Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Implementación de un recurso docente como videojuego sobre HTML desarrollado con Unity

Autor: Carlos Martínez García

Tutor: Antonio Jesús Sierra Collado

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Carrera

Ingeniería de Telecomunicación

Implementación de un recurso docente como videojuego sobre HTML desarrollado con Unity

Autor:
Carlos Martínez García

Tutor:
Antonio Jesús Sierra Collado
Profesor titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Carrera: Implementación de un recurso docente como videojuego sobre HTML desarrollado con Unity

Autor: Carlos Martínez García

Tutor: Antonio Jesús Sierra Collado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

Agradecimientos

A mi familia

A mis maestros

Resumen

Este proyecto consiste en el desarrollo de una aplicación web didáctica enfocada a personas que quieran aprender los pilares básicos de un lenguaje de programación, bien porque estén estudiándolos desde cero o porque quieran repasar estos contenidos.

El proyecto es la continuación de dos proyectos realizados en años anteriores, primero uno en Android, y después otro en IOS, y en este se dará el salto a una aplicación web añadiendo diferentes formas de seguir aprendiendo a la vez que divirtiéndose.

En la página Web se podrá aprender la teoría de lenguajes como Java, C, HTML y CSS, y a través del videojuego se podrá aprender sobre Java mediante escritura de código. El videojuego es del tipo de rol (RPG, role-playing game) con un modo historia, es decir, un juego en el cual los niveles están unidos por una historia común que se va descubriendo y profundizando a medida que se va superando a través de los personajes.

El juego es multiplataforma, inicialmente con una versión de escritorio Web y una versión de escritorio, disponible para Windows en sus arquitecturas de 32 y 64 bits. La versión de escritorio cuenta con un sistema de guardado para que el juego pueda ser completado en diferentes momentos. Esta será descargable desde la página Web.

En este trabajo se ha profundizado en la experiencia del videojuego, dándole gran importancia a la historia y la jugabilidad, así como mapas complejos y extensos o efectos visuales y de sonido. La página web sirve como herramienta de soporte y del videojuego, y completa al juego con manuales para lenguajes de programación y herramientas externas del estilo de nuestro videojuego donde aprenderás jugando.

Como futuro del proyecto se propone la inclusión de cuestionarios y juegos de tipo trivial en la página Web, así como más lenguajes. En cuanto al videojuego, se creará un nuevo conjunto de mapas ambientado en una mazmorra subterránea alargando por tanto la historia y modificando el guion del mismo para conocer más sobre la historia.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIÓNES

1 OBJETIVOS

El gran objetivo del aprendizaje no es el conocimiento, sino la acción
-Herbert Spencer-

El objetivo de este proyecto es el desarrollo de una aplicación web junto con un videojuego que permita el aprendizaje básico de distintos lenguajes de programación. La web tiene intención didáctica, sirviéndose del juego para hacer que este aprendizaje sea divertido.

Para el lenguaje de C se aprenderán elementos como:

- Funciones y variables
- Estructuras de control
- Punteros

Por otro lado, para el lenguaje Java se aprenderán los siguientes elementos:

- Clases
- Objetos, métodos y atributos
- Funciones y variables
- Estructuras de control

Además de esto, se podrán aprender nociones básicas de HTML y CSS desde el apartado de lecciones de la página Web. Y de cara a futuros trabajos, se propone la inclusión de cuestionarios para cada uno de los lenguajes. Se proporcionan también recursos externos como páginas con tutoriales de programación, o páginas con videojuegos didácticos en la línea del que se ha diseñado.

2 INTRODUCCIÓN

Vive como si fueses a morir mañana. Aprende como si fueses a vivir para siempre . -Mahatma Gandhi –

En este apartado veremos una introducción a la historia de los videojuegos, centrándonos en los videojuegos didácticos y diferentes herramientas que se han incluido hasta en las aulas. Además, veremos algunos de los motores gráficos más sonados existentes en el mercado, y de igual manera una comparativa de los softwares de escritura de guion.

2.1 Videojuegos didácticos

La historia de los videojuegos comienza en torno a los años 50, tras la aparición de las primeras computadoras al término de la Segunda Guerra Mundial. Prototipos que podemos considerar como primeros videojuegos son: *¡Tennis For Two (1958) y Spacewar!* (1962). Estos juegos eran apenas accesibles ya que las computadoras solo se localizaban en universidades o estudios de investigación.

En el caso del juego Tennis For Two, se consiguió proyectar en un osciloscopio que se usaba como monitor. Era un juego muy simple de tenis, en el que se usaba un pulsador, y un mando analógico para elegir la dirección a la que devolvías la pelota de tenis.

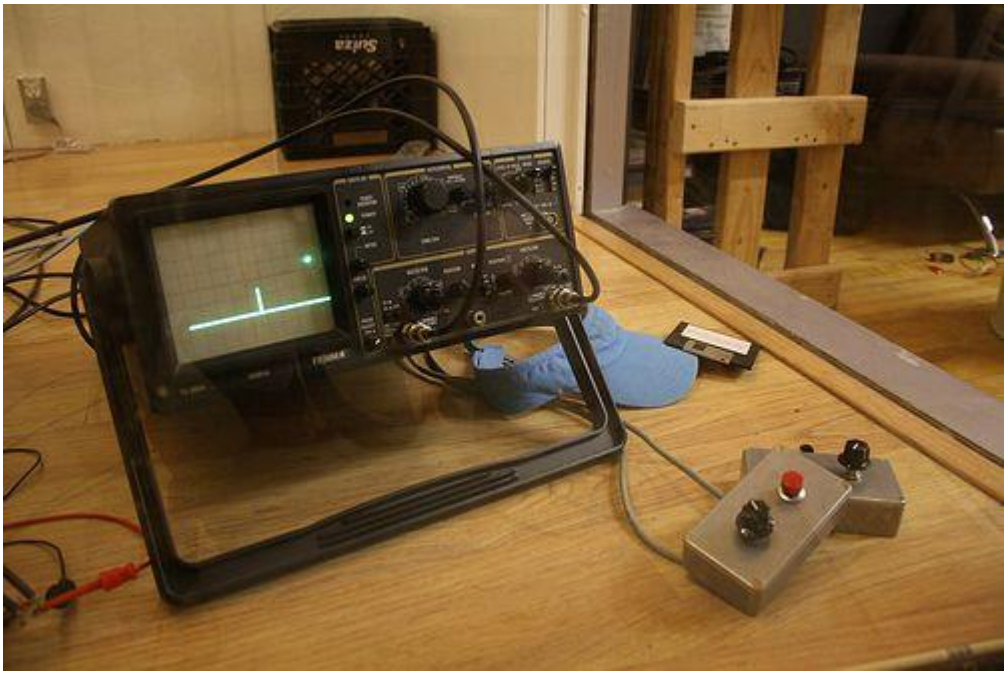

Ilustración 1. Imagen del juego Tennis for Two proyectado en osciloscopio

Hoy en día, los videojuegos han evolucionado mucho desde sus inicios y ahora encontramos que existen videojuegos de cualquier tipo: acción, plataformas, aventuras, de acción en primera persona, multijugador (online) y hasta juegos de carácter educativo. Vamos a centrarnos en estos y en hacer un repaso por las diferentes herramientas que nos van a permitir desarrollarlos y las herramientas para escrituras de guion. Los guiones de los videojuegos han pasado a ser considerados por gran parte de la audiencia de éstos, como una de las partes

más importantes. En algunos casos, estos videojuegos son considerados casi peliculas de mayor extensión, donde se permite al espectador formar parte de la misma.

2.2 Videojuegos didácticos

Los videojuegos didácticos son en realidad una técnica de aprendizaje a través de la diversión cuyo fin es el aprendizaje desde una forma lúdica. Estos videojuegos didácticos captan la atención del usuario, el cual aprende con una mayor motivación y desarrollando capacidades de una manera divertida.

Los videojuegos didácticos están ganando mucha fuerza en la enseñanza con menores, etapa en la que es mayor el aprendizaje. Y en aulas de primaria se aprovechan clases de informática para enseñar a los niños como usar el ordenador y aprender a través de videojuegos en los que deben por ejemplo completar un mapa de España, o aprender inglés rellenando huecos con palabras

Pero no solo están pensados para los niños, ya que cada vez tenemos una gama más amplia de videojuegos de este tipo, permitiendo llegar al público más adulto, permitiendo aprender ciertas áreas, como puede ser un idioma, un lenguaje de programación, o incluso para entrenar diariamente la mente.

Como ejemplos de videojuegos tenemos Minecraft, que es un juego que ayuda al desarrollo de la creatividad del usuario, Civilization para aprender historia, o juegos para el entrenamiento mental diario como los conocidos juegos de Brain Training o Profesor Layton que sumergen al usuario en una historia misteriosa en la que deben avanzar mediante la resolución de puzles y rompecabezas de todo tipo.

Ilustración 2. Imagen de una de las pruebas del juego Brain Training

2.3 Enseñanza en aulas TIC

En los últimos años, la tecnología se ha ido adentrando en la docencia más y más desde bastante temprano. Hoy día no es raro que desde pequeños se usen las tecnologías (los ordenadores, y los teléfonos en casos de edades más adultas) lo cual suele crear controversia sobre si es la mejor forma de enseñanza, pero siempre que use bien la tecnología siempre nos va a traer beneficios, pero no puede ser el único método de enseñanza si no que debe ser un complemento a los métodos más tradicionales.

Por eso se deben entender los videojuegos como una potente herramienta para el desarrollo de capacidades y cualidades del alumno desde tempranas edades. En los últimos años se han realizado numerosos estudios que demuestran los beneficios de la gamificación y la introducción de tecnologías en la educación.

Desde hace bastantes años se usan los ordenadores desde primaria para enseñar nociones básicas de informática y profundizar en conocimientos de geografía, inglés, etc. En cuanto a videojuegos que se han creado un hueco en las aulas tenemos MinecraftEDU, que es una versión del conocido y ya mencionado juego Minecraft. En esta versión los alumnos pueden cooperar y resolver diferentes problemas que irá introduciendo el profesor con una intención didáctica concreta. Los alumnos pueden desarrollar la creatividad mediante la creación de edificios y diferentes construcciones con bloques tipo lego.

Ilustración 3. MinecraftEDU en las aulas

2.4 Herramientas existentes para el diseño de videojuegos

Visto que los videojuegos educativos están en auge y tienen una gran utilidad para el aprendizaje, interesa descubrir ahora que herramientas nos pueden permitir realizar nuestro propio videojuego. Para ello necesitaremos un motor gráfico.

Los motores gráficos son herramientas software que ejecutan un tipo de tareas comunes a otras muchas aplicaciones software. Más concretamente, se puede definir el motor de videojuegos como el framework de software diseñado para crear y desarrollar videojuegos.

Se puede situar el origen del término 'motor grafico', ó 'Game Engine' en inglés, a partir de 1993 tras la aparición del revolucionario juego en primera persona *Doom.* En la realización de este juego se separaron los componentes tales como el renderizado gráfico, las colisiones, el audio y algunos componentes más. La ventaja de esto es la reutilización en juegos posteriores como Quake y Unreal.

Ilustración 4. Imagen del juego Doom (1993)

Los motores gráficos ofrecen al programador un motor de renderizado para los gráficos, el motor de colisiones, y las interacciones entre dichas colisiones y entidades del juego, así como música, animaciones y la inteligencia artificial del juego.

Hay muchas herramientas existentes en el mercado, muchas detrás de empresas, marcas y juegos bastante conocidos, vamos a mencionar algunas de las más importantes como son:

- **Unreal Engine:** De los motores más antiguos y más conocidos mundialmente, perteneciente a la compañía Epic Games, la cual ha ganado mucho valor en los últimos años con lanzamientos como Fortnite que ha revolucionado la jugabilidad multijugador online con su modo de juego "battle royale". Lleva el nombre de uno de los juegos más famosos desarrollados por la compañía que es el Unreal (o Unreal Tournament). Gratuito y con alta compatibilidad.

- **Source:** Motor desarrollado por la empresa Valve Corporation en 2004. No tan compatible como Unreal Engine y no tan usado, pero es gratuito al igual que el de Unreal, siendo este de Valve más accesible y sencillo de usar.

- **CryEngine:** Motor muy potente y versátil, pero con una curva de aprendizaje mucho más larga que las alternativas que se comentan aquí. Es el único completamente de pago de los mencionados y destaca por su extrema calidad gráfica.

- **UbiArt:** Motor creado por Ubisoft y utilizado en su saga de juegos *Rayman.* Caracterizado por su poca versatilidad ya que solo nos permite crear juegos de plataformas en 2D o 2,5D (paso intermedio entre 2D y 3D. Es un tipo de 2D con efectos que hacen parecer que estamos en 3D). Su licencia parece casi imposible de conseguir a no ser que seas parte de Ubisoft, y es demasiado complejo para desarrolladores nuevos.

- **Unity:** Motor de Unity Technologies lanzado en 2005. Se caracteriza por su gran compatibilidad y portabilidad ya que se puede exportar a muchas plataformas. Además, cuenta con una tienda de Assets con la que proporciona muchas herramientas listas para descargar y usar desde el mismo editor.

2.4.1 Unreal Engine

Unreal Engine es el motor gráfico de Epic Games, el cual nació en 1998 con el shooter en primera persona (FPS) *Unreal*. Desde entonces ha ido evolucionando considerablemente y es uno de los motores más sonados y más importanes sobre todo en el desarrollo de juegos de tipo FPS.

Ilustración 5. Imagen del juego Unreal (1998)

En la actualidad tenemos la versión de cuarta generación Unreal Engine 4, la cual se lanzó en 2014 y es gratuita desde 2015. Nos permite desarrollar nuestro videojuego sin tener que pagar mensualidades siempre que no obtengamos un beneficio superior a 3000$ por producto y trimestre.

Destaca por su versatilidad y compatibilidad, permitiéndonos desarrollar juegos simples en 2D hasta videojuegos complejos triple A. Es compatible con PC, PS4, Xbox One, MAC, iOS y Android.

A día de hoy tenemos juegos muy populares creados con este motor como son *Kingdom Hearts III*, *Days Gone, Life is Strange* o *Batman: Arkham City*, entre otros.

Ilustración 6. Imagen dentro del juego Days Gone (2019)

2.4.2 Source

Source es un motor de videojuegos 3D que sucede a partir de 2004 a GoldSource, motor con el que se desarrollaron juegos famosos *como Half Life, Team Fortress y Counter Strike 1.6* a finales de los 90.

La versión más actual y que se usa a día de hoy es Source Engine 2 lanzada en 2015. Entre sus ventajas destaca la gran cantidad de herramientas que ofrece, potencia y su sencillez, siendo gratuito, además. Si pretendes vender los productos y obtener beneficio de ello, si que tendrás que pagar una licencia de alrededor de 25.000$. Como otro punto negativo a añadir es que no goza de tantas actualizaciones y es un motor algo más "viejo" que los de la competencia.

Se han desarrollado con esta herramienta muchos juegos sobre todo de la plataforma de videojuegos de *Steam*. Podemos nombrar dos grandes juegos multijugador online y con carácter competitivo como son *Dota 2 y Counter Strike Global Offensive.*

Ilustración 7. Imagen del juego Counter Strike 1.6 (1996)

Ilustración 8. Imagen de Counter Strike Global Offensive (2012)

En la anterior imagen podemos ver una comparativa en el mismo mapa, entre las versiones de *Counter Strike 1.6* (diseñado con GoldSource en 1996) y *Counter Strike Global Offensive* (con Source Engine 2 en 2012).

2.4.3 CryEngine

CryEngine fue introducido al mercado de motores de videojuegos por la empresa Crytek Studios en 2004 con el lanzamiento del famoso juego *Far Cry*.

CryEngine es sin duda uno de los motores más potentes con enormes efectos de iluminación y realismo y que prácticamente solo Unreal Engine puede competir con el en este aspecto. Sin embargo, es obligatorio pagar 9.99$ al mes para poder empezar a desarrollar y está más enfocado para juegos de consolas, aunque también compatible con PC.

Por tanto, este motor nos va a permitir crear con facilidad escenarios con una potencia gráfica, pero nos costará llevarlo más allá de juegos estilo shooter. Eso sí, además de por el precio, no es recomendado para principiantes ya que se necesitará de muchas horas para tener cierto control sobre el software.

Entre sus títulos más famosos tenemos *Monster Hunter Online, Sniper Ghost Warrior 1 y 2*, y la saga *Crysis.*

Ilustración 9. Imagen del juego Sniper Ghost Warrior (2010)

2.4.4 UbiArt

UbiArt es un motor de juego desarrollado por Ubisoft, pensado para juegos de plataformas 2D (o 2,5D) de animación, donde no prima el realismo si no la diversión y la jugabilidad.

Ese es uno de los inconvenientes, ya que solo puede usarse para juegos de plataformas, es un framework bastante cerrado. Tampoco es sencillo conseguir la licencia por parte de Ubisoft, y además desde la propia empresa de Ubisoft reconocen que esta herramienta no es tan usada porque a pesar de conseguir resultados impresionantes, no es para nada fácil de utilizar.

Entre sus lanzamientos se encuentran los éxitos de la famosa saga de videojuegos Rayman, con títulos en todas las consolas y con compatibilidad en iOS y Android en los últimos años.

Ilustración 10. Imagen del juego Rayman Legends (2013)

2.4.5 Unity

Unity es uno de los motores más usados y esto es debido a su gran potencia, versatilidad, compatibilidad y sencillez. Desde la versión 5, se eliminan las restricciones que tiene la licencia gratuita (en el Anexo se explican las diferentes tarifas y licencias que ofrece Unity).

Inicialmente, Unitiy estuvo asociada a los videojuegos para móvil, pero con las últimas versiones, se han desarrollado grandes éxitos para PC como son *Cuphead, Inside, Rust y Hearthstone.*

Comparándolo con Unreal Engine, que es una de sus grandes competidores, Unity tiene a favor una gran comunidad con una tienda de componentes para desarrollar videojuegos que a su vez hacen que la curva de aprendizaje sea mucho más corta, es de los mejores en compatibilidad y escalabilidad, pero a su vez consume muchos recursos del PC y no llegar a la calidad gráfica de Unreal o CryEngine.

Ilustración 11. Imagen del juego Rust (2013)

Ilustración 12. Imagen del juego Cuphead (2017)

En cuanto a la compatibilidad y escalabilidad mencionada anteriormente, es un éxito Unity, ya que podemos usarlo Windows, iOS o incluso Linux. Una vez que estemos desarrollando, por ejemplo, en Windows y queremos desarrollar un juego para PC, una vez que esté terminado, desde el propio editor se nos da la opción de exportar hasta para 25 plataformas. Entre estas plataformas, además de las habituales, como son todas las consolas, PC, Web y móviles, destacan algunas curiosas como Smart TVs o dispositivos de realidad aumentada, como Oculus Rift. Esto es lo que hace a Unity, una de las mejores herramientas del mercado.

2.5 Guiones de videojuegos

El guion es una de las piezas fundamentales a la hora de desarrollar un videojuego, ya que un gran guion es lo que siempre se recordará. El guion va a definir el transcurso de la historia, y los diálogos que van a hacer que el jugador comprenda lo que está ocurriendo, empatice con los personajes y llegue incluso a emocionarse con ciertas escenas.

La principal diferencia con respecto a un guion de película es que mientras este es cerrado, el de videojuegos es mucho más abierto y debe crear situaciones que guíen al jugador desde su propia libertad dentro de la jugabilidad que se le ofrece.

En el videojuego ganan una gran importancia las misiones, que son pequeñas secuencias, a veces no obligatorias para completar el juego, donde se nos ofrece una mayor información y conexión con la historia. Como videojuegos con una gran historia, que nos sumergen como la mejor de las películas podemos destacar grandes títulos como son: *The Last of Us 1 y 2*, *Heavy Rain*, *Until Dawn* o las sagas de *Life is Strange* y *Uncharted.*

Para terminar, hay que mencionar un recurso usado en *Until Dawn* y *Life is Strange*, y que está ganando mucha popularidad en los títulos con mejores historias y es el llamado "Efecto Mariposa". Estos videojuegos tienen un guion fijo, digamos la base del guion, pero tienen a su vez diferentes ramificaciones en el guión de manera que según las decisiones que vayamos tomando a lo largo de la historia, el desenlace será distinto. Esto nos permite que podamos jugar un mismo juego, una misma historia, disfrutando de la experiencia cada vez ya que las consecuencias no serán las mismas.

Esto fue usado por Netflix, en su película *Bandersnatch* en la que podíamos vivir hasta 5 historias diferentes en una película de 1h y media. En la imagen siguiente podemos ver el diagrama de flujo completo de la película.

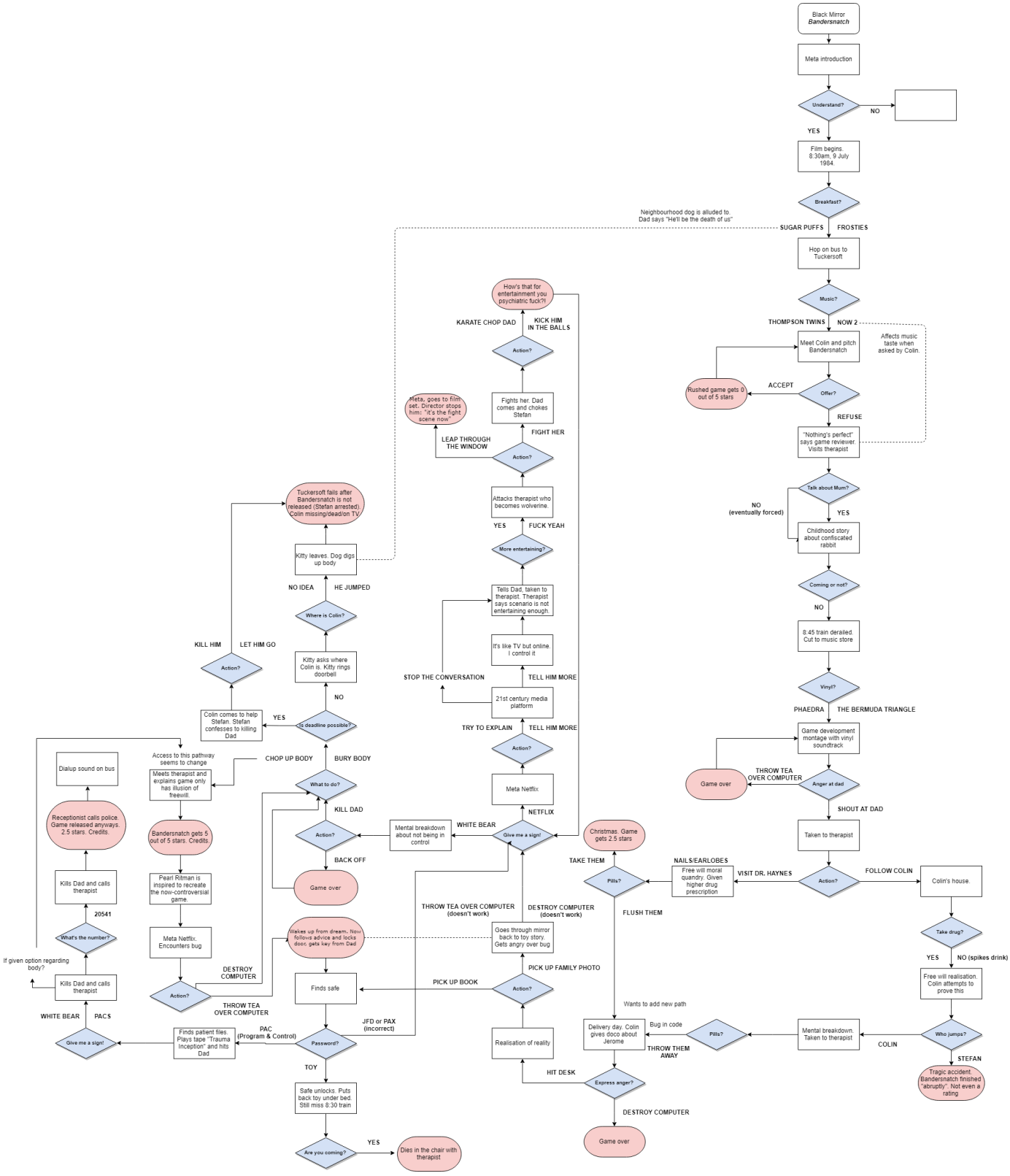

Ilustración 13. Efecto mariposa en guión de la película Bandersnatch

2.6 Herramientas para el desarrollo de guiones de videojuegos

A la hora de escribir guiones, podemos utilizar herramientas tan simples y accesibles por todos como es el caso de Microssoft Word, o cualquier simple editor de texto, pero tenemos disponible en internet gran cantidad de software específico con herramientas más concretas para la escritura de guiones y algunas ayudas como plantillas, asistentes de formato, herramientas de trabajo colaborativo y herramientas para brainstorming, entre otras.

Algunos de los softwares más importantes del mercado son:

2.6.1 Final Draft

Uno de los mejores softwares para escritura de guion sin duda Final Draft. Lanzado en el año 1990, se ha convertido a lo largo de los años en uno de los más prestigiosos. Incluso el propio Guillermo del Toro, director de cine de renombre, halaga este software y reconoce que lo ha usado durante su carrera.

Es sin duda el más caro del mercado, cuya licencia es de 250$, con una breve opción de prueba durante 30 días. Entre sus características destaca por la opción colaborativa, que permite que varios usuarios estén al mismo tiempo modificando contenidos, el speech to text, el cual permite convertir a texto lo que le vayamos narrando con la voz, y su funcionalidad *SmartType* que autocompleta los nombres basando en la memoria del programa, tabula y edita la fuente y estilo según convenga.

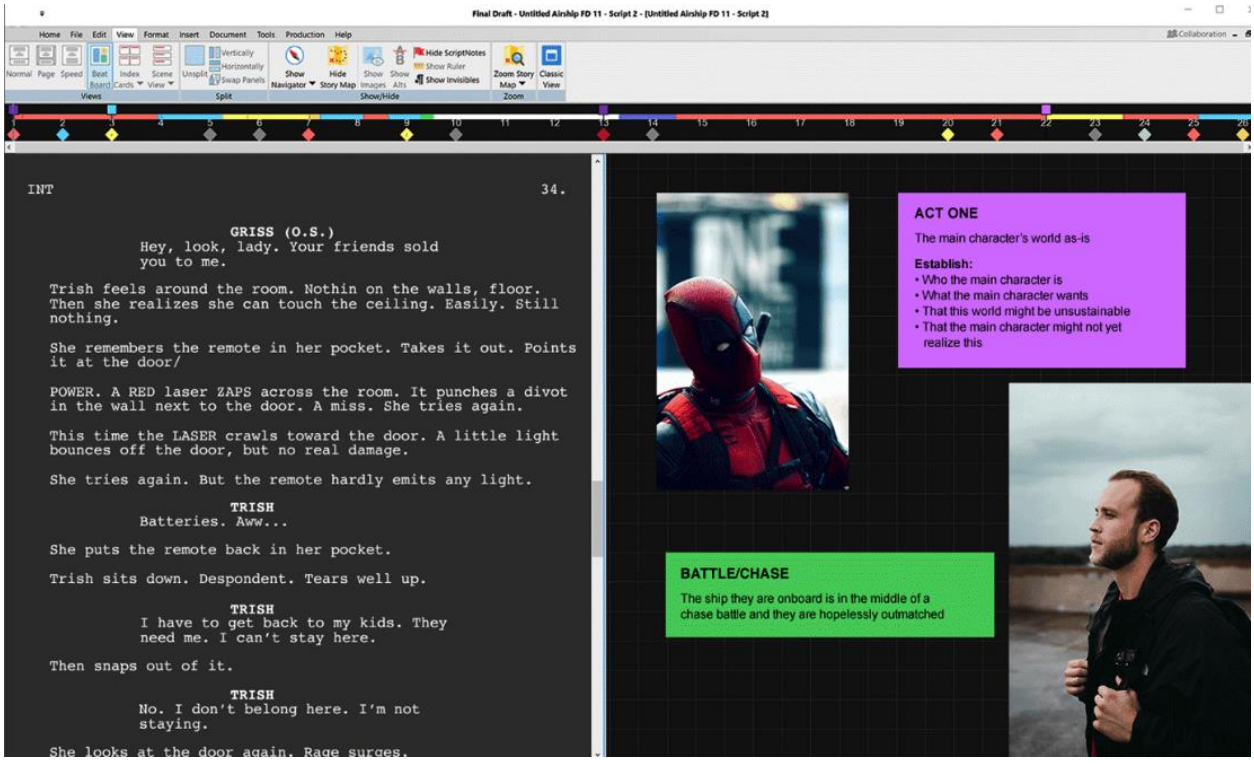

Ilustración 14. Imagen del software Final Draft en su versión 11

2.6.2 Fade In

Otro software muy utilizado a nivel profesional es Fade In. Desarrollado por GCC Productions, en concreto por Kent Tessman (director de cine). Fade In se lanzó en 2011 en versión escritorio para Windows, y desde entonces ha evolucionando considerablemente hasta la versión 3.0, que es la actual lanzada en todas las plataformas, hasta en versión móvil en iOS y Android.

Se sabe que Fade In se ha usado en la industria del cine, tanto en Hollywood como fuera de el. También se ha usado en el área del periodismo ya que nos permite hacer guiones de televisión, radio y etc.
Entre sus características, incluye funcionalidades esperadas por un programa de escritura de guion profesional

cómo es el formato de escenas, acciones de personajes y diálogos según los estándares de la industria. Incluye gestión de revisión y seguros de páginas/escenas para revisar y reeditar múltiples veces en preproducción y producción. Ha recibido muy buenas críticas a nivel internacional, y se sitúa en el top de software de escritura de guiones, a la altura de los que se comentan.

En cuanto a las licencias, también es de pago, pero con un precio mucho más bajo que Final Draft con un precio de 79.95$ en su versión de escritorio y en torno a 5.5$ en versión móvil.

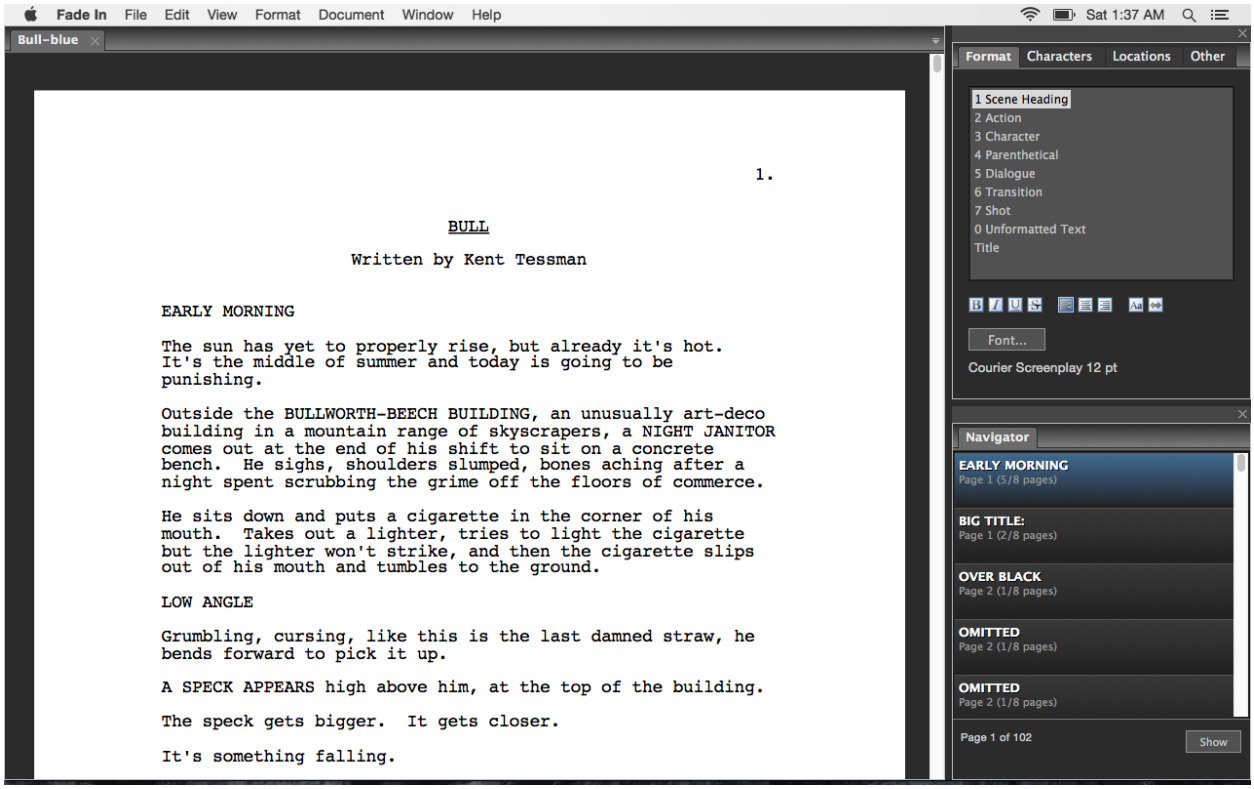

Ilustración 15. Imagen del software Fade In

Ha recibido muy buenas críticas a nivel internacional, y se sitúa en el top de software de escritura de guiones, a la altura de los que se comentan.

En cuanto a ejemplos de guiones que se sabe que se han producido con Fade In tenemos: *Puñales por la espalda, Venom y Star Wars: Rogue One* como películas y la serie de videojuegos *Bioshock.*

2.6.3 WriterDuet

WriterDuet, fundado en 2013 por Guy Goldstein inicialmente como un proyecto por diversión y ha terminado siendo un servicio muy potente. Funciona de manera diferente a los otros programas de escritura de guion ya que ofrece un servicio de pago mensual por 11.99$/mes o anual por 89$/año, con oferta permanente al 50% para estudiantes y profesores.

Es un programa muy interesante que potencia el trabajo colaborativo y usando el servicio de Google Firebase permitiendo la edición colaborativa en tiempo real. Permite la edición offline, sincronizando automáticamente cuando nos conectamos a la red. Compatible con Windows, Mac, iOS y Android y además compatible con otros programas de escritura de guiones permitiéndonos exportar el archivo para que sea accesible por el otro programa o en forma de PDF.

Se usa mayoritariamente para realizar bocetos, escribir y formatear guiones de acuerdo al estándar recomendado por la Academia de Artes y las Ciencias Cinematográficas (AMPAS) pero también es válido para formatos de novelas, teatro y videojuegos.

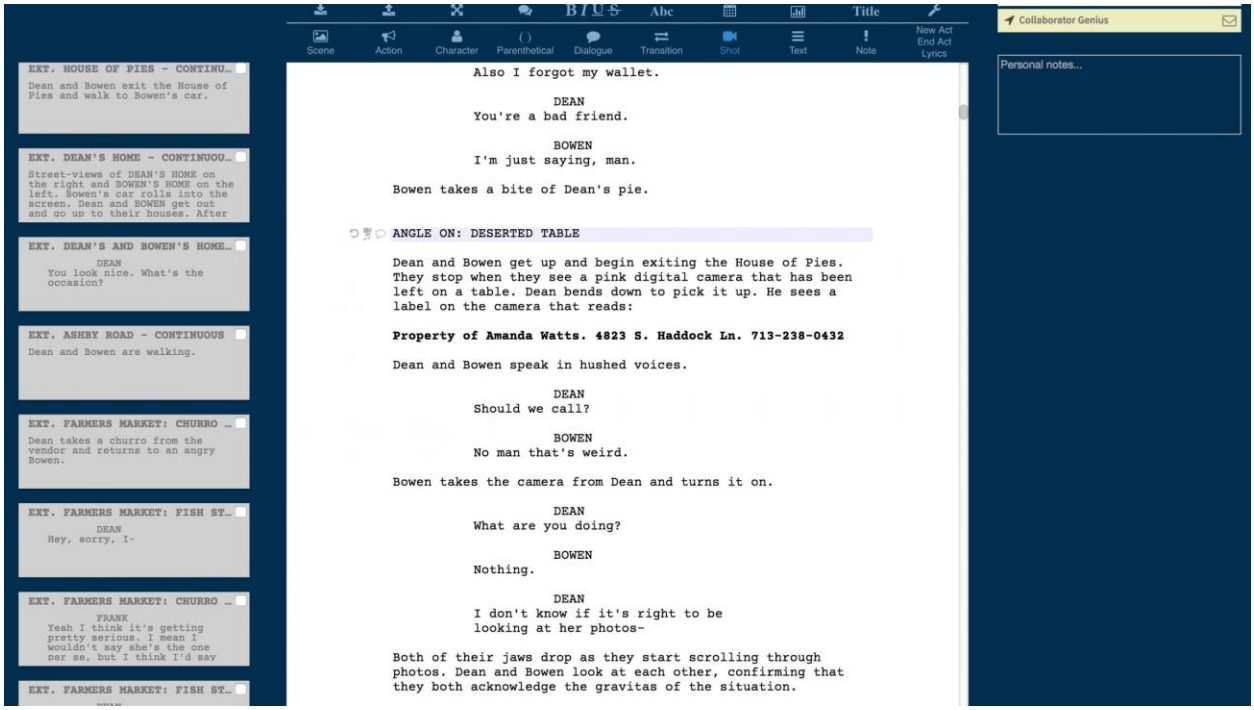

Ilustración 16. Imagen del sofware WriterDuet

2.6.4 Celtx

Celtx es la gran alternativa gratuita en cuanto a software de escritura de guiones. Ofrece opciones de pago en torno a 240$ al año para producciones profesionales con todas las características disponibles, pero también permite usarse en versión gratuita con las características mínimas.

El plan gratuito nos limita con las siguientes restricciones: solo nos permitirá hacer 3 proyectos (incluyendo los que están en la papelera), no permite la edición colaborativa en tiempo real y nuestros proyectos tendrán la marca de agua “Created using Celtx”.

Si nos permitirá importar archivos, sincronizar con dispositivo movil con las apps iOS y Android que ahora si son gratuitas, y también tomar anotaciones y exportar archivos como PDF.

Para comparar el software con el resto de la competencia, vamos a mencionar las características del plan pro. Como se ha mencionado ya, ofrece herramientas colaborativas y también servicio cloud. Ofrece herramientas organizativas como planificadores muy completos y herramientas de presupuesto para los diferentes equipamientos, localizaciones (para películas), etc.

Por ser pro, se te ofrece también herramientas de administración y protección por IP, para gestionar que usuarios y desde que Ips pueden modificar y con que permisos (solo lectura, lectura escritura, etc.). Además, ofrecen plugins de seguridad (autentificación doble factor y otras) y otros plugins orientados a marcas y a desarrollo específico de videojuegos (añadiendo variables y lógica de control).

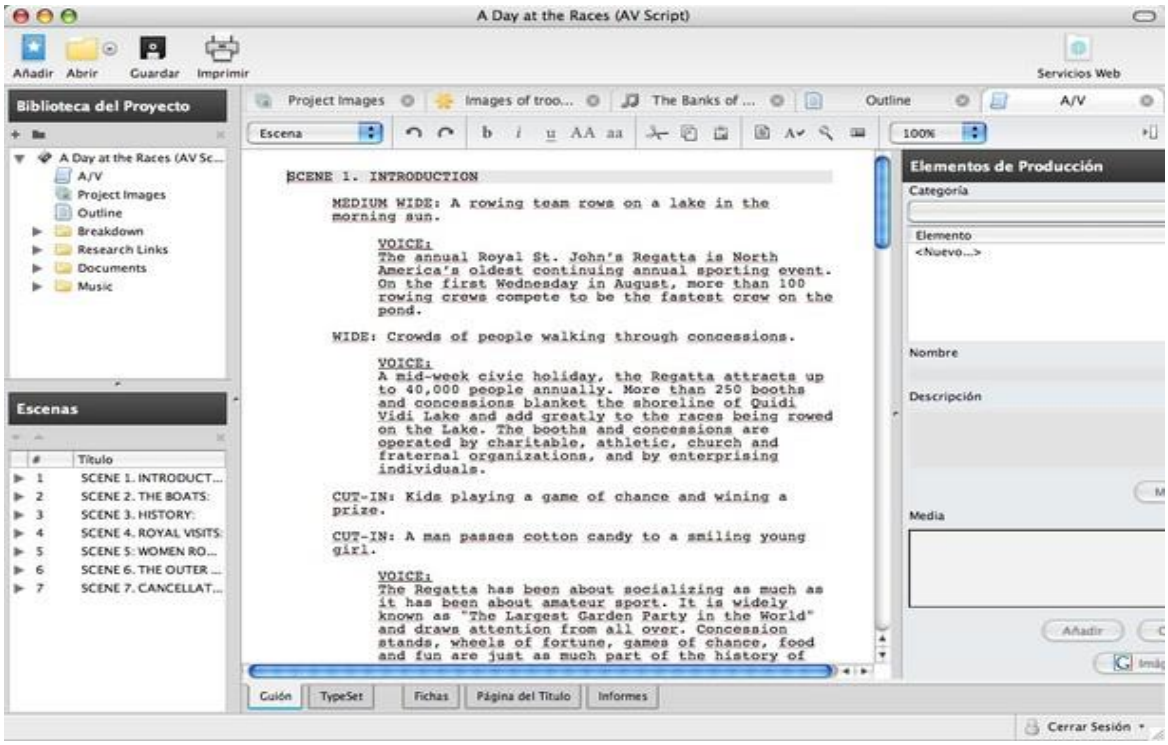

Ilustración 17. Imagen del software Celtx

Para la escritura del guión del juego DungeonCode se ha usado el programa Celtx y a continuación se va a hablar un poco del mismo.

