Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Francisco Daniel Pérez Cano y Don Juan José Jiménez Delgado, tutores del Proyecto Fin de Carrera titulado: Desarrollo de un Serious Game, que presenta David Pulido Marchal, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, MES de 2023

El alumno: Los tutores:

David Pulido Marchal NOMBRE _TUTORES

Índice

Índice de ilustraciones

1. INTRODUCCIÓN

En esta sección se intentarán explicar las distintas motivaciones y las diferentes explicaciones de la decisión de este tema y sobre el trasfondo del mismo en la sociedad actual.

1.1. Motivación

Recuerdo mi primera experiencia con un ordenador. Por aquel entonces era normal el uso de máquinas de escribir para realizar documentos, por lo que mis padres decidieron apuntarme a clases de mecanografía. Llegaba, empezaba una lección y acababa y así una y otra vez, poco a poco iba teniendo esa soltura necesaria y cada vez hacía las cosas más rápido, pero para un niño que acababa de entrar a clase y que veía que sus amigos salían a jugar y se lo pasaban en grande pues la verdad es que era como volver a clase otra vez por la tarde. El problema no era en sí hacer una actividad extraescolar, algunos de mis amigos practicaban algún deporte y recuerdo que se lo pasaban muy bien, pero no veía ningún incentivo en esas clases. Al cabo del tiempo de la clase llena solo quedamos la mitad de esta y los pocos que seguíamos íbamos más por el hábito y por el tiempo que ya habíamos invertido más que por el propio placer que suponían las clases o el hecho de la mejora de la habilidad en la mecanografía.

Para sorpresa de mucha gente, a día de hoy con todos los ordenadores instaurados en el día a día y con el auge de los dispositivos inteligentes, solo el 20% de las personas saben escribir correctamente con un teclado mientras que el 80% restante suele escribir mirando el teclado lo que reduce la eficiencia del uso del mismo y disminuye la velocidad a la hora de escribir en un ordenador (Petersen, 2023).

Prácticamente todos los empleos y niveles de educación requieren o se van adaptando para requerir el uso de sistemas informáticos para un desempeño eficiente. Todas las empresas están compitiendo por ser las primeras en lograr una transformación digital, ya que reconocen que esta evolución tecnológica es un factor determinante en su supervivencia y éxito en un mundo empresarial cada vez más dinámico y digitalizado, por lo que lo van instaurando poco a poco en cada una de sus partes y esta tendencia está también instaurándose en el sector educativo.

Asimismo, el 30% de las empresas no contratarían a alguien que no supiera escribir con un teclado y que además lo consideran una habilidad fundamental en la vida (Domínguez, 2014). En el ámbito educativo, un dato relevante es que el 76 % de los niños británicos de edades comprendidas entre los 7 y los 13 años emplea una computadora para realizar sus deberes escolares, mientras que un 24 % no lo hace.

Por lo que hemos visto, la mecanografía es una ciencia necesaria para el uso correcto y eficiente de las TIC (Tecnologías de la Información y la Comunicación) ya que, aunque se pueda usar correctamente los distintos programas o las herramientas que nos proporciona, una gran parte de este pasan por escribir en el ordenador los distintos comandos o por comunicarse a través de distintos medios como correos electrónicos, redes sociales o herramientas de colaboración como es Microsoft Teams.

1.2. Objetivos del trabajo

El objetivo general es el desarrollo de un prototipo de un videojuego para el aprendizaje de mecanografía usando la plataforma de desarrollo en tiempo real de Unity, ofreciendo distintos niveles que guíen al usuario a la hora de aprender poco a poco todas las teclas del teclado y que planten una serie de desafíos que prueben la velocidad y la precisión a la hora de teclear.

1.3. Análisis de software, estudio de alternativas y viabilidad

Antes de empezar a desarrollar el desarrollo del videojuego, como en cualquier proyecto de desarrollo se tiene que evaluar las diferentes tecnologías existentes en el mercado y enumerar las distintas diferencias a nivel económico, documentación, porcentaje de uso.

1.3.1. Motor gráfico

Tras hacer una investigación y según una encuesta realizada por el DEV 2022, los dos motores más usados son Unity y Unreal Engine en la industria del videojuego en España (Asociación Española de Empresas Productoras y Desarrolladoras de Videojuegos y Software de Entretenimiento, DEV, 2022).

A continuación, se van a presentar los diferentes motores gráficos que se han estudiado y las diferencias entre los mismos:

	Unity 3D	Unreal Engine 5
Fecha de lanzamiento inicial	8 de junio de 2005	5 de abril de 2022
Lenguaje de programación	C#	C++
Plataformas compatibles	Windows (PC), Mac, Universal Windows Platform (UWP), Linux Standalone, iOS, Android, ARKit, ARCore, Microsoft HoloLens, Windows Mixed Reality, Magic Leap (Lumin), Oculus, PlayStation VR, PS5, PS4, Xbox One, Xbox X|S, Nintendo Switch, Google Stadia, WebGL Embedded Linux, QNX	Windows PC, PlayStation 5, PlayStation 4, Xbox Series X, Xbox Series S, Xbox One, Nintendo Switch, macOS, iOS, Android, ARKit, ARCore, OpenXR, SteamVR, Oculus, Linux, and SteamDeck
Desarrollado por	Unity Technologies	Epic Games
IDE compatible	Windows, Mac, Linux	Windows, Mac, Linux
Características destacables	Mejoras a los entornos 2D, animaciones, creación de snapshots	Un framework robusto para juegos multijugador, VFX y un simulador de particulas
Código fuente	Not open-source.	No considerado open-source, pero se puede acceder al código del motor
Pricing	Gratis	Gratis
Facilidad de Uso	Curva de aprendizaje sencilla	Difícil de aprender
Gráficos	Buenos gráficos generales, pero menos refinados que Unreal	Gráficos fotorrealistas utilizados en juegos AAA
Otros aspectos	Tiene una comunidad grande y activa, lo que facilita la colaboración y la resolución de problemas, además de tener mucha documentación	

Tabla 1.1 Características de los motores gráficos

- Similitudes:
    - Gratuitos
    - Ambos no son open-source
    - Permiten lanzar el videojuego desarrollado en distintas plataformas, incluidas las más modernas del mercado, así como entornos de realidad virtual
- Diferencias
    - La curva de aprendizaje de Unity es menor

- En Unity hay un mayor número de tutoriales y de guías. La documentación de Unreal está peor organizada y es más liosa de entender
- Unity es más usado dentro de la comunidad española

Con respecto al precio, ambos motores se cobran un porcentaje de regalías por ingresos brutos en ventas de productos comerciales una vez que se alcanza un umbral determinado, pero como en este caso concreto se usa como un recurso educativo o de investigación en lugar de ser un producto comercial seguirá siendo gratuito, pero esto hay que tenerlo en cuenta si se quiere calcular los costes para lanzarlo como un producto comercial.

1.3.2. Entorno de desarrollo

	Visual Studio Code	Visual Studio
Fecha de lanzamiento inicial	29 de abril de 2015	1 de mayo de 1997
Desarrollado por	Microsoft	Microsoft
Programado en	TypeScript, JavaScript, CSS	C++ y C#
Entorno de desarrollo	No, pero posible a través de extensiones	Si
Plataformas	x86, x86-64 y ARM	x86-64
Lenguajes soportados	Multiples (C++, C#, Fortran, .NET, Java, Python, PHP,...)	Multiples (C++, C#, Fortran, .NET, Java, Python, PHP,...)
Código fuente	Técnicamente open-source pero su descarga está privatizada por Microsoft	No considerado open-source, pero se puede acceder al código del motor
Precio	Gratis	Gratuito con opciones de pago
Tamaño	Ligero (200MB)	Pesado (min 2,3GB y 16GB RAM)
Otros aspectos	Altamente personalizable aunque depende mucho de los plugin de la comunidad	Fuerte entorno de pruebas e integración con Azure. La instalación es sencilla e incluye todos los elementos necesarios para el desarrollo. Puede ser más complejo de lo necesario

Tabla 1.2 Características de los entornos de desarrollo

- Similitudes:
  - Gratuitos
  - Ambos pertenecen a Microsoft
  - Pueden ser ejecutados en ordenadores modernos cuyos procesadores funcionen mediante CISC (Complex Instruction Set Computer)
  - Ambos permiten gran cantidad de lenguajes

- Diferencias
    - Visual Studio Code es más ligero que Visual Studio, lo cual permite ejecutarse en ordenadores que tengan menos recursos
    - Visual Studio Code permite ejecutarse en arquitecturas ARM
    - Visual Studio Code es un editor de código que debe gran parte de su soporte a la comunidad que desarrolla las extensiones mientras que Visual Studio tiene un equipo dedicado a desarrollar sus complementos que están bien incorporados

1.3.3. Git GUI client

Git es un sistema de control de versiones distribuido que rastrea las versiones de los archivos. Los programadores que desarrollan software de forma colaborativa suelen utilizarlo para controlar el código fuente (Git, 2024). Git es un software gratuito y de código abierto compartido bajo la licencia GPL-2.0 únicamente.

	SmartGit	Fork
Fecha de lanzamiento inicial	10 de agosto de 2009	17 de abril de 2016
Desarrollado por	syntevo GmbH	Dan Pristupov y Tanya Pristupova
Programado en	Java (SmartGit, 2023)	C#+WPF para Windows y Swift+Cocoa en Mac (Pristupov, 2020)
Sistemas Operativos	Windows, Mac y Linux	Windows and Mac
Precio	Gratuito con licencia no comercial ligado a una institución de investigación. En caso comercial, de pago como mínimo 89€. También ofrece modelo de subscripción	60$ con una evaluación gratuita de uso indefinido
Otros Aspectos	Posee integraciones con otras aplicaciones y servicios como AzureDevOps, Jenkins, JIRA... (syntevo GmbH, 2024)	

Tabla 1.3 Características de los clientes de Git

- Similitudes:
    - Ambos son productos de pago que poseen una licencia gratuita.
    - Pueden ser instalados en Windows.
    - Poseen todas las funcionalidades básicas de Git, así como integración con Github.
- Diferencias:

- SmartGit posee una licencia más cara y además limita las actualizaciones (syntevo GmbH, 2024) mientras que Fork posee una licencia de pago único con acceso hasta 3 dispositivos.
- SmartGit tiene más funcionalidades e integraciones.

1.3.4. Edición de imágenes

A la hora de realizar todos los sprites y los iconos personalizados es necesario usar un editor de imágenes.

	GIMP	Photoshop
Fecha de lanzamiento inicial	15 de febrero de 1996	19 de febrero de 1990
Desarrollado por	syntevo GmbH	Adobe Systems Incorporated
Programado en	C y GTK (GIMP, 2024)	C++
Sistemas Operativos	Cualquier plataforma amd64(Windows, Linux, Mac), ARM, i386, Power PC	Windows, Mac, iPadOS y Android (Adobe Photoshop, 2024)
Precio	Open Source	Subscripción desde 26,43€ al mes (Adobe, s.f.)
Otros Aspectos		En la ultima versión han añadido IA Generativa que para generar partes de la imagen a través de instrucciones y de la imagen actual

Tabla 1.4 Características de los editores de imágenes

- Similitudes:
  - Ambos pueden editar imágenes rasterizadas y soportan varios modelos de colores como RGB, CMYK.
  - Permiten trabajar con capas y mascaras.
  - Pueden ser instalados en los principales sistemas operativos como Windows, Mac y Android.
  - Permiten editar muchos formatos de imagen, como pueden ser JPEG, PNG, TIFF, BMP…
- Diferencias
  - Photoshop posee una subcripción mensual, mientras que GIMP es gratuito.
  - Photoshop tiene una interfaz más pulida con un diseño más intuitivo.
  - Aunque ambos ofrecen una funcionalidad básica, Photoshop posee funcionalidades más avanzadas ya sea en sí mismo o a través de las diferentes aplicaciones de su ecosistema.

1.4. Tecnologías utilizadas

1.4.1. Equipo de desarrollo

Equipo	Asus VivoBook S512DA-BR097	Tower
CPU	Ryzen R5 3500U: 4 Núcleos, 2MB Caché, 2.0GHz hasta 3.6GHz, 64-bit	Intel(R) Core (TM) i5-9600K CPU @ 3.70GHz 3.70 GHz
RAM	8 GB	16GB
Disco duro	*500 GB SSD*	256GB SSD 1TB HDD
Resolución de pantalla	1366x768	3440 x 1440

Tabla 1.5 Especificaciones de los equipos informáticos

Los elementos en cursiva dentro de la tabla han sido marcados así puesto que no iban incluidos originalmente en los equipos, sino que han sido añadidos o han sustituido a otros componentes que el equipo correspondiente llevaba originalmente.

1.4.2. Software empleado

1.4.2.1. Unity

Ilustración 1.6 Logo de Unity

El motor gráfico núcleo central de la funcionalidad de cualquier videojuego. Sin él un renderizado de gráficos 2D o 3D, un motor de físicas, un detector de colisiones y sus respuestas, el sonido, el ensamblado de scripts y su definición de los comportamientos relacionados con los distintos objetos, la generación de sombras, el uso de animaciones, el uso de inteligencia artificial, juego o funciones online no serían posibles.

Normalmente los motores gráficos proveen una serie de herramientas de desarrollo visual que están provistos por un entorno de desarrollo para poder acceder de forma rápida y sencilla a estas. Muchos desarrolladores intentan

anticiparse a las necesidades e implementan paquetes de software con funcionalidad como IA, física, gráficos o sonido. Estos paquetes normalmente se conocen como middleware porque como en el sentido comercial del término, proporcionan una plataforma de software flexible y reutilizable con una funcionalidad básica necesaria lista para usar lo cual consigue reducir costes y tiempo de desarrollo (O'Neill, 2008). Los motores de videojuegos suelen proveer de abstracción de plataforma, lo que permite que el mismo juego se ejecute en varias plataformas (incluidas consolas de juegos y computadoras personales) con pocos cambios, si es que se realizan alguno, en el código fuente del juego. Algunos de los componentes del motor se pueden quedar obsoletos por lo que se suelen dedicar algunos desarrolladores a actualizar ya sea el componente obsoleto como modificar el motor para desarrollar la funcionalidad existente.

Algunas veces estos cambios se van agrupando con el tiempo creando un árbol de desarrollo como pasó por ejemplo con el motor Quake de id Software:

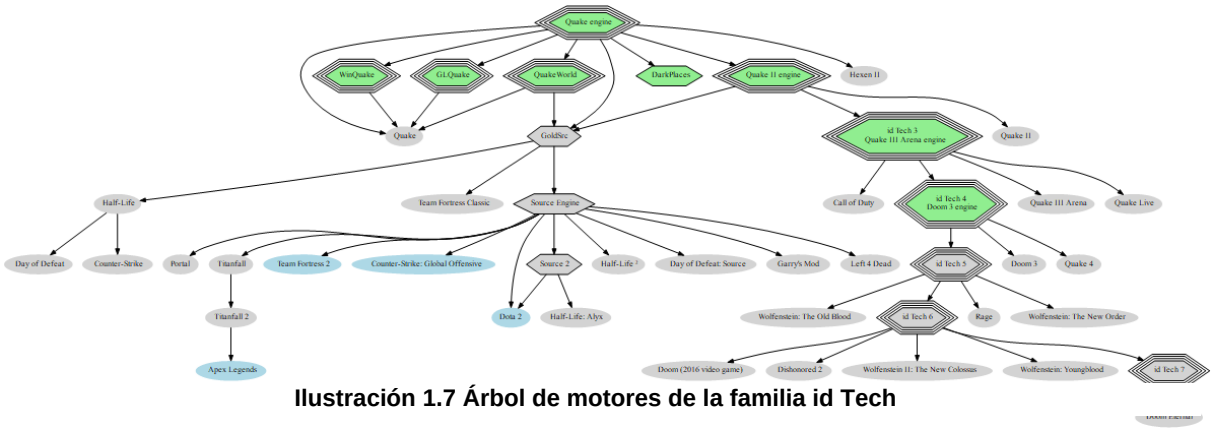


Ilustración 1.7 Árbol de motores de la familia id Tech

Unity es un motor de videojuego que utiliza como lenguaje de programación C# que con su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes (C Sharp, 2024), como gráficos puede usar OpenGL, Direct3D o OpenGL ES, además Unity es conocido por su facilidad de uso, versatilidad y amplia gama de características y herramientas que facilitan la creación de contenido interactivo de alta calidad. En principio tiene 4 sistemas de suscripción hasta la fecha con distintos precios y funcionalidades [H]:

Luego de contrastar los distintos rasgos de los motores se ha decidido usar **Unity Engine**. Lo primero de todo, este motor se ha usado anteriormente en la carrera en la asignatura de Desarrollo de Videojuegos por lo que se dispone más experiencia y por tanto no hay que pasar por un proceso de aprendizaje inicial. Seguidamente, el hecho de tener mucha documentación y una comunidad amplia y activa hace que el proceso de aprendizaje y resolución de problemas sea más accesible y eficiente lo cual reducirá el tiempo de desarrollo y por tanto sea más fácil. Finalmente, el hecho de ser el motor más usado para las distintas empresas de videojuegos de España hace probable que se tenga más documentación en español, lo cual puede suponer una ventaja si se quiere proponer este producto a alguna empresa española para poder continuar el desarrollo

1.4.2.2. Visual Studio Code

Ilustración 1.8 Logo de Visual Studio Code

Para toda la parte de la programación se planteó usar un IDE que se había usado anteriormente para desarrollar videojuegos con Unity, Visual Studio 2019. Sin embargo, aunque dispone de muchas facilidades y perfecta compatibilidad con Unity, a su vez es un programa muy pesado y por tanto ralentizaba el proceso de programación, haciendo de todo el proceso de desarrollo muy ineficiente. Por lo que se planteó una alternativa: Visual Studio Code.

Visual Studio Code es un editor de código fuente que no un entorno de desarrollo integrado desarrollado por Microsoft para Windows, Linux, macOS y Web gratuito y de código abierto basado en Electron, un framework que se utiliza para implementar Chromium y Node.js como aplicaciones para escritorio. Es bastante

más ligero que Visual Studio y permite la instalación de una amplia variedad de extensiones disponibles para agregar funcionalidad adicional según las necesidades. Visual Studio Code es conocido por ser rápido y eficiente en términos de recursos.

Por tanto, para este proyecto viendo las características del equipo que se ha estado utilizando y por la existencia de dos distintas extensiones que son Unity for Visual Studio Code y C# Dev Kit se ha decantado por este editor de código fuente.

1.4.2.3. Fork

Ilustración 1.9 Logo de Fork

Aunque Visual Studio Code tiene una extensión que permite el uso de ramas dentro del mismo se prefiere un entorno visual para la gestión de las distintas ramas que se emplean en el proyecto. Inicialmente en la carrera se usó SmartGit pero había un par de problemas. El primero fue la licencia de uso, si bien actualmente se posee una licencia de uso no comercial gracias a ser estudiante, tras entregar el proyecto dejaríamos de tener acceso a esta licencia y además si luego se intenta comercializar este prototipo estaríamos incumpliendo la misma.

Por lo tanto, se buscó una herramienta fácil, moderna que posea todos los requerimientos típicos, que sea altamente actualizado. Asique este programa que si bien tiene un coste de 59,99$ se puede usar perfectamente con su evaluación gratuita indefinida.

1.4.2.4. GIMP

Ilustración 1.10 Logo de GIMP

GIMP es un editor de gráficos rasterizados gratuito y de código abierto que se utiliza para la manipulación o retoques y edición de imágenes, dibujo de forma libre, transcodificación entre diferentes formatos de archivos de imagen y tareas más especializadas. Es extensible mediante complementos y se puede programar. No está diseñado para usarse para dibujar, aunque algunos artistas y creadores lo han utilizado de esta manera. Es multiplataforma, lo que significa que puede ejecutarse en varios sistemas operativos, incluyendo Windows, macOS y Linux.

La alternativa a este software era usar Adobe Photoshop, un software propietario desarrollado y publicado por Adobe Inc. Es un editor de fotografías, usado principalmente para el retoque de fotografías y gráficos. Es ampliamente considerado como el estándar de la industria en cuanto a edición de imágenes y gráficos, y es utilizado por profesionales en diversos campos, incluyendo diseño gráfico, fotografía, web, y producción de vídeo. Photoshop puede editar y componer imágenes rasterizadas y soporta varios modelos de colores como RGB, CIELAB, CMYK y para ello ha desarrollado un formato propio como PSD y PSB.

Los principales motivos para escoger GIMP son que es un software gratuito y aunque puede carecer de algunas de las funcionalidades más avanzadas presentes en Photoshop como el uso de IA Generativa, tiene las funcionalidades necesarias para desarrollar un prototipo y una comunidad suficientemente grande como para desarrollar extensiones y plugins y realizar tutoriales que permiten desarrollar los assets necesarios en un videojuego.

