerpienteJugador”, en el que estará compuesto por un botón llamado “FinalizaTurnoSerpiente”, por tanto, la metodología a seguir es que cuando un jugador pulse el botón de tirar el dado al llegar al último movimiento. Este botón se desactivará y se desbloqueará el botón “FinalizaTurnoSerpiente”, una vez que se pulse sobre este último

botón activará el botón de tirar el dado y bloqueará el botón “FinalizaTurnoSerpiente”, todo siempre y cuando el dado sea distinto a 6, de lo contrario a esto, estará el botón de tirar el dado activo y el botón “FinalizaTurnoSerpiente” bloqueado.

A continuación, se muestra esta función:

```
public void PasarTurno()
{
    tirar.gameObject.SetActive(false);

    GameObject.Find("FinalizaTurnoSerpiente").GetComponent<Button>().interactable = true;

    GameObject.Find("FinalizaTurnoSerpiente").GetComponent<Button>().onClick.AddListener(() => BloqueoTurno());
}


public void SeguirTurno()
{
    tirar.gameObject.SetActive(true);

    GameObject.Find("FinalizaTurnoSerpiente").GetComponent<Button>().interactable = false;
}
public void CambiarPlayer()
{
    if (num != 5)//cambia el jugador
    {

        PasarTurno();

    }

    else if (num == 5) //jugador juega otro turno
    {

        SeguirTurno();

    }

}
```

Figura 4.155: Función CambiarPlayer()

Por último, se muestra cual es la función que realiza cuando se pulsa sobre el botón de “FinalizaTurnoSerpiente”, y detallar que en la función Start(), este botón estará bloqueado.

```
public void BloqueoTurno()
{
    GameObject.Find("FinalizaTurnoSerpiente").GetComponent<Button>().interactable = false;

    tirar.gameObject.SetActive(true);
}
```

Figura 4.156: Función BloqueoTurno()

4.8.3.3. Función Escala

Esta función lo que va a realizar es que cuando la ficha caiga en el principio de una escalera suba hasta el final, pero nunca al contrario. Por contraparte si caes en la cabeza de una serpiente bajaras hasta su cola pero nunca al contrario. Se ha desarrollado de manera diferente a cuando una ficha caiga en la oca debido a que en esta función directamente se le asigna la posición en concreto y puede atravesar el tablero de lo contrario de la oca si lo hacías de esta manera atravesaba el tablero y del punto de vista de realidad no era así.

La función se va a ejecutar cuando la variable booleana “moveficha” sea falso y caiga en algunas de estas casillas, se utiliza la variable booleana debido a que cuando tiras el dado y se mueve la ficha la variable es verdadera y cuando ha terminado de moverse la variable se pone en falso, de manera que si cae en alguna casilla se activa esta función.

Por otro lado, si cae en alguna de estas casilla se le pasará una función que lo que va a realizar es que ambos botones se bloqueen. La función se llama “BloqueoSerEsca()” y se le pasará el texto si cae en una escalera la función llamada “TextoEscalera()”, de lo contrario si cae en una serpiente, se le pasará la función llamada “TextoSerpiente()”. A continuación, se le indicará la posición en concreto, y se utilizará la función “MoveTowards”. Por último se hará una comprobación, en el que se dirá, si la ficha esta en esa posición, si la respuesta es que si, se guarda la posición y se le pasará la función que haga que desbloque el botón llamada “DesbloquearSerEsca()” y en función de si es una serpiente o escalera, se desactivará los texto correspondientes. Además de pasarle también la función “CambiarPlayer”.

A continuación, se mostrara primero las función de dichos textos y la de bloquear y desbloquear el botón.

```
public void TextoEscalera()
{
    textoEscalera.gameObject.SetActive(true);
    textoEscalera.text = "Subo por la escalera";
}
```

```
public void TextoSerpiente()
{
    textoSerpiente.gameObject.SetActive(true);
    textoSerpiente.text = "Bajo a la cola de la serpiente";
}
```

```
public void TextEscaleraTermina()
{
    textoEscalera.gameObject.SetActive(false);
}
```

```
public void TextSerpienteTermina()
{
    textoSerpiente.gameObject.SetActive(false);
}
```

```
public void BloqueoSerEsca()
{
    if (!moveficha)
    {
        tirar.interactable = false;
        GameObject.Find("FinalizaTurnoSerpiente").GetComponent<Button>().interactable = false;
    }
}
```

```
public void DesbloquearSerEsca()
{
    tirar.interactable = true;
}
```

Figura 4.157: Funciones de los texto y bloqueo del botón Tirar

Por último, para acabar este bloque, se mostrará por completo la función “Escala()”, para entender mejor todo lo explicado anteriormente:

```
public void Escala()
{

    if (moveficha == false)
    {

        switch (listaCambioPosiciones[CurrentPlayer])
        {
            case 1://subo la escalera hasta la posicion 19

                BloqueoSerEsca();
                TextoEscalera();

                Vector3 lista19 = ruta.listaPosicion[19].position;

                listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista19, 2.45f * Time.deltaTime);


                if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[19].position)
                {
                    listaCambioPosiciones[CurrentPlayer] = 19;
                    TextoEscaleraTermina();
                    DesbloquearSerEsca();
                    CambiarPlayer();

                }

                break;

                |
```

```
            case 4://subo la escalera hasta la posicion 11

                BloqueoSerEsca();
                TextoEscalera();

                Vector3 lista11 = ruta.listaPosicion[11].position;

                listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista11, 2.45f * Time.deltaTime);

                if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[11].position)
                {
                    listaCambioPosiciones[CurrentPlayer] = 11;
                    TextoEscaleraTermina();
                    DesbloquearSerEsca();
                    CambiarPlayer();
                }

                break;


            case 7://subo la escalera hasta la posicion 25

                BloqueoSerEsca();
                TextoEscalera();

                Vector3 lista25 = ruta.listaPosicion[25].position;

                listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista25, 2.45f * Time.deltaTime);

                if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[25].position)
                {
                    listaCambioPosiciones[CurrentPlayer] = 25;
                    TextoEscaleraTermina();
                    DesbloquearSerEsca();
                    CambiarPlayer();
                }

                break;
```

```
case 17://subo la escalera hasta la posicion 31

    BloqueoSerEsca();
    TextoEscalera();

    Vector3 lista31 = ruta.listaPosicion[31].position;

    listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista31, 2.45f * Time.deltaTime);

    if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[31].position)
    {
        listaCambioPosiciones[CurrentPlayer] = 31;
        TextoEscaleraTermina();
        DesbloquearSerEsca();
        CambiarPlayer();
    }

    break;

case 22://subo la escalera hasta la posicion 50

    BloqueoSerEsca();
    TextoEscalera();

    Vector3 lista50 = ruta.listaPosicion[50].position;

    listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista50, 2.45f * Time.deltaTime);

    if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[50].position)
    {
        listaCambioPosiciones[CurrentPlayer] = 50;
        TextoEscaleraTermina();
        DesbloquearSerEsca();
        CambiarPlayer();
    }

    break;
```

```
case 42://subo la escalera hasta la posicion 54

    BloqueoSerEsca();
    TextoEscalera();

    Vector3 lista54 = ruta.listaPosicion[54].position;

    listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista54, 2.45f * Time.deltaTime);

    if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[54].position)
    {
        listaCambioPosiciones[CurrentPlayer] = 54;
        TextoEscaleraTermina();
        DesbloquearSerEsca();
        CambiarPlayer();
    }

    break;

case 13: //bajo a la cola de la serpiente 5

    BloqueoSerEsca();
    TextoSerpiente();

    Vector3 lista5 = ruta.listaPosicion[5].position;

    listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista5, 2.45f * Time.deltaTime);

    if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[5].position)
    {
        listaCambioPosiciones[CurrentPlayer] = 5;
        TextoSerpienteTermina();
        DesbloquearSerEsca();
        CambiarPlayer();
    }

    break;
```

```
            case 38: //bajo a la cola de la serpiente 9

                BloqueoSerEsca();
                TextoSerpiente();

                Vector3 lista9 = ruta.listaPosicion[9].position;

                listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista9, 2.45f * Time.deltaTime);

                if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[9].position)
                {
                    listaCambioPosiciones[CurrentPlayer] = 9;
                    TextoSerpienteTermina();
                    DesbloquearSerEsca();
                    CambiarPlayer();
                }

                break;

            case 57: //bajo a la cola de la serpiente 57

                BloqueoSerEsca();
                TextoSerpiente();

                Vector3 lista26 = ruta.listaPosicion[26].position;

                listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista26, 2.45f * Time.deltaTime);

                if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[26].position)
                {
                    listaCambioPosiciones[CurrentPlayer] = 26;
                    TextoSerpienteTermina();
                    DesbloquearSerEsca();
                    CambiarPlayer();
                }

                break;
```

```
            case 53: //bajo a la cola de la serpiente 20

                BloqueoSerEsca();
                TextoSerpiente();

                Vector3 lista20 = ruta.listaPosicion[20].position;

                listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista20, 2.45f * Time.deltaTime);

                if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[20].position)
                {
                    listaCambioPosiciones[CurrentPlayer] = 20;
                    TextoSerpienteTermina();
                    DesbloquearSerEsca();
                    CambiarPlayer();
                }

                break;

            case 47: //bajo a la cola de la serpiente 14

                BloqueoSerEsca();
                TextoSerpiente();

                Vector3 lista14 = ruta.listaPosicion[14].position;

                listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista14, 2.45f * Time.deltaTime);

                if (listaPlayer[CurrentPlayer].transform.position== ruta.listaPosicion[14].position)
                {
                    listaCambioPosiciones[CurrentPlayer] = 14;
                    TextoSerpienteTermina();
                    DesbloquearSerEsca();
                    CambiarPlayer();
                }

                break;
```

```
            case 62: //bajo a la cola de la serpiente 34

                BloqueoSerEsca();
                TextoSerpiente();

                Vector3 lista34 = ruta.listaPosicion[34].position;

                listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista34, 2.45f * Time.deltaTime);

                if (listaPlayer[CurrentPlayer].transform.position == ruta.listaPosicion[34].position)
                {
                    listaCambioPosiciones[CurrentPlayer] = 34;
                    TextoSerpienteTermina();
                    DesbloquearSerEsca();
                    CambiarPlayer();
                }

                break;
        }
    }
}
```

Figura 4.158: Función Escala()

4.8.3.4. Otras funciones

Con respecto a las función del sonido del tirar el dado, y crear otra escena para poder reiniciar o salir, es exactamente igual al de la oca. Lo único que se hará es crear otra escena para estos botones aunque el contexto sea idéntico, debido a que por ejemplo, el botón reiniciar va a cargar dos escenas completamente distintas .

4.9. Multijugador y networking

En esta sección se centrará en explicar como funciona la parte de red de Unity, conceptos y comandos claves para su realización.

La parte de red de Unity tiene un API de Scripting de alto nivel (HLAPI), significa que se tiene acceso a los comando que cubren la mayoría de requerimientos comunes para juegos multijugador sin necesidad de tener que preocuparse por lo detalles de implementación de bajo nivel. La API de Unity se encuentra obsoleto (deprecated) y en un futuro se eliminará. La API de alto nivel permite:

- Controlar el estado de red de nuestro juego utilizando el Network Manager, es decir, el Network Manager es nuestro administrador de red.
- Nos permite jugar en el host, ya que el host es también un cliente.
- Serializar datos.
- Enviar y recibir mensajes de red.
- Enviar comandos desde el cliente a servidor.
- Realiza llamadas de procedimientos remotos (call RPCs) de servidor a clientes.
- Enviar eventos de red de servidores a clientes.

El Networking de Unity es integrado al motor y al editor, permitiendo trabajar con componentes ya ayudas visuales para construir nuestro juego multijugador, nos proporciona:

- Una componente para objetos en red llamado “NetworkIdentity”.
- NetworkBehaviour para scripts en red.
- Sincronización automática configurable de los transforms de los objetos.
- Sincronización automática de variables en el script.
- Soporte para colocar objetos en red en escenas de Unity.
- Componentes de red (Network).

4.9.1. Funciones

Se comentará, las funciones utilizadas en este proyecto, así como la función que desempeña cada uno de ellos.

- ***Network Manager***

El administrado de red es el responsable de gestionar los aspectos de red del juego multijugador. Solo de deberá tener un Network Manager activo en su escena a la vez.

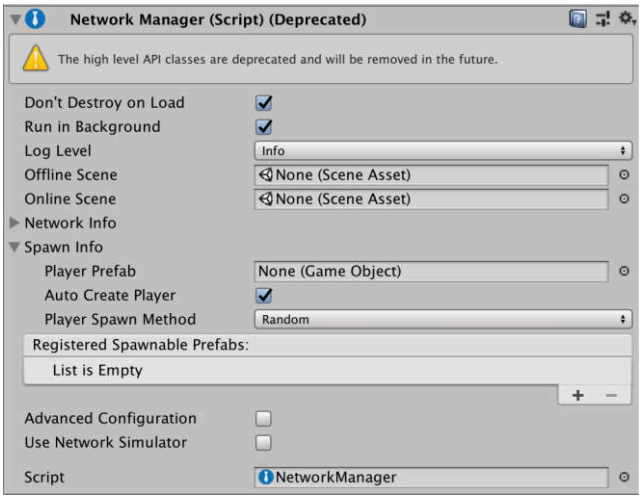


Figura 4.159: Network Manager

- ***Network Manager HUD***

Unity facilita la creación de juegos multijugador con una interfaz de usuario sencilla de usar, tal y como se muestra en la imagen siguiente:

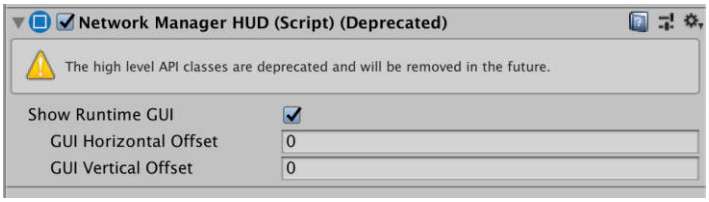


Figura 4.160: Interfaz de Usuario del Network Manager

En este proyecto, no se ha utilizado esta función debido a que se ha creado una interfaz de usuario propia más personalizable, además de que en la documentación de Unity recomendaba el uso de crear una personalizable y adaptarla a tu aplicación.

- ***Network Identity***

Esta componente es la identidad de red y es la encargada de decir si es un “Local Player Authority” o “Server only”.

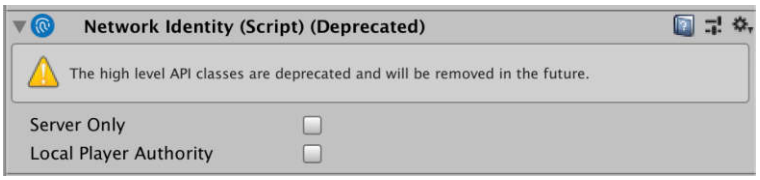


Figura 4.161: Network Identity

- ***Network Transform***

Esta componente, se encarga de la sincronización del objeto en red, como la posición, rotación, escala. Además podemos configurar algunos valores como el tiempo que tarda en enviar los datos, así como el tipo de sincronización, es decir, si se quiere una sincronización rigydbody, si se quire sincronizar solo las componentes XY o todas las componentes.

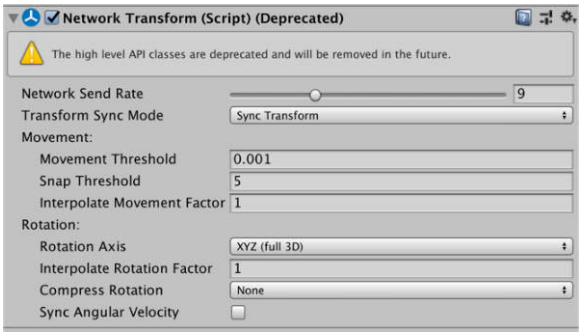


Figura 4.162: Network Transform

- ***Network Start Position***

El Network Start Position, se utiliza para generar el objeto de red en una posición especifica donde quiere que se genere, se puede agregar tantas posiciones a la escena como se desee, debido a que el administrador de red detecta todas esas posiciones de inicio en la escena y cuando se genera el objeto usa la posición y la rotación de una de ellas.

El administrador de red, tiene dos diferentes métodos para la generación de jugadores:

- Random: elige de forma aleatoria donde generar el objeto de red.
- Round Robin: recorrer las opciones de Start Position en una lista de conjuntos.

4.9.2. Comandos

- ***SyncVar***

La sincronización de variables son variables que heredan de NetworkBehaviour, que se sincronizan desde el servidor a los clientes. Cuando un nuevo jugador se une al juego se le envía el último estado de todas las “SyncVar” en los objetos en red que son visibles para ellos. Para especificar que variables se quiere sincronizar hay que añadir el atributo [SyncVar]. Este comando permite sincronizar variables de tipo enteros, float, string entre otros.

El estado de “SyncVar” se aplica a “GameObjects” en los clientes antes de llamar a “OnStartClient()”, por lo que el estado del objeto siempre esta actualizado dentro de “OnStartClient()”.

- ***Command***

Los “Commands” son enviados de objetos jugadores en el cliente a objetos jugadores en el servidor. Por seguridad, “Commands” solamente pueden enviarse desde su objeto jugador, por lo que no se puede controlar los objetos de otros jugadores. Para usar esta función, se necesita que se agregue el atributo personalizado [Commands] y añadir a la función que se le aplica el prefijo “Cmd”. Esta función solo se ve en el servidor cuando un cliente la llama.

- ***ClientRpc***

La función” ClienteRpc” son enviadas desde objetos jugadores en el servidor a objetos jugadores en clientes, pueden ser enviados de cualquier objeto servidor con un “NetworkIdentity”, ya que el servidor tiene autoridad. Por tanto, no hay problemas de seguridad con los objetos del servidor ya que son capaces de enviar estas llamadas. Para usar esta función, se necesita que se agregue el atributo personalizado [ClienteRpc] y añadir a la función que se le aplica el prefijo “Rpc”. Esta función se va a ejecutar en los clientes cuando sea llamada en el servidor.

4.9.3. Creación de la red UNET

En esta sección se va a comentar como se creará las funciones ya comentadas en el primer apartado de este capitulo para poder usar ambos juegos como multijugador.

Lo primero para empezar a crear la parte de red, es en la escena de “RestauranteMesaOcaMulti”, crear un objeto vacío al que se llamará “NetworkOca” en caso del juego de la oca y “NetworkSerpiente” para el caso del juego de la serpiente y escaleras y se le asociará nuestro script “managerOca” y “managerSepiente” respectivamente. Es un

script editable al NetworkManager que viene por defecto, en este script se pondrá las funciones que hará todos los botones de esta escena, como por ejemplo si seleccione el restaurante “Mesón Sebastián”, tendrá ya un dirección IP asignada y una vez que se seleccione el restaurante, saldrá elegir la mesa en la que se encuentre el cliente, en cada mesa tendrá por defecto un puerto asignado, se hace con la intención de que cada mesa tenga una partida diferente. Además en función del puerto que sea nos indicará en que tablero se encuentra. Este mismo script estará compartido en la escena “OcaMulti”, debido a que si es el servidor tendrá que escribir la dirección IP y el puerto para que los clientes puedan iniciar la partida, y por lo que habrá que asociarle estos botones con las funciones creadas en este script.

A continuación, se muestra ambas funciones de la selección del restaurante, a lo que se va a asociar a cada botón:

```
public void Restaurante1()
{
    NetworkManager.singleton.networkAddress = "192.168.1.87";

}
public void Restaurante2()
{
    NetworkManager.singleton.networkAddress = "192.168.1.76";

}
```

Figura 4.163: Función IP de cada restaurante

En el primer restaurante, para asociarle la primera función, se debe de arrastrar el objeto “NetworkOca” y buscar dicha función, a su vez cuando se pulse sobre el restaurante, se desactivará el “CanvasRestaurante”, y se activara el “CanvasMesas”, para la elección de la mesa en la que se encuentre.