2.7 Guión de DungeonCode

La historia nos presenta a dos personajes, Rey y Arcadium, maestro y alumno, respectivamente, los cuales pertenecen a la compañía WhiteHat. Estos son expertos de ciberseguridad y realizan investigaciones desde sus oficinas situadas en un castillo.

Tras la presentación, aparece Alexander, un enemigo de Rey durante años, que pertenece a la compañía BlackHat. Éste ha hackeado el sistema principal y obliga a nuestro personaje principal, Arcadium, a superar una serie de pruebas desde el ordenador central para poder liberarlo. Estas pruebas se van complicando a medida que se van superando.

Los niveles dentro del ordenador, se intercalan con escenas en el salón principal del castillo, donde se producen los principales diálogos que nos cuentan la historia. Arcadium, deberá derrotar 3 veces a Alexander hasta dejar por completo a los integrantes de WhiteHat. Además, Alexander, intenta persuadir a Arcadium para que se una a el y deje lo que había estado haciendo toda su vida.

Esta historia contiene referencias a la famosa saga de Star Wars con la analogía del lado oscuro/ los jedis con las compañías de BlackHat y WhiteHat. Con los mismos poderes (conocimientos en nuestro caso) unos se encargan de hacer el mal y otros de hacer buenas prácticas y combatirlo.

Como se comenta en las líneas futuras, en próximas actualizaciones del juego, aparecerá un nuevo enemigo que nos hará adentrarnos más en la historia de Arcadium y Rey y nos explicará más sobre las intenciones de BlackHat.

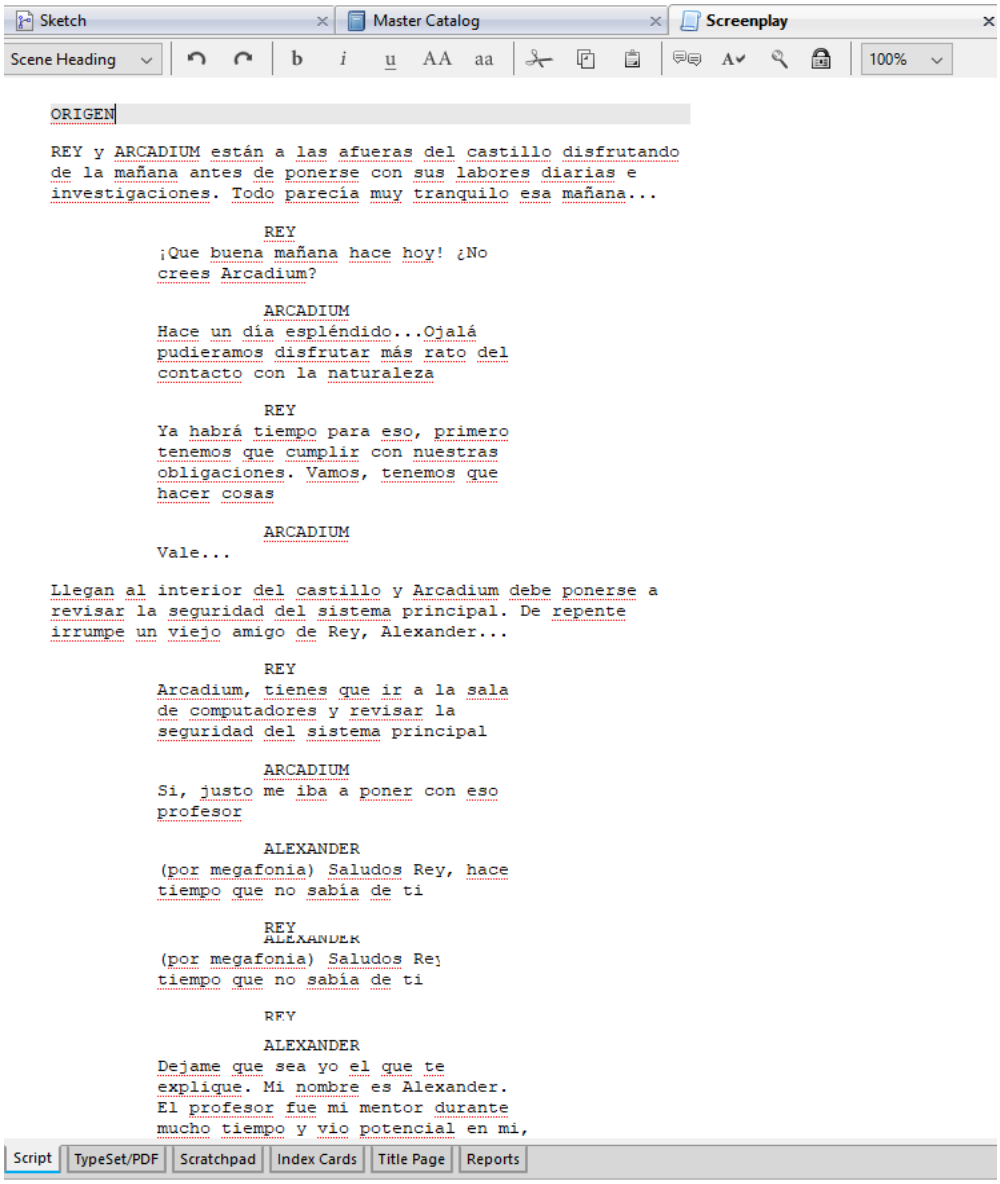

Ilustración 18. Guión de Dungeon Code

Tras esta introducción a la historia de los videojuegos, y comentar los principales motores gráficos y softwares de escritura de guion damos paso a las tecnologías utilizadas donde se señalará el hardware usado para el desarrollo y pruebas del proyecto.

3 TECNOLOGÍAS UTILIZADAS

Vamos a mencionar de manera breve las tecnologías y el hardware utilizado para el diseño y pruebas de este proyecto. Éste ha sido desarrollado sobre el portátil Asus GL552VW de mediados de 2016 con las características que se citan a continuación:

- Procesador Intel Core i7 modelo i7-6700HQ con 4 núcleos a 2.6GHz con una frecuencia turbo máxima de 3.5GHz con 6MB de caché compartida con la tecnología Intel Smart Cache.
- 16 Gb de memoria tipo DDR4 integrada a 2133MHz.
- Pantalla full HD de 15.6”.
- 256GB de almacenamiento con disco duro de tipo SSD.
- Tarjeta Gráfica Nvidia GTX 960M 4GB GDDR5.
- Sistema Operativo Windows 10 Home en su última actualización.

Ilustración 19. Portátil Asus G552VW

Para probar la aplicación web diseñada, se han usado los dos siguientes dispositivos:

1. Iphone 8, modelo de finales de 2017 con las siguientes características:

- Procesador A11 Bionic de 6 núcleos.
- Pantalla retina HD de 4,7”.
- 64GB de almacenamiento.
- 2GB Memoria Ram.
- S.O. iOS 13.3.

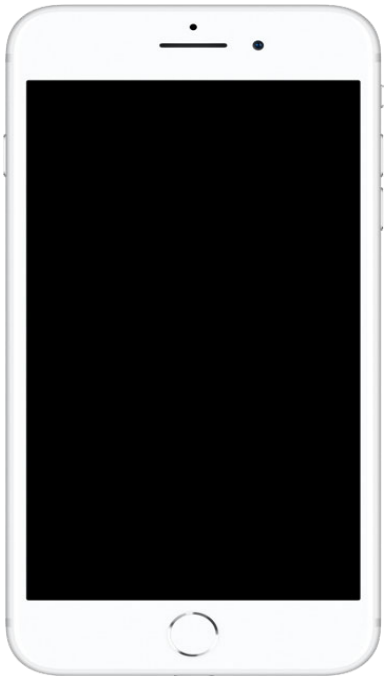

Ilustración 20. Iphone 8

2. Ipad 2019 Wifi con las siguientes características:

- Chip A12 fusión con arquitectura de 64 bits.
- Pantalla retina HD de 10,2”.
- 128GB de almacenamiento.
- 3GB de memoria Ram.
- S.O. iOS 13.3.

Ilustración 21. Ipad Apple 2019 Wifi

A continuación, damos paso a la ingeniería de requisitos en la que se tratarán las necesidades que el sistema ha de cumplir, así como los casos de uso que mostrarán parte del comportamiento del sistema.

4 REQUISITOS DEL SISTEMA

En este capítulo vamos a describir los requisitos que ha de tener el sistema, separando en dos subsistemas: la aplicación web y el videojuego.

Estos requisitos nos van a ayudar a entender de mejor manera el sistema y cuáles son las necesidades que se buscan satisfacer con la elaboración del mismo. Los requisitos van siendo revisados temporalmente, en diferentes versiones, analizando la viabilidad y validando que se están satisfaciendo las necesidades que inicialmente se esperaban.

4.1 Requisitos generales del sistema

Empezamos con los requisitos generales que nos permiten saber a nivel muy amplío que necesidades va a tener el sistema.

4.1.1 Requisitos generales de la aplicación web

RG-01	*Web didáctica*
Versión	*3 (24-04-2020)*
Descripción	*La finalidad de la página es que se aprendan lenguajes de programación a través del juego haciendo que sea más ameno que un estudio tradicional basado solo en la teoría y complementado con las lecciones.*
Importancia	*Muy alta.*
Comentarios	*Ninguno.*

Tabla 1. RG-01: Web Didáctica

RG-02	*Ayudas Web*
Versión	*3 (24-04-2020)*
Descripción	*Para cumplir el requisito RG-01 la web debe de tener lecciones completas sobre cada lenguaje de programación de manera que sean suficientes para tener una noción clara y general del lenguaje. Los juegos harán que se profundice con ejemplos prácticos y dinámicos.*
Importancia	*Muy alta.*
Comentarios	*Ninguno.*

Tabla 2. RG-02: Lecciones Web

RG-03	*La aplicación Web debe tener una interfaz clara y sencilla*
Versión	*3 (24-04-2020)*
Descripción	*La intención de la página es que tenga las secciones bien diferenciadas entre la parte de aprendizaje con lecciones y la parte de videojuegos. No es tan necesario una interfaz muy cargada como lo es que el contenido sí que sea relevante.*
Importancia	*Alta*
Comentarios	*Ninguno.*

Tabla 3. RG-03: Interfaz clara y sencilla

RG-04	*Aplicación web segura*
Versión	*3 (24-04-2020)*
Descripción	*Se requieren de mecanismos de seguridad que no comprometan a los datos de los usuarios (mínimos, ya que no habrá demasiada información) y del sistema servidor que aloja la aplicación Web*
Importancia	*Media/Alta*
Comentarios	*Ninguno.*

Tabla 4. RG-04: Seguridad

RG-05	Fluidez
Versión	*3 (24-04-2020)*
Descripción	*Se busca que los videojuegos corran de manera fluida en un equipo con requisitos bajos.*
Importancia	*Media.*
Comentarios	*Para lograrlo será necesario no cargar demasiado la página y que sea lo más ligera posible dentro de que ofrezca un contenido mínimo*

Tabla 5. RG-05: Fluidez

RG-06	*Portabilidad*
Versión	*3 (24-04-2020)*
Descripción	*Se busca que la aplicación pueda ser accesible de diferentes tipos de dispositivos y cualquier sistema operativo aumentando al máximo la compatibilidad.*
Importancia	*Media*
Comentarios	*Ninguno.*

Tabla 6. RG-6: Portabilidad

4.1.2 Requisitos generales del videojuego

RG-07	*Niveles*
Versión	*3 (24-04-2020)*
Descripción	*El juego de Dungeon Code funciona por niveles, que deben ser desbloqueados y podrán repetirse al cargar partida desde el punto que queramos.*
Importancia	*Alta*
Comentarios	*Ninguno.*

Tabla 7. RG-07: Niveles

RG-08	*Guion e historia*
Versión	*3 (24-04-2020)*
Descripción	*El juego tendrá un guion que se seguirá a lo largo de la historia para que el usuario que juega se sumerja más profundamente en la experiencia de juego*
Importancia	*Alta*
Comentarios	*Ninguno.*

Tabla 8. RG-08: Historia

RG-09	*Ajustes*
Versión	*3 (24-04-2020)*
Descripción	*El juego tendrá un menú de ajustes donde se podrán cambiar ciertos parámetros*
Importancia	*Alta*
Comentarios	*Será necesario definir esos parámetros como el lenguaje o el volumen.*

Tabla 9. RG-09: Ajustes

RG-10	*Tutorial*
Versión	*3 (24-04-2020)*
Descripción	*El juego tendrá un tutorial donde introducirá las mecánicas de movimiento y las explicaciones respectivas a la resolución de los niveles para que el usuario pueda aprender progresivamente.*
Importancia	*Media*
Comentarios	*Ninguno*

Tabla 10. RG-010: Tutorial

4.2 Requisitos funcionales del Sistema

Los requisitos funcionales se centran principalmente en aportar información sobre la funcionalidad y el comportamiento del sistema. Vamos a tener de dos tipos: casos de uso y requisitos de información.

4.2.1 Casos de uso

Los casos de uso es una técnica que se usa para la descripción de interacciones entre el sistema y los actores del sistema (personas u otros sistemas) considerando el sistema como una caja negra, como veremos en los siguientes diagramas.

4.2.1.1 Casos de uso de la aplicación web

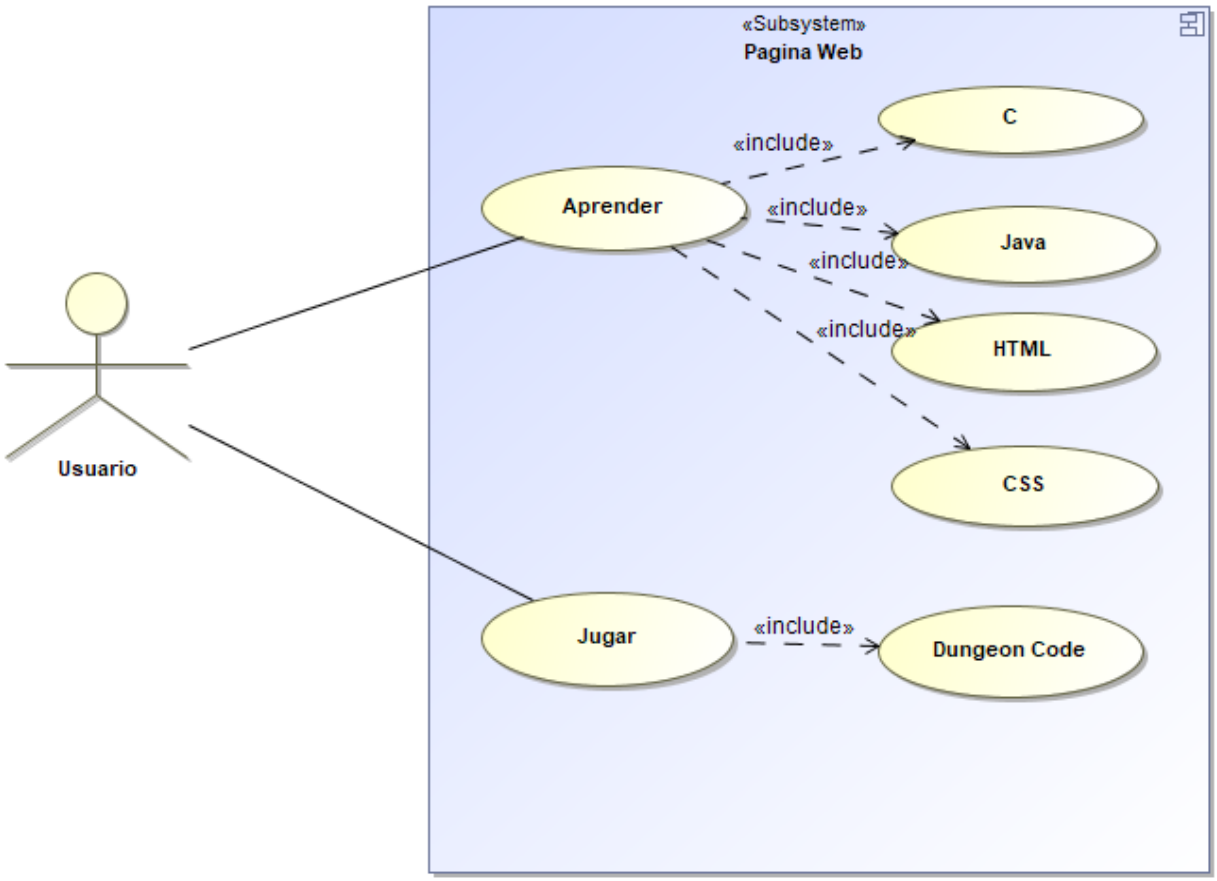

Ilustración 22. Casos de uso pagina web

4.2.1.2 Casos de uso del videojuego

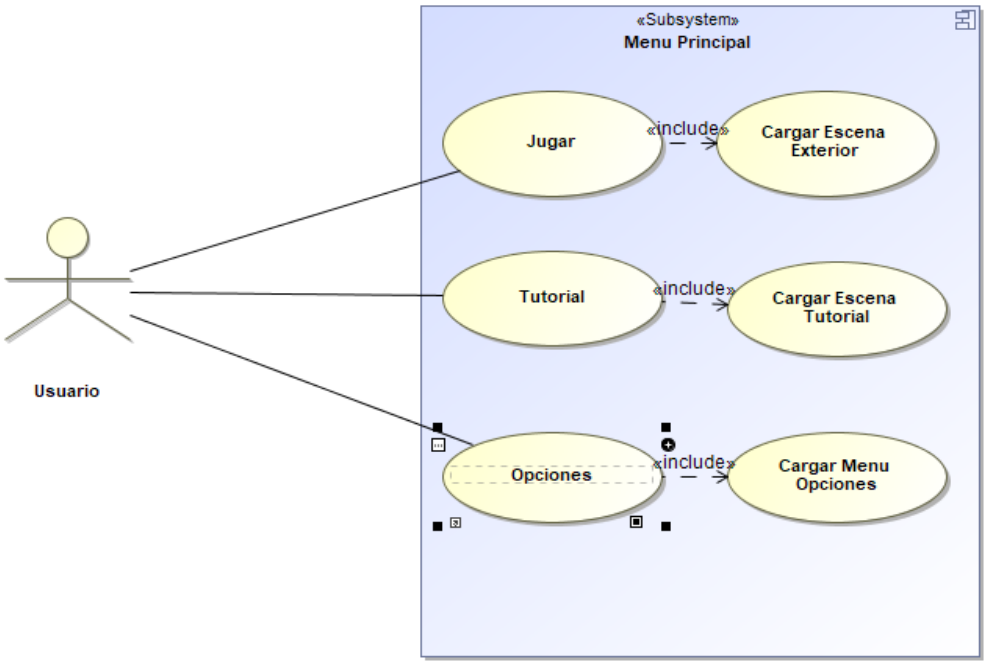

Ilustración 23. Caso de uso menú principal

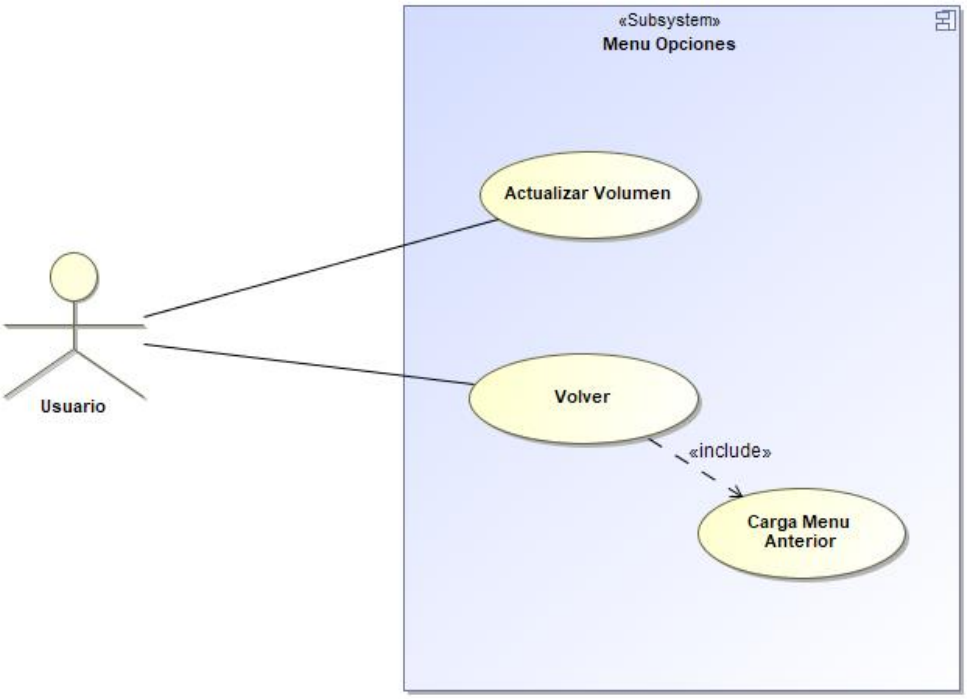

Ilustración 24. Caso de uso menú opciones

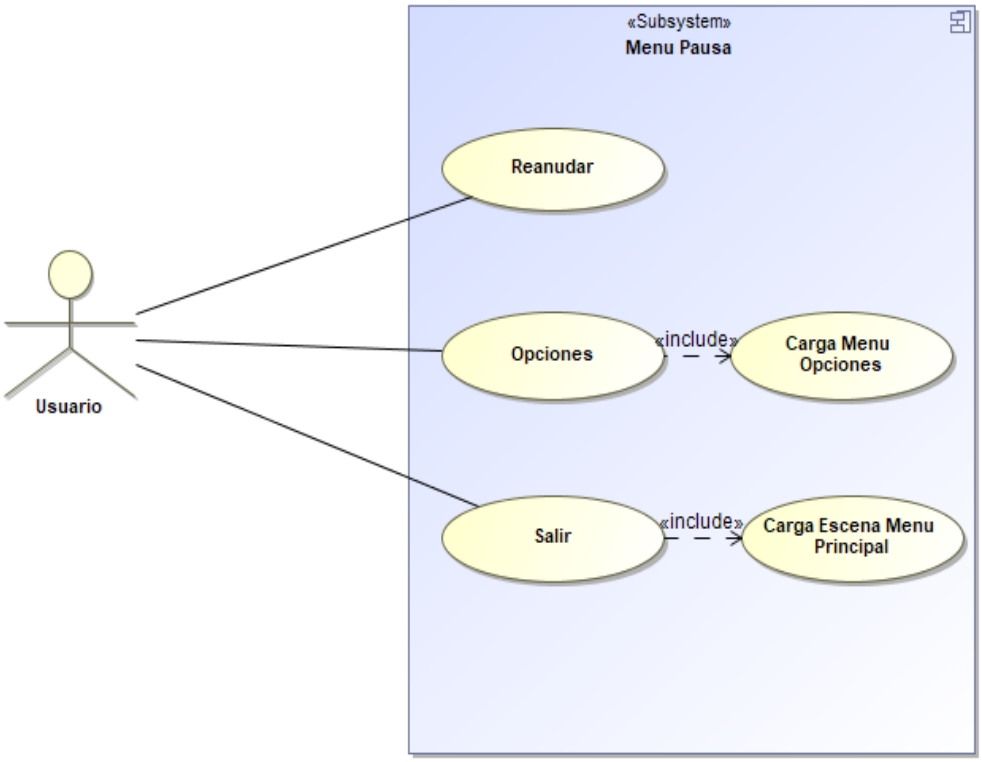

Ilustración 25. Caso de uso menú pausa

4.2.2 Especificación de actores del sistema

A-01	*Usuario*
Versión 3	*24-04-2020*
Descripción	*Este actor representa al usuario final de la aplicación.*
Comentarios	*Ninguno.*

Tabla 11. Actor Usuario

4.2.3 Especificación de casos de uso del sistema

CU-01	*Jugar*	
Versión	*3 (24-04-2020)*	
Dependencias	• *RG-07*	
Precondición	*Ninguna.*	
Descripción	*El sistema deberá comportarse como se describe cuando el usuario pulse el botón de jugar del menú principal tras haber accedido al sistema.*	
Secuencia Normal	**Paso**	**Acción**
	1	*El usuario selecciona Jugar o Cargar para seleccionar el nivel.*
	2	*En caso de pulsar Cargar, el usuario podrá elegir el nivel deseado dentro de los que haya desbloqueado.*
Postcondición	*El sistema carga la escena correspondiente.*	
Importancia	*Alta.*	

Tabla 12. Caso de uso Jugar

CU-02	*Tutorial*	
Versión	*3 (24-04-2020)*	
Dependencias	• *RG-10*	
Precondición	*Ninguna.*	
Descripción	*El sistema deberá comportarse como se describe cuando el usuario pulse el botón de Tutorial del menú principal tras haber accedido al sistema.*	
Secuencia Normal	**Paso**	**Acción**
	1	*El usuario selecciona Tutorial.*
	2	El sistema carga el tutorial.
Postcondición	*El usuario debe superar los distintos niveles del tutorial.*	
Importancia	*Alta*	

Tabla 13. Caso de uso Tutorial

CU-03	*Opciones*	
Versión	*3 (24-04-2020)*	
Dependencias	• *RG-09*	
Precondición	*Ninguna.*	
Descripción	*El sistema deberá comportarse como se describe cuando el usuario pulse el botón de Opciones del menú principal o del menú pausa (ambos con misma funcionalidad)*	
Secuencia Normal	**Paso**	**Acción**
	1	*El usuario selecciona Opciones*
	2	El sistema elimina el menú actual
	3	El sistema carga los elementos del menú opciones
Postcondición	*Se muestra el nuevo menú con diferentes funcionalidades*	
Importancia	*Alta*	

Tabla 14. Caso de uso Opciones

CU-04	*Actualizar Volumen*	
Versión	*3 (24-04-2020)*	
Dependencias	• *RG-09* • *CU-03*	
Precondición	*El usuario debe haberse abierto el menú opciones*	
Descripción	*El sistema deberá comportarse como se describe cuando el usuario use el slider de Volumen del menú opciones*	
Secuencia Normal	**Paso**	**Acción**
	1	*El usuario ajusta el valor del slider a un nuevo valor*
	2	El sistema actualiza llama al componente de sonido con el nuevo valor de sonido
Postcondición	*La música de fondo habrá cambiado de volumen*	
Importancia	*Alta*	

Tabla 15. Caso de uso Actualizar Volumen

CU-05	*Volver*	
Versión	*3 (24-04-2020)*	
Dependencias	• *RG-09* • *CU-03*	
Precondición	*El usuario debe haber abierto el menú opciones*	
Descripción	*El sistema deberá comportarse como se describe cuando el usuario pulse el botón de Volver del menú opciones*	
Secuencia Normal	**Paso**	**Acción**
	1	*El usuario pulsa el botón Volver*
	2	El sistema elimina el menú opciones
	3	El sistema carga los elementos del menú anterior, ya sea el menú de pausa o el principal
Postcondición	*Se muestra el anterior menú*	
Importancia	*Alta*	

Tabla 16. Caso de uso Volver

CU-06	*Reanudar*	
Versión	*3 (24-04-2020)*	
Dependencias	• *RG-09* • *CU-03*	
Precondición	*El usuario debe haberse abierto el menú pausa pulsando escape o mediante el botón integrado en pantalla*	
Descripción	*El sistema deberá comportarse como se describe cuando el usuario pulse el botón reanudar del menú de pausa*	
Secuencia Normal	**Paso**	**Acción**
	1	*El usuario pulsa el botón Reanudar*
	2	El sistema elimina el menú pausa y carga las variables necesarias
	3	El sistema cambia la escala de tiempo para que el juego deje de estar congelado
Postcondición	*Se reanuda el juego tal como estaba en un inicio.*	
Importancia	*Alta*	

Tabla 17. Caso de uso Reanudar

CU-07	*Salir*	
Versión	*3 (24-04-2020)*	
Dependencias	• *RG-09*	
Precondición	*El usuario debe haberse abierto el menú pausa pulsando escape o mediante el botón integrado en pantalla*	
Descripción	*El sistema deberá comportarse como se describe cuando el usuario pulse el botón salir del menú de pausa*	
Secuencia Normal	**Paso**	**Acción**
	1	*El usuario pulsa el botón salir*
	2	El sistema elimina el menú de pausa y carga la escena del menú principal
Postcondición	*Se reanuda el juego tal como estaba en un inicio.*	
Importancia	*Alta*	

Tabla 18. Caso de uso Salir

4.2.4 Requisitos de información

Los requisitos de información describen la información que el sistema debe almacenar para obtener las funcionalidades y comportamiento que se desea.

IRQ-01	*Lecciones de lenguajes*
Versión	*3 (24-04-2020)*
Descripción	El sistema deberá almacenar la información correspondiente a las lecciones de los diferentes lenguajes de programación para servir de ayuda al aprendizaje y resolución de los niveles. En concreto*:*
Datos específicos	• *Lecciones del lenguaje C* • *Lecciones del lenguaje Java*

Tabla 19. Requisito información lenguajes

IRQ-02	*Datos de guardado*
Versión	*3 (24-04-2020)*
Descripción	El sistema deberá almacenar la información correspondiente al guardado para continuar el progreso del videojuego. En concreto*:*
Datos específicos	• *Nivel Alcanzado* • *Posición del personaje*

Tabla 20. Requisito información usuario

IRQ-03	*Ajustes*
Versión	*3 (24-04-2020)*
Descripción	*El sistema deberá almacenar la información de los ajustes de volumen*
Datos específicos	• *Valor de volumen*

Tabla 21. Requisito información ajustes

4.3 Requisitos no funcionales del sistema

Los requisitos no funcionales se refieren a propiedades específicas del sistema como funcionalidad, mantenibilidad, portabilidad, seguridad, etc. y restringen los requisitos funcionales existentes o ayudan a generar algunos nuevos.

4.3.1 Requisitos de fiabilidad

NFR-01	*Fiabilidad*
Versión	*3 (24-04-2020)*
Descripción	*El Sistema deberá de tardar un máximo de 30 minutos para la recuperación de un fallo de caída total, en el 90% de las veces*
Importancia	*Primordial*
Comentarios	*Ninguno*

Tabla 22. Fiabilidad

4.3.2 Requisitos de mantenibilidad

NFR-03	*Mantenibilidad*
Versión	*3 (24-04-2020)*
Descripción	*La inserción de nuevos niveles deberá de ser sencilla*
Importancia	*Alta*
Comentarios	*Ninguno*

Tabla 23. Mantenibilidad

4.3.3 Requisitos de portabilidad

NFR-04	*Portabilidad*
Versión	*3 (24-04-2020)*
Descripción	*La aplicación será accesible desde teléfonos iOS y Android, así como compatible con Windows y Linux*
Importancia	*Media*
Comentarios	*Ninguno*

Tabla 24. Portabilidad

Una vez definidos todos los requisitos del sistema, pasamos al primer diseño del mismo, el cual deberá cumplirlos y puede que nos genere la necesidad de definir otros nuevos. En el siguiente apartado veremos el diseño de las clases más importantes del sistema y como se relacionan entre sí. Además, se introduce la interfaz gráfica.

5 DISEÑO

En este apartado vamos a empezar con un diagrama de clases en el que aparecen las clases más importantes del diseño del videojuego donde podremos ver una relación de estas. Tras esto, se analizará más detalladamente cada una de ellas y se mostrará la interfaz gráfica. El código referente tanto a las clases como a la gestión de las interfaces, se trata más adelante, en el apartado de implementación. Para terminar, se muestran los diagramas de comportamiento referentes a la carga de las partidas guardadas, a la resolución de los niveles y de la gestión de los diálogos.

5.1 Clases

Antes de pararnos a hablar del diseño de las clases, hay que mencionar que todos los scripts son heredados la clase MonoBehaviour, que es la clase base de Unity.

A continuación, se muestra un diagrama de clases, en el que podemos ver una visión global con las clases que tienen más relevancia.

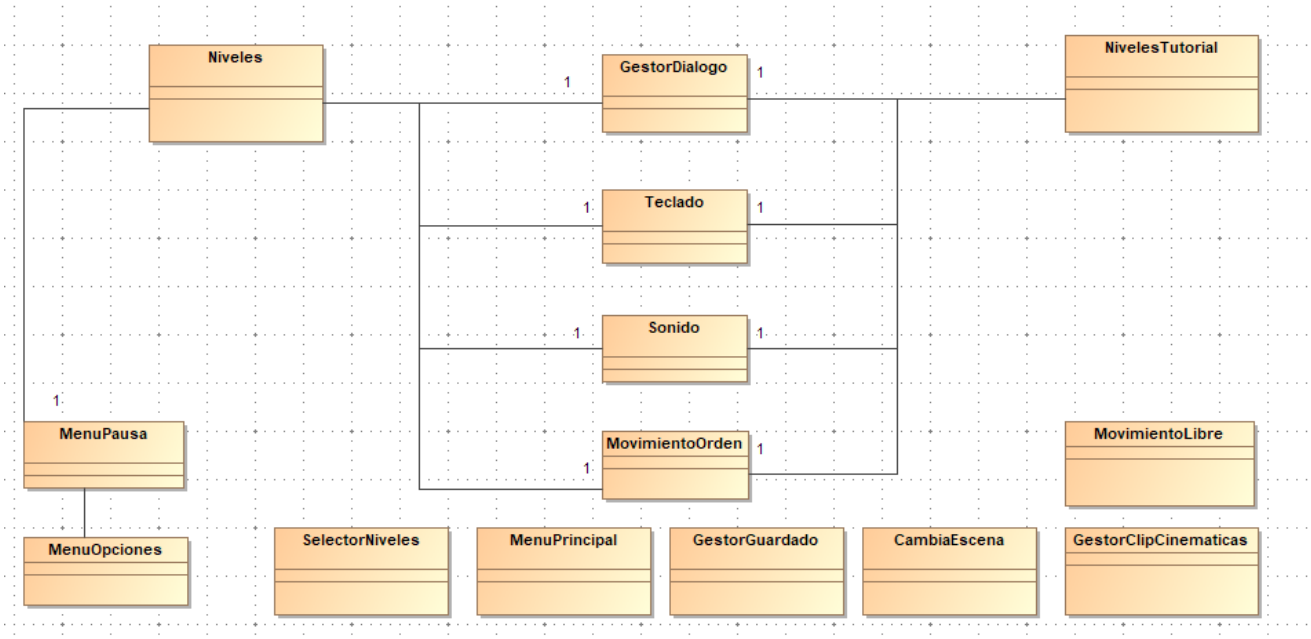

Ilustración 26. Diagrama de clases

5.1.1 Clase MenuPrincipal

Esta clase es la encargada de gestionar el menú de arranque del juego, mediante los listener de los botones y del slider que cambia el volumen de la banda sonora. Tiene una referencia a la clase Sonido que veremos más adelante.

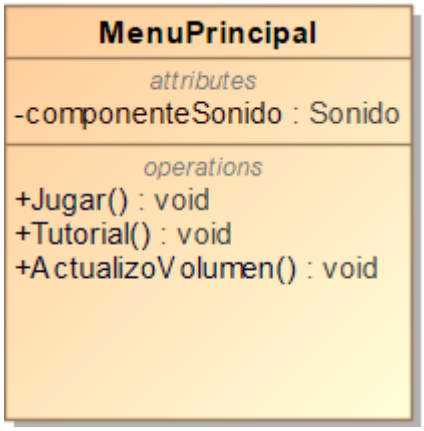

Ilustración 27. Clase MenuPrincipal

5.1.2 Clase MenuPausa

Esta clase se encarga de la gestión del menú de pausa que se abre al pulsar la tecla Escape o el botón incrustado en pantalla en la mayoría de niveles. Tiene una referencia a la clase Sonido que veremos más adelante y a diferentes objetos de Unity como botones, y el MenuPausa, que es un panel del Canvas (interfaz gráfica).

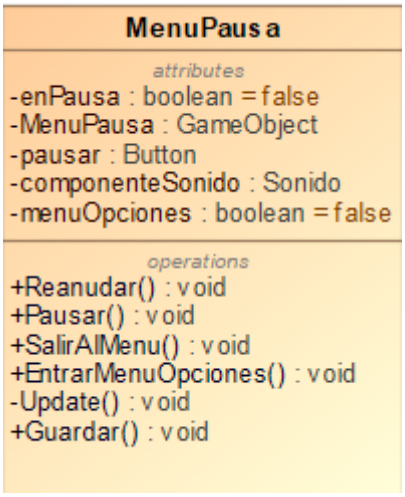

Ilustración 28. Clase MenuPausa

5.1.3 Clase MovimientoLibre

Clase que gestiona el movimiento fuera de los niveles, es decir, el movimiento libre (sin teclado) donde el jugador pulsará las teclas W, A, S, D o las flechas del teclado para moverse. Tiene referencias a las clases Animator, Rigidbody2D y Vector3 de Unity, las cuales se usan para calcular el movimiento y representar la animación de movimiento correspondiente.

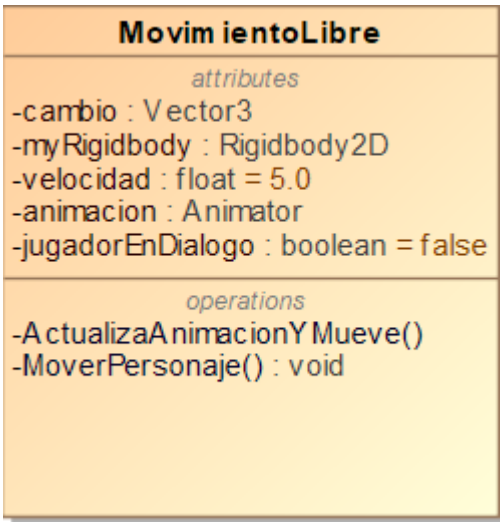

Ilustración 29. Clase MovimientoLibre

5.1.4 Clase MovimientoOrden

Esta clase gestiona la parte del movimiento correspondiente a los niveles (con teclado). Tiene referencias a Rigidbody2D, Vector2 y Animator para gestionar las animaciones cuando se lee el movimiento por teclado.

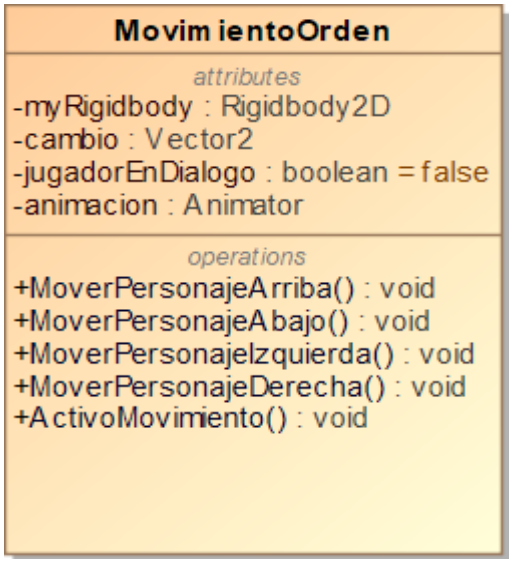

Ilustración 30. MovimientoOrden

5.1.5 Clase Sonido

Esta clase se encarga de la gestión del sonido, cambiando el volumen, parando los clips de audios o reanudándolos. Es muy importante ya que es de las pocas clases que aparece en todas las escenas del juego, para establecer una banda sonora u otra, los clips de los niveles completados, etc. Tiene referencias a las clases AudioClip y AudioSource de Unity para la gestión de los diferentes clips y la única fuente de audio existente por nivel.

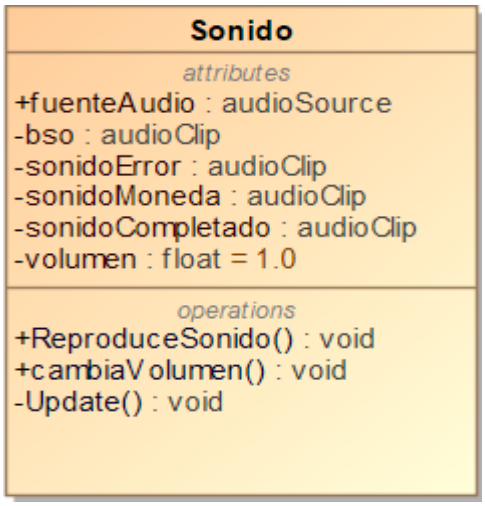

Ilustración 31. Clase Sonido

5.1.6 Clase MovimientoCamara

Esta clase gestiona el movimiento de la cámara asociada a un cierto objeto. Al igual que la clase Sonido, aparece en todas las escenas y suele estar adjuntada al objeto Player que representa a nuestro jugador en partida. Referencia a la clase Transform y Vector2 de Unity para obtener el cambio en la posición del jugador y actualizarlo a la cámara para que se muevas con el.

