TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

TÍTULO

Autor: David Pulido Marchal

Grado: Ingeniería Informática

Director: Juan José Jiménez Delgado

Departamento del director: Informática

Fecha: 10/09/2024

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, septiembre de 2024

El alumno: Los tutores:

David Pulido Marchal Francisco Daniel Pérez Cano

Don Juan José Jiménez Delgado

Índice

Índice de ilustraciones

Índice de tablas

1. INTRODUCCIÓN

En esta sección se explicarán las distintas motivaciones con respecto a la decisión de este tema y sobre el trasfondo de este 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 acabar el trimestre, solo quedamos la mitad de la clase 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 la mejora de la habilidad en la mecanografía.

Para sorpresa de mucha gente, hoy en día 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. 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 un proceso necesario 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 Microsoft Teams.

1.2. Objetivos del trabajo

El objetivo general es el desarrollo de un prototipo de un videojuego para el aprendizaje de mecanografía, 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.

Para poder visualizar de forma correcta el avance, se contará con un sistema de puntuación a través de un marcador global por cada nivel, donde los usuarios compararán su puntuación con sus amigos. Esto fomentará la motivación y la competitividad de los usuarios por mejorar y mejorando el aprendizaje de forma indirecta.

1.3. 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 como se puede observar en la Ilustración 1.1.

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).

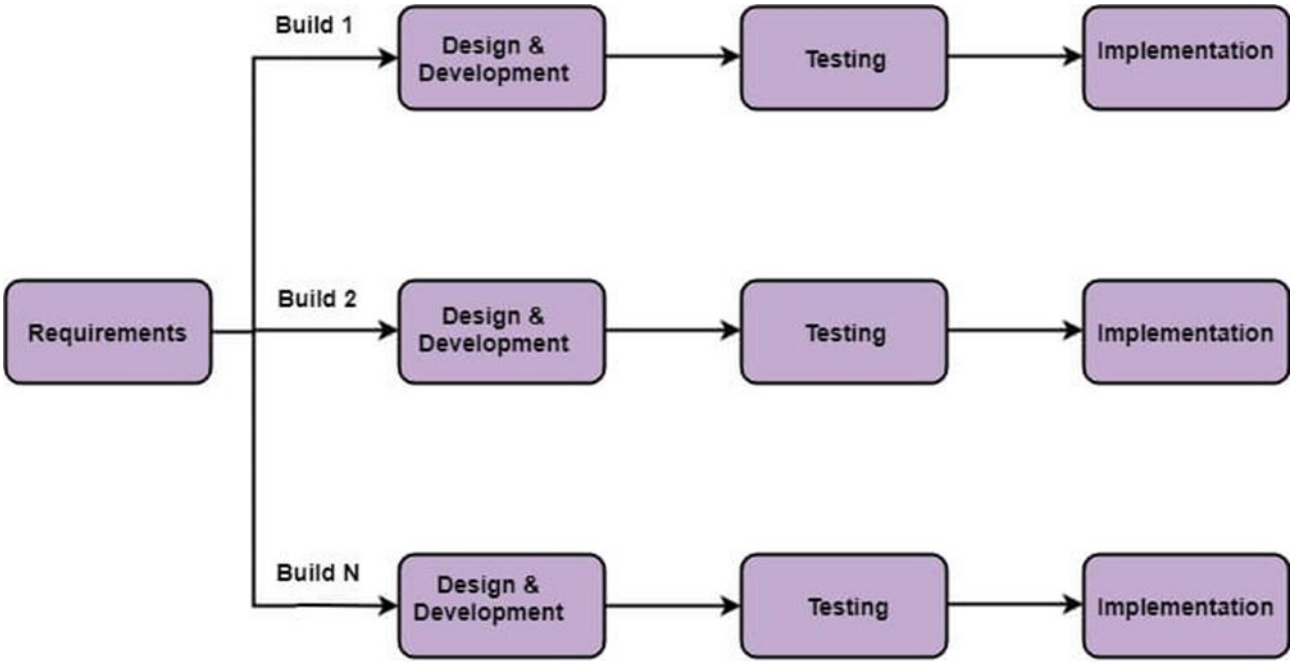

Ilustración 1.1 Funcionamiento de un modelo incremental

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 posee las características más importantes en etapas tempranas y por tanto dar realimentación 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.

1.4. 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.
- **Manual de usuario**: Al final del documento existe otro apartado que detalla los controles y funcionalidades de la interfaz de cada escena.

2. ANÁLISIS

En esta sección se muestran la definición de las diferentes necesidades que tiene el usuario y el equipo de desarrollo. Para ello, se definen los diferentes requisitos del proyecto para definir el alcance. Además, se estudian las diferentes alternativas tecnológicas con las limitaciones técnicas que plantean.

Al finalizar esta primera fase, se plantean la definición de las tareas a realizar y se calculan los diferentes presupuestos del prototipo.

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 sobre los puntos anteriores.

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 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 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**: 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. Además, tras lograr cierto número de aciertos se debe de añadir más intentos de fallos hasta llegar a un límite.

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. Marcador: 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 serán el español e inglés.
    2. Modos de dificultad: Se podrán 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 preconfigurado 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 realimentación de forma inmediata 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 está expuesto en la Tabla 2.1:

ID	Título	Descripción	PH	Valor
1	Menú	• Como usuario • Quiero que el juego disponga de un menú sencillo • Para poder usar mis conocimientos previos	7	10
2	Selector de volumen	• Como usuario • Quiero poder bajar el volumen • Para establecer el volumen ideal	1	3
3	Selector de nivel	• Como usuario • Quiero seleccionar un nivel • Para poder jugar un nivel en concreto	2	10
4	Username	• Como usuario • Quiero introducir mi nombre • Para poder grabar mi puntuación	1	3
5	Puntuación	• Como usuario • Quiero grabar mi puntuación • Para competir con mis amigos	4	10
6	Reinicio nivel	• Como usuario • Quiero poder reiniciar el nivel • Para volver a intentar un fallo sin tener que salir al menú	1	6
7	Menú principal	• Como usuario • Quiero tener la opción de volver al menú • Para cambiar de modo de juego	2	6
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	2	5
9	Ritmo	• Como usuario • Quiero las letras coincidan con la música • Para acertar de forma más sencilla	6	8
10	Dificultad	• Como usuario • Quiero que existan varios niveles de dificultad • Para poder escoger jugar	4	10

según mi habilidad

Tabla 2.1 Especificaciones de las historias de usuario

2.3. Estudio de alternativas y viabilidad

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

2.3.1. 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).

En la tabla 2.2, se presentan 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 partículas.
Código fuente	No open-source.	No considerado open-source, pero se puede acceder al código del motor.

Precio	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 2.2 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 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, será gratuito. Hay que tener en cuenta esta particularidad, sobre todo si se quiere calcular los costes para lanzarlo como un producto comercial.

2.3.2. Entorno de desarrollo

Un entorno de desarrollo es un espacio de trabajo que permite a los desarrolladores crear o modificar una aplicación. En la tabla 2.3, se presentan los diferentes entornos de desarrollo que se han estudiado y las diferencias entre los mismos:

	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	Múltiples (C++, C#, Fortran, .NET, Java, Python, PHP...)	Múltiples (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 2.3 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.

2.3.3. 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. En la tabla 2.4, se presentan los diferentes clientes de Git que se han estudiado y las diferencias entre los mismos:

	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 2.4 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.

2.3.4. Edición de imágenes

A la hora de realizar todos los sprites y los iconos personalizados es necesario usar un editor de imágenes. En la tabla 2.5, se presentan los programas de edición de imagen que se han estudiado y las diferencias entre los mismos:

	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 última versión han añadido IA Generativa que para generar partes de la imagen a través de instrucciones y de la imagen actual.

Tabla 2.5 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 suscripción 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.

2.4. Tecnologías utilizadas

A continuación, se van a listar los diferentes programas que han sido utilizados en el desarrollo de este prototipo. Además, se enunciarán los motivos principales de elección.

2.4.1. Unity

Ilustración 2.1 Logo de Unity

Como se ve en la Ilustración 2.1, en lo que respecta al motor gráfico, se ha decidido utilizar Unity porque la curva de aprendizaje es menor frente al resto de opciones y los resultados que se pueden obtener, en términos de calidad visual, son similares. Además, 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.

2.4.2. Visual Studio Code

Ilustración 2.2 Logo de Visual Studio Code

Como se ve en la Ilustración 2.2, en lo que respecta al entorno de desarrollo, se ha decidido utilizar Visual Studio Code sobre Visual Studio 2019. Visual Studio 2019 ya se había usado anteriormente en la asignatura de Desarrollo de Videojuegos, dispone de muchas facilidades y perfecta compatibilidad con Unity. Sin embargo, es un programa muy pesado y por tanto podría llegar a ralentizar el proceso de programación, haciendo de todo el proceso de desarrollo muy ineficiente.

Visual Studio Code 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.

2.4.3. Fork

Ilustración 2.3 Logo de Fork

Como se ve en la Ilustración 2.3, en lo que respecta al cliente de Git se ha decidido utilizar Fork. Fork es una herramienta fácil, moderna que posea todos los requerimientos típicos y es altamente actualizado. Asique este programa que si bien tiene un coste de 59,99$, se puede usar perfectamente con su evaluación gratuita indefinida.

Se ha decidido sobre SmartGit debido a una razón, 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. Además, para la funcionalidad básica de Git consistente en un control de versiones sencillo,

con varias ramas y con un comparador de texto integrado en la aplicación, Fork cumple de forma correcta. En caso de querer seguir el desarrollo y pagar al desarrollador de la aplicación, el coste de Fork es adecuado para una licencia de por vida.

2.4.4. GIMP

Como se ve en la Ilustración 2.4, en lo que respecta al editor de imágenes se ha decidido utilizar GIMP. Los principales motivos para escoger GIMP son que es un software gratuito y, pese a 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. Además, tiene una comunidad suficientemente grande como para desarrollar extensiones y plugins y realizar tutoriales que permiten desarrollar los recursos necesarios en un videojuego.

Ilustración 2.4 Logo de GIMP

La alternativa a este software era usar Adobe Photoshop, un software propietario desarrollado y publicado por Adobe Inc. Cuyo coste ha supuesto un impedimento considerable.

2.4.5. Audacity

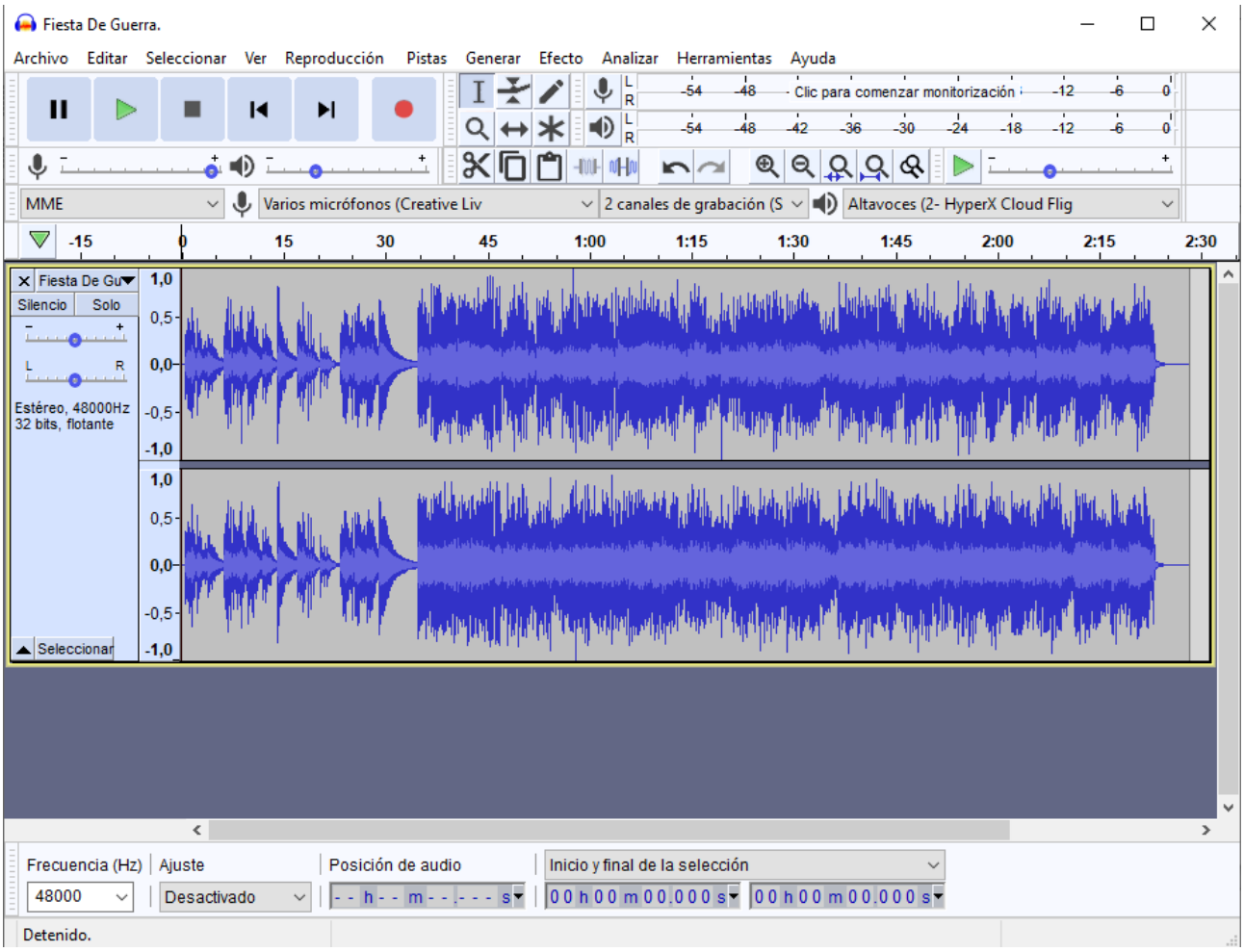

Ilustración 2.5 Interfaz de Audacity

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. El uso principal que se le ha dado ha sido el de recortar canciones y normalizar los niveles de audio. En la Ilustración 2.5 se puede ver un poco de la interfaz de uso del programa.

2.4.6. Probuilder

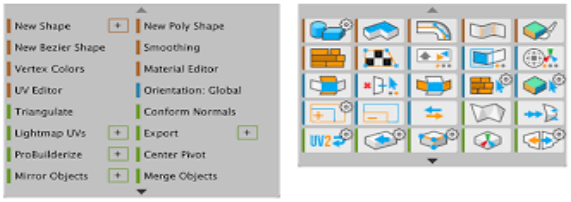

Ilustración 2.6 Interfaz de Probuilder

Como se puede observar en la Ilustración 2.6, Probuilder es un plugin desarrollado para Unity y 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. El uso que tiene esta aplicación es bastante limitado pero para modificar y crear formas sencillas es suficiente.

2.5. Planificación

El objetivo de la planificación es permitir hacer estimaciones de recursos humanos, costes y planificar temporalmente el calendario del proyecto.

2.5.1. Tareas

A la hora de la elaborar las distintas tareas se ha seguido la metodología descrita en el apartado 1.3. De esta forma, cada incremento es una versión entregable que posee un apartado jugable y que puede ser entregado al cliente para que pruebe las nuevas funcionalidades jugables.

Se ha optado por dividir los incrementos por funcionalidad jugable más que por tiempo. Así, el cliente ve un producto que no tiene fallos graves y que cumple la planificación de los incrementos, exceptuando el tiempo. Esto garantiza que el cliente tiene toda la funcionalidad acordada. Además, la funcionalidad principal es implementada en los primeros incrementos.

A la hora de realizar la realimentación con el cliente, se han hecho varias consultas con los tutores cuando estos han estado disponibles y si no fuera posible el caso con familiares y amigos.

Con respecto a los errores, si bien muchos de estos serán encontrados y corregidos en las primeras iteraciones, se ha dispuesto una tarea exclusivamente a

realizar pruebas, intentando encontrar errores a través de comportamientos anómalos y siguiendo una serie de acciones metódicas.

A partir de los requisitos funcionales definidos anteriormente podemos extraer las siguientes tareas y subtareas como se puede ver en la Tabla 2.6, para conseguir realizar el prototipo.

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	A	5
C	Diseño	1. Diseño de diagramas e interfaz 2. Elaboración de la documentación de Análisis y Diseño	B	5
D	Bucle jugable	1. Crear escena principal con elementos estáticos 2. Añadir todas las “letras” 3. Crear script de funcionalidad de bajada y eliminación	B	2
E	Definir sistema generación	1. Definir función de generación 2. Guardar datos en un archivo csv	C	2
F	Añadir música y sonido	1. Investigar como generar audio en Unity 2. Implementar música de fondo 3. Implementar sonidos	C	2
G	Limpieza de elementos	1. Definir un colisionador para eliminar las letras no pulsadas	C	0,5
H	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

I	Salud	1. Implementar sistema de salud 2. Implementar barra visual de vidas	C	1
J	Bucle jugable II	1. Realizar modificaciones visuales del escenario 2. Implementar realimentación visual 3. Añadir pausa al nivel	C	1
K	Puntuación	1. Añadir funciones de Puntuación y combo 2. Visualizar Puntuación y Combo en IU	I	1
L	IU	1. Diseñar la interfaz en todas las pantallas 2. Añadir iconos 3. Implementar un menú principal que permita acceder al bucle jugable 4. Implementar un menú de pausa 5. Implementar un control deslizante para cada tipo de volumen 6. Corregir errores sonido	E, I	4
M	Bucle 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
N	Marcadores	1. Añadido sistema de usuarios 2. Definido e implementado el marcador 3. Marcadores por modo de juego 4. Modificado IU de fin de juego para tener marcador	L. K	4
O	IU II	1. Añadido Menú de Modo Random 2. Realimentación visual de botones y letras	K, M	1,5
P	Letras 3D	1. Diseño 3D de letras 2. Modificar escena para más efecto inmersivo	L	0,5
Q	IU III	1. Implementar menú de selección de niveles 2. Mejoras en interfaz del bucle	N	3

		jugable 3. Implementar paleta de colores		
R	Niveles	1. 5 nivel de prueba 2. Función de cálculo de tiempo 3. Modo depuración para creación de niveles	C	5
S	Modo Random	1. Implementar diccionario inglés 2. Corregir errores de generación	G, L	1
T	Pruebas	1. Realizar prueba todos las pantallas y modos de juego 2. Elaboración de la documentación de la metodología	R	2
U	Documentación	1. Redactar y elaborar la documentación y los entregables.	-	5

Tabla 2.6 Descripción de las tareas totales

Las tareas están divididas en varios Incrementos como se puede apreciar en la Tabla 2.7:

INCREMENTO	INICIO	FIN	TAREAS
0	20/05/24	10/06/24	A, B, C
1	11/06/24	12/06/24	D
2	13/06/24	20/06/24	E, F, G, H
3	21/06/24	01/07/24	I, J, K, L
4	02/07/24	12/07/24	M, N, O, P
5	15/07/24	29/07/24	Q, R, S, T

Tabla 2.7 Especificaciones de incrementos

La única tarea que no ha sido incluida en ningún incremento ha sido la tarea U. La tarea U ha sido llevada a cabo a lo largo del documento, en el cual, al finalizar cada incremento, se han ido recopilando datos y elaborando distintos borradores de esta memoria.

Esto ha sido clave a la hora de la redacción y la continuidad del documento.