1.4.2.5. Audacity

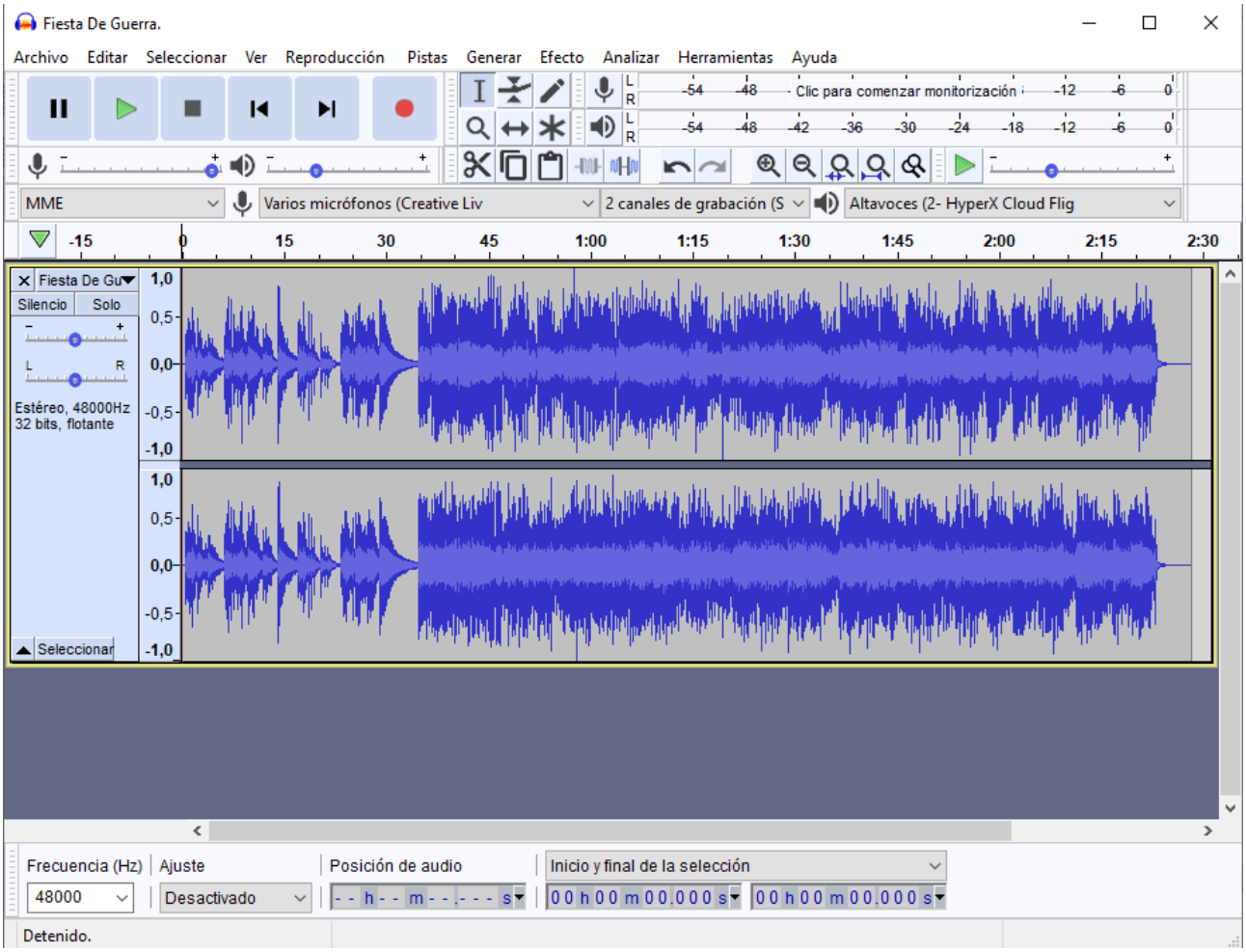


Ilustración 1.11 Interfaz de Audacity

Audacity es una aplicación informática multiplataforma libre que se puede usar para grabación y edición de audio, Se ha utilizado para la normalizar, editar y recortar todos los audios usados en el videojuego.

1.4.2.6. Probuilder

Ilustración 1.12 Interfaz de Probuilder

Probuilder es un plugin desarrollado para Unity, es un híbrido entre una herramienta de modelado 3D y un diseñador de niveles. Los objetos 3D usados en el videojuego han sido modelados a través de esta herramienta.

1.5. Metodología de desarrollo

El modelo incremental es un enfoque de desarrollo de software donde el producto se diseña, implementa y prueba en incrementos pequeños y manejables. En lugar de entregar el producto en su totalidad este se divide en incrementos de tal modo que cada entrega es un producto operativo pero incompleto cuyo periodo de tiempo es breve. Los incrementos se estructuran de forma que los requisitos principales se desarrollan en los primeros incrementos. Cada incremento pasa por un ciclo completo de desarrollo, que incluye el análisis, diseño, implementación y pruebas. Los requisitos del incremento no suelen ser modificados hasta que ha sido terminado, aunque puede ser común surgir cambios de un incremento a otro, se suelen incluir en la planificación de un incremento posterior (Tpoint Tech, 2011), (Ureña Lopez & Gómez Espínola, 2016).

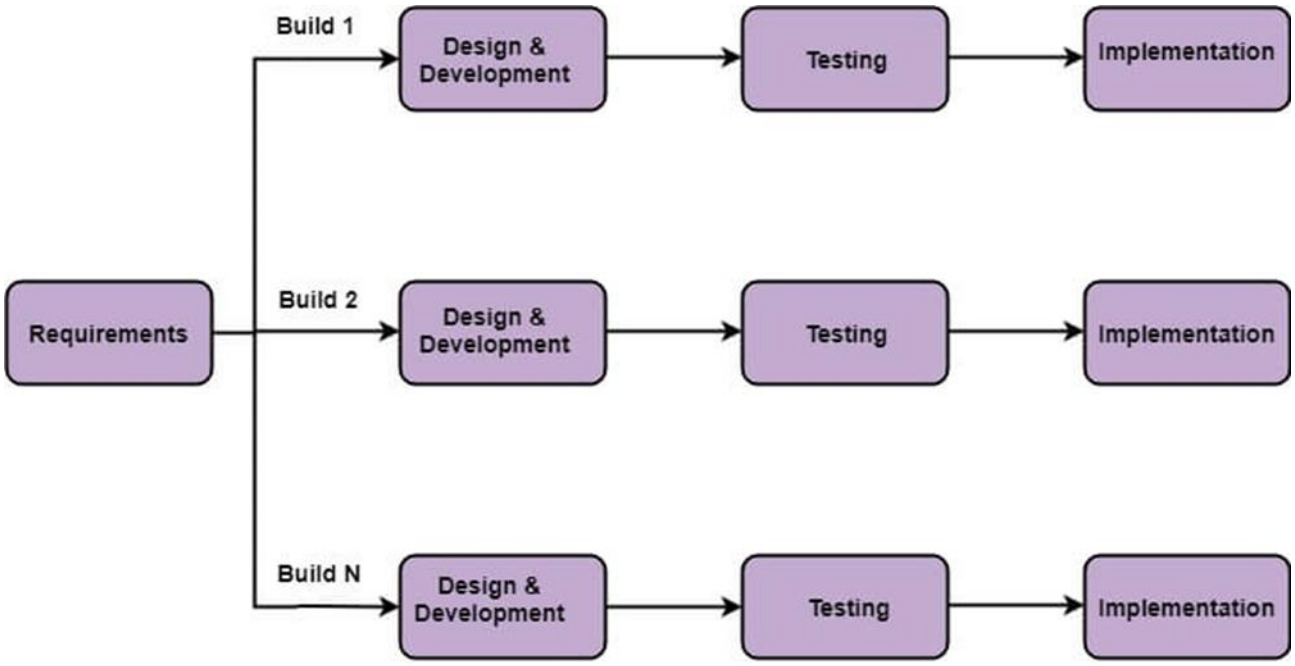


Ilustración 1.13 Funcionamiento de un modelo incremental

Este proceso de desarrollo de software tiene una serie de ventajas que podemos definir como:

- Los errores de los requisitos importantes son reconocidos en etapas tempranas de desarrollo.
- Los clientes pueden usar el software que poseen las características más importantes en etapas tempranas y por tanto dar feedback relevante
- El riesgo de que el proyecto no llegue a una etapa final disminuye
- Los incrementos primordiales poseen un mayor número de pruebas.
- La carga de trabajos es más manejable y permite distribuir más fácilmente el trabajo
- La gran cantidad de entregas permite que el cliente se mantenga involucrado con el proyecto

Y las principales desventajas son las siguientes:

- Requiere de una gran capacidad de planificación
- Dificultad a la hora de acomodar determinados requisitos en un incremento

1.6. Alcance

Una vez definidos los objetivos del proyecto, se debe determinar qué documentos se incluirán:

- **Ejecutable operativo**: Es un videojuego compilado que debe ser entregado en la fecha fijada y cumplir con todos los requisitos listados. Será accesible a través de internet descargándolo a través del repositorio de github o a través del ejecutable subido a la plataforma.
- **Código fuente**: En la entrega final se incluirá el código fuente necesario, no solo para arrancar el videojuego, sino también para su consulta, dado que se hace referencia a este a lo largo del documento. La etapa de desarrollo se explica detalladamente en el punto 4.
- **Manual de instalación**: Al final del documento se incluye una sección que explica cómo descargar y ejecutar el entorno de desarrollo del videojuego. (TODO ANEXO)

- **Manual de usuario**: Al final del documento existe otro apartado que detalla los controles y funcionalidades de la interfaz de cada escena. (TODO ANEXO)

2. ANÁLISIS

2.1. Requisitos

En ingeniería, un requisito es una condición que debe cumplirse para que el resultado de un trabajo sea aceptable. Es una descripción explícita, objetiva, clara y a menudo cuantitativa de una condición que debe cumplir un material, diseño, producto o servicio (Requirement, 2024).

El Glosario estándar IEEE de terminología de ingeniería de software define un requisito como (The Institute of Electrical and Electronics Engineer, 1991):

1. Una condición o capacidad que necesita un usuario para resolver un problema o alcanzar un objetivo
2. Una condición o capacidad que debe cumplir o poseer un sistema o un componente del sistema para satisfacer un contrato, una norma, una especificación u otro documento impuesto formalmente
3. Una representación documentada de una condición o capacidad como en 1 o 2

Los requisitos en ingeniería de software suelen clasificarse en dos categorías:

- **Requisitos funcionales**: Describen lo que hace un sistema o lo que se espera que haga, es decir, su funcionalidad. Suelen ir acompañados de diagramas de casos de uso. Una función es descrita como un conjunto de entradas, comportamientos y salidas.
- **Requisitos no funcionales**: describen aspectos del sistema que están relacionados con el grado de cumplimiento de los requisitos funcionales. Por tanto, se refieren a todos los requisitos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento. Suelen ser las restricciones o condiciones que impone el cliente al programa que necesita.

2.1.1. Requisitos Funcionales

1. **Interfaz de Usuario**: Conjunto de elementos que permiten al jugador seleccionar distintas opciones y que informa al jugador de los distintos datos que posee el videojuego
    1. Menú principal: El prototipo dispondrá de una pantalla de inicio que que permitirá al jugador seleccionar entre los distintos modos de juego y configurar las opciones
    2. IU: Configuración: Se dispondrá de un menú accesible desde el menú principal y en cualquier momento en cualquier modo de juego que permite configurar el sonido del juego
    3. IU: Pantalla de juego: En cualquier modo de juego se dispondrá de una interfaz de usuario clara que muestre al menos la puntuación y cualquier sistema que implique una condición de derrota
    4. IU: Modos de Juego: Se dispondrá de un menú propio para configurar algún parámetro necesario en el modo de juego
2. **Mecánicas comunes**: Conjunto de funcionalidades principales que serán comunes entre los distintos modos de juego.
    1. Generación de letras: Los elementos a destruir, conocidos como “letras”, deberán de salir en la parte superior de la pantalla pudiendo configurar su frecuencia. Además se debe de ser capaz de configurar para que su generación sea determinada.
    2. Eliminación de letras: Las “letras” deberán de ser eliminadas al pulsar una tecla concreta mientras mantengan contacto con una zona concreta del escenario conocida como “hit zone”.
    3. Número de fallos: Existe un sistema que cuenta el número de “letras” que no se ha logrado eliminar y que al alcanzar un cierto umbral indica el fin del juego
    4. Movimiento de las letras: Las “letras” deben de bajar con una velocidad configurable
3. **Puntuación**: Cualquier acción del jugador en los distintos modos de juego deben de ser reflejados en la puntuación

1. Acierto y combos: Cuando se elimina una “letra” se deberá de aumentar la puntuación del juego. En caso de encadenar varios aciertos este aumento será mayor.
2. Errores: En caso de que una “letra” no sea eliminada, se deberá de restar un número determinado a la puntuación del juego. En caso de que se pulse una tecla incorrecta también se reducirá la puntuación del juego.
3. Leaderboard: Se debe de disponer de un registro de puntuaciones ordenado indicando el nombre o apodo del jugador, así como la puntuación obtenida en un modo de juego concreto con unos parámetros concretos.

4. **Modo de juego**: Random: En este modo de juego se deberán de generar las “letras” de forma aleatoria o siguiendo unas distribuciones porcentuales predefinida
    1. Alfabetos: El número de alfabetos que serán posibles para escoger será el español e inglés
    2. Modos de dificultad: Se podrá escoger entre 3 modos de dificultad que modificarán la velocidad y el tiempo de generación de las “letras”.
5. **Modo de juego**: Niveles: Existirá un modo de juego en el cual se podrá seleccionar un nivel concreto que será pre-configurada y generará las “letras de forma” determinada.
    1. Selección de niveles y mundo: Los niveles estarán divididos en mundos a través de un menú

2.1.2. Requisitos No Funcionales

Se proponen los siguientes requisitos no funcionales:

- El prototipo deberá de tener un ejecutable que pueda ser ejecutado en Windows
- El prototipo deberá de tener una interfaz adaptada a 16:9 sin importar la resolución
- El juego debe de proporcionar feedback inmediato sobre las acciones del usuario ya sea de forma visual o de forma auditiva

- Se deberá de disponer de una guía de controles indicando las diferentes opciones
- Todas las funciones del código deberán de estar documentadas
- La interfaz deberá de tener un código de colores que sea compatible con los distintos tipos de daltonismo
- El sistema de puntuaciones deberá de ser persistente entre sesiones.

2.2. Historias de Usuario

Una Historia de Usuario (HU) (Menzinsky, y otros, 2022) describe una funcionalidad que debe incorporar un sistema de software y aporta valor al cliente. Están descritas en lenguaje natural. Los valores por defecto que debería de tener una historia de usuario son las siguientes:

- Descripción: Suele tener una estructura de 3 campos
  - Como [rol de usuario].
  - Quiero [objetivo].
  - Para poder [beneficio].
- Estimación: aproximación del tiempo para implementar la historia, normalmente medido mediante puntos de historia (PH)
- Prioridad: se suele calcular mediante un sistema e indica el orden de implementación de las historias.

Para este prototipo se añadirán dos campos opcionales:

- ID: identificador único de la historia de usuario
- Valor de negocio: Es una cuantificación de la aportación de la historia de usuario al cliente. Este campo junto con el de Estimación permitirán calcular la Prioridad.

Por tanto, definidos los campos, el listado de las historias de usuario es el siguiente:

ID	Título	Descripción	PH	Valor
1	Menú sencillo	• Como usuario • Quiero que el juego disponga de un menú sencillo • Para poder usar mis conocimientos previos		
2	Selector de volumen	• Como usuario • Quiero poder bajar el volumen • Para establecer el volumen ideal		
3	Selector de nivel	• Como usuario • Quiero seleccionar un nivel • Para poder jugar un nivel en concreto		
4	Username	• Como usuario • Quiero introducir mi nombre • Para poder grabar mi puntuación		
5	Puntuación	• Como usuario • Quiero grabar mi puntuación • Para competir con mis amigos		
6	Reinicio nivel	• Como usuario • Quiero poder reiniciar el nivel • Para volver a intentar un fallo sin tener que salir al menú		
7	Menú principal	• Como usuario • Quiero tener la opción de volver al menú • Para cambiar de modo de juego		
8	Efectos auditivos	• Como usuario • Quiero que al pulsar una tecla halla un indicativo auditivo • Para comprobar si he realizado una acción correcta o no		
9	Ritmo	• Como usuario • Quiero las letras coincidan con la música • Para acertar de forma más sencilla		
10	Dificultad	• Como usuario • Quiero que existan varios niveles de dificultad • Para poder escoger jugar		

según mi habilidad

2.3. Planificación

2.3.1. Tareas

Definidos los requisitos funcionales se generan tareas para lograr llevar a cabo dicho objetivo, realizando varias subtareas:

ID	Tarea	Subtareas	Procedentes	Duración (días)
A	Estudio Previo	1. Análisis del problema 2. Estudio de metodología y tecnologías	-	6
B	Planificación	1. Obtención de requisitos 2. Planificación de tareas 3. Estimación de costes 4. Diseño de diagramas e interfaz	A	10
C	Loop jugable	1. Crear escena principal con elementos estáticos 2. Añadir todas las “letras” 3. Crear script de funcionalidad de bajada y eliminación	B	2
D	Definir sistema generación	1. Definir función de generación 2. Guardar datos en un archivo csv	C	2
E	Añadir música y sonido	1. Investigar como generar audio en Unity 2. Implementar música de fondo 3. Implementar sonidos	C	2
F	Limpieza de elementos	1. Definir un colider para eliminar las letras no pulsadas	C	0,5
G	Alfabetos	1. Leer e investigar estudios sobre el porcentaje de usos de las letras en inglés y español 2. Implementar uso de letras por porcentaje	-	1
H	Salud	1. Implementar sistema de salud 2. Implementar barra visual de vidas	C	1
I	Loop jugable II	1. Realizar modificaciones visuales del escenario 2. Implementar feedback visual 3. Añadir pausa al nivel	C	1
J	Score	1. Añadir funciones de Score y combo 2. Visualizar Score y Combo en UI	I	1
K	UI	1. Diseñar la interfaz en todas las pantallas 2. Añadir iconos 3. Implementar un menú principal que permita acceder al loop jugable 4. Implementar un menú de	E, I	4

		pausa 5. Implementar un slider para cada tipo de volumen. 6. Corregir errores sonido		
L	Loop jugable III	1. Mejorado sistema de colisiones 2. Implementado fin de juego 3. Sistema de dificultad 4. Velocidad y tiempo de generación en csv	D, I	2
M	Leaderboard	1. Añadido sistema de usuarios 2. Definido e implemntado leaderboard 3. Leaderboard por modo de juego 4. Modificado UI de fin de juego para tener leaderboard	L. K	4
N	UI II	1. Añadido Menú de Modo Random 2. Feedback visual los botones y letras	K, M	1,5
O	Letras 3D	1. Diseño 3D de letras 2. Modificar escena para más efecto inmersivo	L	0,5
P	UI III	1. Implementar menú de selección de niveles 2. Mejoras en interfaz del loop jugable 3. Implementar paleta de colores	N	3
Q	Niveles	1. 5 nivel de prueba 2. Función de cálculo de tiempo 3. Modo Debug para creación de niveles	C	5
R	Modo Random	1. Implementar diccionario inglés 2. Corregir errores de generación	G, L	1

Tabla 2.1 Descripción de las tareas totales

Incrementos:

INCREMENTO	INICIO	FIN	TAREAS
0	20/05/24	10/06/24	A, B
1	11/06/24	12/06/24	C
2	13/06/24	20/06/24	D, E, F, G
3	21/06/24	01/07/24	H, I, J, K
4	02/07/24	11/07/24	L, M, N, O
5	12/07/24	23/07/24	P, Q, R

Tabla 2.2 Especificaciones de incrementos

2.3.2. Diagrama de Gantt

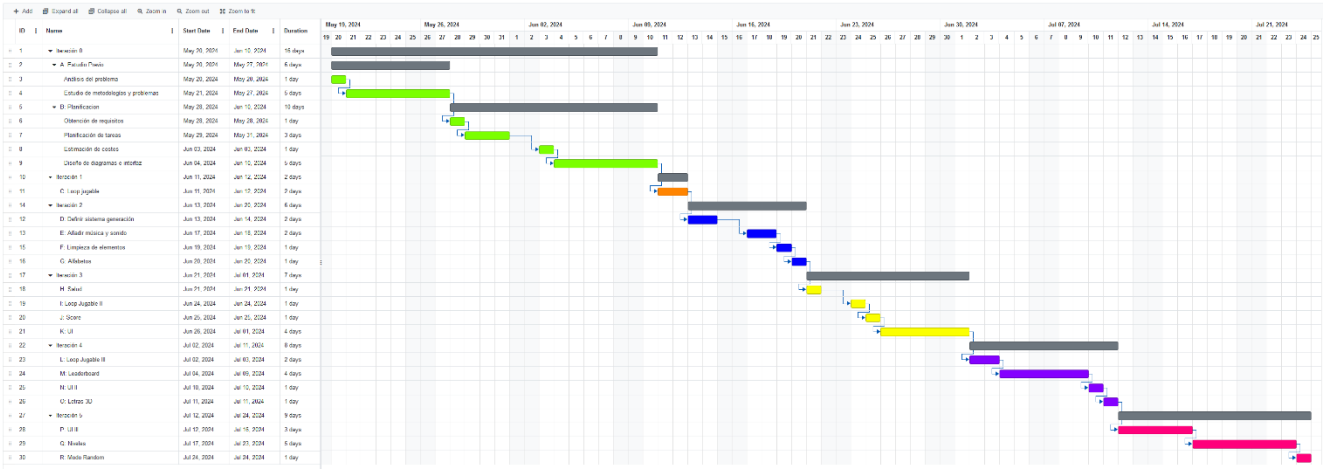


Ilustración 2.3 Diagrama de Gantt

El anterior diagrama se muestran todas las tareas de forma secuencial por la realización de este proyecto por una sola persona. Aquí un enlace para ver la imagen completa:

https://drive.google.com/drive/folders/174HSGGI0nout7JrUOhjBE_jrxHyFQQd6?usp=sharing

2.3.3. Variaciones

- El sistema que ha sido modificado más veces ha sido el script de bajada principal del loop jugable. Al principio estaba diferenciada por los distintos modos de juego y tenía una estructura simple de dos clases, en la tarea I fue modificada una vez intentando eliminar esas diferenciaciones y fue modificada otra vez en la tarea L añadiendo una tercera clase.
- La tarea O que en principio era sencilla, fue más complicada de implementar de lo necesario. Se tuvo que cambiar el planteamiento general de la escena y muchos más elementos fueron modificados haciendo un retraso considerable de tres días más.
- Se tuvo que corregir varios fallos de sonido que se encontraban en la implementación del sonido del incremento anterior, retrasando ligeramente el tiempo.