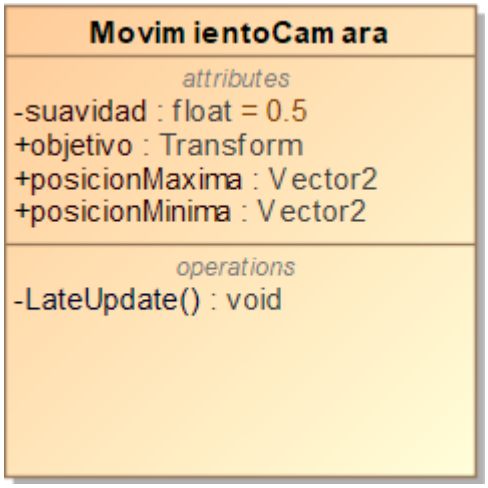

Ilustración 32. Clase MovimientoCamara

5.1.7 Clase CambioHabitacion

Esta clase gestiona las animaciones de movimiento entre habitaciones con puerta atravesable y escaleras. Tiene referencias a las clases Vector2 y Vector3 de Unity y la clase MovimientoCamara, para que la cámara se mueva con el jugador cuando atraviesa una puerta y se "teleporta".

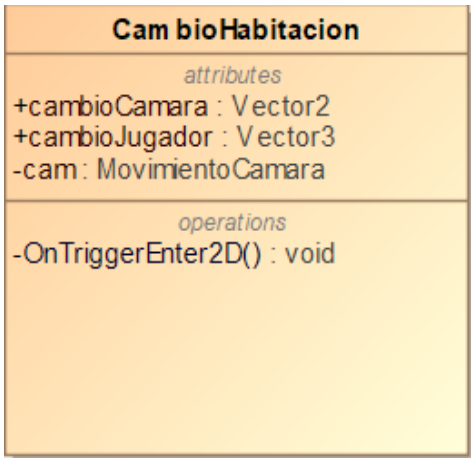

Ilustración 33. Clase CambioHabitacion

5.1.8 Clase CambioEscena

Esta clase gestiona los cambios de escenas, y muestra una animación de difuminación para darle más dinamismo. Referencia al panel como GameObject, que representará a un objeto del tipo Panel del Canvas (interfaz gráfica).

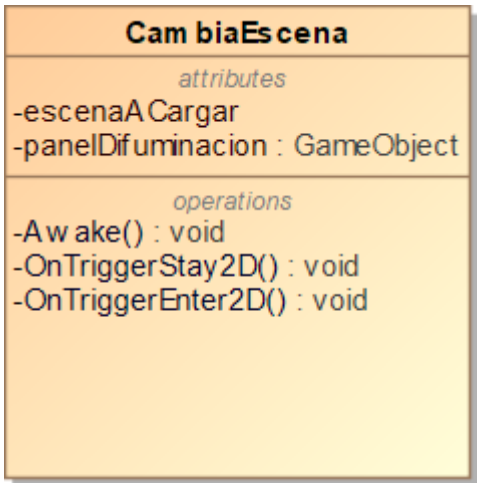

Ilustración 34. Clase CambioEscena

5.1.9 Clase Niveles

Clase que gestiona la lógica de los niveles. Es una de las clases más completas y cuenta con referencias a las clases de Sonido, Teclado, GestorClipsCinemáticas y la clase GestorDialogo.

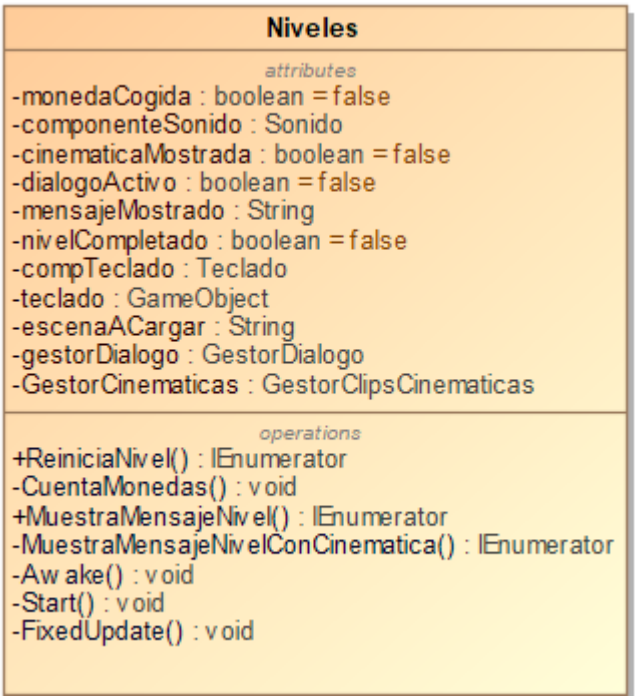

Ilustración 35. Clase Niveles

5.1.10 Clase NivelesTutorial

Clase con utilidad parecida a la anterior que gestiona el funcionamiento de los niveles del tutorial, que son parecidos a los niveles genéricos, pero con un diálogo inicial a modo de ayuda. Tiene referencias a las clases MovimientoOrden y GestorDialogo que veremos después.

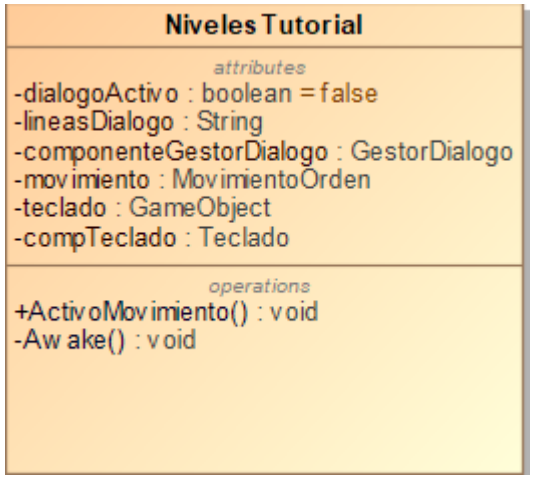

Ilustración 36. Clase NivelesTutorial

5.1.11 Clase Tutorial

Clase que gestiona las escenas Tutorial y TutorialCompletado, es decir, las escenas previas y posterior a los niveles del tutorial.

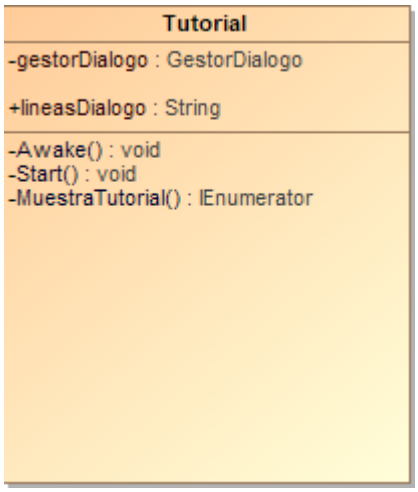

Ilustración 37. Clase Tutorial

5.1.12 Clase Dialogos

Esta clase se encarga de gestionar los diálogos mostrados por las interacciones dentro del juego cuando el jugador tiene el control del mismo. Es decir, todos los diálogos que se muestran al acercarse a los carteles, a Rey, etc. Tiene referencias a las clases MovimientoOrden, MovimientoLibre, y GestorDialogo, así como a las clases GameObject y Text de Unity para la gestión de los textos.

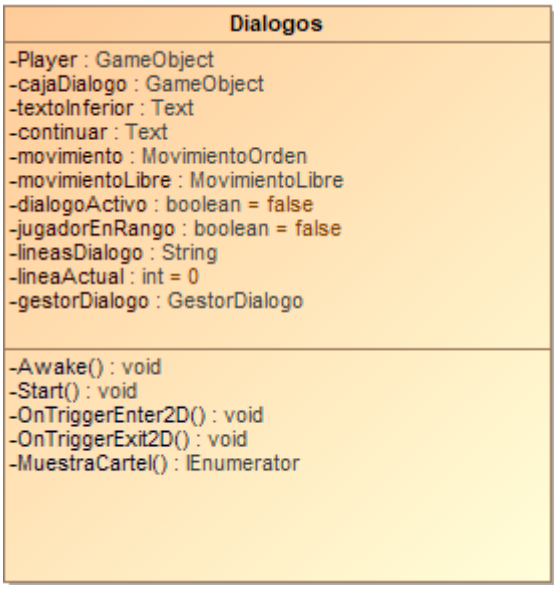

Ilustración 38. Clase Dialogos

5.1.13 Clase GestorDialogo

Esta es la clase más importante para la gestión de diálogos. Todas las demás clases que tratan con diálogos, se apoyan en esta para mostrar los diálogos, en diferentes situaciones o posiciones según interese. Además, se encarga de la des/activación de los objetos de texto.

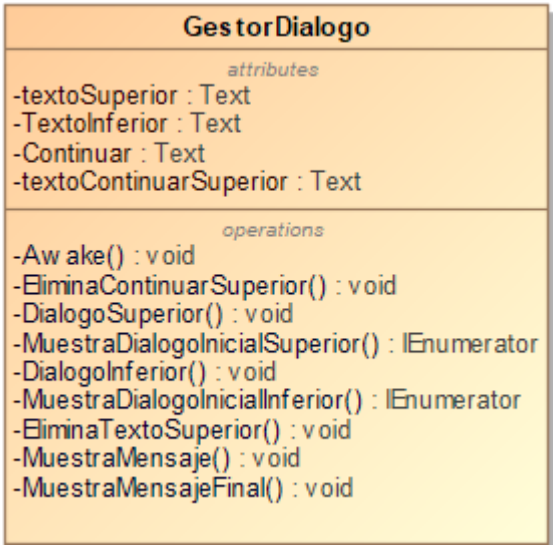

Ilustración 39. Clase GestorDialogo

5.1.14 Clase CartelesHabitacion

Esta clase se encarga de la gestión de los carteles que aparecen y desaparecen al entrar en diferentes habitaciones o zonas de forma informativa para que el jugador vaya conociendo las diferentes zonas del mapa.

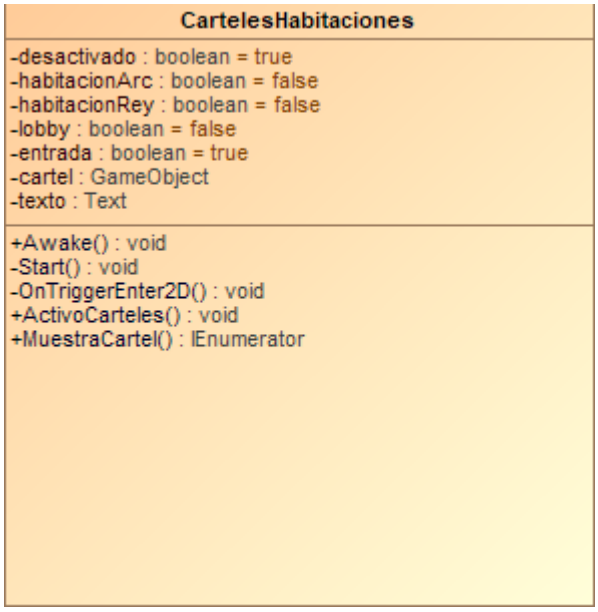

Ilustración 40. Clase CartelesHabitacion

5.1.15 Clase GestorClipsCinematicas

Clase que gestiona la reproducción y pausa de los diferentes clips de cinemáticas de una misma escena. Contiene referencias a la clase PlayableDirector, y a los objetos de la interfaz gráfica y del gestor de cinemáticas para su correcto funcionamiento.

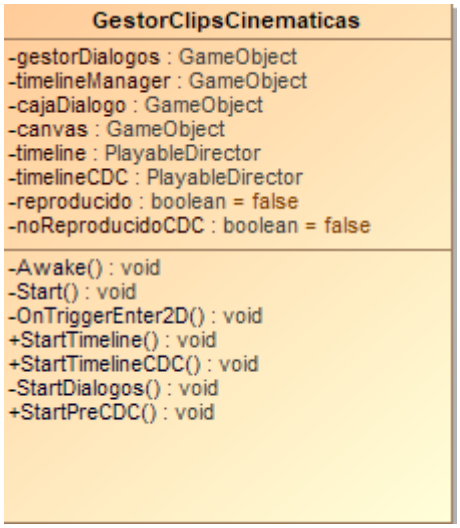

Ilustración 41. Clase GestorClipsCinematicas

5.1.16 Clase GestorCinematicaDialogos

Esta clase gestiona los diálogos que forman parte de las cinemáticas. Contiene referencias a la clase PlayableDirector de Unity, y se complementa con la herramienta RPGTalk (de la cual se hablará más adelante) y sus librerías para una correcta gestión de los diálogos, las cinemáticas y el ajuste de variables para cuando el jugador vuelve a tener el control del videojuego.

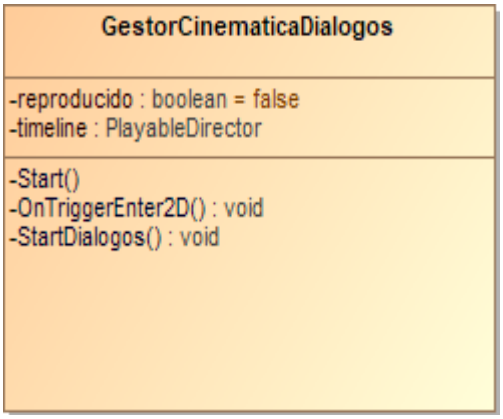

Ilustración 42. Clase GestorCinematicaDialogos

5.1.17 Clase Teclado

Clase que gestiona el teclado, mostrándolo tras los diálogos, y el movimiento del personaje dentro de los niveles.

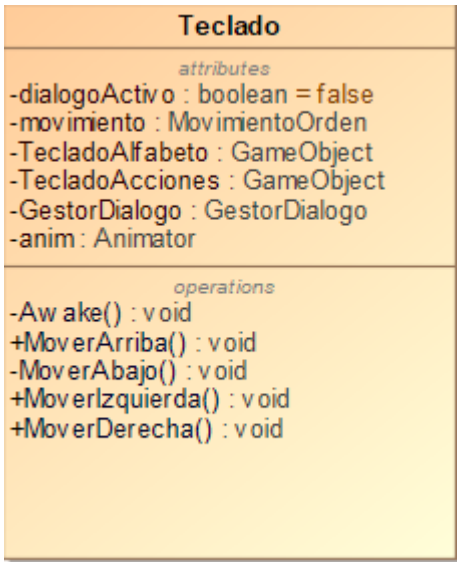

Ilustración 43. Clase Teclado

5.1.18 Clase GestorGuardado

Clase que gestiona el guardado del jugador con su posición y nivel usando la clase DatosJugador.

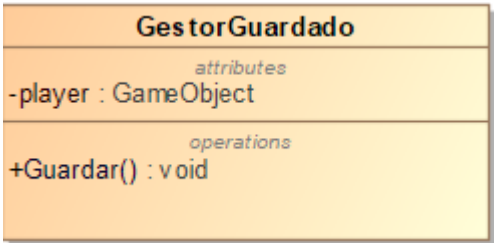

Ilustración 44. Clase GestorGuardado

5.1.19 Clase DatosJugador

Clase que almacena tanto la posición como el nivel del jugador para cargar correctamente.

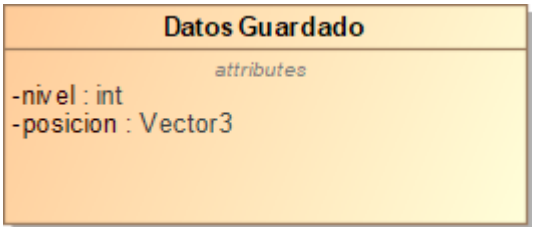

Ilustración 45. Clase DatosJugador

5.1.20 Clase Jugador

Clase que gestiona las animaciones del jugador cuando requiere una interacción con el juego.

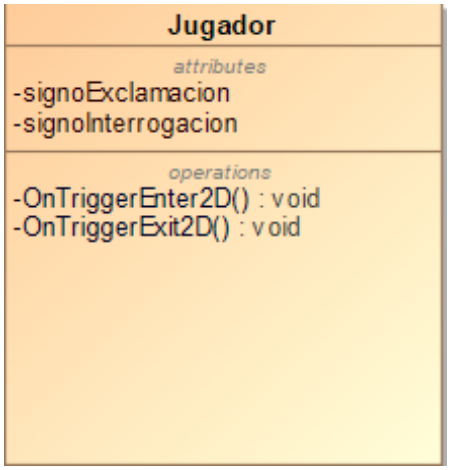

Ilustración 46. Clase Jugador

5.2 Interfaz Gráfica

A continuación, se va a mostrar el diseño de las diferentes interfaces gráficas de la aplicación.

5.2.1 MenuPrincipal

A continuación, se muestra el menú principal final, con los botones de arranque de juego, carga por donde guardamos la última vez, la opción de tutorial, o el menú de opciones donde podremos ajustar el volumen.

Ilustración 47. Interfaz MenuPrincipal

5.2.2 MenuPausa

El menú de pausa es el siguiente. Nos permite guardar en el punto que estamos, o bien continuar por donde vamos. Además, tiene un menú opciones igual que el de la pantalla principal, y una opción de salir directamente al menú principal.

Ilustración 48. Interfaz MenuPausa

Además, en la esquina superior derecha, mientras estamos jugando, aparecerá un icono de pausa en todas las escenas y momentos en los que el jugador pueda pausar.

Ilustración 49. Interfaz PausaEnPantalla

5.2.3 MenuOpciones

El menú de opciones nos permite ajustar el volumen a través de una slider que establecerá el volumen total de la música.

Ilustración 50. Interfaz MenuOpciones

5.2.4 SeleccionNiveles

Esta interfaz se usa para cargar por donde iba el jugador. Por defecto están todos los niveles bloqueados y en función de en que nivel se guardó por última vez, se desbloque la posibilidad de empezar por el nivel por el cual íbamos, o por alguno de los anteriores si el jugador quisiera repetirlos.

Ilustración 51. Interfaz SelecciónNiveles

Si por ejemplo, habíamos guardado la partida en el nivel 3, al intentar cargar se nos permitirá empezar desde los siguientes niveles.

Ilustración 52. Interfaz SelecciónNiveles2

5.2.5 Interfaz Teclado

Para la resolución de los niveles se usa un teclado. En este caso está dividido en dos: uno para las acciones y otro con el alfabeto para escribir.

Ilustración 53. Interfaz TecladoAcciones

Ilustración 54. Interfaz TecladoAlfabeto

5.3 Comportamiento

A continuación, se detalla el comportamiento del sistema en las situaciones de carga de niveles, de reproducción de diálogos y de resolución de niveles.

5.3.1 Diagramas de actividad

Para modelar el comportamiento, usaremos diagramas de actividad. Estos diagramas UML muestran un flujo de control haciendo énfasis en la secuencia de comportamiento y las condiciones de control.

5.3.1.1 Carga de niveles

Este diagrama describe el comportamiento cuando se pretende realizar una carga del archivo de guardado. En este caso, no se carga directamente el nivel, sino que se ofrece al usuario la opción de poder cargar niveles anteriores. Si no existe un archivo de guardado, se ofrece la opción por defecto en la que se empieza por la introducción.

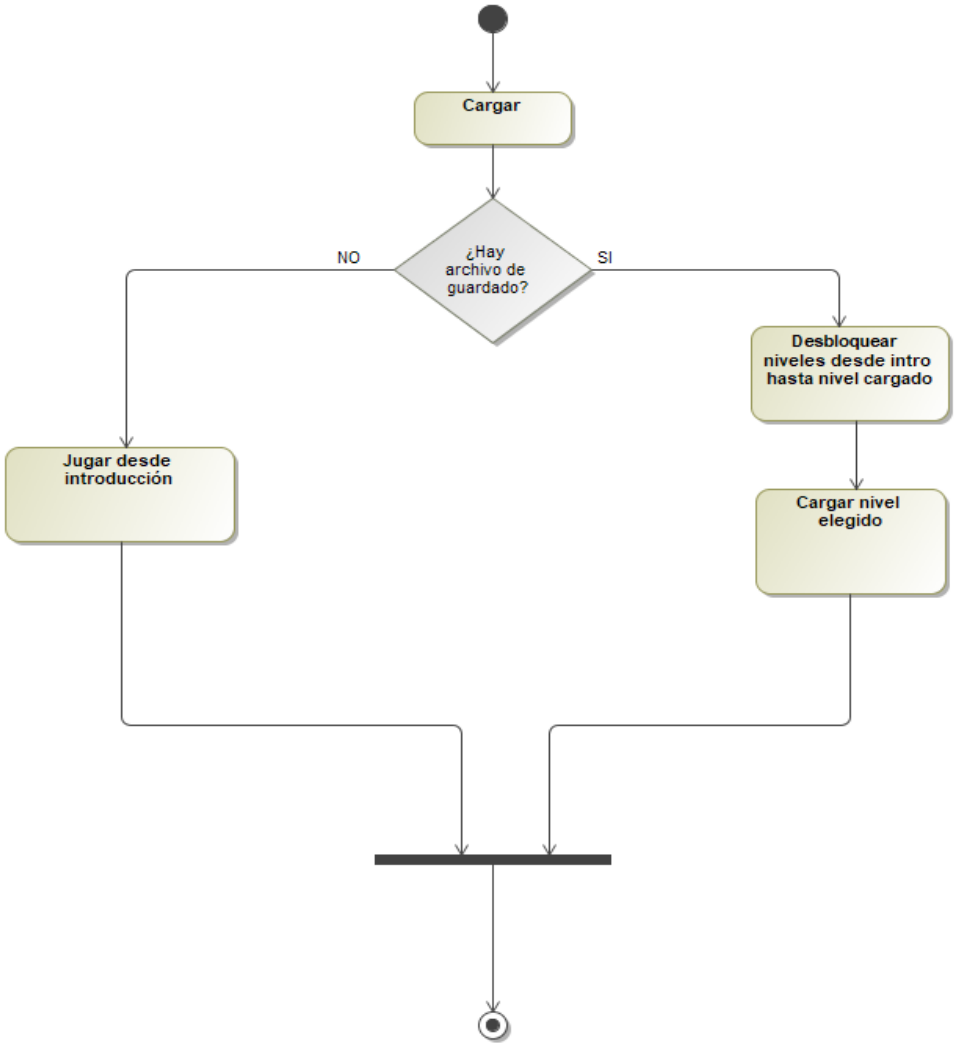

Ilustración 55. Diagrama de secuencia carga

5.3.1.2 Diálogos

Este diagrama muestra el comportamiento cuando se muestra un diálogo, ya sea con los diferentes carteles del juego, al completar un nivel, o cuando se habla con alguno de los personajes.

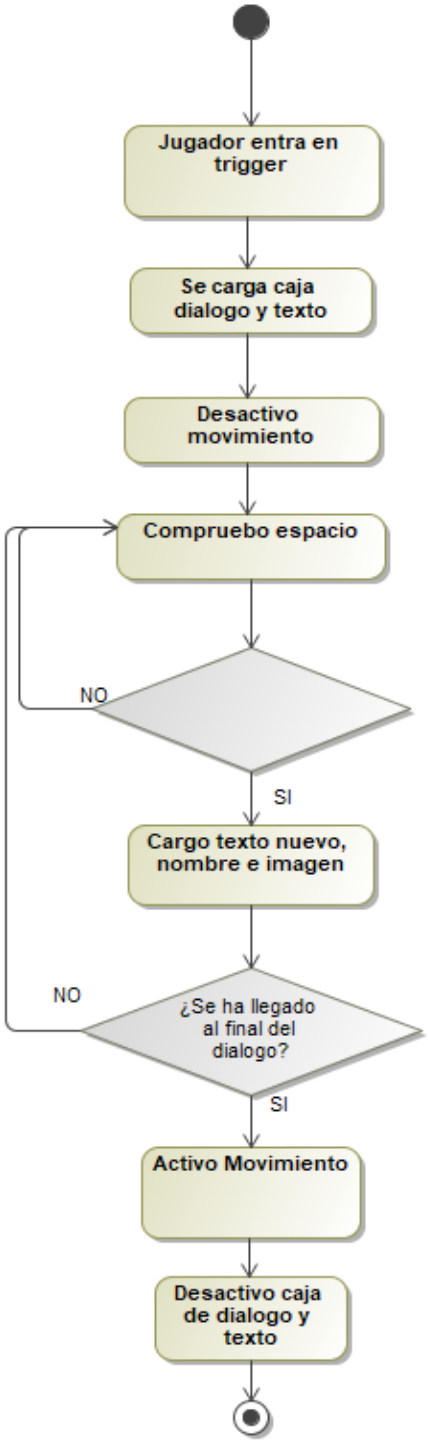

Ilustración 56. Diagrama de secuencia diálogos

5.3.1.3 Niveles

Este diagrama muestra el comportamiento que se produce en los niveles, en los que se reproduce la música, se cuenta el número de monedas y se muestra el mensaje de nivel actual. Tras esto, se comprueba si se está en los niveles 1,3 y 6 que son los niveles que tienen una cinemática adicional para mostrar más de la historia. En caso que no lo sea, se activa el teclado y el movimiento y se mantiene en espera hasta que se recojan las monedas (en caso de existir) se llegue al ordenador o se caiga en uno de los pinchos. Si se llega al PC, habiendo recogido todas las monedas, se reproduce el clip de sonido de nivel completado y se carga el siguiente.

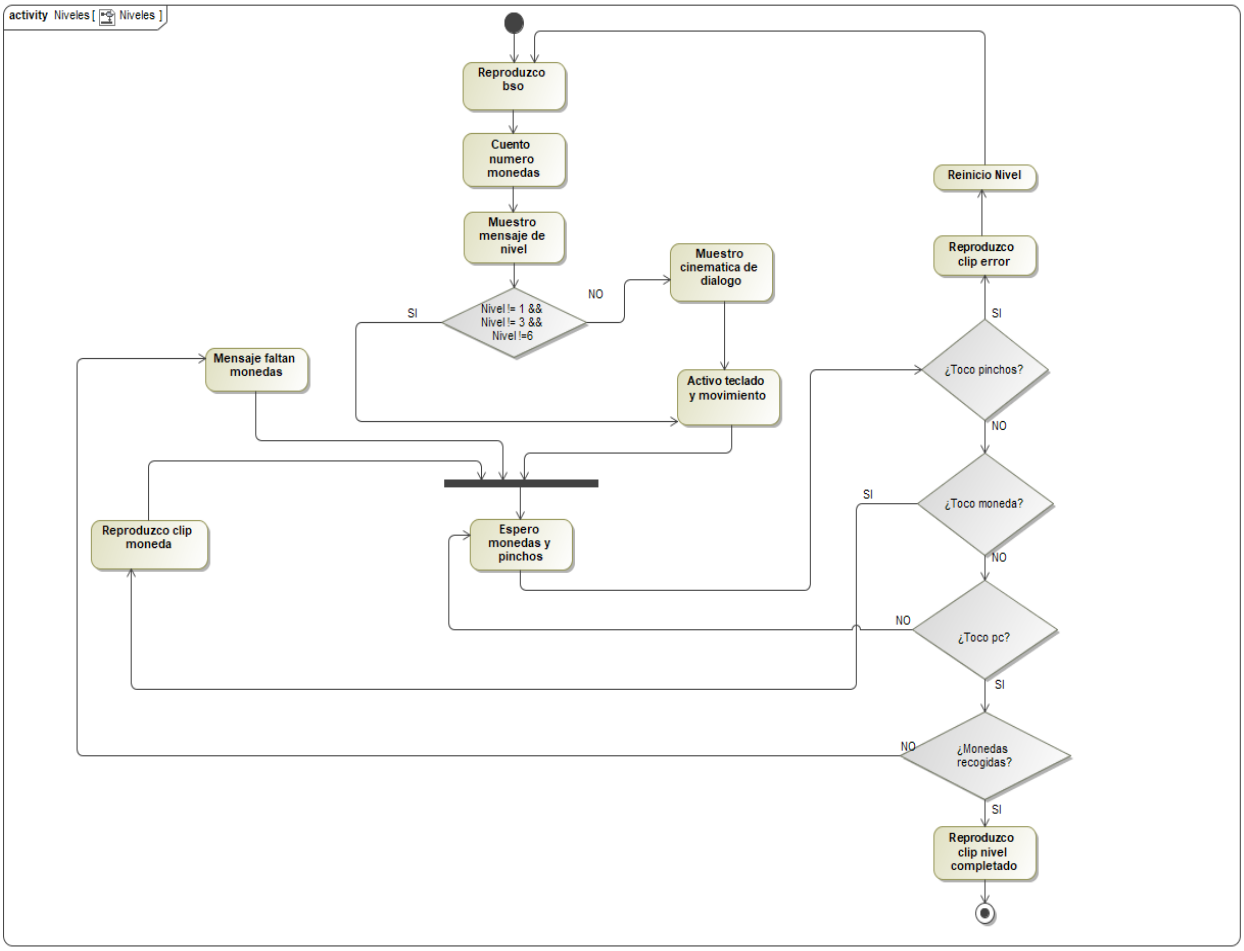

Ilustración 57. Diagrama de secuencia niveles

Mostrado el diseño, vamos a adentrarnos en cómo se ha implementado cada una de las partes del proyecto, desde las interfaces y los paneles de menú, hasta la lógica de movimiento o de resolución de niveles, así como los efectos de sonido, cinemáticas y diálogos de los personajes.

6 IMPLEMENTACIÓN

Con respecto a la implementación, se va a explicar cómo se han realizado las diferentes interfaces gráficas, niveles, mapas, objetos, jugadores y todo el código que implementa la jugabilidad e interacciones de dichos objetos.

6.1 Interfaz Gráfica

Antes de explicar cómo crear la escena de menú del juego vamos a hablar del Canvas, el cual es obligatorio para el diseño de la interfaz de usuario. El Canvas es un Game Object de Unity y representa el área total donde van a estar insertados los elementos de la interfaz de usuario. Por tanto, lo primero que hay que hacer es crearlo y con ello, la imagen de fondo. Haciendo clic derecho en la ventana de Hierarchy (nuestra jerarquía de objetos), clicamos en UI y luego en Image y se nos creara tanto automáticamente el Canvas y la imagen que usaremos de fondo

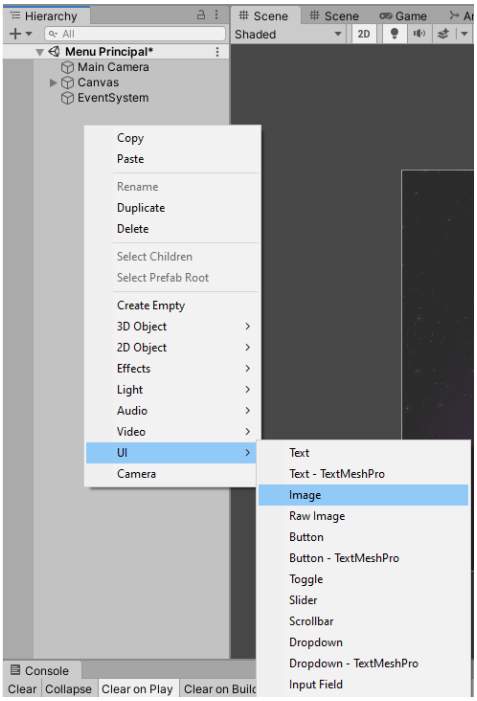

Ilustración 59. Añadiendo imagen fondo

Ahora solo queda sustituir la imagen predefinida por nuestro fondo seleccionado, y modificar la escala de colores si así lo quisiéramos

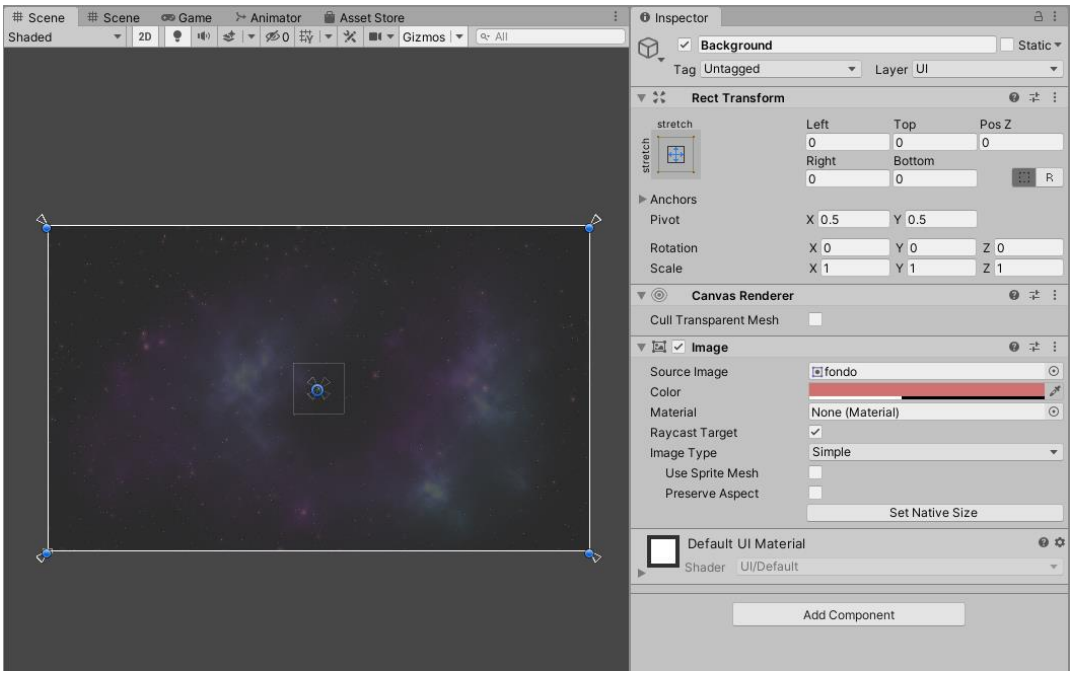

Ilustración 60. Fondo menú principal terminado

Una vez hecho esto, vamos a pasar a la implementación de botones. Primero de todo, vamos a insertarlos en pantalla y luego programaremos el código con la funcionalidad requerida. Para los botones, tenemos que hacer lo mismo que en la ilustración x anterior de añadir la imagen de fondo, pero haciendo clic en Button. El botón contiene, la caja del botón en sí, y el texto contenedor. Eliminamos la imagen (blanca) de la caja del botón cambiando su color por negro, y rellenamos el texto que quiera tener el botón en otro color. El resultado será algo parecido a lo que vemos en la siguiente imagen.

Ilustración 61. Creando el primer botón

Esto lo repetimos para el número de botones que queremos, pero antes de eso, vamos a dar uso a una herramienta que proporciona la Asset Store (la tienda de Unity) y descargamos gratuitamente la herramienta TextMeshPro.

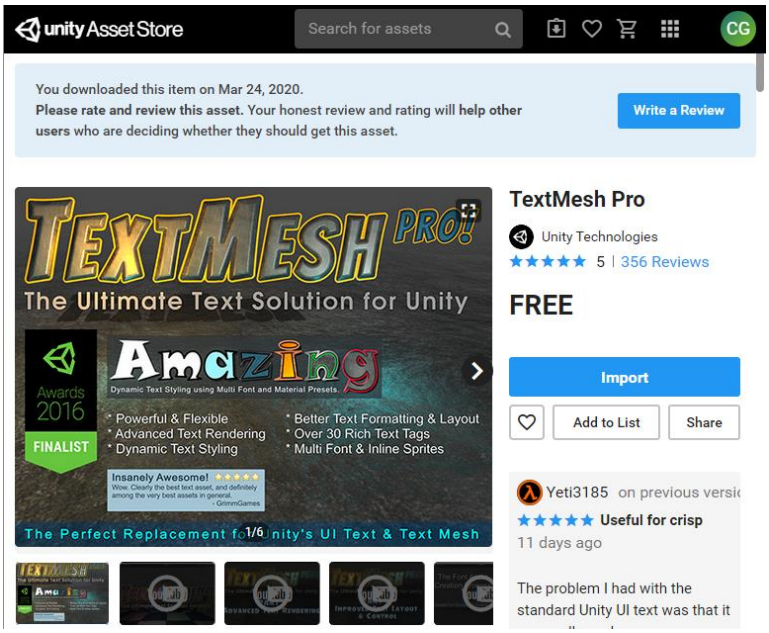

Ilustración 62. Importando TextMeshPro

Una vez hacemos clic en importar, se instala la herramienta, la cual nos permite editar de mejor manera los textos, colores, botones etc. Haciendo cambios en los colores y en las fuentes obtenemos un resultado como el siguiente

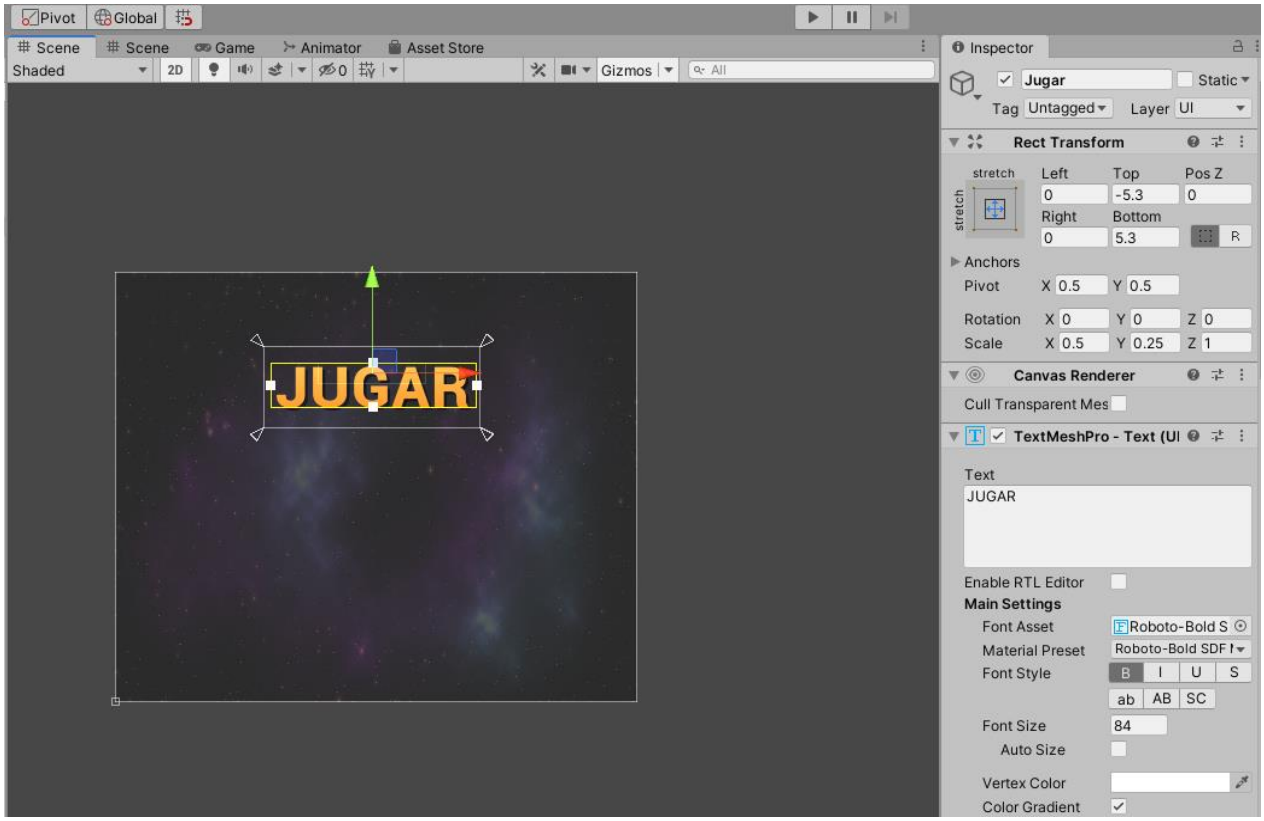

Ilustración 63. Primer botón añadido con estilos de TextMeshPro

Repitiendo esto el número de veces que se quiera, obtendremos un menú parecido al siguiente, donde por ahora, solo tenemos diseñada la interfaz gráfica.

