



Universitat
de les Illes Balears

TRABAJO DE FIN DE GRADO

VIDEOJUEGO EN REALIDAD VIRTUAL

Sergio Garcia Puertas

Grado de Ingeniería Informática

Escuela Politécnica Superior

Año Académico 2020-21

VIDEOJUEGO EN REALIDAD VIRTUAL

Sergio Garcia Puertas

Trabajo de Fin de Grado

Escuela Politécnica Superior

Universidad de las Illes Balears

Año Académico 2020-21

Palabras clave del trabajo:

videojuego, realidad virtual, escape room, multijugador, unity

Nombre Tutor/Tutora del Trabajo Antoni Oliver Tomàs

Nombre Tutor/Tutora del Trabajo Antonio Bibiloni Coll

Se autoriza la Universidad a incluir este trabajo en el Repositorio Institucional para su consulta en acceso abierto y difusión en línea, con fines exclusivamente académicos y de investigación

Autor		Tutor	
Sí	No	Sí	No
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Este trabajo de fin de grado se lo dedico principalmente a mis amigos, que además de hacerme más ameno el tiempo que le he dedicado, también me han ayudado dándome su opinión a medida que lo he ido desarrollando. También se lo dedico a mis padres, que son los que me han mantenido durante la carrera y no hubiese podido dedicarle el mismo tiempo a los estudios si no fuera por ellos.

Tabla de contenido

Índice de figuras	iii
Índice de tablas	v
Nomenclatura.....	vii
Resumen.....	ix
Capítulo 1. Introducción.....	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Alcance del proyecto	2
1.4. Estructura del documento.....	2
Capítulo 2. Estado del arte	3
2.1. Soluciones existentes a problemas similares	3
2.2. Análisis y selección de las herramientas	4
Capítulo 3. Gestión del proyecto.....	7
3.1. Metodología de trabajo	7
3.2. Planificación temporal.....	8
3.3. Gestión de los interesados	9
Capítulo 4. Desarrollo del proyecto	11
4.1. Análisis.....	11
4.1.1. Usuarios.....	11
4.1.2. Requisitos de usuario	11
4.2. Diseño.....	12
4.2.1. Diseño de la arquitectura	12
4.2.2. Diseño de las pantallas del juego	13
4.2.3. Diseño de la base de datos.....	15
4.2.4. Diseño del escape room	16
4.2.5. Diseño de la página web de los analistas	24
4.3. Implementación	25
4.3.1. Funcionamiento de Unity.....	25
4.3.2. Selección de herramientas.....	26

Índice de figuras

4.3.3. Explicación técnica de cada prueba	29
4.3.4. Integración con la base de datos.....	38
4.3.5. Multijugador.....	44
4.3.6. Análisis de los datos	48
Capítulo 5. Resultados.....	55
Capítulo 6. Conclusiones	61
Bibliografía	63

Índice de figuras

Figura 1. The Experiment: Escape room	3
Figura 2. Belko VR: An Escape Room Experiment	3
Figura 3. Tablero Kanban	8
Figura 4. Arquitectura de la aplicación	12
Figura 5. Lógica de las pantallas de juego	13
Figura 6. Diseño de la pantalla de inicio (izquierda), de la pantalla de registro (centro) y de la pantalla de inicio de sesión (derecha).....	14
Figura 7. Diseño de la pantalla de menú.....	14
Figura 8. Diseño de la pantalla para crear sala (izquierda) y la pantalla para empezar partida (derecha)	15
Figura 9. Modelo de la base de datos	16
Figura 10. Proceso de diseño de un escape room	18
Figura 11. Pruebas y flujo de juego del escape room	20
Figura 12. Vista de la planta del escape room	21
Figura 13. Imagen que aparece en la pantalla 1	22
Figura 14. Imagen que aparece en la pantalla 2	22
Figura 15. Imagen que contiene el pendrive.....	23
Figura 16. Instrucción que aparece en la puerta para entrar en la sala 2.....	23
Figura 17. Colocación de los tres elementos en el candado final	24
Figura 18. Interfaz de la página web de análisis	24
Figura 19. Diagrama de estados del menú de selección de la página web.....	25
Figura 20. Editor de Unity	26
Figura 21. Ejemplo de interactor e interactable	27
Figura 22. Componentes "XR Direct Interactor" (izquierda) y "XR Grab Interactable" (Derecha)	28
Figura 23. Botones de los mandos de Oculus	29
Figura 24. Elementos que componen la prueba 1	29
Figura 25. Botón para encender el PC (izquierda) y collider del dedo índice (derecha).....	30
Figura 26. Componente "OrdenadorEncendido"	31
Figura 27. Candado numérico	31