2.5.2. Diagrama de Gantt

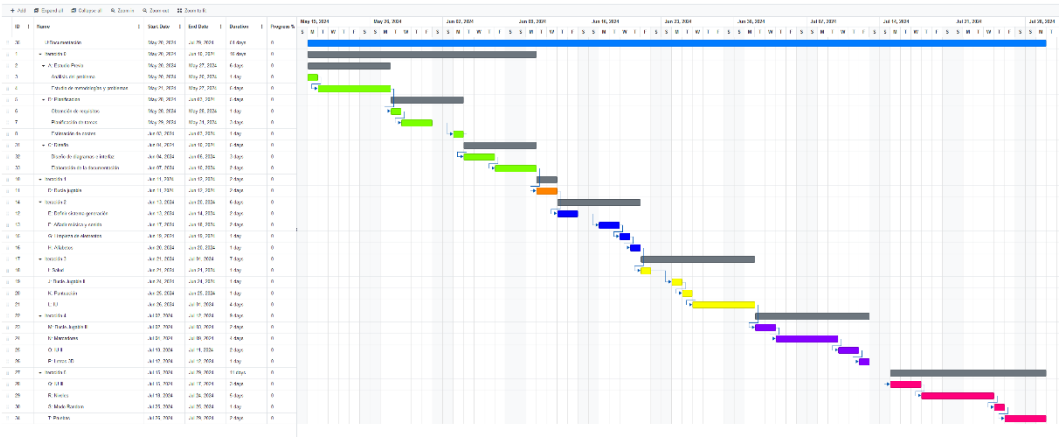

Ilustración 2.7 Diagrama de Gantt

En la Ilustración 2.7, 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

2.5.3. Variaciones

- La funcionalidad que se ha sido modificada más veces ha sido la función de bajada de letras del bucle 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.

2.6. 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 158h. Según Indeed (Indeed, 2024), el sueldo promedio de este puesto es de 18€/h y por tanto el coste total es de 2.844€.

Por otra parte, el programador se estima que trabajará 280h siendo el precio de 17€/h según Indeed (Indeed, 11) por lo que el coste total es de 4.760€. Por tanto, el coste del personal queda reflejado en la Tabla 2.8.

Puesto Trabajador	Coste mensual (8x5)	Coste (€/h)	Tiempo trabajado	Coste Total
Analista	2.880€	18€	128h	2.844€
Programador	2.720 €	17€	280h	4.760€
Total				7.604 €

Tabla 2.8 Salario de los trabajadores del proyecto

Como bien se ha descrito en el apartado 2.4.2, gracias a que el software ha sido programas gratuitos, open-source o utilizando pruebas de evaluación no hay costes en este apartado. Sin embargo, se va a hacer un desglose de cada uno de los programas utilizados y se harán los cálculos de software pagando las licencias en caso de existir una prueba gratuita.

En caso de que el coste de algún software se encuentre en dólares estadounidenses, se va a estimar una tasa de cambio de 1 EUR equivale a 1 dólar estadounidense. Por tanto, el desglose de cada uno de los programas utilizados se visualiza en la Tabla 2.9.

Programa	Objetivo	Coste
Unity	Motor Gráfico	0€
Visual Studio Code	Entorno de desarrollo	0€
Fork	Control de versiones	60€
GIMP	Edición de imágenes	0€
Audacity	Edición de música y sonido	0€
Probuilder	Modelaje 3D	0€
Total		60€

Tabla 2.9 Costes del software

Un equipo informático promedio suele tener un valor de 800€ y se estima que puede llegar a tener entre 5 a 10 años de vida útil. Se tomará como vida útil el valor mínimo de esa estimación, por lo que la vida útil del equipo será de 5 años Suponiendo que la duración del proyecto es aproximadamente de 3 meses y, que los costes del equipo son amortizables, los costes finales del equipo utilizado en el desarrollo son de 40€. Por tanto, los costes derivados del equipo se pueden observar en la Tabla 2.10.

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

Tabla 2.10 Tabla de amortizaciones

Finalmente, el presupuesto total es la suma de los costes presentados anteriormente, presentado en la Tabla 2.11.

Descripción del coste	Valor (€)
Coste de personal	7.604€
Coste hardware	40€
Coste del software	60€
Subtotal	7.704€
Costes Indirectos (5%)	385,2€
Margen (10%)	770,4€
Beneficio (10%)	770,4€
Antes de impuestos	9.630€
IVA (21%)	2.022,3€
Presupuesto total	11.652,3€

Tabla 2.11 Tabla de costes totales

A comentar con respecto a esta tabla que se han añadido varios costes extra. Uno de ellos y el más importante es el de costes indirectos. Los costes indirectos son gastos extra asociados a un proyecto. Normalmente los costes indirectos suelen estar dedicados al del coste de alquiler de un espacio de trabajo, así como los costes de luz e internet.

Luego he de destacar que se han introducido otros dos márgenes más, que son el margen de errores y el beneficio. El margen de errores garantiza que, en caso de cualquier imprevisto de desarrollo o fallo de planificación, se pueda asumir el coste. El beneficio es lo que se estima generar en la venta del producto.

Finalmente, no se poseen conocimientos excelsos de administración y dirección de empresas, pero por ley, se debe de declarar un IVA de cada producto vendido, por lo que se ha añadido en el cálculo de costes. Esto hace que el presupuesto total del proyecto sea de 11.652,3€.

3. DISEÑO

En esta sección se muestra la estructura y el diseño del prototipo. Para ello, primero se presenta 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 acciones principales:

- 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

El flujo de trabajo consiste principalmente en abrir varios submenús, como se puede ver en la Ilustración 3.1:

- Abrir varios menús de selección de modo de juego o el de configuración
- Introducir el nombre de usuario, que permite usarlo que permite usarlo en las puntuaciones y que guarda los ajustes de sonido.
- Salir del juego

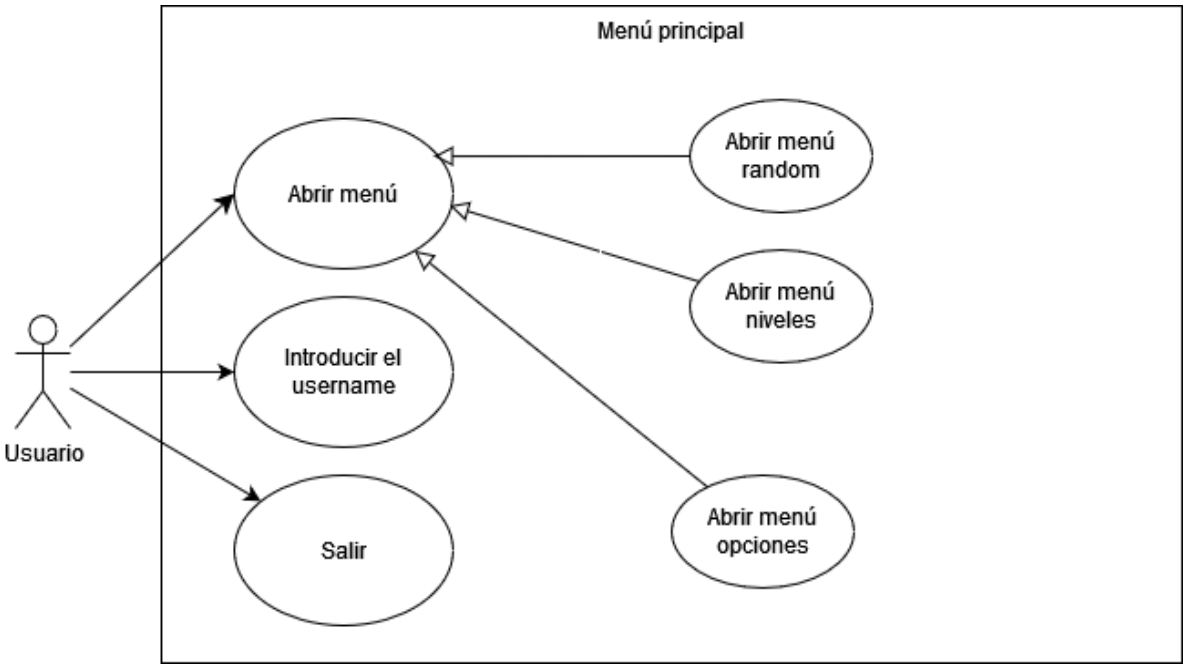

Ilustración 3.1 Diagrama de caso de uso del menú principal

3.1.2. Menú random

El flujo del caso de uso consiste en elegir entre los parámetros que existen y finalmente iniciar el juego, como se puede ver en la Ilustración 3.2. 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.

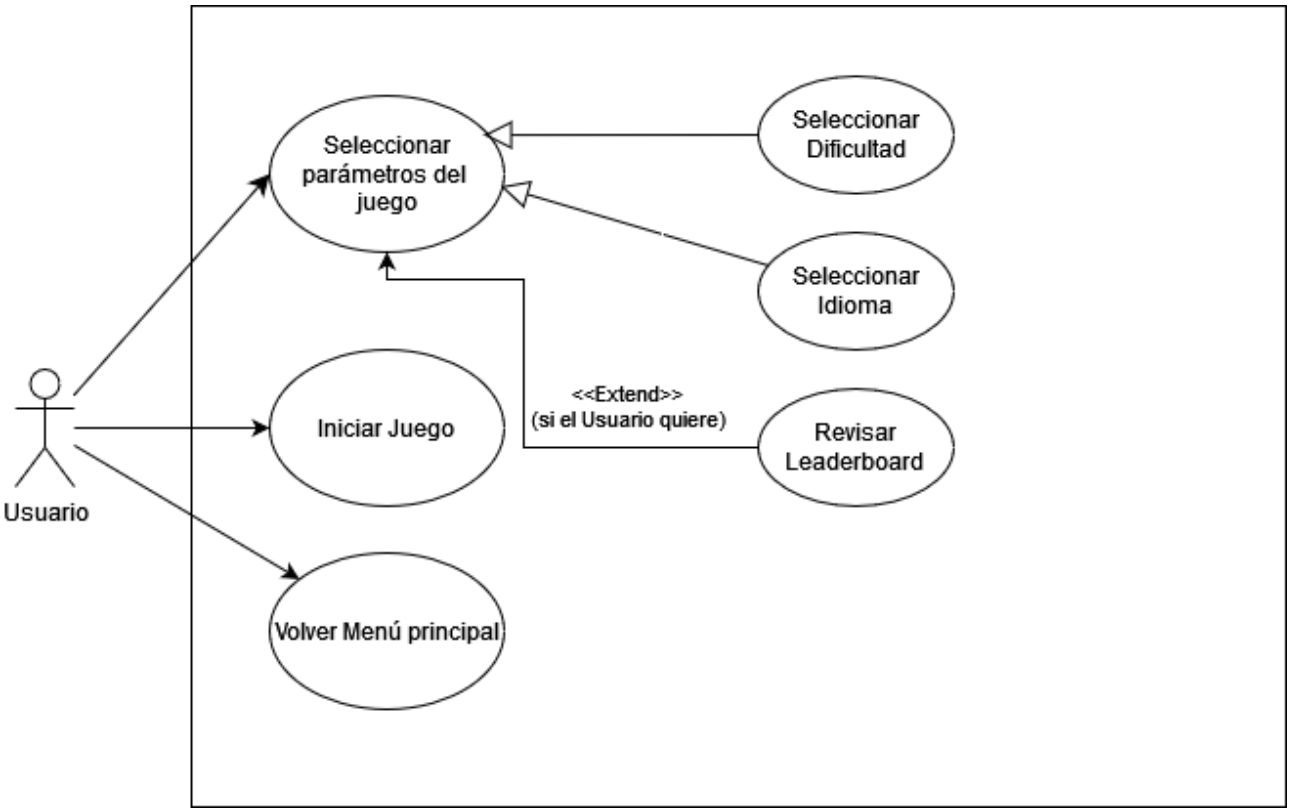

Ilustración 3.2 Diagrama de caso de uso del submenú Random

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, como se puede ver en la Ilustración 3.3:

- Seleccionar Mundo
  - Seleccionar Nivel

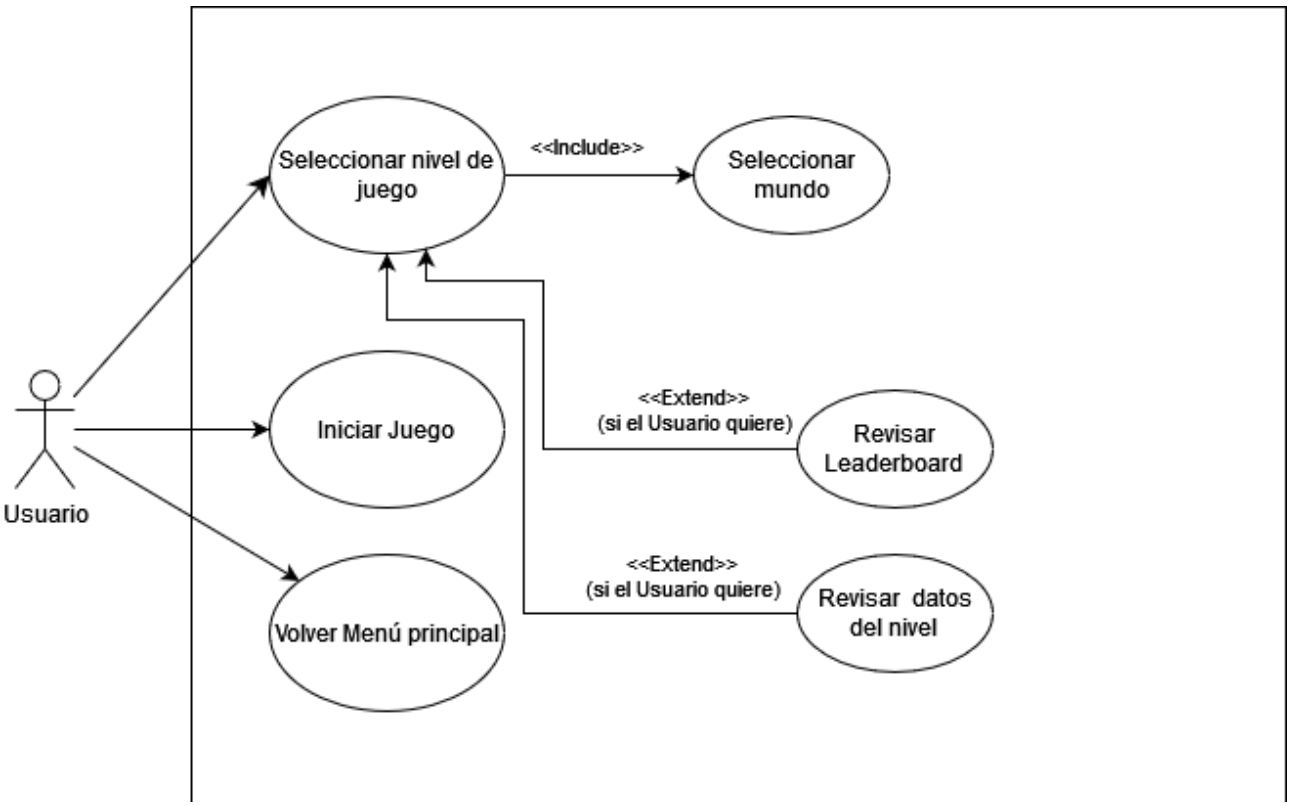

Ilustración 3.3 Diagrama de caso de uso del submenú Niveles

3.1.4. Menú opciones

En este menú se puede modificar los valores de sonido de tres elementos, como se puede ver en la Ilustración 3.4:

- General
  - Música
  - Efectos Especiales

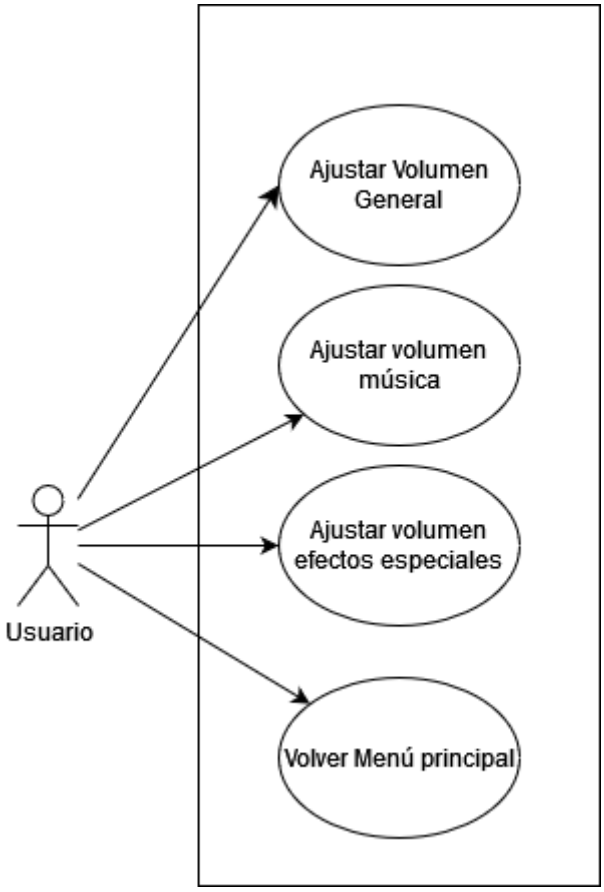

Ilustración 3.4 Diagrama de caso de uso del submenú Ajustes

3.1.5. Pantalla de juego

El flujo de trabajo tiene una particularidad. Pulsar una tecla del teclado tiene más complejidad 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.

Esto se puede ver en la Ilustración 3.5.

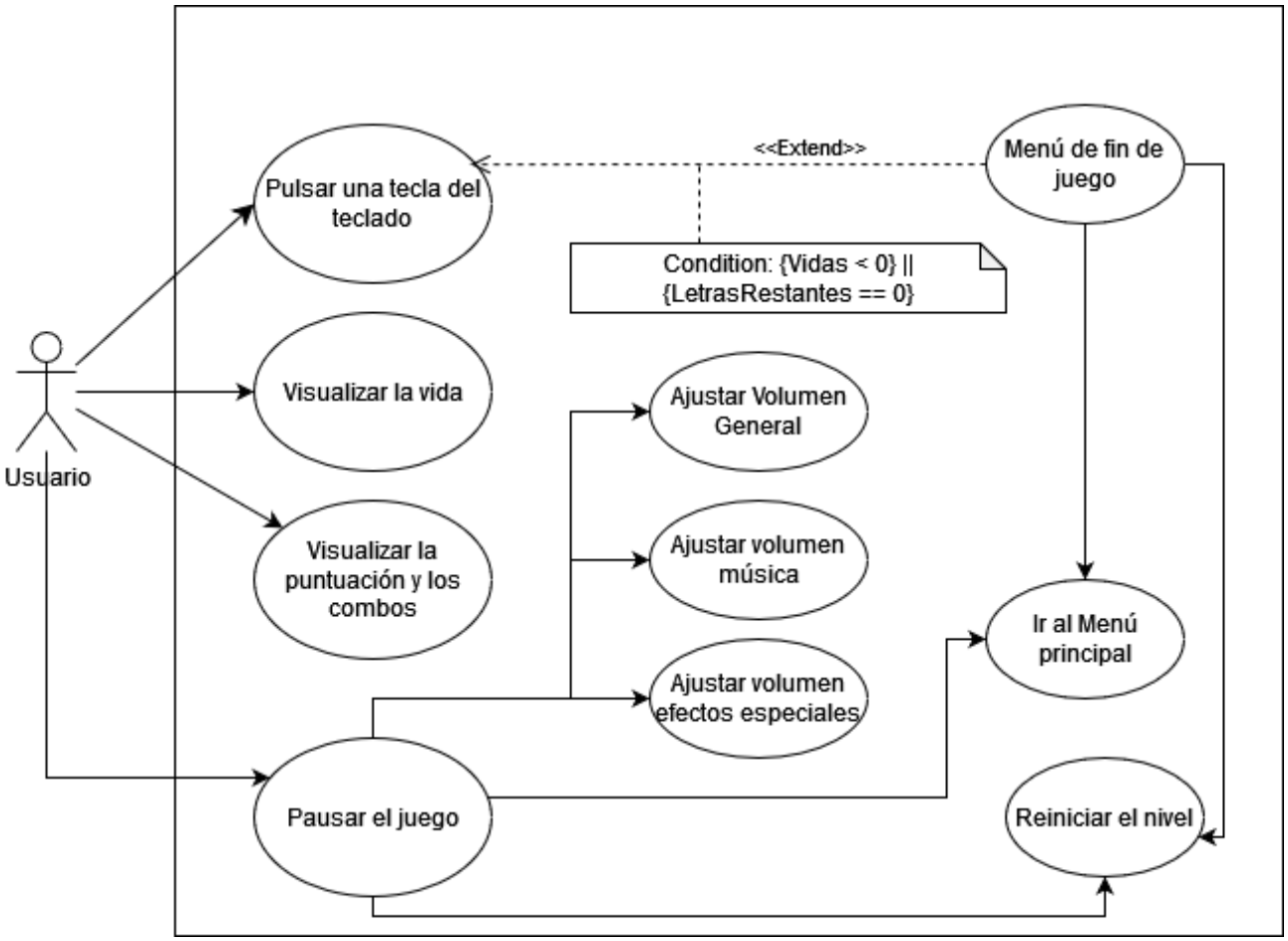

Ilustración 3.5 Diagrama de caso de uso del bucle jugable

3.2. Diagramas 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 Bucle Jugable.

Como se observa en la Ilustración 3.6, 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.