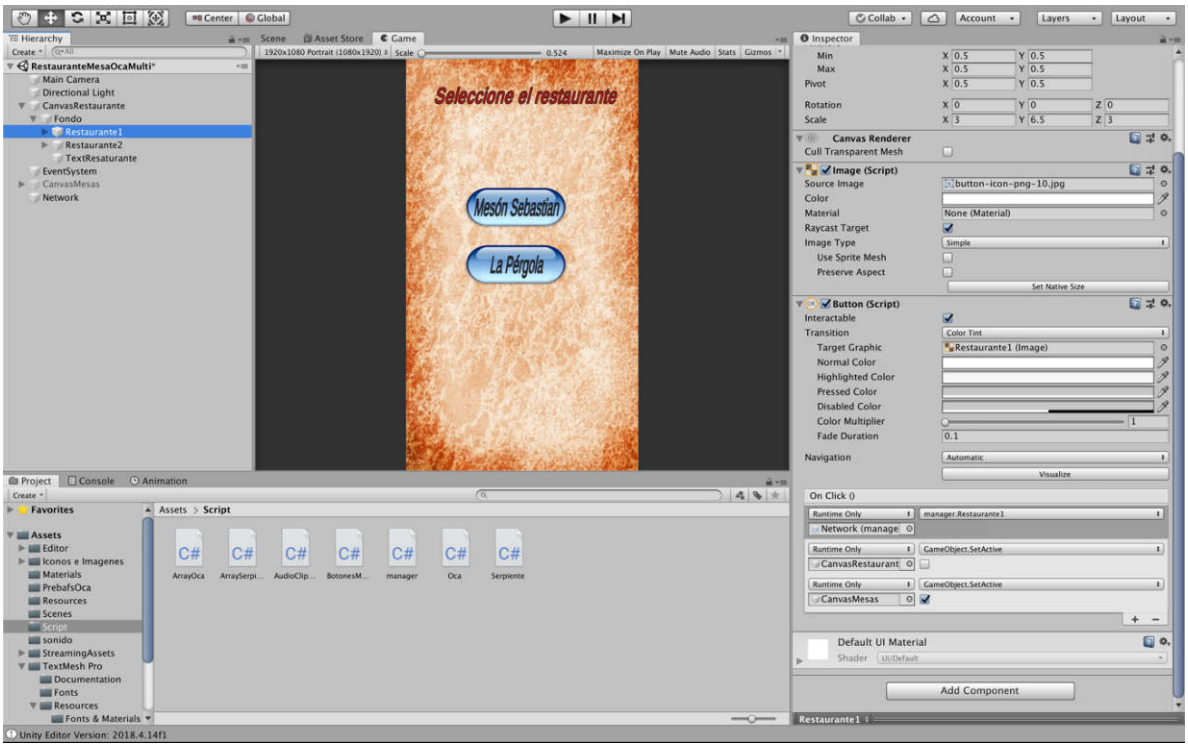


Figura 4.164: Funciones que componen Restaurante1

A continuación, se muestra las función de cada mesa, en ellas son similares las funciones lo único que varía es el puerto, en cada función tiene su puerto asociado, se carga la escena “OcaMulti”, e se inicia el cliente, con la función que nos proporciona Unity “StartClient()”:

```
public void PuertosMesa1()
{
    int puertoMesa1 = 1111;

    NetworkManager.singleton.networkPort = puertoMesa1;

    SceneManager.LoadScene("OcaMulti");

    NetworkManager.singleton.StartClient();

}
public void PuertosMesa2()
{
    int puertoMesa2 = 2222;

    NetworkManager.singleton.networkPort = puertoMesa2;

    SceneManager.LoadScene("OcaMulti");

    NetworkManager.singleton.StartClient();
}

public void PuertosMesa3()
{
    int puertoMesa3 = 3333;

    NetworkManager.singleton.networkPort = puertoMesa3;

    SceneManager.LoadScene("OcaMulti");

    NetworkManager.singleton.StartClient();
}
public void PuertosMesa4()
{
    int puertoMesa4 = 4444;

    NetworkManager.singleton.networkPort = puertoMesa4;

    SceneManager.LoadScene("OcaMulti");

    NetworkManager.singleton.StartClient();
}
```

```
public void PuertosMesa5()
{
    int puertoMesa5 = 5555;

    NetworkManager.singleton.networkPort = puertoMesa5;

    SceneManager.LoadScene("OcaMulti");

    NetworkManager.singleton.StartClient();
}
public void PuertosMesa6()
{
    int puertoMesa6 = 6666;

    NetworkManager.singleton.networkPort = puertoMesa6;

    SceneManager.LoadScene("OcaMulti");

    NetworkManager.singleton.StartClient();

}

public void PuertosMesa7()
{
    int puertoMesa7 = 7777;

    NetworkManager.singleton.networkPort = puertoMesa7;

    SceneManager.LoadScene("OcaMulti");

    NetworkManager.singleton.StartClient();

}
```

Figura 4.165: Funciones del puerto de cada mesa

Estas funciones, se le asociara a cada botón, al igual que antes, se arrastra el objeto “NetworkOca” a la sección “OnClick()”, de cada botón y se selecciona la mesa:

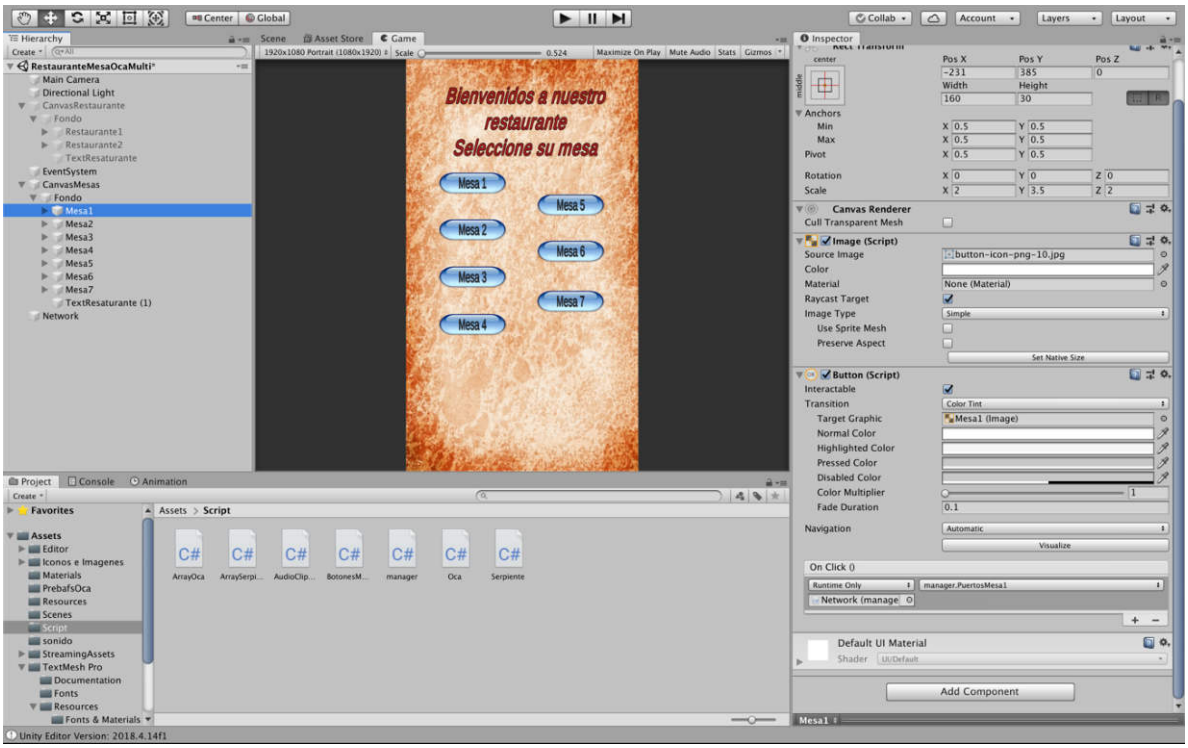


Figura 4.166: Vista de la selección de mesas

Para el juego de la serpiente y escalera, se va a realizar de igual manera lo único que cambiará será cargar la escena de “OcaMulti” a la escena “SerpienteMulti”, para ello se tendrá que replicar esta misma escena.

Tanto en ambas escenas comentadas anteriormente, se creará un “Canvas” para que el servidor indique el puerto y la dirección IP (los objetos compuestos por este Canvas tendrá una posiciones diferentes para el servidor y el cliente). Además este “Canvas” para los clientes se desactivará, ya que los clientes lo han indicado en la selección del restaurante y de la mesa en la que se encuentra.

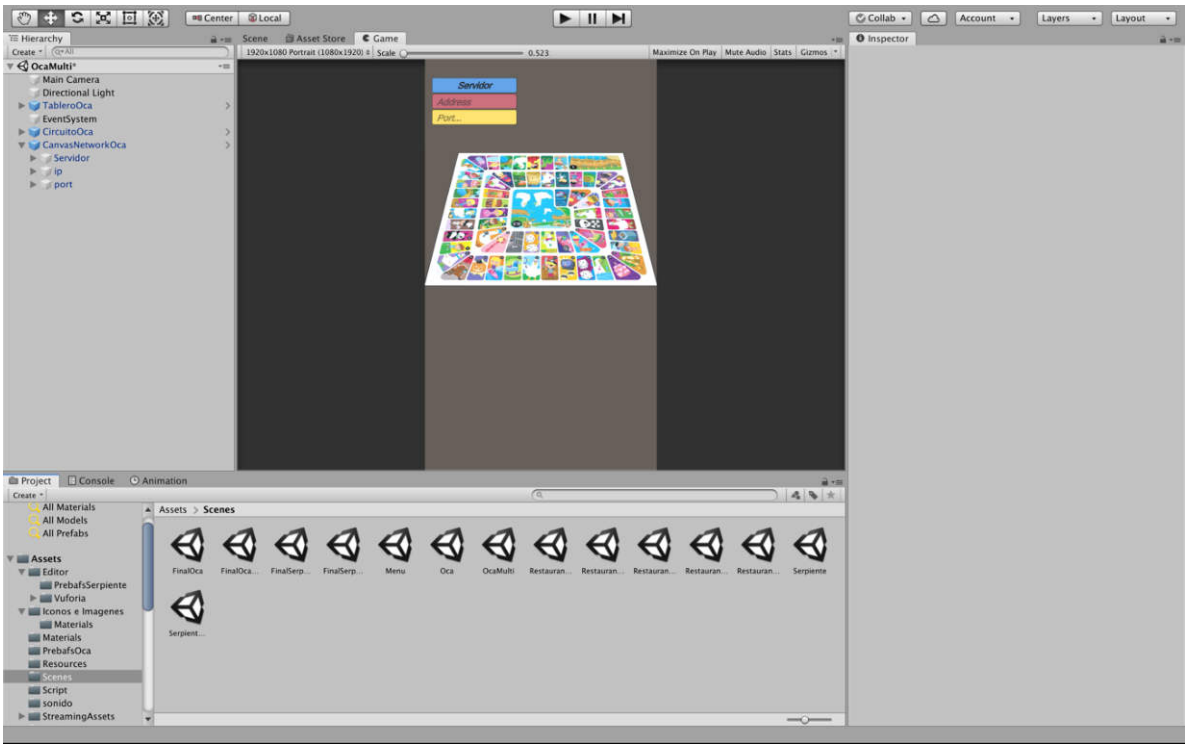


Figura 4.167: Vista del CanvasNetworkOca

Como en la escena anterior, se creará un objeto vacío llamado “NetworkOca”, y se le asociará el script “managerOca”, en este script se añadirá las funciones para los tres botones, y por consiguiente se le asociará a los botones. Para poder asociarle los botones, se necesita crearlo en el script y que sean de tipo “Public”, para que nos aparezca en el visor “Inspector”.

A continuación, se muestra las funciones de la dirección IP y del puerto. Por consiguiente, en la función “Server()”, se le añadirá estas dos función, además de comprobar si el puerto es por ejemplo “1111”, le diremos que es el “Tablero 1”, para indicar cual es el tablero en función del puerto, dentro del objeto “TableroOca”, se creará un “Canvas” llamado “CanvasTablero” en el cual estará compuesto por un objeto de tipo “TextMeshPro” llamado “tablero” (este objeto tendrá una posición diferente para el servidor y el cliente). Por otro lado en esta función se bloqueará este botón una vez que se pulse sobre él.

```
public void Puertos()
{
    int puerto;
    int.TryParse(Port.text, out puerto);
    NetworkManager.singleton.networkPort = puerto;

}
public void ip()
{
    NetworkManager.singleton.networkAddress = Ip.text;

}

public void Server()
{

    main.gameObject.SetActive(true);
    ar.gameObject.SetActive(false);
    tarjetas.gameObject.SetActive(false);

    ip();

    Puertos();

    NetworkManager.singleton.StartHost();

    if (NetworkManager.singleton.networkPort == 1111)
    {
        tablero.text = "Tablero 1";
    }

    else if (NetworkManager.singleton.networkPort == 2222)
    {
        tablero.text = "Tablero 2";
    }

    else if (NetworkManager.singleton.networkPort == 3333)
    {
        tablero.text = "Tablero 3";
    }

    else if (NetworkManager.singleton.networkPort == 4444)
    {
        tablero.text = "Tablero 4";
    }

    else if (NetworkManager.singleton.networkPort == 5555)
    {
        tablero.text = "Tablero 5";
    }

    else if (NetworkManager.singleton.networkPort == 6666)
    {
        tablero.text = "Tablero 6";
    }

    else if (NetworkManager.singleton.networkPort == 7777)
    {
        tablero.text = "Tablero 7";
    }

    Servidor.interactable = false;
}
```

Figura 4.168: Funciones del Servidor

Se muestra la imagen de los botones del “CanvasNetworkOca” asociados a la red (objeto NetworkOca):

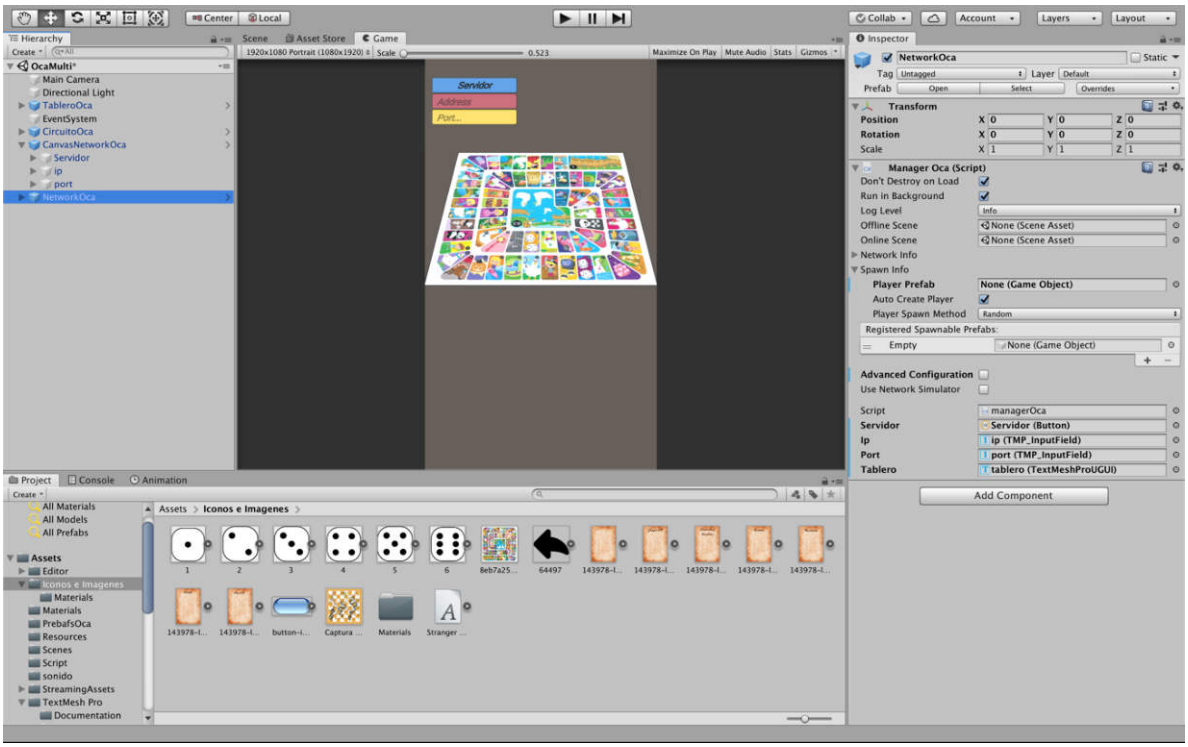


Figura 4.169: Asociar los objetos al Network Manager

Otro configuración que se hará dentro de esos script, es que cuando el numero máximo de jugadores existentes en una partida sea mayor o igual a 5, el servidor lo desconectara y el cliente desconectado se irá a la escena de “RestauranteMesaOcaMulti.cs” o “RestauranteMesaSerpienteMulti.cs” en función de que juegos este accediendo. De manera que le máximo de jugadores en una partida sea solo de 4.

A continuación, se muestra esas dos funciones que dispone unity tanto cuando un jugador accede a ese servidor como cuando un jugador se desconecta.

```
public override void OnServerConnect(NetworkConnection conn)
{
    base.OnServerConnect(conn);

    if (numPlayers >= 5)
    {
        Debug.Log("hay muchos");

        conn.Disconnect();

        return;

    }
}
```

```
public override void OnClientDisconnect(NetworkConnection conn)
{
    base.OnClientDisconnect(conn);

    Destroy(NetworkManager.singleton.gameObject);

    NetworkManager.Shutdown();

    SceneManager.LoadScene("RestauranteMesaOcaMulti");

}
```

Figura 4.170:Funciones del Servidor y Cliente

4.9.4. Configuración Network Manager

En este apartado, es una continuación del anterior apartado donde se comentará los cambios del Network Manager, así como el objeto que estará en red y algunas modificaciones de este objeto.

En el objeto “NetworkManager” se marcará las casillas “Don’t Destroy on Load” (esta propiedad se utiliza para controlar si Unity debe o no destruir el NetworkManager cuando se cambia de escena), “Run in Background”(esta opcion permite que el programa se ejecute cuando se encuentra en segundo plano), “Script CRC Check” (esta opción verifica CRC entre el servidor y el cliente que asegura que los script NetworkBehaviour coincidan), “Packet Fragmentation”(esta opción permite fragmentar los paquetes que son más grandes que maxPacketSize, hasta un máximo de 64K), y “Auto Create Player”(esta opción es la más importante porque sino se marca cuando se conecta o cambia de escena no va a aparecer).

Por otro lado, se cambiará la opción “MaxDelay”, por defecto aparece con 10ms el máximo retraso que puede haber, en cambio se pondrá más bajo ese valor para intentar conseguir el menor retraso posible en el juego.

A continuación, se mostrará el objeto “NetworkManager” con las opciones comentadas anteriormente activadas.

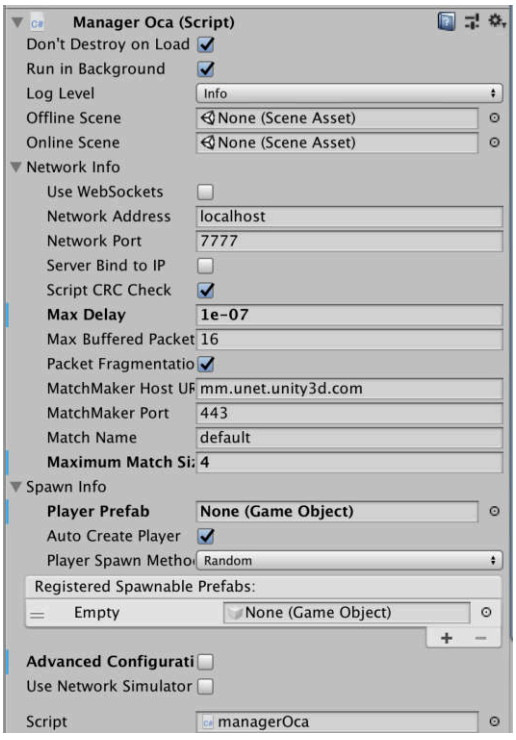


Figura 4.171: Configuración de los parámetros del Network Manager

Para añadir el “Player Prefab” se necesita que el objeto tenga un componente “NetworkIdentity”, y se lo configurará como “Local Player Authority”, además del componente “NetworkTransforms” que hará que en todas las instancias se pueda ver el cambio de posición, rotación y escala, además se le indicará que el tipo de sincronización será “Sync Transform”, y la velocidad de enviar las actualizaciones por segundo sea de 25, se le pone un número alto por que el objeto enviado a la red contiene muchos elementos que tiene que actualizarse como el turno, los textos, el color, entre otros.

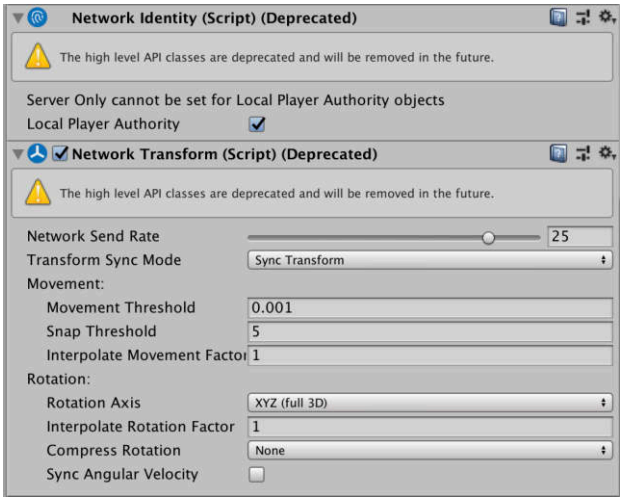


Figura 4.172: Componentes necesarios del PlayerPrebafs

El objeto que se envía a la red, será una copia del objeto “FichaOcaJugador” porque como se ha comentado antes se tiene que incluidos estos elementos de red, a este objeto se le asociará el script “OcaMulti.cs”, para usar este script en la red se necesita usar la librería “using UnityEngine.Networking”, y declarar el script como “NetworkBehaviour” en lugar de “MonoBehaviour”, además por otro lado, de desactivar el “CanvasBotones”, y por código decirle que si es “LocalPlayer” este “Canvas” se active, esto se hace debido a que si no se desactiva se van a crear varios “Canvas” tantos como clientes haya en la red, y esto hará que se manejes los botones de otros clientes que no sea el suyo propio, de esta manera solo el jugador local tendrá su propio panel de botones. Esta condición se indicará en la función “Start()”:

```
if (isLocalPlayer)
{
    canvasPanel.GetComponentInChildren<Canvas>().enabled = true;
}
```

Figura 4.173: Activación del CanvasPanel si es objeto local

Por otro lado, cuando un cliente mueva solo su objeto y no todos los objetos que haya en la red uno por cada cliente, hay que indicarle que si no es “LocalPlayer” que no

realice nada, esta condición se realizará en la función “Update()”, tal y como se muestra a continuación:

```
if (!isLocalPlayer)
{
    return;
}
```

Figura 4.174: Condición para no mover objetos de otros jugadores

Por último, se tiene que poner la posición del objeto ya que cuando se instancia se coloca en la posición (0,0,0), como se comento en el apartado funciones se va a utilizar la función “StartPosition”, para ello dentro del objeto de administrador de red se creará un objeto vacío, al que se añadirá esta función y se pondrá la posición (2.25f,0.18f,4.11f). El administrador cuando se inicie buscará este objeto y lo colocará en la posición comentada.