Índice de figuras

Figura 28. Botón "1" pulsado	32
Figura 29. Elementos que componen la prueba 2	33
Figura 30. Socket interactor para conectar el pendrive al PC.....	34
Figura 31. Eventos del socket interactor del PC.....	34
Figura 32. Pizarra que contiene la ecuación necesaria para resolver la prueba 3.....	35
Figura 33. Planetas instanciados necesarios para resolver la prueba 3.....	36
Figura 34. Puntero para destruir el planeta correspondiente	37
Figura 35. Jaula que contiene el candado final (izquierda) y Socket Interactor del candado final (derecha)	37
Figura 36. Eventos del candado final	38
Figura 37. Menú visto desde el editor	41
Figura 38. Panel del menú.....	42
Figura 39. Elementos iniciales de la página web.....	50
Figura 40. Selección de sala - página web.....	51
Figura 41. Resultado: menú inicial (izquierda), menú de registro (centro) y menú de inicio de sesión (Derecha).....	55
Figura 42. Resultado - Menú principal	55
Figura 43. Resultado - Crear sala.....	56
Figura 44. Escape room visual 1	56
Figura 45. Escape room visual 2	56
Figura 46. Escape room visual 3	57
Figura 47. Escape room visual 4	57
Figura 48. Escape room visual 5	57
Figura 49. Resultado - Mapa de calor de un escape room.....	58
Figura 50. Resultado - Mapa de calor de una partida	58
Figura 51. Resultado - Mapa de calor de un jugador en una partida.....	59

índice de tablas

Tabla 1. Comparativa entre los principales motores de videojuegos	4
---	---

Nomenclatura

LTIM	Laboratori de Tecnologies de la Informació Multimèdia.
UIB	Universitat de les Illes Balears.
Escape room	Juego que consiste en salir de una o varias habitaciones resolviendo pruebas.
Activo	Elemento que se puede utilizar más de una vez. Conocido en inglés como “asset”.
VR	Realidad Virtual (Virtual Reality).
Feedback	Retroalimentación.
PC	Ordenador Personal (Personal Computer).
BD	Base de datos.
API	Interfaz de programación de aplicaciones (Application Programming Interface).
Script	Conjunto de líneas de código que realizan una o varias funciones.
Low poly	Que está formado por pocos polígonos.
TFG	Trabajo de Fin de Grado.
Hitbox	Área de colisión.
Framework	Marco de trabajo.
Collider	Colisionador.

Resumen

Para este trabajo de fin de grado hemos desarrollado una aplicación, la cual está formada por un videojuego y una página web. El videojuego se puede jugar en pc utilizando un equipo de realidad virtual y trata de resolver un escape room. También hemos añadido la posibilidad de jugar multijugador. En la página web se muestra un mapa de calor de las posiciones de los jugadores en las diferentes partidas.

Con tal de que esto sea posible, hemos integrado el juego con una base de datos. Para esto también es necesario guardar información como qué escape rooms hay en el juego, las pruebas que hay en cada sala, las partidas que se juegan, los jugadores que participan, etc.

Primero hemos realizado un análisis de los usuarios que van a utilizar tanto el juego como la página web y, a partir de este análisis, hemos definido los requisitos de la aplicación. Entonces hemos hecho el diseño de las diferentes partes que conforman esta aplicación como lo son la arquitectura ésta, la interfaz de las diferentes pantallas del juego, el modelo relacional de la base de datos, el propio escape room que hemos desarrollado y la interfaz de la página web así como la lógica de como se ha de actualizar de manera dinámica.