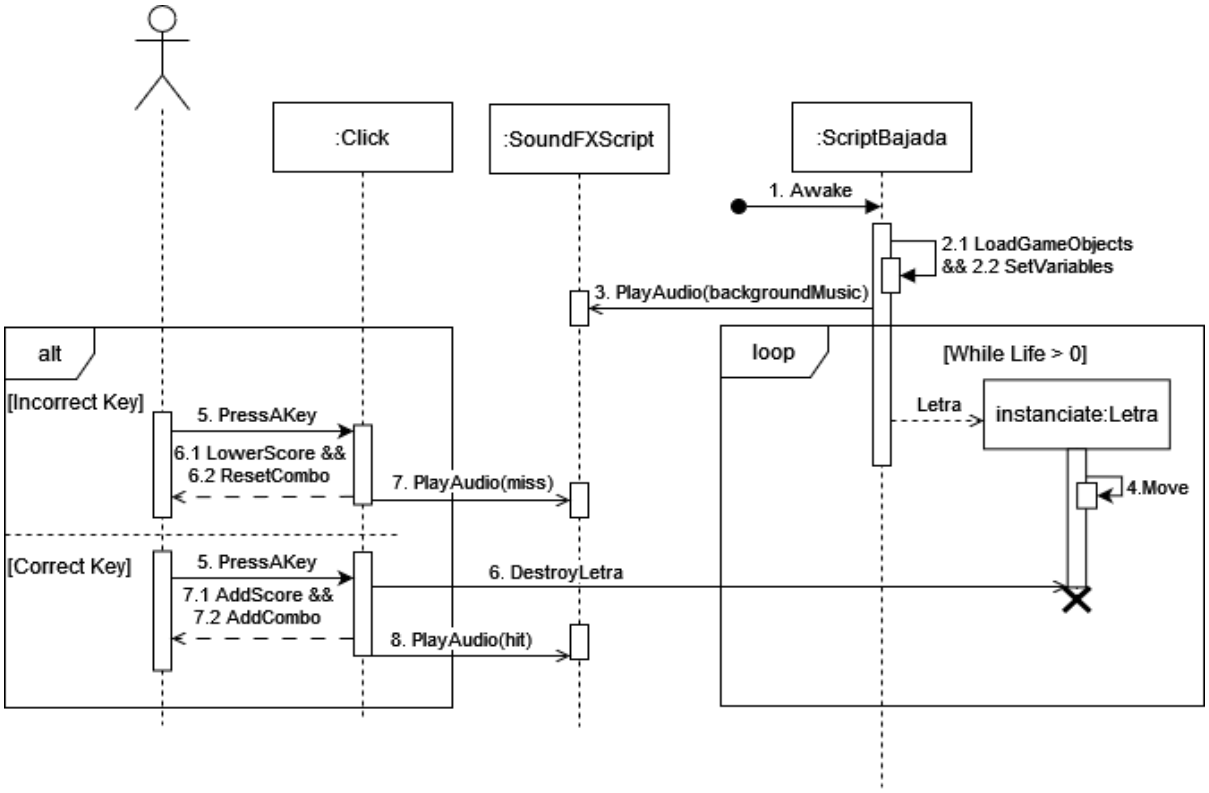

Ilustración 3.6 Diagrama de secuencia del Bucle Jugable

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.

3.3. Diagramas de actividad

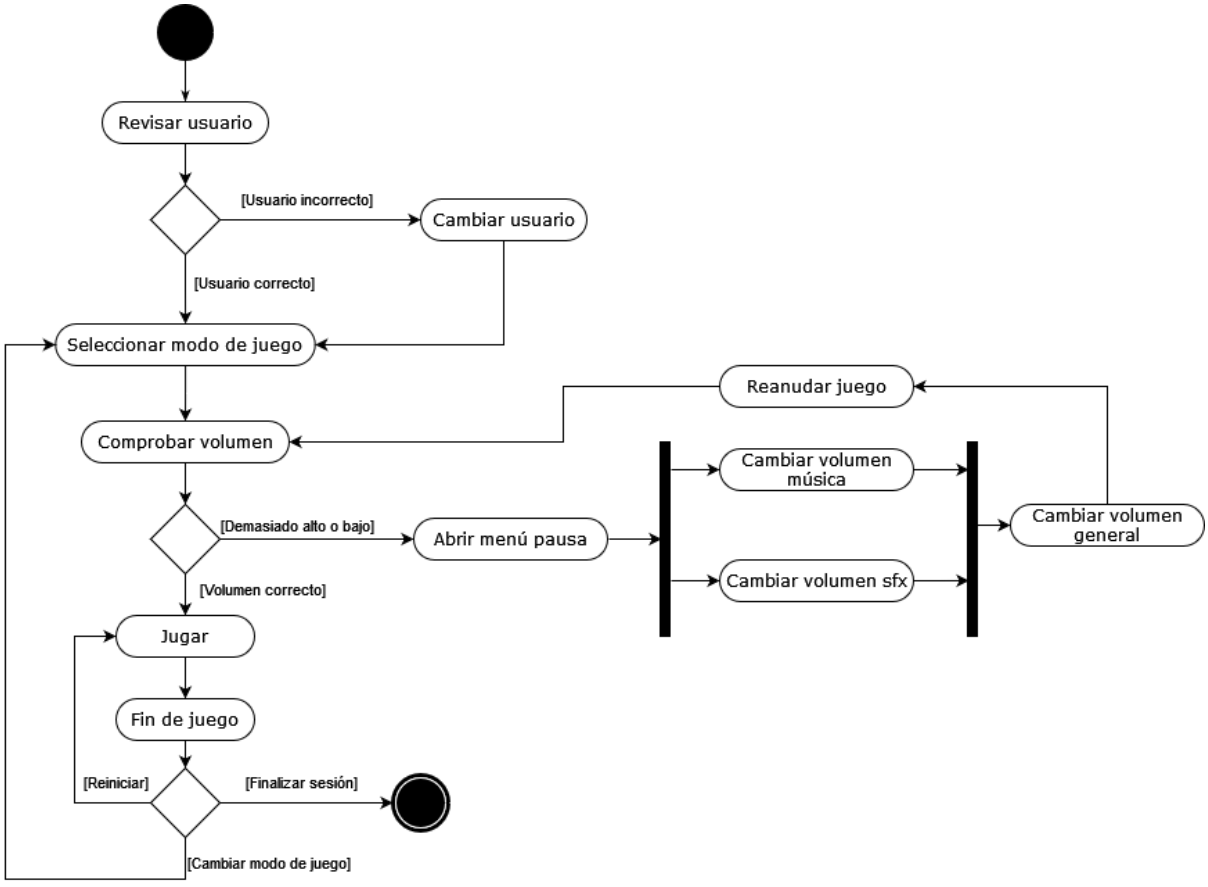

Ilustración 3.7 Diagrama de actividad del flujo normal de juego

Como se ve en la Ilustración 3.7, el diagrama de actividad muestra como un usuario seguiría un flujo que se ha considerado el estándar.

El flujo comienza desde que abre el juego hasta que llega a finalizar la sesión, pasando por las diferentes decisiones que toma el usuario. De esta forma, el usuario

lo primero que hace es comprobar el nombre de usuario. El usuario se guarda entre sesiones, pero si es su primera ejecución o si tiene un ordenador compartido es necesario cambiarlo. Si bien aquí podría haberse introducido una acción de cambiar los ajustes, no se vio necesario. Muchos usuarios priorizan jugar a cambiar los ajustes previamente y simplemente los cambian si encuentran algún problema. Se ha optado por seguir este flujo.

Luego selecciona un modo de juego. Si bien aquí se podría haber introducido un nodo de decisión, es poco relevante ya que en ambos casos llegaría a la escena jugable.

Aquí es cuando se ha optado por introducir los cambios de los ajustes a través de la acción de Comprobar volumen y su nodo de decisión. Luego el usuario juega normalmente hasta que llega a la pantalla de fin de juego, independientemente del resultado de la partida.

Aquí puede optar por repetir el nivel si ha perdido o quiere una mayor puntuación, cambiar de nivel o modo de juego que consiste en volver al menú principal y finalizar la sesión.

3.4. Storyboard

A continuación, se muestran los storyboards de las diferentes ventanas de juego.

3.2.1. Menú principal

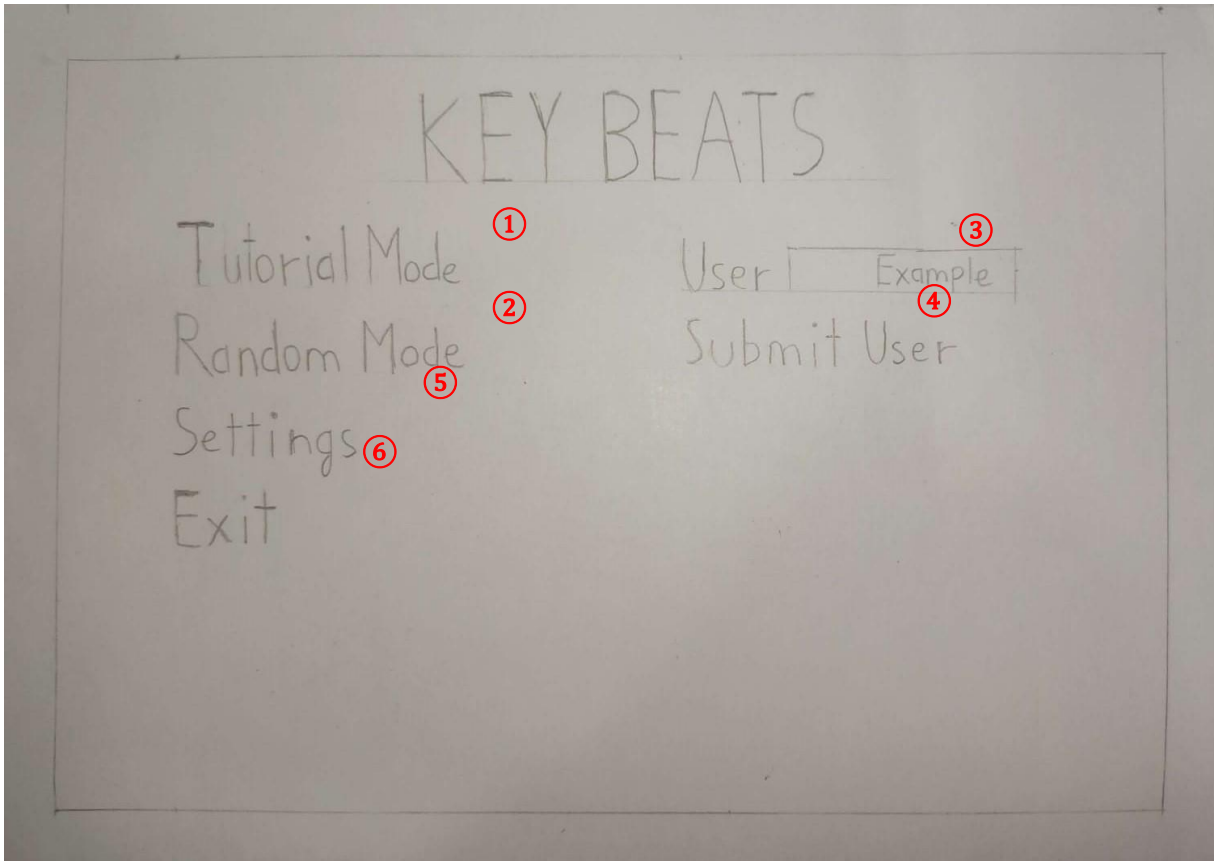

Ilustración 3.8 Storyboard menú principal

Aquí se muestra el menú que aparece nada más ejecutar el juego. Como se puede ver en la Ilustración 3.8, 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

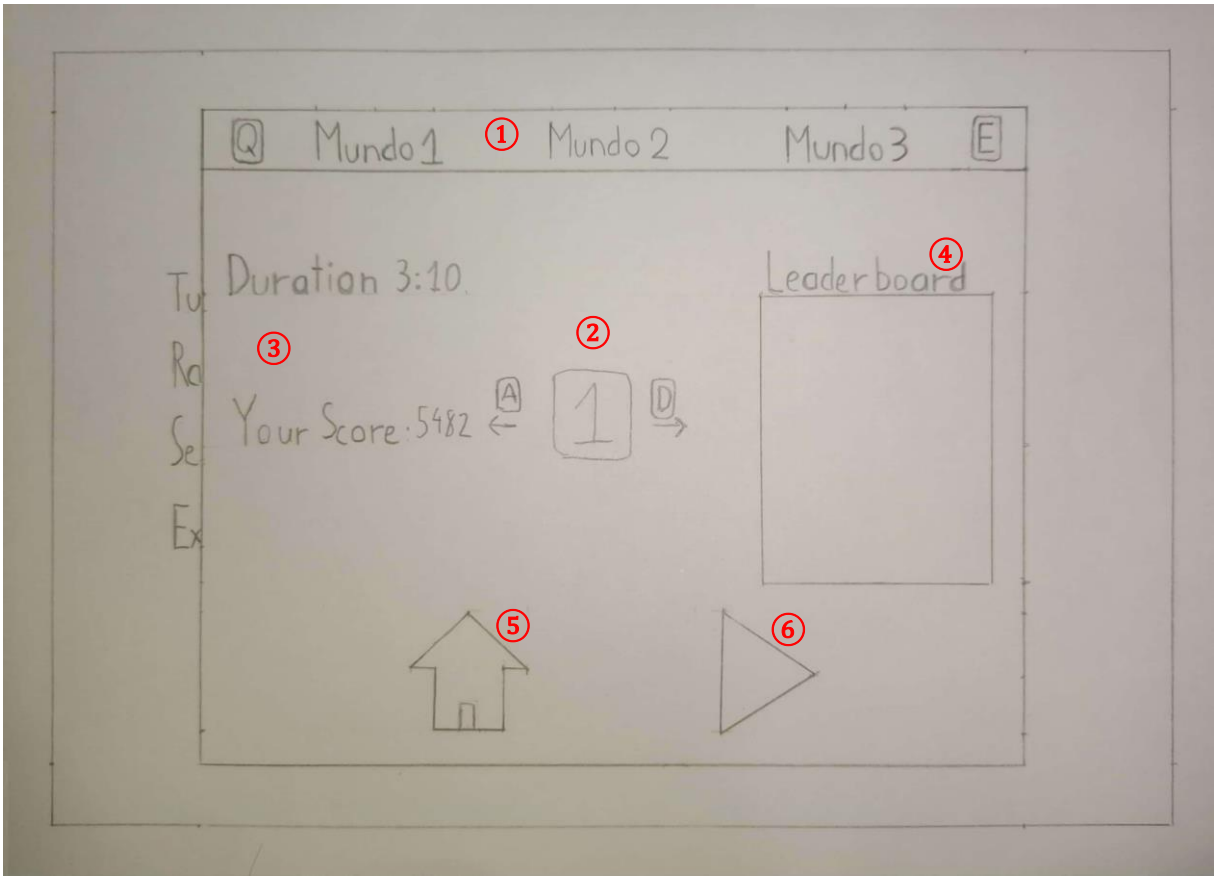

Ilustración 3.9 Storyboard submenú selector de niveles

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, como se puede ver en la Ilustración 3.9.

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

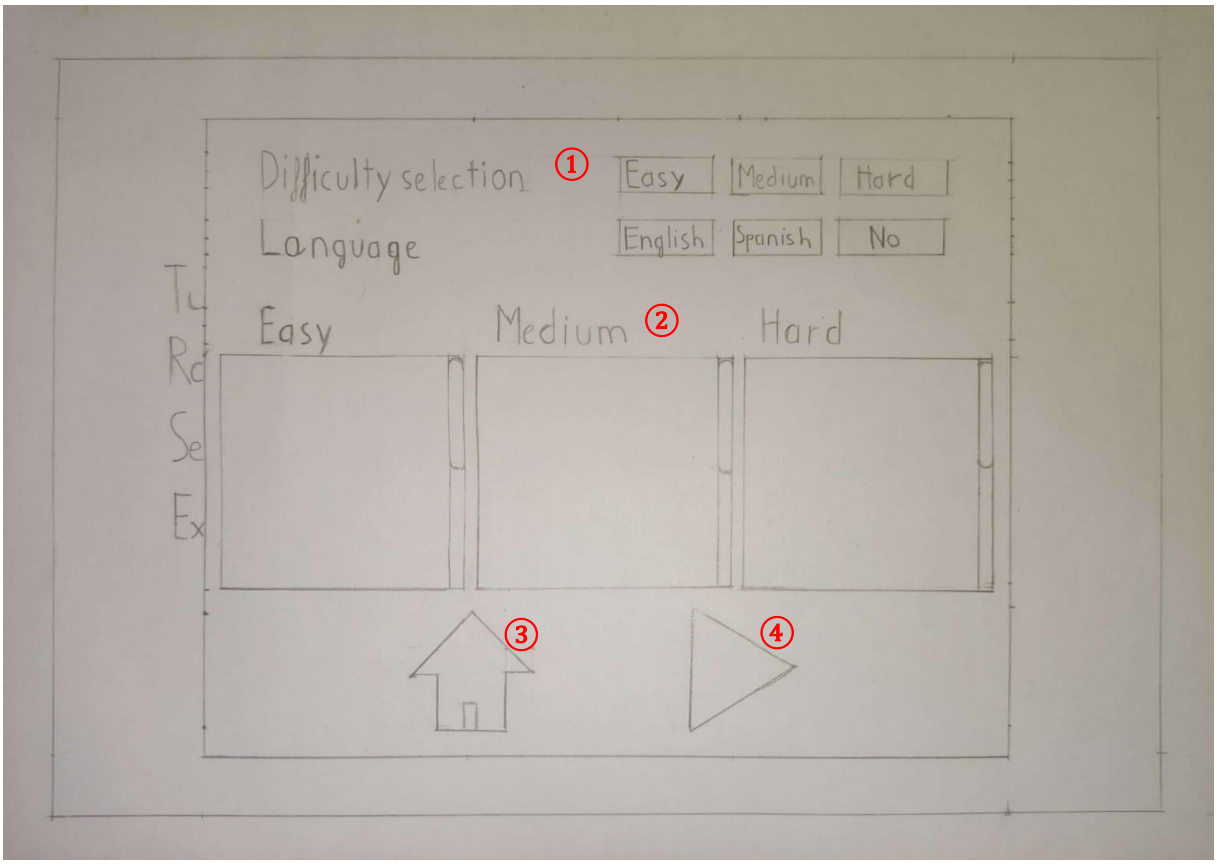

Ilustración 3.10 Storyboard submenú modo random con o sin diccionario

Como se puede ver en la Ilustración 3.10, 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