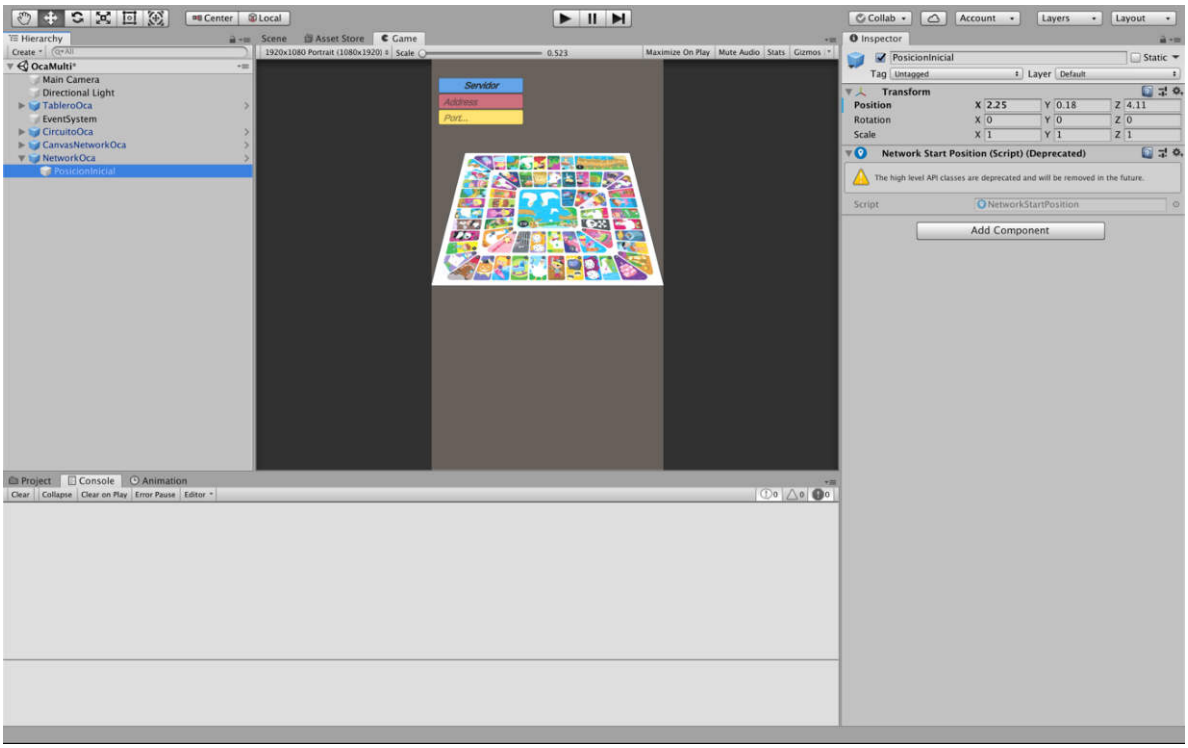


Figura 4.175: Añadir Network Start Position

4.9.5. Sincronización del texto

En este apartado, se verá como se va a sincronizar los texto en todos los clientes, a excepción del servidor que no lo verá ya que el solo va a estar observando además de estar recibiendo y devolviendo a todos los clientes los comandos. Debido a que es realizar los mismo en ambos juegos, se explicaran en conjunto. La estructura en la ventana de jerarquía será de la misma manera tal y como se hizo en el apartado capitulo 4.7 y 4.8, pero con la diferencia que al no ser lienzo del objeto de red, se buscará por código estos

objetos de esto, y se activarán o desactivarán casa uno de ellos dependiendo de la función que se utilice en cada caso.

Se ha realizado de esta manera, porque si es lienzo del objeto de red y este objeto “Canvas” este activo se replicarán tantos “Canvas” como cliente haya en la red, y si se desactiva este objeto “Canvas”, por mucho que se utilice los comandos para sincronizar como se hizo con el “CanvasBotones”, solo lo podrá ver el suyo y no el resto, por este motivo se ha realizado de esta manera.

Para la sincronización de los textos, se usa los comando “ClientRpc” y “Commands”, explicados en el anterior apartado. Primeros se usará el comando “ClientRpc” y se incluirá dentro todo lo que se desee que se sincronice, por consiguiente se usará el comando “Commands” a la cual se hará una llamada a la anterior función, esto se realiza así debido a que “Commands” se ejecuta solo en el servidor y no en los clientes por lo que el “ClientRpc” le devolverá la llamada a todos los clientes.

Con respecto, a los cambios que hay que realizar será cambiar las funciones de los anteriores capítulos por las funciones “Commands”.

En el anexo 7 se muestra los estados de sincronización tanto del juego de la oca como de la serpiente y escalera.

4.9.6. Sincronización del color

En este apartado, se va a comentar la sincronización del color de las fichas de cada cliente, para que cada usuario que se conecte tenga un color distinto, y pueda tener el color sincronizado de otros usuarios.

Para ello, se utilizará el modificador “static” a nuestra lista, esto lo que hará es crear una única copia de la lista en todo la red, de lo contario si no se utiliza el modificador se creará en cada instancia su propia lista. A continuación se muestra la lista con el modificador “static”:

```
public static List<Color> listaColores = new List<Color> {Color.red, Color.green, Color.grey, Color.blue, Color.cyan, Color.magenta, Color.yellow, Color.black, new Color(1f, 0.44f, 0.16f, 1f), new Color(0.3f, 0.4f, 0.6f, 0.3f),

new Color(0.55f, 0f, 1f, 1f), new Color(0.63f, 0.33f, 0.11f, 1f), new Color(0.60f, 0.07f, 0.07f, 1f) };
```

Figura 4.176: Declaración de lista colores

Por otro lado se declarará una variable de tipo “Color”, llamada “randomColor” a la que se va a utilizar el [SyncVar], para que actualice los colores desde el servidor a todos los clientes, además de igualar esta variable a la función “crearOb()” en la función “OnStartClient()”, tal y como se muestra en la imagen:

```
[SyncVar]
public Color randomColor;
```

```
public override void OnStartClient()
{
    if (isServer)
    {
        randomColor = crearOb(); //igualamos la variable sincronizada a los objeto de forma que tenga asignado el color por igual

    }
}
```

Figura 4.177: Declaración de la variable Color y función OnStartClient

En la función crearOb(), es una función de tipo “Color” en la cual nos va a devolver un color aleatorio, una vez que haya elegido el color se lo va a asignar a la ficha y lo va a eliminar de la lista, a través de la propiedad “Remove”, por otro lado, cuando la lista sea igual a 0, se le vuelve añadir los colores a esa lista y realiza el mismo proceso.

```
public Color crearOb()
{
    if (listaColores.Count >= 0)
    {

        ficha.GetComponent<Renderer>().material.color = randomColor;//asigna un color al objeto de la lista seleccionado y se iguala a la variable sincronizada

        listaColores.Remove(randomColor);

        Debug.Log(listaColores.Count);

        if (listaColores.Count == 0)
        {

            Debug.Log("no hay mas colores");

            listaColores.Add(Color.red);
            listaColores.Add(Color.green);
            listaColores.Add(Color.gray);
            listaColores.Add(Color.blue);
            listaColores.Add(Color.cyan);
            listaColores.Add(Color.magenta);
            listaColores.Add(Color.yellow);
            listaColores.Add(Color.black);
            listaColores.Add(new Color(1f, 0.44f, 0.16f, 1f));
            listaColores.Add(new Color(0.3f, 0.4f, 0.6f, 0.3f));
            listaColores.Add(new Color(0.55f, 0f, 1f, 1f));
            listaColores.Add(new Color(0.63f, 0.33f, 0.11f, 1f));
            listaColores.Add(new Color(0.60f, 0.07f, 0.07f, 1f));

            ficha.GetComponent<Renderer>().material.color = randomColor;//asigna un color al objeto de la lista seleccionado y se iguala a la variable sincronizada

            listaColores.Remove(randomColor);


        }
    }

    return listaColores[UnityEngine.Random.Range(0, listaColores.Count)];//obtenemos de la lista un color aleatorio
}
```

Figura 4.178: Función crearOb()

4.9.7. Identificación de cada cliente

En este apartado, se verá como dependiendo del identificador de cada cliente tendrá una altura determinada, esto se hace con la intención de que cuando caiga en una posición más de dos clientes se pueda ver que hay una ficha y no que parezca que se lo ha comido.

Como se comentó que el máximo de clientes en una partida eran 4, se tendrá cuatros alturas diferentes y se irán replicando estas alturas cuando se termine el ciclo de estos cuatro clientes, es decir que el identificador 2, y 6 tendrá la misma altura, y así sucesivamente. A continuación se mostrará una imagen sobre esto:

```
Vector3 lista = ruta.listaPosicion[listaCambioPosiciones[CurrentPlayer]].position;

if (id == 2||id==6 || id == 10 || id == 14 || id == 18)
{
    lista.y = 0.18f;
}
if (id == 3 || id == 7 || id == 11 || id == 15 || id == 19)
{
    lista.y = 0.30f;
}
else if (id == 4 || id == 8|| id == 12 || id == 16 || id == 20)
{
    lista.y = 0.38f;
}
else if (id == 5 || id == 9 || id == 13 || id == 17)
{
    lista.y = 0.45f;
}

listaPlayer[CurrentPlayer].transform.position = Vector3.MoveTowards(listaPlayer[CurrentPlayer].transform.position, lista, 1.8f * Time.deltaTime);
```

Figura 4.179: Cambio de altura para cada cliente

Esta parte se replicará en todos los cambios de posiciones como en las casillas especiales, ya que el Vector3 se le da una posición en concreto así como cambiarle el nombre al Vector3.

4.9.8. Sincronización del turno de los usuarios

En este apartado del capitulo, se verá como se creará el turno en la red, en el que cada cliente deberá respectar su turno, si un cliente que no le corresponde tirar pulsa sobre el botón, saltará un mensaje diciendo que no es su turno.

La metodología para el turno, es un cliente pulsa sobre el botón, cuando acaba el movimiento de los dados este botón desaparecerá y se activará el botón de finalizar el turno, dando una única posibilidad para que el cliente pulse sobre el botón, y pase el turno al siguiente jugador (sumando un punto en el marcador que estará sincronizado en la red y que será el “id” correspondiente a cada cliente). Una vez que se pulse sobre dicho botón, volverá a salir el botón de tirar, ya por mucho el intente pulsar su ficha no se va a mover, dado que ya ha pasado el turno.

En este caso, se va a crear un “Canvas” llamado “CanvasTurnoOca” para el juego de la oca y otro para el juego de la serpiente, este “Canvas” se añadirá como un “prebafs” y lo que se hará es buscarlo desde el código en lugar de que este como un objeto local para cada cliente.

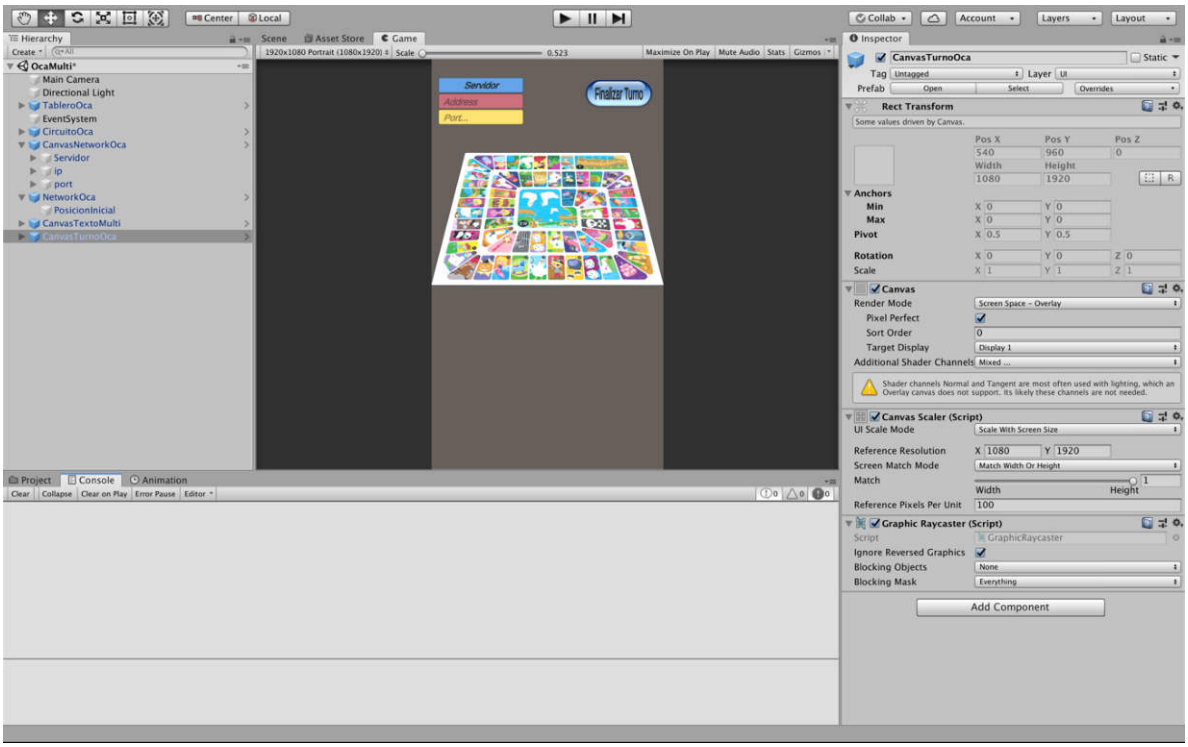


Figura 4.180: Vista del CanvasTurnoOca

Lo primero es crear tres variables de tipo estática, para que solo exista una única copia en la red, además de otra variable que no será de tipo estática para saber cual es el "id" de cada cliente, ya que cada copia de la instancia tendrá un "id" diferente.

Esas tres variables será el turno (que corresponda al número del cliente que le toca tirar), el turno estará inicializado a 2 debido a que el servidor es el primer cliente y le corresponde el id=1 y el servidor no va a jugar solo va a ver la partida, las otras dos variables será el turnmin y turnmax.

El turnmin corresponde al valor mínimo del cliente, es decir si tenemos tres clientes conectados el valor mínimo será 2, además el turnmin también se iniciará a 2.

El turnmax corresponde al valor máximo del cliente, es decir si tenemos tres cliente conectado además del servidor, el valor máximo será 4.

```
public int id;
public static int myPoint=2;
public static int turnmax;
public static int turnmin=2;
```

Figura 4.181: Declaración de las variables para el turno

Una vez que se tiene definido estas variables, en la función Start(), se obtendrá el id a través del “netId”, además de calcular el turno máximo y el mínimo a través de las función “max” y “min” respectivamente.

```
id = int.Parse(netId.ToString());

turnmax = Mathf.Max(myPoint, id);

turnmin = Mathf.Min(turnmin, turnmax);
```

Figura 4.182: Obtención del identificador, máximo y mínimo

Por otro lado, en esta misma función se buscará el botón de finalizar el turno, y se pondrá que este desactivado al iniciar el juego si es cliente local, para que cada cliente cuando inicie el juego si le da por pulsar el botón, no hay trampas y se lo pase a otro jugador, además de asignarle la función que hará “OnClick” al pulsar sobre dicho botón.

```
if (isLocalPlayer)
{
    GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().interactable = false;
}

GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().onClick.AddListener(() => PulsaTurno(myPoint));
```

Figura 4.183: Encontrar objetos en la escena

A continuación, se pasará a explicar la función que hará el botón “Finalizar Turno”. La función “PulsaTurno(myPoint)”, estará constituido por las funciones “ClientRpc” y “Commands”, debido a que se quiere que el texto este sincronizado en el momento que sume el punto, así como si el punto es mayor que el numero de clientes, pase al turno mínimo para poder realizar otra ronda de tiradas, tal y como se muestra a continuación:

```
[ClientRpc]
public void RpcSumaTurno(int mypoint)
{

    myPoint = mypoint;

    myPoint += 1;

    if (myPoint >= turnmax + 1)
    {
        myPoint = turnmin;
        RpcActualiza(myPoint);

    }
    else
    {
        GameObject.Find("TurnoText").GetComponentInChildren<TextMeshProUGUI>().text = "" + myPoint;
    }

}

[Command]
public void CmdSumaTurno(int mypoint)
{
    RpcSumaTurno(mypoint);
}


public void PulsaTurno(int myPoint)
{

    CmdSumaTurno(myPoint);
}
```

Figura 4.184: Sincronización del punto

Se realizará otra función llamada “BloqueoTurno”, que realice que cuando pulse sobre el botón finalizar el turno, este botón se desactive y que aparezca el botón de tirar los dados.

```
public void BloqueoTurno()
{

    GameObject.Find("FinalizaTurno").GetComponent<Button>().interactable = false;

    Tirar.gameObject.SetActive(true);
}
```

Figura 4.185: Función BloqueoTurno()

De forma paralela a estas funciones, se debe pensar que si vuelve a pulsar el botón tirar se moverá la ficha, para ello en la función “TirarDados()”, se le indicará mediante una condición, que si el número del turno no coincide con la “id” de los clientes que no realice nada. Además de indicarle por texto que no es su turno, para esto último se le añade al “prebabs” en la red otro componente de tipo “TextMeshPro” (este no se sincronizará con todos los jugadores sino que será de forma local), tal y como se ve en la siguiente imagen.

```
if (myPoint != id)
{
    textTurno.text = "No es tu turno";
    StartCoroutine(TerminaTextoTurno());

    return;
}
```

```
IEnumerator TerminaTextoTurno()
{
    yield return new WaitForSeconds(0.5f);
    textTurno.gameObject.SetActive(false);
}
```

Figura 4.186: Añadir condiciones en la función TirarDados()

Si se cae en alguna casilla especial, como por ejemplo en el oca cuando acabe el movimiento indicarle que el botón de finalizar el turno siga bloqueado y el tirar los dados activo para que pueda volver a tirar, algo similar en las casillas de perder turno que se hará lo contrario, además de añadirle en este caso la función de “BloqueoTurno”. A continuación se mostrará ambos casos, debido a que es realizar lo mismo en varias parte del código.

```
public void PasarTurno()
{
    Tirar.gameObject.SetActive(false);

    GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().interactable = true;

    GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().onClick.AddListener(() => BloqueoTurno());
}


public void SeguirTurno()
{
    Tirar.gameObject.SetActive(true);

    GameObject.Find("FinalizaTurnoOca").GetComponent<Button>().interactable = false;
}
```

Figura 4.187: Función PasarTurno() y SeguirTurno()

En la lógica hacia atrás del juego de la oca, en el caso de que no caiga en una oca, se tiene que pasar el turno, para ello en lugar de salir el botón de tirar saldrá el botón de finalizar turno. Por otro lado, en la comprobación cuando acabe el movimiento de una oca,

puente, o la ficha vaya hacia atrás solo se utilizará la componente x y z para la comprobación, ya que la altura de la ficha ha cambiado como se vio en el apartado 4.9.7, el resto del código es el mismo que se vio en el apartado 4.7.

4.9.9. Sincronización de la escena final

En este apartado, se comentará lo que ocurrirá cuando un jugador gane la partida. En este caso cuando un jugador gana la partida se cambia de escena para elegir si el quiere reiniciar o directamente salir de la aplicación tal y como se vio en los capítulos anteriores de ambos juegos, lo que se hará en el caso de multijugador es que cuando algún jugador gane le salte a todos los jugadores excepto al servidor el cambio de escena así como destruir el objeto de red para que no se instancie múltiples replicas de red y no interrumpa la conexión, si esto no se hace no saltará un error en la consola diciendo que ya existe una comunicación.

Para realizar este añadido nuevo, se realizará a través de los comandos utilizados en la UNET de Unity (ClientRpc y Commands).

A continuación se mostrará la función que sincronizará el cambio de escena así como desactivar todos los objetos.

```
[ClientRpc]
public void RpcEspera()
{

    Boton.gameObject.SetActive(false);
    Boton1.gameObject.SetActive(false);
    Tirar.gameObject.SetActive(false);
    FondoPanel.gameObject.SetActive(false);
    GameObject.Find("CanvasTextoMulti").GetComponentInChildren<CanvasGroup>().alpha = 0;
    GameObject.Find("CanvasTurnoOca").GetComponentInChildren<CanvasGroup>().alpha = 0;

    if (!isServer) //Hacemos que todos los clientes excepto el servidor cambie de escena
    {

        Destroy(NetworkManager.singleton.gameObject); //destruyer el objeto Network

        NetworkManager.Shutdown();

        SceneManager.LoadScene("FinalOcaMulti"); //Esta escena lo que hace es elegir a cada jugador si quiere continuar jugando o salir del juego

    }


}

[Command]
public void CmdEspera()
{

    RpcEspera(); //Cuando el primer jugador ha ganado le manda a todos el cambio de escena a "final", para elegir que es lo que quiere hacer

}
```

Figura 4.188: Sincronización de la Función Espera()

4.9.9.1. Reiniciar el juego

En el botón de reiniciar el juego en multijugador lo que hará es cambiar de escena para que vuelva a entrar en el juego.

Además cuando se reinicia el juego, también se debe de actualizar el punto que indica a quien jugador le toca, así como el turno mínimo, ya que esta inicializado a dos

como se explico antes. Por esta razón quien pulse sobre este botón se actualizará ambas variables y los jugadores que deseen iniciar sin haber jugado antes iniciará con este nuevo marcador actualizado.

A continuación se muestra una imagen con el código implementado:

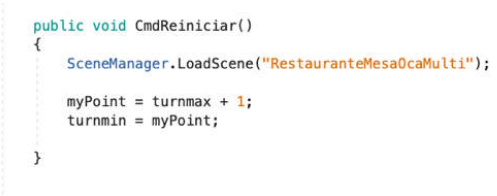


Figura 4.189: Función Reiniciar()

Esta función será de manera independiente a cada cliente, como se ve en la imagen el punto será el turno máximo más uno para que sea el siguiente jugador y no su anterior, además de igualar el turno mínimo al punto, esto se hace porque el “id” es de manera incrementada, es decir si soy el cliente número 4 cuando pulse dicho botón el número es el 5.

Por otro lado, ya que esta función es independiente a cada cliente, hay que incluir en la función “Espera”, que si es servidor también actualice el punto de quien le toca tirar y el turno mínimo, debido a que como este no cambia de escena siempre el turno mínimo será de dos debido a que se inicializo a dos esta variable.