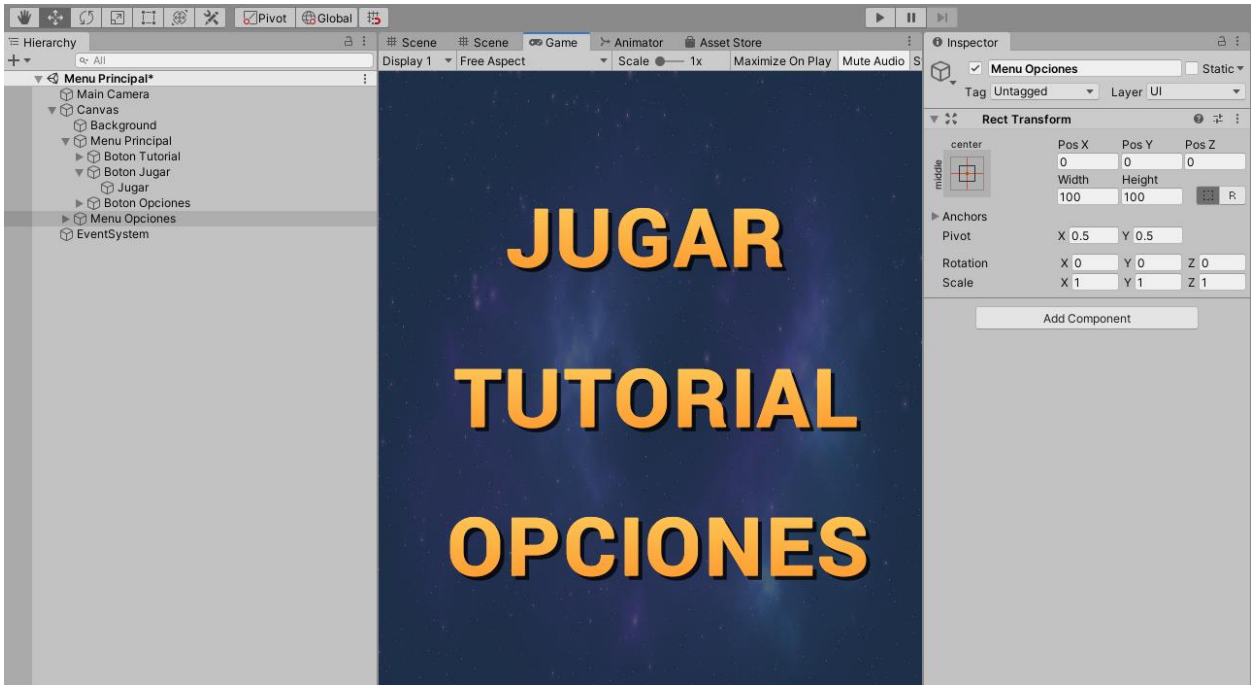

Ilustración 64. Menú de arranque completado

Antes de pasar a explicar el desarrollo del código asociado a dichos botones, vamos a mostrar el menú de opciones que se abre cuando se clica sobre el botón de opciones, desplegando un nuevo menú diseñado de forma similar al menú principal. Este menú nos permite ajustar el sonido usando un slider.

Ilustración 65. Menú de opciones con slider para ajustar sonido

6.3.1 Funcionalidad botones

Se va a explicar como se puede programar para que los botones tengan su funcionalidad de dos maneras más distintas: una completamente usada con código y otra con parte de código y la otra parte usando la interfaz de la propia aplicación Unity, la cual nos simplifica las cosas.

La forma más sencilla es mediante la interfaz de Unity. Para eso clicamos en nuestro botón, por ejemplo en Boton Jugar y en la ventana de Inspector si bajamos hasta el final del apartado Button, se puede apreciar un apartado llamado Onclick() que está vacío. Tenemos que hacer clic en el símbolo '+' para añadir una funcionalidad.

Donde aparece None (Object) debemos arrastrar el Objeto que contiene el script con el código asociado al botón, que en este caso es Menu Principal, que es un objeto vacío que contiene los botones Jugar, Tutorial y Opciones. Una vez arrastrado, donde aparece No Function buscamos el script que contiene el código, y la función a ejecutar que en este caso se llama Jugar()

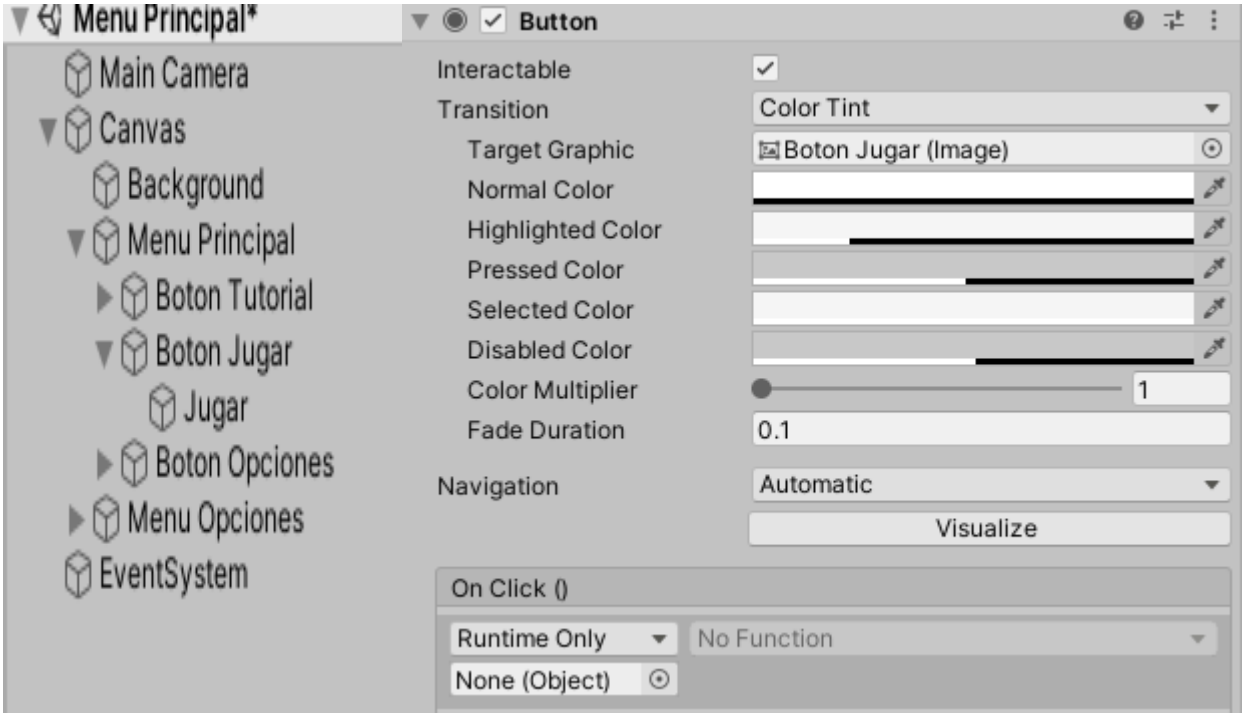

Ilustración 66. Funcionalidad al botón Jugar

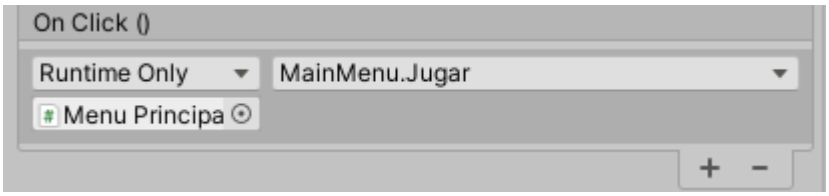

Ilustración 67. Listener botón Jugar

Ahora solo queda definir que hace la función jugar. En este caso lo que queremos es que se cargue la primera escena del videojuego que será la llamada EscenaExterior. El código necesario es el siguiente con los siguientes imports para la gestión de escenas.

El Código lo que hace es obtener mediante el GetActiveScene() la escena actual, y luego carga con el método LoadScene de la clase SceneManager, definida por Unity, la siguiente escena que aparezca en la build. Como van a estar ordenadas según el índice la siguiente escena al menú principal (índice 0) será EscenaExterior (índice 1) y de igual manera con el botón Tutorial y la escena correspondiente (índice 2)

```
using UnityEngine;
using UnityEngine.SceneManagement;

  public void Jugar ()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
    public void Tutorial()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);
    }
```

Ilustración 68. Conexión botones con código

Ilustración 69. Indice de niveles en los ajustes de la build

La segunda forma de asociar el código de los botones es hacerlo completamente por código, creando una variable para cada uno los botones y asociándole un listener al clic que llame a las mismas funciones.

```
    private Button jugar;
    private Button tutorial;
    private Button opciones;

    // Start is called before the first frame update
    void Start()
    {
        jugar = GameObject.Find("Boton Jugar").GetComponent<Button>();
        tutorial = GameObject.Find("Boton Tutorial ").GetComponent<Button>();
        opciones = GameObject.Find("Boton Opciones").GetComponent<Button>();
    }

    // Update is called once per frame
    void Update()
    {
        jugar.onClick.AddListener(() => Jugar());
        tutorial.onClick.AddListener(() => Tutorial());
        opciones.onClick.AddListener(() => Opciones());
     }
```

Ilustración 70. Asociación código a botones completamente por código

6.3.2 Boton pausa en pantalla

Para pausar el juego, además de usando la tecla Escape, se proporciona un botón con un logotipo de pausa en la esquina superior derecha. Al clicar en el, a través del listener, se llama a la función Pausar de la clase MenuPausa.

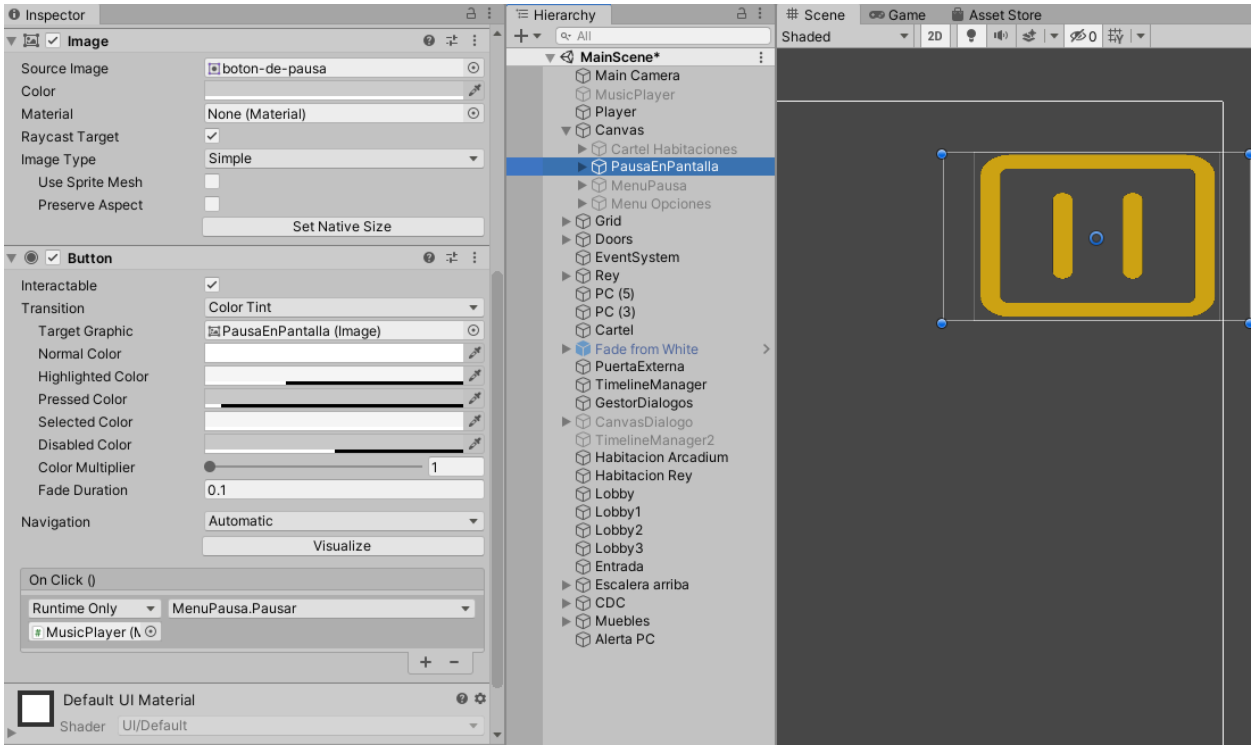

Ilustración 71. Botón pausa en pantalla

```
public void Pausar()
    {
        if (pausar !=null)
            pausar.gameObject.SetActive(false);
        menuPausa.SetActive(true);
        Time.timeScale = 0f;
        enPausa = true;

    }
```

Ilustración 72. Código asociado al botón Pausa

Con el código mostrado anteriormente, se desactiva el botón de pausa en pantalla, ya que no lo queremos dentro del menú pausa, y activamos este último. Además, hacemos que el juego pare por completo, al asignarle como escala de tiempo 0, ya que todo el código depende de los fotogramas y no está detectando ninguno. Por último, asigna el valor true a la variable enPausa para que el programa sepa que estamos ya dentro del menú pausa.

Esta misma función es la que se llama al pulsar Escape desde la función Update, como se muestra a continuación junto con el código de reanudar para comprender como funciona por completo este script.

```
void Update()
  {
      if (Input.GetKeyDown(KeyCode.Escape) && !menuOpciones)
      {

          if (enPausa)
              Reanudar();

          else
              Pausar();

      }
  }
```

Ilustración 73. Código lectura de escape para pausa

```
  public void Reanudar()
  {
      if (pausar != null)
          pausar.gameObject.SetActive(true);
      menuPausa.SetActive(false);
      Time.timeScale = 1f;
      enPausa = false;

  }
```

Ilustración 74. Código asociado al botón reanudar

Reanudar () es el método inverso a Pausar, cuyo fin es reestablecer la escala de fotogramas a su valor por defecto, activa el botón de pausa en pantalla, y desactiva el menú de pausa que se nos había mostrado.

Cada fotograma, la función Update chequea si pulsamos escape, para reanudar o pausar el programa según si está ya parado o no. La otra condición para parar o reanudar, es que no esté mostrado el menú opciones, ya que este ya tiene un botón integrado en pantalla para salir de el y no funciona mediante el escape.

6.3.3 Pantalla de selección niveles

La pantalla de selección de niveles nos permite cargar la partida por donde la dejamos. Para crearla, se ha reutilizado el panel del MenuPrincipal, y cambiado los botones de Jugar, Cargar,Tutorial y Opciones por los botones de los niveles. En cuanto al código, vamos a explicar cómo se realiza la carga y se desbloquean los niveles.

```
if (File.Exists(Application.dataPath + "/save.txt"))
        {
            string cadenaAux = File.ReadAllText(Application.dataPath +
"/save.txt");
            DatosJugador datos = JsonUtility.FromJson<DatosJugador>(cadenaAux);

            botones = this.GetComponentsInChildren<Button>();

            j = datos.nivel;
            for (int k = 1;k<j ; k++)
            {
                imagenes[k - 1] = GameObject.Find("Image" + k);
                if (imagenes[k - 1] != null)
                {
                    contadorImagenes++;
                    imagenes[k - 1].SetActive(false);
                    Debug.Log("Numero de imagenes: " + contadorImagenes);
                }
                else
                    break;
            }
            for ( j = datos.nivel; j < tam; j++)
            {
                botones[j].interactable = false;
            }
        }
        else
            Debug.LogError("No hay archivos de guardado");
```

Ilustración 75. Código selector de niveles

Lo primero que se hace es comprobar si existe, el fichero de guardado “save.txt” y en caso de no existir, se registra en el log de errores el mensaje de “No hay archivos de guardado”.
Si existe un fichero de guardado, se obtienen y almacenan en una variable de la clase DatosJugador, usándose Json para el formateo. Esos datos serán el nivel y la posición (sólo usada en el último nivel). El nivel se usa para eliminar las imágenes de los candados desde la escena inicial hasta la escena en que se guardó, y para hacer que los botones desde el nivel siguiente al guardado hasta el final, no sean interactuables.

6.2 Sonido

Para la música de fondo y los diferentes sonidos se usa un componente Audio Source que se asocia a un GameObject y se configura a través del código y del editor. Este AudioSource reproducirá uno o varios clips en función de las interacciones del jugador. Todos los clips del videojuego se encuentran situados en la carpeta SFX.

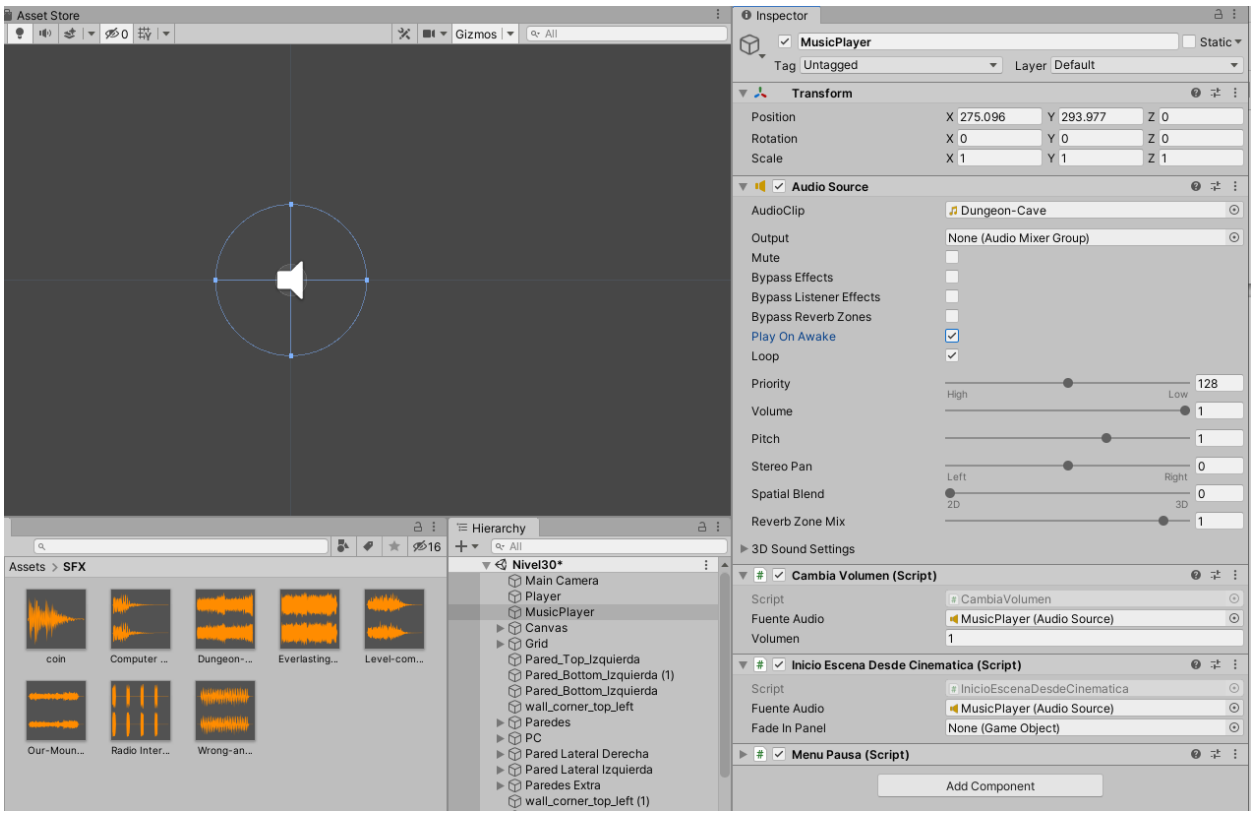

Ilustración 76. Componente AudioSource y diferentes clips en SFX

Como podemos ver en la imagen tenemos un componente Audio Source, en el objeto MusicPlayer (es el nombre que se le ha dado a lo largo de todas las escenas construidas) y que en este caso reproduce la música de fondo en el caso de los niveles, que es diferente a la mostrada en otras escenas. Desde el editor configuramos que se reproduzca en el arranque, marcando Play On Awake, y que se escuche en bucle marcando Loop.

Desde código tenemos dos funciones, una de ellas es CambiaVolumen que se encarga de gestionar el volumen de la música desde el menú opciones dentro del menú pausa y la función IniciaDesdeCinematica. Esta última función se usa tanto para el arranque post cinematicas como para los cambios de escena, y se encarga de esperar a que haya se produzca una animación del tipo Fade explicada anteriormente, y entonces una vez que se ha terminado esta, arrancar el clip indicado.

Eso lo vamos a ver mejor explicando, en la parte de los niveles, en la que se activarán efectos de sonido si fallamos, si cogemos las monedas o si completamos el nivel. Para ello vamos a ver parte del script llamado Niveles, que es el que gestiona todos los niveles. Vamos a centrarnos en la parte de sonido, ya que se explicará posteriormente este script en detalle.

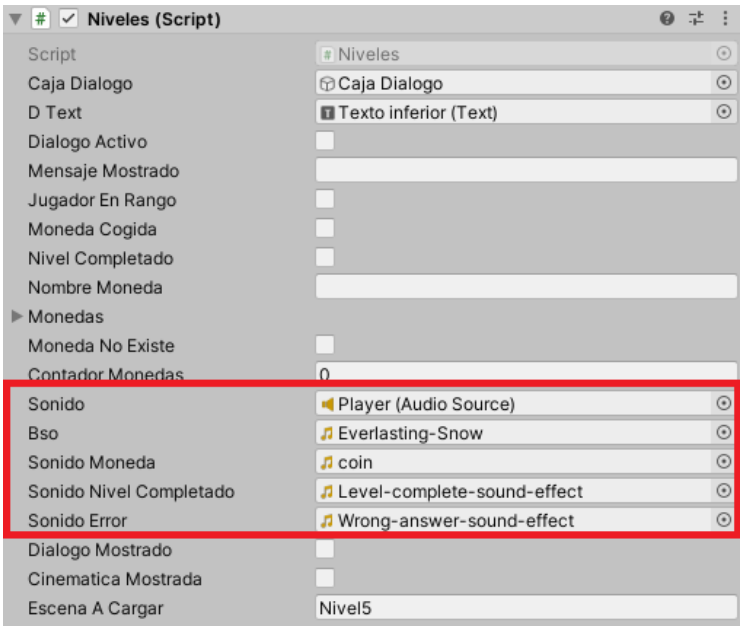

Ilustración 77. Gestion de sonido en script Niveles

Este script se asocia al jugador (GameObject Player) y en el tenemos los diferentes clips que vamos a necesitar en los niveles: la música de fondo, el sonido de las monedas, el sonido de error y el de nivel completado. Además, tendrá asociado un componente AudioSource, en Player también, pero estará vacío ya que en este caso lo configuramos todo desde código.

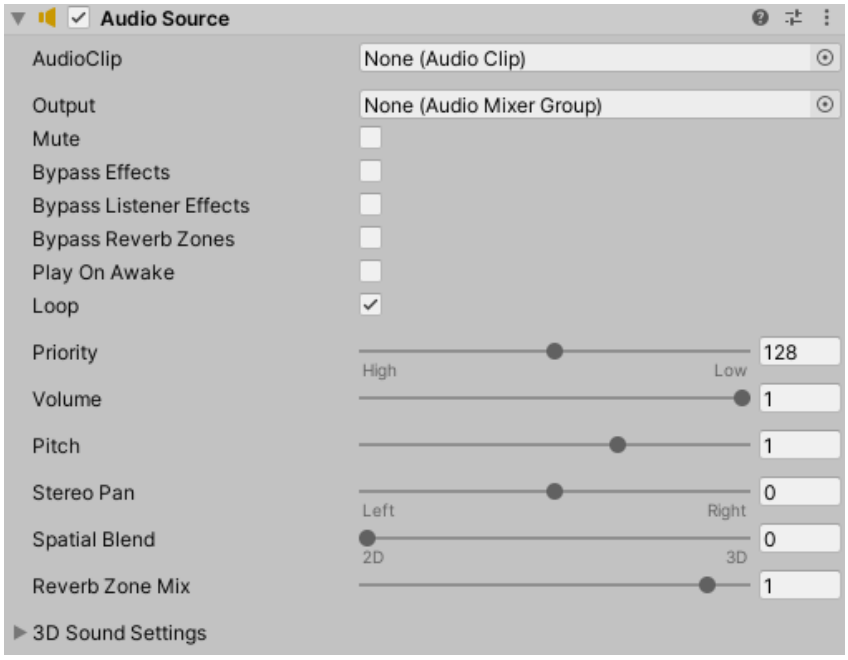

Ilustración 78. AudioSource en player configurada desde código

```
    void Start()
    {

        sonido = GetComponent<AudioSource>();
        sonido.PlayOneShot(bso, 0.2f);
        sonido.loop = true;

    }

private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Pinchos"))
        {

             sonido.PlayOneShot(sonidoError, 0.7f);

        }

        if (collision.CompareTag("PC"))
        {

            sonido.PlayOneShot(sonidoNivelCompletado, 0.7f);
        }

        if(collision.CompareTag("Moneda"))

        {

             sonido.PlayOneShot(sonidoMoneda, 0.7f);

        }

    }
```

Ilustración 79. Código del componente sonido en clase niveles

En el arranque, se obtiene el componente AudioSource que está asociado al objeto Player y se reproduce la música de fondo con la propiedad loop activada, para que se escuche en bucle.

Cuando se entra en una Objeto que sea Trigger, se compara con la etiqueta de ese objeto para ver si es un pincho, un PC o una moneda y se reproduce el clip concreto. Más adelante veremos con mayor detalle el resto del código de este script.

Para terminar, comentar brevemente el script CambiaVolumen en el que se ajusta el sonido desde el menú pausa, con los valores que se dan al botón de tipo slider mostrado anteriormente.

```
public class CambiaVolumen : MonoBehaviour
{

    public AudioSource fuenteAudio;
    public float volumen;

    void Start()
    {

        fuenteAudio = GetComponent<AudioSource>();
        if (fuenteAudio != null)
            fuenteAudio.Play();
    }

  public void SetVolume(float volumen)
    {
        fuenteAudio.volume = volumen;
    }

}
```

Ilustración 80. Código script CambiaVolumen

Al igual que antes, obtenemos la referencia a la fuente de Audio, y si no está desactivada, reproducimos la música de fondo. Una vez que pausamos el juego y accedemos al menú opciones, dentro del menú pausa, tendremos el slider que tiene un listener asociado que llama a la función SetVolume.

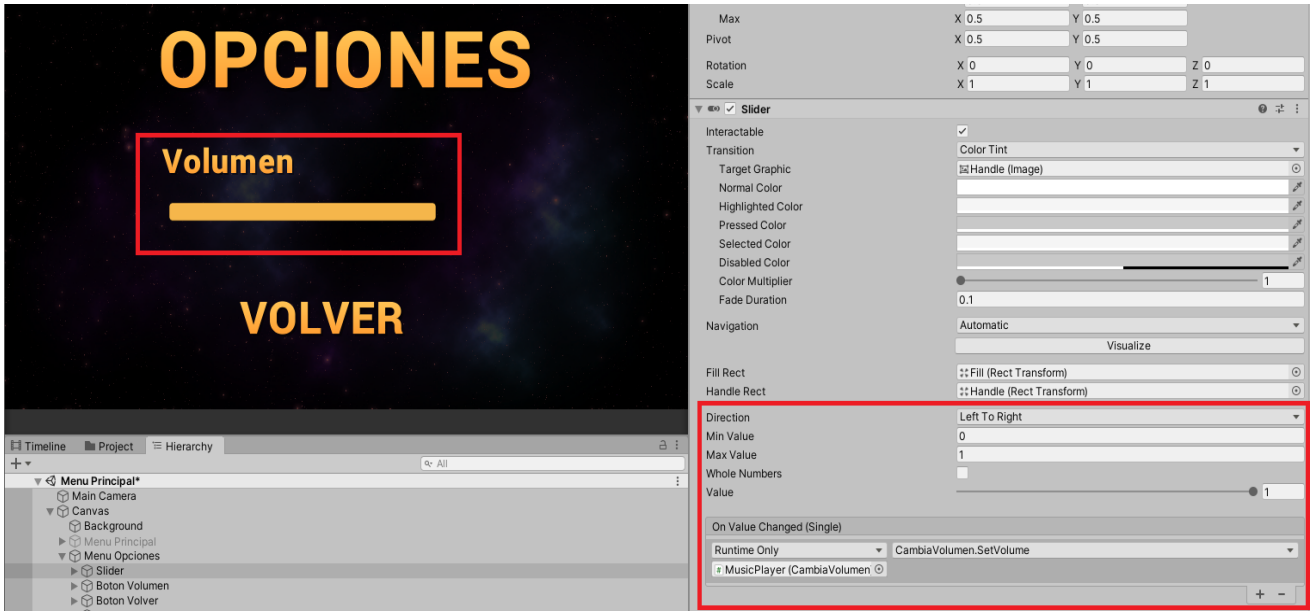

Ilustración 81. Slider con listener asociado a CambiaVolumen

Configuramos desde el editor el slider, de forma que los valores máximos y mínimos se correspondan con los que se pueden asignar al componente AudioSource (de 0 a 1) y que empiece con valor máximo, es decir con el slider completamente a la derecha. Cada vez que hagamos una modificación del slider, llamará a la función con dicho valor como float y lo asignará con la función setvolume que vimos en el código anterior.

6.3 Diseño de mapas

El diseño de mapas, se puede hacer de dos formas, arrastrando directamente los ítems desde la carpeta Asset y colocándolos manualmente en las posiciones que queremos y uniéndolos entre ellos (por ejemplo, en el caso de las paredes) o mediante el Tile Palette. La forma más recomendada de hacerlo y más rápida es mediante el Tile Palette que se explicará a continuación, pero en algunos mapas, era más conveniente hacerlo de la otra forma ya que las colisiones de los Tilemaps en Unity provocan en muchas ocasiones errores.

La forma más básica, consiste pues en arrastrar los sprites a la escena actual, y posicionarlos correctamente. Es conveniente usar el Grid, y colocar los sprites en coordenadas con números exactos para mayor facilidad (por ejemplo, x= 0, y= 0; o x= 1,5 y= 0,5).

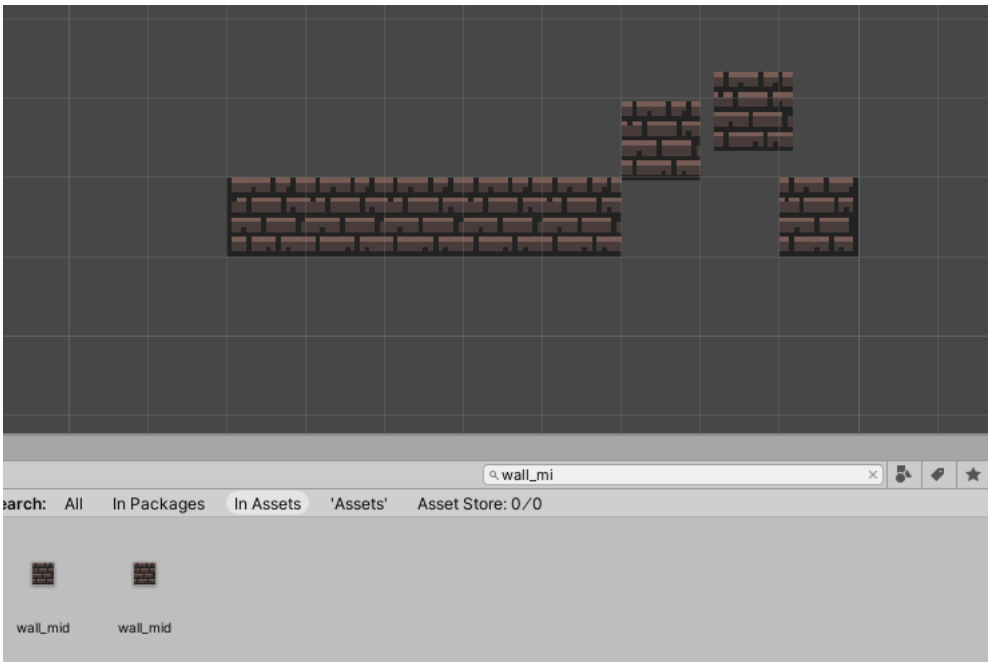

Ilustración Ç¨BNIKÇ&/T%82. Diseño de mapas mediante arrastre de sprites

Usando las herramientas de movimiento de Unity, por ejemplo, pulsando la tecla V podemos seleccionar dos esquinas que se quieren unir, completando por tanto la pared de la ilustración anterior. De esta forma se han construido algunos de los prefabs de paredes, para reutilizarlos en diferentes niveles.

6.4.1 Prefabs de mapas

De esta forma se han construido algunos de los prefabs de paredes, para reutilizarlos en diferentes niveles. Consiste en la creación de los mapas y de las paredes uniendo uno a uno los bloques de sprites. Una vez que tenemos el bloque conjunto que queremos (por ejemplo, una pared superior), creamos un prefab que es como una especie de plantilla en la que se agrupan varios sprites para poder reutilizarlo como si fuera un único bloque

Como se ha comentado, no es la forma más rápida, pero debido a los problemas que dan las colisiones en los Tilemaps, se han usado para los niveles, haciendo que la colisión del jugador al introducir por teclado las órdenes,

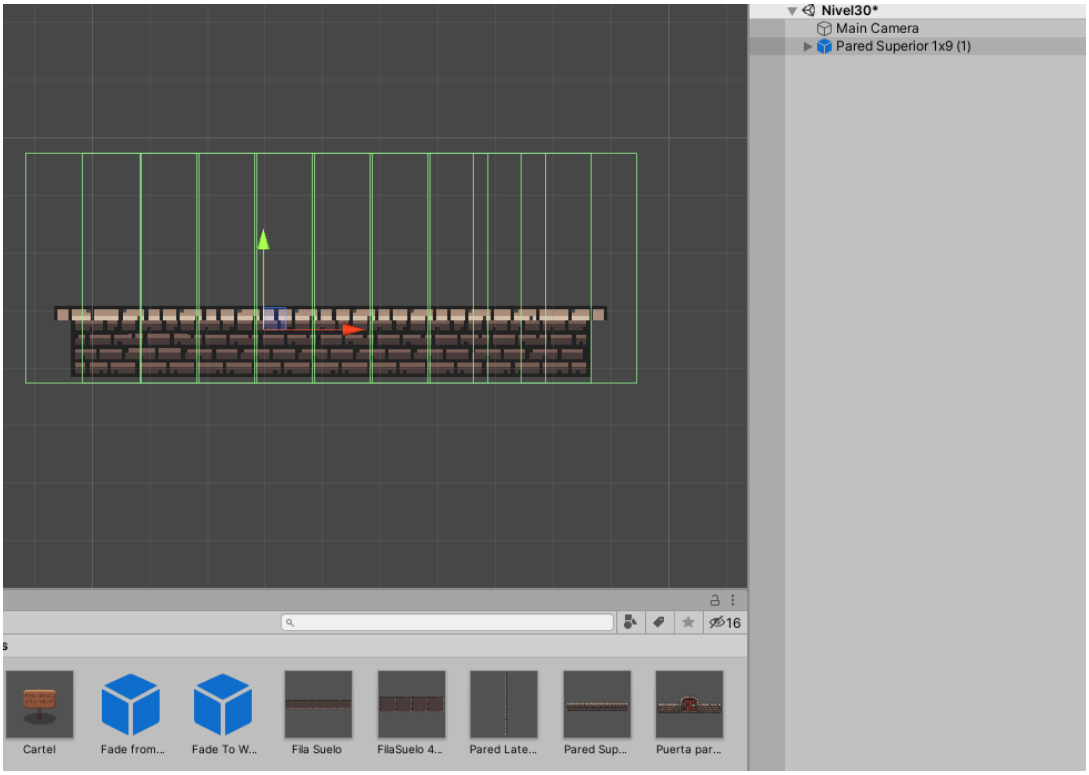

Ilustración 83. Prefab Pared Superior

Usando los prefabs de pared superior y paredes laterales y usando las herramientas de movimiento, rápidamente obtenemos paredes como las siguientes con la colisión ya incorporada en el prefab.

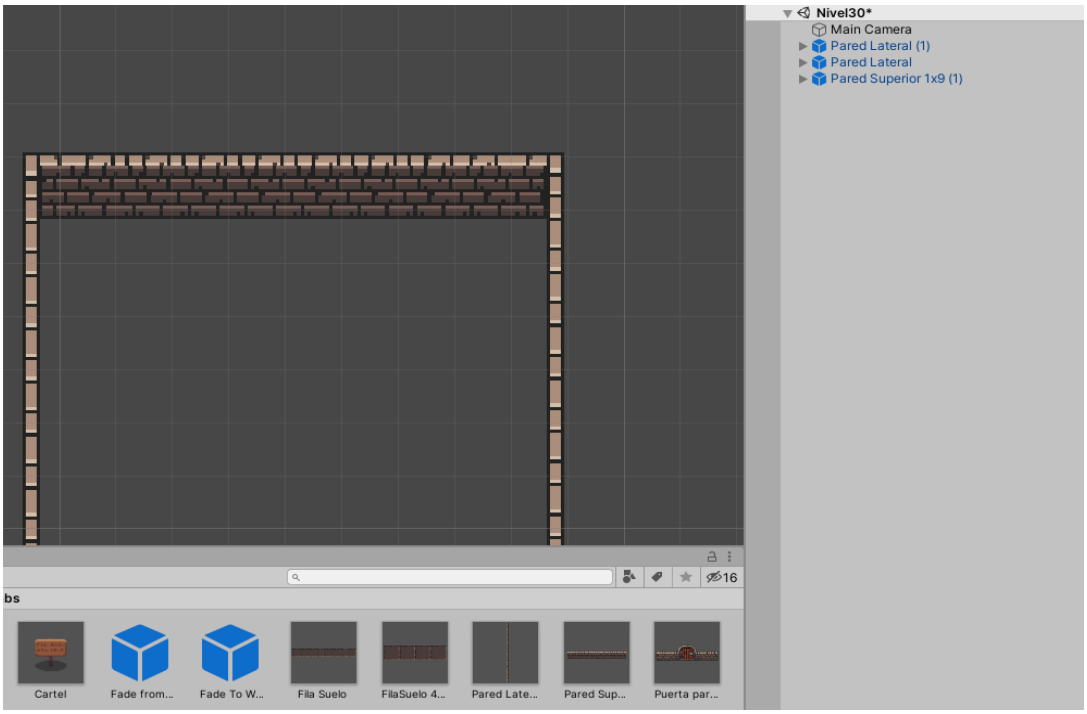

Ilustración 84. Ejemplo habitación cerrada7

Para el suelo se pueden reusar prefabs de varios tamaños, pero al no necesitar colisión es mucho mejor hacerlo con el Tile Palette.

6.4.2 Tilemaps y Tile Palette

Continuando con el ejemplo anterior vamos a explicar cómo usar el Tile Palette y terminar el suelo y mostrar posteriormente como se han creado algunos de los mapas enteramente usando esta herramienta. El Tile Palette (paleta de losas o azulejos) es una herramienta usada en Unity para la creación de mapas 2D, en el que los diferentes sprites que se usan para los mapas, se usan como si fuesen paletas de pintura, simplificando el desarrollo de mapas, sobre todo para el suelo.

Para usar el Tile Palette, lo primero que tenemos que hacer es crear el Tilemap, haciendo clic derecho en la jerarquía de la escena actual y en 2D object clicamos en Tilemap como se puede apreciar en la siguiente imagen.

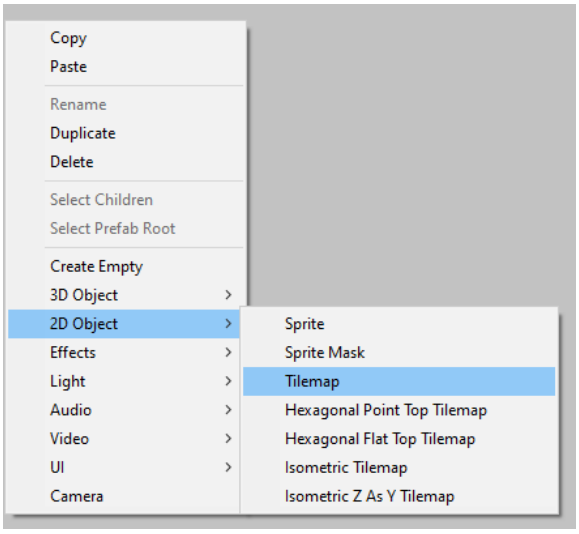

Ilustración 85. Creación del Tilemap

El Tilemap, nos crea automáticamente un nuevo grid que será el contenedor de los diferentes tilemaps y uno de ellos por defecto llamado Tilemap. Lo vamos a renombrar por Floor ya que será el que contenga exclusivamente el suelo. Una vez hecho esto, en la barra de herramientas superior, dentro de Window, en la pestaña 2D hacemos clic en Tile Palette. Esto nos abrirá una pestaña nueva con la herramienta de TilePalette.