2.4. Estimación de costes

En cualquier proyecto y sobre todo en un prototipo es importante realizar una propuesta en función de las necesidades del proyecto de cara a un posible inversor. En esta propuesta se suele introducir todos los costes asociados, desde los más obvios como el salario de los trabajadores por puesto, la compra o alquiler de los equipos informáticos, el alquiler de un local y otros costes.

Para la realización del prototipo se ha propuesto un equipo de dos personas o una persona que realiza dos puestos:

- Un programador
- Un analista

Basándose en la planificación anterior el analista tiene una carga de trabajo de 128h. Según Indeed, el sueldo promedio de este puesto es de 16€/h y por tanto el coste total es de 2048€

Por otra parte, el programador ha trabajado 256h siendo el precio de 10€/h según Indeed por lo que el coste total es de 2560€

Puesto Trabajador	Coste mensual (8x5)	Coste (€/h)	Tiempo trabajado	Coste Total
Analista	2.560€	16€	128h	2.048€
Programador	1.600€	10€	256h	2.560€
Total				4.608€

Tabla 2.4 Salario de los trabajadores del proyecto

Como bien se ha descrito en el apartado 1.4.2, gracias a que el software ha sido programas gratuitos o bien open-source no hay costes en este apartado

Un equipo informático promedio suele tener un valor de 800€ y se estima que puede llegar a 5 años de vida útil. Por tanto, los costes derivados son los siguientes.

Valor hardware	Vida útil (años)	Amortización mensual	Amortización en 3 meses
800€	5	13,33€	40€
Total			40€

Tabla 2.5 Tabla de amortizaciones

Finalmente, el presupuesto total es la suma de los costes presentados anteriormente.

Descripción del coste	Valor (€)
Coste de personal	4.608€
Coste hardware	40€
Subtotal	4.648€
Costes Indirectos (5%)	232,4€
Margen (10%)	464,8€
Beneficio (10%)	464,8€
Antes de impuestos	5.810€
IVA (21%)	1.278,2€
Presupuesto total	7.088,2€

Tabla 2.6 Tabla de costes totales

Hay 3 puntos que se le han sumado al coste de desarrollo:

- Los costes indirectos, que suman el alquiler del espacio de trabajo, luz e internet.
- Un margen que garantiza que, en caso de cualquier imprevisto de desarrollo o un fallo de planificación, se pueda asumir el coste
- Un margen de beneficio

Calculados todos los gastos hay que sumarle los impuestos, como mínimo el IVA.

3. DISEÑO

En esta sección se procede a explicar cómo está estructurado el prototipo, presentando primero las posibles acciones en cada escena o menú a través de los diagramas de caso de uso, luego se describe la interfaz indicando los distintos aspectos de diseño. Finalmente se establece un storyboard con el flujo principal.

3.1. Diagramas de casos de uso

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su iteración con otros usuarios o sistemas. El usuario tiene dos principales acciones:

- Interactuar con el sistema o interfaz
- Cambiar de escena

Por tanto, al haber en el videojuego solamente dos escenas podemos establecer dos grandes casos de uso. Menú principal y pantalla de juego.

Para una correcta comprensión de este diagrama hay varios subsistemas que serán explicados en diagramas separados, se indicarán de otro color mediante una relación de extensión.

3.1.1. Menú principal

En esta escena solo se puede realizar varias opciones que son similares, abrir varios submenús:

- Abrir varios menús de selección de modo de juego:
- Modo random
- Modo niveles
- Abrir el menú de configuración que actualmente solo modifica los volúmenes de audio
- Introducir el nombre de usuario, que permite usarlo que permite usarlo en las puntuaciones y que guarda los ajustes de sonido
- Salir del juego

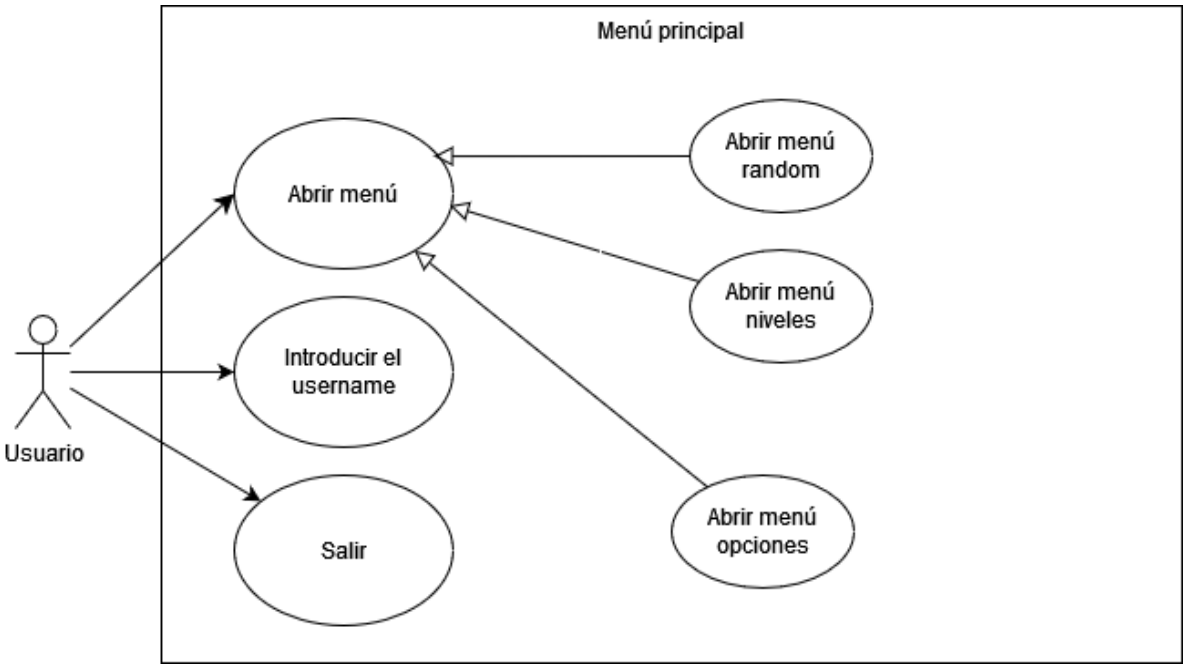


Ilustración 3.1 Diagrama de caso de uso del menú principal

3.1.2. Menú random

El flujo del caso de uso consiste en elegir entre los parámetros que existen y finalmente iniciar el juego. Los diferentes parámetros que actualizar son los siguientes:

- Dificultad para elegir entre 3 opciones (Fácil, Medio y Difícil)
- Idioma si se quiere usar los porcentajes de uso de inglés o español o se quiere un modo completamente aleatorio

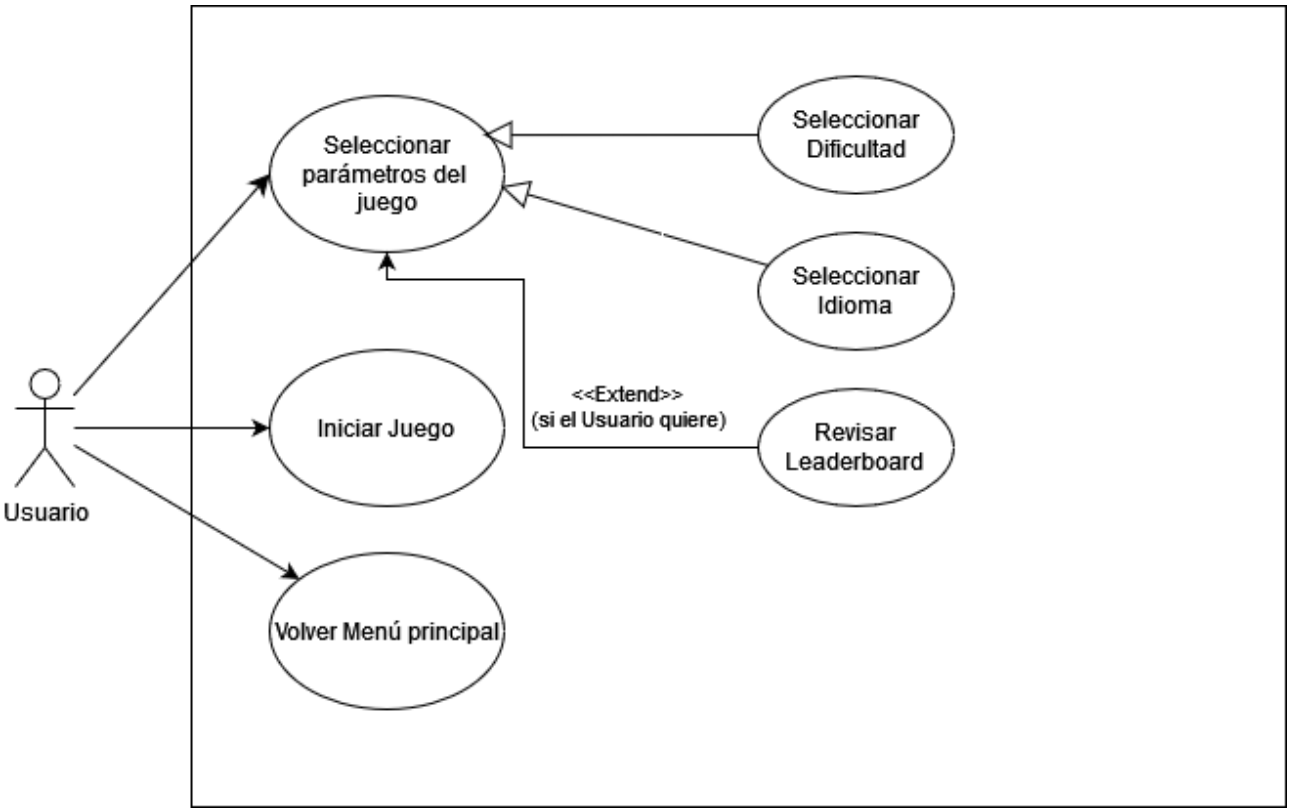


Ilustración 3.2 Diagrama de caso de uso del submenú Random

3.1.3. Menú niveles

El flujo de trabajo es parecido al anterior, la diferencia radica en que los parámetros ahora se distribuyen en modo árbol:

- Seleccionar Mundo

- Seleccionar Nivel

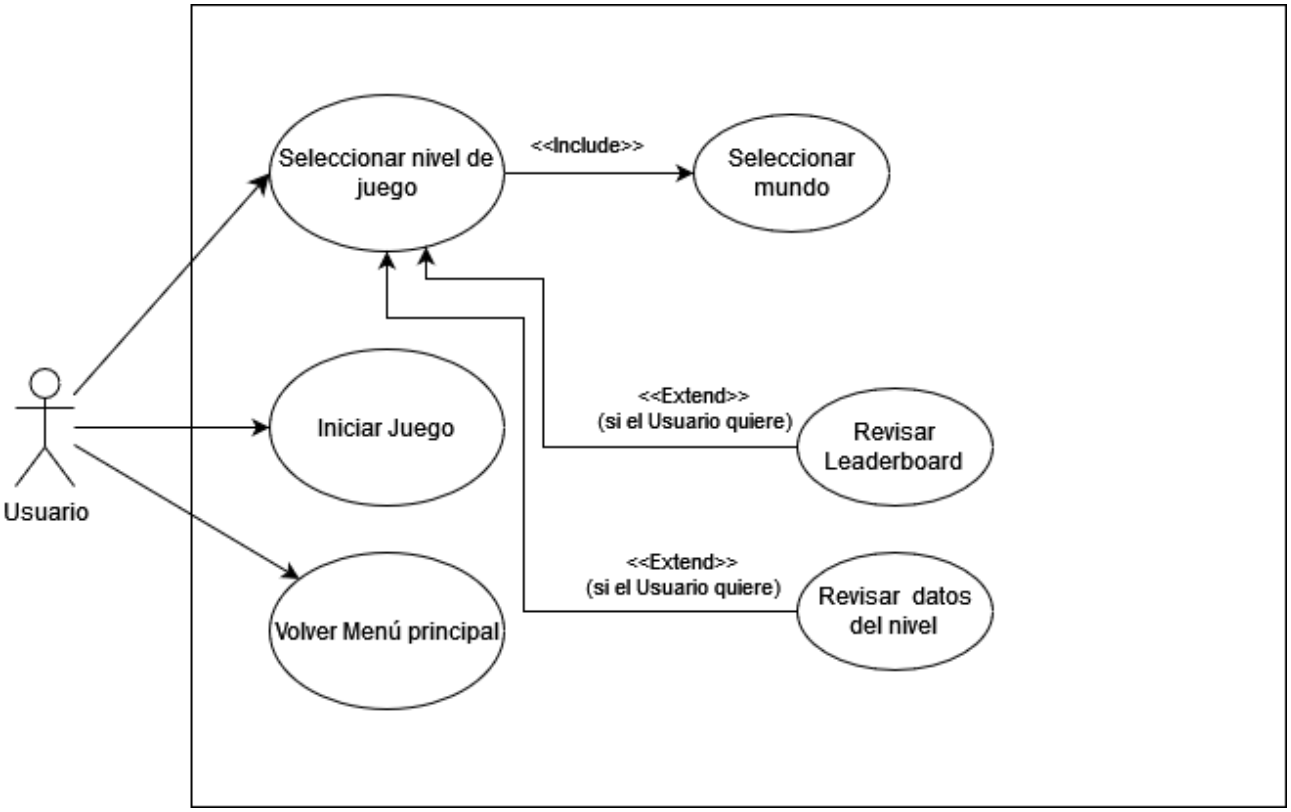


Ilustración 3.3 Diagrama de caso de uso del submenú Niveles

3.1.4. Menú opciones

En este menú se puede modificar los valores de sonido de tres elementos:

- General
    - Música
    - Efectos Especiales

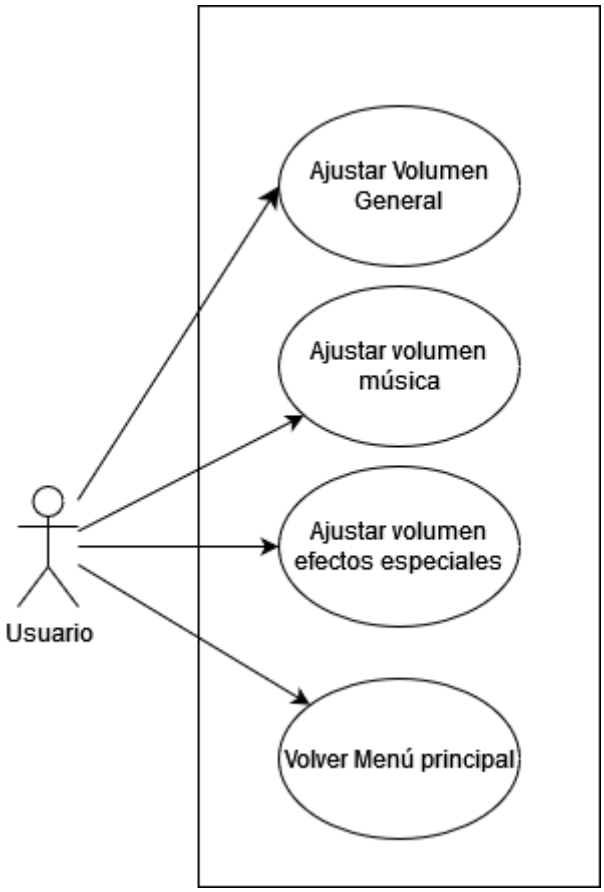


Ilustración 3.4 Diagrama de caso de uso del submenú Ajustes

3.1.5. Pantalla de juego

Esta escena es bastante compleja, aunque las acciones que puede tomar el usuario son reducidas:

- Pulsar una tecla del teclado
- Visualizar la vida
- Visualizar la puntuación y los combos
- Pausar el juego

Aunque dos opciones, Pausar el juego y Pulsar una tecla del teclado tienen más complejidad. El primero es una alteración del menú de opciones y el segundo es complejo por los diferentes estados en los cuales se puede encontrar el programa

- Sin letras en la zona de golpeo
- Con letras en la zona de golpeo y se pulsa la tecla correcta
- Con letras en la zona de golpeo y se pulsa una tecla incorrecta

De forma que los únicos estados que merecen

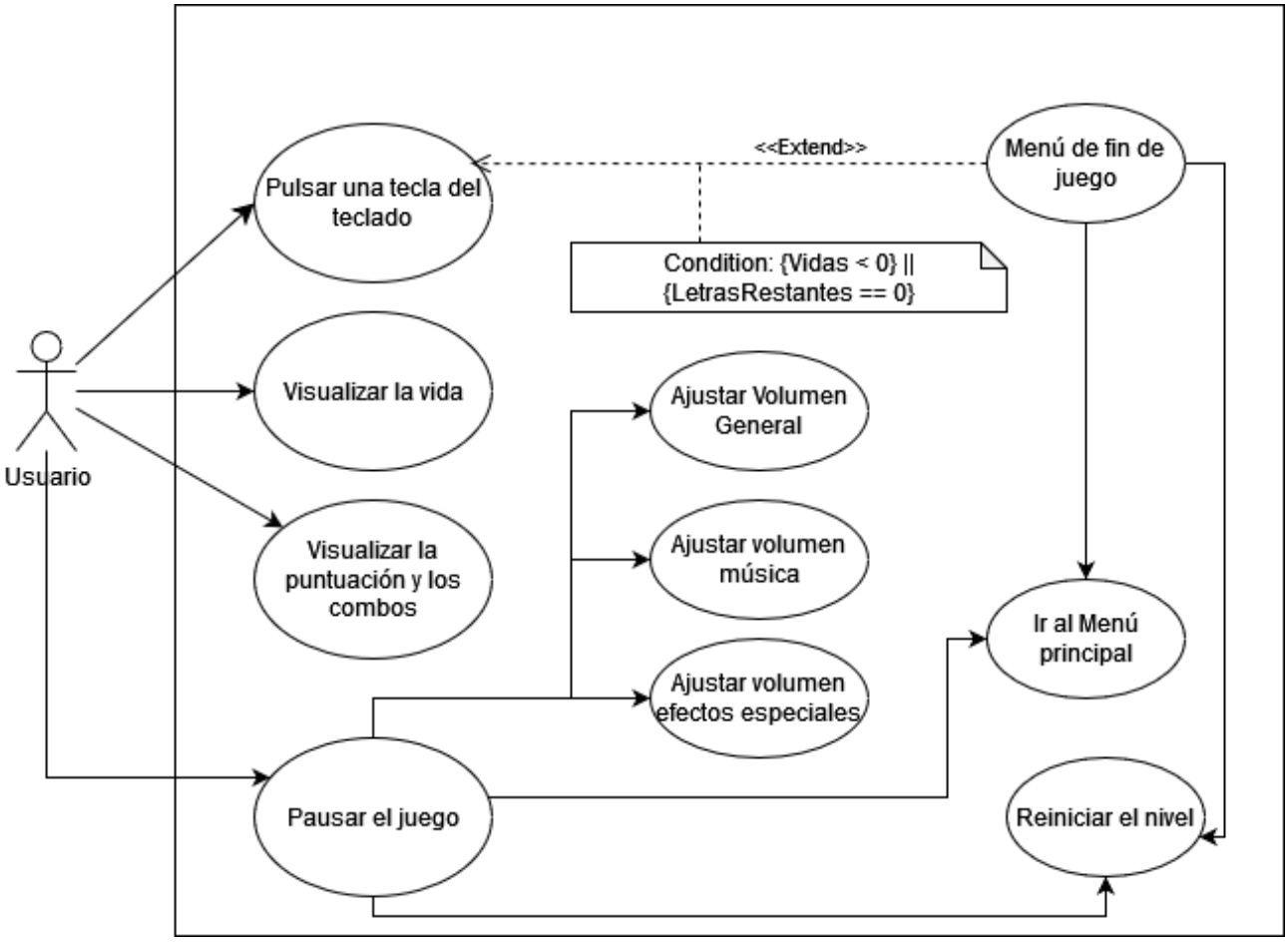


Ilustración 3.5 Diagrama de caso de uso del loop jugable

3.2. Storyboard

A continuación, se visualizan los storyboards de los casos de uso y de la aplicación.