Se muestra la actualización de la función comentada, para tener una mejor aclaración sobre esta función.

```
[ClientRpc]
public void RpcEspera()
{


    Boton.gameObject.SetActive(false);
    Boton1.gameObject.SetActive(false);
    Tirar.gameObject.SetActive(false);
    FondoPanel.gameObject.SetActive(false);
    GameObject.Find("CanvasTextoMulti").GetComponentInChildren<CanvasGroup>().alpha = 0;
    GameObject.Find("CanvasTurnoOca").GetComponentInChildren<CanvasGroup>().alpha = 0;


    if (!isServer) //Hacemos que todos los clientes excepto el servidor cambie de escena
    {


        Destroy(NetworkManager.singleton.gameObject); //destruyer el objeto Network

        NetworkManager.Shutdown();

        SceneManager.LoadScene("FinalOcaMulti"); //Esta escena lo que hace es elegir a cada jugador si quiere continuar jugando o salir del juego

    }

    else //solo para el servidor se actualiza el turnmin ya que como este no cambia de escena va a ser siempre 2 el turnmin por eso se realiza esto
    {

        myPoint = turnmax + 1;
        turnmin = myPoint;




    }
}


[Command]
public void CmdEspera()
{

    RpcEspera(); //Cuando el primer jugador ha ganado le manda a todos el cambio de escena a "final", para elegir que es lo que quiere hacer

}
```

Figura 4.190: Actualizar el punto al servidor en la Función Espera()

4.9.9.2. Salir del juego

En esta función, no hay que realizar ningún cambio, pero si en la función Start(), debido en el supuesto caso de que se ha realizado una partida con tres clientes, y deciden salir del juego. Si da la casualidad de que se quiere volver a jugar como el turno máximo anterior era de 4, al iniciar de nuevo, estos clientes se inicia pero el marcador o punto de ellos es de 2, entonces se debe indicarle que si el objeto de tipo texto es distinto a el marcador. Además de si el punto o marcador, y el turno mínimo es de 2 y el turno máximo es distinto de 2, se actualiza el marcador o punto al turno máximo, en este caso a 5 y el turno mínimo igual al punto o marcador.

```
public void Actualizar()
{
    if (GameObject.Find("TurnoText").GetComponentInChildren<TextMeshProUGUI>().text != "" + myPoint && turnmin == 2 && myPoint == 2 && turnmax != 2)
    {

        CmdActualizaNuevo(turnmax);
        myPoint = turnmax;
        turnmin = Mathf.Min(myPoint, turnmax);

    }

    else
    {

        RpcActualiza(myPoint);

    }
}


[ClientRpc]
public void RpcActualiza(int myPoint)
{
    GameObject.Find("TurnoText").GetComponentInChildren<TextMeshProUGUI>().text = "" + myPoint;

}

[ClientRpc]
public void RpcActualizaNuevo(int turnmax)
{

    GameObject.Find("TurnoText").GetComponentInChildren<TextMeshProUGUI>().text = "" + turnmax;

}

[Command]
public void CmdActualizaNuevo(int turnmax)
{
    RpcActualizaNuevo(turnmax);

}
```

Figura 4.191: Función Actualiza()

El primer cliente, en este supuesto caso que se ha expuesto, el primer cliente entra en la primera condición, el resto de clientes inician en la otra condición, ya que el “myPoint” ya no es dos sino cinco en el caso que se ha supuesto, esto es porque se ha actualizado el valor.

4.10. Implementación de la Realidad Aumentada

En esta sección se contará como insertar la “ARCamera” así como las “ImageTarget” que son las que se usarán como marcadores de la aplicación, y cambios en los ajustes para que se pueda usar esta tecnología.

4.10.1. Incorporación de la cámara de Realidad Aumentada

En primer lugar, se desactiva la “MainCamera”, y se añade la “ARCamera”, para ello se situará en la ventana jerarquía, se pulsa en el botón derecho →Vuforia Engine→ ARCamera.

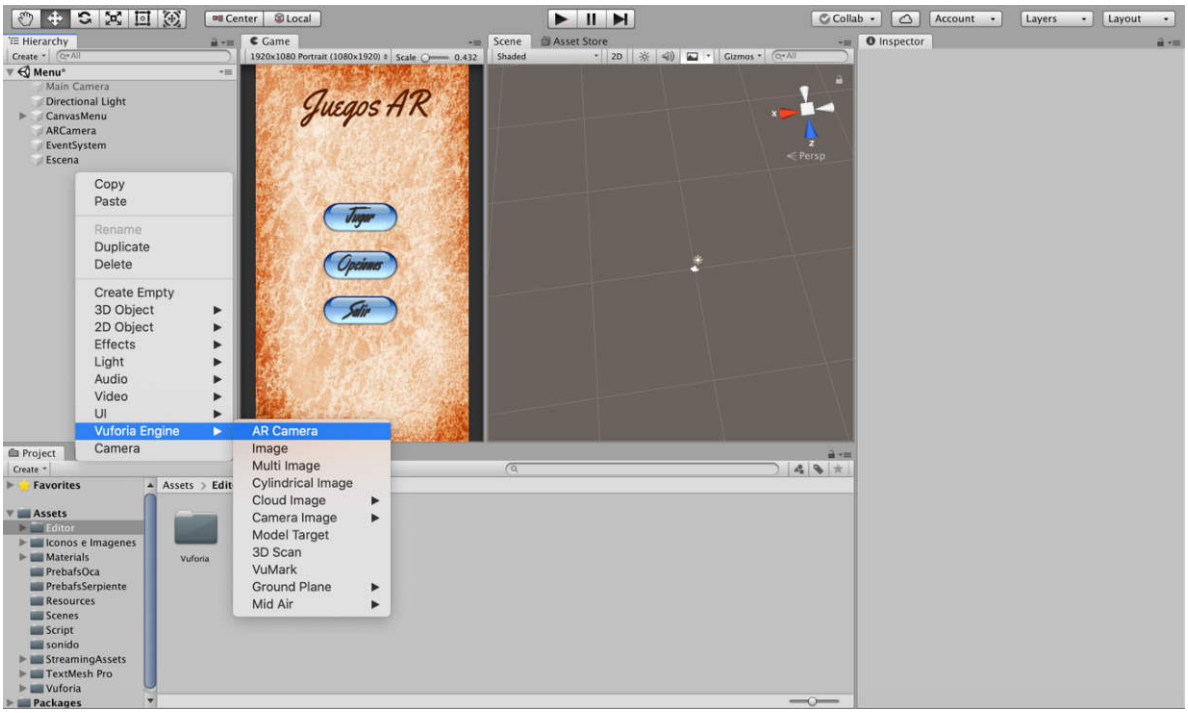


Figura 4.192: Añadir AR Camera en la escena

En la ventana inspector de “ARCamera”, se cambiará algunos parámetros. En la sección Camera en la opción “Clear Flags”, se utilizará “Depth Only”, por otra lado en la misma sección se activará “Allow HDR”, y en la sección Vuforia Behaviour (Script) se cambia de “DEVICE” a “FIRST_TARGET”. En la imagen siguiente se muestra los cambios:

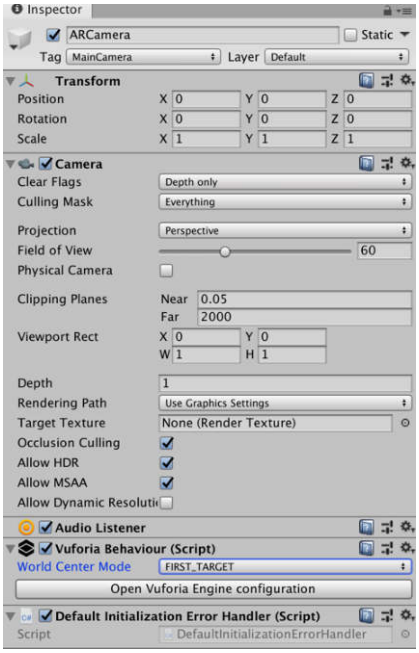


Figura 4.193: Componentes de la AR Camera

En la imagen anterior se pulsará sobre “Open Vuforia Engine configuration”, esto

es para poner la licencia de Vuforia.

A continuación en la imagen se pulsa en “Add Licence” o bien introducir en el navegador la siguiente url: https://developer.vuforia.com/ , y descargarnos la licencia, para la descargar nos pide que nos registremos.

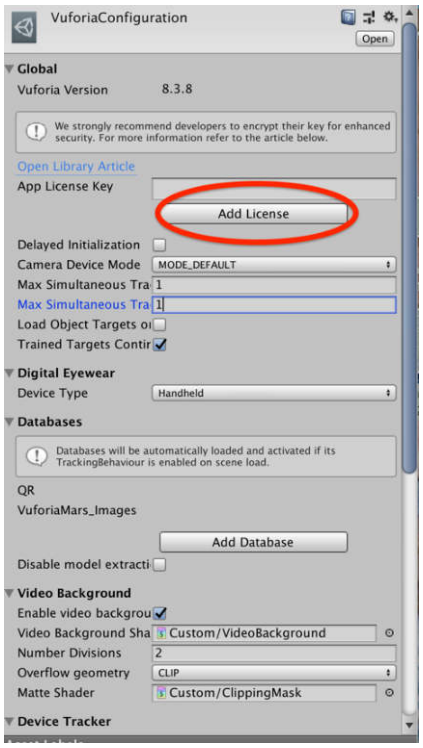


Figura 4.194: Vuforia Configuration

Una vez que se ha registrado y se acceda a nuestra cuenta, se selecciona “Get Development Key”, y saldrá lo siguiente:

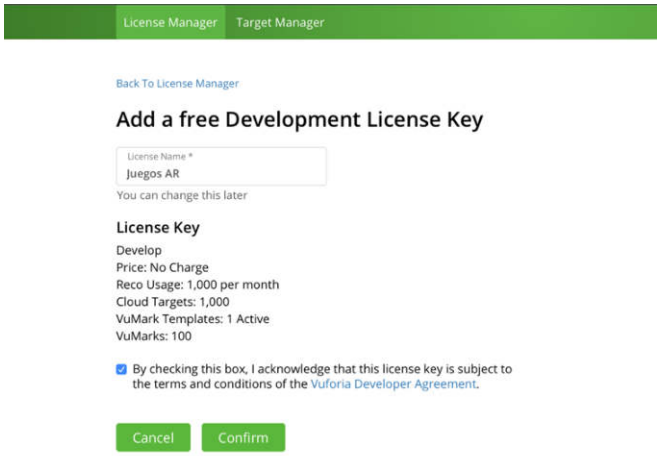


Figura 4.195: Crear licencia de Vuforia

Se pulsa en confirmar y se almacenará esta licencia en License Manager, como

se muestra ahora en la imagen siguiente:

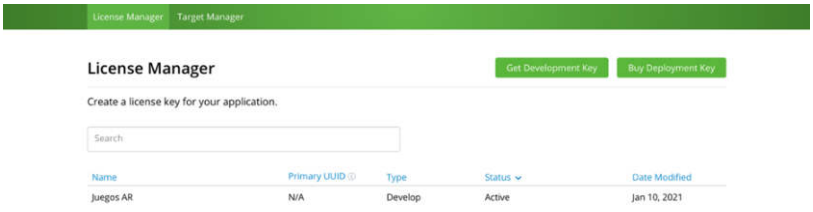


Figura 4.196: Lista de licencias de Vuforia

Se pulsa en “Juegos AR”, y aparecerá el código de la licencia:

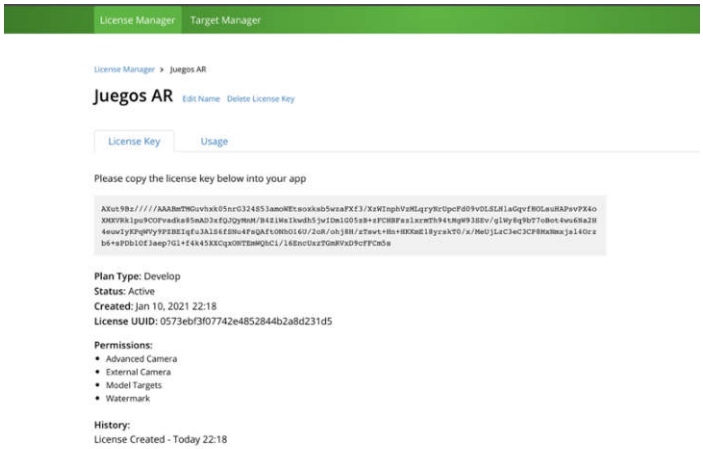


Figura 4.197: Licencia de Juegos AR

Se copiará el recuadro gris en Unity, en la sección “App License Key”:

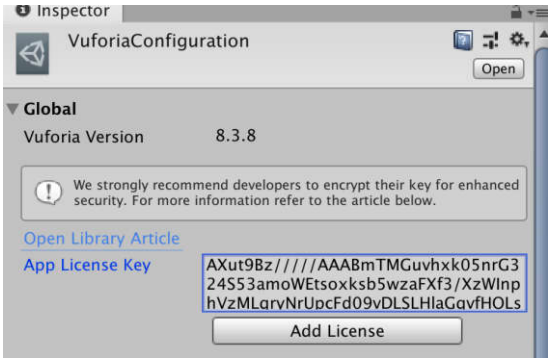


Figura 4.198: Insertar licencia en el proyecto

Por otra parte, en la pestaña de “Player Settings”, se activará la opción de Vuforia Augmented Reality. Para ello, se selecciona la pestaña File→Build Settings

Figura 4.199: Ventana de Player Settings

En la opción de “Player Setting” en la sección “XR Settings”, se activará la opción Vuforia Augmented Reality. Se tendrá que activar para todas las plataformas.

Figura 4.200: Activar Augmented Reality

Por último, hay que especificar que el servidor no utilizará la cámara “ARCamera” sino la “MainCamera”, para se realizará una modificación del script “managerOca” y “managerSerpiente”, en la función “Server()”, que hará desactivar la cámara de realidad aumentada y se activará la main camera, así como lo códigos QR ya que no lo va a necesitar, esto se hace con la intención que el servidor pueda estar observando todas las partidas del restaurante.

```
main.gameObject.SetActive(true);
ar.gameObject.SetActive(false);
tarjetas.gameObject.SetActive(false);
```

Figura 4.201: Modificación de la función “Server()”

4.10.2. Incorporación de las ImageTarget

Se comentará como se añade los marcadores para nuestra aplicación. En la url comentada anteriormente en la anterior sección, se pulsa sobre la sección “Target Manager”, que se puede ver al lado de la pestaña de la licencia

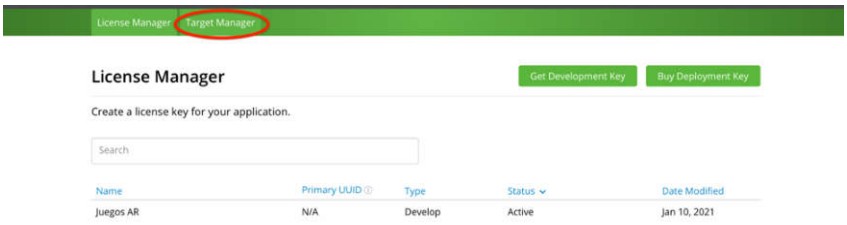


Figura 4.202: Ir a la ventana Target Manager

Dentro de la sección de Target Manager se creará una base de datos en el cual está compuesta por siete códigos QR.

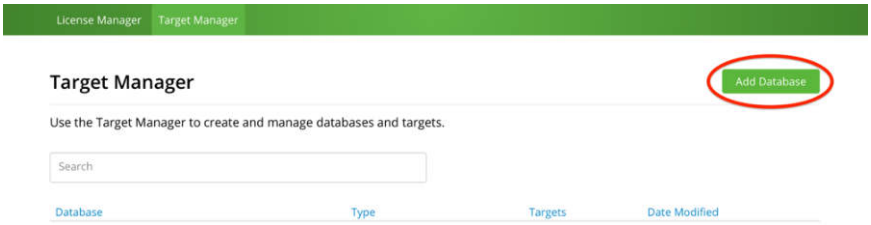


Figura 4.203: Añadir base de datos

Se pulsará sobre “Add Database” para crear una base de datos, que incluirá todas las imágenes que se utilizará como marcadores en nuestro proyecto de realidad aumentada.

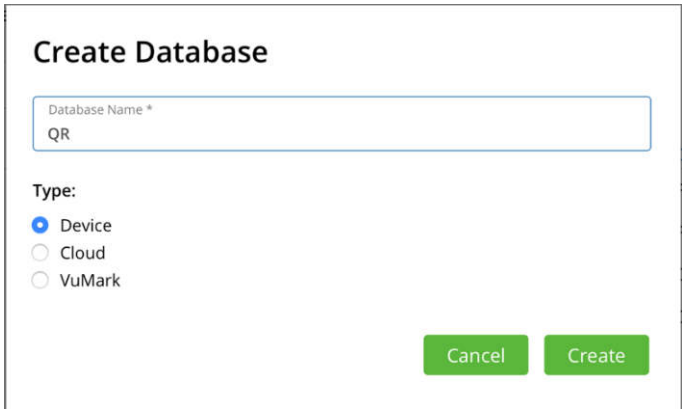


Figura 4.204: Creando la base de datos QR

Una vez creada, se pulsa sobre el nombre de la base de datos y no saldrá algo parecido a esto:

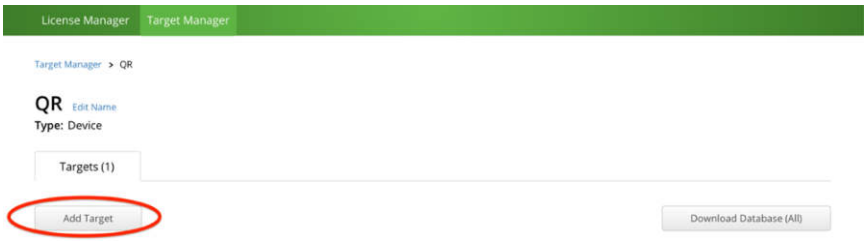


Figura 4.205: Botón de añadir marcador

Se pulsa sobre la opción “Add Target”, nos aparece una opción donde se puede elegir el tipo de Target que se quiere, pero en este caso se va a elegir “Single Image”.

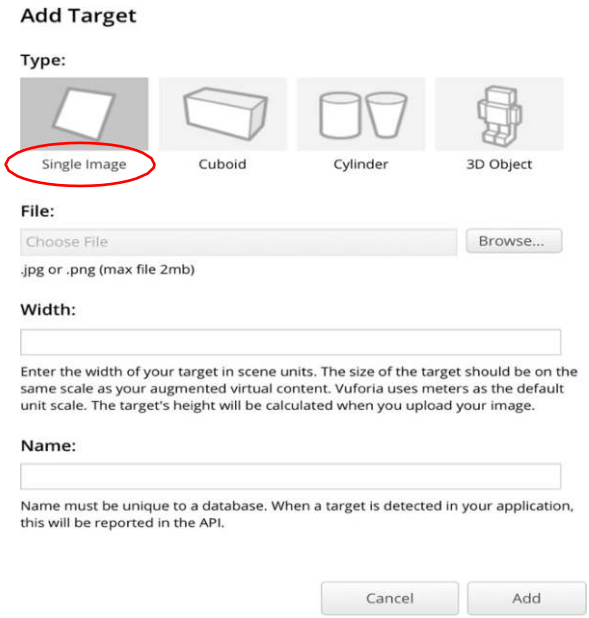


Figura 4.206: Configuración del marcador

Para elegir la imagen, se pulsa sobre “browse” y se elige la imagen , en “width” se debe de indicar un tamaño que tenga dentro de un ítem, y ponerle un nombre.

Una vez que se haya agregado todos los códigos QR, realizando el mismo procedimiento para cada una, se pulsa sobre “Download Database”, para descargar la base de datos.

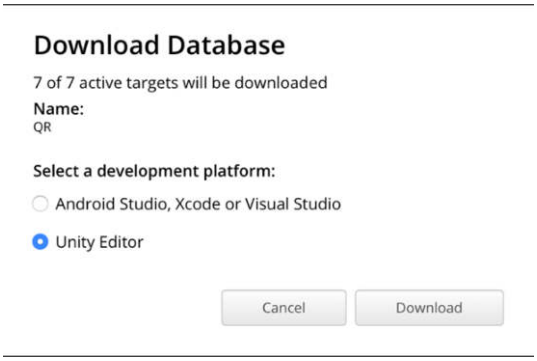


Figura 4.207: Descargar la base de datos

Cuando este descargado, se importará el paquete para poder utilizarlo. A continuación, en la ventana de jerarquía se creará un objeto vacío llamado “Tarjetas” en el cual contendrá las siete Imagen Target que se usará en este proyecto. Para crear las Imagen Target se realizará de la misma manera que aparece en la imagen.