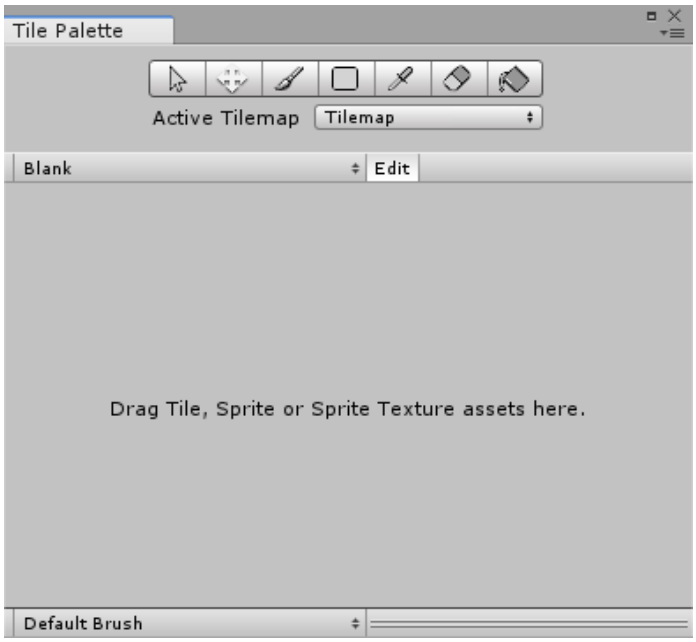

Ilustración 86. Herramienta Tille Palette

Donde nos dice Blank, vamos a crear una nueva paleta y la guardaremos en una carpeta, para ello hemos creado en el proyecto una denominada Tile Palette y dentro de ella hemos creado subcarpetas. No es necesario hacer esto, se pueden añadir todos los objetos a una sola paleta, pero es recomendable tener varias paletas distribuidas en subcarpetas para diferentes tipos de sprites. Por tanto, creamos una llamada Ground Dungeon que va a representar el suelo de la mazmorra. Una vez hecho esto, debemos arrastrar desde Asset, los sprites que queremos, que en este caso son los del suelo.

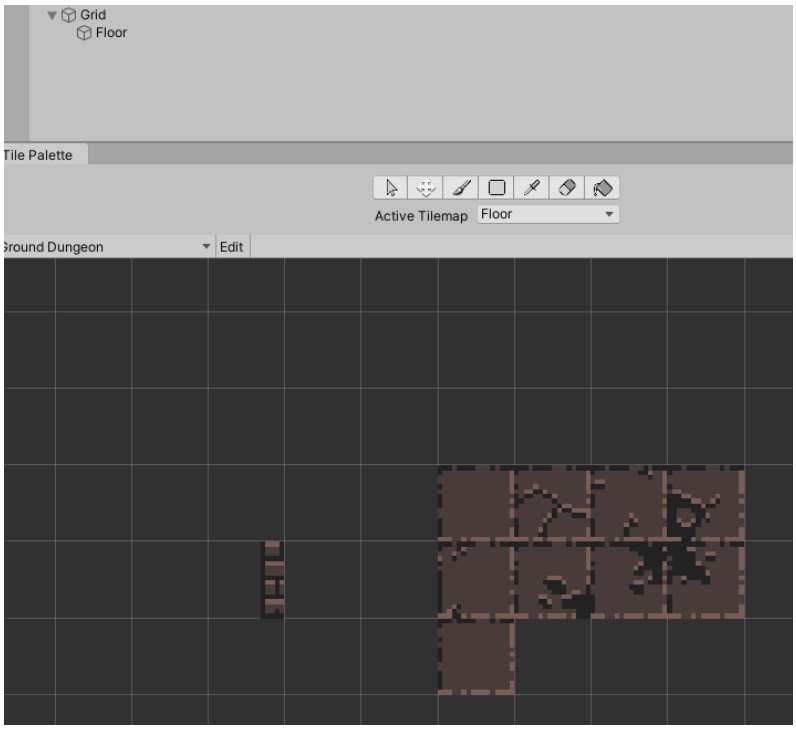

Ilustración 87. Tile Palette Ground Dungeon con Tilemap Floor

Una vez tenemos esto, tenemos que elegir el Sprite que queremos pintar, y podemos usar la herramienta del pincel para pintarlo cuadro a cuadro, o podemos seleccionar la herramienta caja de relleno (la herramienta rectangular) para rellenar directamente todo el suelo. El resultado será el siguiente.

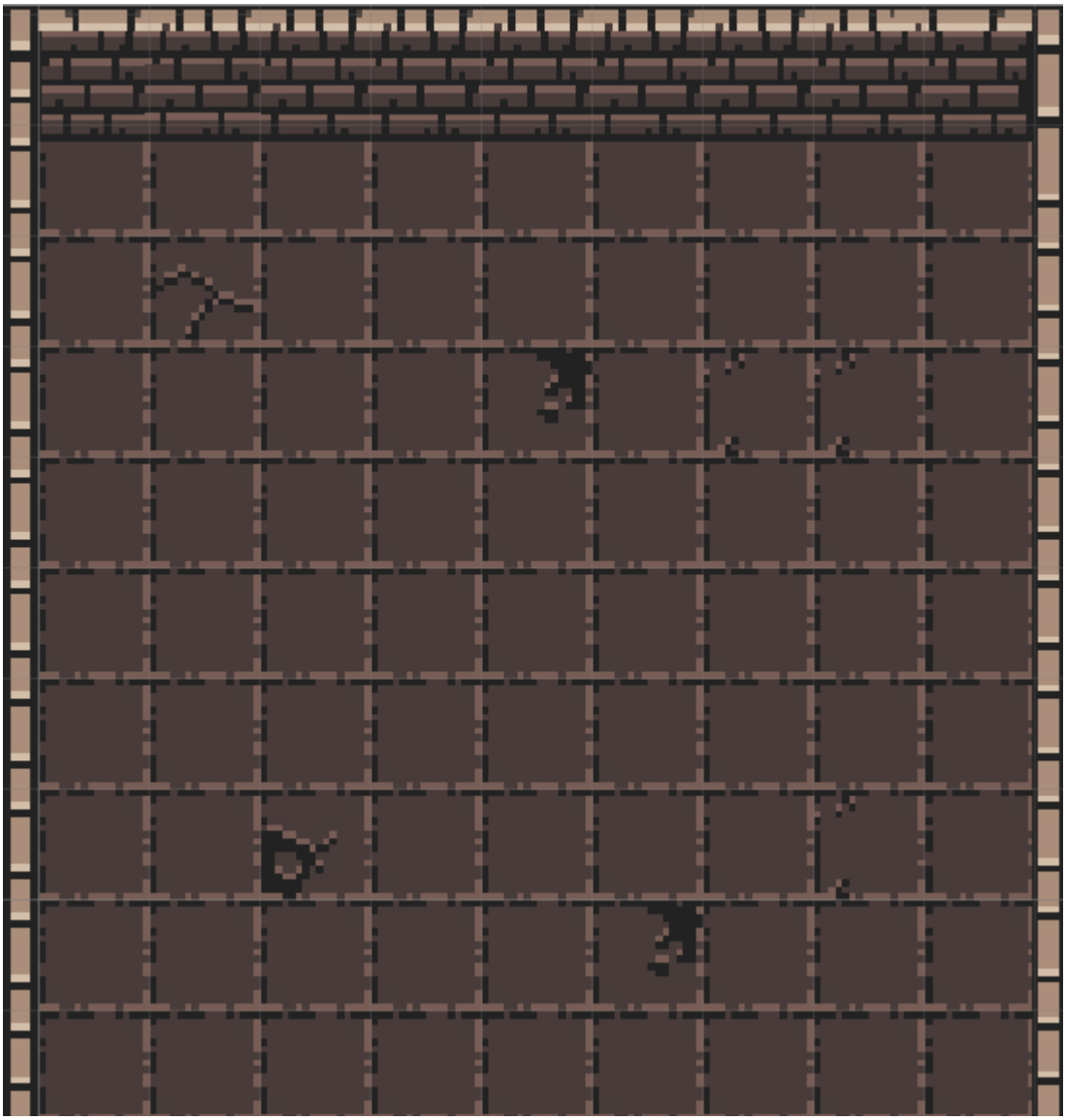

Ilustración 88. Relleno con Tile Pallette

Para terminar, vamos a ver la aplicación de los Tilemaps a un mapa completo y más extenso con diversas capas, con colisión incluida como es el caso de la primera escena (EscenaExterior) y la escena principal (MainScene)

En EscenaExterior, tenemos 3 capas: ground, colisión e intermedia. Vamos a ir seleccionando desde el inspector que el editor nos la enseñe cada una por separado para entender mejor como funciona cada una de ellas.

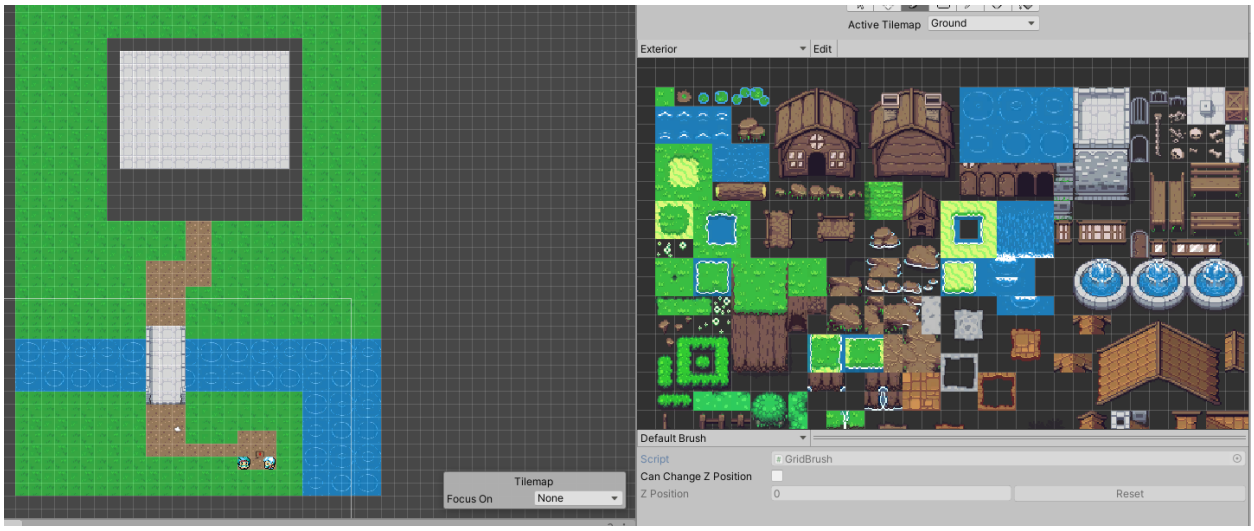

Ilustración 89. Capa Ground en EscenaExterior

Importante resaltar que el orden de la capa en el inspector debe ser inferior a las otras para que efectivamente actue de suelo y este por debajo del resto de capas siguientes.

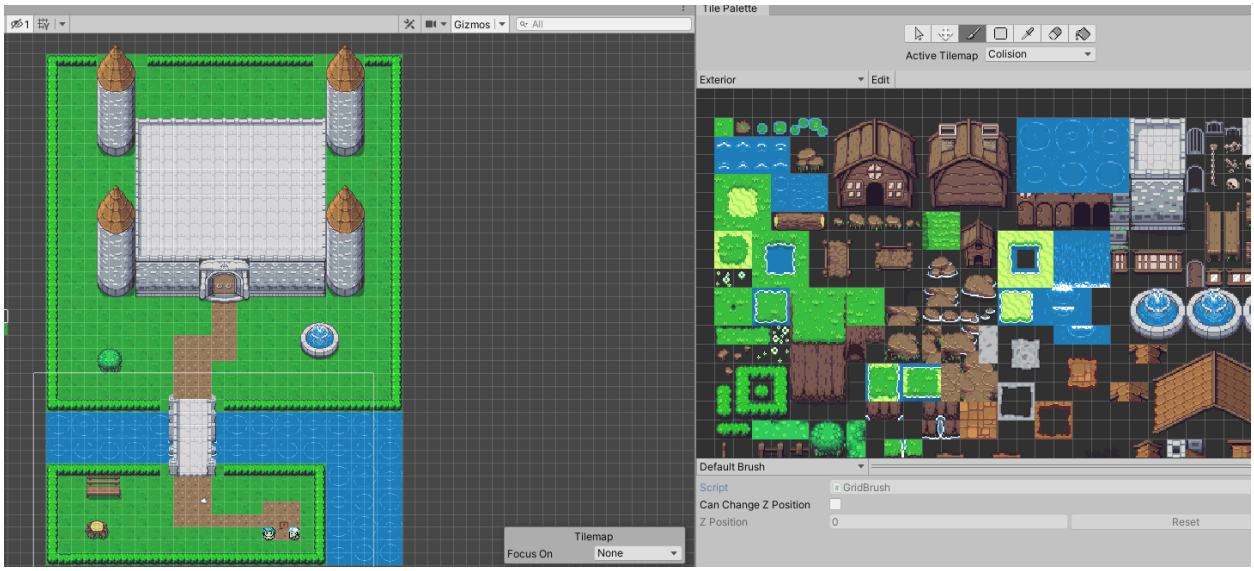

Ilustración 90. Añadimos capa Colision

Al mostrar la capa colisión vemos lo que aparece en la ilustración anterior. En esta capa están todos los objetos que formen parte de paredes, estructuras, construcciones, etc.

A esta capa se le ha puesto un orden de capas igual a 2, para situar en medio la capa intermedia que se explicará después para que se usa.

A la capa de colisión hay que añadir 3 componentes: Tilemap Collider 2D, RigidBody2D y Composite Collider.

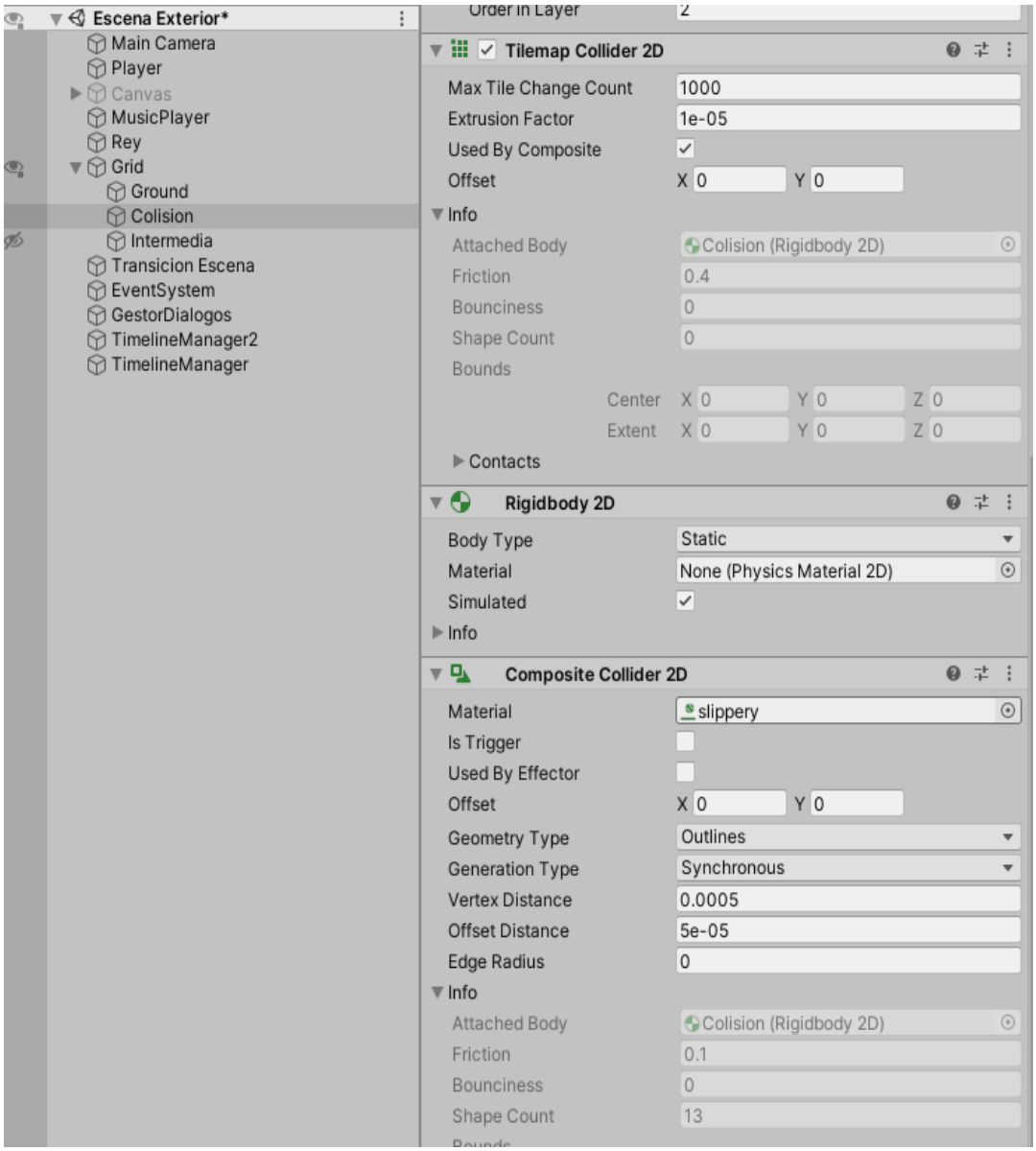

Ilustración 91. Componentes necesarios en capa Colision

Estos componentes son fundamentales para las colisiones. Con el componente Rigidbody2D estamos diciéndole a Unity que todo lo que pintemos con esa capa va a ser un objeto rígido y debemos cambiar el tipo de cuerpo de dinámico (por defecto) a estático ya que forman parte de estructuras.

El Tilemap Collider se encarga de generar las formas para cada componente del Tilemap y debemos marcar la opción Used by Composite para adjuntarlo al Composite Collider y este sea el que gestione sus colisiones.

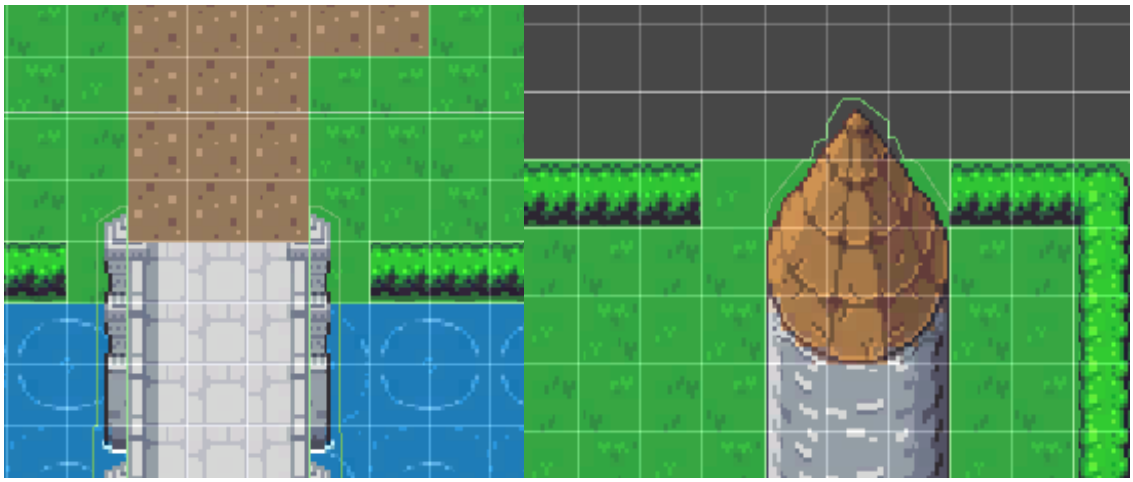

Ilustración 92. Necesidad de capa intermedia

La capa intermedia se usa para corregir lo que se ve en la anterior imagen. Hay diferentes sprites, que no cubren un cuadrado entero del grid, o simplemente hay situaciones en las que se necesita que debajo de un Sprite con colisión (caso del castillo) necesitas otro en una capa inferior sin que se reemplace. Esta capa (o capas) puede tener colisión o puede ser meramente estético para terminar el Tilemap como debería. Si activamos la capa intermedia vemos como el resultado final es el esperado.

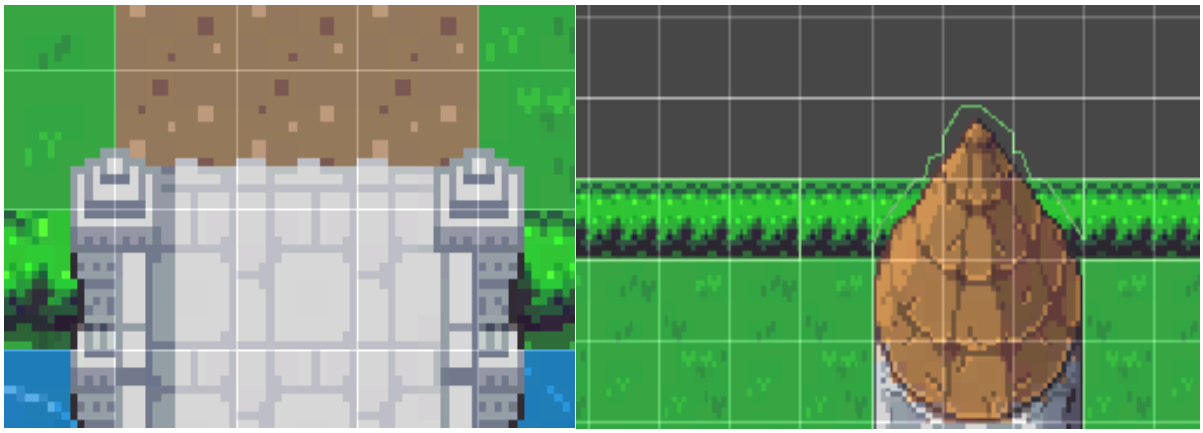

Ilustración 93. Tilemap solucionado con capa intermedia

Para resumir, y terminar de aclarar, en esta escena concreta, tenemos la capa Ground (o Floor) con un orden de capa igual a 0, una capa intermedia con orden de capa igual a 1 y una capa colisión con orden de capa igual a 2. Esto es lo que hace que haya una prioridad entre capas y por eso consigamos el resultado de la ilustración anterior.

6.4 Jugabilidad

En este apartado se van a explicar los fundamentos de la jugabilidad, como son los dos tipos de movimiento existentes y las mecánicas de los niveles.

6.4.1 Movimiento

En las escenas a lo largo del juego se diferencian dos tipos de movimiento, como se ha comentado a lo largo de este documento. Se alternan escenas de movimiento libre para que el jugador explore los diferentes rincones del mapa y se sienta más integrado en la historia, así como los niveles en los que el movimiento se consigue a través de ordenes enviadas por teclado, que es la mecánica fundamental para avanzar y terminar el videojuego.

Vamos a empezar por el script de MovimientoLibre, ya que es el más sencillo de los dos. Este script está leyendo cada fotograma la entrada de teclado para detectar si se pulsan las teclas de movimiento separándolas en vertical (W y S; flecha arriba y flecha abajo) y horizontal (A y D; flecha izquierda y flecha derecha). Además, se lee si se pulsa la tecla Shift Izquierdo para duplicar la velocidad del personaje.

```
void Update()
  {
      cambio = Vector3.zero;
      cambio.x = Input.GetAxisRaw("Horizontal");
      cambio.y = Input.GetAxisRaw("Vertical");
      ActualizaAnimacionYMueve();
      if (Input.GetKeyDown(KeyCode.LeftShift))
          velocidad = 10f;
      if (Input.GetKeyUp(KeyCode.LeftShift))
          velocidad = 5f;

  }
```

Ilustración 94. Código lectura teclado para movimiento

Una vez leída la entrada, se llama a la función ActualizaAnimaciónYMueve () la cual llama a la función MoverPersonaje, que se encarga de mover al personaje en la dirección pulsada, y posteriormente reproduce la animación correspondiente. Las animaciones se explicarán en el siguiente apartado.

```
void ActualizaAnimacionYMueve()
    {
        if (cambio != Vector3.zero)
        {
            MoverPersonaje();
        }
    }
```

Ilustración 95. Código movimiento personaje

```
  void MoverPersonaje()
    {
        if (!this.jugadorEnDialogo)
        {
            myRigidbody.MovePosition(transform.position + cambio * velocidad *
Time.deltaTime);
        }
    }
```

Ilustración 96. Código función MoverPersonaje

Si el jugador no está en un diálogo, se actualiza la posición. La variable jugadorEnDialogo será actualizada a través de la clase GestorDialogos que se detallará posteriormente y con los gestores de cinemáticas que llaman a funciones de activar movimiento cuando terminan.

La función MoverPersonaje, se base en el método MovePosition de la clase RigidBody2D, a la cual se llama con la posición actual del jugador, a la cual se le suma la velocidad multiplicada por deltaTime y por cambio. Cambio representa la dirección de cambio que se ha leído antes en el método update y deltaTime es el tiempo en que se han estado pulsando las teclas de movimiento.

Para el caso del movimiento por las órdenes de teclado, el código difiere un poco, ya que la variable cambio no se obtiene directamente de las teclas pulsadas de teclado, si no que cada botón o acción introducida, tiene asociada una función.

```
public void MoverPersonajeArriba()
    {

  if (!this.jugadorEnDialogo)
        {

         cambio.x = 0;
         cambio.y = 1;
         myRigidbody.MovePosition(myRigidbody.position+cambio);

        }
```

Ilustración 97. Código MovimientoOrden

De igual manera y cambiando las coordenadas, se completan las funciones MoverPersonajeIzquierda, MoverPersonajeDerecha y MoverPersonajeAbajo.

Además, se utiliza la función desactiva movimiento, llamada al reproducir las cinemáticas para bloquear el movimiento del personaje mientras se están produciendo los diálogos.

```
public void DesactivoMovimiento()
    {
        this.jugadorEnDialogo = true;
    }
```

Ilustración 98. Código DesactivoMovimiento

6.4.2 Teclado

Las funciones del teclado usan las funciones de la clase MovimientoOrden, además de actualizar las animaciones del personaje.

```
public void MoverArriba()
{
    {
        anim.SetFloat("moverY", 1.0f);
        anim.SetFloat("moverX", 0.0f);
        movimiento.MoverPersonajeArriba();
    }
}


public void MoverDerecha()
{
    {
        anim.SetFloat("moverX", 1.0f);
        anim.SetFloat("moverY", 0.0f);
        movimiento.MoverPersonajeDerecha();
    }
}
public void MoverAbajo()
{
    {
        anim.SetFloat("moverX", 0.0f);
        anim.SetFloat("moverY", -1.0f);
        movimiento.MoverPersonajeAbajo();
    }
}
public void MoverIzquierda()
{
    {
        anim.SetFloat("moverX", -1.0f);
        anim.SetFloat("moverY", 0.0f);
        movimiento.MoverPersonajeIzquierda();
    }
}
```

Ilustración 99. Código de movimiento desde el teclado

6.4.3 Niveles

Vamos a detallar aquí la clase Niveles, que gestiona toda la lógica de los niveles que hay que resolver para completar el videojuego.

```
    void Start()
      {
          componenteSonido = GetComponent<Sonido>();
          cinematicaMostrada = false;
          monedas = new GameObject[tam];
          contadorMonedas = 0;
          monedaCogida = false;

          nivelCompletado = false;

          dialogoActivo = false;

          CuentaMonedas();


          gestorDialogo.EliminaContinuarSuperior();
          if (SceneManager.GetActiveScene().name != "Nivel3" &&
    SceneManager.GetActiveScene().name != "Nivel1")
              StartCoroutine(MuestraMensajeNivel());
          else
          {
              StartCoroutine(MuestraMensajeNivelConCinematica());
          }

      }
```

Ilustración 100. Código arranque script Niveles

Tras preparar las diferentes variables necesarias, el programa empieza invocando la función CuentaMonedas() que detecta si hay monedas o no en el nivel , y cuantas en caso de que existan.

```
void CuentaMonedas()
    {
        for (int k = 1; ; k++)
        {
            monedas[k - 1] = GameObject.Find("Moneda" + k);
            if (monedas[k - 1] != null)
            {
                contadorMonedas++;
            }
            else
                break;

        }

        if (monedas[0] == null)
        {
            monedaNoExiste = true;
        }
        else
            monedaNoExiste = false;

    }
```

Ilustración 101. Función CuentaMonedas en niveles

Este método busca los objetos que haya en el nivel cargado con los diferentes nombres Moneda más el número de moneda con el que están organizados y las almacena en la variable contadorMonedas.

Una vez termina, se produce la llamada a la función que muestra el mensaje del nivel actual en el que estamos. Aquí tenemos dos posibilidades: si el nivel actual es el 1,3, o 6, se llama a la función MuestraMensajeNivelConCinematica() y en caso contrario a la función MuestraMensajeNivel(). Esto es así porque en estos niveles se produce una cinemática con un breve diálogo para darle profundidad a la historia. Se va a mostrar la función MuestraMensajeNivelConCinematica() ya que es muy parecido al otro método con la diferencia de que MuestraMensajeNivel() no tiene la última línea que invoca al gestor de cinemáticas reproduciendo el clip que contiene el dialogo.

```
IEnumerator MuestraMensajeNivelConCinematica()
  {
      gestorDialogo.MuestraMensaje();
      dialogoActivo = true;
      yield return new WaitForSecondsRealtime(3);
      gestorDialogo.EliminaTextoSuperior();
      dialogoActivo = false;
      cajaDialogo.gameObject.SetActive(true);
      gestorDialogos.gameObject.SetActive(true);
      cinematicasManager.StartTimeline();
  }
```

Ilustración 102. Función MuestraMensajeNivelConCinematica

Importante mencionar sobre el tipo de la función. Al ser Unity basado en fotogramas, para algunas funcionalidades, necesitamos que el programa se pare en una función sin que el programa entero quede paralizado. Por ejemplo, para los bucles While, que se usan en los diálogos, si no usamos una función de tipo IEnumerator, el programa se quedará colgado. En este caso, usamos una función de este tipo, porque la intención es que, tras mostrar el mensaje de nivel, mediante la función MuestraMensaje de la clase GestorDialogo, (que trataremos en próximos apartados) se produzca una espera de unos 3 segundos antes de mostrar la cinemática, o en caso de que no se trate de los niveles 1,3 o 6, antes de mostrar el teclado y botones.

Tras esto, se limpia el texto superior con el mensaje de nivel, y se activa la caja inferior de diálogos para darle paso al clip de diálogo y que se muestre correctamente. Una vez terminado el clip de diálogo, se activa el teclado para poder completar el nivel.

La mayoría del código se gestiona en la función OnTriggerEnter2D(), con diferentes triggers a lo largo de los mapas, para los objetos de tipo PC, moneda o pinchos.

```
    private void OnTriggerEnter2D(Collider2D collision)
        {
            if (collision.CompareTag("Pinchos"))
            {
                mensajeMostrado = mensajeFallido;
                gestorDialogo.MuestraMensaje(mensajeMostrado);
                componenteSonido.ReproduceSonido(ERROR, 0.7f);
                dialogoActivo = true;
                StartCoroutine(ReiniciaNivel());
            }
            if (collision.CompareTag("PC"))
            {
                if (monedaCogida || monedaNoExiste)
                {
                    mensajeMostrado = mensajeCompletado;
                    gestorDialogo.MuestraMensajeFinal(mensajeMostrado);
                    componenteSonido.ReproduceSonido(COMPLETADO, 0.7f);
                    nivelCompletado = true;

                }
                else
                {
                    mensajeMostrado = mensajeMoneda;
                    gestorDialogo.MuestraMensaje(mensajeMoneda);
                }
            }
            if(collision.CompareTag("Moneda"))
            {
                contadorMonedas--;
                if (contadorMonedas == 0)
                    monedaCogida = true;
                nombreMoneda = collision.gameObject.name;
                int i = (int) Char.GetNumericValue(nombreMoneda[13]);
                Debug.Log(i);
                monedas[i-1].SetActive(false);
                componenteSonido.ReproduceSonido(MONEDA, 0.7f);

            }
        }
```

Ilustración 103. Código de detección de colisiones

Si la colisión se produce con un Pincho, se carga en mensajeMostrado la constante mensajeFallido con el mensaje de fallo, y se muestra el mensaje. Se reproduce el sonido correspondiente como se ha explicado anteriormente y se llama a la función ReiniciaNivel que reinicia tras una breve espera para poder leer el mensaje de nivel fallido.

Si la colisión se produce con el PC, se comprueba primero si se han recogido todas las monedas o no existían en el nivel actual, lo que significaría que el nivel se ha completado con éxito. En ese caso se carga el mensaje de nivel completado, y se muestra reproduciendo el sonido. Si no se cumplió la condición de que no se habían recogido las monedas entonces se muestra el mensajeMoneda que informa al usuario de que debe recoger todas las monedas antes de superar el nivel.

Si la colisión se produce con una moneda, se reduce el contador de monedas, y se elimina desde código la instancia que tenemos en la escena actual, haciendo que la moneda desaparezca y reproduciendo el sonido de moneda recogida.

Una vez que se llegó a completar el nivel, se cambió el valor de nivelCompletado a true de manera que la función fixedUpdate() lo detectará llamando a la función cambiaEscena() tras esperar unos segundos para poder leer el mensaje y escuchar el sonido de nivel superado con éxito.

```
void FixedUpdate()
    {

        if (nivelCompletado)
        {
            StartCoroutine(CambiaEscena());
        }


    }



IEnumerator CambiaEscena()
    {

        yield return new WaitForSecondsRealtime(4);
        SceneManager.LoadScene(this.escenaACargar);

    }
```

Ilustración 104. Código para cambiar de escena al fin de nivel

6.5 Animaciones

Este capítulo va a describir cómo se consigue el funcionamiento de las animaciones, particularizadas para nuestro personaje Arcadium, y su movimiento.

6.5.1 Clips de animación

El primer paso para crear las animaciones es la creación de los clips de las diferentes animaciones. Para ello abrimos la herramienta Animation en la barra de herramientas superior, clicando en Window, después en Animation y finalmente clicamos de nuevo en Animation Al clicar sobre el personaje, o el Sprite que queramos animar veremos algo similar a la siguiente ilustración.

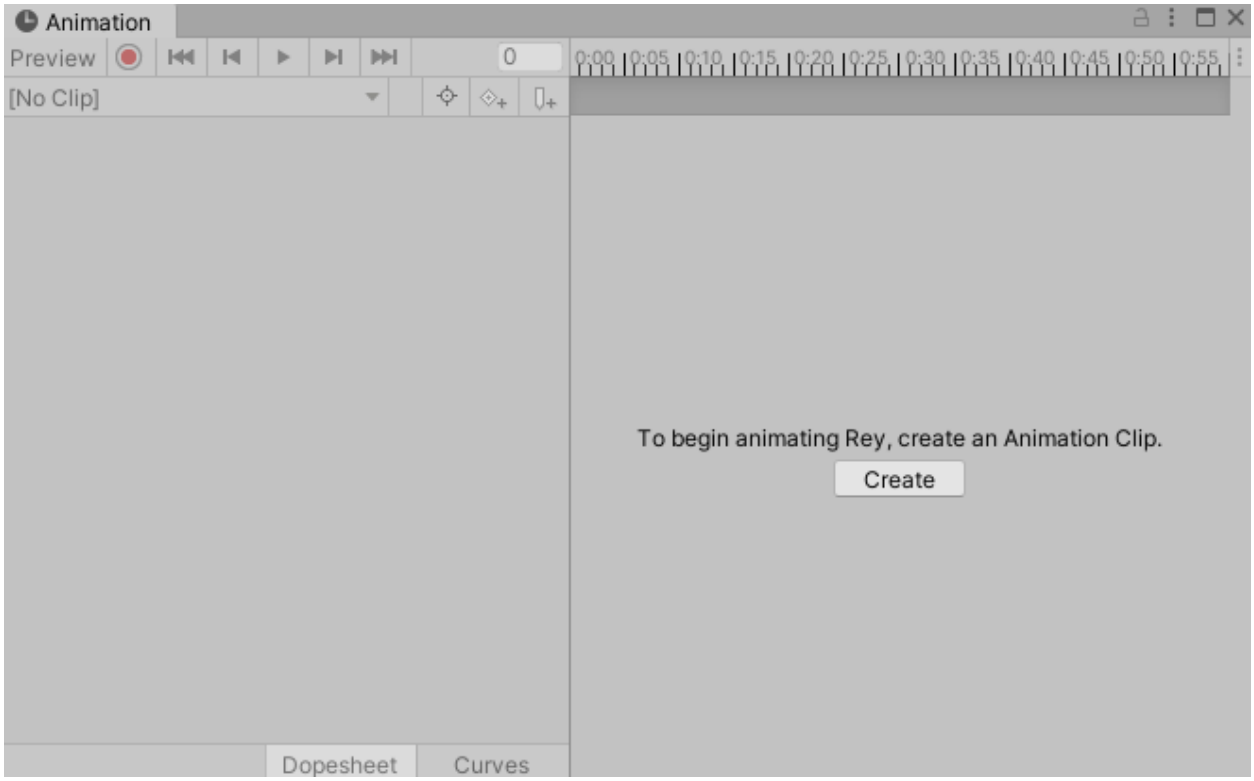

Ilustración 105. Herramienta animación

Al clicar en créate, nos pedirá una carpeta para almacenar los clips y un nombre para el clip de animación que vamos a crear. Además nos añadirá un componente a nuestro objeto con el mismo nombre que el objeto (en nuestro caso sobre el jugador Player).

Vamos a explicar cómo crear un clip con un ejemplo concreto. Por ejemplo, crearemos el clip de animación correspondiente a nuestro personaje andando hacia la derecha, andarDerecha. Tenemos que añadir a la línea temporal que aparece, los sprites que queremos que se representen.

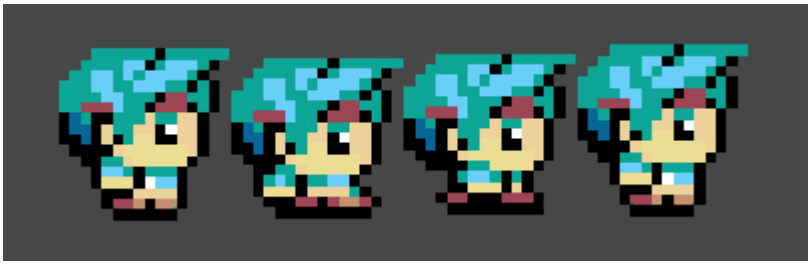

Ilustración 106. Sprites de movimiento dirección derecha

Estos sprites, son los que corresponden a la animación de andar hacia la derecha, así que lo que hacemos es arrastrar uno a uno y colocarnos donde nos convenga.

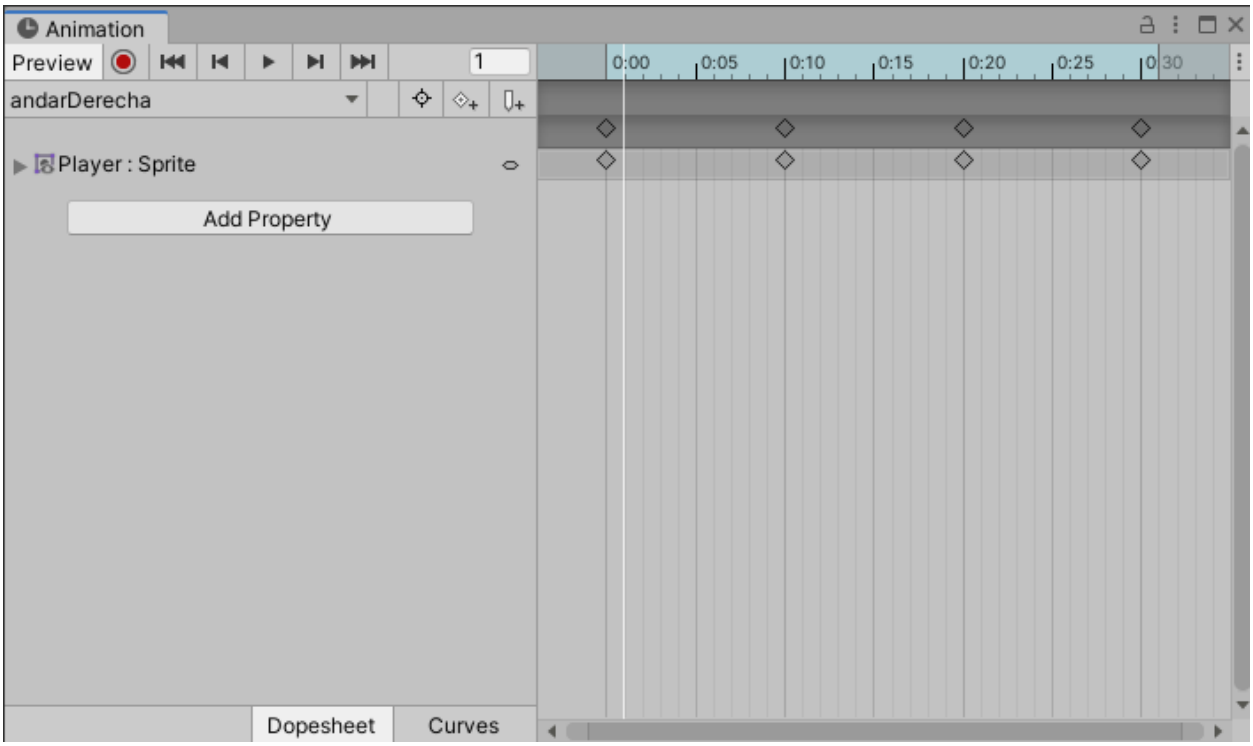

Ilustración 107.Animación andarDerecha creada con sprites

Repetimos para todas las direcciones y también para las posiciones de reposo y tendremos todas las animaciones de movimiento del jugador. De la misma forma se han animado las monedas de los niveles para hacer que giren.