3.2.1. Menú principal

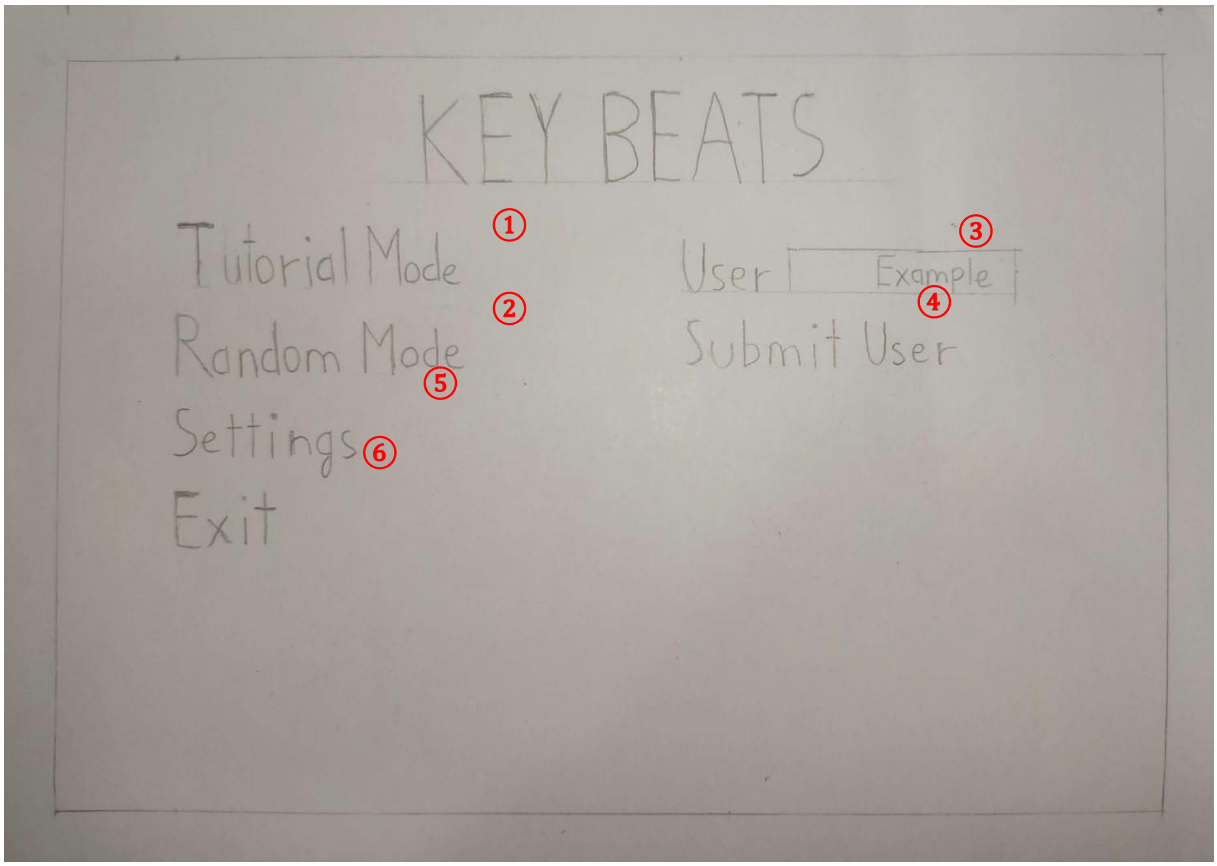


Ilustración 3.6 Storyboard menú principal

Aquí se muestra el menú que aparece nada más ejecutar el juego. Podemos distinguir que las opciones principales de juego aparecen listadas primero mientras que al final están las opciones menos relevantes.

A la hora de seleccionar los juegos hay dos opciones principales, Modo Tutorial (1) y Modo Random (2). A la derecha de estos se encuentra un campo de texto en el cual se introduce el nombre del jugador (3) con su botón de guardar cambios (4).

Finalmente, en la parte inferior izquierda encontramos las dos últimas opciones, la pantalla de Ajustes (5) y el botón de Salir (6). Al clicar en cualquiera de los tres menús anteriores se abre un submenú que veremos a continuación.

3.2.2. Menú Niveles

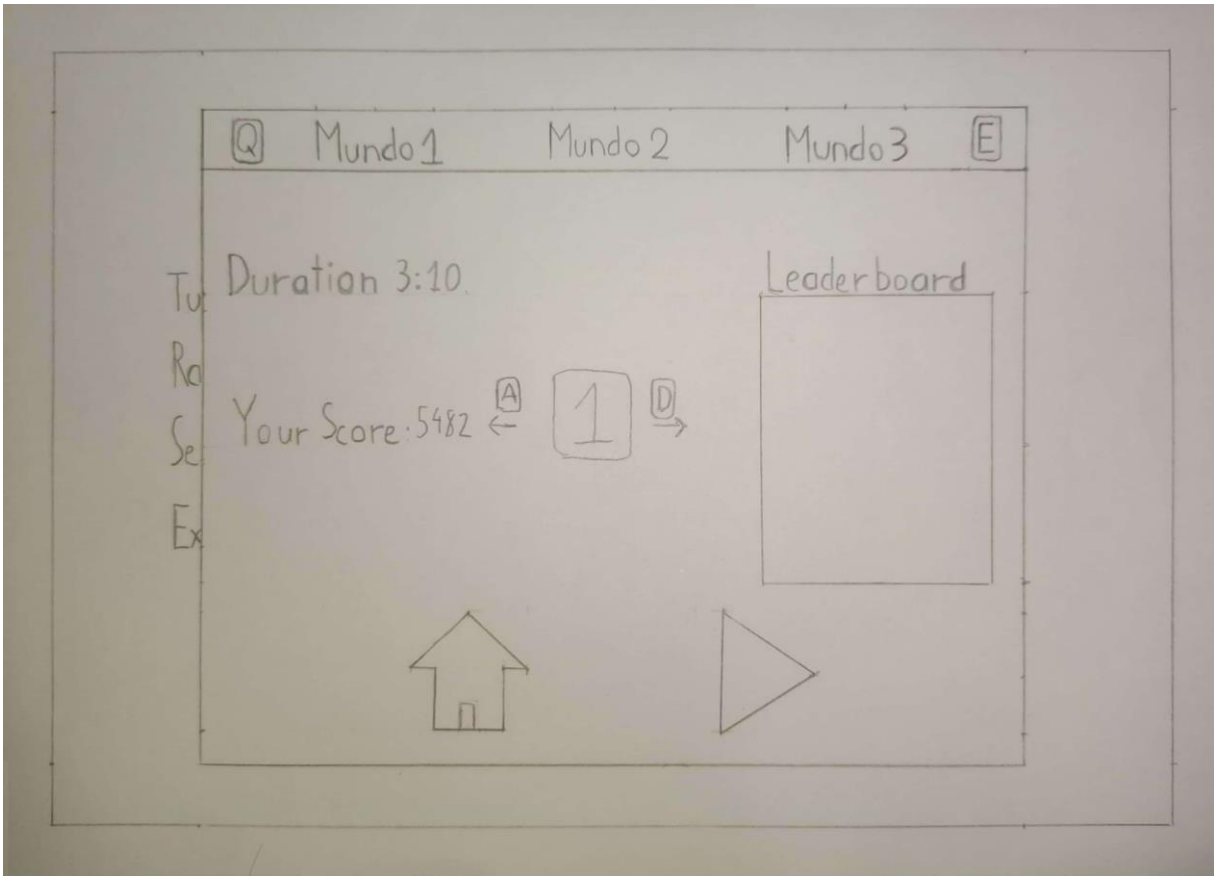


Ilustración 3.7 Storyboard submenú selector de niveles

Este submenú contiene 2 apartados principales la parte superior (1) indica un selector de mundo que podrá ser cambiado mediante las teclas Q y E para avanzar y retroceder respectivamente.

Cuando se haya seleccionado el mundo se pasará a la parte central, el selector de nivel (2). Aquí a través de las letras A y D se podrá seleccionar un nivel como se ha descrito anteriormente.

Cuando se haya seleccionado el nivel veremos que en la parte central izquierda (3) se mostrará información relativa al nivel tal como la duración o la puntuación obtenida anteriormente. También se indicará en la parte central derecha (4) una tabla de puntuación con los mejores resultados anteriormente.

Para concluir, tenemos la parte inferior que tiene un botón de volver al menú principal (5) o el botón más importante de este submenú, que es el botón de iniciar el juego (6)

3.2.3. Menú Random

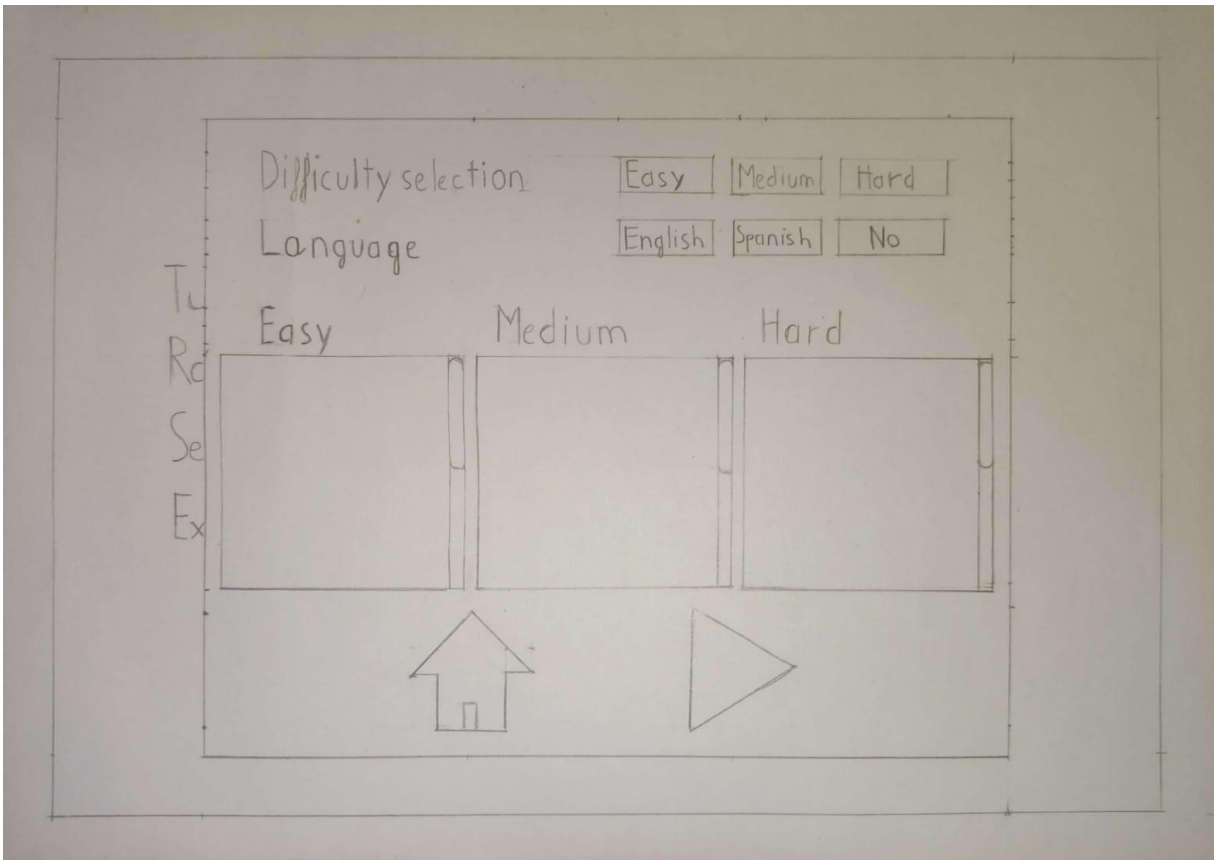


Ilustración 3.8 Storyboard submenú modo random con o sin diccionario

Este submenú se compone de dos partes: la parte superior (1) que posee un par de selectores indicando la dificultad del juego y si se quiere seleccionar un idioma a la hora de generar las “letras” y la parte central (2) que muestra diferentes tablas de puntuación con los mejores resultados agrupados por nivel de dificultad.

Para concluir, tenemos la parte inferior que tiene un botón de volver al menú principal (3) o el botón más importante de este submenú, que es el botón de iniciar el juego (4)

3.2.4. Menú Ajustes

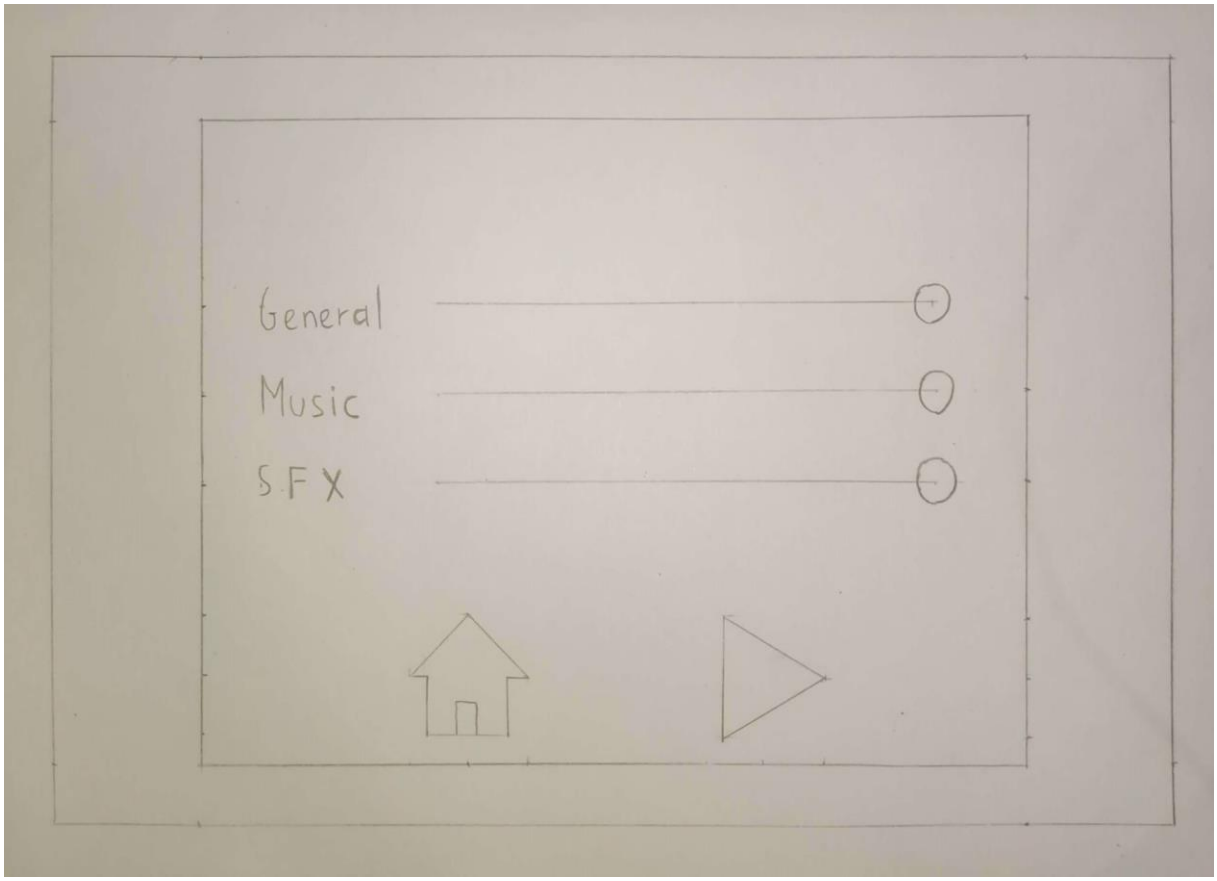


Ilustración 3.9 Storyboard submenú ajustes

Este submenú se compone de dos partes: la parte superior (1) que posee tres sliders que permiten modificar el volumen del juego a través del ratón y la parte inferior que tiene un botón de volver al menú principal (3) o de resumir el juego (4)

3.2.5. Loop Jugable

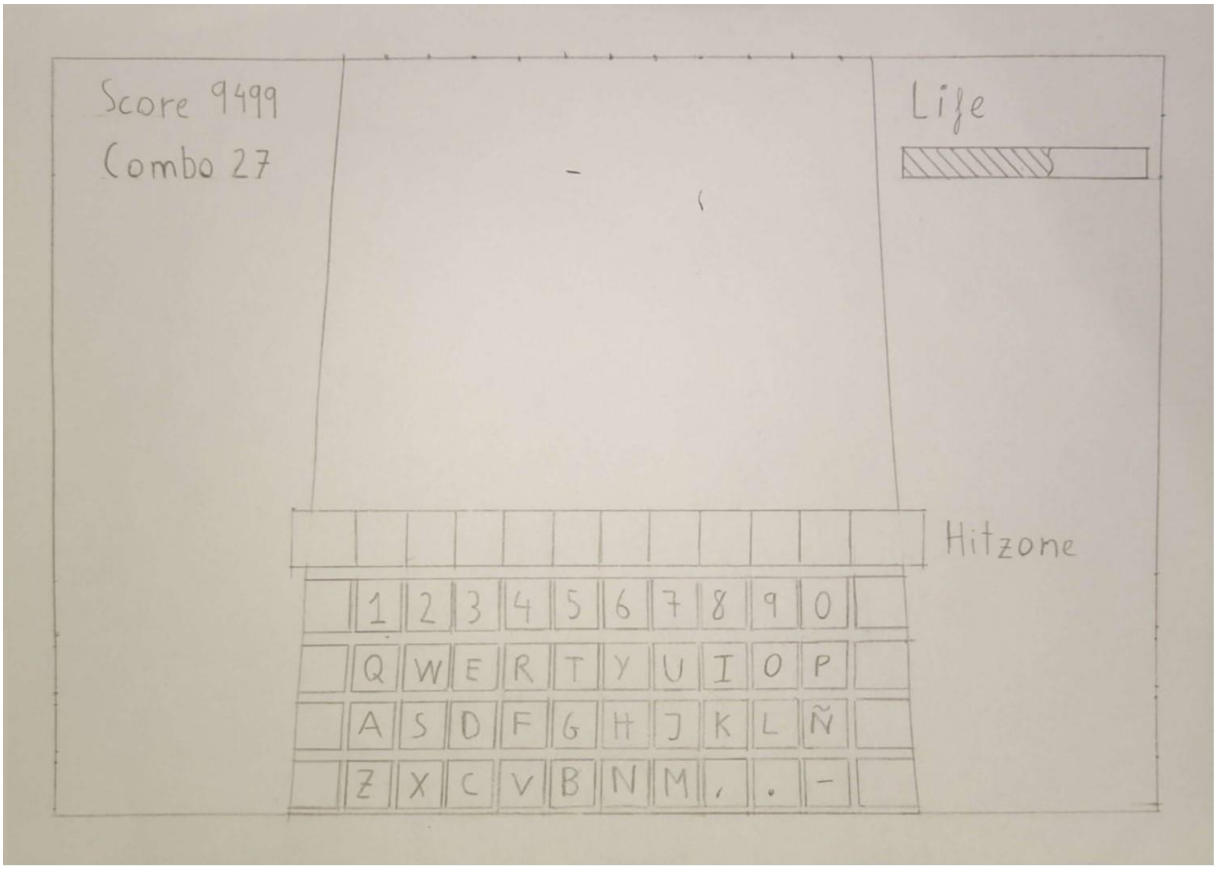


Ilustración 3.10 Storyboard de la escena jugable

Esta es la interfaz que se muestra cuando se selecciona un modo de juego. Podemos distinguir 2 partes principales, los laterales izquierdo y derecho y la parte central.

Empezando por la parte central podemos distinguir las diferentes letras que van cayendo (1) hacia un rectángulo grande conocido como “Hit Zone” o zona de golpeo (2). Por último, abajo vemos un teclado que sirve como referencia visual del teclado físico como ayuda al usuario (3).

Si nos centramos en los laterales podemos ver en la parte superior izquierda dos valores, el superior (4) indica la puntuación actual mientras que el inferior (5) indica la cantidad de aciertos consecutivos que llevamos. En la parte superior derecha tenemos una barra de vida (6) que trasmite de forma intuitiva el porcentaje de vida que se posee. Finalmente, en la parte centrar derecha tenemos un letrero que indica al usuario la zona de golpeo (7).