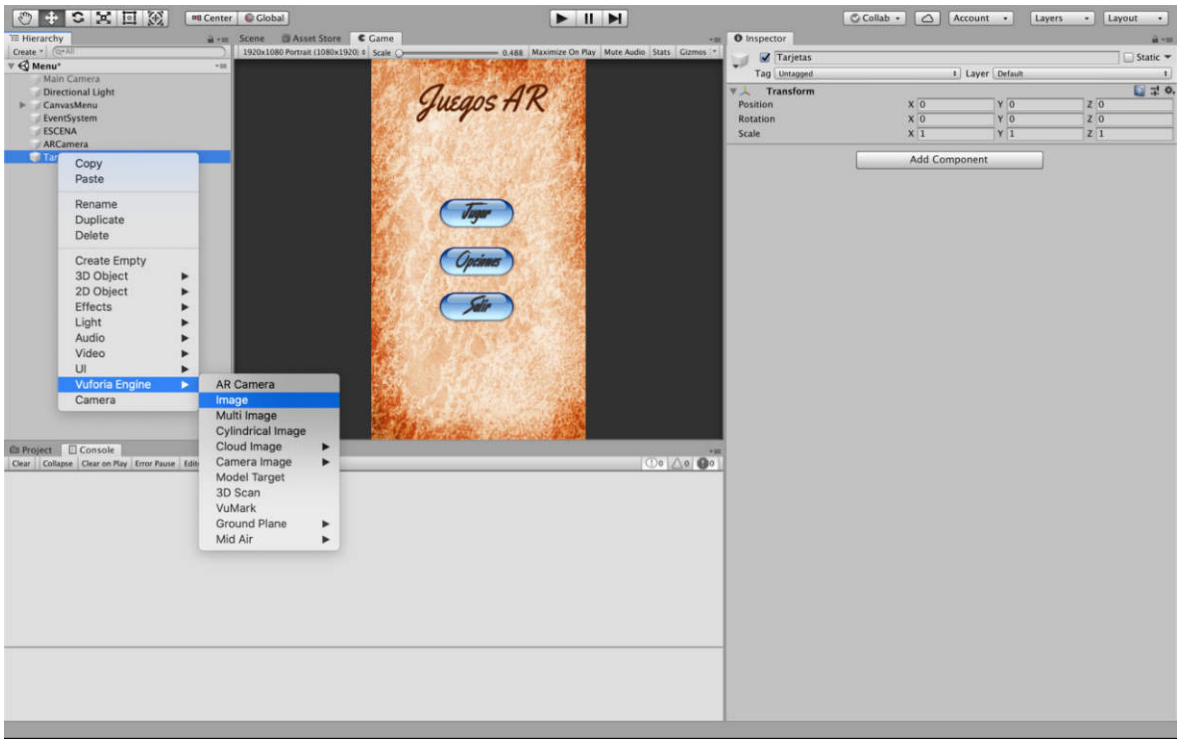


Figura 4.208: Añadir un marcador a la escena

Una vez que se tiene añadido las siete “ImageTarget” se procederá a la selección de cada “ImageTarget” con un código QR, tal y como se muestra a continuación:

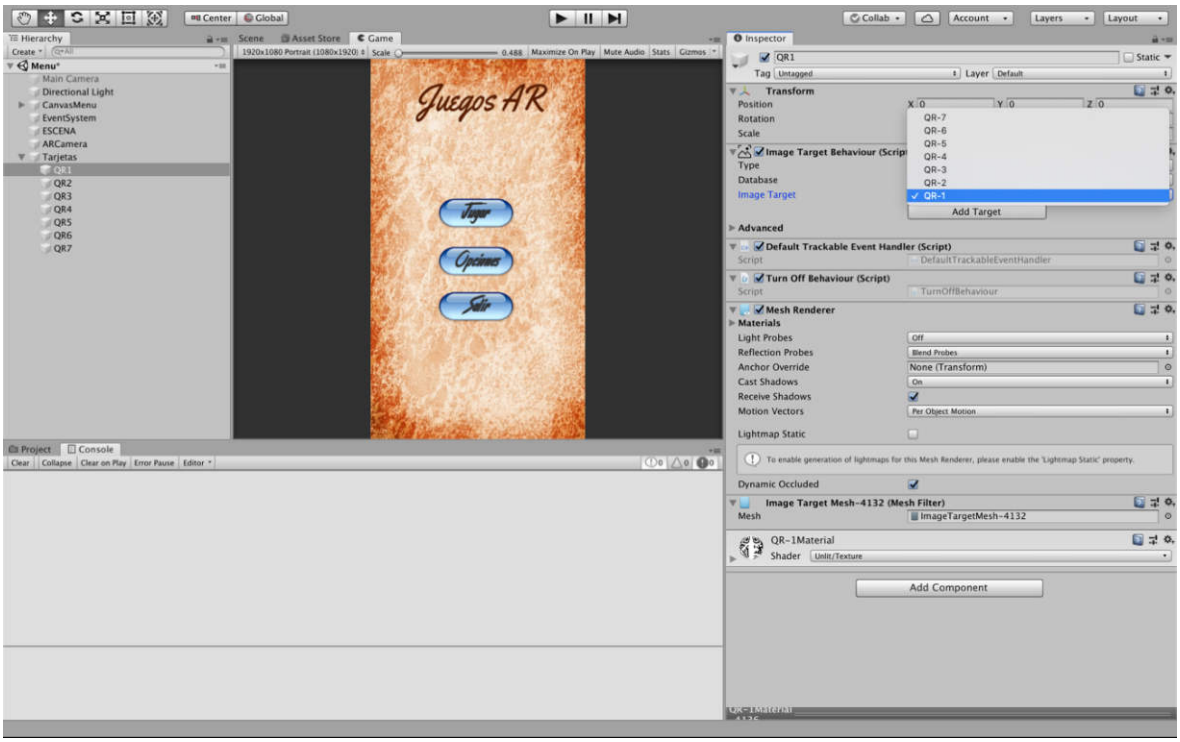


Figura 4.209: Selección del marcador de la base de datos

A continuación, se realizará algunos cambios de posición rotación y escala en el "Canvas" de cada escena, para que este a la misma altura que los códigos QR. A continuación se muestra un ejemplo del "CanvasMenu".

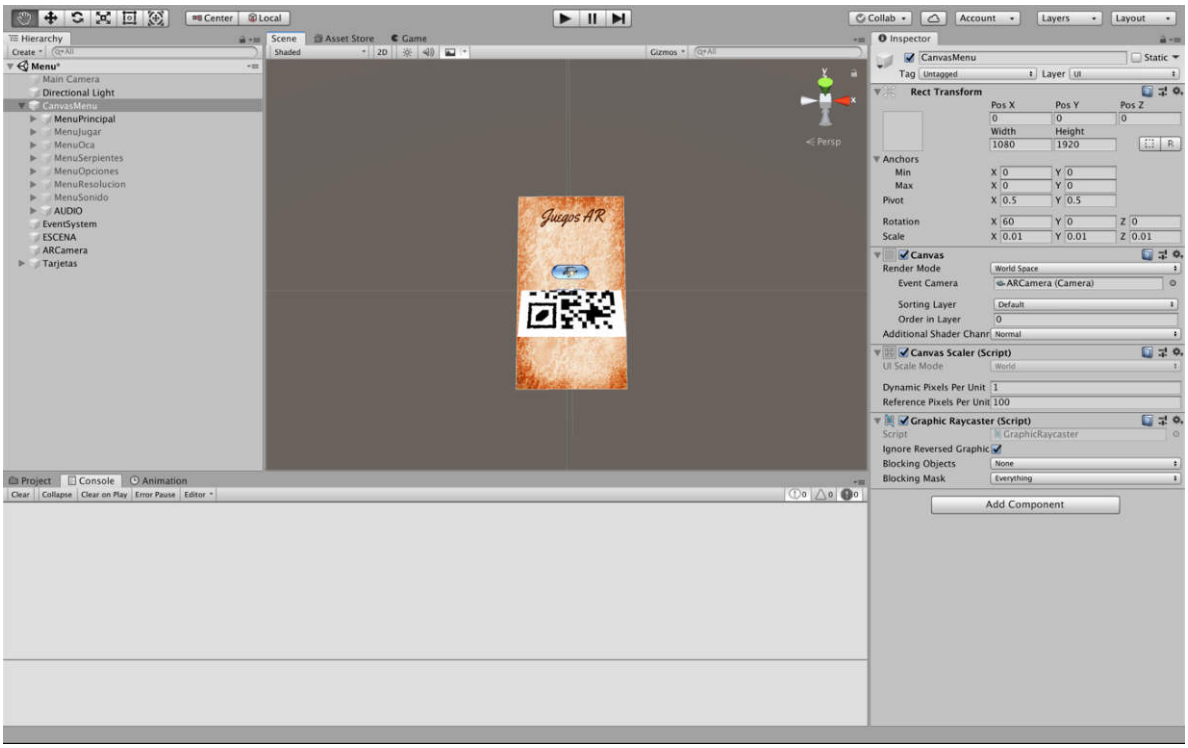


Figura 4.210: Cambio de parámetros en el Canvas

4.10.3. Detección del marcador

En esta sección se compondrá de tres script, en función de la escena en la que se encuentre se utilizará uno u otro. Los tres script se encarga de realizar que elementos aparece en la escena en función del estado en el que se encuentre dicho marcador, para ello se realizará una modificación del script que viene en conjunto con ImageTarget llamado "DefaultTrackableEventHandler".

El primer script llamado "DeteccionObjeto.cs", será utilizado todas las escenas excepto en las escenas de selección del restaurante y de la mesa, en el cual desactivará el renderizado de todos los elementos que conforma el objeto padre, cuando no detecte el marcador. Por el contrario cuando detecte el marcador de activará el renderizado de todos los objetos compuestos por el objeto padre.

Por otro lado, en el objeto padre se añadirá un "CanvasGroup" de forma que se pueda manejar una interfaz de usuario de forma global sin necesidad de hacerlo de manera individual, por ello en el script cuando no detecte el marcador, esta "CanvasGroup" accederá al método "Alpha" y lo pondrá a 0, esto hará que sea invisible, y cuando detecte el marcado este lo pondrá a 1, esto hará que sea visible. Por otro lado el "CanvasGroup", también accederá al método "BlocksRaycasts" que estará desactivado cuando no encuentre el marcador de forma que bloqueará la transmisión del rayo, y cuando encuentre el marcador permitirá la transmisión del rayo. Aquí hay una modificación en la parte red, ya que el servidor no va a utilizar esta tecnología, sino que va a estar observando, en la función Start(), tanto del script "OcaMulti.cs" como de "SerpiernteMulti.cs", el servidor tendrá tanto el tablero como el texto del tablero activado.

A continuación se muestra el fragmento de código que compete a este script que se ha comentado:

```
public void OnTrackableStateChanged(
    TrackableBehaviour.Status previousStatus,
    TrackableBehaviour.Status newStatus)
{
    m_PreviousStatus = previousStatus;
    m_NewStatus = newStatus;

    Debug.Log("Trackable " + mTrackableBehaviour.TrackableName +
              " " + mTrackableBehaviour.CurrentStatus +
              " -- " + mTrackableBehaviour.CurrentStatusInfo);

    if (newStatus == TrackableBehaviour.Status.DETECTED ||
        newStatus == TrackableBehaviour.Status.TRACKED ||
        newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
    {

        objeto.GetComponentInChildren<CanvasGroup>().alpha = 1;
        objeto.GetComponentInChildren<CanvasGroup>().blocksRaycasts = true;
        foreach (Renderer objeto in objeto.GetComponentsInChildren<Renderer>())
        {
            objeto.enabled = true;

        }

    }
    else if (previousStatus == TrackableBehaviour.Status.TRACKED &&
             newStatus == TrackableBehaviour.Status.NO_POSE)
    {

        objeto.GetComponentInChildren<CanvasGroup>().alpha = 0;
        objeto.GetComponentInChildren<CanvasGroup>().blocksRaycasts = false;
        foreach (Renderer objeto in objeto.GetComponentsInChildren<Renderer>())
        {
            objeto.enabled = false;

        }

    }
    else
    {
        // For combo of previousStatus=UNKNOWN + newStatus=UNKNOWN|NOT_FOUND
        // Vuforia is starting, but tracking has not been lost or found yet
        // Call OnTrackingLost() to hide the augmentations

        objeto.GetComponentInChildren<CanvasGroup>().alpha = 0;
        objeto.GetComponentInChildren<CanvasGroup>().blocksRaycasts = false;
        foreach (Renderer objeto in objeto.GetComponentsInChildren<Renderer>())
        {
            objeto.enabled = false;

        }


    }
}
#endregion // PROTECTED_METHODS
```

Figura 4.211: Script “DeteccionObjeto.cs”

El GameObject “objeto”, dependerá de la escena en la que nos encontremos, por ejemplo en la escena “OcaMulti”, este objeto será “TableroOca” así como en la escena “FinalOcaMulti”, el objeto será “PanelFinal”, se mostrará ambas imágenes para mayor aclaración.

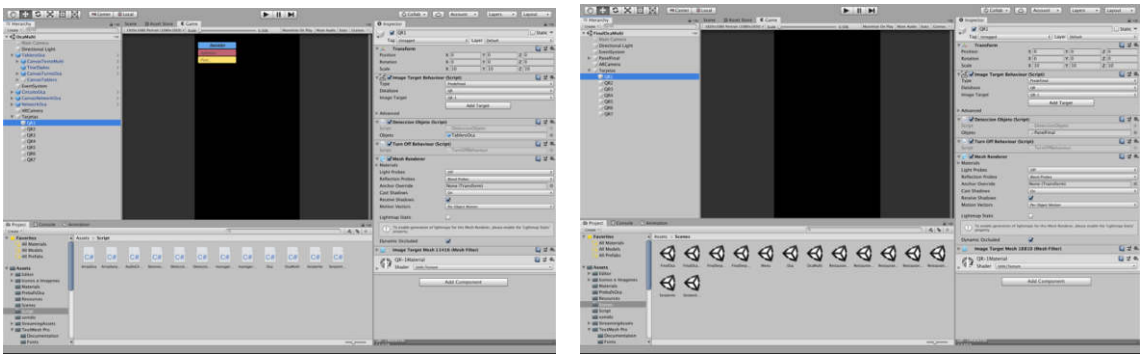


Figura 4.212: Añadir el objeto que detecta el marcador

Por consiguiente, el segundo script llamado “DeteccionRestauranteMesa.cs”, estará compuesto por el mismo fragmento de código anterior al que se le añadirá un conjunto de array de botones que en función de que código QR lea desactivará todos excepto el que no se encuentra dentro de ese array.

A continuación se muestra dicho script:

```
public void OnTrackableStateChanged(
    TrackableBehaviour.Status previousStatus,
    TrackableBehaviour.Status newStatus)
{
    m_PreviousStatus = previousStatus;
    m_NewStatus = newStatus;

    Debug.Log("Trackable " + mTrackableBehaviour.TrackableName +
              " " + mTrackableBehaviour.CurrentStatus +
              " -- " + mTrackableBehaviour.CurrentStatusInfo);

    if (newStatus == TrackableBehaviour.Status.DETECTED ||
        newStatus == TrackableBehaviour.Status.TRACKED ||
        newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
    {

        objeto.GetComponentInChildren<CanvasGroup>().alpha = 1;

        foreach (Renderer objeto in objeto.GetComponentsInChildren<Renderer>())
        {
            objeto.enabled = true;

        }
        foreach (Button miboton in BotonesMesas)
        {
            miboton.interactable = false;
        }
    }

    else if (previousStatus == TrackableBehaviour.Status.TRACKED &&
             newStatus == TrackableBehaviour.Status.NO_POSE)
    {

        objeto.GetComponentInChildren<CanvasGroup>().alpha = 0;

        foreach (Renderer objeto in objeto.GetComponentsInChildren<Renderer>())
        {
            objeto.enabled = false;

        }
        foreach (Button miboton in BotonesMesas)
        {
            miboton.interactable = true;
        }
    }

    else
    {
        // For combo of previousStatus=UNKNOWN + newStatus=UNKNOWN|NOT_FOUND
        // Vuforia is starting, but tracking has not been lost or found yet
        // Call OnTrackingLost() to hide the augmentations

        objeto.GetComponentInChildren<CanvasGroup>().alpha = 0;
        foreach (Renderer objeto in objeto.GetComponentsInChildren<Renderer>())
        {
            objeto.enabled = false;

        }
    }
}
#endregion // PROTECTED_METHODS
```

Figura 4.213: Script "DeteccionRestauranteMesa.cs"

Si se detecta el código QR1, se desactivarán todos las mesas excepto la mesa 1, si por ejemplo se detecta el código QR2, se desactivarán todas las mesas excepto la mesa 2 y así sucesivamente. A continuación se muestra ambas imágenes para una mayor aclaración:

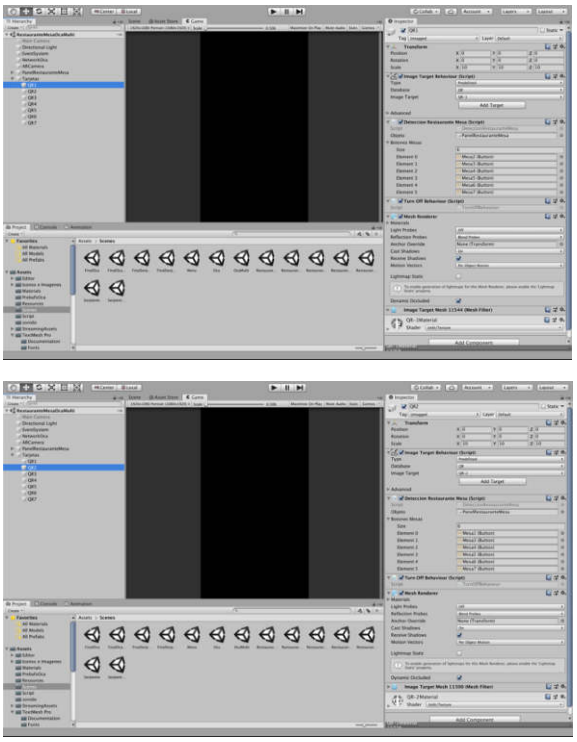


Figura 4.214: Añadir los objeto que detecta el marcador

Por otro lado, el script “DeteccionTablero.cs”, será usado en las escenas “Oca” y “Serpiente”, para saber en que tablero se encuentra el usuario, en red esto se hacia a través del puerto pero en este caso no se va a poder realizar por el puerto ya que en el modo de un jugador esta opción no esta disponible.

Por tanto, en este script lo que hará es que cuando detecte el marcador iguale el texto a la variable de tipo String creada en dicho script, y en el editor en cada tarjeta se podrá para cada QR de forma manual el nombre que se llamará el texto en nuestro caso “Tablero 1”, si es el código QR1, “Tablero 2”, si es el código QR2, y así sucesivamente con todos los códigos QR.

A continuación se mostrará dicho script:

```
public void OnTrackableStateChanged(
    TrackableBehaviour.Status previousStatus,
    TrackableBehaviour.Status newStatus)
{
    m_PreviousStatus = previousStatus;
    m_NewStatus = newStatus;

    Debug.Log("Trackable " + mTrackableBehaviour.TrackableName +
              " " + mTrackableBehaviour.CurrentStatus +
              " -- " + mTrackableBehaviour.CurrentStatusInfo);

    if (newStatus == TrackableBehaviour.Status.DETECTED ||
        newStatus == TrackableBehaviour.Status.TRACKED ||
        newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
    {

        GameObject.Find("tablero").GetComponentInChildren<TextMeshProUGUI>().enabled = true;
        GameObject.Find("tablero").GetComponentInChildren<TextMeshProUGUI>().text = tablero;

    }
    else if (previousStatus == TrackableBehaviour.Status.TRACKED &&
             newStatus == TrackableBehaviour.Status.NO_POSE)
    {

        GameObject.Find("tablero").GetComponentInChildren<TextMeshProUGUI>().enabled = false;

    }
    else
    {
        // For combo of previousStatus=UNKNOWN + newStatus=UNKNOWN|NOT_FOUND
        // Vuforia is starting, but tracking has not been lost or found yet
        // Call OnTrackingLost() to hide the augmentations

        GameObject.Find("tablero").GetComponentInChildren<TextMeshProUGUI>().enabled = false;

    }
}
#endregion // PROTECTED_METHODS
}
```

Figura 4.215: Script “DeteccionTablero.cs”

Como se ha comentado, la variable de tipo “String” llamada “tablero” como se ve en la imagen anterior se igualará al objeto de tipo “TextMeshPro” creada en la escena, y en cada QR se le indicará el texto deseado. A continuación, se muestra dos imágenes para ver como se añade este texto de forma manual.

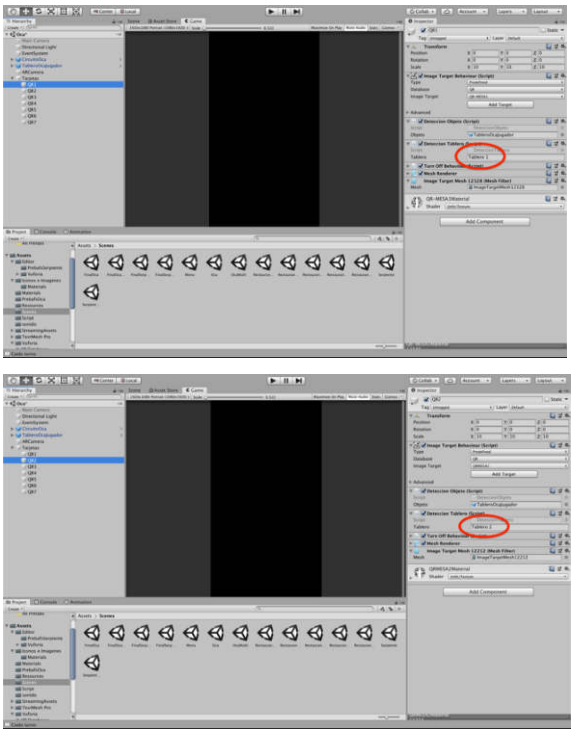


Figura 4.216: Añadir los textos a los QR

Por último, se creará un nuevo script para las escenas que están compuestas solo por un “Canvas”, este script llamado “GirarObjetoCanvas”, será necesario porque si se cambia la dirección de la cámara, el usuario verá el “Canvas” desde otra perspectiva y como lo se quiere es que el usuario tenga una perspectiva que apunte al dispositivo que lee el código QR de forma que lo tenga enfrente. Lo que e hace con este script es cambiar los ejes YZ de la dirección del objeto virtual en este caso el “Canvas” de la escena, de forma que sea igual a los ejes YZ de la dirección de la cámara. Por tanto, mire en la dirección que se mire aparezca el objeto virtual en frente al usuario, y no en una dirección opuesta o girado 90º.