6.5.2 Animator

El siguiente paso es crear las variables de animación, en caso de que fuesen necesarias, y la máquina de estados. Para el movimiento hemos creado dos variables flotantes: MoverX y MoverY y una variable booleana: moviendo.

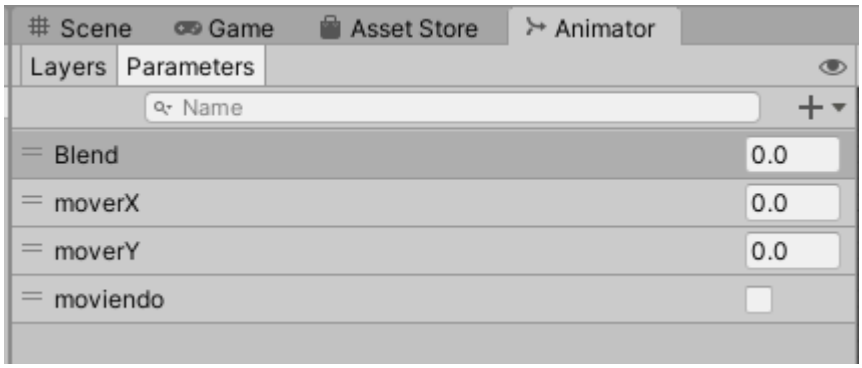

Ilustración 108. Parámetros de animación

Creadas las variables, toca crear las máquinas de estado. Se han creado dos: Reposo y Andando.

Para ello creamos un estado vacío en el animador, por ejemplo, Reposo. Hacemos doble clic en el para entrar al árbol y añadir las diferentes transiciones de estados. Hacemos que, según las coordenadas, se llame a los diferentes clips de reposo en cada una de las 4 direcciones. El resultado será como la siguiente ilustración

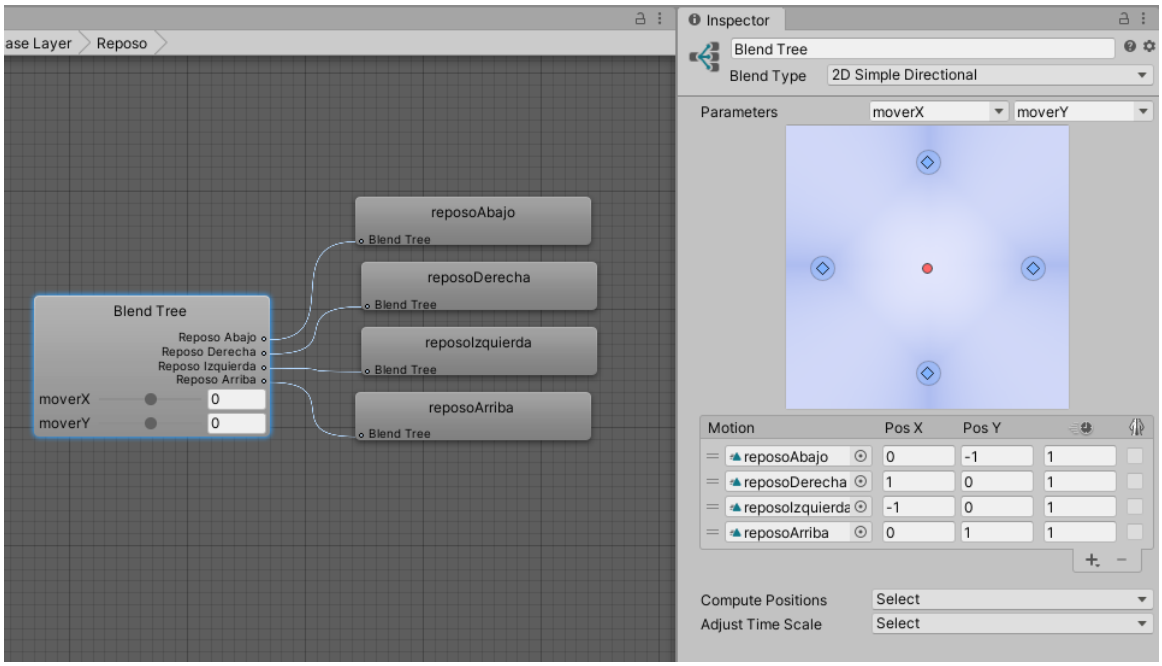

Ilustración 109. Árbol de animación de reposo

Se repite el mismo proceso para el árbol de estados correspondiente a las animaciones de Andando. Para terminar, nos queda la condición que hace la transición de estar en reposo a estar andando. Esto lo conseguiremos con el parámetro booleano moviendo.

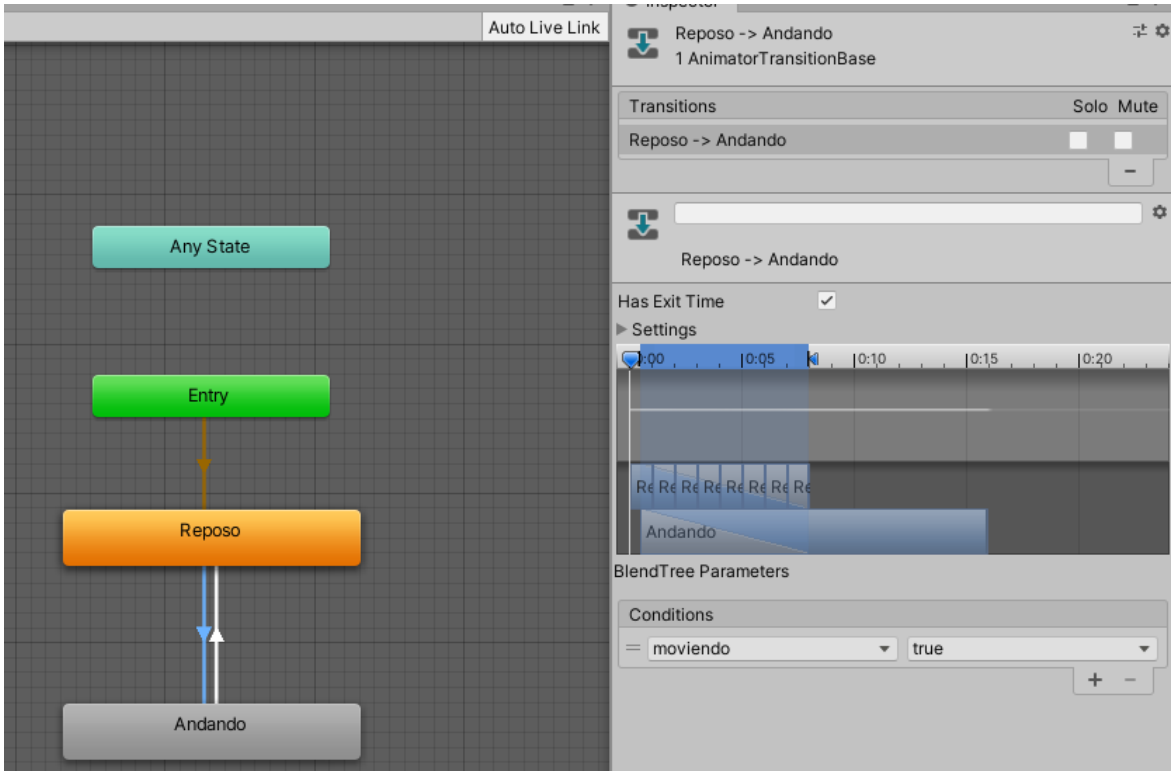

Ilustración 110. Máquina de estados

6.5.3 Animaciones desde código

El último paso es darles valor a los parámetros desde código de manera que en cuanto haya una actualización de los mismos, sea la máquina de estados la encargada de mostrar la animación adecuada. Esto lo podemos ver en los scripts MovimientoLibre y MovimientoOrden ya que al mover al personaje cambian el valor de los parámetros MoverX, MoverY y moviendo.

```
void ActualizaAnimacionYMueve()
  {
      if (cambio != Vector3.zero)
      {
          MoverPersonaje();
          animacion.SetFloat("moverX", cambio.x);
          animacion.SetFloat("moverY", cambio.y);
          animacion.SetBool("moviendo", true);
      }
      else
      {
          animacion.SetBool("moviendo", false);
      }
  }
```

Ilustración 111. Código actualiza animación y movimiento

6.6 Cinemáticas

Para poder desarrollar la historia del videojuego, se usan las cinemáticas "in game" de forma que se producen interacciones fuera del control del jugador en la que se desarrollan ciertos diálogos o acciones que marcan el transcurso de la historia del juego.

Estas cinemáticas nos permiten darle al juego dinamismo, evitando que el juego pueda hacerse pesado mostrando demasiados diálogos durante el transcurso del juego, y reservando los diálogos durante la jugabilidad para las acciones esenciales para avanzar en la historia.

En Unity, esto se ha conseguido con dos entidades esenciales, las cinemáticas en sí, que son denominadas Timeline, pertenecientes a la clase TimelineAsset del paquete System de Unity, y la clase PlayableDirector que contendrá la escena timeline y se encarga de gestionarla. En el juego se han añadido en objetos denominados TimelineManager.

6.7.1 Timeline

El editor Timeline fue introducido a partir de la versión 2019.1 y nos permite crear también secuencias de juego, de audio o efectos de partículas complejas, aparte de las cinemáticas.

Cada cinemática está formada por un TimelineAsset y una instancia Timeline y el editor los va modificando a ambos al mismo tiempo. El objeto de TimelineAsset es la cinemática como tal, y la instancia de tipo Timeline es la que asocia la cinemática con el GameObject en la escena actual a través del objeto del tipo PlayableDirector.

6.7.2 PlayableDirector

La clase PlayableDirector instancia un objeto de clase PlayableAsset y se encarga de la gestión de los objetos reproducibles. Esta clase es la que se referencia en las clases propias GestorClipsCinematicas y GestorCinematicaDialogos para la reproducción y pausa de los clips según diferentes interacciones del videojuego.

6.7.3 Creación de una cinemática

El primer paso es crear un objeto que contendrá un componente PlayableDirector. En nuestro caso hemos creado un objeto llamado TimelineManager.

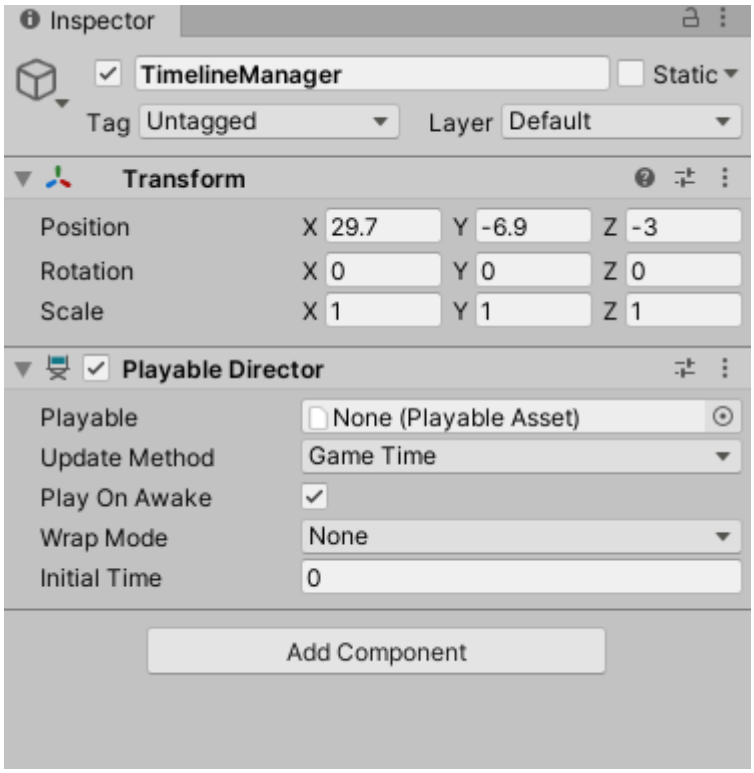

Ilustración 112. Objeto TimelineManager

El siguiente paso es decirle al TimelineManager el contenido que puede reproducir, es decir, el campo Playable. Para ello vamos a crear una nueva carpeta llamada Timeline que contendrá todas las secuencias o cinemáticas reproducibles en el juego.

Para crear una nueva cinemática, hacemos clic derecho sobre la carpeta Timeline y clicamos en Create →Timeline.

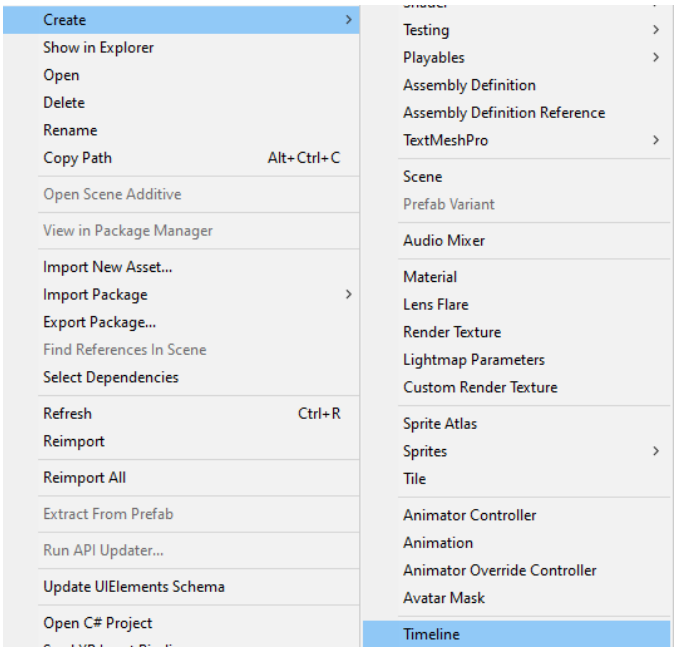

Ilustración 113. Creación de la cinemática

Una vez hecho esto, necesitamos abrir la herramienta Timeline de Unity, haciendo clic, en la barra de herramientas superior, en Windows → Sequencing → Timeline.

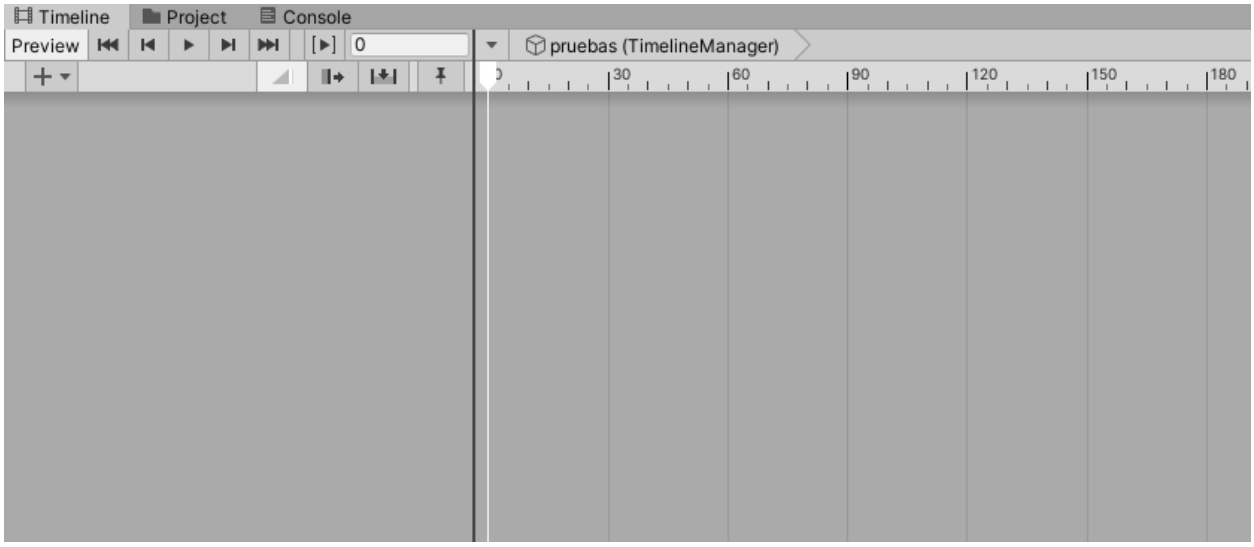

Ilustración 114. Herramienta Timeline

Desde aquí podremos hacer diferentes acciones como son mover los personajes, diferentes objetos, animarlos, hacerlos desaparecer o incluso des/activar objetos como la caja de diálogos, el reproductor de música o un clip de diálogo. Para explicarlo con mayor claridad, vamos a verlo ejemplificado con unos fragmentos las primeras cinemáticas del juego llamadas DialogoExterior y PrimeraEscena. Vamos a explicar la cinemática PrimeraEscena y posteriormente en la parte del diálogo RPGTalk, veremos la cinemática DiálogoExterior.

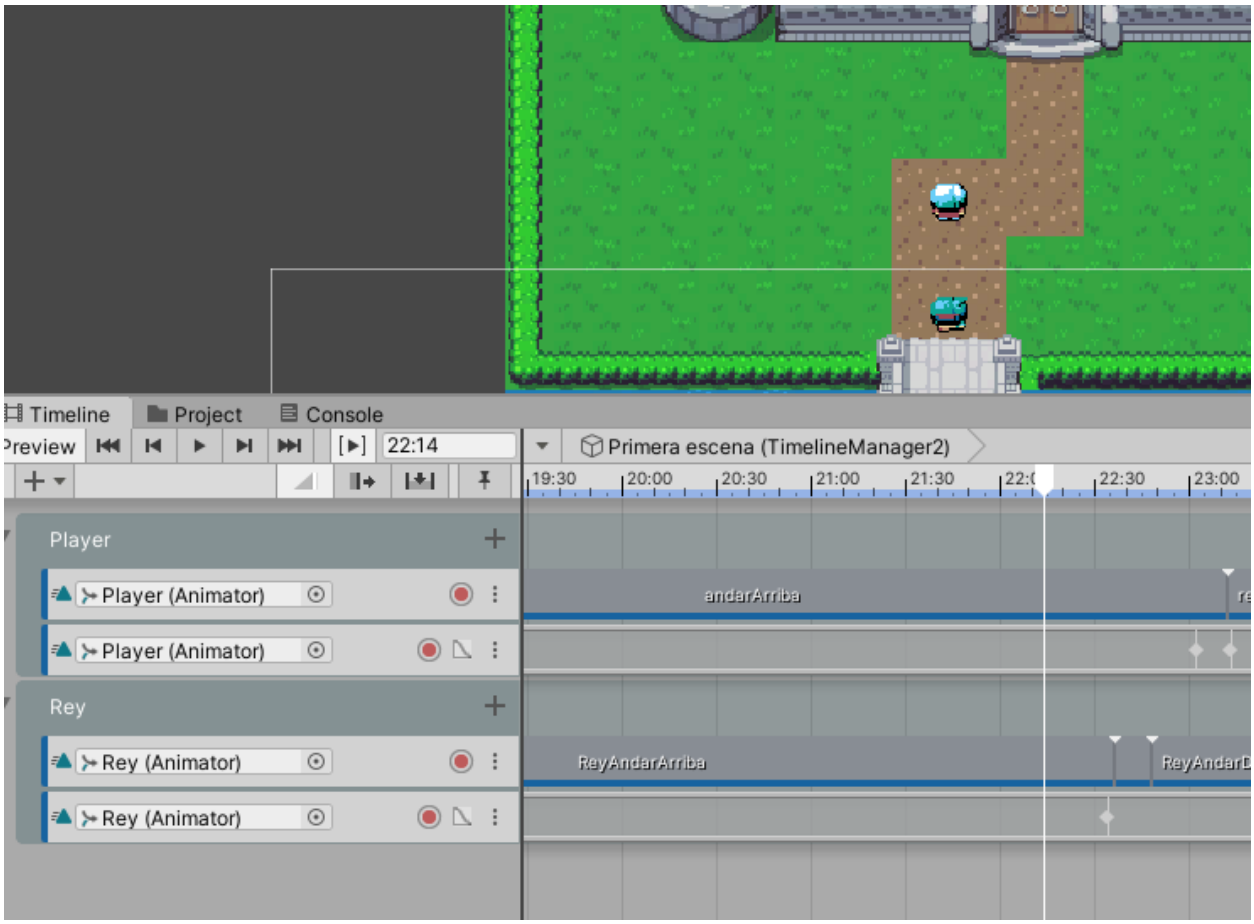

Ilustración 115. Cinemática Escena Exterior 1

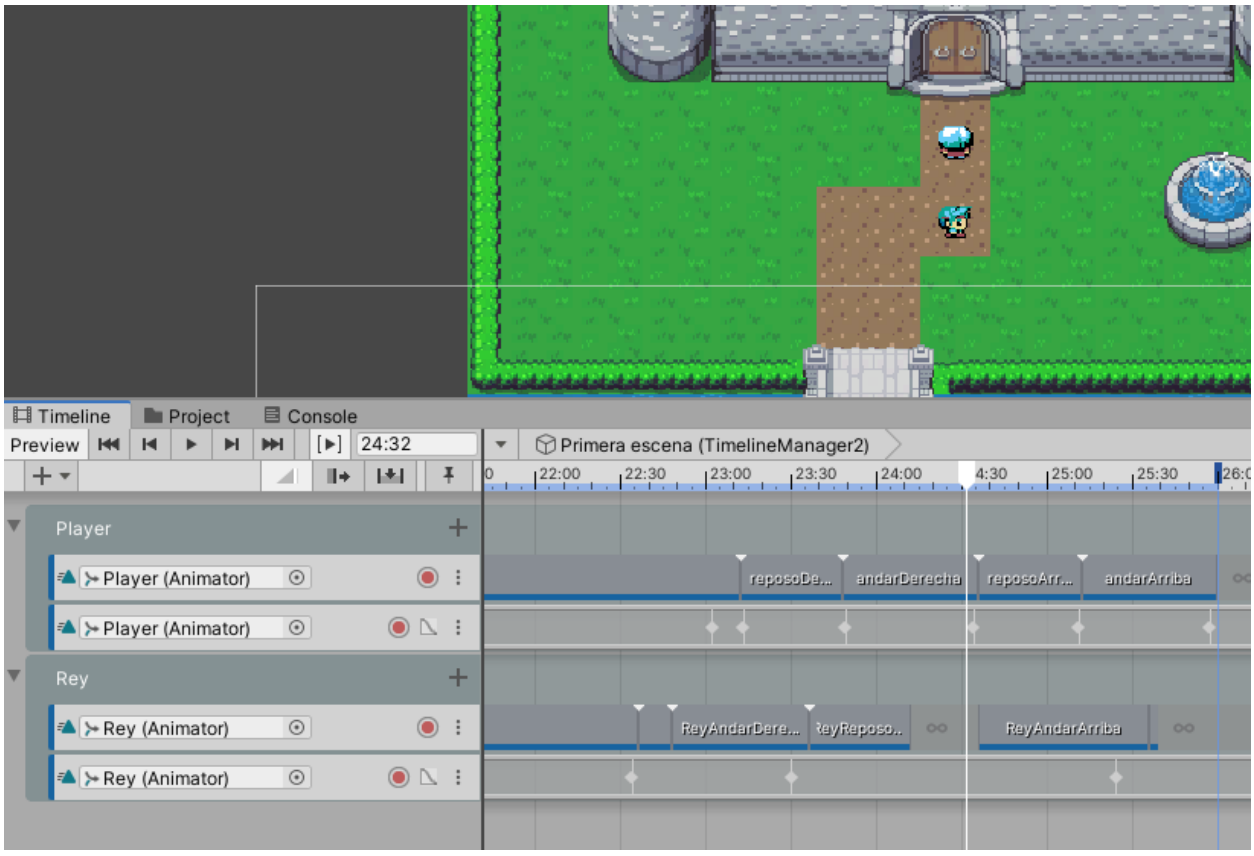

Ilustración 116. Cinemática Escena Exterior 2

Como podemos ver en las cinemáticas, tenemos dos pistas de animación para cada Sprite, uno para Player (Arcadium) y otro para Rey. En una de las pistas se graban las animaciones que queremos que se muestren en pantalla y en la otra se graba la posición en el tiempo concreto en que debe estar el personaje. Por ejemplo, cómo podemos ver en la segunda ilustración, tenemos a Player con la animación mostrada andarDerecha, y cuando llega al punto final, la posición grabada como un rombo en la otra pista de animación, se produce el cambio a reposo arriba, y por tanto nuestro jugador empezará a mirar hacia arriba.

De esta forma, vamos estableciendo las marcas temporales, indicando la posición en la que queremos que estén nuestros objetos en cada fotograma o marca de tiempo, y la animación que se quiere mostrar de ese objeto. Cuando esté completado para todos los objetos de la escena, tendremos nuestra cinemática completa.

6.7 Dialogos

Los diálogos podemos diferenciarlos en dos grupos, los diálogos de cinemáticas que están usados con una herramienta de la tienda de Unity llamada RPGTalk, la cual nos ayuda mucho para hacer las secuencias mencionadas en el punto anterior, y el resto de diálogos dentro del juego ya sea con jugadores, y objetos interactuables como carteles y PC. Estos últimos se basan en la clase GestorDialogos en la que tenemos diferentes funciones para el tratamiento del texto.

6.7.1 RPGTalk

La herramienta RPGTalk es muy útil para la gestión de diálogos, y se ha usado especialmente en las cinemáticas. RPGTalk nos permite con gran facilidad ajustar parámetros de diálogo como son la velocidad del texto, efectos de sonido al escribir el texto, mostrar imágenes y nombres de los que están hablando entre otras cosas.

Vamos a explicar cómo usar esta herramienta con el ejemplo de la segunda escena, MainScene. En ella tendremos un GameObject con un componente RPGTalk. En nuestro caso se llama GestorDialogo.

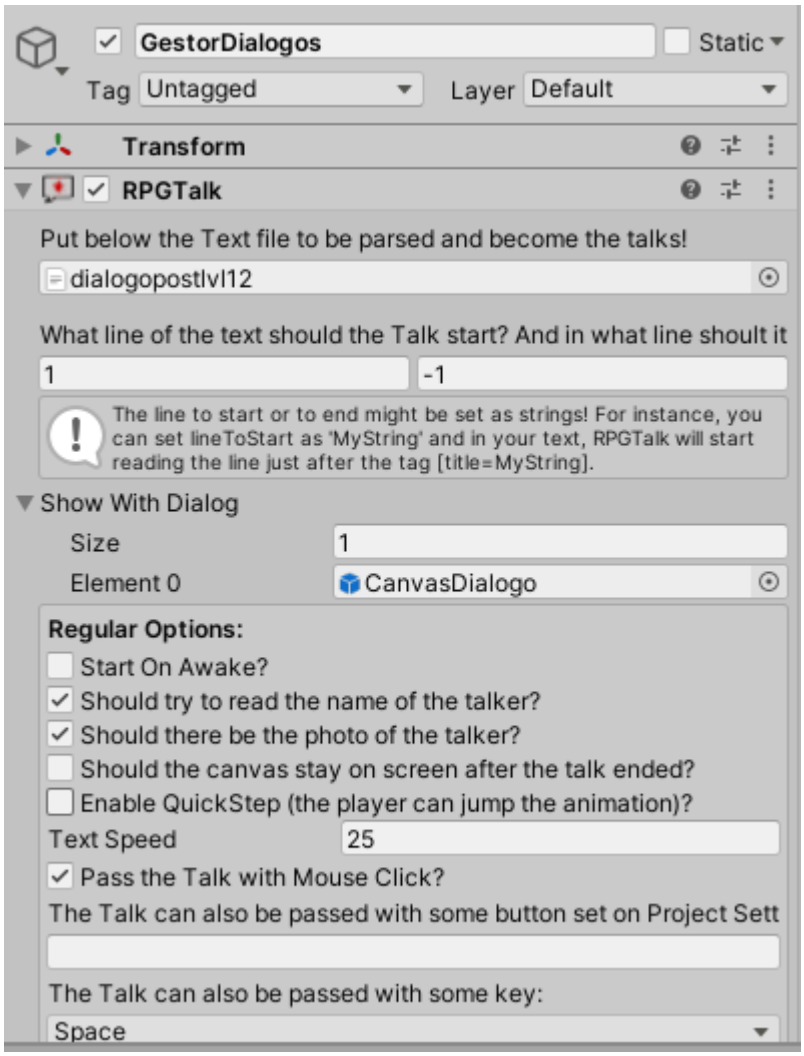

Ilustración 117. Objeto RPGTalk

En Show With Dialog seleccionamos el Canvas o la caja de diálogo que se muestra cuando se reproduce el diálogo. Se han marcado las opciones de leer el nombre y foto del interlocutor, lo cual se explicará más tarde.

La velocidad de todos los diálogos se ha fijado a 25 y se ha activado la posibilidad de saltar el diálogo con el clic izquierdo del ratón o con el espacio.

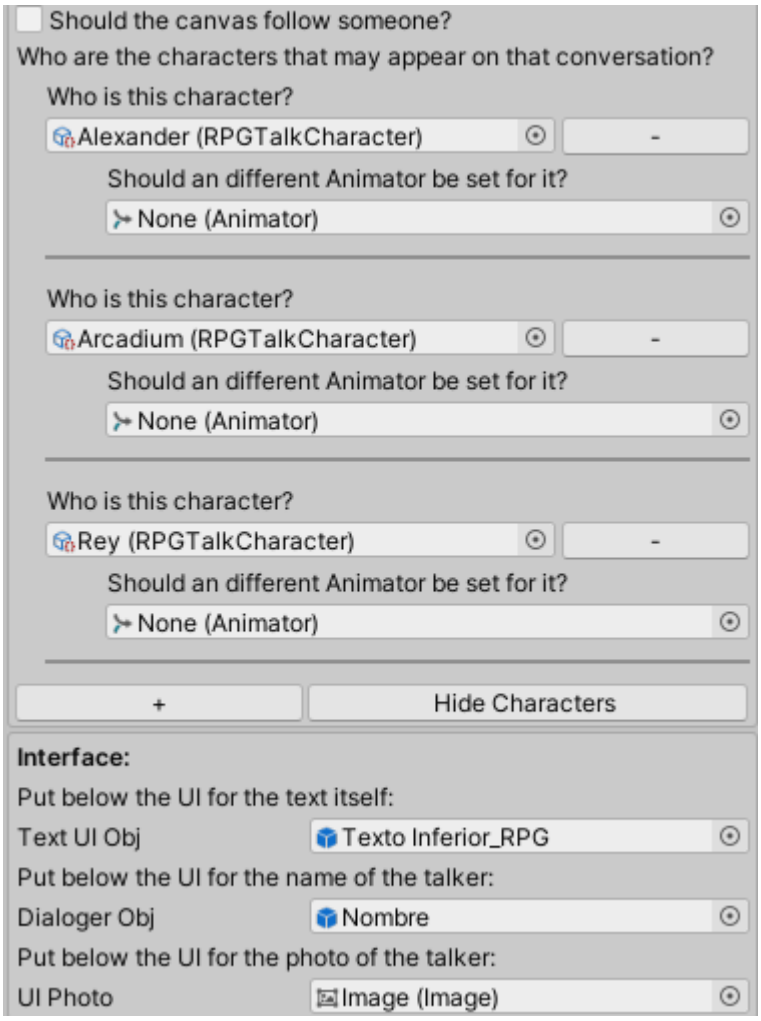

Ilustración 118. Objeto RPGTalk 2

En los ajustes de personaje, se añaden los 3 personajes que tenemos en escena (se deben crear como objetos RPGTalkCharacter) y en el apartado de interfaz, señalamos cada una de las interfaces de usuario que corresponderán al texto del diálogo (Text UI Obj), nombre del interlocutor (Dialoger Obj) y la foto del interlocutor (UI Photo)

Para finalizar, se ha añadido un clip de audio genérico cada vez que se escribe el texto, y un sonido especial cada vez que interviene nuestro protagonista, Arcadium.

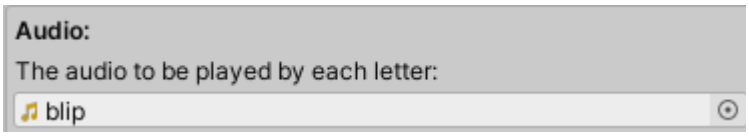

Ilustración 119. Clip de texto

6.7.2 Ficheros de texto

Para mostrar correctamente el nombre y foto de los personajes, y para añadir efectos de sonido durante el transcurso del diálogo, se usan los ficheros de texto, los cuales contendrán todo el diálogo que forma parte del guion. Siguiendo el ejemplo anterior, el fichero del dialogo se llama dialogoMainScene y es el siguiente:

I

```
Rey: Arcadium, tienes que ir a la sala de computadores y revisar la seguridad del sistema principal
Arcadium: [dub=0] Si, justo me iba a poner con eso profesor
Alexander: [dub=2] Saludos Rey, hace tiempo que no hablábamos
Rey: [dub=1] Pero qué? No puede ser...
Arcadium: [dub=0] Que ocurre profesor?
Alexander: Dejame que sea yo el que te explique. El profesor fue mi mentor durante años vio mucho potencial en mi...
Alexander: pero tenia miedo que superase sus capacidades y me alie con la fundacion Blackhat.
Arcadium: [dub=0] Espera...Blackhat?No son esos los que se encargan de hacer el mal?
Rey: Asi es Arcadium. Desde que te tuve como aprendiz vi en ti potencial y ...
Rey: he estado preparandote para superar a Alexander en caso de que esto ocurriera algun dia...
Alexander: Bla Bla Bla. Habeis terminado ya?
Alexander: Arcadium, te he puesto una serie de pruebas en los ordenadores que deberas superar para liberar el sistema
Rey: No se si aún estás listo para esto Arcadium...
Arcadium: [dub=0] Confía en mi maestro!
Alexander: Te estoy esperando en el centro de calculo
```

lustración 120. Fichero dialogoMainScene.txt

Con RPGTalk se pueden introducir “comandos” en el fichero como por ejemplo marcar donde queremos que empiece a leer y donde queremos que termine si escribimos [title=cutscene1_start]. Al no haberlo especificado, en nuestro caso lee el fichero por completo. De la misma forma, lee los nombres que haya delante del símbolo “:” y los usa para mostrar la imagen y el nombre, pero no se mostrará en el diálogo.

Para los efectos de sonido se usan los dub (doblajes). En este fichero tenemos el comando [dub=0] cada vez que interviene Arcadium para emitir su sonido de conversación y el comando [dub=2] para el efecto de alarma cuando aparece Alexander.

Desde el propio editor de Unity, tendremos que añadirle al objeto GestorDialogo mostrado anteriormente, un componente Dub Sounds de RPGTalk donde especificamos la correspondencia entre clips de sonido y doblajes

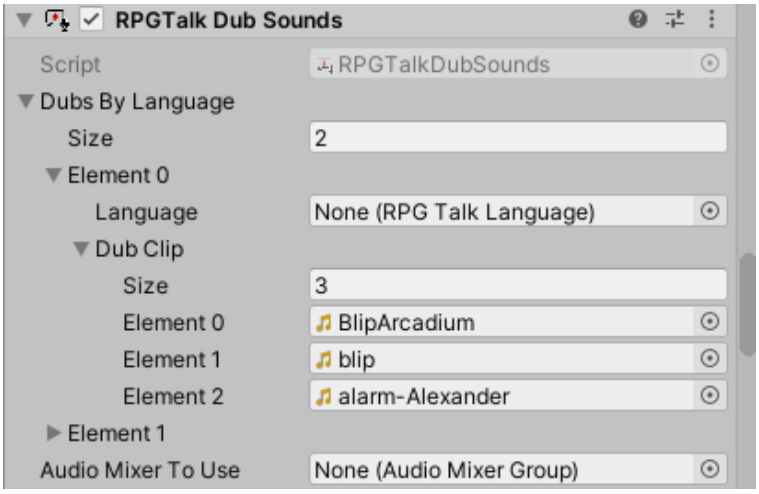

Ilustración 121. Componente Dub Sounds

6.7.3 Diálogos en cinemáticas

El último paso es que este diálogo forme parte de las cinemáticas y para eso tenemos que crear un Timeline Manager como los explicados en el apartado anterior, pero que solo contendrá una pista de cinemáticas RPGTalk

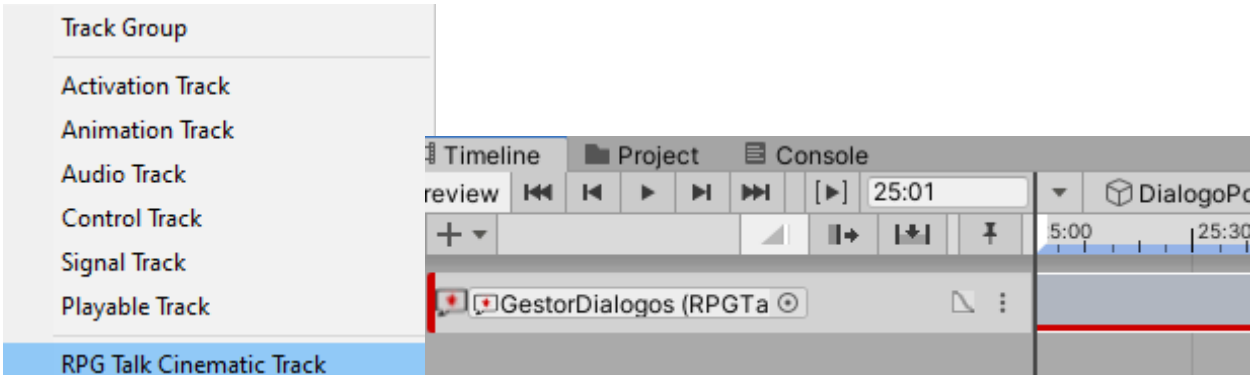

Ilustración 122. Creación pista de RPGTalk

El clip de RPGTalk durará por defecto 100segundos, de forma que de tiempo suficiente al jugador a leer los diálogos pausadamente, pero que no se quede en espera infinita. Se ha marcado la opción de que espere a una acción del jugador (pulsar espacio, o clic izquierdo del ratón) para continuar el texto y una velocidad por defecto de 25 como se comentó anteriormente.

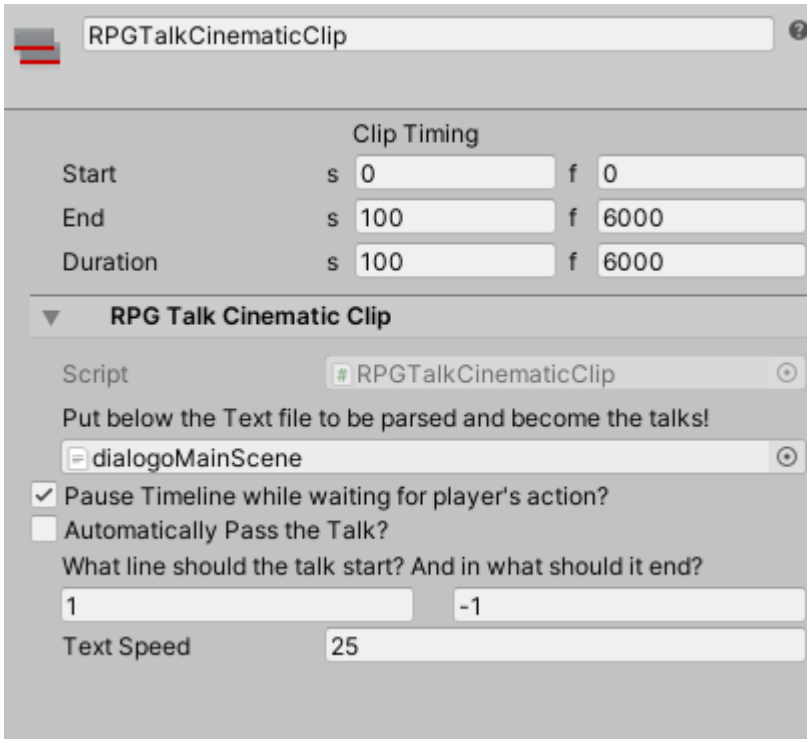

Ilustración 123. Parámetros cinemática RPGTalk

6.7.4 Clase GestorDialogos

El gestor de diálogos es la clase que hemos creado que contiene la totalidad de funciones de gestión de los textos de las diferentes interfaces y además es usada en los diálogos o textos que no usen el RPGTAlk. Por ejemplo, se usa en los mensajes de los niveles, textos de los carteles, o del tutorial. Además, se usa en todas las escenas para la eliminación de los textos y caja de diálogo, y la preparación de las interfaces previa y posterior a las cinemáticas.