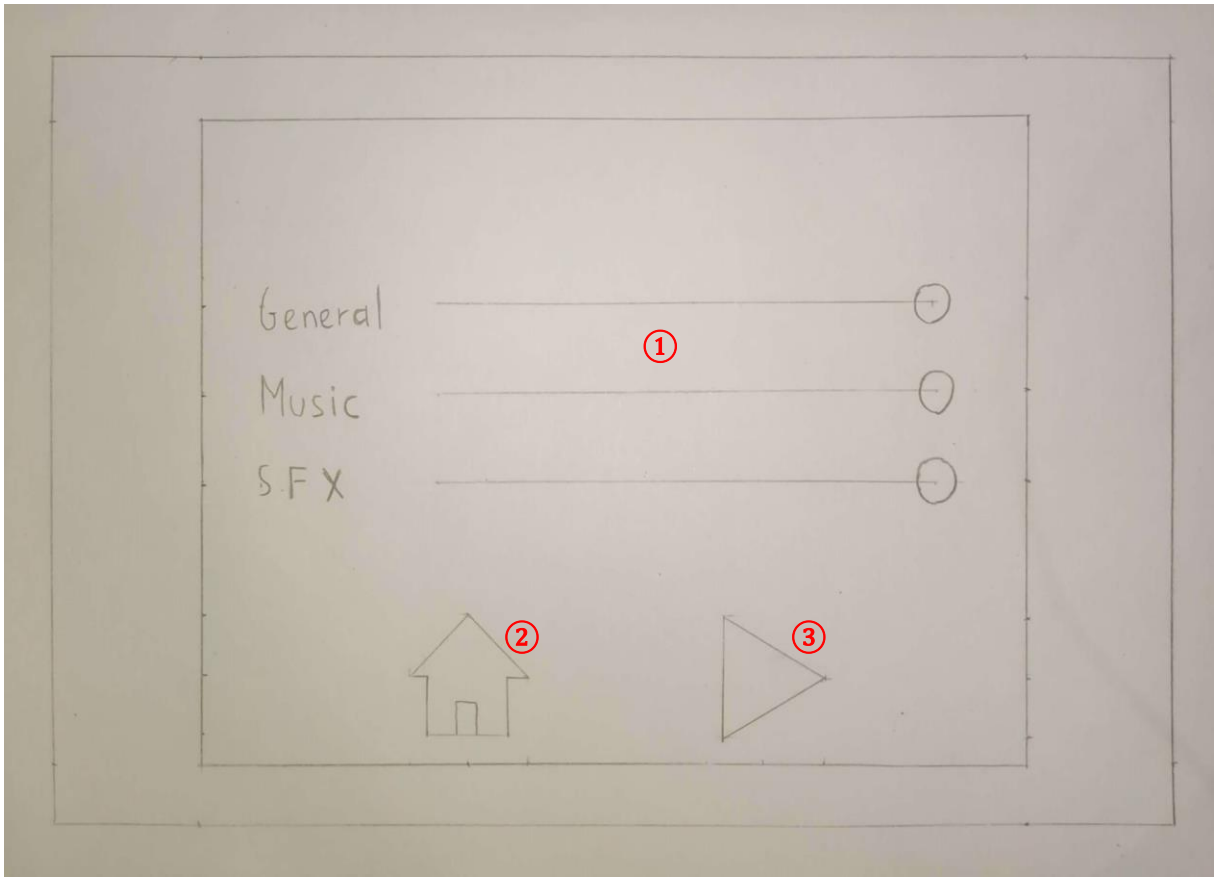

Ilustración 3.11 Storyboard submenú ajustes

Como se puede ver en la Ilustración 3.11, 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 (2) o de resumir el juego (3).

3.2.5. Bucle Jugable

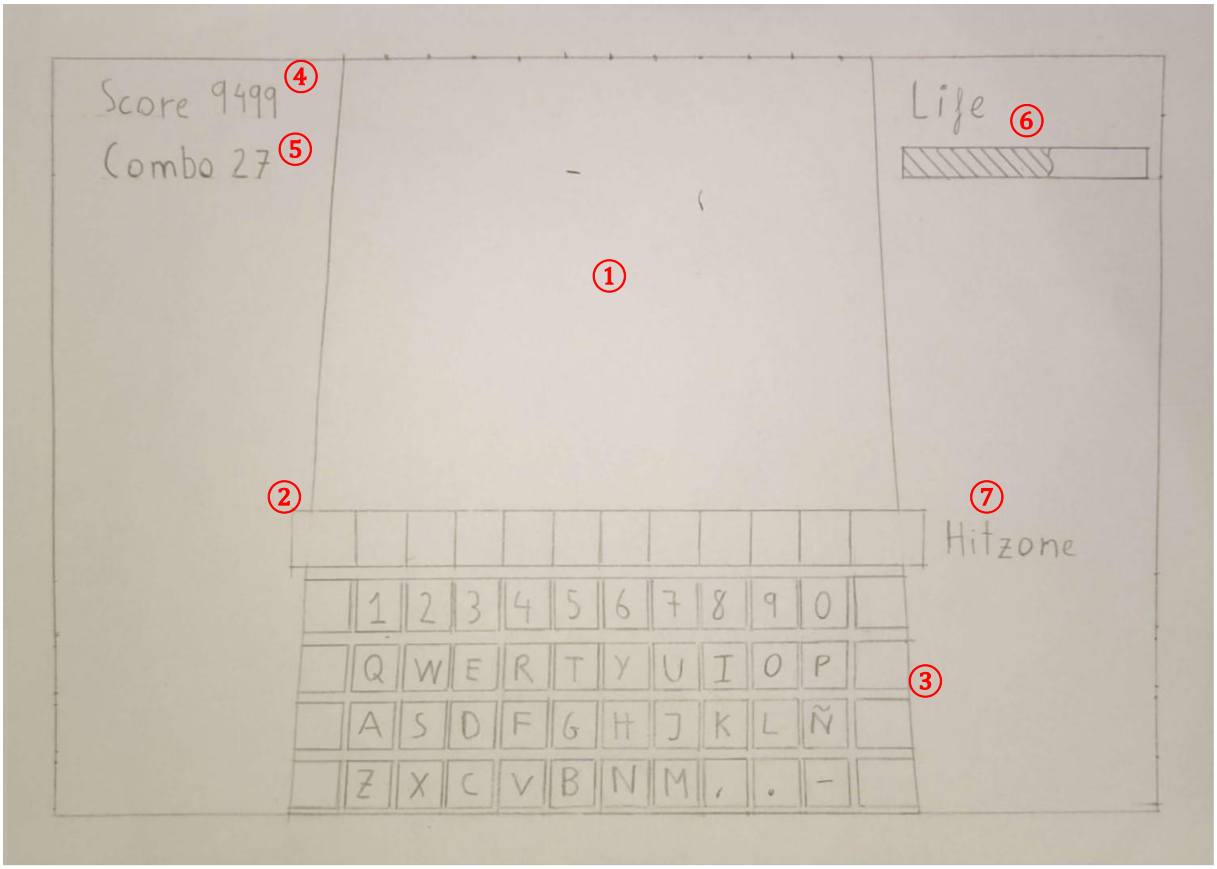

Ilustración 3.12 Storyboard de la escena jugable

Como se puede ver en la Ilustración 3.12, 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 la zona de juego (1), donde las diferentes letras van cayendo 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

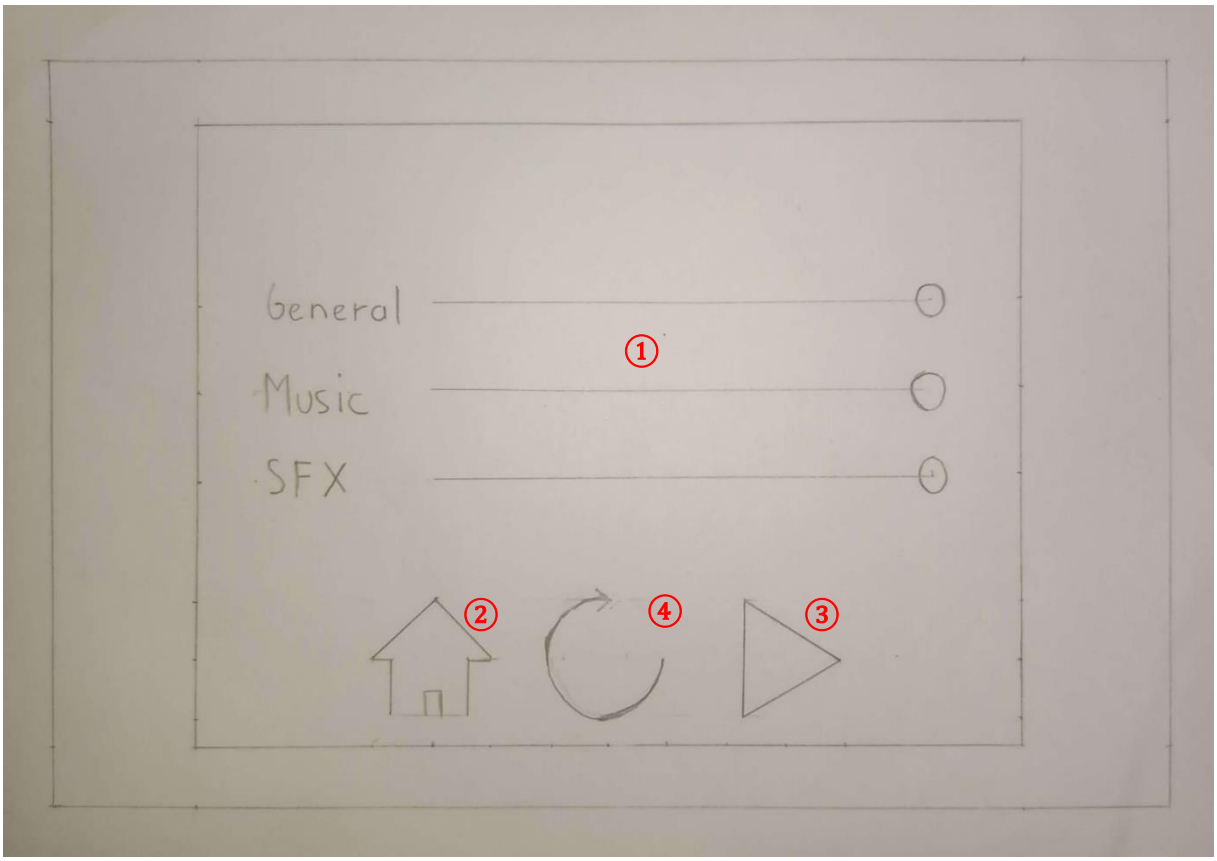

Ilustración 3.13 Storyboard menú pausa

Como se puede ver en la Ilustración 3.13, 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 (2) o de resumir el juego (3) o de reiniciar el nivel (4).

3.2.7. Fin de Juego

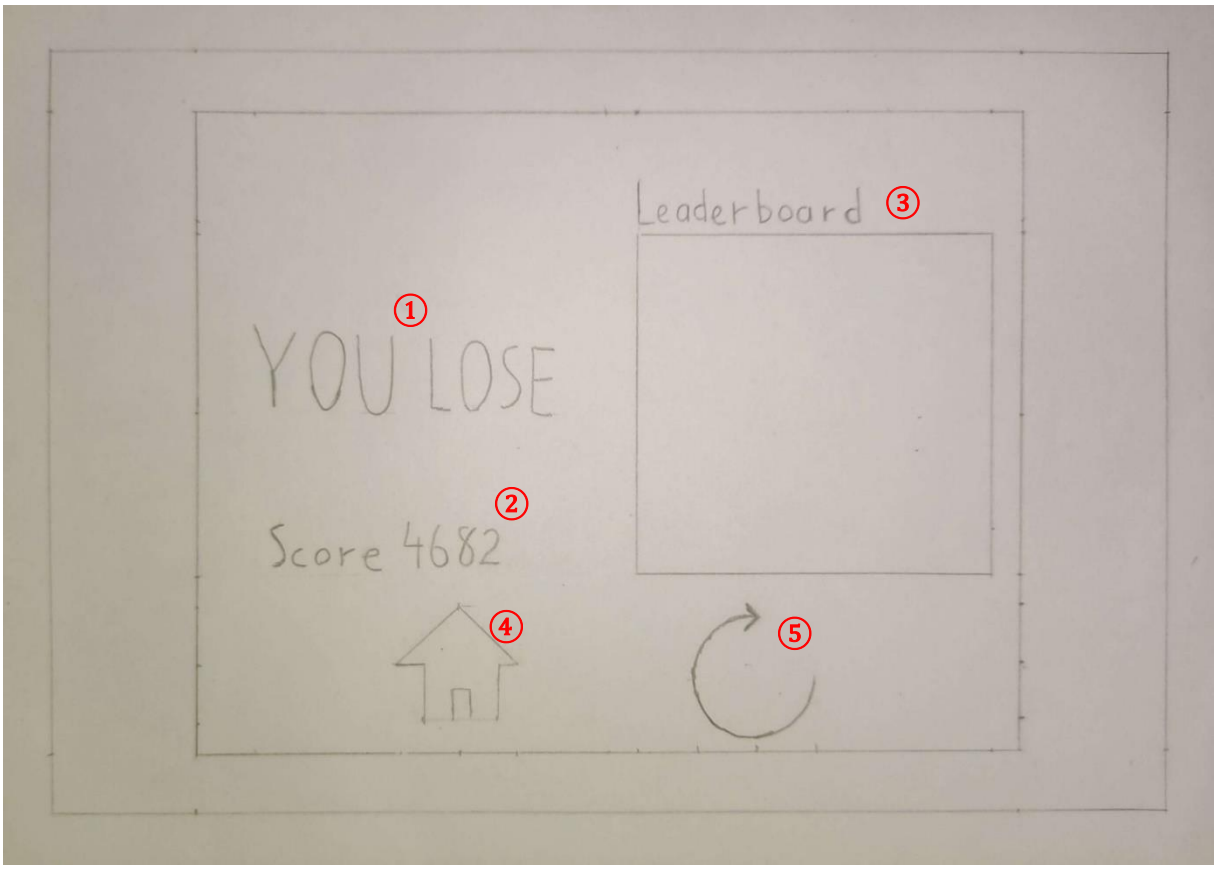

Ilustración 3.14 Storyboard menú fin de partida con condición de derrota

Como se puede ver en la Ilustración 3.14, 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 marcador (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 principal (4) o un botón a la derecha para reiniciar el nivel (5).

3.5. Código de colores

A la hora de diseñar una página web, los colores juegan un papel vital en la experiencia de usuario. Partiendo de que, unos colores adecuados sirven para hacer una navegación más intuitiva, utilizable. Además, 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 (Bianchi, s.f.). Aparte de tener ambas opciones previamente mencionadas, Coolors posee un analizador de colores y un inspector de contraste. Estas 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, degradando la intensidad de estos, como se puede ver en la Ilustración 3.15.

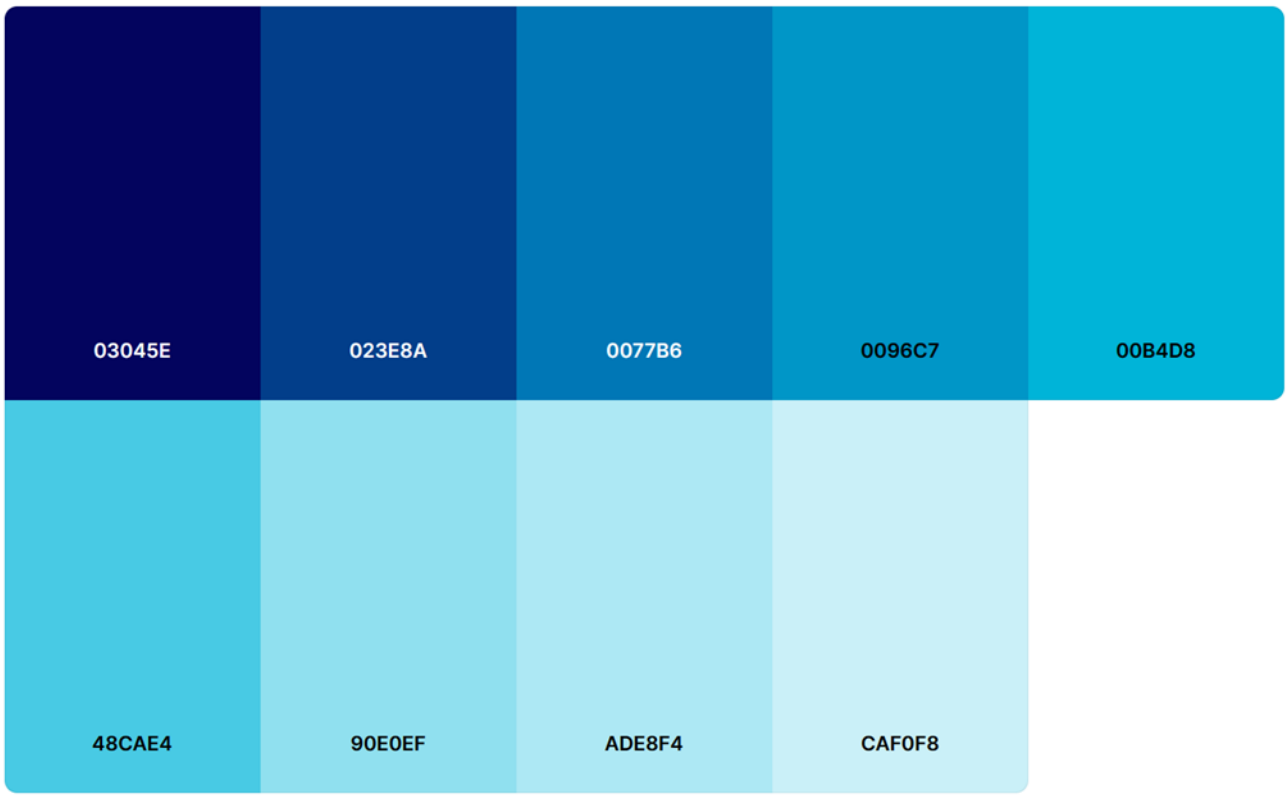

Ilustración 3.15 Paleta de colores monocromática

Se puede acceder a esta paleta a través del siguiente enlace:

https://coolors.co/palette/03045e-023e8a-0077b6-0096c7-00b4d8-48cae4-90e0ef-ade8f4-caf0f8

Adicionalmente, se han usado un par de colores para generar contraste entre esta base, permitiendo identificar elementos importantes de forma intuitiva. Ambos colores son el E63946 y 49A078.

Finalmente, usando otra herramienta web como es Coloring for Colorblindness (Nichols, s.f.), podemos comprobar cómo se ven realmente la paleta completa con los distintos tipos de daltonismo en la Ilustración 3.16.