Una vez hecho el diseño de los diferentes componentes, pasamos a la implementación de éstos. La explicación se realiza en el orden en que se ha ido realizando el trabajo. Primero se explica técnicamente cómo se ha creado el escape room. Esta explicación se hace para cada una de las pruebas que lo conforman. A continuación, explicamos como se ha integrado la base de datos de tal manera que se puedan guardar y recibir datos desde el juego. En este apartado no se incluye el almacenamiento y tratamiento de los datos referentes al posicionamiento de los jugadores necesarios para realizar el apartado de análisis de la página web.

Seguido, se explican todos los cambios que hemos tenido que realizar con tal de adaptar el videojuego de un solo jugador a multijugador. Finalmente, se explica cómo se envían y almacenan los datos necesarios para realizar el mapa de calor y cómo se ha realizado la página web que utiliza estos datos.

Por último, hay un capítulo dónde se muestran imágenes de cómo ha quedado tanto el videojuego como la web y otro capítulo con las conclusiones que hemos obtenido tras realizar este TFG.

Capítulo 1. Introducción

En este capítulo se explican las razones que nos han llevado a realizar esta aplicación como trabajo de fin de grado. También se definen los objetivos de este proyecto y el alcance.

1.1. Motivación

En la actualidad, el mercado está lleno de aplicaciones online, ya sean páginas web, aplicaciones móviles, videojuegos, etc. Durante la carrera se ve en diversas asignaturas cómo desarrollar páginas web y aplicaciones móviles y las diferentes tecnologías que se utilizan. Sin embargo, ninguna de las que yo he cursado estaban enfocadas en el desarrollo de videojuegos.

El área que más me interesa y en la que me gustaría profundizar más, y en un futuro trabajar, es la del desarrollo de aplicaciones. Tanto aplicaciones web, como móviles, como videojuegos. Como he dicho, durante la carrera he visto una parte de esta área, pero otra no. Por tanto, elegí hacer un videojuego para acabar de ver la parte que me faltaba. Evidentemente, lo que he visto, tanto de aplicaciones web y móvil como de videojuegos, es solo una pequeña parte, pero me es suficiente para introducirme en este ámbito y hacerme una idea general de cómo es y de cómo se trabaja.

El juego consistirá en completar un escape room, que consiste en salir de una o varias salas resolviendo pruebas. Personalmente he hecho varios y me parece una actividad muy entretenida y creo que la experiencia que se vive se puede trasladar de manera muy fiel a un videojuego.

1.2. Objetivos

El objetivo de este TFG es desarrollar un juego a pequeña escala, que tenga alguno de los componentes principales que tienen la mayoría de los juegos que salen al mercado en la actualidad. Los cuatro componentes que hemos decidido implementar han sido los siguientes:

1. **El juego básico**, que se podría jugar de manera individual. Con esta parte aprenderemos a utilizar el motor de videojuego elegido.
2. **Integración con una base de datos**. Durante la carrera hemos visto cómo se integra una base de datos en un servidor remoto con aplicaciones web y móvil, pero no con un juego realizado en uno de los motores de videojuego más populares.
3. **Multijugador**. Durante la carrera no hemos visto cómo se gestiona el multijugador de un videojuego y es algo que yo personalmente quería implementar. Esto se debe a que el punto anterior y el siguiente lo hemos aplicado sobre aplicaciones web y hacerlo sobre un videojuego sigue los mismos principios. Sin embargo, este es un área completamente nueva y que quiero aprender.
4. **Análisis de datos**. Pretendemos aplicar algún método para hacer un análisis sobre los datos que se recogen a partir de los jugadores. Esto es importante para aumentar las probabilidades de que un juego tenga éxito.

Es importante resaltar que la intención no es hacer un juego que gráficamente sea impecable, o que ofrezca una jugabilidad como la que puede ofrecer un juego AAA [1], sino simular la arquitectura que tendría un videojuego de verdad. Este razonamiento cobra importancia a la hora de tomar ciertas decisiones más adelante. La aplicación se realizará de tal manera que, si

1. Introducción

se quisiese, se pudiese expandir añadiendo más escape rooms y gestionando toda la información de manera adecuada.