Vamos a analizar en detalle el código con ejemplos para ver mejor donde se usa cada una de las funciones más relevantes de esta clase.

Para los mensajes iniciales superiores, como es el caso del texto de explicación del tutorial, o del título de los niveles Para ello se usa la función.

```
private IEnumerator MuestraDialogoInicialSuperior()
  {
      dialogoActivo = true;
      textoContinuarSuperior.gameObject.SetActive(true);
      if (textoSuperior.gameObject.activeSelf == false)
          textoSuperior.gameObject.SetActive(true);

      if (libre)
          movimientoLibre.jugadorEnDialogo = true;
      else
          movimiento.jugadorEnDialogo = true;

      while (lineaActual < lineasDialogo.Length)
      {

          textoSuperior.text = lineasDialogo[lineaActual];
          if (dialogoActivo && Input.GetKeyDown(KeyCode.Space))
                  lineaActual++;

          yield return null;
      }
      cajaDialogo.SetActive(false);
      textoSuperior.gameObject.SetActive(false);
      textoContinuarSuperior.gameObject.SetActive(false);
      dialogoActivo = false;
      lineaActual = 0;

      if (libre)
          movimientoLibre.jugadorEnDialogo = false;
      else
          movimiento.jugadorEnDialogo = false;

  }
```

Ilustración 124. Código GestorDialogos

Antes de nada, explicar el porqué del tipo de la función. En Unity, el código se va actualizando (mediante el método update) cada fotograma. En las funciones que requieren bucles (como es el caso de una reproducción de un diálogo de varias líneas) es necesario definirla como este tipo, para que la función no entre en bucles infinitos. También ha sido usada y explicada anteriormente, en ocasiones en las que se quiera realizar una espera sin que el programa se quede colgado.

Lo primero que hace es comprobar si la caja de texto superior está desactivada, y en ese caso activarla para que el texto pueda ser visible. Además, bloquea el movimiento comprobando la variable libre, que nos dice si estamos en un nivel de movimiento libre o de órdenes, bloqueando o bien el teclado, o la lectura de las teclas.

Tras esto se produce el bucle de lectura de las líneas de diálogo, esperando que el jugador pulse espacio hasta que se completa el diálogo. Una vez completado, se desbloquea el movimiento, se elimina la caja de texto, y se vuelve a poner la línea actual por defecto a 0 (para los casos de carteles en los cuales los diálogos se muestran varias veces).

Para los diálogos de los carteles o con los personajes dentro del juego, se usa la función MuestraCartel (). Está función no se va a mostrar ya que es muy parecida, cambiando la gestión de textos superiores por los inferiores, con el respectivo mensaje de continuar y la caja de diálogo que se debe eliminar a posteriori. Para añadir más variedad al juego, se ha hecho que estos diálogos se reproduzcan automáticamente al acercarnos al trigger concreto (cartel, personaje de Rey, etc) y además se ha creado una zona mayor en la cual se nos muestra un icono de diálogo sobre el personaje para que sepa antes de acercarte que hay una zona de diálogo disponible.

Ilustración 125. Icono diálogo disponible

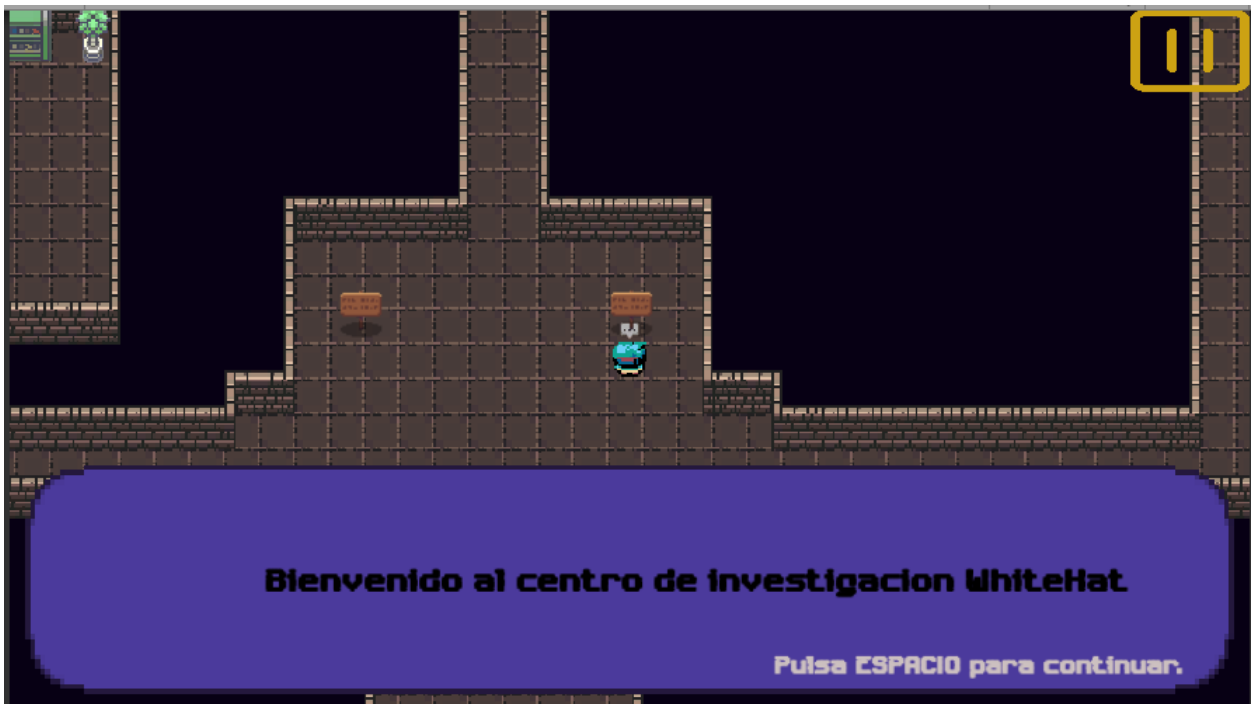

Ilustración 126. Diálogo carteles

6.7.5 TextMeshPro

La herramienta TextMeshPro se usa para crear textos que tengan mejor estética ya que las opciones que nos ofrece Unity por defecto son muy escasas. En el videojuego se ha utilizado en los botones. Vamos a mostrar como ejemplo el botón Jugar del menú principal.

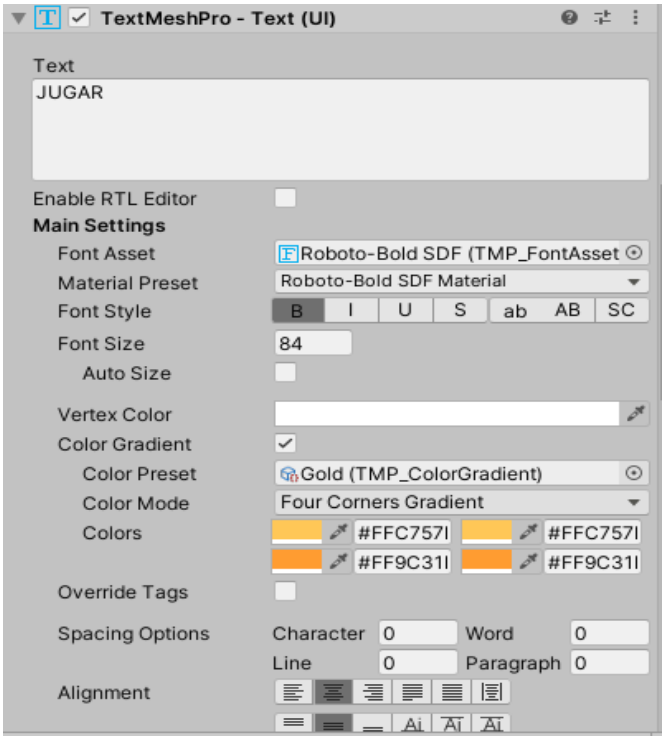

Ilustración 127. TextMeshPro

Lo más destacable es la opción Color Gradient que permite darle un efecto de profundidad, y que no sea el texto de un color fijo. Como color se ha creado una paleta denominada Gold con tonalidades oro.

Ilustración 127. Botón jugar con estilo TextMeshPro

A continuación, se proponen una serie de pruebas que el sistema debe cumplir y posteriormente se prueban hasta ser superadas con éxito, modificando por tanto la implementación en caso de que fuese necesario.

7 PRUEBAS

En este apartado se van a definir la totalidad de las pruebas que se usarán para comprobar que el comportamiento del sistema cumple con lo esperado. Estas pruebas van a estar centradas en el funcionamiento correcto de las interfaces de menú de pausa y sus funciones de guardado y ajuste de volumen, en el movimiento correcto del personaje, y el funcionamiento lógico de los niveles. Además, se han testeado la colisión con todas las paredes de los mapas de manera que no se produzcan comportamientos anómalos.

7.1 Pruebas de sistema

Guardado	CP-01
Descripción: Comprobación de que el juego se guarda correctamente probando en varios casos posibles.	
Pasos: • Se ejecuta el juego sin ningún archivo de carga debiéndose ver que el botón Cargar no está accesible • Se guarda en un nivel • Se vuelve al menú principal y se pulsa el botón cargar • Se comprueba que se carga correctamente el nivel seleccionado	
Resultado: Superado con éxito	

Tabla 25. Guardado

Cambia Volumen	CP-02
Descripción: Comprobación de que se cambia correctamente el volumen de la música de fondo desde el menú opciones, ya sea desde el menú principal o desde el menú de pausa.	
Pasos: • Se siguen los pasos del CU-04 • Se ajusta el volumen al deseado	
Resultado: Superado con éxito	

Tabla 26. Cambio de Volumen

Pausa	CP-03
Descripción: Comprobación de que, al abrir el menú de pausa, el juego se pausa y reanuda correctamente y que, en caso de interrumpir un diálogo, se elimina y se recupera correctamente.	
Pasos: • Se pausa el juego en medio de un diálogo o cinemática • El juego se pausa y aparece el menú de opciones, eliminando el cuadro de diálogo • Pulsando en reanudar debe aparecer el cuadro de diálogo de nuevo	
Resultado: Superado con éxito	

Tabla 27. Pausa

Animaciones	CP-04
Descripción: Comprobación de que, al mover el personaje, sobre todo en los niveles, ya que el modo de movimiento por órdenes es más problemático, se muestra correctamente la animación de movimiento en la dirección en que se mueve el personaje.	
Pasos: • Escribimos las ordenes Arcadium.AndarArriba, Arcadium.AndarIzquierda, Arcadium.AndarDerecha y Arcadium.AndarAbajo • El personaje se mueve una unidad y muestra la animación correctamente	
Resultado: Superado con éxito	

Tabla 28. Animaciones

Conteo de monedas y mensajes correctos	CP-05
Descripción: Comprobación de que al arrancar un nivel se cuentan correctamente las monedas, se hacen desaparecer al recogerlas, y no se permite avanzar el nivel hasta que esté complet	
Pasos: • Se arranca el nivel 5 que es el nivel que se va a probar • El jugador llega al final del nivel no recogiendo la única moneda • Se muestra el mensaje de que es necesario recoger todas las monedas • Se recoge la moneda y se completa correctamente	
Resultado: Superado con éxito	

Tabla 29. Conteo de monedas y mensajes correctos

Reinicio de nivel si personaje muere	CP-06
Descripción: Comprobación de que, si el personaje pisa unos pinchos o cae en un agujero, el nivel es reiniciado correctamente y se muestra el mensaje de nivel fallido.	
Pasos: • Se arranca el nivel 2 en el que se va a realizar esta prueba • Se anda sobre los pinchos • Se debe mostrar el mensaje de nivel fallido y cargarse correctamente el mismo nivel	
Resultado: Superado con éxito	

Tabla 30. Reinicio de nivel si personaje muere

7.2 Ejecución de pruebas

A continuación, mostraremos los pasos llevados a cabo para ejecutar las pruebas redactadas y comprobar si realmente se están cumpliendo como deben.

7.2.1 Prueba 1

Arrancamos el juego sin archivos de guardado, siendo imposible de usar el botón de cargar.

Ilustración 128. Prueba de guardado y carga 1

Tras esto hacemos clic en jugar y guardamos en alguno de los niveles, por ejemplo, en el nivel 2. Reiniciamos el juego y se deberá haber desbloqueado el botón de cargar. Además, si lo pulsamos se deberán de haber desbloqueado los niveles hasta el 2

Ilustración 129. Prueba de guardado y carga 2

Ilustración 130. Prueba de guardado y carga 3

7.2.2 Prueba 2

Para realizar la prueba de cambio de volumen, vamos a ir al menú opciones desde el menú principal y comprobar como durante la ejecución, cambia el volumen del componente de sonido asociado a medida que usamos el slider.

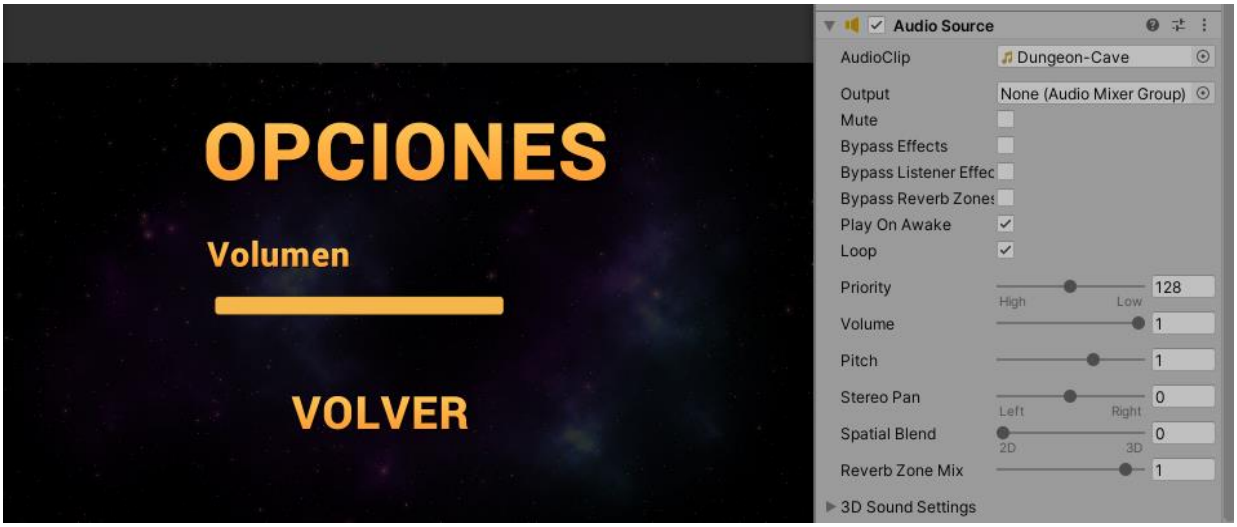

Ilustración 131. Prueba de ajuste volumen 1

Como podemos comprobar, en el inspector nos aparece que la barra de volumen está al 1, que es el valor máximo predefinido para el volumen en Unity, y a su vez, podemos ver en el menú que inicialmente el slider está orientado hacia la derecha lo que significa que también está a su máximo.

Clicando sobre el slider de volumen, cambiamos el volumen y comprobamos que efectivamente, el volumen de la música de fondo se ajusta a los valores que establecemos.

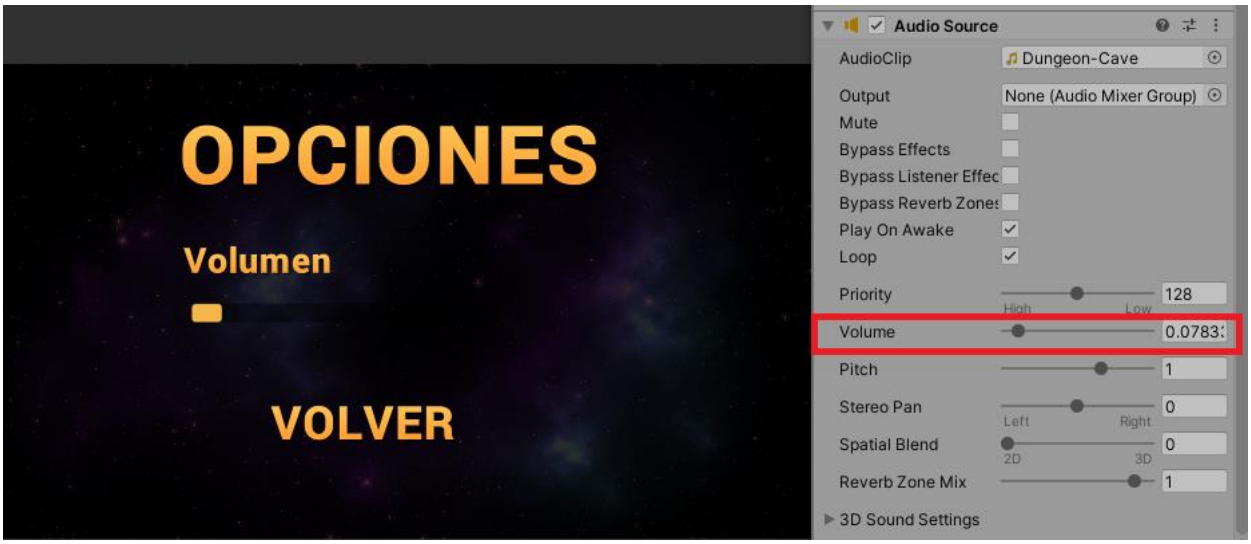

Ilustración 132. Prueba de ajuste de volumen 2

7.2.3 Prueba 3

Para realizar esta prueba, vamos a irnos hasta la segunda escena del juego, MainScene, donde se produce un dialogo inicial, y podremos comprobar si funciona correctamente la pausa durante un diálogo que forma parte de una cinemática y que la interrumpe

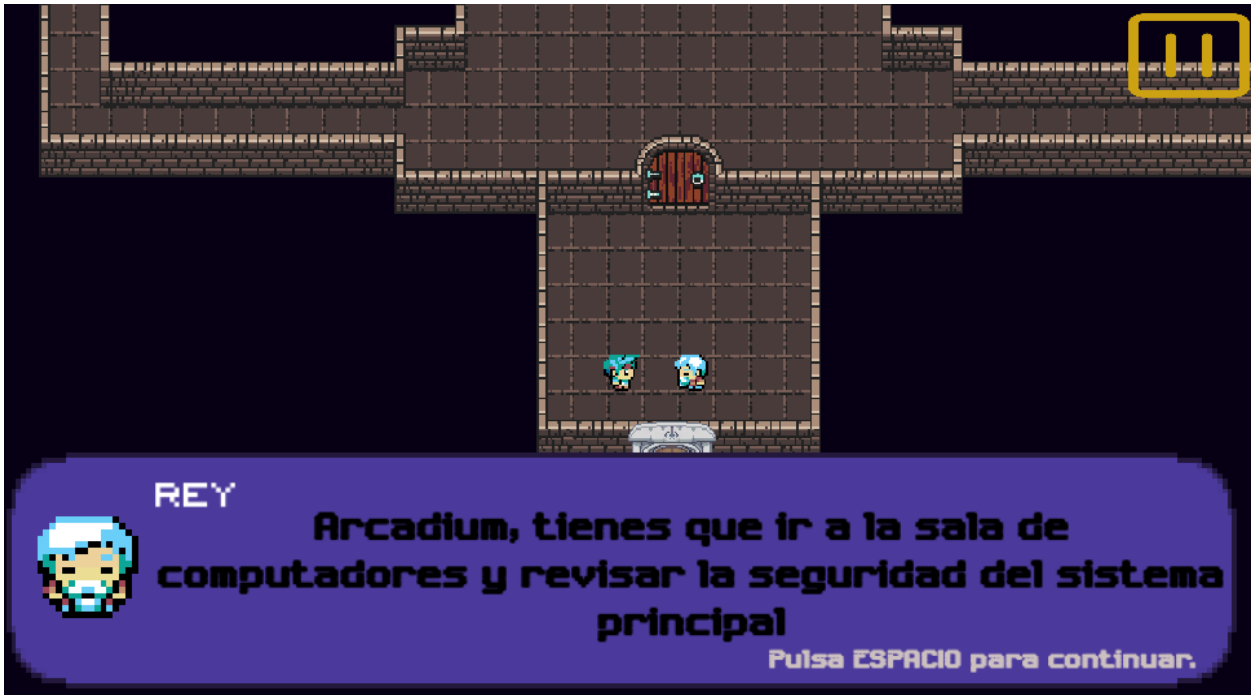

Ilustración 133.Prueba de pausa 1

Como podemos apreciar en la imagen, durante la cinemática se muestra el botón de pausa en pantalla, lo que nos indica que esa escena es pausable, por lo tanto, si accedemos al menú de pausa pulsando el botón en pantalla o la tecla ESCAPE, deberá abrirse el menú de pausa.

Ilustración 134. Prueba de pausa 2

Una vez comprobado que el menú de pausa funciona bien, para pasar por completo la prueba, deberá de recuperarse el diálogo por donde iba al pulsar en reanudar o volver a pulsar la tecla ESCAPE.

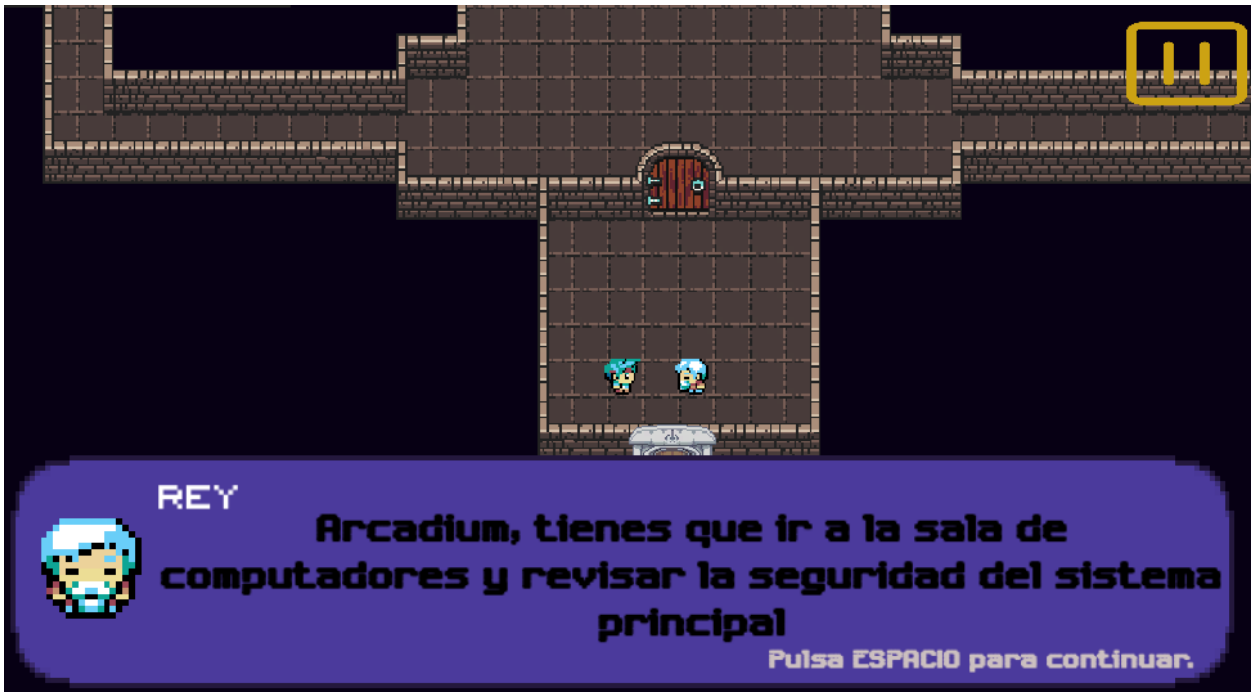

Ilustración 135. Prueba de pausa 3

7.2.4 Prueba 4

Realizamos la prueba en el nivel 1. La posición inicial es la siguiente

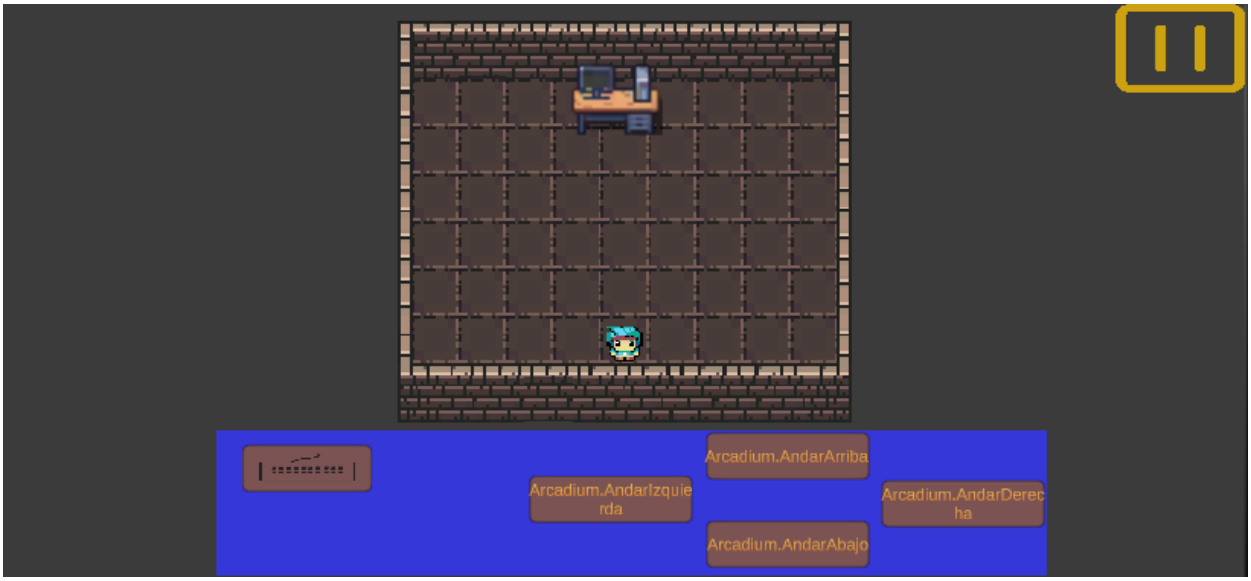

Ilustración 136. Prueba animaciones 1

Se pretende que al mover en una dirección además de realizarse el movimiento en una unidad, se reproduzca la animación correspondiente de movimiento en esa dirección, de forma de que el personaje mire a esa dirección. Si pulsamos derecha o izquierda vemos como sí que se cumple.

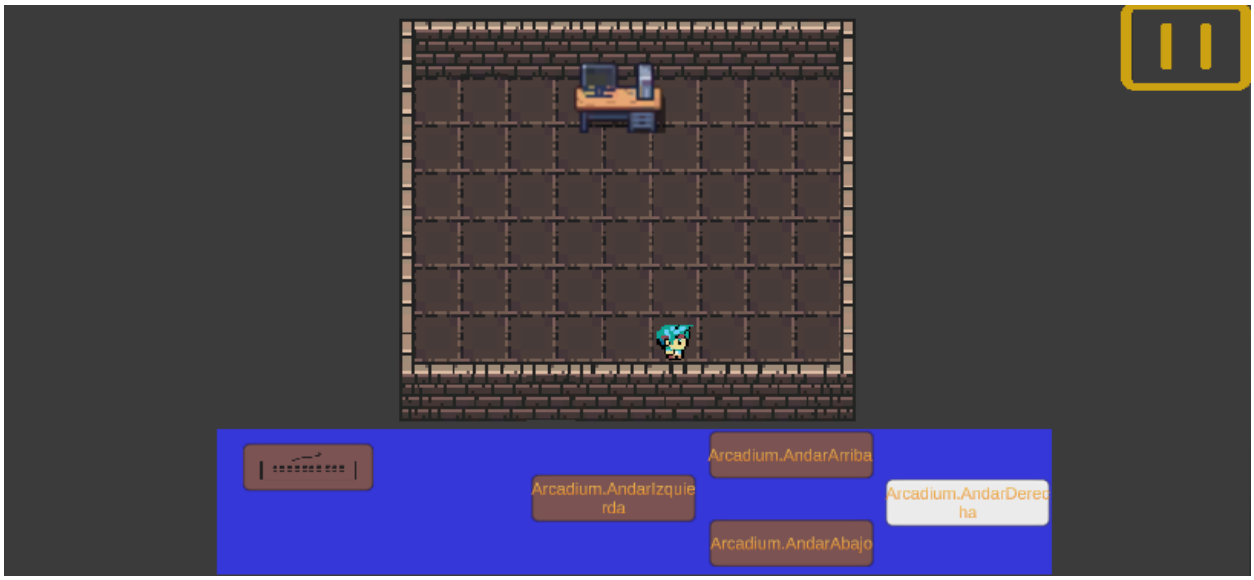

Ilustración 137. Prueba animaciones 2

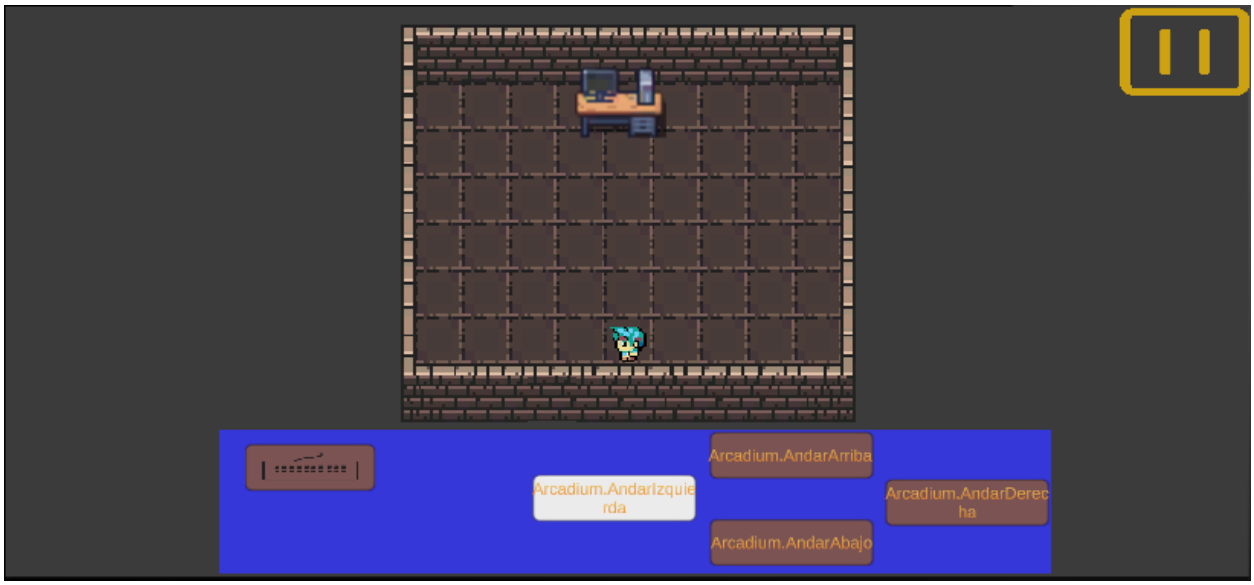

Ilustración 138. Prueba animaciones 3

7.2.5 Prueba 5

Hacemos la prueba en el nivel 5. Nos acercamos al final del nivel, sin recoger la moneda que se encuentra situada abajo.

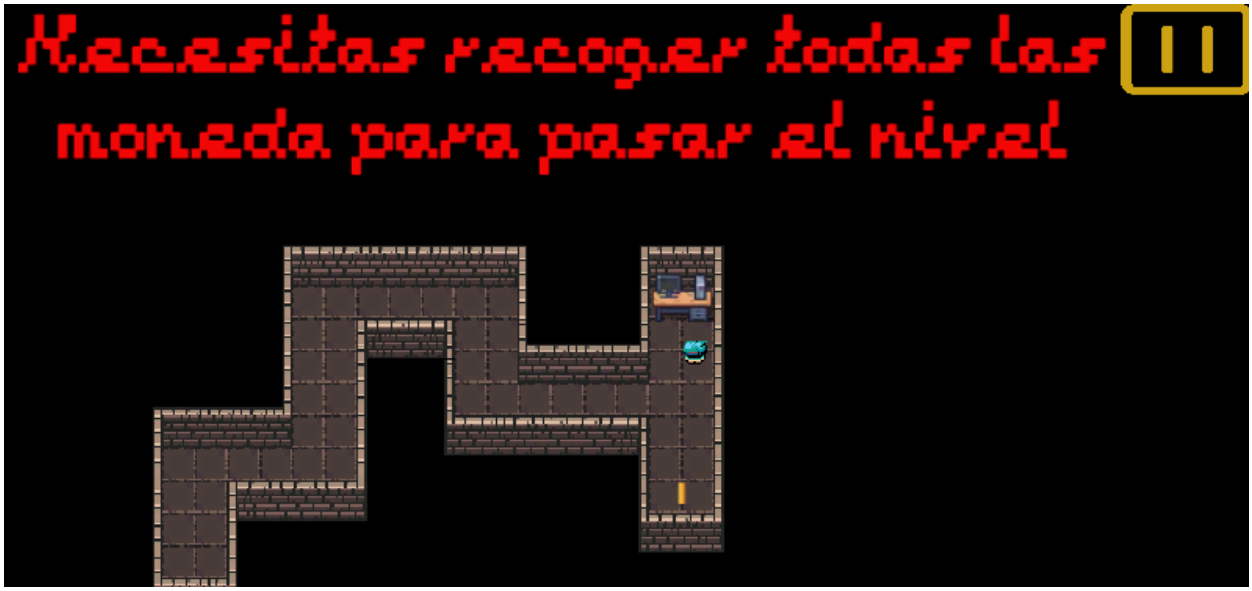

Ilustración 139. Prueba conteo monedas y mensajes 1

Ilustración 140. Prueba conteo monedas y mensajes 2

7.2.6 Prueba 6

Prueba realizada en el nivel 2. Si tocamos uno de los pinchos se muestra el mensaje de nivel fallido y se reproduce el sonido correspondiente al mensaje.

Ilustración 141. Prueba reinicio nivel si personaje muere 1

Tras esto el juego se reiniciará una y otra vez hasta que el jugador supere el nivel, o decida salir del juego.

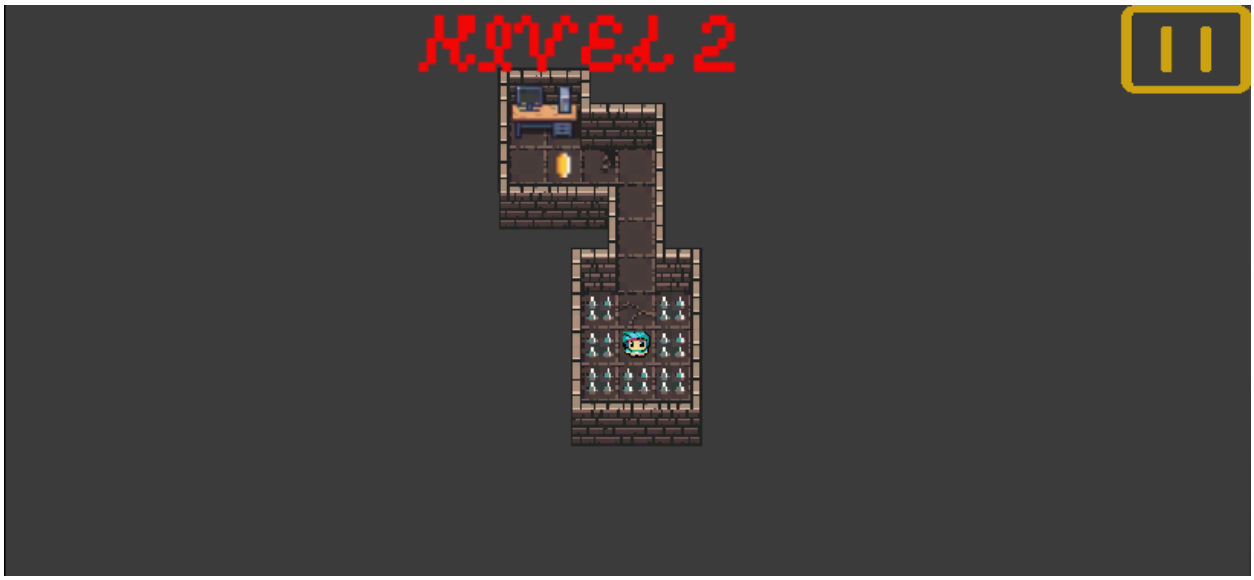

Ilustración 142. Prueba reinicio nivel si personaje muere 2

8 PLANIFICACIÓN

Este apartado muestra la planificación que se ha seguido a lo largo del proyecto. Para ilustrarlo de la mejor manera se ha elaborado un diagrama de Gantt como se puede apreciar en la siguiente imagen.

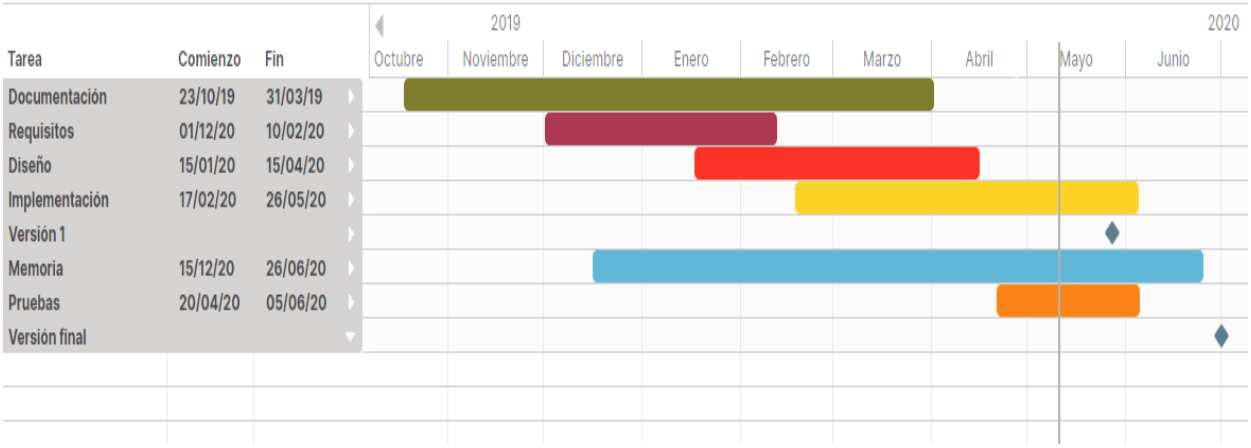

Ilustración 143. Diagrama Gantt

El proyecto se inicia con la parte de documentación, en la que se buscó información sobre Unity, C# y sobre videojuegos didácticas en general. También en esta parte se incluye la lectura de las memorias de los proyectos anteriores con los cual está relacionado, y el testeo de las aplicaciones finales. Este proceso durará hasta casi el final del proyecto de manera paralela a medida que se va complementando la memoria.

Tras la primera parte de documentación, se procede a plantear los requisitos generales que tendrá el sistema. Esta primera versión, junto con el diseño marca los primeros pasos en la implementación. Se empieza con un primer diseño y a raíz de este van apareciendo nuevos requisitos que no eran tan obvios al principio y una mayor especificación en cuanto a los casos de uso.

Una vez terminada la especificación de requisitos y un primer diseño, entramos en la implementación. Se comienza con un diseño básico de la página Web y los primeros mapas de Unity. Tras esto, se empieza a codificar el movimiento, las transiciones de escena habitaciones, etc. Hasta que se completa una primera versión.

Tras la primera versión, definimos las pruebas que deberá pasar el sistema y las probamos. A la vez que esto, se sigue con la implementación arreglando bugs, añadiendo niveles, cinemáticas, historia, y todo el contenido que le faltaba al juego en este punto.

Una vez terminadas todas las pruebas y la implementación correspondiente, se documenta todo en la memoria y se llega a la versión final que será la entregada.

9 LINEAS FUTURAS

En este apartado se van a comentar las mejoras y actualizaciones que se plantean para el futuro. En concreto, se trata de la versión 2.0 del juego Dungeon Code, con una actualización de la página LearnGaming de manera que sea mucho más completa, a partir de la experiencia ganada realizando este proyecto.