Este script, lo tendrá todos los códigos QR, para que pueda hacer esta modificación en cada uno de ellos, y se le indicará que objeto se quiere que realice esta modificación, para ello en cada escena se le arrastrará el objeto “Canvas”, a la variable creada “giroCanvas”. A continuación, se muestra dicho script:

```
public class GirarObjetoCanvas : MonoBehaviour
{
    public GameObject giroCanvas;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        GameObject camara = GameObject.Find("ARCamera");
        giroCanvas.transform.rotation = Quaternion.Euler(60, camara.transform.eulerAngles.y, camara.transform.eulerAngles.z);
    }
}
```

Figura 4.217: Script “GirarObjectoCanvas.cs”

5. PRUEBAS REALIZADAS

En este capitulo se mostrará algunas imágenes de las pruebas que se ha ido haciendo para finalizar este proyecto. Las pruebas se han realizado en diferentes dispositivos, a continuación se detalla cuales han sido los dispositivos utilizados:

- Portátil Macbook Pro
- iPhone 12 Pro
- Xiaomi mi 8
- Xiaomi mi A3
- Huawei P Smart
- Tablet Samsung Tab A

Se verá primeramente algunas imágenes del modo “Un Jugador”, tanto de la oca como el juego de la serpiente y escalera:

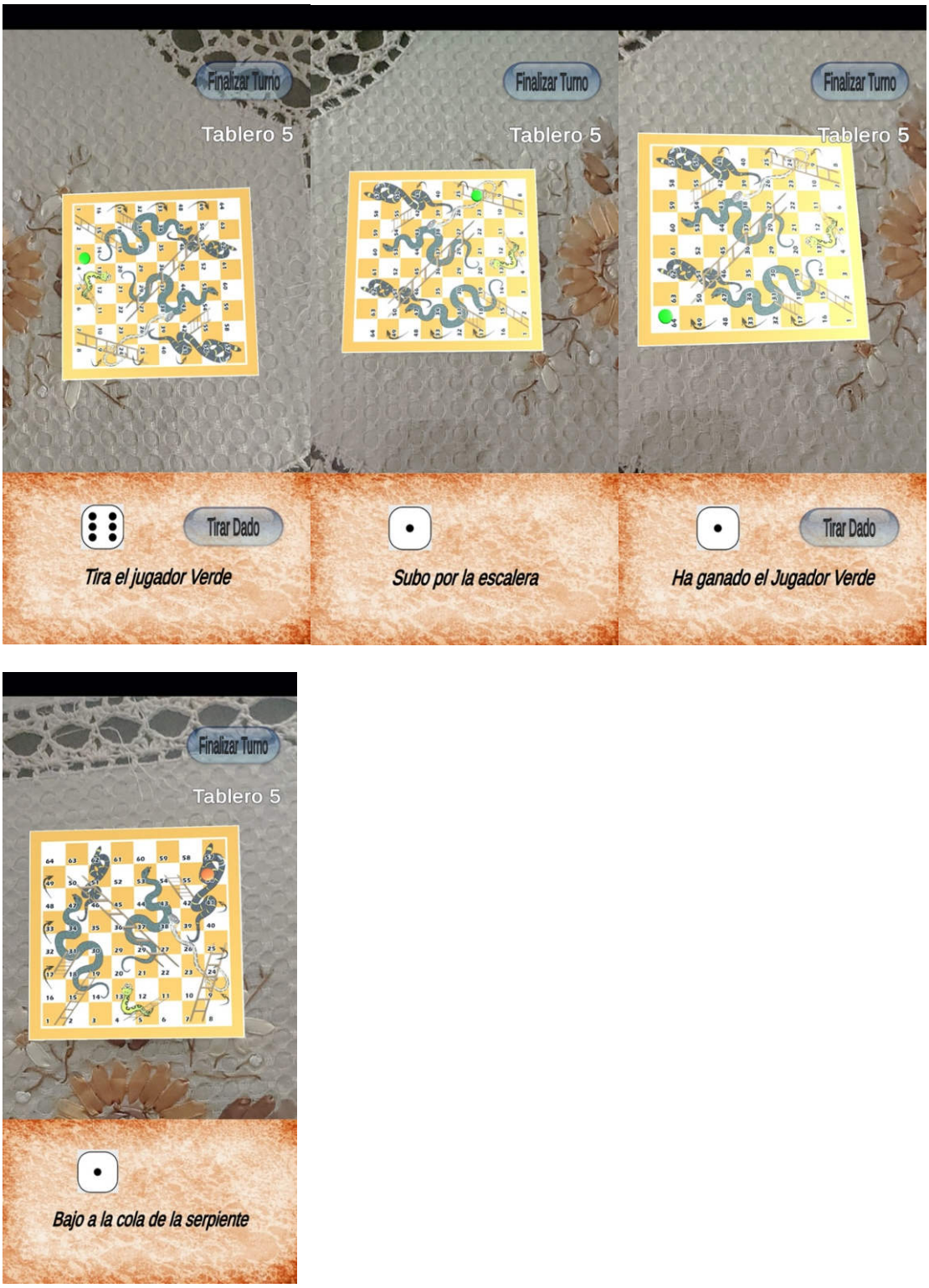


Figura 5.1: Pruebas del modo "Un Jugador"

Por consiguiente, se vera las imágenes del modo multijugador tanto de la oca como el juego de la serpiente y escaleras:

Figura 5.2: Pruebas del modo multijugador

6. CONCLUSIONES Y LINEAS FUTURAS

La principal conclusión de este proyecto es concluir que se han conseguido todos los objetivos especificados de este proyecto descritos en el capitulo 3.

A breve resumen, se ha desarrollado una aplicación para móviles y Tablet de realidad aumentada que permita a más de un jugador a través de una conexión a internet.

Para el desarrollo de este proyecto, ha sido necesario saber como funcionan los juegos en red, así como algunas funciones especificas para poder llevarlo a cabo. Por otro lado entender como funciona la tecnología de Realidad Aumentada con el uso de marcadores, así como el propio programa de Unity, y de tener alguna base en programación.

A nivel personal, ha sido satisfactorio trabajar en este ámbito y como poco a poco se ha ido cumpliendo los objetivos de este proyecto. La trayectoria de este proyecto ha sido larga dada que la mayoría de los campos que presenta el proyecto no había sido explorados con anterioridad así como su dificultad. Por otro lado, ha sido interesante como a través de la realidad aumentada te puedas sumergir en lo virtual y como conectar con esos objetos que son totalmente virtuales.

Con respecto a líneas futuras de este proyecto, se podría crear una conexión bluetooth, así como poder conectarse con otros establecimientos para jugar con amigos que se encuentren en otro lugar.

Se podría implementar un sistema de puntos, en el cual tenga algunos descuentos en el restaurante donde se ha jugado la partida, así como realizar torneos entre varias mesas del restaurante.

Permitir acceder con la cuenta de google, y cuando haya finalizado la partida poder hacer comentarios que estén enlazados con la aplicación TripAdvisor.

Implementar juegos con una lógica más compleja que permita al usuario tener una amplia variedad de juegos, y escoger el que más le guste.

7. PRESUPUESTO

En este capitulo, se detallará el presupuesto de toda la realización de este proyecto, en el que se tendrá en cuenta el coste del ingeniero, así como costes de luz, transporte y comida entre otros. Para más detalle, se muestra una imagen todo desglosado de cada coste.

TFG

PRESUPUESTO

Detalle

GASTOS	CANTIDAD €/h	HORAS	GASTO FINAL
Ingeniero Senior	31,00 €	200	6.200,00 €
Curso de aprendizaje	3,00 €	100	300,00 €
Licencia de Unity (estudiante)	0,00 €	0	0,00 €
Licencia de Vuforia (Developer)	0,00 €	0	0,00 €
Otros	4,60 €	420	1.932,00 €
Total			**8.432,00 €**

Figura 7.1: Presupuesto del TFG

8. BIBLIOGRAFÍA Y REFERENCIAS

[1] https://www.innovae.eu/la-realidad-aumentada/

[2] https://rockcontent.com/es/blog/realidad-aumentada/

[3] https://iat.es/tecnologias/realidad-aumentada/

[4] https://www.vass.es/estos-son-los-sectores-en-los-que-triunfa-la-realidad-aumentada/

[5] https://itcl.es/blog/cinco-sectores-donde-encontrar-realidad-aumentada/

[6] https://expansion.mx/tecnologia/2019/11/06/el-negocio-detras-de-las-apps-de-realidad-aumentada-moda

[7] https://colaboraeducacion30.juntadeandalucia.es/educacion/colabora/documents/131195/980947/Realidad+aumentada+con+Aumentaty/dac0d367-cada-4627-9289-3a51ed0aba36?version=1.0&previewFileIndex=

[8] https://www.xataka.com/espacionokia/realidad-aumentada-la-historia-desde-la-ciencia-ficcion-hasta-la-aplicacion-practica

[9] https://blogthinkbig.com/realidad-aumentada-origen

[10] https://www.onirix.com/es/el-pasado-presente-y-futuro-de-la-realidad-aumentada/

[11] http://wwwavancesdelcelular.weebly.com/historia.html

[12]http://informaticanatalau.weebly.com/62/tecnicas-de-visualizacion-de-realidad-aumentada

[13] https://sites.google.com/site/realidadvirtualmk/tecnicas-de-visualizacion

[14] http://larealidadaumentada.weebly.com/teacutecnicas-de-visualizacioacuten.html

[15] http://rcasado.es/raumentada.html

[16]https://es.wikipedia.org/wiki/Realidad_aumentada#Software_para_la_realidad_aumentada

[17] https://www.neosentec.com/realidad-aumentada/

[18] https://www.onirix.com/es/aprende-sobre-ra/diferentes-tipos-de-realidad-aumentada/

[19] https://ardev.es/realidad-aumentada/

[20]https://es.wikipedia.org/wiki/Realidad_aumentada#Software_para_la_realidad_aumentada

[21]https://byzness.elperiodico.com/es/innovadores/20200219/hololens-2-microsoft-trabajar-realidad-mixta-empresas-7854453

[22] https://www.apple.com/la/augmented-reality/

[23] https://www.amara-marketing.com/blog-pymes/app-movil-realidad-aumentada-ikea

[24] https://www.elsate.com/viewtopic.php?t=1876

[25] https://blog.terranea.es/realidad-aumentada-futuro-automovil/

[26] https://www.sobreruedas.news/destacados/vw-lanza-primera-aplicacion-de-realidad-aumentada/

[27] https://www.neosentec.com/arkit-vs-arcore-realidad-aumentada-movil/

[28] https://www.fnac.es/Arcore-y-Arkit-Mas-alla-de-Pokemon-Go/cp1375/w-4

[29]https://www.applesfera.com/ios/arcore-es-la-reaccion-de-google-contra-el-arkit-de-apple

[30] https://androiddistribution.io/#/

[31]https://xpell.co/aproximadamente-600-millones-de-personas-usan-realidad-aumentada-movil/

[32] https://www.masterd.es/blog/que-es-unity-3d-tutorial/

[33] https://docs.unity3d.com/es/530/Manual/LearningtheInterface.html

[34] https://docs.unity3d.com/es/530/Manual/ProjectView.html

[35] https://docs.unity3d.com/es/530/Manual/GameObjects.html

[36] https://docs.unity3d.com/es/2018.4/Manual/PartSysWhatIs.html

[37] https://docs.unity3d.com/es/530/Manual/UsingDX11GL3Features.html

[38] https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)

[39] https://3dbusinessschool.com/las-cinco-mejores-caracteristicas-de-unreal-engine-4/

[40] https://docs.viromedia.com/docs/viro-platform-overview

[41] https://reactnative.dev

[42] https://es.wikipedia.org/wiki/Juego_de_la_oca

[43] http://www.juegodelaoca.com/Reglamento/reglamento.htm

[44] https://es.wikihow.com/jugar-%22Serpientes-y-escaleras%22

[45] https://deserpientes.net/juego-serpientes-y-escaleras

[46]http://www.colegiomiralbueno.es/archivos/11/Juegos_para_combatir_el_aburrimiento_en_tiempos_de_confinamiento-2.pdf

[47] https://docs.unity3d.com/es/2018.4/Manual/UNetOverview.html

[48] https://docs.unity3d.com/es/530/Manual/UnityManual.html

[49] Eva M Sánchez Lavi. *Carta de restaurante virtual mediante realidad aumentada AR.* Trabajo de Fin de Grado: Universidad de Jaén, 2019

[50] José Reyes Romero. *Guía para museos con realidad aumentada AR*. Trabajo de Fin de Grado: Universidad de Jaén, 2019

[51] David Sánchez Illescas, Orquesta virtual basada en realidad aumentada AR. Trabajo de Fin de Grado: Universidad de Jaén, 2019.

9. GLOSARIO DE TERMINOLOGÍA EMPLEADA

Se especificará y detallará algunos términos empleados en este proyecto, para una mayor compresión al lector sobre el trabajo realizado.

- **UI:** Sus siglas corresponde a Interfaz de Usuario, es el modo en que el jugador puede interaccionar con la aplicación, como por ejemplo el uso de botones para mover un determinado objeto.
- **Create Empty:** Objeto vacío, se puede utilizar para hacer referencia a un objeto.
- **Canvas:** Es el área donde todos los elementos de interfaz de usuario deben estar presentes, como textos, botones, imágenes entre otros. El Canvas, es un GameObject con un componente Canvas en el, y todos los elementos de interfaz de usuarios son hijos de este
- **Prebafs:** Son objetos reutilizables, y creados con una serie de características dentro de la vista del proyecto, que serán instanciados en la aplicación cuando sea pertinente y tantas veces que sea necesario.
- **Sprite:** Son objetos gráficos en 2D, estos son esencialmente texturas estándar.
- **RawImage:** Muestra una imagen no interactiva al usuario, ya que son utilizados para decoración, iconos, etc…
- **Renderer:** Es el proceso de producción de una imagen que esta basado en datos tridimensionales.
- **GameObject:** Son entidades que se pueden colocar en las escenas, en el cual tendrá componente llamado “Transform”, en el que indicará cual es su posición, rotación y escala en dicha escena.
- **Imagen Target:** Rastrea una imagen objetivo en la pantalla, se usa para utilizar la Realidad Aumenta y actúan como marcador.
- **Local Player:** Realiza la función de que solo el objeto propio (local), pueda hacer uso de ella.
- **ClientRpc:** Es un comando que se utiliza en juegos multijugdor, y la función que realizan es enviar información desde el servidor a todos los clientes.
- **Commands:** Es un comando que se utiliza en juegos multijugador, y la función que realizan es enviar la información desde el cliente al servidor.
- **SyncVar:** Son variables de scripts que heredan de NetworkBehaviour, que se sincroniza desde el servidor a los clientes.

- **AudioSource:** Reproduce cualquier tipo AudioClip en la escena además de poder configurarlo.

- **AudioClip:** Contiene información de audio utilizada por Audio Sources. Unity soporta mono, stereo y assets de audio multi-canal.
- **Corrutina:** Es una forma especial de hacer que la lógica suceda con el tiempo. Una corrutina es realmente solo un método con un tipo de devolución de “IEnumerator”.
- **Framework:** Se define como una especie de esquema de trabajo para realizar el desarrollo de software.

ANEXO 1: INSTALACIÓN Y PUESTA EN MARCHA DE UNITY

En este Anexo se tiene como objetivo como instalar el motor del juego, realizar los primeros pasos para la utilización del programa como por ejemplo como poner la cámara, seleccionar las opciones necesarias, introducir la licencia, entre otros. Además se incluirá como exportar la aplicación para utilizarlo en nuestros dispositivos móviles.

En primer lugar, debemos descargarnos el programa para ello se ira a la siguiente url: https://unity.com/es

Una vez que se este en la pagina web, iremos a la sección de “Descargar”, y le daremos a “Unity”.

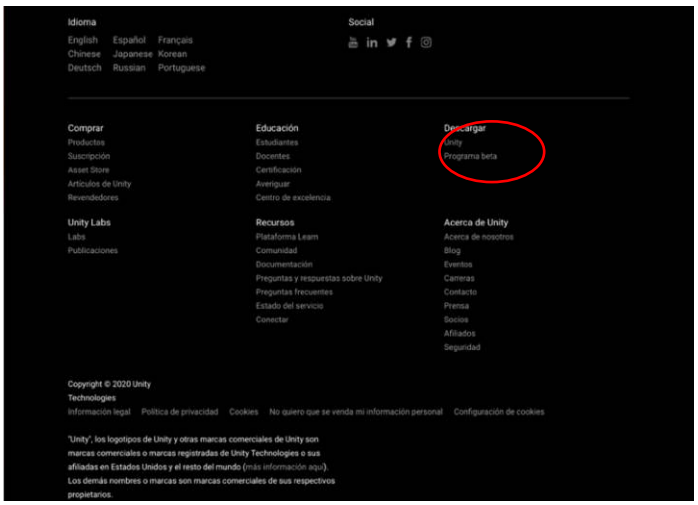


Figura A1.1: Descargar el programa

Saldrá la siguiente pestaña:

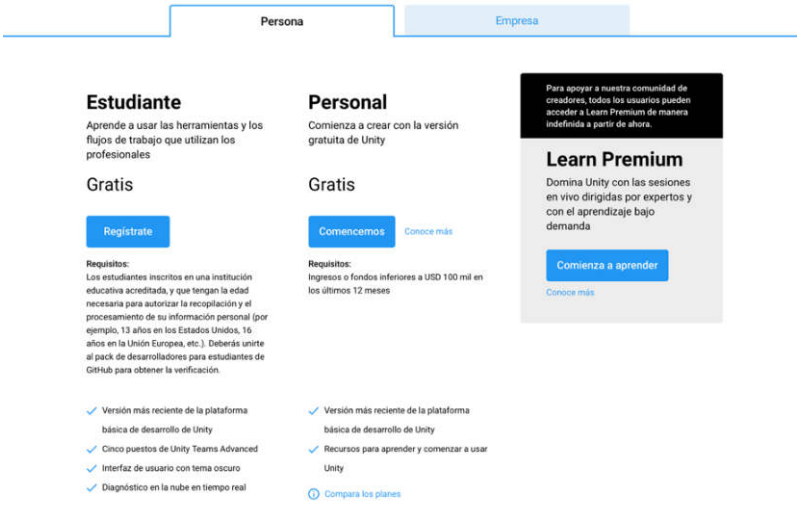


Figura A1.2: Versión de descarga

Se descargará la versión personal, que es totalmente gratis. Cuando hayamos dado a “Comencemos”, saldrá otra pestaña y buscaremos “Versiones de Unity”, y se seleccionará “Todas las versiones de Unity”.

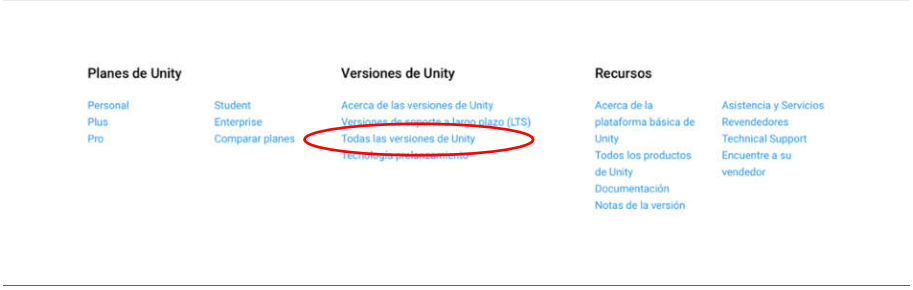


Figura A1.3: Todas las versiones de Unity disponibles

Saldrá todas las versiones que dispone Unity, en este caso se ha instalado la versión Unity 2018.4.14 para Mac. Comentar que esta versión de Unity es una versión de soporte a largo plazo (LTS), la diferencia de usar LTS, es que se puede recibir correcciones de Unity por un periodo de tiempo más largo sin tener que actualizar las versiones de Unity. Este soporte extendido es útil para algunos proyectos porque actualizar un proyecto a una nueva versión requiere de más trabajo. Por este motivo, se ha decido utilizar esta versión de Unity.

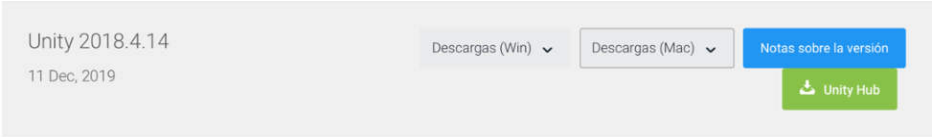


Figura A1.4: Versión de Unity

Cuando este descargado, se instalará el programa y nos pedirá que seleccionemos lo siguientes componentes mostrados en la siguiente imagen:

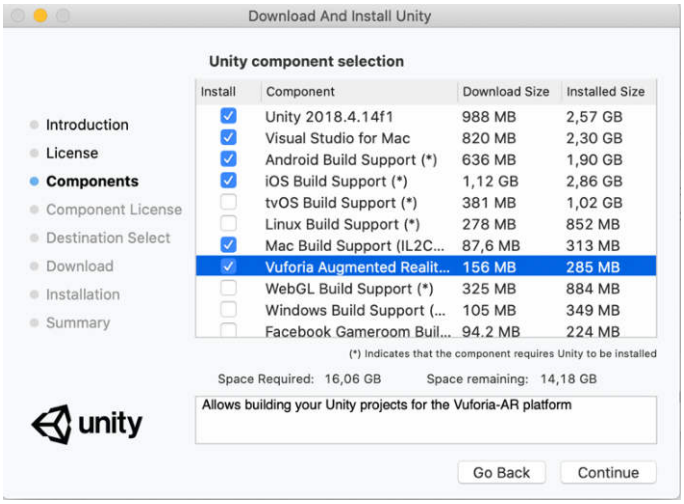


Figura A1.5: Componentes añadidos a Unity en la Instalación

Una vez ya instalado nos saldrá una pestaña parecido a lo que se mostrará a continuación:

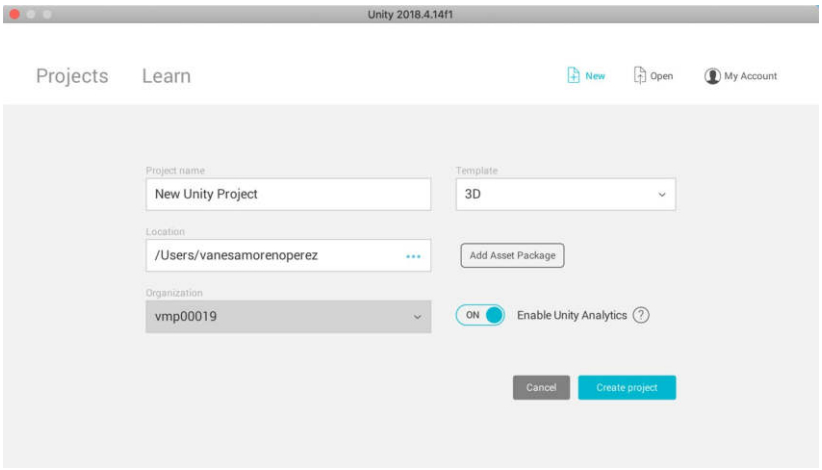


Figura A1.6: Crear un nuevo proyecto en Unity

Le damos a “New” para crear un proyecto, le pondremos el nombre que queramos y tenemos varias opciones de cómo queramos que sea el proyecto, en nuestro caso, he puesto en 3D, ya que lo que queremos tener un juego en tres dimensiones.