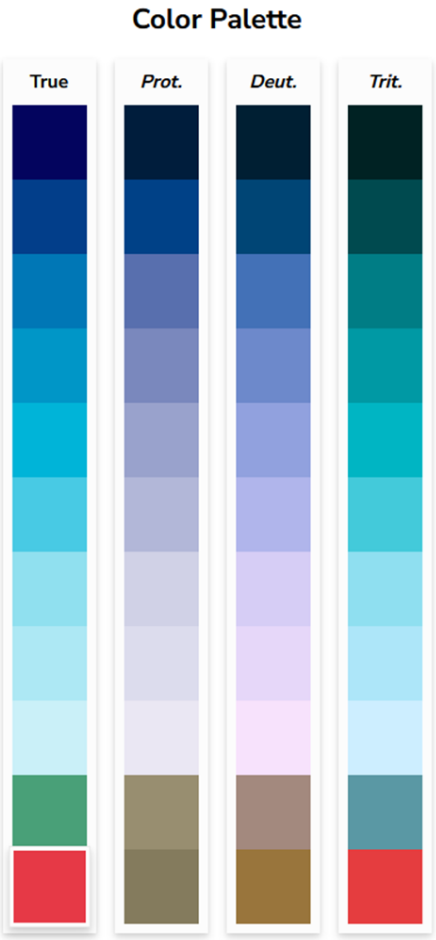

Ilustración 3.16 Comprobación de colores para gente con daltonismo

4. DESARROLLO

En este capítulo se expondrán todo el trabajo realizado paso a paso. Como bien se describió en el apartado 1 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, componiéndose las tareas A, B y C. 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.

Asimismo, se han repasado todos los diagramas explicados en la asignatura de Fundamentos de Ingeniería del software y se han valorado los diferentes diagramas que pueden implementarse. De esta forma, se ha decidido implementar varios diagramas de casos de uso que explican algunas escenas y los distintos modos de juego. Finalmente, se han diseñado los diferentes menús del juego, así como plantear la estructura de la interfaz de usuario utilizada.

4.2. Iteración 1

Esta iteración se compone de la tarea D. 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. Estos han sido inclinados 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.

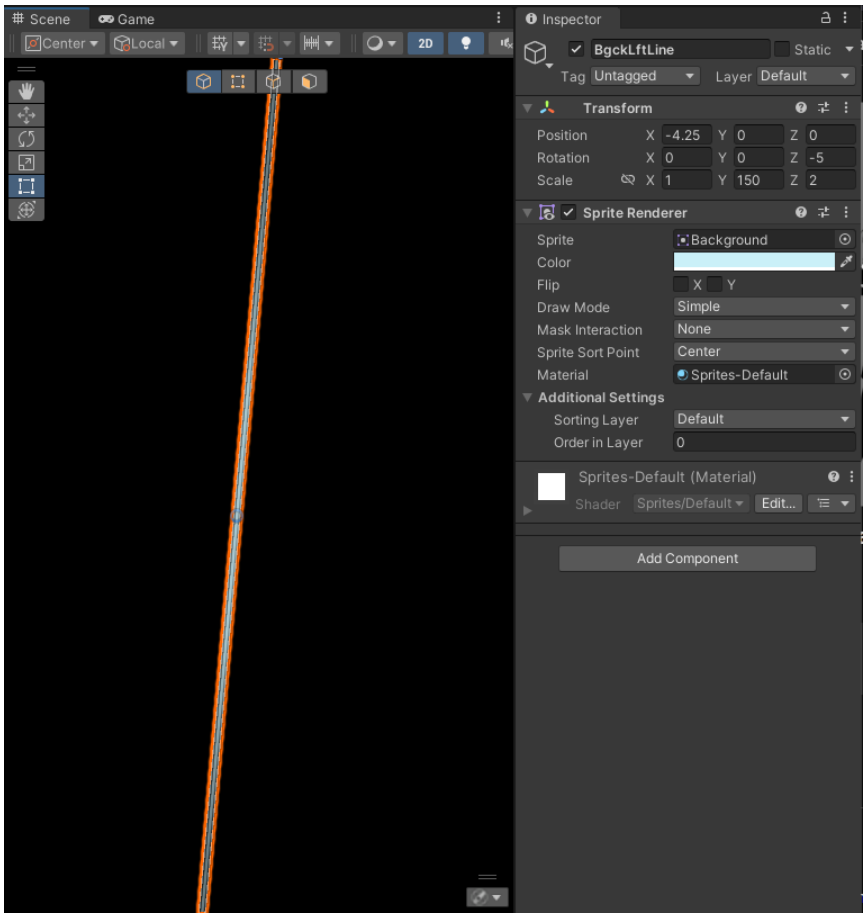

Ilustración 4.1 Ajuste y configuración del borde

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 con un efecto de neón y se han divido las letras por columnas, según se muestra en la Tabla 4.1.

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 4.1 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 se muestre como la Ilustración 4.2.

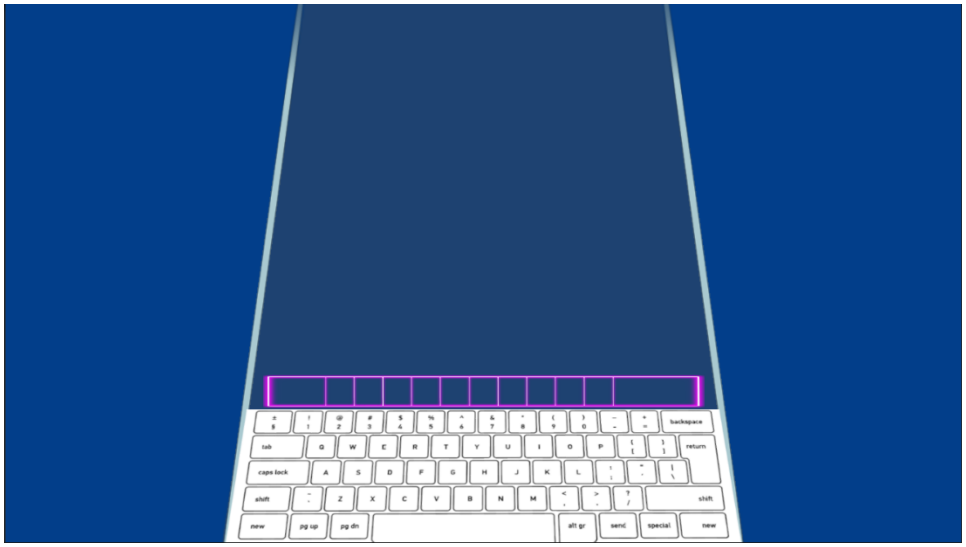

Ilustración 4.2 Disposición final de la escena

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. Para ello se recortó cada letra individualmente. El ejemplo de uno de estos Sprites se puede apreciar en la Ilustración 4.3.

Ilustración 4.3 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 de cada una de ellas, indicando la posición X para que la letra coincida con el hueco de la HitZone definido previamente.

Para el movimiento se ha dispuesto de un script singleton asociado a cada uno de los Prefabs. Un singleton es un patrón de diseño que garantiza que solo exista un elemento de esta clase y, además, permite crearlo en cualquier momento de su ejecución. Este singleton tiene un valor único que guarda el valor de velocidad de los objetos asociados.

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. Primero realiza una función de carga de todos los objetos. 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.

A la hora de generar los objetos, es un bucle infinito que genera letras cada cierto intervalo. La posición X viene determinada por el Prefab del objeto. La posición Y es generada gracias a un GameObject conocido como *topScreen,* que se sitúa en la parte superior de la pantalla. No se ha guardado la posición Y en el prefab porque es dependiente de la resolución de la pantalla.

Este método se puede observar en la Ilustración 4.4.

```
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 4.5 Método de generación de letras

Toda la funcionalidad relacionada con el teclado está llevada a cabo por otro script conocido como *ClickScript,* asociado al colisionador de la HitZone. La configuración de este colisionador se puede observar en la Ilustración 4.5.

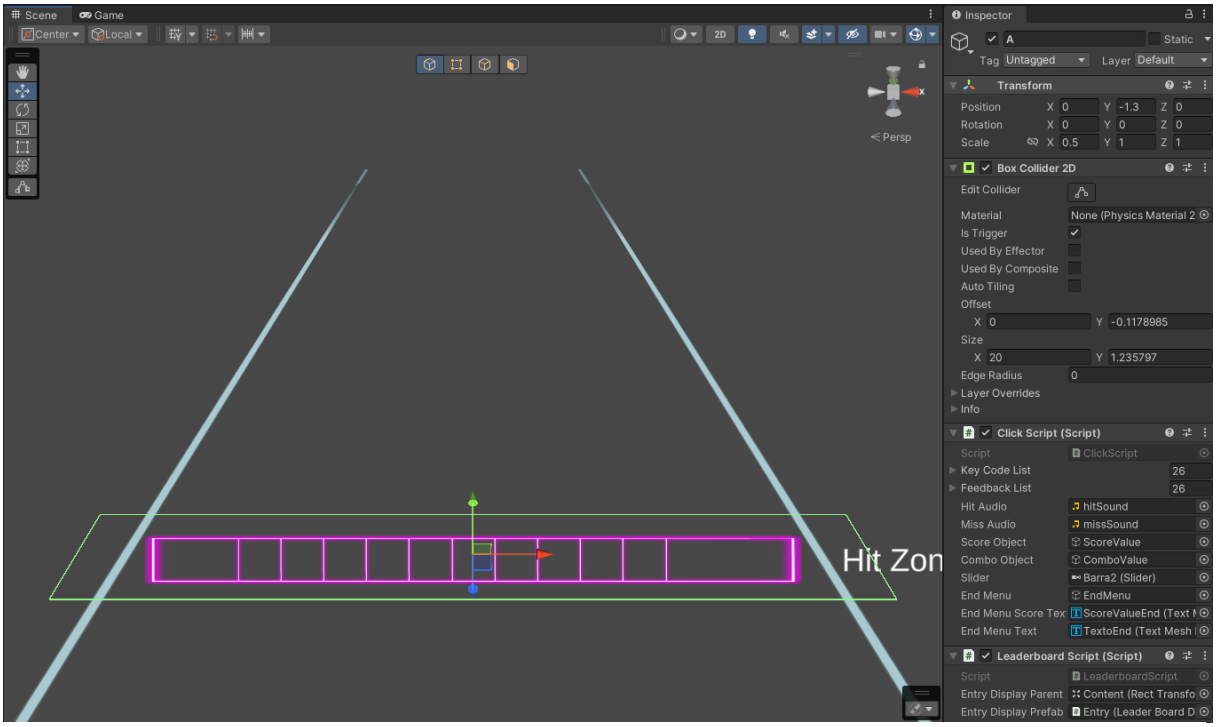

Ilustración 4.4 Parámetros del colisionador de la HitZone, así como del script ClickScript

Cuando una letra esté en dentro del colisionador se habilitará la eliminación de la letra correspondiente gracias a la función OnTriggerEnter() y se introducirá en la lista de objetos colisionados *collisedObjectColliderList.*

Cuando se pulse una tecla se comprobará si hay algún Collider2D en la lista collisedObjectColliderList. Si es así, se recorrerá la lista de objetos colisionados y se comprobará de que la letra coincida. En caso de que coincida la letra, se destruirá y se eliminará de la lista de objetos colisionados. Esta funcionalidad se puede observar en la Ilustración 4.6.

```
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 4.6 Funcionalidad de comprobación de las letras con la pulsación de teclado

4.3. Iteración 2

Esta segunda iteración las tareas a realizar son la E, F, G y H. 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.

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

De esta forma, el método de generación de letras en *LetterManager* se puede observar en la Ilustración 4.7.

```
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 4.7 Método de generación de letras en los niveles

El método lo único que hace es sacar la *Key* del *LevelKey* y buscar el recurso en *dicctionaryList*. Luego genera la letra y se espera el tiempo indicado en *TimeToWait,* repitiendo el proceso hasta que la lista está vacía.

Ahora pues, para guardar el nivel de forma persistente fácilmente, se decidió usar un archivo csv. Un archivo csv guarda datos en forma de tabla y divide varios valores por un delimitador, por lo que es muy sencillo obtener valores de forma eficiente. De esta forma, solo habría 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];[Retraso primera letra]
[Letra];[Tiempo de aparición];[Tiempo de espera]

Por tanto, un ejemplo sería la Ilustración 4.8.

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

Ilustración 4.8 Ejemplo de estructura de un nivel

En esta, se puede observar que la primera línea tiene los datos mencionados previamente.

- **Ruta del archivo de música**: Está escrita en formato relativo, por tanto, es dependiente del directorio donde se ejecuta. Normalmente es un archivo .mp3, pero podría ser cualquier formato de música.
- **Velocidad**: Parámetro usado en el cálculo de velocidad. Normalmente es un multiplicador de una velocidad base.
- **Tiempo total**: Metadato usado en el selector de niveles, no tiene ninguna funcionalidad jugable.
- **Retraso primera letra**: Usado en la generación de la primera letra, indica el tiempo en segundos que tiene que esperar el programa para generar la letra.

Las siguientes líneas tienen una correspondencia con los atributos de la clase *LevelKey,* previamente mencionada.

De esta forma, podremos en el Awake() llamar a la función ReadLevel(), abrir el archivo csv. Luego se lee la primera línea y se establecen los parámetros del nivel.

```
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 4.9 Método de lectura de un nivel

Finalmente, se va leyendo línea a línea el archivo, creando *LevelKey* en el proceso, e introduciéndolos en la lista *levelList*, como se muestra en la Ilustración 4.9.

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. Para crear un AudioClip se ha de indicar un archivo de audio en un formato compatible. Para este prototipo se ha utilizado el formato .mp3, pero Unity permite otros formatos como .aif, .wav o .ogg.

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 está 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, *SoundFXScript*. 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, dividido en 3 tipos: general, music y sfx.

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

Para garantizar que se han eliminado los elementos, se modificó la funcionalidad de *ClickScript.* En primera instancia se planteó usar OnTriggerExit2D() para eliminar los objetos, pero existía un error. Unity tiene una particularidad, que

cuando se destruye un objeto detecta que ha salido del colisionador ejecutando OnTriggerExit2D() y dando un error de null pointer al intentar volver a eliminar el objeto.

Para solucionar esto, se implementó una bandera conocida como *scored.* Si un objeto ha sido eliminado como consecuencia de un acierto, la función OnTriggerExit2D() no destruye el objeto. En caso contrario, el objeto es eliminado al salir del colisionador.

4.3.4. Alfabetos

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 4.2 Porcentaje de letras según el idioma

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, por tanto, 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). El porcentaje usado está indicado en la Tabla 4.2.

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. La clase *KeyRandom* se encarga de gestionar toda esta funcionalidad. Tiene los siguientes atributos:

- **Key**: La letra a guardar.
- **Percentaje**: El porcentaje de uso de la letra en el alfabeto del que procede
- **Cumulative**: El porcentaje del uso de la letra en el alfabeto más la suma de los porcentajes de las letras que le preceden.