Se pretenden añadir versiones de los niveles para diferentes lenguajes, así como añadir mayores funcionalidades al teclado, para una mayor variedad de acciones y un mayor aprendizaje con el juego. Además, se pretende aumentar los niveles y alargar la historia, con pequeños cambios en el guion que permitan alargar la trama y seguir la estructura que ya tiene el videojuego. Esto conlleva a la aparición de un nuevo enemigo, Norman, que pondrá de nuevo a prueba a Arcadium.

En cuanto a la página Web, se pretende completar con mayor profundidad las lecciones de los lenguajes que ya se tienen, y añadir algunos nuevos. Además, se complementaría con cuestionarios online que prueben el conocimiento de los usuarios.

10 BIBLIOGRAFÍA

Plantillas de ingeniería de requisitos:

http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/407

http://www.juntadeandalucia.es/servicios/madeja/contenido/subsistemas/ingenieria/ingenieria-requisitos

http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/462

Documentación videojuegos:

http://www.aikaeducacion.com/tendencias/los-videojuegos-transforman-aula/

https://www.microsoft.com/es-es/makecode/about

https://education.minecraft.net/trainings/code-builder-for-minecraft-education-edition/

https://education.minecraft.net/

Motores Gráficos:

https://blogthinkbig.com/motores-graficos

https://www.vidaextra.com/listas/si-quieres-hacer-tus-propios-juegos-estos-son-los-mejores-motores-que-vas-a-encontrar

https://baboonlab.odoo.com/blog/noticias-de-marketing-inmobiliario-y-tecnologia-1/post/unreal-engine-4-el-motor-grafico-que-ofrece-realismo-al-maximo-23

https://es.wikipedia.org/wiki/Unreal_Engine#Unreal_Engine_4

https://www.aragon.es/documents/20127/674325/Estado%20del%20arte%20GameEngines%20y%20su%20impacto%20en%20la%20industria.pdf/db827568-09ef-e931-01e8-d293b9fca834

https://unity.com/es/solutions/game

https://en.wikipedia.org/wiki/UbiArt_Framework

https://www.cryengine.com/

https://www.unrealengine.com/en-US/

https://developer.valvesoftware.com/wiki/Main_Page

Guiones:

https://creamundi.es/los-mejores-softwares-guion-escribir-comodamente/

https://www.gametopia.es/learning/article/10/2018/52/como-escribir-un-guion-de-videojuego#

Software de escritura de guiones:

https://www.celtx.com/index.html

https://writerduet.com/

https://www.finaldraft.com/

https://www.fadeinpro.com/

Manuales Unity:

https://docs.unity3d.com/Manual/index.html

Libros C# y Unity:

Jared Halpern: *Developing 2D Games with Unity. Independent Game Programming with C#*
New York, NY, USA; e-ISBN: 978-1-4842-3772-4; ISBN: 978-1-4842-3771-7

Alan Thorn: *Learn Unity for 2D Game Development.*
New York, NY 10013, e-ISBN: 978-1-4302-6229-9

Venita Pereira: *Learning Unity 2D Game Development by Example*
Birmingham, UK, ISBN: 978-1-78355-904-6

Jamie Chang: *Learn C# in One Day and Learn It Well*
ISBN: 978-1-51880-027-6

11 ANEXO

11.1 Anexo A: Instalación de Unity

Unity ofrece 5 tipos de licencias divididas en dos ramas; en la parte de licencias individuales tenemos la **personal** (gratuita) y la **premium** (15$ / mes) y la parte de negocio con los planes **plus** (35$ / año), **pro** (125$ / año) y el **enterprise** cuyo precio se negocia con Unity al estar customizado para la empresa concreta. Para el desarrollo de este proyecto hemos utilizado la personal. Las principales características de las versiones de pago son: mejor control y análisis sobre los datos, mejoras multijugador, bancos de pruebas usando el servicio de nube de Unity y acceso al código fuente en el modo enterprise.

En la parte de negocio, a partir de cierto beneficio de la empresa, Unity te obliga a pagar cierta suscripción superior desde la personal a la pro.

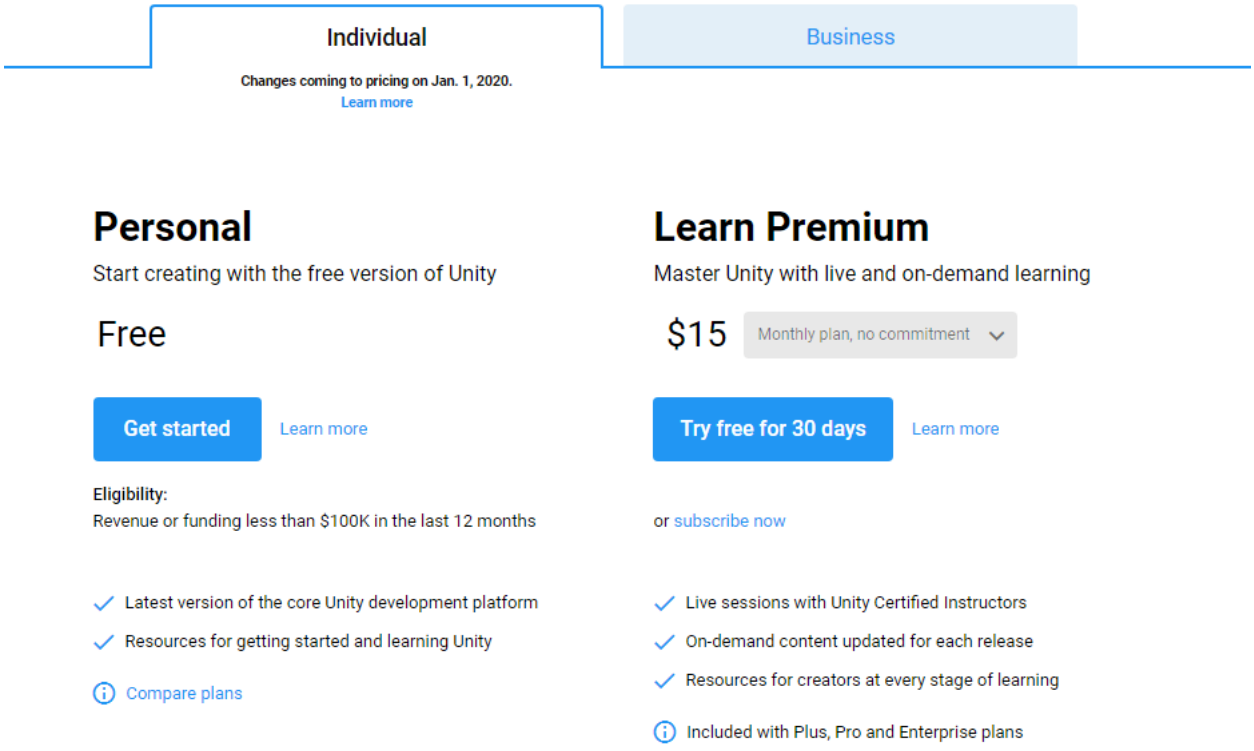

Ilustración 144. Tarifas Unity

Al instalar nos pedirá que componentes queremos añadir a la build del programa como vemos en la siguiente imagen.

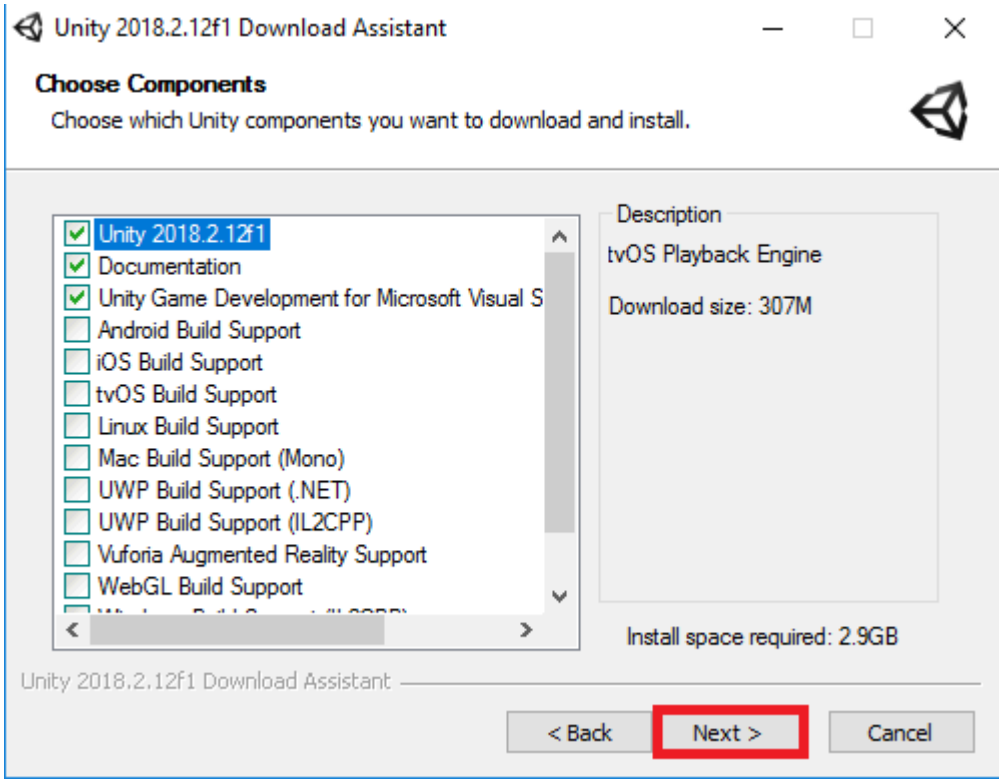

Ilustración 145. Primeros pasos instalación

En nuestro caso añadiremos el soporte Android, iOS y WebGL.
En cuanto finalice la instalación nos pedirá que creemos una cuenta e iniciemos sesión

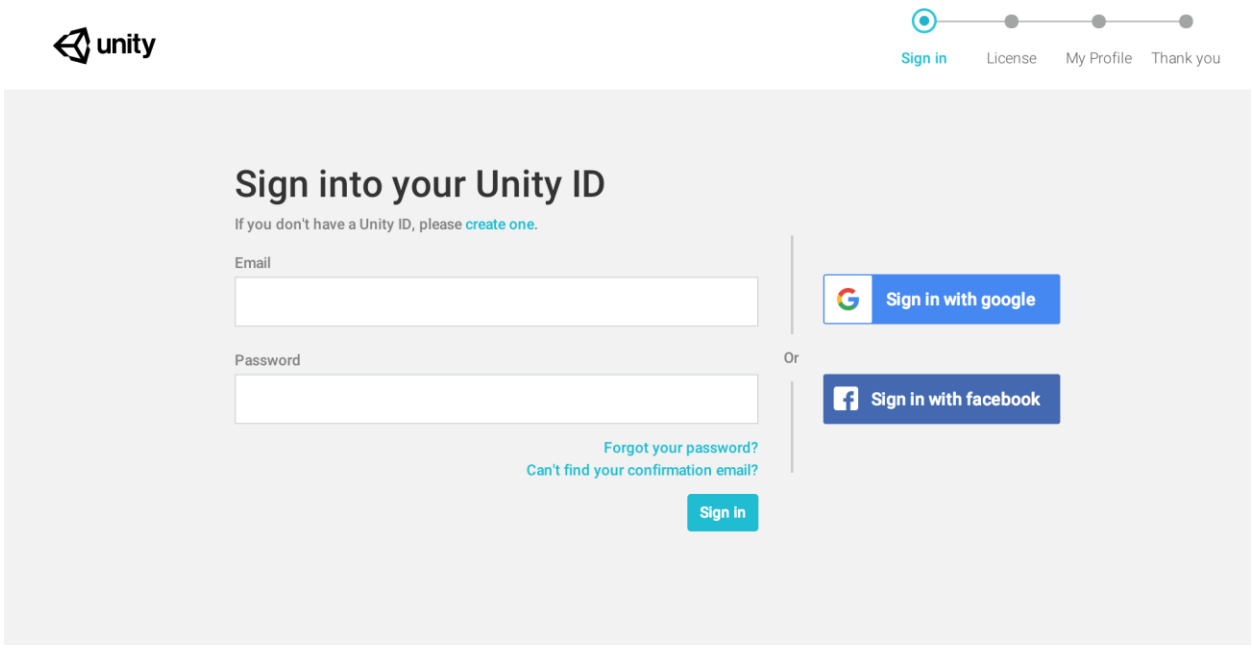

Ilustración 146. Inicio sesión Unity

Una vez loggeados llegamos a la pantalla de proyectos Unity y de Learning donde podremos tanto gestionar los

proyectos como acceder a tutoriales y contenido de la documentación oficial de Unity.

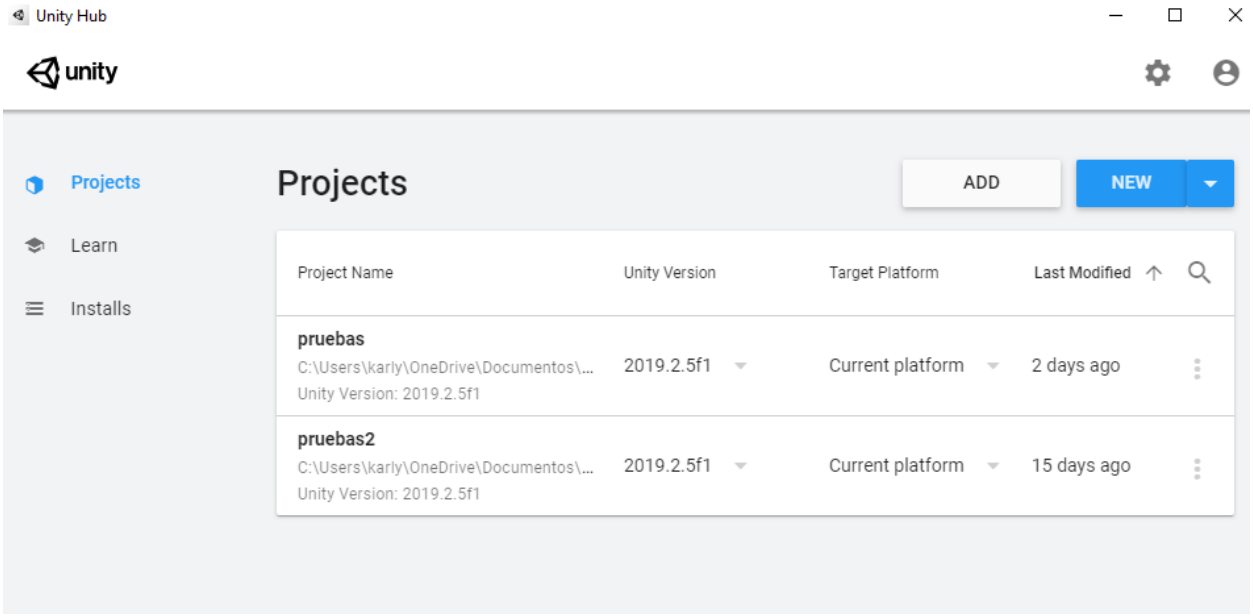

Ilustración 147. Vista de proyectos

11.2 Anexo B: Lenguaje C#

C#, también conocido como C Sharp, es un lenguaje de programación orientado a objetos desarrollado por Microsoft a principios de los años 2000. En cuanto a sintaxis, se parece bastante a C++ y Java, y suele ser bastante fácil de aprender.

11.2.1 Estructura de un programa en C#

Un programa en C# consta, por lo general, de las siguientes partes:

- Directivas: Usadas con la palabra clave *using*, para incluir un espacio de nombres. Similar al *include* de C o el *import* de Java.
- Declaración de espacio de nombres: Un espacio de nombres es una agrupación de código que guarda cierta relación (clases relacionadas).
- El método Main(): al igual que en C o en Java, es el punto de entrada al programa.
- Comentarios

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HolaMundo
{
    //Un programa simple mostrando hola mundo

    class Programa
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Holla Mundo");
            Console.Read();
        }
    }
}
```

Ilustración 148. Código ejemplo C#

Además, tendremos:

- Clases
- Métodos y atributos de clase
- Constructores de clase

```
public class Punto
{
    public int x, y;
    public Punto(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

Ilustración 149. Código clases ejemplo C#

11.2.2 Variables y operadores

Las variables son nombres dados a datos que necesitamos almacenar y manipular en nuestro programa. Para ello tenemos una serie de tipos de datos:

- Entero: *int* es usado para números enteros en un rango desde -2,147,483,648 hasta 2,147,483,647.
- Byte: *byte* se refiere también a números enteros, pero en un rango más pequeño desde 0 a 255.
- Float: *float* se refiere a números flotantes, que son números con decimales.
- Double: *double* es usado para números flotantes, pero en un rango muchísimo más amplio que *float*.
- Decimal: *decimal* nos permite almacenar un número decimal como *float* o *double,* pero con una mayor precisión
- Carácter: *char* es usado para almacenar caracteres según el estándar Unicode
- Boolean: *bool* almacena los valores *true* o *false* y es usado para las sentencias de control.

Además, tenemos los siguientes operadores:

- Asignación: El signo = nos sirve para realizar la asignación a una variable.
- Suma: Usamos el signo + para la suma.
- Resta: Usamos el signo – para la resta.
- Multiplicación: Usamos el signo * para la multiplicación.
- División: Usamos el signo / para la división.
- Resto: El signo % nos devuelve el resto de una división.
- Incremento: Con ++ aumentamos en 1 unidad una variable.
- Decremento: Con – disminuimos en 1 unidad una variable.

11.2.3 Métodos

Un método es un bloque de código que completa una tarea concreta y que solo se ejecuta cuando es llamado desde cierta parte del programa. Sigue la siguiente sintaxis:

```
[Especificador Acceso] [Tipo] NombreFuncion(Lista parámetros)
{

    CuerpoDeLaFunción

}
```

Ilustración 150. Sintáxis métodos

El especificador de acceso establece el nivel de acceso/visibilidad para el método. En C# tenemos los siguientes modificadores de acceso:

- Public: El código es accesible para todas las clases.
- Private: El código es accesible solo desde dentro de la propia clase.
- Internal: El código es accesible desde su propio ensamblado.
- Protected: El código es accesible desde dentro de la clase, o desde una clase que haya heredado de la que contenga el método.

El tipo que la función retorna se corresponde con los tipos de datos vistos anteriormente. En caso de que la función no devuelva nada, su tipo será *void.*

El nombre de la función es un identificador único y distingue entre mayúsculas y minúsculas. Tras el nombre, aparecen entre paréntesis y separados por comas, los parámetros que se le pasan a la función como dato.

11.2.4 Estructuras de control

Las estructuras de control nos permiten variar el flujo normal de un programa en función de una serie de condiciones. En C# tenemos las sentencias de control *if, switch* y los bucles *for, while* y *do while.*

11.2.4.1 Sentencia If

La sentencia *if* es una de las más usadas. Nos permite que el programa evalúe si una condición concreta se ha producido, y actuar adecuadamente en función del resultado de la evaluación.

```
if (condición1)
 {
 Si se cumple condición1 ejecuto este código
 }
else
 {
 Si no se cumple condición 1 ejecuto este código
 }
```

Ilustración 151. Sentencia If C#

Además, se pueden anidar varias sentencias *if/else* haciendo múltiples selecciones.

11.2.4.2 Sentencia Switch

La sentencia *switch* es similar a *if con* la diferencia que se evalúa una sola vez, y se compara con cada sentencia *case* eligiendo uno u otro bloque de código

```
switch (variable usada para comparar valores)
{
case primerCaso:
        Código a ejecutar;
        break;
case segundoCaso:
        Código a ejecutar;
        break;
 case tercerCaso:
        Código a ejecutar;
        break;
 case default:
        Código a ejecutar;
        break;
 }
```

Ilustración 152. Sentencia Switch C#

11.2.4.3 Bucle For

El bucle for ejecuta una determinada secuencia de código repetidamente hasta que la condición que se prueba deja de ser válida.

```
   for (valor inicial;condicion a probar; modificaciones a los valores)
{

    bloque de código a ejecutar

}
```

Ilustración 153. Bucle For C#

11.2.4.4 Bucle While

Como su nombre indica, el bucle while ejecuta repetidamente un bloque de instrucciones mientras la condición siga siendo válida. Por tanto, es necesario, que los valores de las variables que se usan para la condición se modifiquen de alguna manera para evitar quedarnos en un bucle infinito ya que la condición nunca deja de ser cierta.

```
while (condición sea cierta)
{

    Bloque código a ejecutar

}
```

Ilustración 154. Bucle While C#

11.2.4.5 Bucle Do While

El bucle Do While es muy parecido al bucle While, con la diferencia que nos asegura que el bloque de código sea ejecutado mínimo una vez, ya que se hace la primera comprobación después de la primera ejecución.

```
do
{

    bloque de código a ejecutar

} while (condición sea cierta);
```

Ilustración 155. Bucle Do While C#

11.3 Anexo C: Página Web alojada en GitHub Pages

Por último, se va a documentar la página Web que complementa al juego para formar esta herramienta didáctica Además se explicará el servicio de GitHub Pages usado para el alojamiento de la página y las pruebas de la misma.

La página Web, titulada como LearnGaming, consta de un apartado de lecciones para los lenguajes: C, Java, CSS y HTML y próximamente se plantea añadir los lenguajes SQL y JavaScript. A lo largo de las lecciones, encontraremos enlaces a los apartados de herramientas externas dentro de la propia web, en la cual tendremos enlaces a páginas externas y recursos que complementarán las lecciones y aportan ejemplos concretos. En el apartado de herramientas externas tendremos además enlaces a 3 páginas externas en las que se nos ofrecen juegos para aprender a programar: CodinGame, Checkio y CodeCombat.

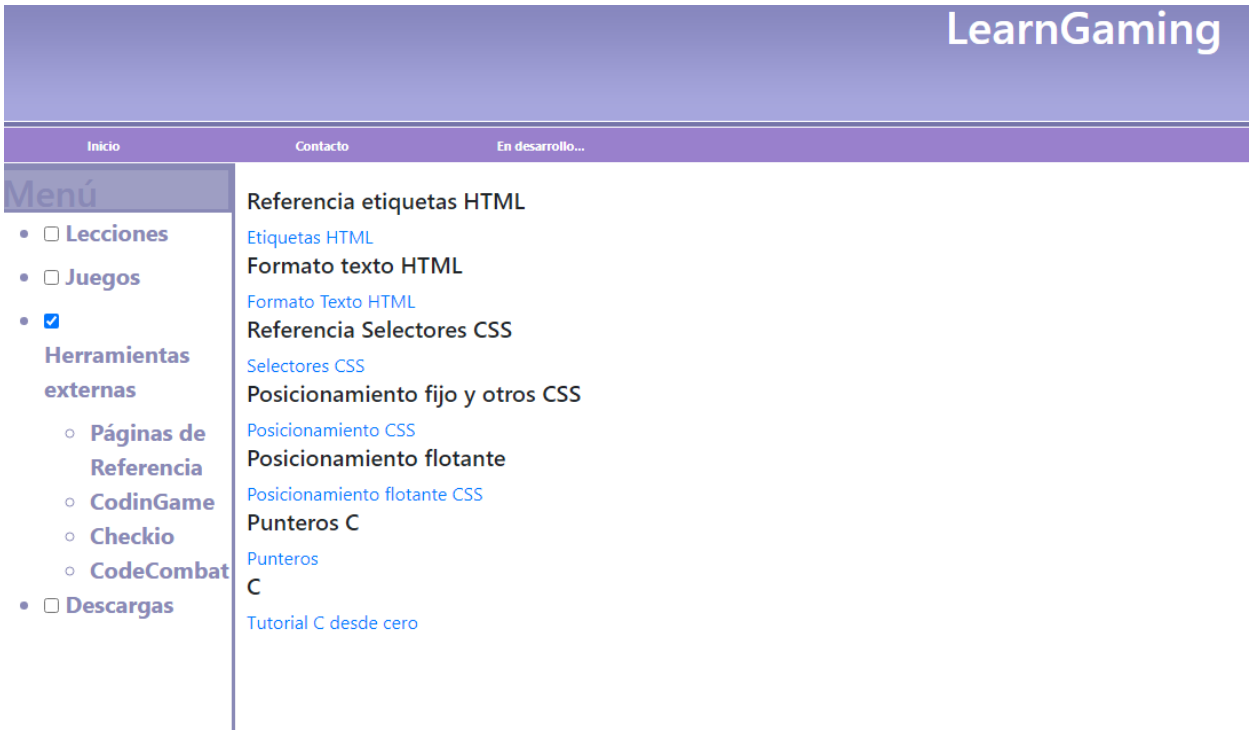

Ilustración 156. Herramientas externas en Web

En el apartado de juegos, actualmente solo existe el juego realizado en este proyecto, Dungeon Code, el cual está incrustado para jugar online, con opción de jugarlo a pantalla completa.

Ilustración 157. Dungeon Code en Web

Por último, tenemos el apartado de descargas, donde se pueden descargar las diferentes versiones del videojuego Dungeon Code para diferentes sistemas operativos.

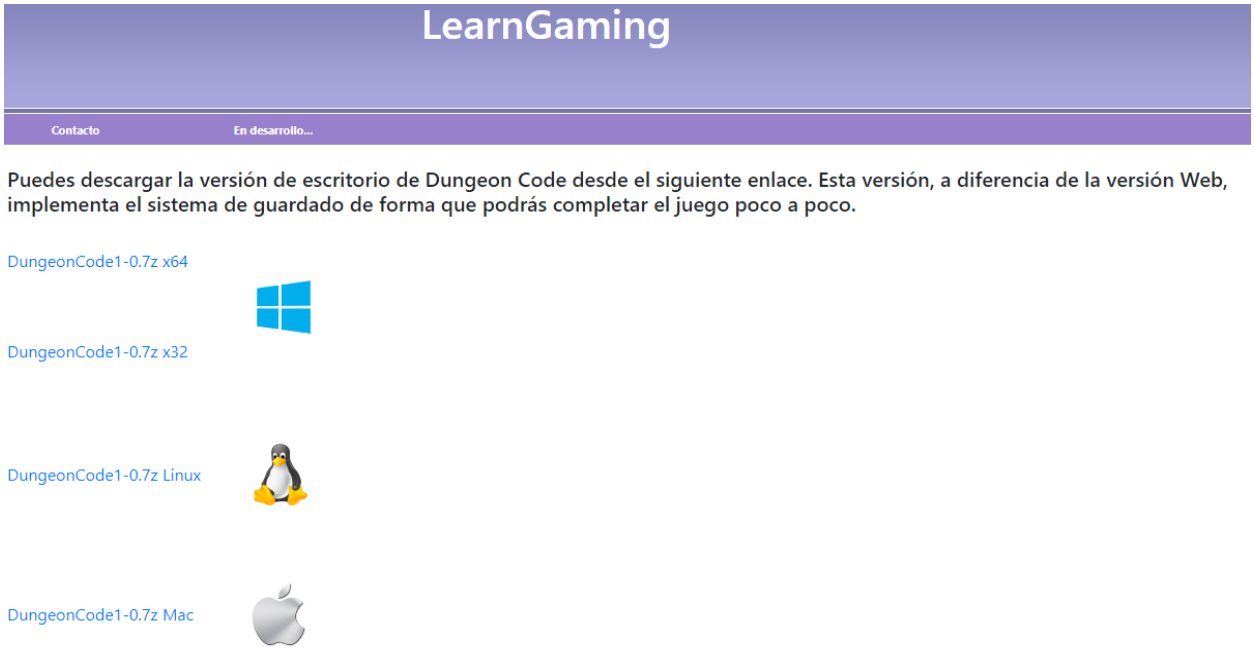

Ilustración 158. Descargas en Web

11.3.1 Servicio Github Pages

GitHub Pages es un servicio muy útil el cual nos permite alojar páginas webs directamente desde nuestros proyectos que estén subidos en nuestro repositorio de GitHub. Este servicio es completamente gratuito, pero no está permitido para webs de negocio o con intenciones comerciales. Además, se prohíben páginas que presenten:

- Contenido o actividades ilegales
- Actividades o contenido violento o amenazante
- Actividades que comprometan usuarios o servicios de GitHub
- Contenido obsceno sexual

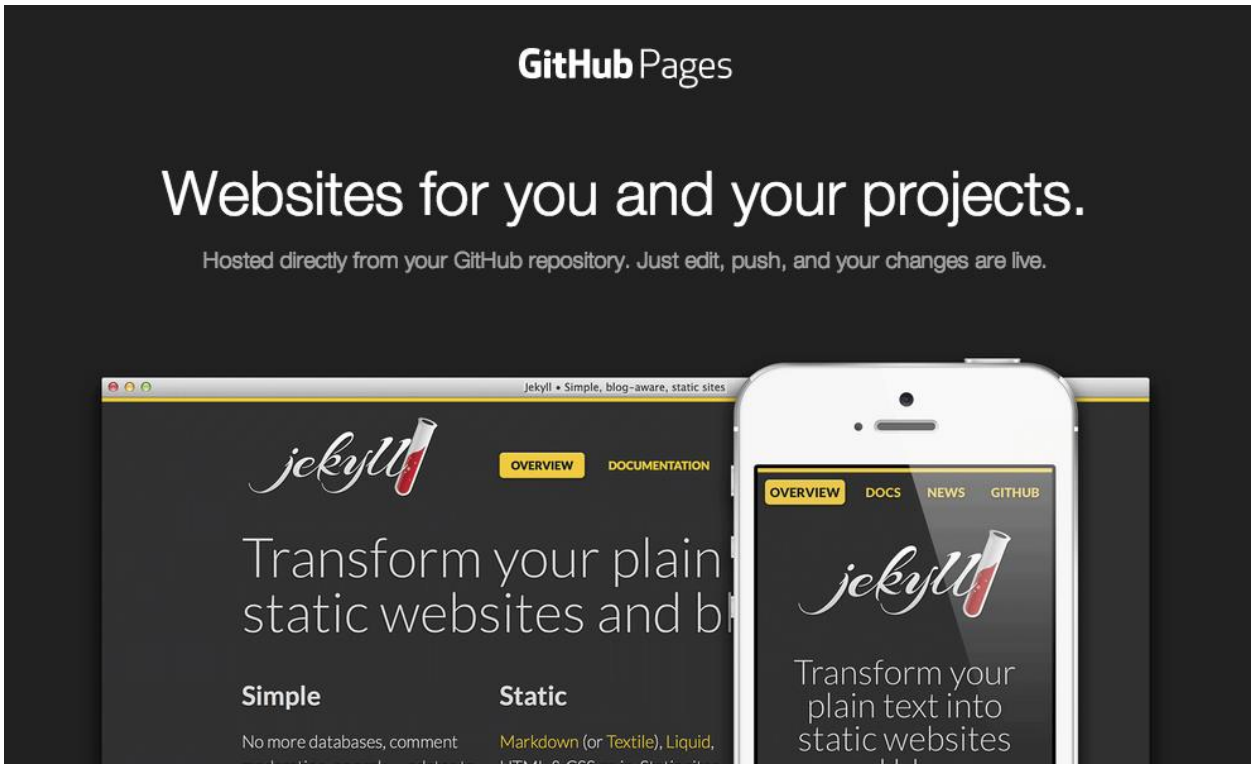

Ilustración 159. GitHub Pages

Para poder hostear nuestra web con GitHub Pages, tenemos que hacer unos pasos previos. Para empezar, tenemos que crear un repositorio que va denominarse de la siguiente forma NombreUsuarioGitHub.github.io.

En mi caso el repositorio será el siguiente:

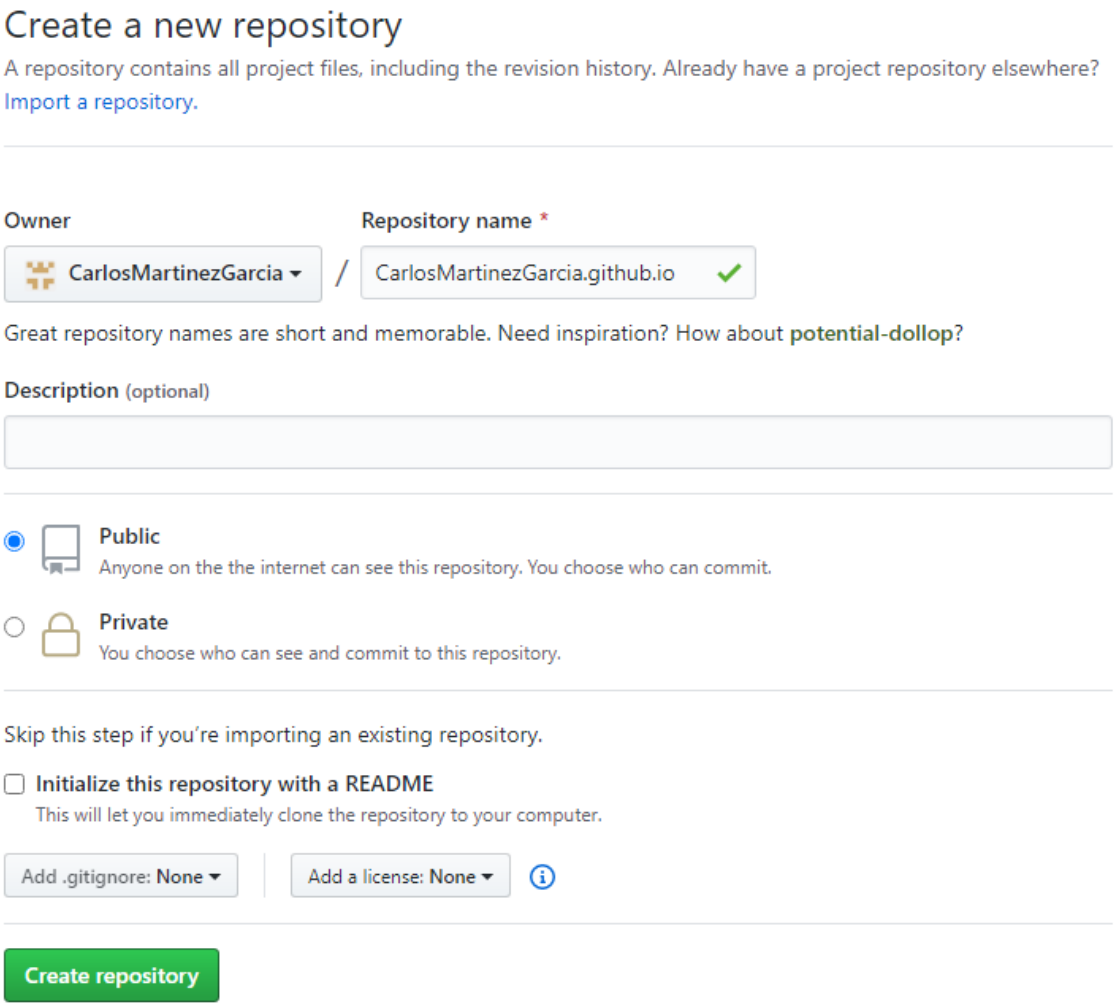

Ilustración 160. Creación repositorio Web

A continuación, nos aparecerá una imagen como la siguiente indicando que hemos creado el repositorio web y que está esperando que se suban los archivos sobre la página Web. En nuestro caso lo importaremos desde otro repositorio donde tenemos ya el código.

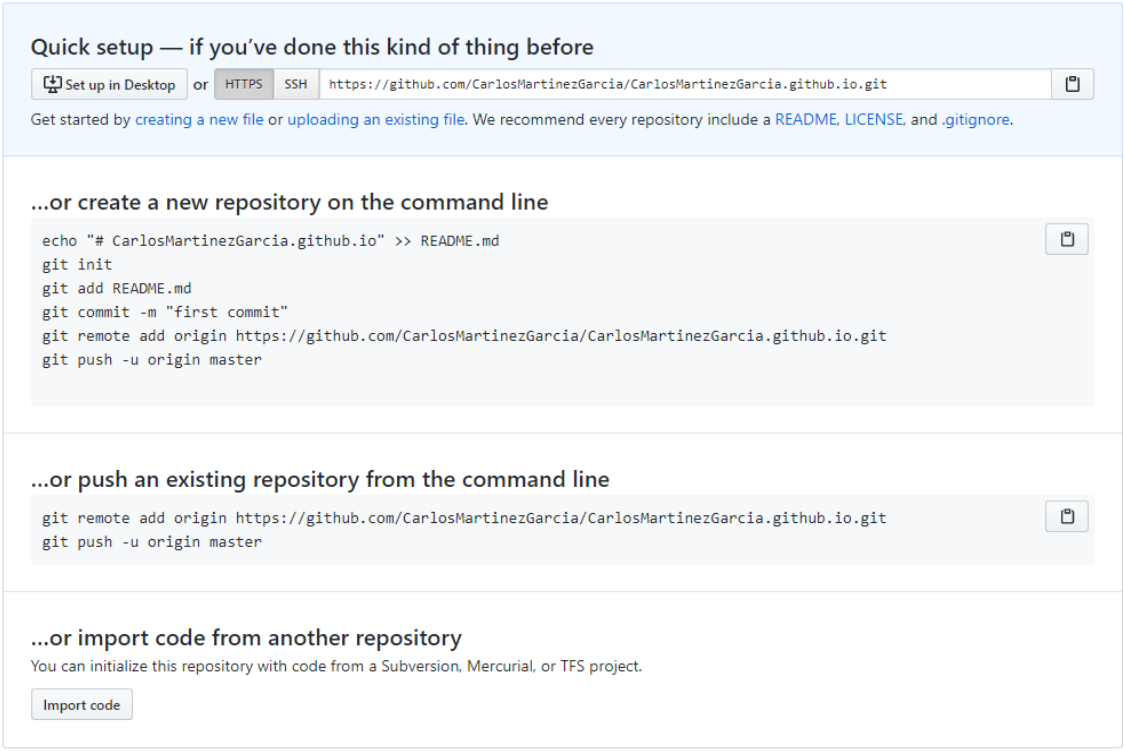

Ilustración 161. Repositorio Web Creado

Desde el apartado de ajustes de GitHub, tenemos un apartado dedicado a GitHub Pages donde podemos comprobar que nuestra web ya está alojada y se nos indica la URL.

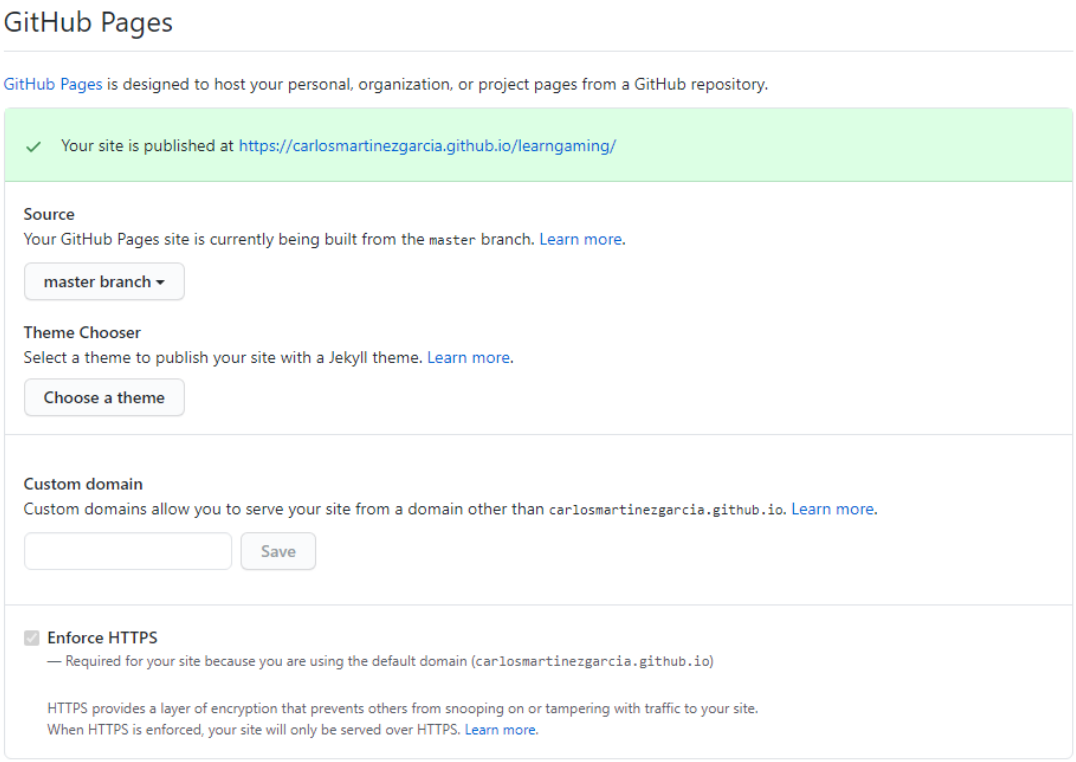

Ilustración 162. Página publicada

Ilustración 163. Página publicada 2