3.2.6. Menú de pausa

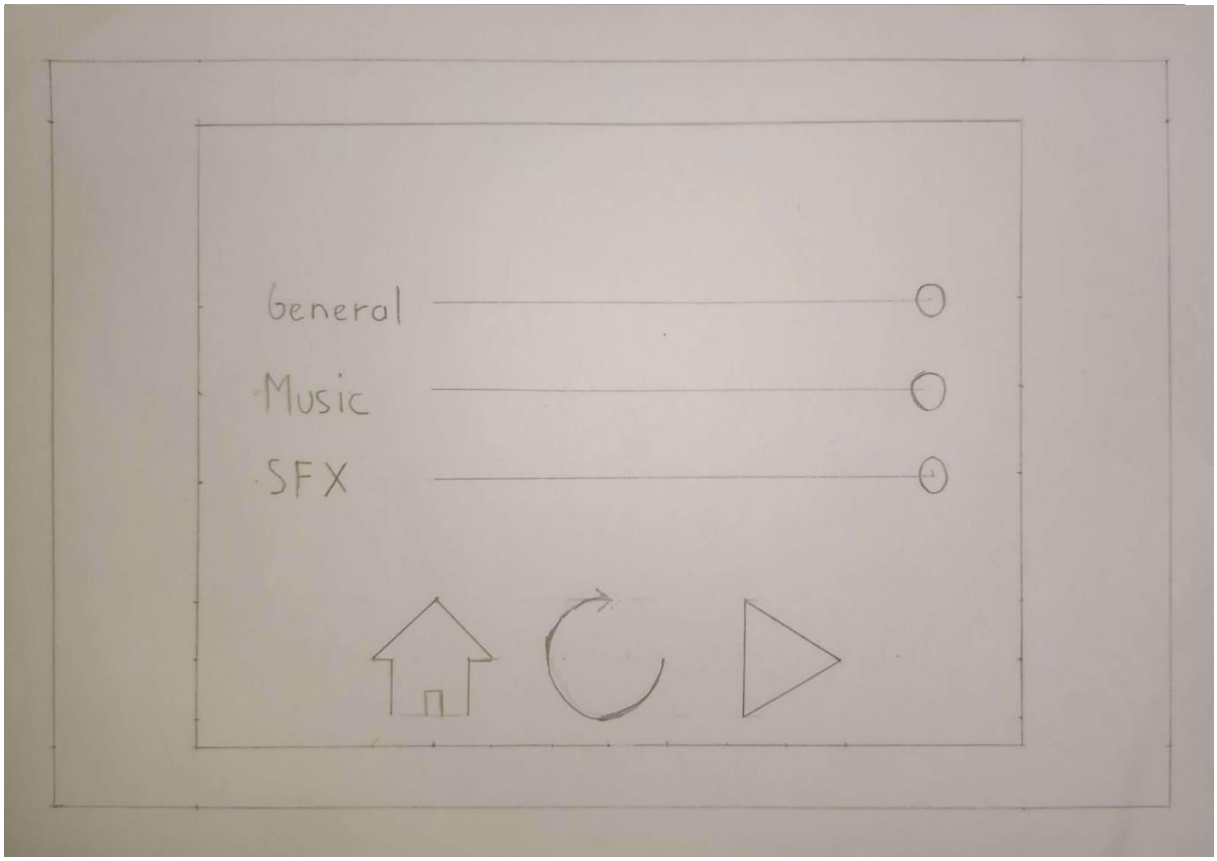


Ilustración 3.11 Storyboard menú pausa

Este submenú es bastante parecido al menú de ajustes y se compone de dos partes: la parte superior (1) que posee tres sliders que permiten modificar el volumen del juego a través del ratón y la parte inferior que tiene un botón de volver al menú principal (3) o de resumir el juego (4) o de reiniciar el nivel (5)

3.2.7. Fin de Juego

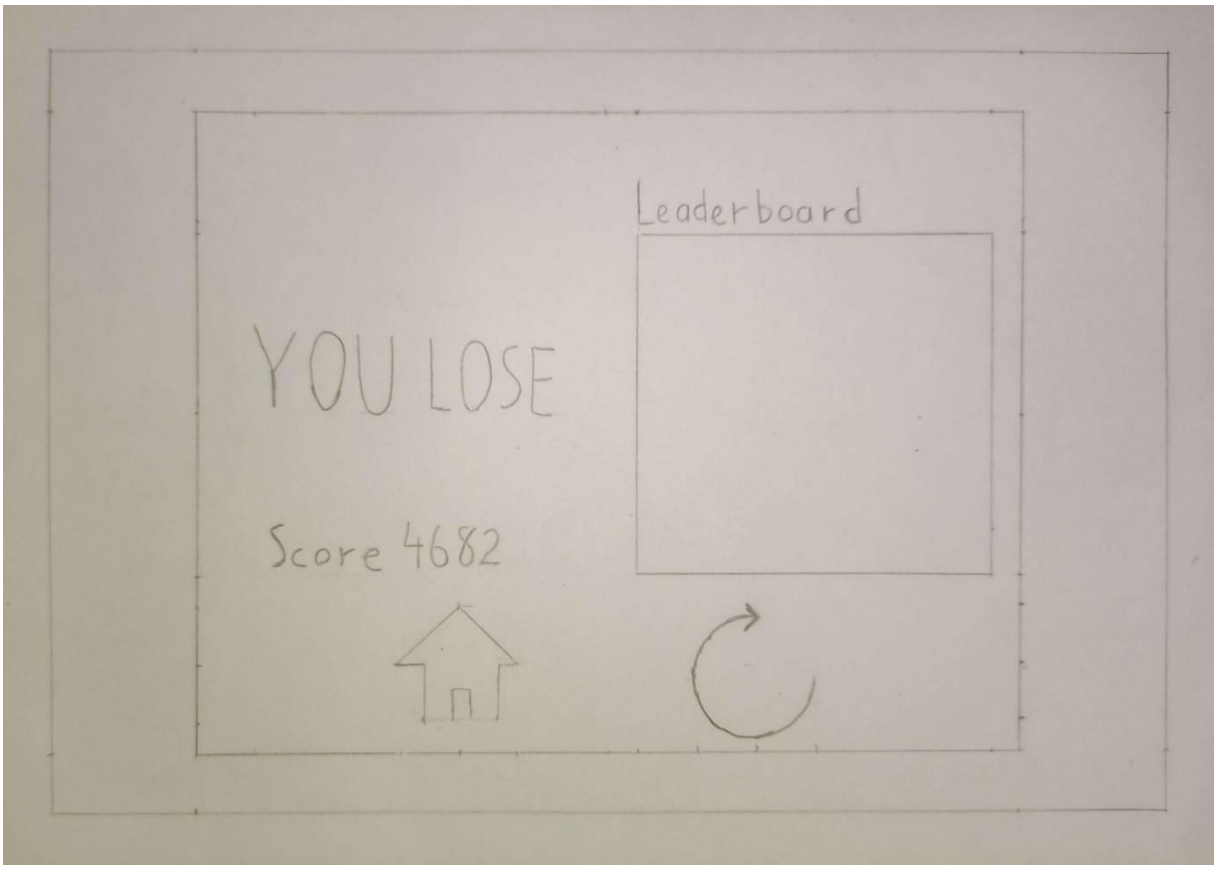


Ilustración 3.12 Storyboard menú fin de partida con condición de derrota

Este menú sólo aparece cuando se cumple la condición de fin de juego. En la parte izquierda hay un mensaje indicando si ha sido de forma positiva o negativa (1) y la puntuación total (2). A la derecha tenemos el leaderboard (3), ya sea del nivel o del nivel de dificultad del modo random.

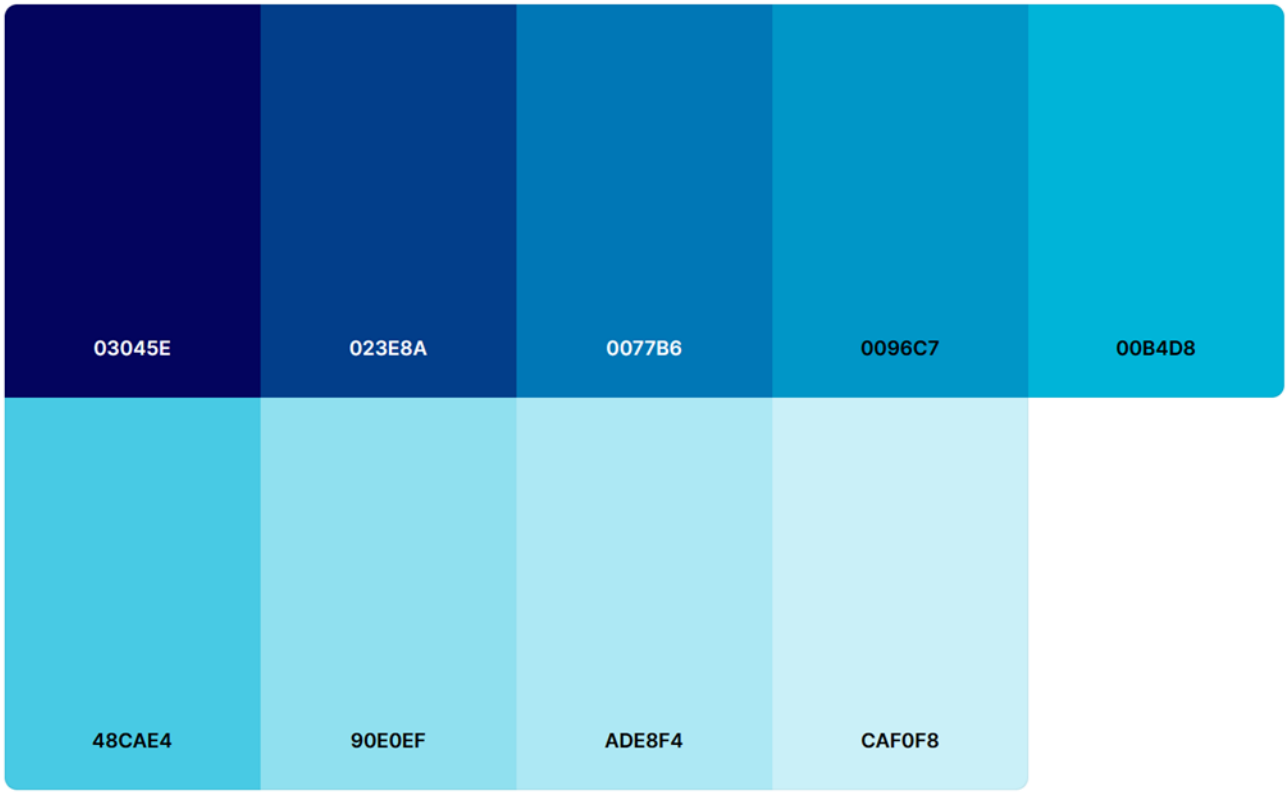
Finalmente, abajo tenemos un botón a la izquierda para volver al menú o un botón a la derecha para reiniciar el nivel.

3.3. Código de colores

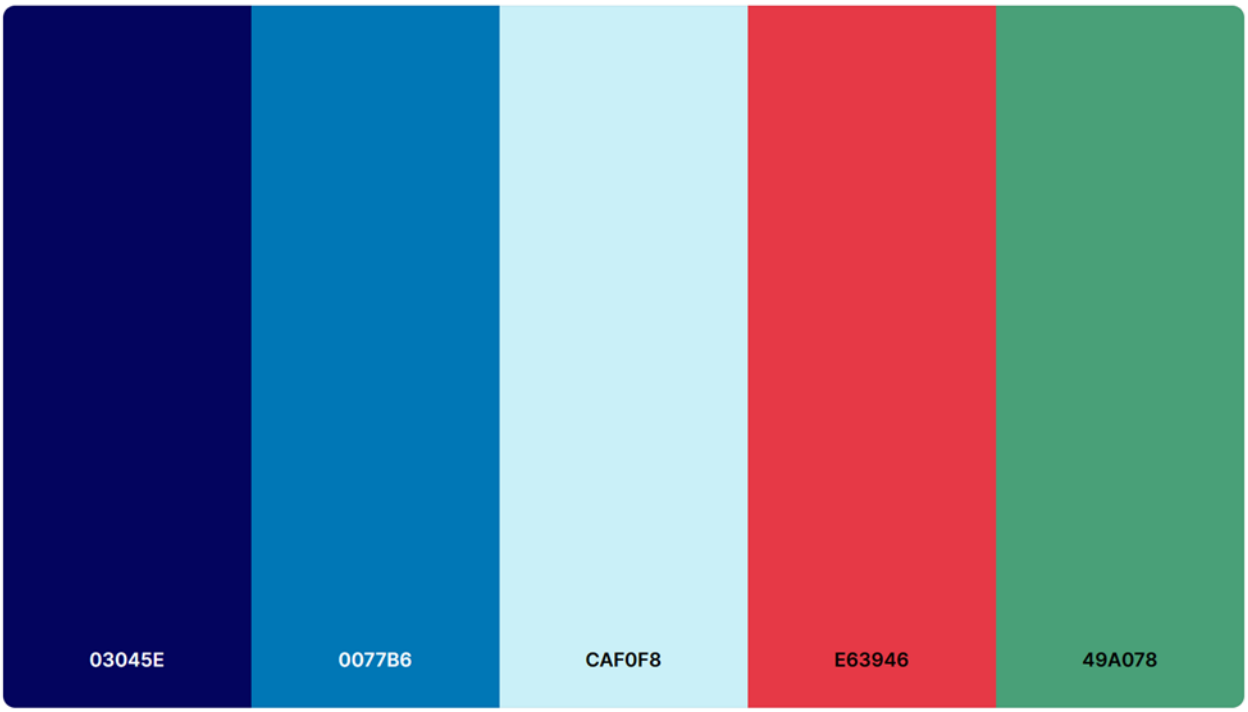
A la hora de diseñar una página web, los colores juegan un papel vital en la experiencia de usuario. Partiendo que unos colores adecuados sirven para hacer una navegación más intuitiva, utilizable y hasta que sirven para que personas con problemas de accesibilidad como el daltonismo puedan usar la aplicación web. Por tanto, a la hora de diseñar este prototipo se tuvo en cuenta este elemento de cara a la accesibilidad.

Para lograr este objetivo se ha asegurado de que todos los elementos existentes en la interfaz sigan una paleta de colores. Por suerte, existen páginas que o bien poseen un catálogo con diferentes paletas de colores o bien tienen generadores que crean una paleta en segundos, como es el caso de Coolors[T]. Aparte de tener ambas opciones previamente mencionadas, posee un analizador de colores y un inspector de contraste que son herramientas que ayudan a la hora de elegir colores y comprobar como interactúan unos colores con otros o cómo los distintos tipos de daltonismo puede percibir el color o inclusive la paleta.

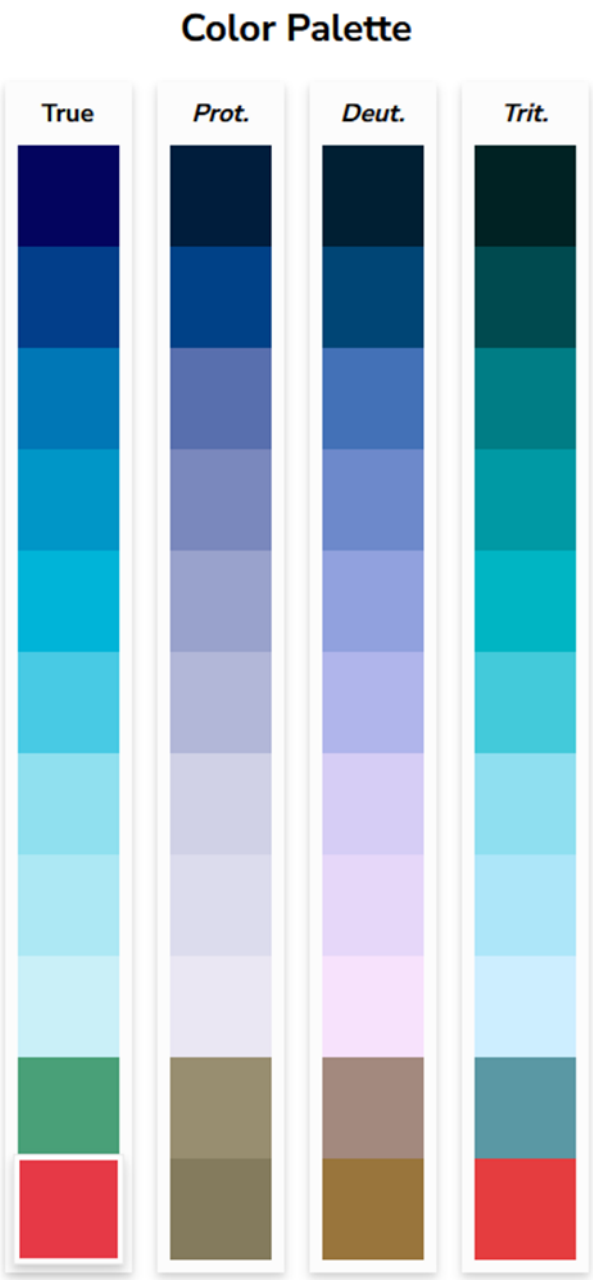
Para mantener la cantidad de colores bastante limitada se ha usado una paleta de colores mayoritariamente monocromática[U], degradando la intensidad de estos.



Adicionalmente se han usado un par de colores para generar contraste entre esta base, permitiendo identificar elementos importantes de forma intuitiva, como pueden ser enemigos. [V]

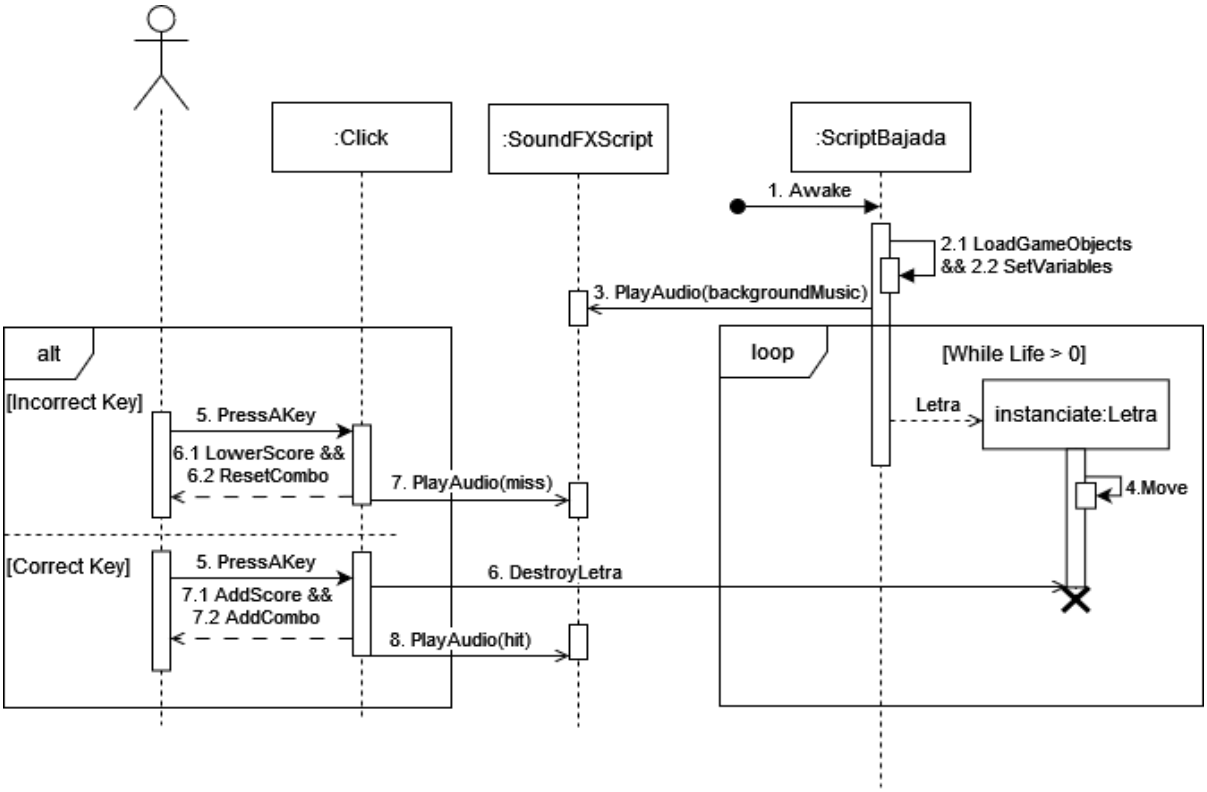


Finalmente usando otra herramienta web como es Coloring for Colorblindness[W] podemos comprobar cómo se ven realmente la paleta completa con los distintos tipos de daltonismo.


Color Palette
True
Prot.
Deut.
Trit.

3.4. Diagrama de Secuencia

A la hora de explicar cómo interactúan varios sistemas juntos en un videojuego en los cuales hay varios objetos con uno o varios componentes cada uno, se ha visto necesario realizar diagramas de secuencia al menos sobre el caso de uso Loop Jugable.



Este primer diagrama está basado en el modo de juego Random y se centra en como interactúan 3 sistemas entre sí, los cuales son: Click, ScriptBajada y SoundFXScript. El jugador solo interactúa con Click que sirve cono interfaz entre el Usuario y el sistema de juego.

El flujo normal del sistema empezaría por la carga de la escena. Antes de empezar el primer frame, la clase ScriptBajada obtiene los parámetros de invocación y carga los elementos y las estructuras de datos que va a utilizar ya sea un nivel predefinido o los porcentajes de las letras.

Tras esto empezaría el primer frame del juego. Lo primero que se realiza es reproducir la música de fondo, continuando con un bucle que se encarga de crear

los objetos Letra. Estos objetos sólo tienen una función que es bajar la pantalla hasta que son destruidos.

El rol del jugador en esta escena es pulsar la tecla correcta del teclado en el momento preciso que es cuando la Letra llegue a la “Hit Zone” como vimos en el apartado 3.2.5. Para ello cuando se pulsa una tecla, Click se encarga de comprobar si:

- Hay alguna Letra sobre la Hit Zone
- En caso de que fuera así, si la tecla coincide con el valor de la Letra

A su vez esto puede resultar en dos opciones:

- Incorrecto: El jugador se ha equivocado a la hora de pulsar la tecla, ya sea no midiendo bien el momento o porque ha pulsado una tecla que difiere de la Letra. En estos casos se penaliza al jugador restándole puntuación y reiniciando el valor del combo a 0. Finalmente, Click llama a SoundFXScript y se reproduce un sonido de fallo.
- Correcto: El jugador ha acertado a la hora de pulsar la tecla. Aquí primero Click manda destruir la Letra, luego aumenta la puntuación y el combo y finalmente llama a SoundFXScript y se reproduce un sonido de éxito.

4. DESARROLLO

En este capítulo se expondrán todo el trabajo realizado paso a paso. Como bien se describió en el apartado 2 se está utilizando un modelo incremental, por lo que se facilitará la legibilidad dividendo este capítulo en tantos apartados como iteraciones existen.

4.1. Iteración 0

Esta iteración se compone de toda la funcionalidad que realizará el Analista y compone todo el trabajo realizado en el apartado 1 a 3 de este documento. Desde el estudio de las tecnologías como informándose de los requisitos necesarios o realizando la instalación de estas, así como realizar pruebas de rendimiento en los equipos.

A parte, se ha realizado un manual de instalación y un manual de usuario que se encuentran en los anexos. En caso de querer realizar pruebas sobre todo el código o se quiere realizar un seguimiento del trabajo realizado se recomienda instalar el software necesario siguiendo el Anexo A: manual de instalación.

4.2. Iteración 1

Esta iteración se compone de la tarea C. Esta tarea consiste en crear una escena de juego en la cual se añaden los elementos estáticos de los mismos (fondo, zona de juego), crear las Letras que son los enemigos y la funcionalidad de como bajan y un sistema de eliminación de estas.