Una vez que hayamos configurado la creación del proyecto no saldrá la siguiente interfaz, en cual trabajaremos en nuestro proyecto.

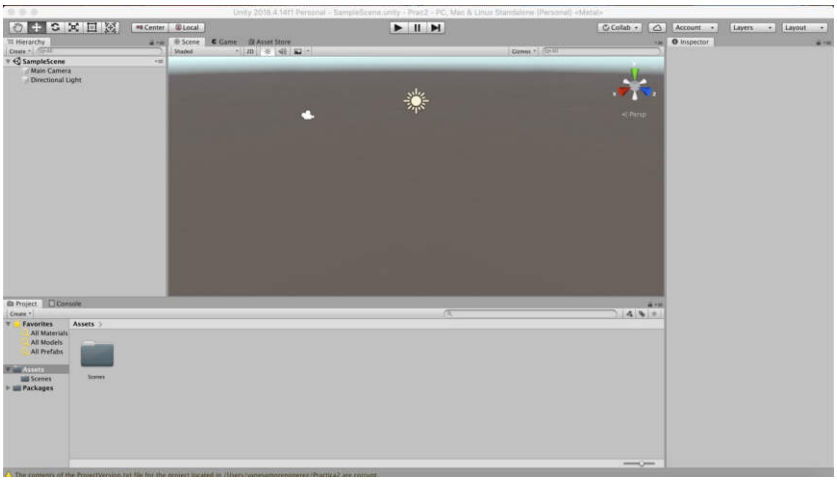


Figura A1.7: Vista de la Interfaz de Unity

Para elegir la plataforma, para ello se pulsa sobre “File” y se selecciona “Build Setting”, de esta manera no saldrá todas las plataformas que se podrá usar, en este proyecto usaremos Android y iOS. A continuación pulsamos en la pestana de “Switch Platform” para poder trabajar con la plataforma Android y de iOS.

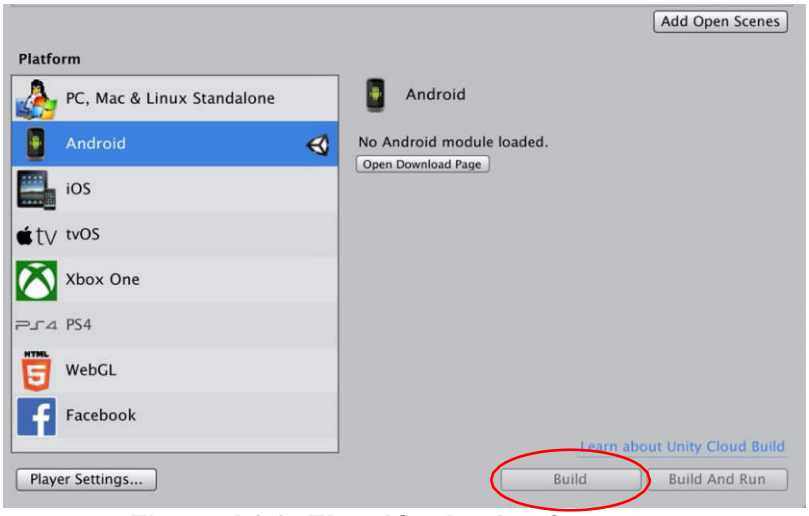


Figura A1.8: Elección de plataformas

ANEXO 2: REGLAS DE LA OCA

En este juego se va a seguir las reglas con 2 dados, a continuación se va a citar en este anexo las normas de este juego:

- Al inicio de la partida el jugador obtiene un 9 en la suma de ambos dados, puede ocurrir que:
    - Si los dados son 5 y 4, el jugador avanza a la casilla 53.
    - Si los dados son 6 y 3, el jugador avanza a la casilla 26.
- El orden del turnos, se guiará conforme los usuarios se vayan conectando, es decir, si hay por ejemplo 3 jugadores el primero en entrar tendrá el privilegio de empezar a jugar.
- Si el jugador cae en el laberinto (casilla 42), este tiene que retroceder a la casilla 30.
- Si el jugador cae en el puente (casilla 6 y 12), este tendrá que avanzar o retroceder hacia el otro puente, y volverá a tirar, recordando la frase “De puente a puente y tiro porque me lleva la corriente”. Por ejemplo, si el jugador cae en la casilla 6, este avanza a la casilla 12 y vuelve a tirar el mismo jugador.
- Si el jugador cae en los dados (casilla 26 y 53), este tendrá que avanzar o retroceder hacia el otro dado, y volverá a tirar, recordando la frase ”De dado a dado y tiro porque son cuadrados”. Poe ejemplo, si el jugador cae en la casilla 26, este avanza a la casilla 53, y vuelve a tirar el mismo jugador.
- Si el jugador cae en la calavera (casilla 58), este tiene que irse a la casilla de salida, y volver a empezar de nuevo.
- Si el jugador cae en alguna de las casilla de la oca (casillas 5,9,14,18,23,27,32,36,41,45,50,54,59), tendrá que ir hacia la siguiente oca y volver a tirar de nuevo, recordando la frase “De oca en oca y tiro porque me toca”.
- Si el jugador cae en la posada (casilla 19), este tiene que perder dos turnos.
- Si el jugador cae en el pozo (casilla 31), este tiene que perder dos turnos.
- Si el jugador cae en la cárcel (casilla 56), este tiene que perder tres turnos.
- A partir de la casilla 60, solo se puede jugar con un dado.
- Para ganar la partida, tienes que entra con los dados justo en la casilla 63, sino tienes que retroceder las posiciones sobrantes de los dados o del dado.

ANEXO 3: REGLAS DE LAS SERPIENTES Y ESCALERAS

En este juego se va a seguir las reglas con un solo dado, a continuación se va a citar en este anexo las normas de este juego:

- Cada jugador, deberá avanzar tantas casillas como el dado indique.
- El orden del turnos, se guiará conforme los usuarios se vayan conectando, es decir, si hay por ejemplo 3 jugadores el primero en entrar tendrá el privilegio de empezar a jugar.
- Si un jugador cae en una casilla donde se indica el principio de la escalera, deberá avanzar hasta el final de ella y quedarse en esa posición. Esta regla al contrario no se puede utilizar, solo es valida en sentido ascendente.
- Si un jugador cae en una casilla donde se indica la cabeza de una serpiente, este deberá bajar hasta la cola de la serpiente y quedarse en esa posición. Esta regla al contrario no se puede utilizar, solo es válida en sentido descendente.
- Si un jugador, saca un 6, este pondrá repetir la tirada. No hay limites para esta regla. Dependiendo de la versión del juego esta regla puede sufrir alguna variante. En este caso se seguirá la regla de que cuando salga un 6 se volverá a tirar el mismo jugador.
- Para finalizar la partida, el jugador deberá llegar con el numero del dado justo en la casilla 64, si el dado es mayor que las casillas por recorrer se deberá retroceder las posiciones sobrantes del dado.

ANEXO 4: MANUAL DE USUARIO DE LA APLICACIÓN

En este anexo, se explicará como usar la aplicación creada, así como las dos modalidades de juego que hay incorporadas.

Lo primero, se tiene que descargar la aplicación para poder usarla, en el establecimiento hay un código QR que lo enlaza con Drive en el caso de la plataforma Android y la App Store para el caso de iOS.

Cuando este descargada y se proceda a abrir la aplicación nos pedirá un permiso para poder hacer uso de la cámara, una vez que se ha aceptado el permiso ya se puede hacer uso de la aplicación leyendo el marcador QR que hay en la mesa donde este sentado el cliente.

Cuando el cliente lee el código QR, lo primero que se va a encontrar va a ser un menú de la aplicación, el usuario puede cambiar el tipo de resolución así como el sonido y efectos que tiene nuestro menú. Por otra parte en el opción jugar, el usuario tendrá acceso a dos juegos, en el cual elija la que elija tendrá dos modalidades de juego.

- ***Modalidad de juego simple***

  En esta modalidad, el usuario puede jugar al juego pero solo estará él, es decir, que solo vale para un único jugador. Se ha incluido esta modalidad, aunque ambos juegos estén pensado entre 2-4 jugadores, pero pueda dar la casualidad de que solo quiera jugar una persona, sin necesidad de conectarse a la wifi del local para poder jugar.

- ***Modalidad de juego multijugador***

  En esta modalidad, el usuario se puede conectar con los jugadores presentes en las mesas, en el cual pueden disfrutar de una partida entre los comensales. Cada mesa, tendrá una partida diferente, debido a que en la selección de la mesa el usuario esta indicando a que puerto del servidor se va a conectar. El usuario necesita tener el wifi activado del local para poder hacer uso de esta modalidad.

Tanto en una modalidad como en otra, cuando el jugador que sea gane la partida, podrá elegir si quiere de nuevo jugar o salir de la aplicación.

ANEXO 5: MANUAL DE INSTALACIÓN DE EJECUTABLES DE LA APLICACIÓN

En este anexo, se indicará como generar los distintos ejecutables que tendrá tanto el propietario del restaurante como el del usuario.

Los ejecutables del propietario del restaurante serán los siete servidores del juego de la oca y los otros siete servidores del juego de la serpiente y escalera. Por tanto cada ejecutable estará basado en una única escena, en caso del juego de la oca, la escena será “OcaMulti”, y en el otro caso la escena será “SerpienteMulti”.

Para guardar cada ejecutable, cada servidor se llamara de la siguiente manera, y lo guardaremos en la carpeta llamada “EjecutablesJuegosAR”:

- “OcaServi1” para el caso del puerto “1111”.
- “OcaServi2” para el caso del puerto “2222”.
- “OcaServi3” para el caso del puerto “3333”.
- “OcaServi4” para el caso del puerto “4444”.
- “OcaServi5” para el caso del puerto “5555”.
- “OcaServi6” para el caso del puerto “6666”.
- “OcaServi7” para el caso del puerto “7777”.
- “SerpienteServi1” para el caso del puerto “0111”.
- “SerpienteServi2” para el caso del puerto “0222”.
- “SerpienteServi3” para el caso del puerto “0333”.
- “SerpienteServi4” para el caso del puerto “0444”.
- “SerpienteServi5” para el caso del puerto “0555”.
- “SerpienteServi6” para el caso del puerto “0666”.
- “SerpienteServi7” para el caso del puerto “0777”.

Esta carpeta la compartiremos mediante drive a través del siguiente enlace: https://drive.google.com/drive/folders/1RFiDFGAZqSMkick7U38ff4LXkRNAf3gD?usp=sharing

A continuación se mostrará algunas imágenes de cómo crear los ejecutables de los servidores que tendrá el propietario del restaurante:

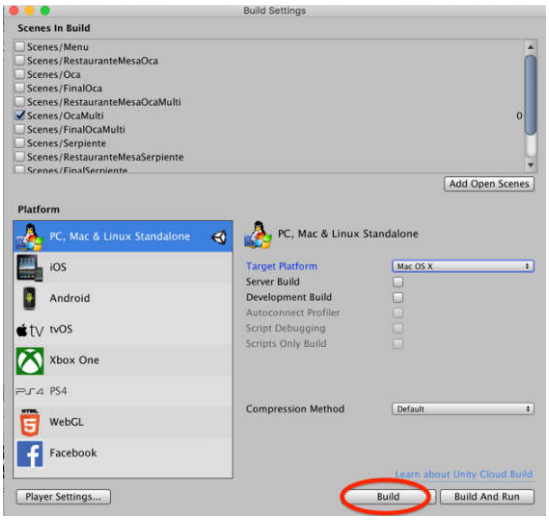


Figura A5.1: Crear la apk del servidor Oca

Cuando se pulse sobre el botón “Build”, saldrá una ventana como la que se muestra a continuación para guardar el archivo en una determinada carpeta así como elegir el nombre.

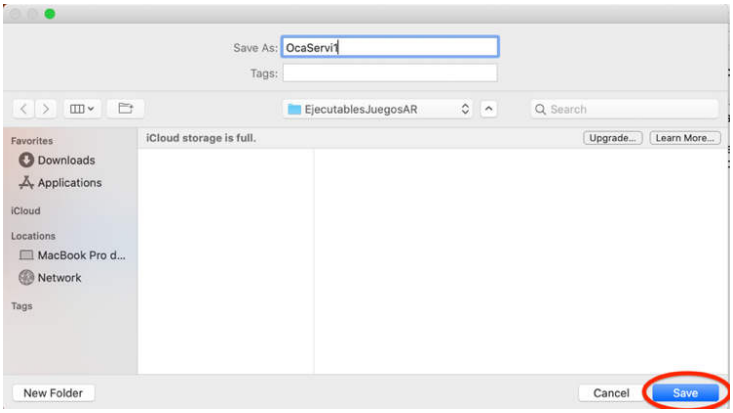


Figura A5.2: Guardar la apk del servidor Oca

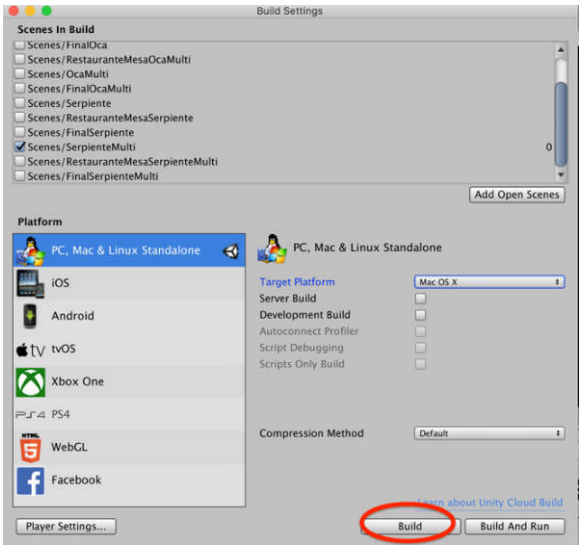


Figura A5.3: Crear la apk del servidor Serpiente

Una vez que se pulse sobre el botón marcado en rojo, saldrá una ventana como la de antes y se realiza lo mismo.

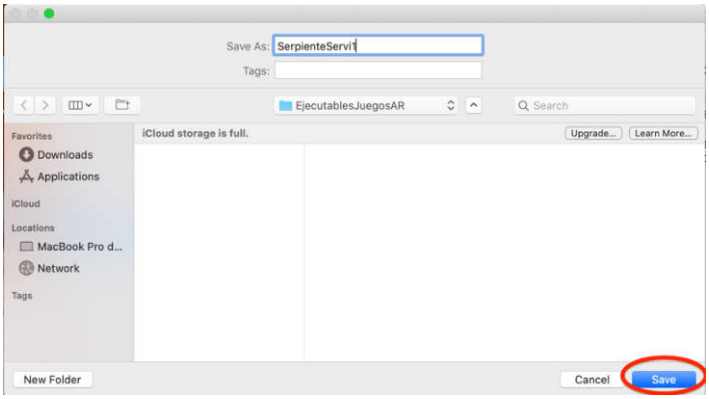


Figura A5.4: Guardar la apk del servidor Serpiente

Por último pasaremos al ejecutable que tendrá el usuario que estará formado todas las escenas, para ello seleccionaremos todas escenas que antes salía en blanco.

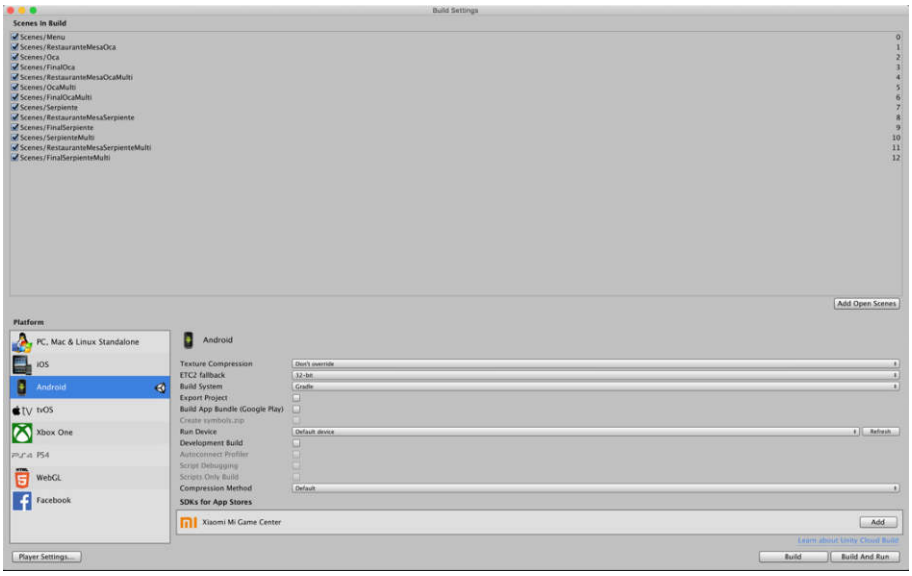


Figura A5.5: Crear la apk del usuario

Este ejecutable que está formado por doce escena que el usuario tendrá disponible, lo guardaremos como “JuegosAR” en la carpeta mencionada anteriormente.

Para el caso de Android como se ha comentado en el anterior anexo estará la “.apk” en drive y a través de un código QR se enlazará a esa carpeta. Dada que la plataforma de Android no hace falta tener una cuenta de desarrollador para distribuir nuestra aplicación, se realizará a través de drive y con el código QR lo llevará hacia esa carpeta para que el usuario final lo pueda descargar sin ningún problema ya que Android no tiene ningún problema de seguridad a la hora de instalar aplicaciones desde un desarrollador externo como presenta iOS.

Para el caso de la plataforma de iOS es necesario subir la extensión “.ipa” a la AppStore para que el usuario lo pueda instalar, de otro modo seria imposible por el tema de seguridad que presenta esta plataforma, ya que no permite instalar aplicaciones de desarrolladores externos. También como se ha comentado en el anterior anexo de forma breve, en el caso de que se subiera la aplicación se realizaría de la misma manera que Android, a través de un código QR nos llevaría a la AppStore donde se encontrará almacenada esta aplicación para que simplemente no se tenga que buscar y lo pueda descargar fácilmente.

A continuación se mostrara algunos de los pasos que se debe de seguir para el caso de subir la aplicación a esta plataforma, aunque en este proyecto esta finalidad no se ha llevado a cabo debido a que Apple presenta un proceso de revisión de la aplicación.

Lo primero que hay que hacer es generar el ejecutable para iOS, para ello en Unity en la ventana de “Build Setting”, se selecciona la plataforma “iOS”, y se generará el archivo

y se guardará en la carpeta “EjecutablesJuegosAR”. Para el caso de la plataforma de iOS, se genera una carpeta, tal y como se puede ver en la siguiente imagen.

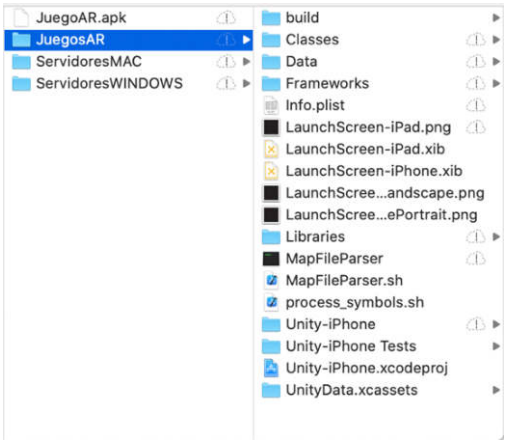


Figura A5.6: Componentes del archivo iOS

Dentro de esa carpeta, se abrirá en Xcode el archivo “Unity-iPhone.xcodeproj”, donde se realizará la firma de la aplicación, así como si se quiere cambiar cualquier configuración como por ejemplo el título de la aplicación, icono, versión, la orientación que se desea en la aplicación, a partir de que versión de iOS se puede utilizar, así como otro cambios que se pueden realizar. Si estos cambios se ha realizado en Unity, permanecerá estos en Xcode.

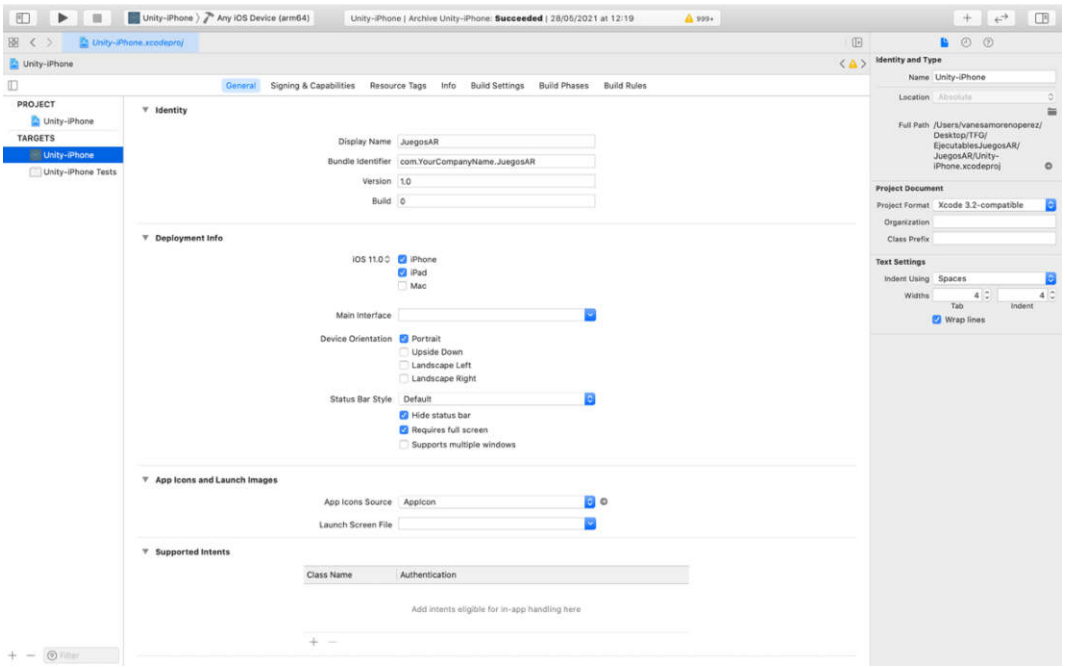


Figura A5.7: Vista General de Xcode

Para seguir, lo siguiente que se debe de hacer es firmar el archivo, para ello se va a la opción “Signing & Capabilities”. A continuación se muestra una imagen.