Esta clase tiene una función propia que actúa como constructor como se ve en la Ilustración 4.10. Esta función calcula primero el porcentaje acumulado de la letra y la añade a una lista.

```
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 4.10 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 genera un número aleatorio con *sumTotalPercentageList.* Luego realiza una búsqueda binaria, encuentra el índice correcto de la lista y genera la letra.

4.4. Iteración 3

En esta tercera iteración las tareas a realizar son la I, J, K y L. 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 subapartado 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. Estos anchors se han probado para mantener una relación de aspecto de 16:9. El diseño final de la interfaz se puede ver en la Ilustración 4.11.

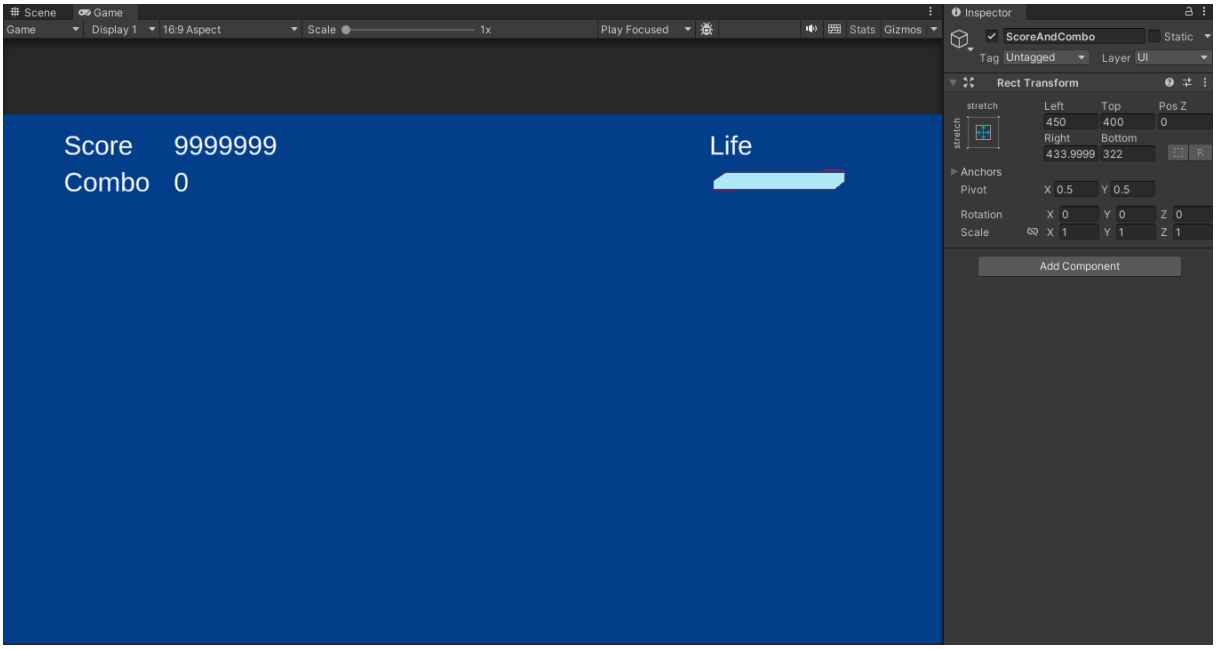

Ilustración 4.11 Interfaz de la escena GameZone

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. Esto se ha logrado a través de PlayerPrefs. PlayerPrefs permite guardar varios datos y variables de forma nativa en Unity para distintos usuarios y es consistente entre sesiones. El diseño final de la interfaz se puede ver en la Ilustración 4.12.

Toda la gestión de todos los botones del menú principal y de la gestión del cambio de usuario es gestionado por *MainMenuScript.*

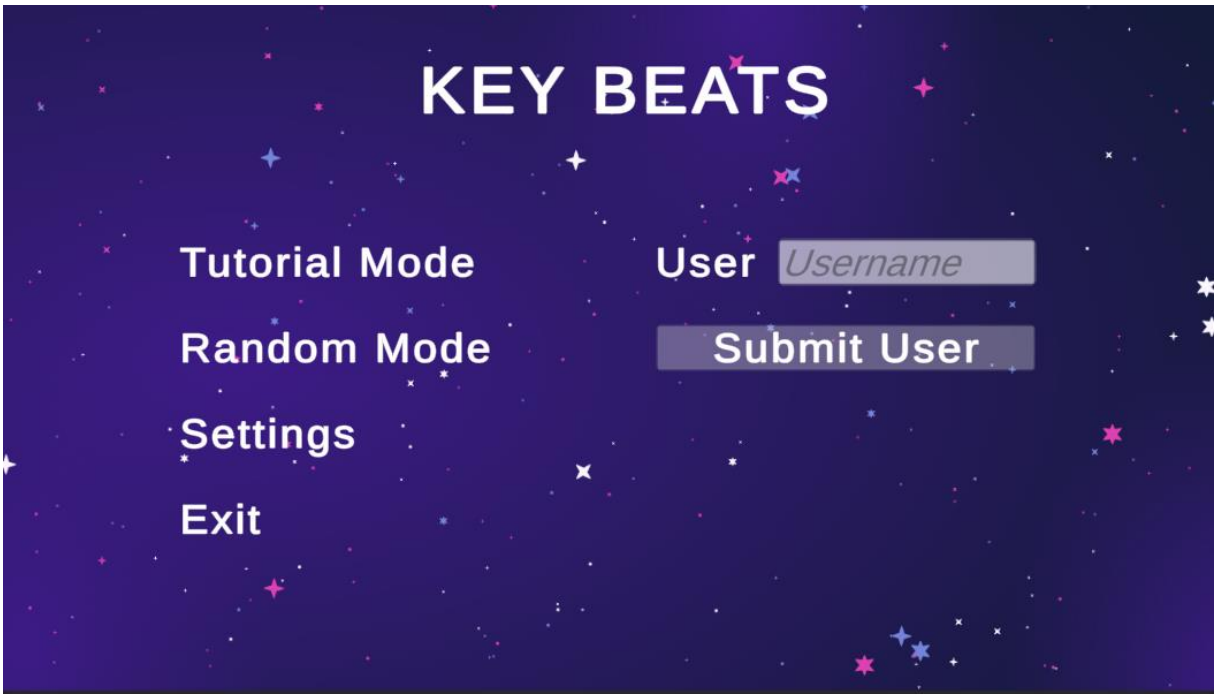

Ilustración 4.12 Interfaz de la escena MainMenu

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 controles deslizantes y varios iconos al fondo, ejerciendo como botones, como se puede observar en la Ilustración 4.13.

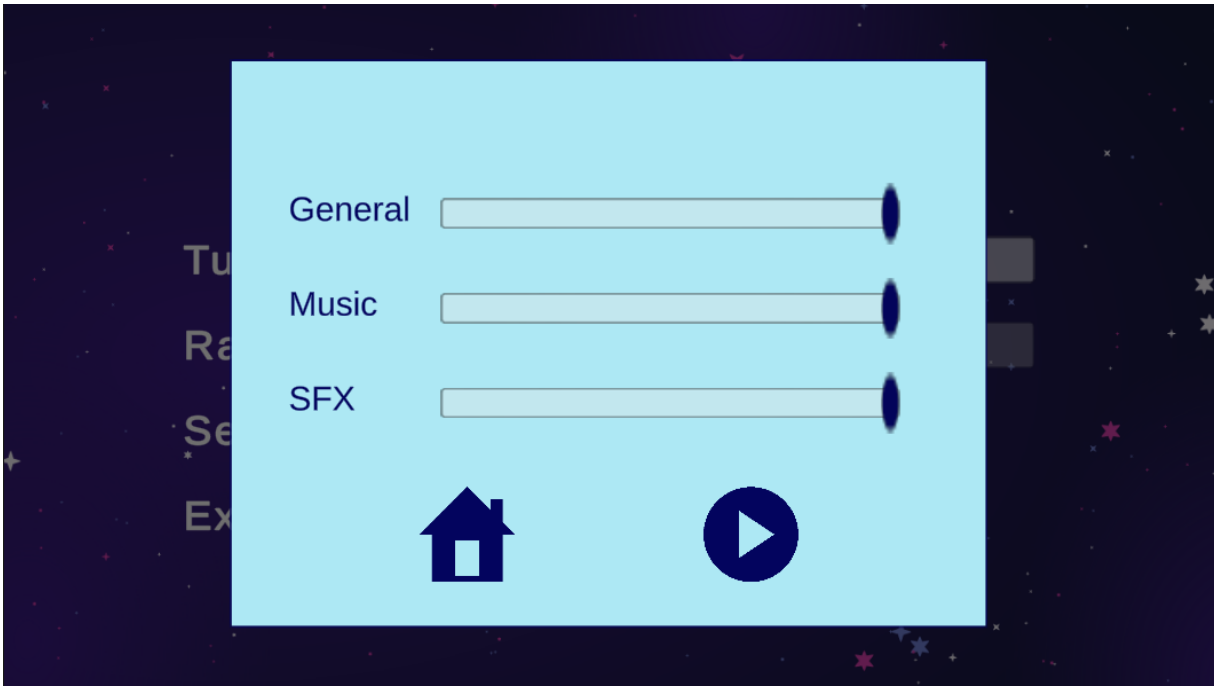

Ilustración 4.13 Interfaz del submenú de ajustes

Para la funcionalidad de los controles deslizantes se ha tenido en cuenta un script llamado *SoundScript,* que usa también PlayerPrefs. Lo primero que hace este script es que cuando se carga la escena, comprueba el usuario y recupera los valores de volumen guardados previamente. Luego establece los controles deslizantes de cada volumen a la posición adecuada.

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 control deslizante y son llamadas en *SetVolumeFromSlider(type).* La configuración del control deslizante se puede observar en la Ilustración 4.14.

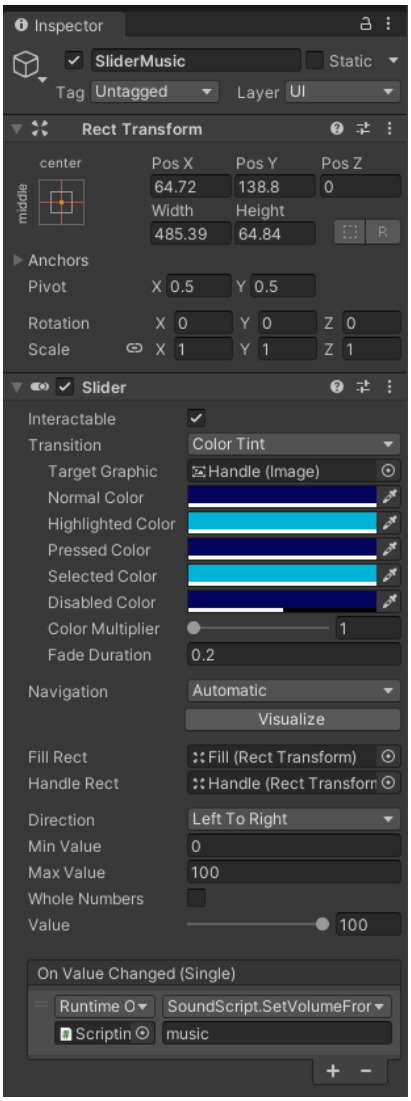

Ilustración 4.14 Valores del control deslizante que gestiona la música

Luego, en la parte inferior vemos dos botones. Ambos tienen la misma funcionalidad, guardar los cambios y volver al menú principal, pero es para dar la sensación al jugador de que se han guardado los cambios. El jugador podría malentender la situación y pensar que, si se pulsa el botón de volver al menú principal, no se guardan los cambios.

Finalmente, el menú de pausa que se observa en la Ilustración 4.15 es exactamente igual al menú de configuración, simplemente que tiene una función de reinicio de nivel, volviendo a cargar la escena con los parámetros con la cual fue ejecutada.

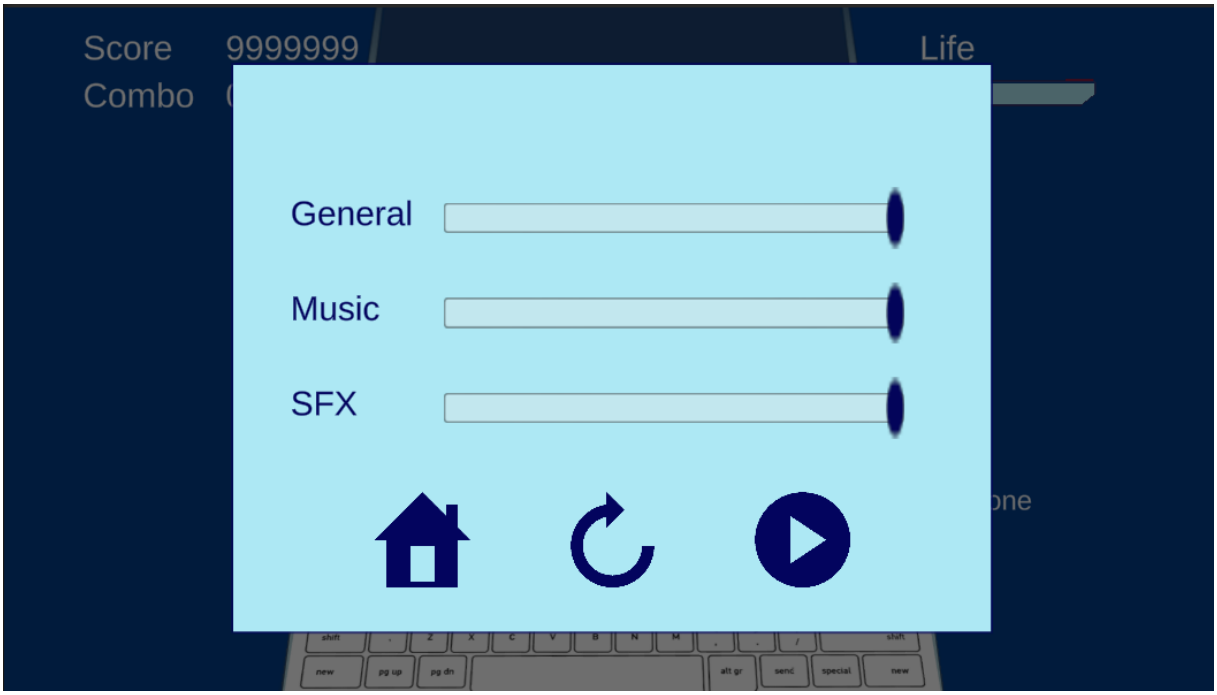

Ilustración 4.15 Interfaz del submenú de pausa

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 colisionador por la parte inferior.
- Por error se ha pulsado una tecla que no coincide con cualquier letra en el colisionador o no hay ninguna letra en el colisionador.

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. En los eventos anteriores definidos, 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 sí 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, hay que modificar el script de *ClickScript* para implementar estos eventos.

4.4.3. Salud

Para el sistema de salud se ha preferido usar un sistema sencillo de vidas con un valor máximo de 20 vidas. 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 va 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 colisionador por la parte inferior. Se va a restar una vida por cada letra que se elimine de esta forma.

Todos estos cambios son realizados en el script *ClickScript,* parte de estos en el método OnTriggerExit().

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. Además, se ha modificado *ClickScript* para que, en caso de que el menú de pausa esté activo, se desactive todas interacciones del teclado a excepción de la tecla Escape

Cuando se vuelve a pulsar la tecla Escape cambia el tiempo de vuela al valor Time.timeScale=1, esconde el menú y reproduce los audios pausados. Este script también se encarga de reiniciar el nivel a través de la función Restart(), recargando la escena. A la hora de reiniciar el nivel, es importante también volver a poner el valor de tiempo a Time.timeScale=1.

4.5. Iteración 4

En esta cuarta iteración las tareas a realizar son la M, N, O y P. 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 y su implementación. Esta parte fue incluida en el apartado 4.3.1.

Lo primero que se realizó fue la implementación del diseño de la interfaz del modo de fin de juego. Esta 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 marcador. Finalmente, en la parte inferior hay un par de botones de reiniciar el nivel y volver al menú principal. El diseño final del menú de fin de juego se puede observar en la Ilustración 4.16.

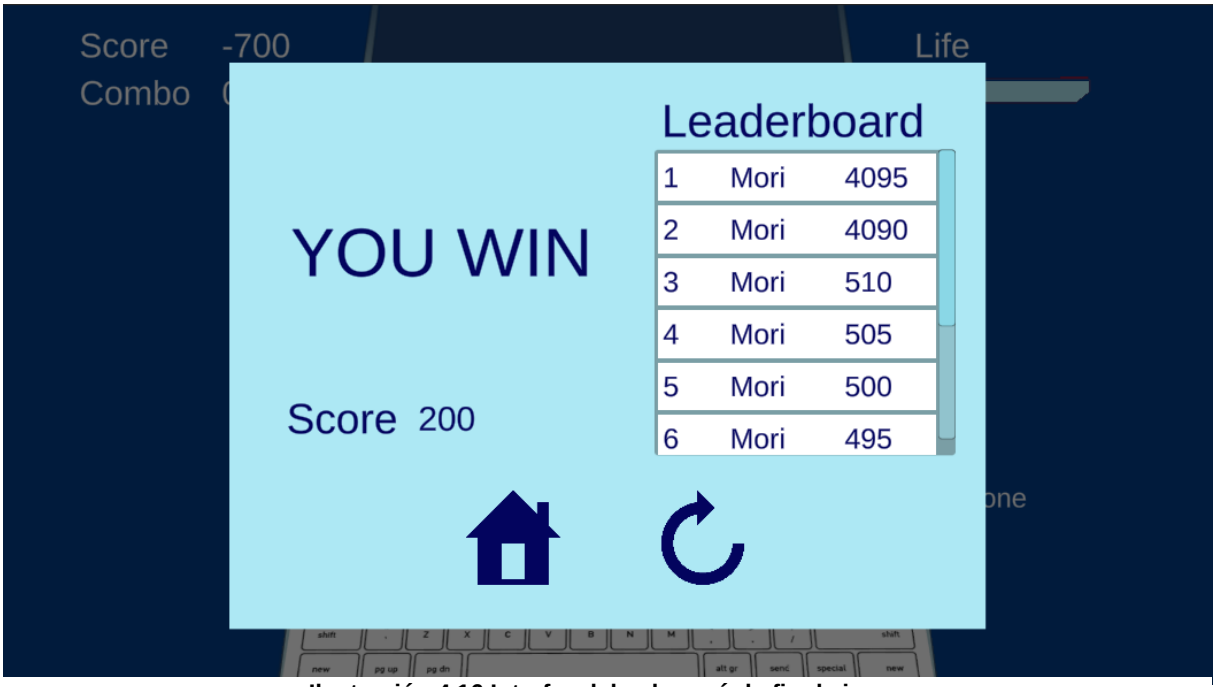

Ilustración 4.16 Interfaz del submenú de fin de juego

Diseñada la interfaz, se procede a implementar la funcionalidad. Como se vio anteriormente, se implementó un sistema de vidas. Esta funcionalidad se encuentra en *ClickScript*, más concretamente, en la función OnTriggerExit(). Cuando un objeto es eliminado como consecuencia de un error, se comprueba si todavía quedan vidas. En caso de no tener vidas, muestra el menú de fin de juego, modifica el valor de score mostrado. Además, muestra y guarda el nuevo marcador generado.

Aparte, se desea que, al eliminar la última letra de un nivel, el juego termine. Para ello hay que modificar la función *PlayLevel()* en *LetterManagement.* Cuando se genera la última letra, se guarda la referencia del último objeto creado y se

comprueba cada segundo si ha sido eliminado. Cuando se elimina, se muestra el menú de fin de juego, se modifica el valor de score mostrado y se actualiza el marcador.

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.

4.5.2. Marcadores

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 se encarga de modificar los valores mostrados individualmente y, LeadeBoardScript que posee toda la lógica de ordenación, mostrado de marcador en la interfaz 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.

De esta forma el funcionamiento normal del sistema es que se llame a SubmitUser(score). Esta carga el marcador correspondiente al modo de juego actual, usando las variables de *LetterManagement* como el level path o el nivel de dificultad, como se ve en la Ilustración 4.17.

```
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 4.17 Sistema de importación de marcadores