4.2.1. Crear escena y añadir elementos estáticos

Para comenzar se procede a crear una escena llamada *GameZone* donde se van a desarrollar toda la lógica del juego. Posteriormente se utilizan dos sprites 2D para delimitar los bordes de la zona de juego que tienen este borde ha sido inclinado 5º en el eje Z para dar sensación de profundidad simulando las vías de un tren (Ilustración 4.1). A su vez hay un fondo en la zona de juego de un color más oscuro con forma trapezoidal.

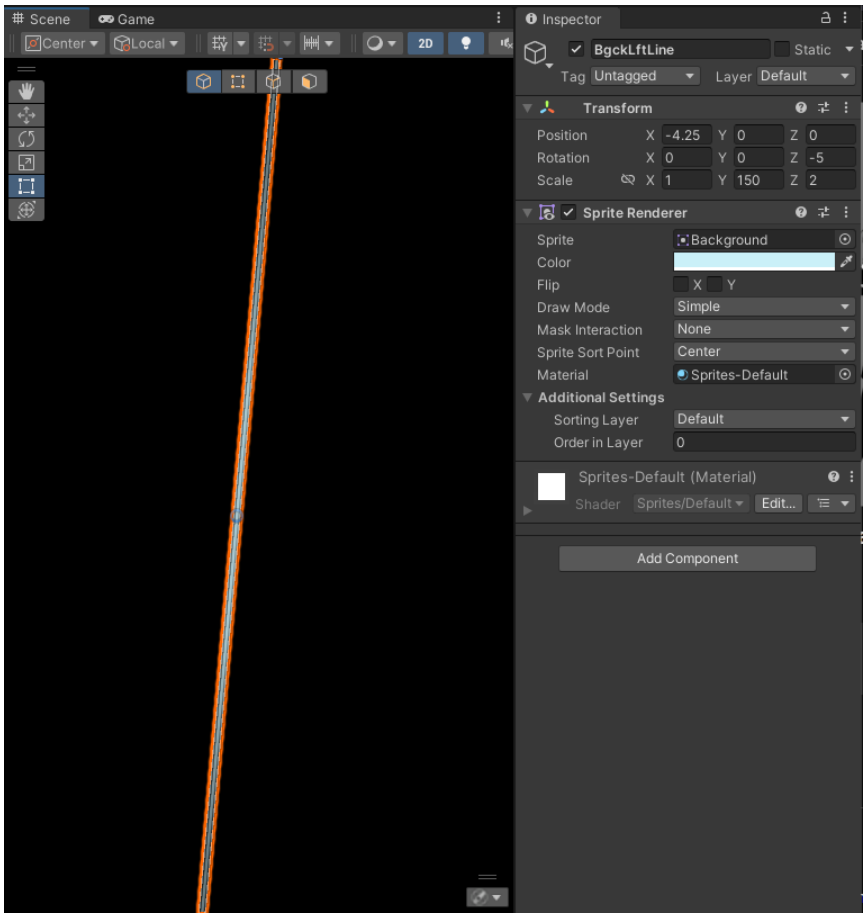


Ilustración 4.1 Ajuste y configuración del borde

Para añadir el teclado, se ha adquirido una imagen de un teclado de un sitio web y se ha modificado con GIMP, primero eliminando la marca de agua y corrigiendo ruido de la imagen y finalmente estirando la misma a una forma trapezoidal.

Igualmente, se ha usado el editor de imágenes de GIMP para crear la HitZone, para ello se ha creado un rectángulo dividido en cuadrados más pequeños y se han divido las letras por columnas de la siguiente forma:

Columna de izquierda a derecha	Letras asociadas
Columna 1	Q, A, Z
Columna 2	W, S, X
Columna 3	E, D, C
Columna 4	R, F, V
Columna 5	T, G, B
Columna 6	Y, H, N
Columna 7	U, J, M
Columna 8	I, K
Columna 9	O, L
Columna 10	P

Tabla 4.2 División de las letras en la HitZone

Finalmente se ha inclinado la cámara 50º en el eje X. Esto hace que la disposición de la escena quede de esta manera:

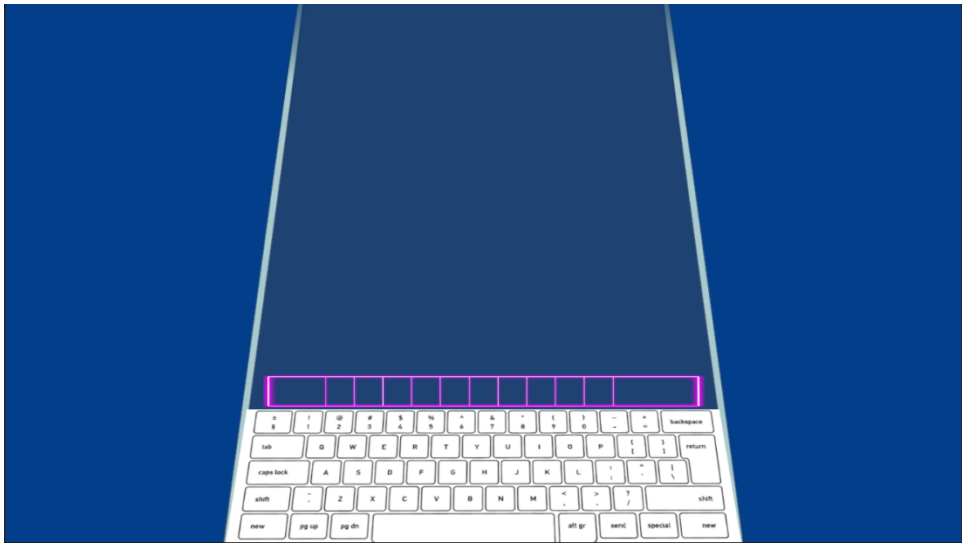


Ilustración 4.3 Disposición final de la escena

4.2.2. Añadir todas las “letras”

Para la realización de esta tarea se procedió a usar el editor de imágenes de GIMP y editar las letras del teclado previamente creado individualmente.

Ilustración 4.4 Sprite de la letra A

Luego se importaron los sprites de todas las letras que van de la A-Z y se creó un Prefab a cada una de ellas, indicando la posición X para que la letra coincida con el hueco de la HitZone.

Para el movimiento se ha dispuesto de un script singleton asociado a cada uno de los Prefabs, este singleton tiene un valor único que guarda el valor de velocidad de los objetos asociados.

```
0 references
public class MovementScript : MonoBehaviour
{
    1 reference
    public static float speed = 0.1f;
    0 references
    void Start()
    { }

    // Update is called once per frame
    0 references
    void Update()
    {
        transform.Translate(Vector2.down * speed * Time.deltaTime);
    }
}
```

Ilustración 4.5 Código de Movimiento

4.2.3. Crear script de funcionalidad de bajada y eliminación

En primera instancia se planteó la generación de letras que tendría el modo de juego Random sin ningún tipo de diccionario.

Para ello se definieron 2 aspectos fundamentales:

- Primero se cargarían todos los objetos y se elaborarían todas las estructuras de datos antes del primer frame de ejecución, en primera instancia se utilizó Start() que se ejecuta antes del primer frame. Posteriormente se cambió a Awake() que se ejecuta antes de que cargue la escena.
- Luego se utilizaría desarrollo basado en componentes para el funcionamiento de la lógica. Por tanto, este script se encarga de la creación de los objetos, pero luego cada uno de ellos se encarga de su funcionamiento de forma concurrente.

Por tanto, para esta primera iteración se planteó un script llamado *LetterManager* que gestiona todas las letras realizando primero una función de carga de todos los objetos y para ello se usan dos estructuras de datos:

- **letterList**: que es un *ArrayList<GameObject>* que es donde se guardan todas las letras empleadas en el modo Random, ya que aquí solo nos interesa poder sacarlas con un número aleatorio
- **dicctionaryList**: que es un *Dictionary<string, GameObject>* usando como par de claves un string con la letra y el GameObject asociado.

```
1 reference
void LoadGameObjects()
{
    //Load the gameObjects to random mode
    letterList = new ArrayList();
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/A") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/B") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/C") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/D") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/E") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/F") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/G") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/H") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/I") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/J") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/K") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/L") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/M") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/N") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/O") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/P") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/Q") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/R") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/S") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/T") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/U") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/V") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/W") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/X") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/Y") as GameObject);
    letterList.Add((GameObject)Resources.Load("GameObjects/Letras/Z") as GameObject);
    //Load in dicctionary for the levels
    dicctionaryList = new Dictionary<string, GameObject>();
    foreach (GameObject letter in letterList)
    {
        dicctionaryList.Add(letter.name, letter);
    }
}
```

Ilustración 4.6 Función de carga de objetos

A la hora de generar los objetos, es un bucle infinito que genera letras cada cierto intervalo. La posición Y es generada gracias a un GameObject conocido como *topScreen* que se sitúa en la parte superior de la pantalla.

```
1 reference
IEnumerator CreateObjectRandomV3()
{
    while (true)
    {
        int random;
        GameObject actualLetter;
        random = UnityEngine.Random.Range(0, letterList.Count);
        actualLetter = (GameObject)letterList[random];
        movingObject = Instantiate(actualLetter, new Vector3(actualLetter.transform.position.x, topScreen.transform.position.y, -0.2f), Quaternion.identity);
        yield return new WaitForSeconds(spawningSpeed);
    }
}
```

Ilustración 4.8 Método de generación de letras

Toda la funcionalidad relacionada con el teclado está llevada a cabo por otro script conocido como *ClickScript* asociado al collider de la HitZone.

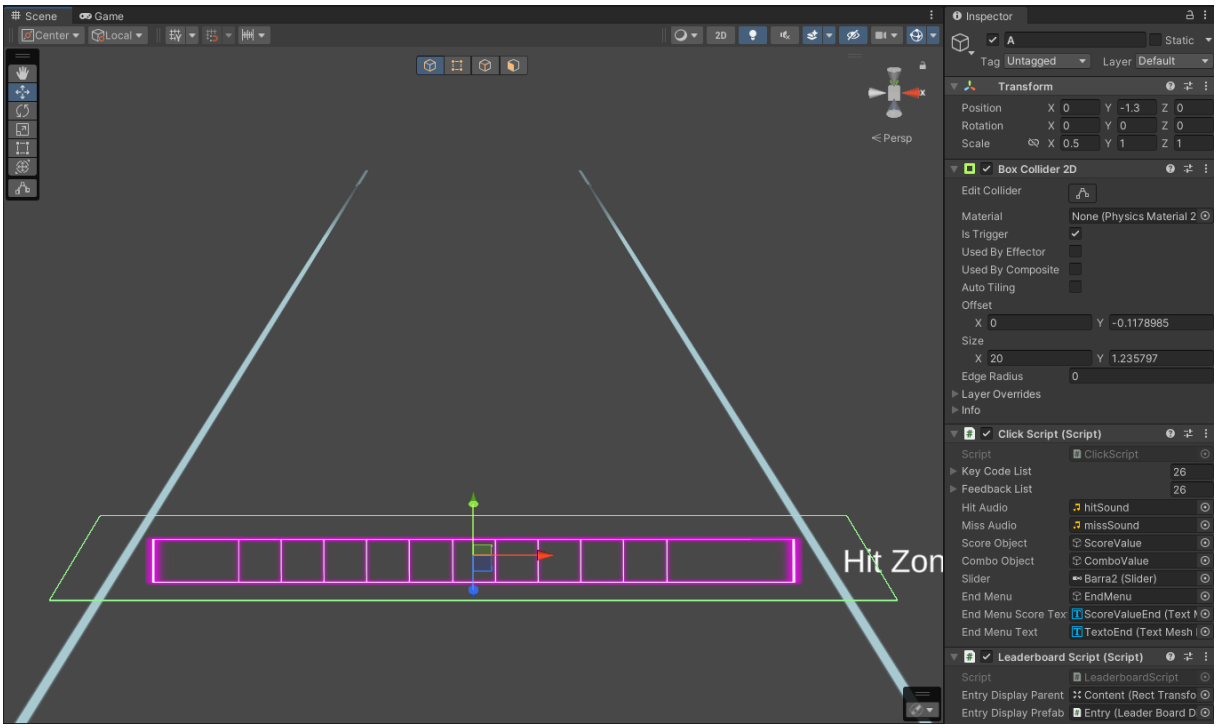


Ilustración 4.7 Parámetros del Collider de la HitZone, así como del script ClickScript

Cuando una letra esté en dentro del script se habilitará la eliminación de la letra correspondiente.

```
0 references
void OnTriggerEnter2D(Collider2D other)
{
    collisedObjectColliderList.Add(other);
}
```

Ilustración 4.9 Funcionalidad de la colisión de colliders

Cuando se pulse una tecla habrá la siguiente comprobación. Se recorrerá la lista de objetos colisionados, en caso de que la letra coincida se destruirá la letra y se eliminará de la lista de objetos colisionados

```
if (collisedObjectColliderList[y].tag == keyCodeList[i].ToString())
{
    //Se mete en la función pero no modifica la variable del score
    scored = true;
    Collider2D destroyCollider = collisedObjectColliderList[y];
    collisedObjectColliderList.Remove(collisedObjectColliderList[y]);
    Destroy(destroyCollider.gameObject);
    //ReproducirSonido
    SoundFXScript.instance.PlayAudio(hitAudio, 1f, MusicTypes.sfx,0);
    scored2=true;
    break;
}
```

Ilustración 4.10 Funcionalidad de comprobación de las letras con la pulsación de teclado

4.3. Iteración 2

Esta segunda iteración las tareas a realizar son la D, E, F y G. Estas tareas corresponden con la implementación principal de los niveles y de la generación por alfabetos, así como de gestionar los elementos no destruidos por el jugador.

4.3.1. Definir sistema de generación

Un nivel es una serie de letras que son determinadas previamente a la ejecución del videojuego. Por tanto, es necesario establecer las clases y métodos que van a generar las letras en un tiempo determinado, las estructuras de datos donde se van a guardar esas letras y gestionar la persistencia de los niveles de forma que sea sencillo para el diseñador de estos.

Se definió una clase llamada *LevelKey* que tiene tres atributos:

- **Key**: La letra a guardar
- **TimeToAppear**: El tiempo que tarda la letra en aparecer desde el inicio de nivel
- **TimeToWait**: El intervalo de tiempo entre la aparición de la letra actual y la siguiente en el nivel.

```
10 references
public class LevelKey
{
    3 references
    string key; //Letter
    4 references
    int timeToAppear; //In miliseconds
    4 references
    int timeToWait; //Time to wait for the next key

    2 references
    public LevelKey(string key, int timeToAppear, int timeToWait)
    {
        this.key = key;
        this.timeToAppear = timeToAppear;
        this.timeToWait = timeToWait;
    }
```

Ilustración 4.11 Clase LevelKey con el constructor por parámetros

Como el nivel se guarda de forma lineal se ha usado un ArrayList para gestionar los *LevelKey* en el script llamado **levelList**

De esta forma el método de generación de letras en *LetterManager* sería el siguiente:

```
1 reference
private IEnumerator PlayLevel()
{
    foreach (LevelKey levelKey in levelList)
    {
        if(levelKey.getKey()!="None"){
        GameObject letter = dicctionaryList[levelKey.getKey()];
        movingObject = Instantiate(letter, new Vector3(letter.transform.position.x, topScreen.transform.position.y, -0.2f), Quaternion.identity);}
        yield return new WaitForSeconds((float)(levelKey.getTimeToWait() / 1000.0));
    }
    //After that, when the movingObject is deleted that means that the level is finshed
    StartCoroutine(CheckIfObjectIsDestroyed());
}
```

Ilustración 4.12 Método de generación de letras en los niveles

Ahora pues para la facilidad de como guardar el nivel de forma persistente se decidió usar un archivo csv. De esta forma solo había que gestionar un cabecero donde se guardarían los parámetros del nivel, ya sea velocidad o indicar la ruta del archivo de música y luego tendríamos todas las letras indicadas.

La estructura sería la siguiente:

[Ruta del archivo de música];[Velocidad];[Tiempo total];[Delay primera letra]
[Letra];[Tiempo de aparición];[Tiempo de espera]

Por tanto, un ejemplo sería el siguiente:

```
Audio/Music/world 1-level 1;1,5;1:20;7,6
F;0;2400
F;2400;2400
J;4800;2400
F;7200;2400
```

De esta forma podemos en el Awake() leer e introducir las letras de esta forma:

```
1 reference
void ReadLevel()
{
    levelList = new ArrayList();
    bool firstLine = true;
    foreach (string line in File.ReadLines(levelPath, Encoding.UTF8))
    {
        if(firstLine){
        firstLine=false;
        string[] words = line.Split(';');
        backgroundMusic = (AudioClip)Resources.Load(words[0]);
        MovementScript.speed= float.Parse(words[1]);
        delayed = float.Parse(words[3]);
        }else{
        string[] words = line.Split(';');
        LevelKey key = new LevelKey(words[0], Int32.Parse(words[1]), Int32.Parse(words[2]));
        levelList.Add(key);
        }
    }
}
```

Ilustración 4.13 Método de lectura de un nivel

4.3.2. Añadir música y sonido

A la hora de implementar los sonidos se investigó en la documentación oficial cómo Unity puede gestionar los audios y los sistemas de sonido. Unity usa Audio Sources que es un componente que se puede añadir a los GameObjects, este a través de AudioClip permite reproducir sonidos desde posiciones específicas del juego.

Por tanto, como nuestro juego no requiere de sonidos posicionales se han añadido dos Audio Sources al objeto que maneja el script del juego que esta centrado en la escena. Luego para manejar el sistema de volumen se ha optado por instanciar 3 Audio Mixers. Un Audio Mixer es un objeto que permite gestionar varios Audio Sources y modificar sus parámetros de volumen o introducir filtros de audio aunque no se encuentren instanciados en la escena.

Ahora pues, si bien se podría añadir estos dos Audio Sources a todos los elementos que los usan, se ha optado por crear otro script público singleton. Este script tiene todos los Audio Sources y los Audio Mixers disponibles en el juego. De esta forma cuando otro objeto quiere reproducir un sonido, solamente tiene que llamar a la función de PlayAudio() indicando el AudioClip a reproducir, si quiere que haya un retraso en la reproducción, su volumen y ante todo el tipo de audio. Este tipo de audio está delimitado por un enum conocido como MusicTypes.

```
0 references
public void PlayAudio(AudioClip audioClip, float volume, MusicTypes type, float delayed){
    if (type == MusicTypes.music){
        sources[0].volume = volume;
        sources[0].outputAudioMixerGroup = mixer[0];
        sources[0].clip= audioClip;
        sources[0].PlayDelayed(delayed);
    }
    else{
        sources[1].volume = volume;
        sources[1].outputAudioMixerGroup = mixer[1];
        sources[1].PlayOneShot(audioClip);
    }
}
```

Ilustración 4.15 Método de reproducción del audio

```
2 references | 0 references | 1 reference | 0 references
public enum MusicTypes {general, music, sfx};
```

Ilustración 4.14 Tipos de audio disponibles

Aunque bien se indica un tipo de audio general, en la práctica no puede ser llamado nunca, aunque se planteó su uso para indicar cualquier tipo de notificación del juego o si hubiera algún error.

4.3.3. Limpieza de elementos

Aquí se modificó la funcionalidad de *ClickScript.*

En primera instancia se planteó usar OnTriggerExit2D(), pero Unity tiene una particularidad que cuando se destruye un objeto detecta que ha salido del collider ejecutando OnTriggerExit2D() dando un error de null pointer.

Para solucionar esto se implementó un flag conocido como *scored* y en caso de que el objeto ha sido puntuado no se destruye el objeto.

```
collisedObjectColliderList.Remove(other);
if (scored == true)
{
    score = (int)(score + 100 + 100 * combo * 0.01);
    scoreText.text = ("" + score);
    combo++;
    comboText.text = ("" + combo);
    if (combo % 5 == 0)
        slider.value++;
    scored = false;
}
else
{
    combo = 0;
    comboText.text = ("" + combo);
    score = score - 20;
    scoreText.text = ("" + score);
    Destroy(other.gameObject);
    slider.value--;
}
```

Ilustración 4.16 Funcionalidad OnTriggerExit2D

4.3.4. Alfabetos

Hay varias formas de delimitar el porcentaje de uso de las letras según el idioma. Muchos investigadores utilizan los diccionarios para delimitar el porcentaje, pero esto no refleja el uso real del idioma en el día a día. Otros, sin embargo, prefieren utilizar textos literarios como por ejemplo Don Quijote para el cálculo de este.

El consenso no está claro y a la falta de este, este apartado carece de rigor científico. Por este motivo, se ha planteado usar simplemente Wikipedia que reúne varios porcentajes de uso de varios estudios de varios idiomas (Letter Frecuency, 2024).