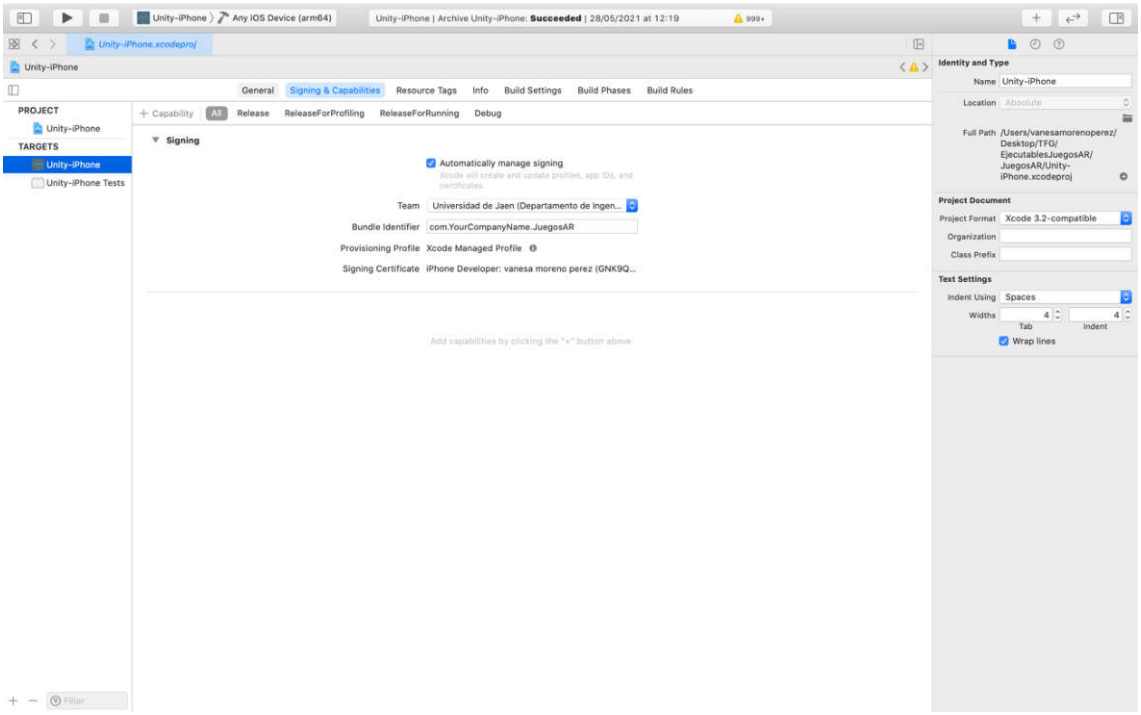


Figura A5.8: Firma de la aplicación

Se debe de seleccionar la opción de firma automática para evitar que sea menos lioso y ahorrar tiempo, así como seleccionar el equipo. Si esto no se realiza, no se puede pasar al siguiente paso.

El paso siguiente, es seleccionar la ventana de “Product” y elegir la opción “Archive”, de forma que Xcode va a empaquetar la app para así subirla a la tienda. Una vez que el proceso finaliza, saldrá la siguiente pestaña.

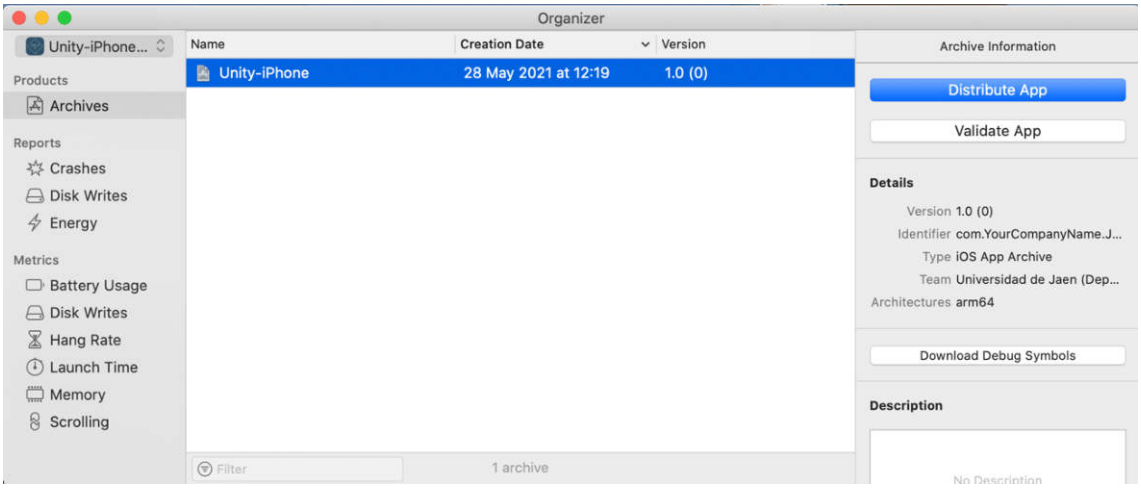


Figura A5.9: Distribuir App

Se pulsa sobre la opción “Distribute App”, y se seleccionará el método de distribucción “App Store Connect”.

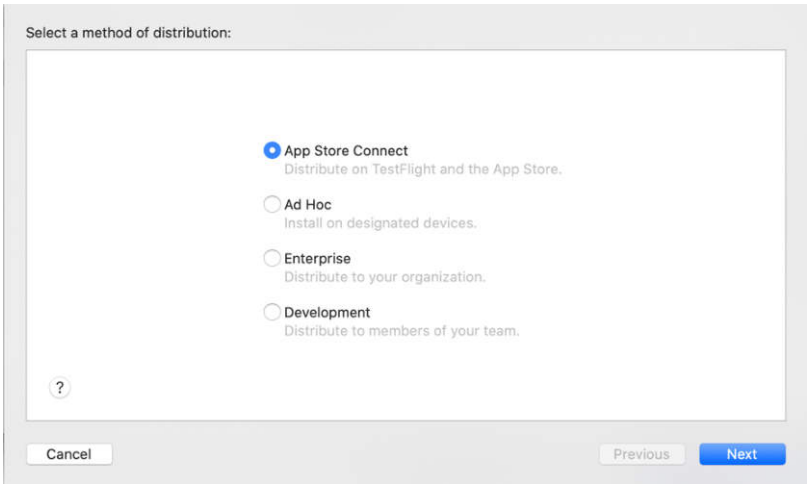


Figura A5.10: Selección del método de distribución

Por consiguiente, saldrá otra pestaña para seleccionar el destino y será la opción "Upload" que enviar la aplicación a la "App Store Connect". Una vez que se ha seleccionado esa opción, saldrá esta ventana y se seleccionará ambas opciones:

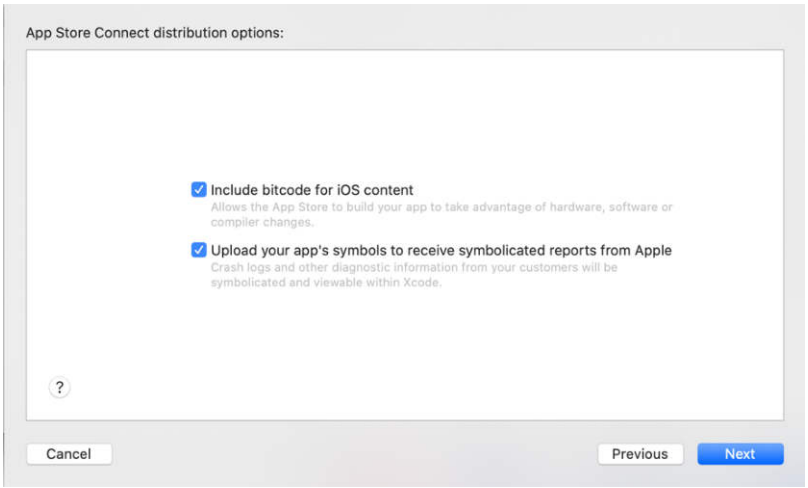


Figura A5.11: Selección de las opciones de distribución

Por último, saldrá una ventana para seleccionar si se firma automática o manualmente, en este caso tal y como se marco con anterioridad se vuelve a seleccionar firma automática, y empezará a generarse el fichero ".ipa" y el proceso en Xcode se da por finalizado.

Lo siguiente y último para mandar la aplicación a la "AppStore", será irnos a la pagina web: https://appstoreconnect.apple.com/login, y registrar nuestra app con la cuenta de desarrollador.

ANEXO 6: CÓDIGOS QR

Se mostrará el código QR que tendrá cada mesa, como se comento en el capitulo 4, cada código activará un botón con su mesa correspondiente, de manera que sea fácil para el usuario identificar cual es su mesa en la que esta sentado.

- Mesa 1:

Figura A6.1: QR mesa 1

- Mesa 2:

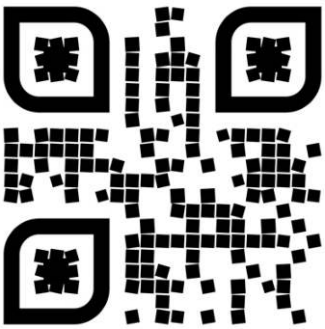


Figura A6.2: QR mesa 2

- Mesa 3:

Figura A6.3: QR mesa 3

- Mesa 4:

Figura A6.4: QR mesa 4

- Mesa 5:

Figura A6.5: QR mesa 5

- Mesa 6:

Figura A6.6: QR mesa 6

- Mesa 7:

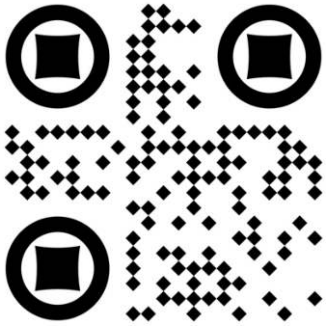


Figura A6.7: QR mesa 7

Se muestra a continuación una imagen, donde se puede ver el código QR en una mesa:

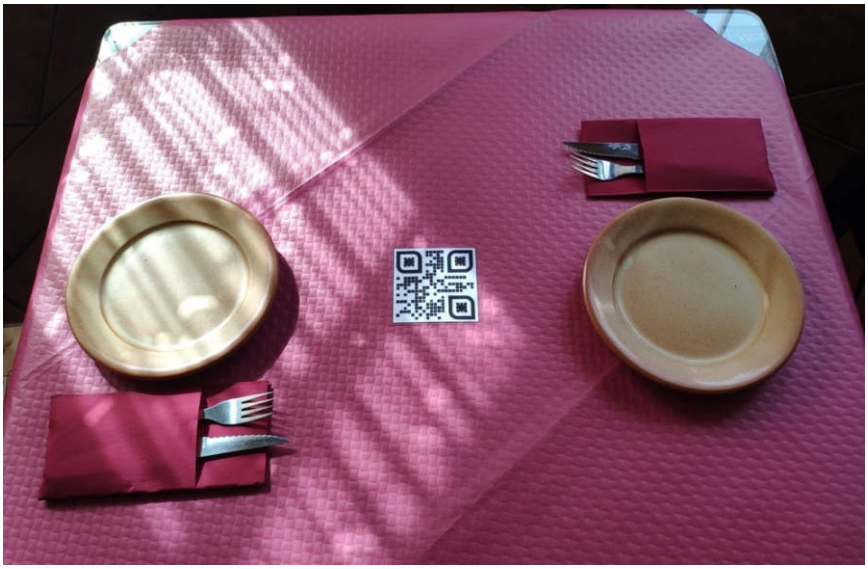


Figura A6. 8: Vista del QR en una mesa

ANEXO 7: SINCRONIZACIÓN DE MENSAJES

Para el juego de la oca las funciones sincronizadas son las que aparece a continuación:

```
[ClientRpc]
public void RpcTextoJugador()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = true;

    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = false;

    if (!isServer) //Se hace para que no salga en el servidor
    {
        NameColores();
        GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().text = "Tira el Jugador " +
TextColor;
    }

}

[Command]
public void CmdTextoJugador()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los
clientes
{
    RpcTextoJugador();
}

[ClientRpc]
public void RpcTerminaJugador()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
}


[Command]
public void CmdTerminaJugador()
{
    RpcTerminaJugador();
}

[ClientRpc]
public void RpcTextoOca()
{
    GameObject.Find("TextoOcaPuente").GetComponentInChildren<TextMeshProUGUI>().enabled = true;
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;

    if (!isServer)//Se hace para que no salga en el servidor
    {
        GameObject.Find("TextoOcaPuente").GetComponentInChildren<TextMeshProUGUI>().text = "De Oca en Oca y tiro
porque me toca";
    }
}
```

```
[Command]
public void CmdTextoOca()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los
clientes
{
    RpcTextoOca();
}

[ClientRpc]
public void RpcTextoPuente()
{
    GameObject.Find("TextoOcaPuente").GetComponentInChildren<TextMeshProUGUI>().enabled = true;
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;

    if (!isServer)//Se hace para que no salga en el servidor
    {
        GameObject.Find("TextoOcaPuente").GetComponentInChildren<TextMeshProUGUI>().text = "De Puente en Puente y
tiro porque me lleva la corriente";
    }
}

[Command]
public void CmdTextoPuente()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los
clientes
{
    RpcTextoPuente();
}

[ClientRpc]

public void RpcTerminaOcaPuente()
{
    GameObject.Find("TextoOcaPuente").GetComponentInChildren<TextMeshProUGUI>().enabled = false;

}

[Command]
public void CmdTerminaOcaPuente()
{
    RpcTerminaOcaPuente();
}


[ClientRpc]
public void RpcTextoDados()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = true;

    if (!isServer)//Se hace para que no salga en el servidor
    {

        GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().text = "De Dado a Dado y
tiro porque son cuadrados";

    }
}
```

```
[Command]
public void CmdTextoDados()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los clientes
{
    RpcTextoDados();
}


[ClientRpc]
public void RpcTextoInicial()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = true;

    if (!isServer)//Se hace para que no salga en el servidor
    {
        NameColores();

        GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().text = "Vuelve a la casilla de salida el Jugador " + TextColor;

    }
}


[Command]
public void CmdTextoInicial()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los clientes
{
    RpcTextoInicial();
}


[ClientRpc]
public void RpcTextoRetroceder()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = true;

    if (!isServer)//Se hace para que no salga en el servidor
    {
        NameColores();

        GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().text = "Retrocede a la casilla 30 el \n Jugador " + TextColor;

    }
}


[Command]
public void CmdTextoRetroceder()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los clientes
{
    RpcTextoRetroceder();
}
```

```
[ClientRpc]
public void RpcTextoPierde3Turno()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = true;
    if (!isServer)
    {
        NameColores();
        GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().text = "Pierde 3 turnos
el Jugador " + TextColor;
    }

}
[Command]
public void CmdTextoPierde3Turno()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos
los clientes
{
    RpcTextoPierde3Turno();
}


[ClientRpc]
public void RpcTextoPierde2Turno()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = true;
    if (!isServer)
    {
        NameColores();
        GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().text = "Pierde 2 turnos
el Jugador " + TextColor;
    }

}
[Command]
public void CmdTextoPierde2Turno()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos
los clientes
{
    RpcTextoPierde2Turno();
}

[ClientRpc]
public void RpcTextoPierde1Turno()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = true;
    if (!isServer)
    {
        NameColores();
        GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().text = "Pierde 1 turno
el Jugador " + TextColor;
    }

}
```

```
[Command]
public void CmdTextoPierde1Turno()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos
los clientes
{
    RpcTextoPierde1Turno();
}

[ClientRpc]
public void RpcTextoPierde0Turno()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = true;
    if (!isServer)
    {
        NameColores();
        GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().text = "Vuelve a jugar a
la proxima el Jugador " + TextColor;
    }

}



[Command]
public void CmdTextoPierde0Turno()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos
los clientes
{
    RpcTextoPierde0Turno();
}

public void TextoPerderTurnos()
{

    if (listaPerderTurno[CurrentPlayer] == 3)
    {
        CmdTextoPierde3Turno();
        CmdTerminaCasillaEspecial();
    }
    if (listaPerderTurno[CurrentPlayer] == 2)
    {
        CmdTextoPierde2Turno();
        CmdTerminaCasillaEspecial();
    }
    if (listaPerderTurno[CurrentPlayer] == 1)
    {
        CmdTextoPierde1Turno();
        CmdTerminaCasillaEspecial();
    }
    else if (listaPerderTurno[CurrentPlayer] == 0)
    {
        CmdTextoPierde0Turno();
        CmdTerminaCasillaEspecial();
    }


}
```

```
[ClientRpc]
public void RpcTextoAvanza26()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = true;


    if (!isServer)//Se hace para que no salga en el servidor
    {
        NameColores();
        GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().text = "Avanza a la
casilla 26 el Jugador " + TextColor;

    }
}


[Command]
public void CmdTextoAvanza26()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los
clientes
{
    RpcTextoAvanza26();
}

[ClientRpc]
public void RpcTextoAvanza53()
{

    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = true;

    if (!isServer)//Se hace para que no salga en el servidor
    {
        NameColores();
        GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().text = "Avanza a la
casilla 53 el Jugador " + TextColor;
    }
}


[Command]
public void CmdTextoAvanza53()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los
clientes
{
    RpcTextoAvanza53();
}

IEnumerator TerminaCasillaEspecial() //Se hace con la intencion de que el cliente que pulse el boton al caer en
texto directo se desactive con unos segundos
{
    yield return new WaitForSeconds(1.5f);
    GameObject.Find("TextoCasillaEspecial").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
}
```

```
[ClientRpc]
public void RpcTerminaCasillaEspecial()
{
    StartCoroutine(TerminaCasillaEspecial());
}


[Command]
public void CmdTerminaCasillaEspecial()
{
    RpcTerminaCasillaEspecial();
}


[ClientRpc]
public void RpcTextoGanador()
{

    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoGanador").GetComponentInChildren<TextMeshProUGUI>().enabled = true;

    if (!isServer)//Se hace para que no salga en el servidor
    {
        NameColores();
        GameObject.Find("TextoGanador").GetComponentInChildren<TextMeshProUGUI>().text= "Ha ganado el Jugador " +
TextColor;

    }
}

[Command]
public void CmdTextoGanador()//Cuando el cliente entra en la casilla final se lo envia a todos los clientes
{
    RpcTextoGanador();
}
```

Por consiguiente, se muestras las funciones sincronizadas para el juego de la serpiente y escalera:

```
[ClientRpc]
public void RpcTextoJugador()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = true;

    if (!isServer) //Se hace para que no salga en el servidor
    {
        NameColores();
        GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().text = "Tira el jugador " +
TextColor;

    }
}
```

```
[Command]
public void CmdTextoJugador()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los
clientes
{
    RpcTextoJugador();
}


[ClientRpc]
public void RpcTextoJugadorTermina()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;

}


[Command]
public void CmdTextoJugadorTermina()
{
    RpcTextoJugadorTermina();
}


[ClientRpc]
public void RpcTextoEscalera()
{
    GameObject.Find("TextoEscalera").GetComponentInChildren<TextMeshProUGUI>().enabled = true;

    if (!isServer)//Se hace para que no salga en el servidor
    {
        GameObject.Find("TextoEscalera").GetComponentInChildren<TextMeshProUGUI>().text = "Subo por la escalera";

    }
}


[Command]
public void CmdTextoEscalera()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los
clientes
{
    RpcTextoEscalera();
}


[ClientRpc]
public void RpcTextoEscaleraTermina()
{

    GameObject.Find("TextoEscalera").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
}

[Command]
public void CmdTextoEscaleraTermina()
{
    RpcTextoEscaleraTermina();
}
```

```
[ClientRpc]
public void RpcTextoSerpiente()
{
    GameObject.Find("TextoSerpiente").GetComponentInChildren<TextMeshProUGUI>().enabled = true;


    if (!isServer)//Se hace para que no salga en el servidor
    {
        GameObject.Find("TextoSerpiente").GetComponentInChildren<TextMeshProUGUI>().text = "Bajo a la cola de la
serpiente";

    }
}


[Command]
public void CmdTextoSerpiente()//Cuando el cliente pulse el boton automaticamente el servidor lo envia a todos los
clientes
{
    RpcTextoSerpiente();
}

[ClientRpc]
public void RpcTextoSerpienteTermina()
{
    GameObject.Find("TextoSerpiente").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
}

[Command]
public void CmdTextoSerpienteTermina()
{
    RpcTextoSerpienteTermina();
}


[ClientRpc]
public void RpcTextoGanador()
{
    GameObject.Find("TextoJugador").GetComponentInChildren<TextMeshProUGUI>().enabled = false;
    GameObject.Find("TextoGanador").GetComponentInChildren<TextMeshProUGUI>().enabled = true;

    if (!isServer)//Se hace para que no salga en el servidor
    {
        NameColores();
        GameObject.Find("TextoGanador").GetComponentInChildren<TextMeshProUGUI>().text = "Ha ganado el Jugador " +
TextColor;

    }
}

[Command]
public void CmdTextoGanador()//Cuando el cliente entra en la casilla final se lo envia a todos los clientes
{
    RpcTextoGanador();
}
```