Luego, añade el score del usuario y exporta el nuevo marcador, sobrescribiendo el archivo previamente guardado. Finalmente, elimina la visualización del marcador para evitar algún error que pudiera generarse y lo repinta como se ve en la Ilustración 4.18.

```
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 4.18 Pintado de marcadores en escena

Los datos mostrados en el marcador se pueden mostrar en la Ilustración 4.19. Finalmente, se puede integrar el marcador con la función EndGame() de *ClickScript* e introducir un marcador en la pantalla de fin de juego.

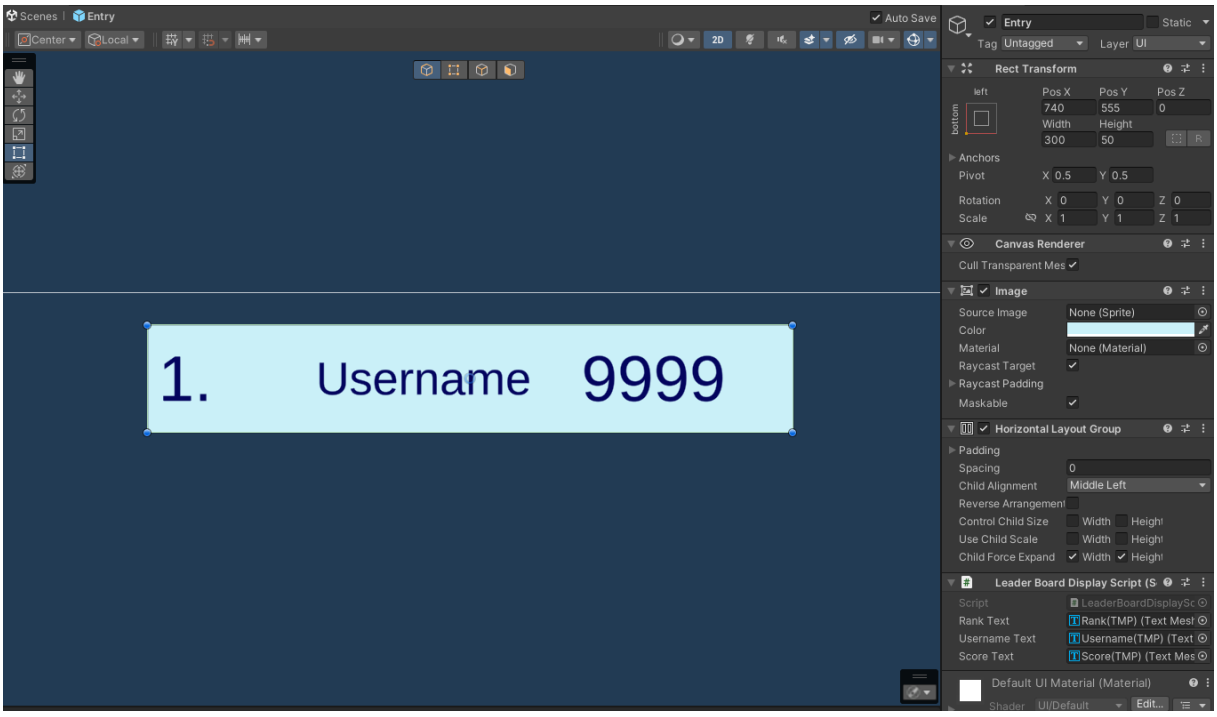

Ilustración 4.19 Prefab del elemento de los marcadores

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.

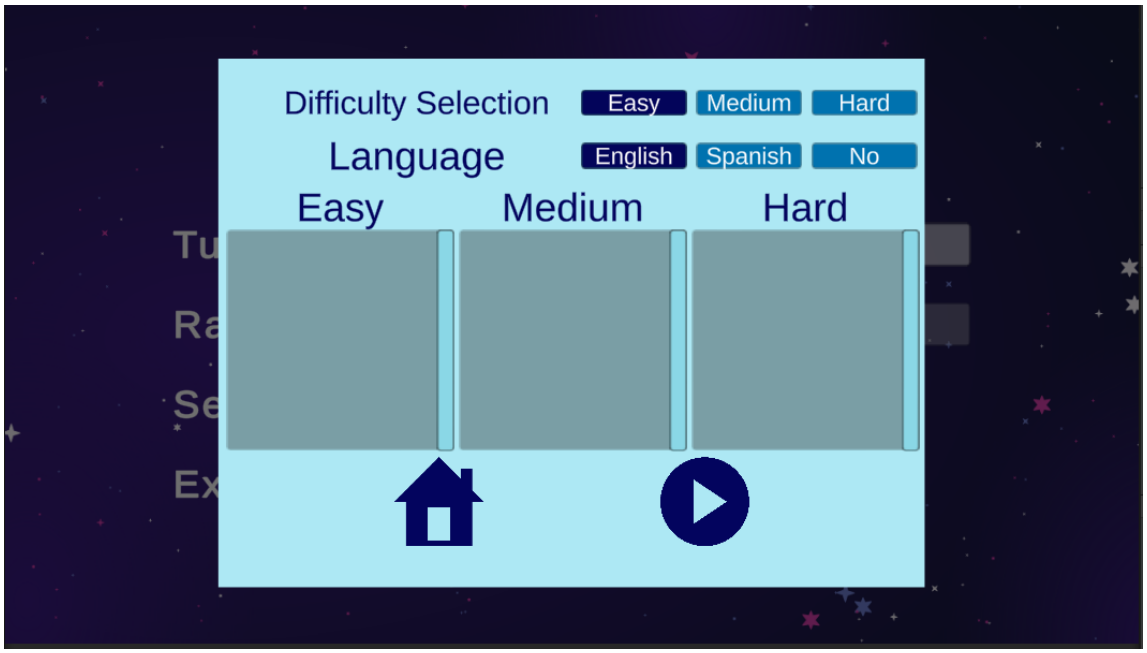

Ilustración 4.20 Interfaz del submenú random

Cuando se cambia uno de estos valores se modifican las variables de *LetterManagerScript* a través de la función *setDifficulty()* de *MainMenuScript,* de modo que cuando se pulsa el botón de inicio, están las variables ya seleccionadas y no hay problemas de pasar variables entre escenas. El diseño final de la interfaz se puede observar en la Ilustración 4.20.

Tanto la gestión de la ventana del menú, como de los botones de inicio de nivel y de volver al menú principal es gestionado por *MainMenuScript.*

4.5.4. Letras 3D

Para la realización de las letras 3D se ha usado un plugin 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, como se puede observar en la Ilustración 4.21. Esto, con una buena iluminación de escena, ha sido suficiente para crear una sensación de profundidad.

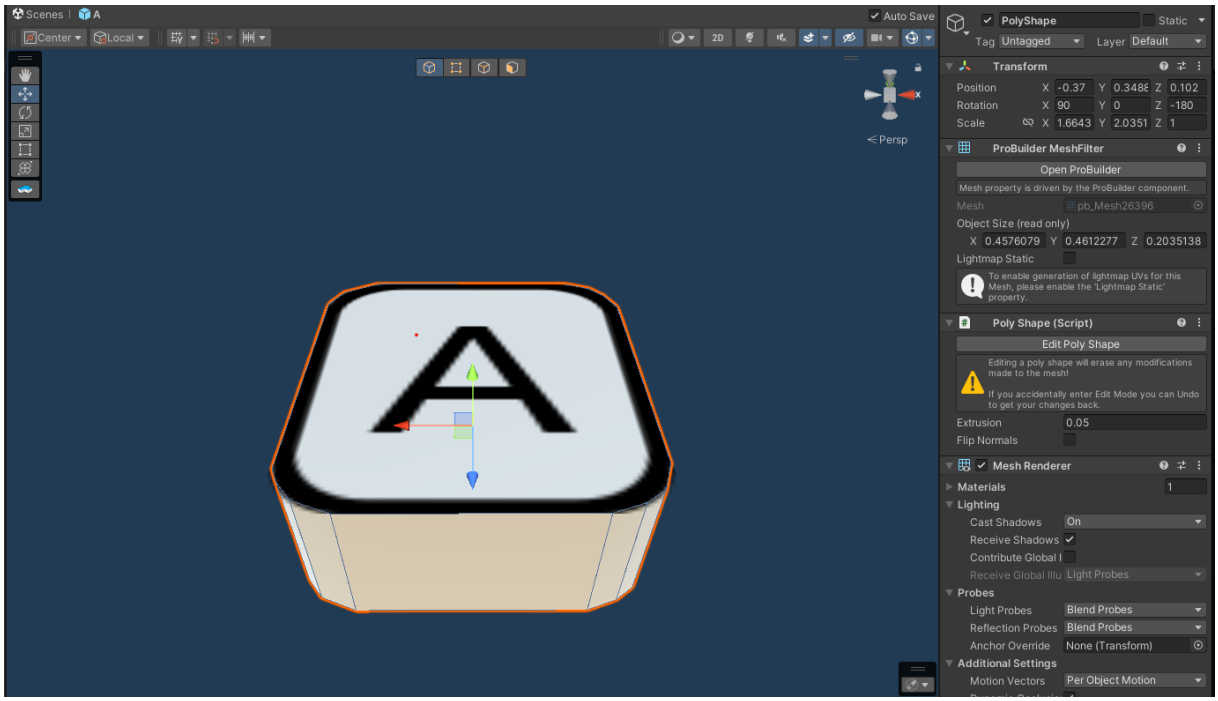

Ilustración 4.21 Datos del modelo 3d en el editor de ProBuilder

4.6. Iteración 5

En esta quinta iteración las tareas a realizar son la Q, R, S y T. Estas tareas corresponden con la implementación de parte de la interfaz, concretamente del Menú Niveles y su funcionalidad, modificar toda la parte de la interfaz para añadir la paleta de colores. Además, se tienen que crear los niveles de práctica. Para ello se ha creado un modo Debug para crear niveles. Finalmente, se ha implementado diccionario inglés en el Modo Random y se han realizado pruebas.

4.6.1. Interfaz modo niveles

Para el submenú del modo niveles hay dos secciones diferenciadas, la parte superior indica los mundos y la parte central indica los niveles, como se puede ver en la Ilustración 4.22.

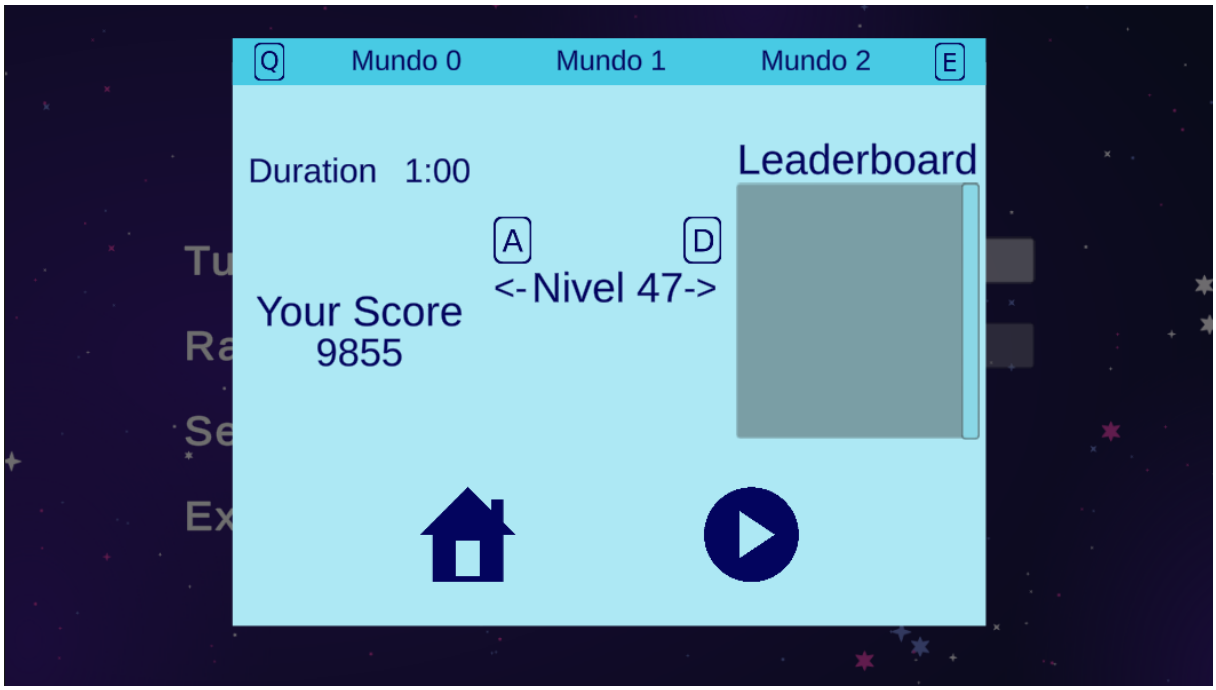

Ilustración 4.22 Interfaz del submenú niveles

Un mundo es un conjunto de niveles. Todos los datos de los niveles y los mundos son cargados cuando se carga la escena. Para ello, mira todos los niveles que hay en la carpeta de niveles y los agrupa por mundos. Estos niveles tienen una estructura:

[mundo]-[nivel].csv

Por ejemplo, un nivel puede ser *world 1-level 1.csv*. Por lo que agrupará todos los niveles que hay con prefijo “world 1”, pero el juego no sabe cuántos niveles hay previamente.

Volviendo a la interfaz se usan las letras Q y E para moverse entre los mundos. Cuando se cambia un mundo se actualiza el nivel a mostrar. Al actualizarse el nivel se cargan los metadatos de puntuación, duración y se carga el marcador del nivel. Para cambiar el nivel, se usan las letras A y D. Al iniciar el nivel, se modifican las variables de *LetterManagerScript* de modo no hay problemas de pasar variables entre escenas.

Todas las capturas de interfaz previamente hechas en esta memoria han sido realizadas con la paleta de colores final. Simplemente, en esta parte del trabajo se

modificó toda la interfaz y se implementó la paleta de colores expuesta en el apartado 3.3.

Tanto la gestión de la ventana del menú, como de los botones de inicio de nivel y de volver al menú principal es gestionado por *MainMenuScript.*

4.6.2. Niveles

Se han planteado seis niveles a implementar. Los primeros cinco niveles servirían de entrenamiento y el último funcionaría como una prueba para comprobar que los conocimientos previamente enseñados han dado resultados.

Para ello, se van a explicar los distintos niveles y sus características:

- Nivel 1:
  - Música: DJ Okawari - Flower Dance
  - Ritmo: 50BPM
  - Tiempo: 57 segundos
  - Número de letras: 35
  - Intencionalidad: Aprender a posicionar la mano en el teclado y aprender las letras L y F
- Nivel 2:
  - Música: Museum (Insect Room) - Animal Crossing: New Horizons (OST)
  - Ritmo: 60 BPM
  - Tiempo: 1 min 36 segundos
  - Número de letras: 35
  - Intencionalidad: Aprender las letras D y K
- Nivel 3:
  - Música: Kick Back (Chainsaw Man Opening 1) Lofi – Nemu
  - Ritmo: 89 BPM
  - Tiempo: 1 min 13 segundos
  - Número de letras: 52
  - Intencionalidad: Aprender las letras S y L
- Nivel 4:
  - Música: Tuca donka – CURSEDEVIL, DJ FKU, Skorde

  - Ritmo: 13 BPM
  - Tiempo: 1 min 15 segundos
  - Número de letras: 55
  - Intencionalidad: Aprender la letra A e intercalar con letras de otros niveles de forma asidua
- Nivel 5:
  - Música: 08 Hard Trap Beat & Instrumental 2016 "1:AM E 92"
  - Ritmo: 80 BPM
  - Tiempo: 1 min 12 segundos
  - Número de letras: 40
  - Intencionalidad: Aprender la letra R y U e intercalar con letras de otros niveles de forma asidua
- Nivel 6:
  - Música: La distancia para un duelo – Shiro Sagisu
  - Ritmo: 80 BPM
  - Tiempo: 1 min 53 segundos
  - Número de letras: 86
  - Intencionalidad: Testear el progreso del usuario

Para la correcta realización de los niveles se ha implementado un modo depuración activado por la tecla espacio. Muestra los diferentes datos en formato separado por punto y coma al pulsar una tecla. Además, se ideó un código que permite recalcular el nivel de *timeToWait* a *timeToAppear* e viceversa. Gracias a esta funcionalidad, la construcción de los niveles fue realizada de forma más sencilla.

4.6.3. Pruebas

Aquí se muestran las diferentes pruebas realizadas como parte de la tarea T. Principalmente se exponen los diferentes errores encontrados en el juego. En algunos casos se han implementado las soluciones de estos y en otros, simplemente es un aviso que deberá de ser corregido en caso de que este prototipo continue su desarrollo.

Las pruebas consisten en probar cada una de las funcionalidades diferentes del proyecto y, ante todo, pensar que el usuario no realizará el flujo normal de juego. Para ello se ha utilizado el apartado 3.4 para listar todas las posibles “situaciones” en las cuales se puede encontrar el juego. Luego se han aplicado las siguientes pruebas:

- Pulsar teclas del teclado rápidamente.
- Pulsar varias teclas a la vez.
- Pulsar teclas que no conllevan ninguna acción en la situación actual.
- Cambiar varias veces parámetros de un sistema de forma rápida.

Por ejemplo, un usuario podría por ejemplo confundirse de parámetros del modo random y volver al menú principal, pero cambiar los parámetros antes de volver a iniciar el juego. Otra situación podría ser pulsar letras mientras el juego se encuentra pausado o en el menú principal o en cualquier situación.

Por tanto, pese a que pueda pasar por alto ciertas situaciones, se ha comprobado situaciones altamente poco probables, lo cual me garantiza que el sistema, al menos en su funcionalidad principal, es bastante robusto.

El primero de los errores ya fue comentado en el apartado 4.3.3. Este error consistía en un problema a la hora de eliminar las letras cuando se acertaba. Para corregirlo se implementaron varias banderas que servían como semáforos. De esta forma, se podía filtrar en la función de OnTriggerExit() si una letra era resultado de un acierto o de un error.

Otro error se encuentra al pausar el juego en los primeros segundos de un nivel del modo Niveles. Al reanudar el juego, podemos observar que la música se reproduce instantáneamente. El error se encuentra identificado en la función PlayAudio() de *SoundFXScript*. Esto es debido a que la funcionalidad de PlayDelayed() no es correlativa a la función de Time.timeScale. La solución ha consistido en guardar el tiempo de retraso del clip a reproducir y guardar el tiempo cuando se ejecutó la función PlayDelayed(). Luego a la hora de pausar la música se comprueba si el tiempo que ha transcurrido es inferior al tiempo de retraso del clip. Si es así se guarda el tiempo restante. Luego a la hora de reanudar los audios,