De esta forma el porcentaje de uso sería el siguiente:

Letra	% de uso Español	% de uso Inglés
A	12,0%	8,17%
B	1,4%	1,49%
C	4,5%	2,78%
D	5,6%	4,25%
E	13,1%	12,70%
F	0,7%	2,23%
G	1,0%	2,02%
H	0,7%	6,09%
I	6,0%	6,97%
J	4,2%	0,25%
K	0,0%	1,77%
L	4,8%	4,03%
M	3,0%	2,41%
N	6,5%	6,75%
O	8,3%	7,51%
P	2,4%	1,93%
Q	0,8%	0,10%
R	6,6%	5,99%
S	7,7%	6,33%
T	4,4%	9,06%
U	3,8%	2,76%
V	0,9%	0,98%
W	0,0%	2,36%
X	0,2%	0,25%
Y	0,9%	1,97%
Z	0,5%	0,07%

Tabla 4.17 Porcentaje de letras según el idioma

Luego a la hora de generar las letras se vuelve a necesitar de una clase para gestionar tanto el porcentaje de las letras, como el porcentaje acumulado para la generación aleatoria, todo esto llevado a cabo por la clase *KeyRandom.*

```
2 references
public float percentaje { get; set; }
1 reference
public float cumulative { get; set; }
2 references
public string letra { get; set; }

0 references
public KeyRandom(string letra, float percentaje)
{
    this.percentaje = percentaje;
    this.letra = letra;
}

1 reference
public KeyRandom(string letra, float percentaje, float cumulative)
{
    this.percentaje = percentaje;
    this.letra = letra;
    this.cumulative = cumulative;
}
```

Ilustración 4.18 Definición de la clase KeyRandom y sus constructores

A su vez tiene una función propia que actúa como constructor, esta función calcula primero el porcentaje acumulado de la letra y la añade a una lista. Como vuelve a ser necesario la generación de un número aleatorio se ha escogido la lista para poder generar las letras.

```
52 references
public static void Add(string name, float percentage, List<KeyRandom> lista)
{
    float cumulative = (lista.Count == 0) ? percentage : lista[^1].cumulative + percentage;
    lista.Add(new KeyRandom(name, percentage, cumulative));
}
```

Ilustración 4.19 Método estático de la clase que permite añadir una letra a una lista

Al final de la generación de las letras se suman todos los valores de los porcentajes y se guardan en una variable llamada *sumTotalPercentageList*. De esta forma tenemos definidas las funciones de generación y las estructuras necesarias para gestionar los porcentajes correctamente.

Por tanto la función que genera las letras esta vez genera un número aleatorio con *sumTotalPercentageList.* Luego realiza una búsqueda binaria, encuentra el índice correcto de la lista y genera la letra.

```
1 reference
IEnumerator CreateObjectLanguage()
{
    while (true)
    {
        float random;
        GameObject actualLetter;
        random = UnityEngine.Random.Range(0.0f, sumTotalPercentageList);
        int index = BinarySearch(random, percentageList);
        actualLetter = (GameObject)letterList[index];
        movingObject = Instantiate(actualLetter, new Vector3(actualLetter.transform.position.x, topScreen.transform.position.y, -0.2f), Quaternion.identity);
        yield return new WaitForSeconds(spawningSpeed);
    }
}
```

Ilustración 4.20 Método de generación de letras por porcentajes

4.4. Iteración 3

En esta tercera iteración las tareas a realizar son la H, I, J y K. Estas tareas corresponden con la implementación de parte de la interfaz diseñada y su funcionalidad, establecer la puntuación y el sistema de salud e implementar la pausa en el sistema.

4.4.1. Interfaz de menú principal y escena jugable

Siguiendo las directrices marcadas por el apartado 3.2 donde se definen los storyboards de todas las pantallas, solamente es colocar los elementos en un Canvas y agruparlos de forma que sea sencillo luego su manejo.

En este pequeño sub-apartado se tendrán en cuenta las subtareas de la tarea H y J ya que simplemente es su colocación en la escena. Para la implementación son simplemente UI_Text alojados en la escena con anchors con porcentajes de la escena.

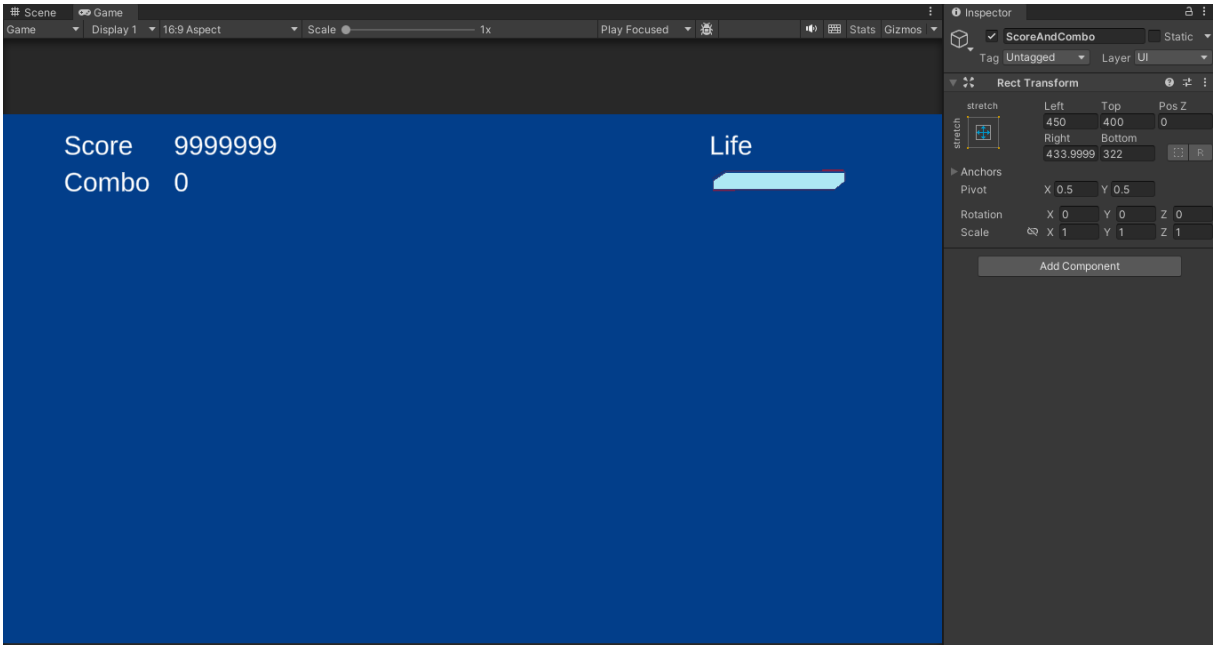


Ilustración 4.21 Interfaz de la escena GameZone

Sucesivamente se ha implementado el menú principal, que consiste en un fondo estático de estrellas con varias letras que sirven como botones. Cuando se pasa el pasa el ratón por encima, el texto se resalta en otro color indicando que es un botón. Exceptuando la funcionalidad del menú de ajustes, los demás submenús se han dejado para incrementos posteriores.

Además, en la parte derecha del menú hay un Input Field que permite introducir el nombre de usuario. Esto tiene dos principales usos: sirve para guardar los ajustes y permite guardar la puntuación.

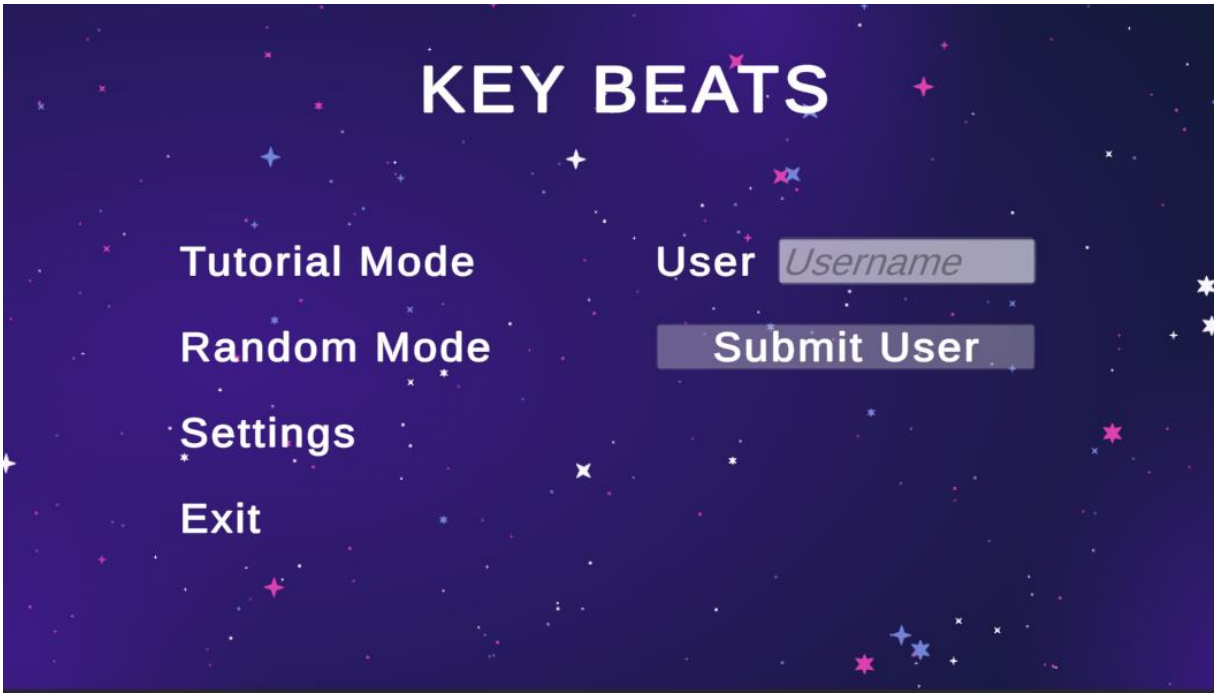


Ilustración 4.22 Interfaz de la escena MainMenu

Finalmente, el menú de ajustes y el menú de pausa comparten muchos elementos. El diseño de ambos menús es un panel cuadrado que aparece, oscureciendo los demás elementos de la escena y que consta de 3 sliders y varios iconos al fondo, ejerciendo como botones.

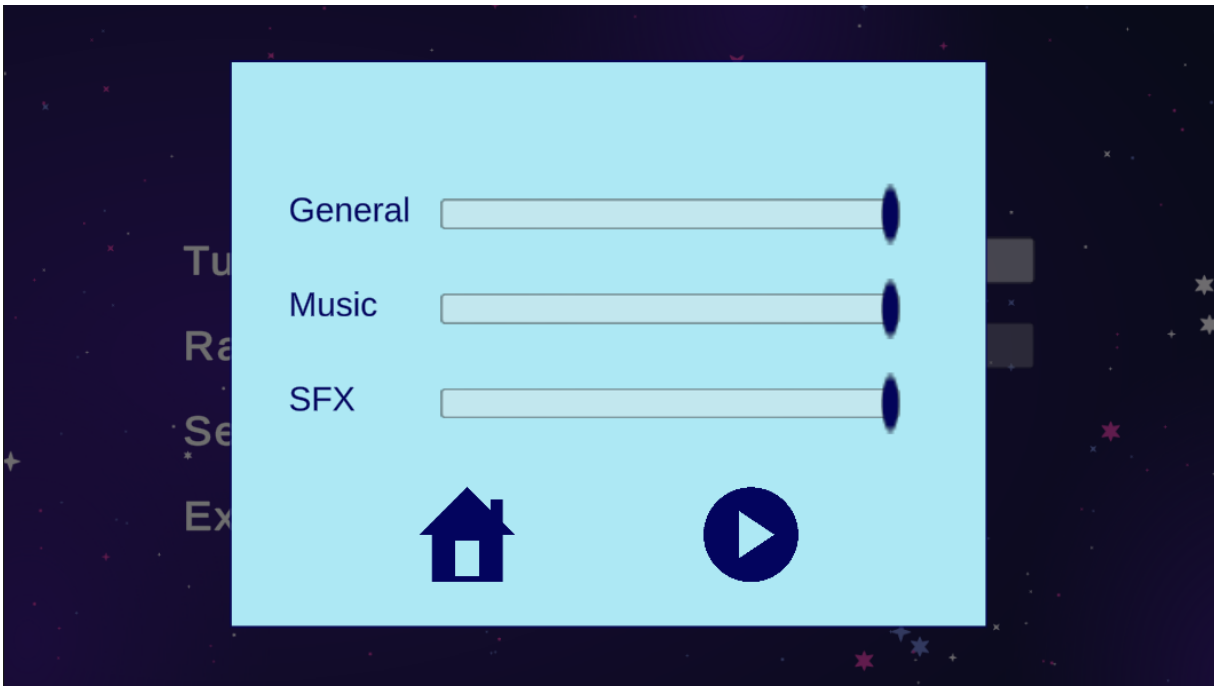


Ilustración 4.23 Interfaz del submenú de ajustes

Para la funcionalidad de los sliders se ha tenido en cuenta un script llamado SoundScript. Este script usa la funcionalidad de PlayerPrefs para grabar variables entre sesiones. Lo primero que hace este script es que cuando se carga la escena establece los valores de los sliders al valor guardado, estableciendo el volumen en el proceso.

Todo esto lo hace a través de 3 funciones similares llamadas *SetVolumenGeneral*, *SetVolumenMusic* y *SetVolumeSFX*. Estas funciones establecen el volumen del mixer correspondiente al valor del slider y son llamadas en *SetVolumeFromSlider(type).*

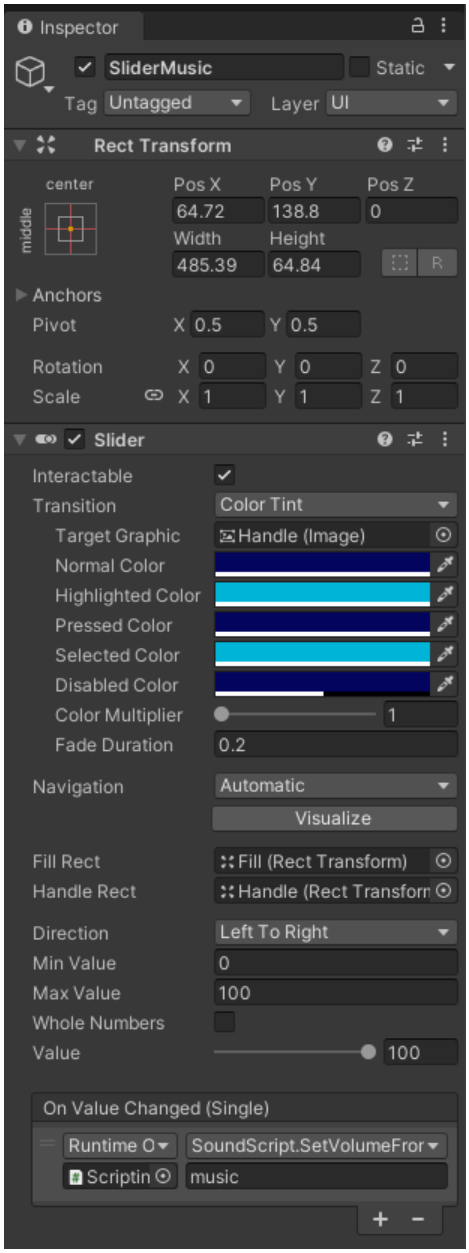


Ilustración 4.24 Valores del slider que gestiona la música

Luego en la parte inferior vemos dos botones. Realmente ambos devuelven al menú principal y se guardan los cambios igualmente, pero es para dar la sensación al jugador de que se han guardado los cambios, ya que podría malentenderse y pensar que si se vuelve al menú principal no se guardan los cambios.

```
0 references
public void Home() //Captura en el punto
{
    Time.timeScale = 1;
    SceneManager.LoadScene("MainMenu");
}
1 reference
```

Ilustración 4.25 Funcionalidad del botón home

Finalmente, el menú de pausa es exactamente igual a este, simplemente que tiene una función de reinicio de nivel, volviendo a cargar la escena.

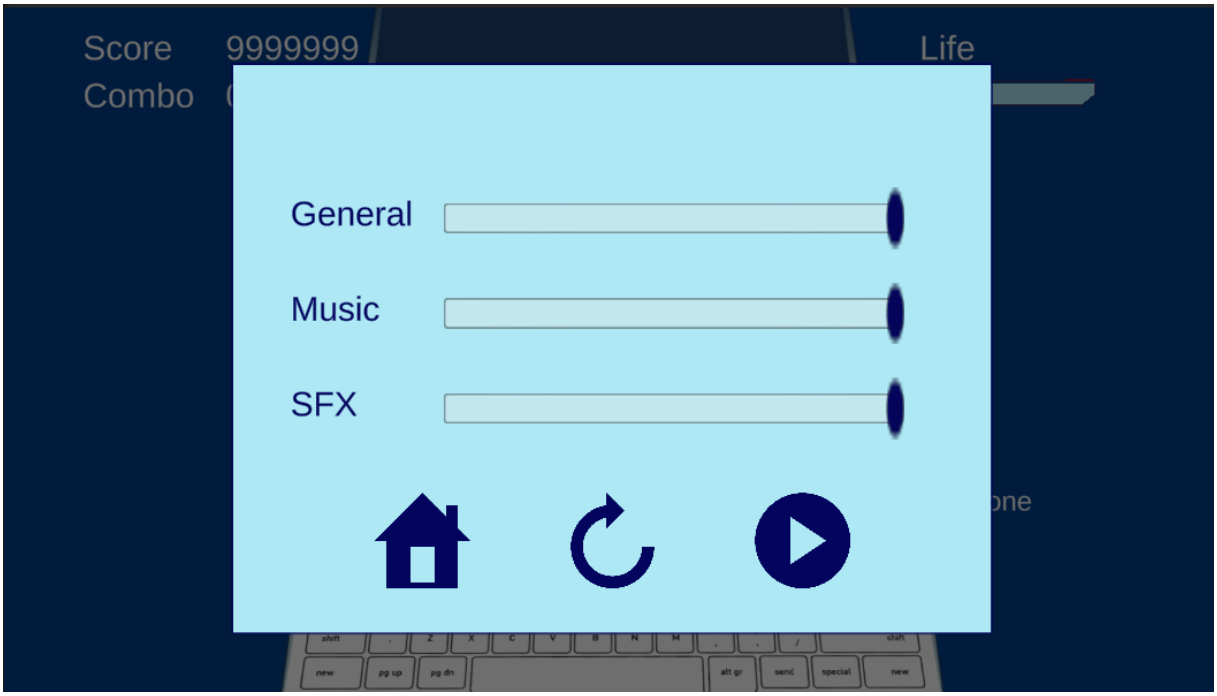


Ilustración 4.26 Interfaz del submenú de pausa

4.4.2. Score

Para la puntuación se anotan varios eventos distintos que pueden desencadenar un cambio de puntuación:

- Se ha acertado una letra
- No se ha conseguido acertar una letra y ha salido del collider por la parte inferior
- Por error se ha pulsado una tecla que no coincide con cualquier letra en el collider o no hay ninguna letra en el collider.

Además, se ha planteado premiar el que se acierten varias letras de forma consecutiva. De esta forma hay dos variables definidas claras: **score** con la puntuación y **combo** con el número consecutivo de aciertos. Luego de los eventos anteriores solamente el segundo implicará un reinicio en el valor de combo.

El primer evento es el más importante ya que es el único evento que sumará puntos en el juego y además tendrá una interacción con el valor de combo. De esta forma se ha establecido que la función matemática será la siguiente:

$$score_n = score_0 + \sum_{i=0}^{n-1} (100 + combo_i)$$

Donde:

- $score_n$ es la puntuación después de n iteraciones
- $score_0$ es la puntuación inicial, 0 al inicio del juego o el valor que haya al reiniciar el valor de combo
- n es el número total de valores de combo
- $combo_i$ es el valor de combo en la iteración i

Por tanto, el valor de una letra por si misma es 100 y luego es modificada por el valor de combo.

A partir del segundo evento es más sencillo ya que son valores planos, siendo el segundo evento tal que:

$$score_{new} = score - 20$$

$$combo_{new} = 0$$

Y el tercer evento sería:

$$score_{new} = score - 5$$

Por último, parte de la implementación ya estaba definida en *ClickScript* en las ilustraciones 4.16 y se complementa con esta parte de código:

```
if(scored==false&&scored2==false){
SoundFXScript.instance.PlayAudio(missAudio, 1f, MusicTypes.sfx,0);
score = score - 5;
scoreText.text = ("" + score);
}
if(scored2==true)
    scored2=false;
```

Ilustración 4.27 Extensión de la funcionalidad de ClickScript para un fallo de tecla

4.4.3. Salud

Para el sistema de salud se ha preferido usar un sistema sencillo de vidas con un valor máximo de 20. Habiéndose definido el sistema de puntuación en el apartado 4.4.2, vamos a tomar prestados los eventos definidos allí. De esta forma se han establecido dos situaciones en las cuales se van a modificar el sistema de salud.

- En el primer evento, el cual consiste en acertar una letra, se va a sumar una vida si se consigue acertar 5 veces seguidas
- No se ha conseguido acertar una letra y ha salido del collider por la parte inferior se va a restar una vida por cada letra que se elimine de esta forma

La implementación ya ha sido reflejada en las ilustraciones 4.26 para la funcionalidad del script de *ClickScript* y la ilustración 4.16 para el OnTriggerExit() del mismo script.

4.4.4. Añadir el menú de pausa

Para ello se ha creado un script llamado *PauseScript* al cual se le ha asociado el menú de pausa. En caso de que se pulse la tecla Escape en cualquier momento del juego se ejecuta la función Pause(). Esta función muestra el menú de pausa de forma activa y usa el sistema de Unity de tiempo a través de Time.timeScale=0 y en caso de que haya algún sonido activo lo pausa.

Cuando se vuelve a pulsar la tecla Escape cambia el tiempo, esconde el menú y reproduce el audio.

```
1 reference
public void Pause()
{
    PauseMenu.SetActive(true);
    Time.timeScale = 0;
    if(soundFXScript!=null)
    soundFXScript.Pause();
}
1 reference
public void Resume()
{
    PauseMenu.SetActive(false);
    Time.timeScale=1;
    if(soundFXScript!=null)
    soundFXScript.Resume();
}
```

Ilustración 4.29 Código de los métodos Pause y Resume del PauseScript

```
0 references
public void Pause(){
    foreach(AudioSource source in sources)
        source.Pause();
}

0 references
public void Resume(){
    foreach(AudioSource source in sources)
        source.UnPause();
}
```

Ilustración 4.28 Código de los métodos Pause y Resume del SoundFXScript

4.5. Iteración 4

En esta cuarta iteración las tareas a realizar son la L, M, N y O. Estas tareas corresponden con la implementación de parte de la interfaz, concretamente del Menú Random y su funcionalidad, establecer los marcadores y la pantalla de fin de juego, así como definir los niveles de dificultad.

4.5.1. Fin de juego y sistema de dificultad

También se realizó la subtarea de establecer la cabecera de los archivos csv pero fue explicado en el apartado 4.3.1

Lo primero que se realizó fue realizar la implementación del diseño de la interfaz del modo de fin de juego. Este consiste en un texto en la parte izquierda indicando si se ha ganado o perdido y la puntuación actual, en la parte derecha se encuentra el leaderboard. Finalmente, en la parte inferior hay un par de botones de reiniciar el nivel y volver al menú principal

[Captura interfaz]

Implementada la interfaz, se procede a implementar la funcionalidad en el juego. Como se vio anteriormente se implementó un sistema de vidas, por lo que en *ClickScript* se establece un check que compruebe si todavía quedan vidas y sino que muestre el menú de fin de partida.

```
        if(slider.value<=0){
            //Mostrar escena de gameOver
            EndLevel("You Lose More luck next time");
        }
    }
    1 reference
    public void EndLevel(string text){
            Time.timeScale = 0;
            EndMenuText.text = text;
            EndMenuScoreText.text = scoreText.text;
            EndMenu.SetActive(true);
            //Llamamos al leaderboard
            this.GetComponent<LeaderboardScript>().SubmitUser(score);
    }
```

Ilustración 4.30 Comprobación del las vidas restantes y función de fin de juego

Aparte, se quiere que cuando se elimine la última letra de un nivel se acabe el juego. Para ello hay que modificar la función *PlayLevel()* en *LetterManagement.* Para ello se guarda la referencia del último objeto creado y se comprueba cada segundo si ha sido eliminado.

```
1 reference
private IEnumerator PlayLevel()
{
    foreach (LevelKey levelKey in levelList)
    {
        if(levelKey.getKey()!="None"){
        GameObject letter = dicctionaryList[levelKey.getKey()];
        movingObject = Instantiate(letter, new Vector3(letter.transform.position.x, topScreen.tr
        yield return new WaitForSeconds((float)(levelKey.getTimeToWait() / 1000.0));
    }
    //After that, when the movingObject is deleted that means that the level is finshed
    StartCoroutine(CheckIfObjectIsDestroyed());
}
1 reference
private IEnumerator CheckIfObjectIsDestroyed(){
    while(movingObject!=null){
        yield return new WaitForSeconds(1.0f);
    }
    click.EndLevel("YOU WIN");
}
```

Ilustración 4.31 Comprobación de la última letra generada

Para el selector de dificultad, se ha creado un enumerado Difficulty con tres valores: Easy, Medium y Hard. Este enumerado será llamado por el menú principal para seleccionar la dificultad.

Luego se ha establecido valores distintos de velocidad y tiempo de generación para cada uno de ellos.

```
1 reference
private void settingSpeedAndSpawnRate()
{
    if (dificulty == Difficulty.Easy)
    {
        MovementScript.speed = 3f;
        spawningSpeed = 1.6f;
    }
    if (dificulty == Difficulty.Medium)
    {
        MovementScript.speed = 4f;
        spawningSpeed = 0.8f;
    }
    if (dificulty == Difficulty.Hard)
    {
        MovementScript.speed = 5f;
        spawningSpeed = 0.4f;
    }

}
```

Ilustración 4.32 Distintos valores en los diferentes modos de dificultad

4.5.2. Leaderboard

El sistema de marcadores trajo más quebraderos de cabeza de los necesarios. Todos los marcadores usan un content view al cual se le van añadiendo prefabs con los datos del usuario.

Por tanto, la funcionalidad se dividió en dos Scripts: LeaderBoardDisplayScript que está enganchado a cada uno de los prefabs y LeadeBoardScript que posee toda la lógica de ordenación y guardado de archivos de los marcadores.

La clase que guarda los datos de un jugador se llama *PlayerInfo* y posee dos atributos: name y score. También se sobrescribe el comparador de objetos para compararlos por score y luego alfabéticamente.

```
21 references
public class PlayerInfo : IComparable<PlayerInfo>
{
    4 references
    public string name;
    6 references
    public int score;
    4 references
    public PlayerInfo(string name, int score)
    {
        this.name = name;
        this.score = score;
    }
    0 references
    public int CompareTo(PlayerInfo other)
    {
        // Comparar por puntuación, y si son iguales, comparar por nombre
        if (score == other.score)
        {
            return name.CompareTo(other.name);
        }
        return other.score.CompareTo(score); // Orden descendente por puntuación
    }
    1 reference
```

Ilustración 4.33 Definición de clase PlayerInfo

De esta forma el funcionamiento normal del sistema es que se llame a SubmitUser(score). Esta carga el leaderboard actual usando las variables de *LetterManagement* como el level path o el nivel de dificultad. Luego añade el score del usuario y exporta el nuevo marcador y finalmente repinta el marcador.

```
1 reference
void ImportLeatherBoard()
{
    collectedStats = new SortedSet<PlayerInfo>();
    string route;
    if (LetterManagerScript.randomMode == true)
    {
        route = "./Assets/LeaderBoard/random-" + LetterManagerScript.dificulty.ToString() + ".csv";
    }
    else
    {
        string levelPath = LetterManagerScript.levelPath;
        string level = levelPath.Split('/').Last().Split('.').First();
        route = "./Assets/LeaderBoard/level-" + level + ".csv";
    }
    try
    {
        foreach (string line in File.ReadLines(route, Encoding.UTF8))
        {
            string[] words = line.Split(';');
            PlayerInfo player = new PlayerInfo(words[0], Int32.Parse(words[1]));
            collectedStats.Add(player);
        }
```

Ilustración 4.35 Sistema de importación de marcadores

```
3 references
private void OnLeaderboardLoaded(List<PlayerInfo> collectedStats)
{
    foreach (Transform t in _entryDisplayParent)
        Destroy(t.gameObject);
    int rank = 1;
    foreach (var t in collectedStats)
    {
        CreateEntryDisplay(t, rank);
        rank++;
    }
}

1 reference
private void CreateEntryDisplay(PlayerInfo entry, int rank)
{
    var entryDisplay = Instantiate(_entryDisplayPrefab.gameObject, _entryDisplayParent);
    entryDisplay.GetComponent<LeaderBoardDisplayScript>().SetEntry(entry, rank, false);
}
```

Ilustración 4.34 Pintado de marcadores en escena

Por tanto finalmente se puede integrar el leaderboard con la función EndGame() de *ClickScript* e introducir un marcador en la pantalla de fin de juego

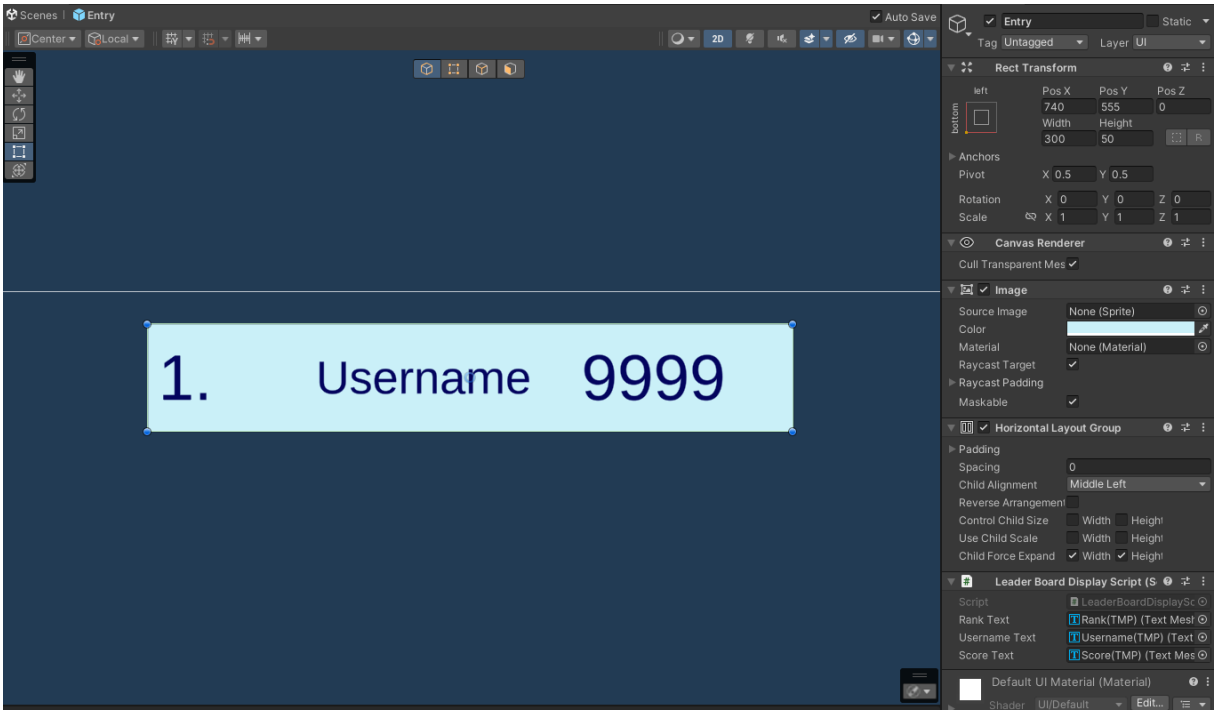


Ilustración 4.36 Prefab del elemento de los marcadores

4.5.3. Interfaz modo random

Para el submenú del modo random se han dispuesto de dos Toogle Group, uno para seleccionar la dificultad y otro para seleccionar el diccionario de idiomas. Luego en la parte inferior se tienen los marcadores de los diferentes modos de dificultad.

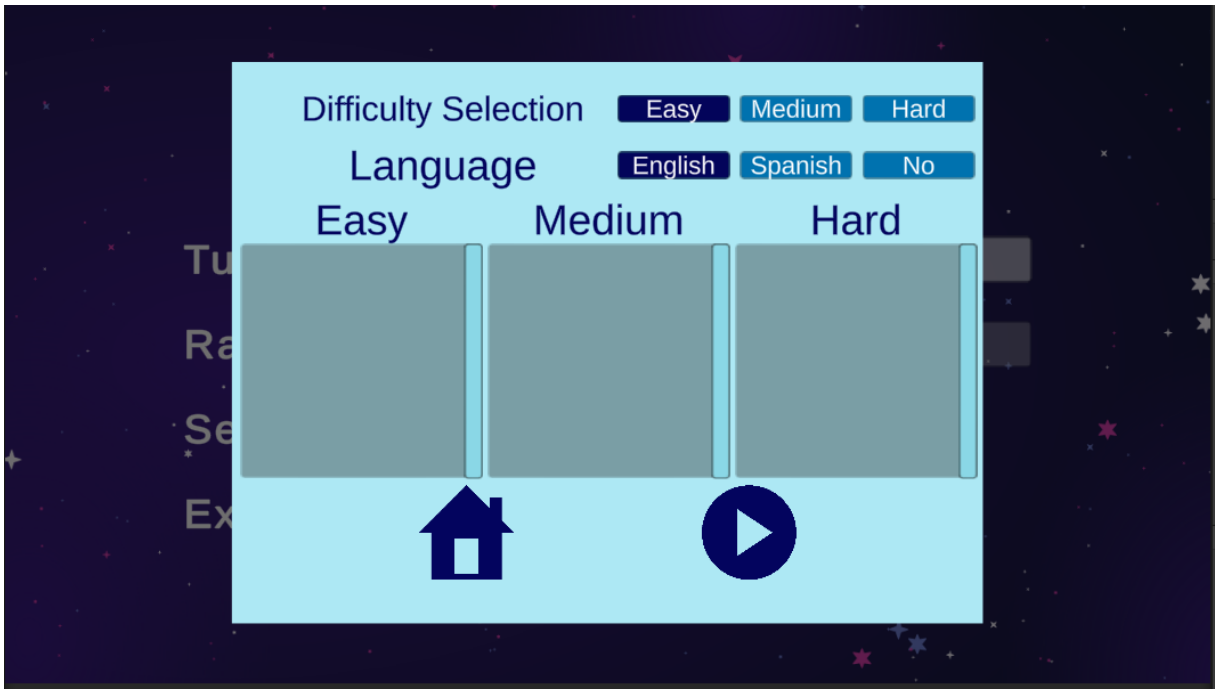


Ilustración 4.38 Interfaz del submenú random

Cuando se cambia uno de estos valores se modifican las variables de *LetterManagerScript* de modo que cuando se pulsa el botón de inicio, están las variables ya seleccionadas y no hay problemas de pasar variables entre escenas.

```
0 references
public void randomMode()
{
    LetterManagerScript.randomMode = true;
    SceneManager.LoadScene("GameZone");
}
```

Ilustración 4.37 Código del botón de play en el menú random

4.5.4. Letras 3D

Para la realización de las letras 3D se ha usado un plugín de Unity conocido como ProBuilder. Este plugin permite de forma sencilla modificar mesh ya existentes o crear nuevas.

Para este proyecto simplemente ha sido crear un prisma con la forma del Sprite 2D. Esto con una buena iluminación de escena ha sido suficiente para crear una sensación de profundiad

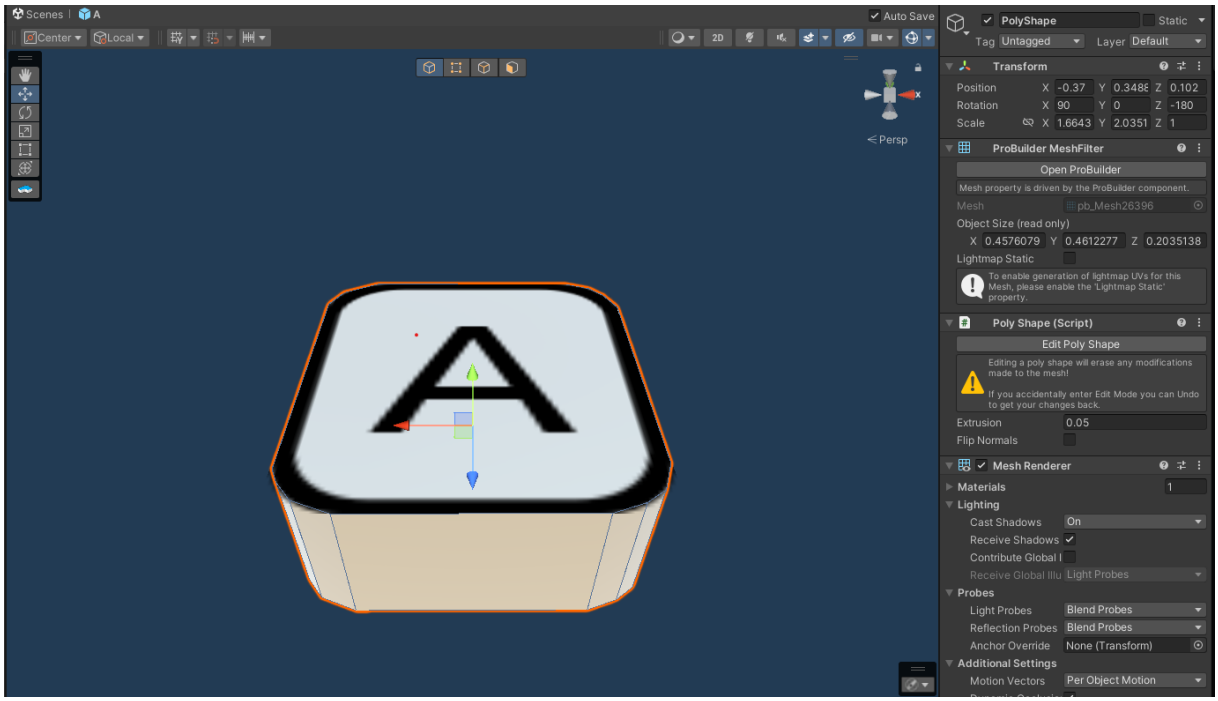


Ilustración 4.39 Datos del modelo 3d en el editor de ProBuilder

4.6. Iteración 5

#TODO Se está realizando actualmente, por lo que se realizará mas adelante

5. CONCLUSIONES

Tras la realización de este prototipo se puede dejar claro que la elaboración de un prototipo es compleja ya que cualquier elemento en el tiene que ser estudiado previamente y realizado una estimación de costes.

A su vez, se ha podido profundizar en el motor gráfico Unity y se han aprendido desde funcionalidades nuevas hasta optimización de código y recursos dentro del mismo, usando para ello todo el conocimiento que se ha aprendido de Desarrollo de Videojuegos, Fundamentos de Ingenieria de Software, Animación 3D y postprocesamiento, Ingeniería gráfica o Gestión y Control de Proyectos Informáticos entre otras.

Por último, cabe destacar que la idea de realizar un serious game da pie a más investigación en el campo de la enseñanza y en buscar nuevas ideas de hacer que una tarea que es repetitiva y laboriosa se vuelva más amena o incluso divertida.

Bibliografía

Adobe. (s.f.). *Comparar Planes*. Recuperado el 22 de Julio de 2024, de Adobe: https://www.adobe.com/es/products/photoshop/plans.html

Adobe Photoshop. (5 de Julio de 2024). Recuperado el 22 de Julio de 2024, de Wikipedia: https://en.wikipedia.org/wiki/Adobe_Photoshop

Amaro Calderón, S. D., & Valverde Rebaza, J. C. (2007). Metodologías Agiles. *Universidad Nacional de Trujillo*, 37.

Asociación Española de Empresas Productoras y Desarrolladoras de Videojuegos y Software de Entretenimiento, DEV. (8 de Junio de 2022). *DEV Desarrollo Español de Videojuegos.* Recuperado el 21 de Julio de 2024, de https://www.dev.org.es/images/stories/docs/libro%20blanco%20del%20desarrollo%20espanol%20de%20videojuegos%202022.pdf

Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001). *Manifesto for Agile Software Development*. Recuperado el 21 de Julio de 2024, de http://agilemanifesto.org/authors.html

Bianchi, F. (s.f.). *Coolors*. Obtenido de https://coolors.co/

C Sharp. (9 de Mayo de 2024). Recuperado el 21 de Julio de 2024, de Wikipedia: https://es.wikipedia.org/wiki/C_Sharp

Domínguez, J. L. (4 de Septiembre de 2014). *Weduvi*. Recuperado el 21 de Julio de 2024, de La importanica de la Escritura: http://weduvi.com/la-importancia-de-la-escritura/

Game Engine. (15 de Julio de 2024). Recuperado el 21 de Julio de 2024, de Wikipedia: https://en.wikipedia.org/wiki/Game_engine

GIMP. (14 de Mayo de 2024). Recuperado el 22 de Julio de 2024, de Wikipedia: https://es.wikipedia.org/wiki/GIMP

Git. (9 de Julio de 2024). Recuperado el 21 de Julio de 2024, de Wikipedia: https://en.wikipedia.org/wiki/Git

Letter Frecuency. (25 de Julio de 2024). Recuperado el 28 de Julio de 2024, de Wikipedia: https://en.wikipedia.org/wiki/Letter_frequency

Menzinsky, A., López, G., Palacio, J., Sobrino, M., Álvarez, R., & Rivas, V. (Agosto de 2022). *Historias de Usuario.* Recuperado el 22 de Julio de 2024, de Scrum Manager: https://www.scrummanager.com/files/scrum_manager_historias_usuario.pdf

O'Neill, J. (15 de Enero de 2008). *GameDaily*. Recuperado el 21 de Julio de 2024, de Archive.org: https://web.archive.org/web/20090830205358/http://www.gamedaily.com/articles/features/my-turn-the-real-cost-of-middleware/71334/?biz=1

Petersen, W. (6 de Febrero de 2023). *Type IT!* Recuperado el 21 de Julio de 2024, de https://touchtypeit.co.uk/just-how-many-people-can-touch-type

Pristupov, D. (13 de Noviembre de 2020). *[@git_fork] C#+WPF on Windows and Swift+Cocoa on Mac.* Obtenido de X. https://x.com/AlbertoChargoy/status/1327077294100373504

Requirement. (26 de Junio de 2024). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Requirement

SmartGit. (13 de Agosto de 2023). Recuperado el 21 de Julio de 2024, de Wikipedia: https://ru.wikipedia.org/wiki/SmartGit/Hg

syntevo GmbH. (19 de Julio de 2024). *Integrations.* Recuperado el 21 de Julio de 2024, de syntevo Docs: https://docs.syntevo.com/SmartGit/Latest/; https://github.com/syntevo/syntevo.github.io

syntevo GmbH. (19 de Julio de 2024). *Licensing*. Recuperado el 21 de Julio de 2024, de syntevo Docs: https://docs.syntevo.com/SmartGit/HowTos/Licensing.html

The Institute of Electrical and Electronics Engineer. (1991). *IEEE Standar Glossary of Software Engineering Teminology.* New York: IEEE Publications,U.S.

Tpoint Tech. (2011). *Incremental Model*. Recuperado el 21 de Julio de 2024, de JavaTPoint: https://www.javatpoint.com/software-engineering-incremental-model

Ureña Lopez, L. A., & Gómez Espínola, J. I. (2016). Tema 2. Modelos del Proceso. *Universidad de Jaén*. Recuperado el 21 de Julio de 2024