simplemente se hace una corrutina para lanzar un hilo de ejecución y, el hilo que reanuda la música es mandado a dormir el tiempo guardado anteriormente.

El siguiente error se encuentra identificado al cambiar de usuario. El submenú de Niveles mantiene el valor de score del usuario previo al cambio si el nuevo usuario no tiene una puntuación previa.

El último error se encuentra identificado cuando se pausa un nivel. Se permitía al usuario pulsar teclas mientras se encontraba en modo de pausa. Esto se solucionó con una comprobación del parámetro de tiempo, para averiguar si el juego está pausado.

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 él tiene que ser estudiado previamente y realizar 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 Ingeniería de Software, Animación 3D y posprocesamiento, Ingeniería gráfica o Gestión y Control de Proyectos Informáticos entre otras.

A nivel personal, creo que ha sido más un reto y quitarme el “gusanillo” de realizar un prototipo real que otra cosa. Hoy en día es complicado entrar en la industria del videojuego y conozco a mucha gente que son más capaces y constantes y no consiguen trabajar de esto. No creo que sea imposible, pero es un esfuerzo elevado para conseguir trabajar en el mercado de videojuegos móviles, que es donde está el dinero actualmente. No creo que eso sea algo que quiera hacer en un futuro.

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.

Anexo I. Manual de instalación del sistema

El prototipo está disponible en el siguiente enlace:

https://github.com/DPM1997/TFGMecanography

Aquí lo único que hace falta es descargar el .zip y lanzar el ejecutable. En caso de que se quiera acceder al contenido del juego se han de instalar 2 programas:

Visual Studio Code

Para instalar Visual Studio Code es necesario:

1. Acceder a la página oficial de descargas: https://visualstudio.microsoft.com/es/
2. Descargar el instalador compatible con el sistema operativo del equipo. En este caso, se ha instalado la versión de Windows x64. Cuando se haya descargado, seguir los pasos de instalación que marca el asistente. Actualmente se usa la versión 1.93
3. Descargar las extensiones de C#, C# Dev Kit, Unity, Unity Code Snippets y Unity Toolbox, como se puede ver en la Ilustración I-1.

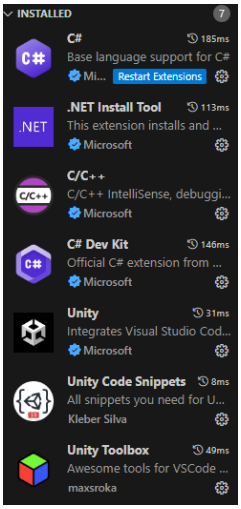

Ilustración I.1 Extensiones instaladas en Visual Studio Code

Unity

Para instalar Unity es necesario:

1. Acceder a la página oficial de descargas: https://unity.com/es/download
2. Descargar el instalador. Este instalador descargará Unity Hub.
3. Para acceder a la versión exacta del editor es necesario acceder a: https://unity.com/es/releases/editor/archive
4. En esta página, buscar la versión 2022.3.8f.
5. Cuando se haya instalado la versión, se puede acceder al proyecto.

Anexo II. Manual de usuario

En esta sección se explican como interactuar con las escenas y cuáles son los parámetros modificables.

Esta sección está dividida en tantos apartados como escenas y la estructura será similar al apartado 3.2.

Menú principal

Esta escena es la primera escena que se muestra al entrar en la aplicación. La primera interacción que existe es en la parte derecha donde hay un InputField para introducir el usuario. De normal, habrá un usuario por defecto, por lo que será necesario borrar e introducir el nuevo usuario. Cuando se haya realizado se podrá clicar en el botón de Submit User.

Al pulsar en los botones de Tutorial Mode, Random Mode y Settings se abrirá otros menús que serán explicados más adelante.

Finalmente, si se pulsa el botón de Exit, se saldrá del juego.

Este menú se puede ver en la Ilustración II.2.

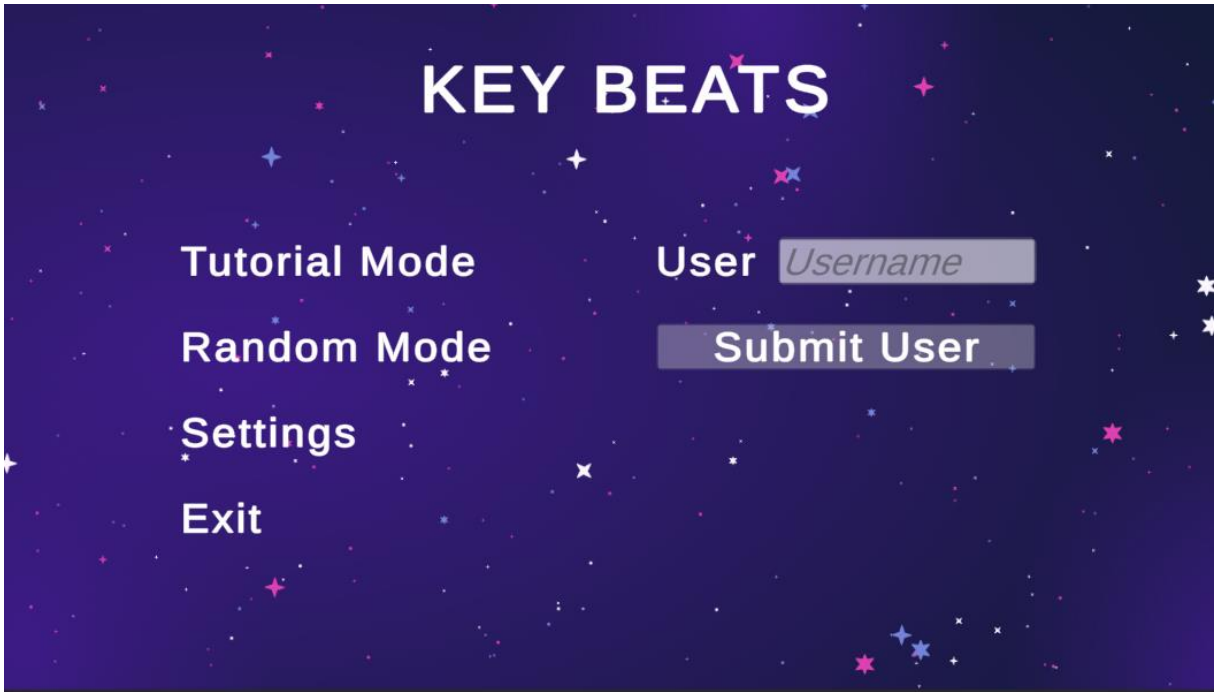

Ilustración II.2 Interfaz menú principal

Tutorial Mode

Este menú, que se puede ver en la Ilustración II.3, debe de ser interactuado con el teclado aparte del ratón. Se podrá mover de mundo con las letras Q y E y se podrá cambiar de nivel con las letras A y D. En la parte derecha se podrá visualizar el marcador del nivel, pudiendo hacer deslizamiento vertical con el ratón en la barra vertical a la derecha de este.

Al finalizar la selección, se podrá pulsar el botón de inicio de juego. Además, si se quiere retroceder al menú principal, se puede pulsar el botón con forma de casa.

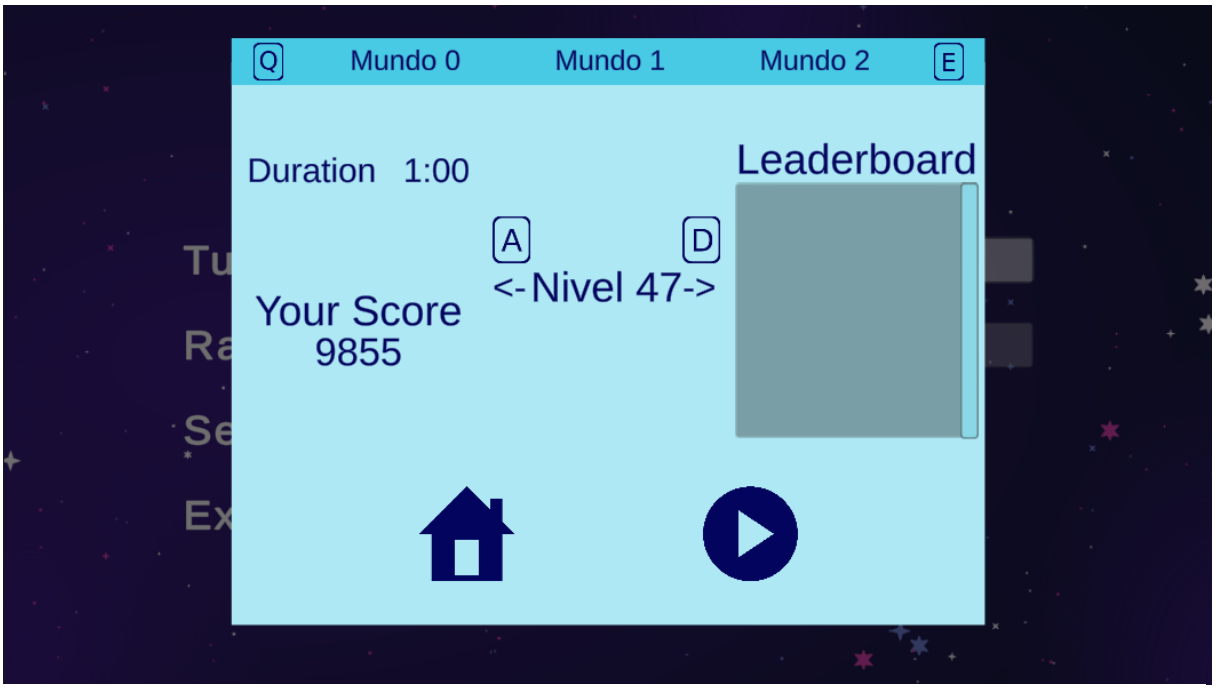

Ilustración II.3 Interfaz del menú de niveles

Random Mode

En este menú, que se puede ver en la Ilustración II.4, hay dos selectores principales y la opción seleccionada se visualizará de un color más oscuro. El selector superior indica el nivel de dificultad y se puede seleccionar entre 3 niveles: Easy, Medium y Hard.

El selector inferior indica si se quiere utilizar un diccionario para generar las letras. Hay disponibles 3 opciones: Diccionario de inglés, diccionario de español y no utilizar ningún diccionario.

En la parte inferior se encuentran varios marcadores, pudiendo hacer deslizamiento vertical con el ratón en la barra vertical a la derecha de este.

Al finalizar la selección, se podrá pulsar el botón de inicio de juego. Además, si se quiere retroceder al menú principal, se puede pulsar el botón con forma de casa.

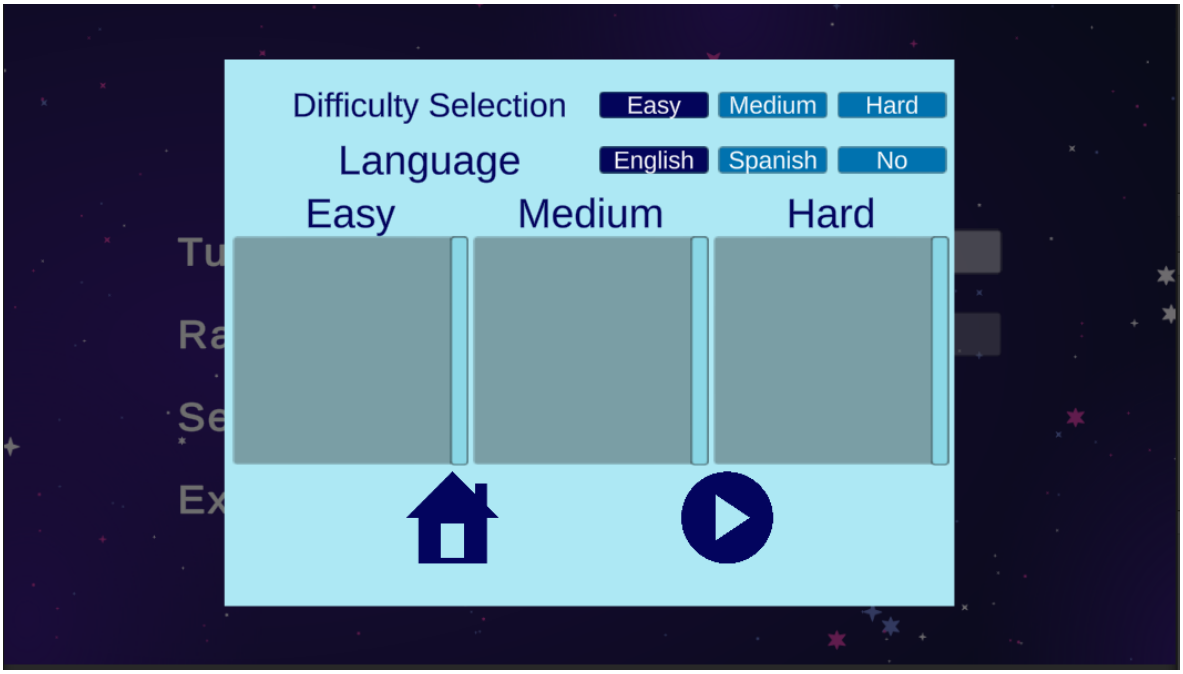

Ilustración II.4 Interfaz del menú random

Ajustes

En este menú hay tres selectores controles deslizantes. Cada uno de ellos marca el nivel de volumen de los distintos tipos de sonido. La única excepción a estos es el general, que baja tanto el volumen de la música, como de los efectos de sonido. Este menú se puede ver en la Ilustración II.5.

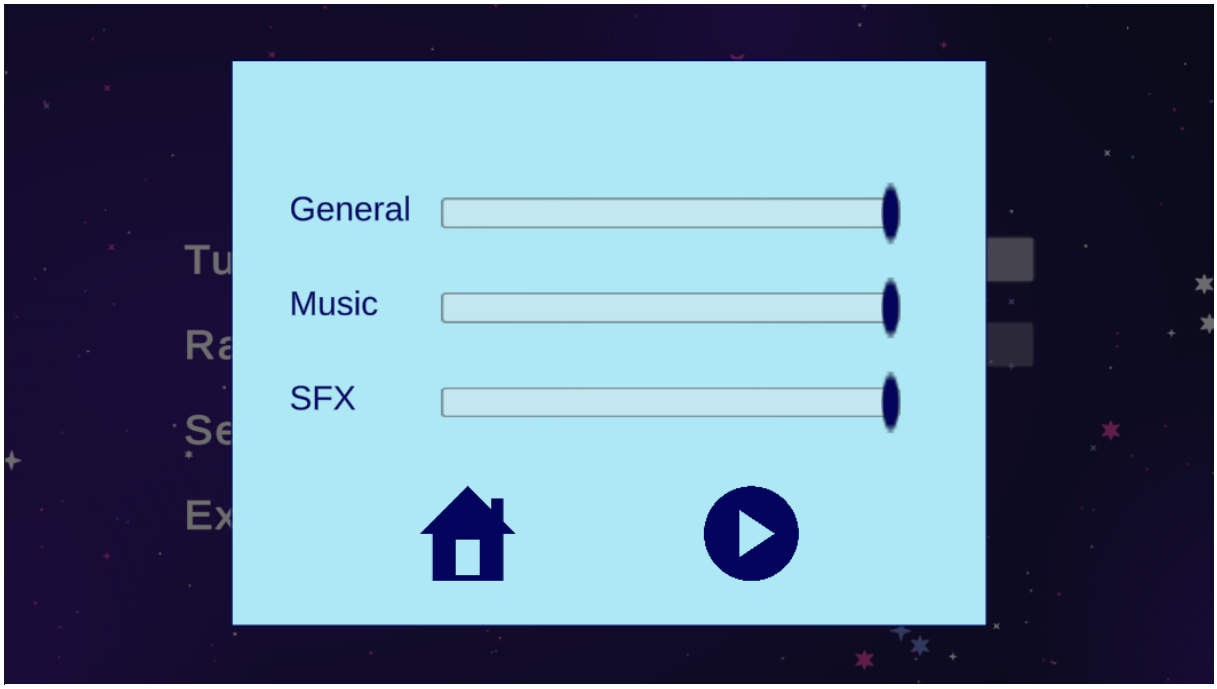

Ilustración II.5 Interfaz del menú de ajustes

Pausa

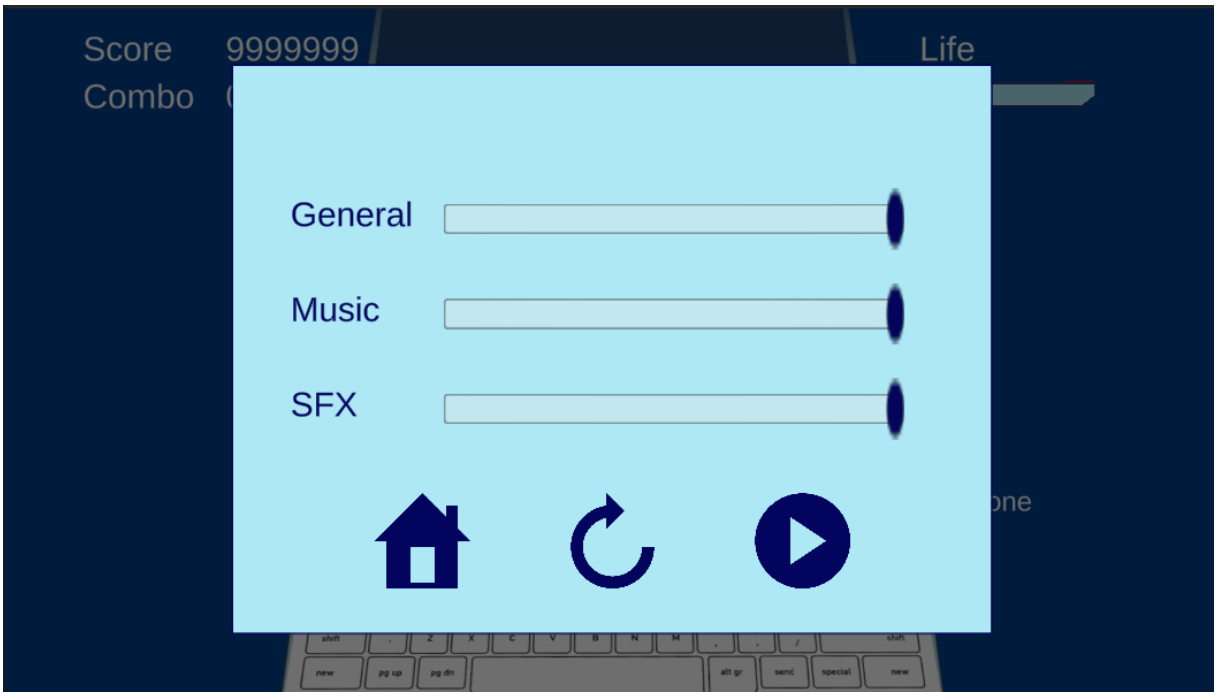

Ilustración II.6 Interfaz del menú de pausa

Es bastante similar con el menú de Ajustes. En este menú hay tres selectores controles deslizantes. Cada uno de ellos marca el nivel de volumen de los distintos tipos de sonido. La única excepción a estos es el general, que baja tanto el volumen

de la música, como de los efectos de sonido. Este menú se puede ver en la Ilustración II.6.

La diferencia principal radica en el botón de reinicio de nivel.

Zona de Juego

En esta escena no hay ninguna interacción con el ratón. Hay que pulsar la tecla que coincida con la letra que va cayendo de la parte superior de la pantalla como se puede ver en la Ilustración II.7.

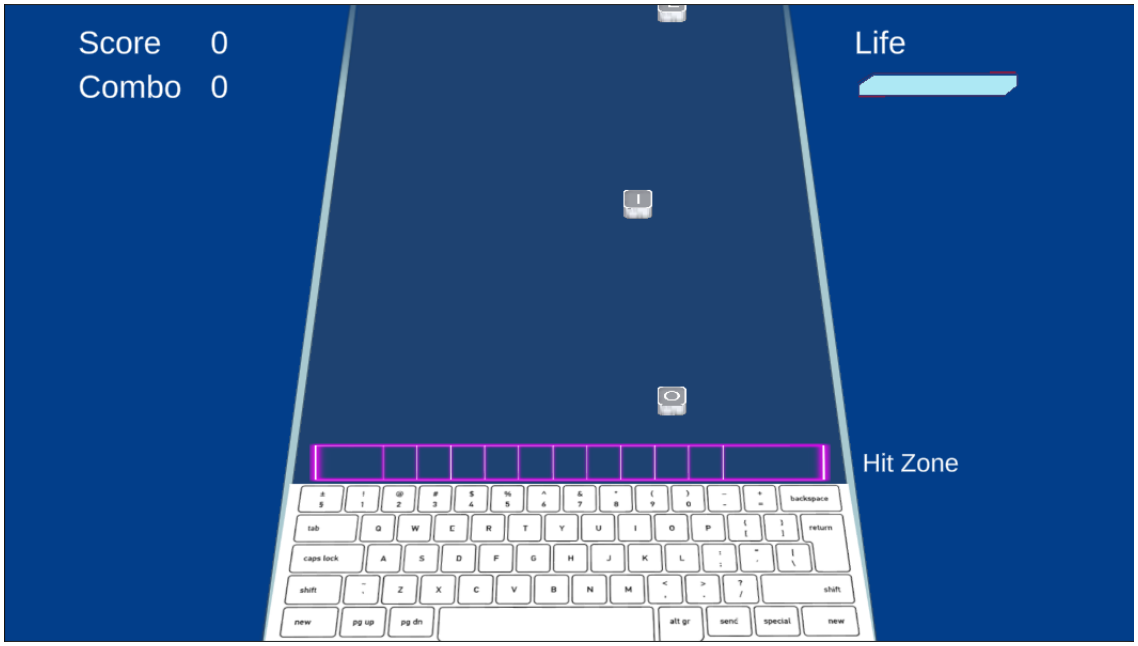

Ilustración II.7 Captura del juego en funcionamiento en la escena GameZone

Fin de juego

En este menú hay 1 marcador en la parte derecha, pudiendo hacer deslizamiento vertical con el ratón en la barra vertical a la derecha de este.

Al finalizar la visualización, se podrá pulsar el botón de reinicio de juego. Además, si se quiere retroceder al menú principal, se puede pulsar el botón con forma de casa. Este menú se puede ver en la Ilustración II.8.

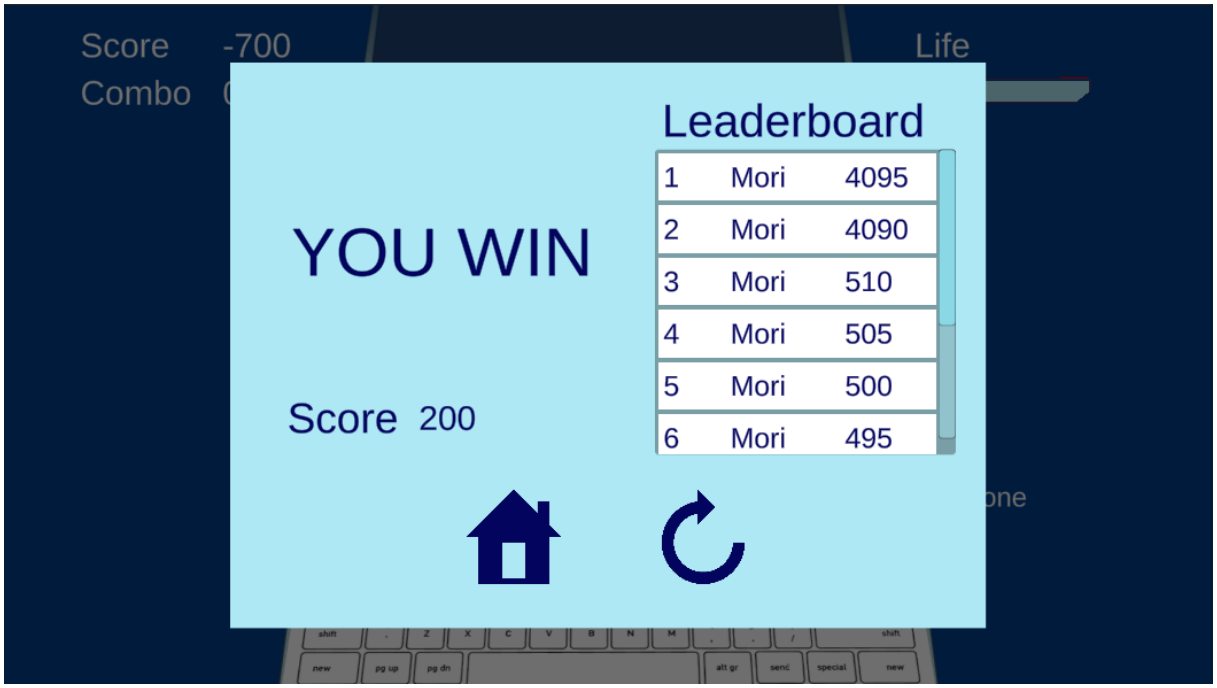

Ilustración II.8 Captura del juego en funcionamiento en la escena GameZone

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

Indeed. (2024 de Agosto de 11). *Indeed*. Recuperado el 05 de Septiembre de 2024, de https://es.indeed.com/career/desarrollador-junior/salaries

Indeed. (31 de Agosto de 2024). *Indeed*. Recuperado el 5 de Septiembre de 2024, de https://es.indeed.com/career/analista-programador/salaries

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

Nichols, D. (s.f.). *Coloring for Colorblindness*. Recuperado el 4 de Septiembre de 2024, de https://davidmathlogic.com/colorblind/#%2303045E-%23023E8A-%230077B6-

%230096C7-%2300B4D8-%2348CAE4-%2390E0EF-%23ADE8F4-%23CAF0F8-%2349A078-%23E63946

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