1.3. Alcance del proyecto

Comentaremos el alcance respecto a cada uno de los componentes:

1. **Juego básico:** se trata de realizar un escape room que se pueda jugar en realidad virtual. Está pensado para que una partida dure alrededor de 10-15 minutos. De este bloque de trabajo haremos el diseño de las pruebas y del flujo de juego que tendrá, pero no haremos un diseño gráfico de todos los elementos que forman el escape room. La razón principal es que hace esto llevaría mucho tiempo e impediría realizar alguno de los otros bloques.
2. **Integración con una base de datos:** gestionará tanto el registro e inicio de sesión como la consulta y el envío de datos sobre las partidas que hagan los jugadores.
3. **Multijugador:** se deberán poder crear partidas para jugar en grupo. Esto implica implementar la gestión de los elementos con los que interactuarán los diferentes jugadores. También se deberán crear los elementos pertinentes en el menú para crear partidas y unirse a ellas.
4. **Análisis de datos:** este apartado consiste principalmente en crear una página web donde se muestre un mapa de calor a partir de los datos de posicionamiento que se vayan generando durante las partidas. También se incluye en este apartado el envío de éstos por parte de los jugadores al servidor. Únicamente se almacenarán datos sobre la posición, pero se creará la arquitectura de tal manera que, si se quisiesen guardar otros datos para hacer otro tipo de análisis, se pudiese hacer.

1.4. Estructura del documento

El primer apartado a continuación es el del estado del arte. Aquí se exponen las tecnologías más populares que existen para realizar un proyecto como el mío y se comentan algunos de los rasgos más importantes de los videojuegos actuales de escape rooms.

Después tenemos el apartado de gestión. Aquí explicamos la metodología de trabajo que hemos utilizado, así como la gestión de diversas áreas del proyecto como pueden ser el tiempo y los interesados.

Posteriormente se encuentra el capítulo de desarrollo. Este se divide en tres partes. El análisis de la aplicación, el diseño de las diferentes partes y finalmente una explicación detallada de cómo se han implementado los diferentes componentes.

En el capítulo de resultados se muestra cómo queda la aplicación al completo, uniendo los 4 componentes.

Finalmente se encuentran las conclusiones que hemos sacado tras realizar este proyecto y la bibliografía utilizada.

Capítulo 2. Estado del arte

En este capítulo se estudian las diferentes soluciones que existen para nuestro problema. En este caso miramos los juegos más populares cuyas características sean las mismas al nuestro, y también analizamos las principales herramientas que existen para desarrollarlos.

2.1. Soluciones existentes a problemas similares

Actualmente ya existen muchos juegos similares al que hemos de desarrollar para este proyecto. Únicamente buscando en la plataforma Steam [2] las palabras “escape room” nos aparecen 942 resultados. Pese a que no todos son compatibles con la realidad virtual, muchos de ellos sí lo son. En estos últimos años se ha expandido mucho esta área y es que posibilita a las personas relacionarse con sus amigos o conocidos de una manera más cercana, pese a estar a kilómetros de distancia.

Al hacer la búsqueda, los dos juegos compatibles con VR con más relevancia son: “The Experiment: Escape room” y “Belko VR: An Escape Room Experiment”. Se pueden ver imágenes del primero en la Figura 1 e imágenes del segundo en la Figura 2.



Figura 1. The Experiment: Escape room



Figura 2. Belko VR: An Escape Room Experiment

2. Estado del arte

Del primer juego llama la atención la ambientación de suspense, que al jugarlo en realidad virtual hace que los jugadores se metan mucho más en el papel. El segundo juego tiene unos gráficos menos realistas, pero de igual manera intenta simular una oficina real. Probablemente el aspecto más importante de un escape room en realidad virtual es ambientar el escenario para que el jugador se inmersa lo máximo posible y pueda tener una experiencia similar a la de hacer un escape room en persona.

El juego “The experiment” si ofrece la posibilidad de jugar multijugador, en cambio “Belko VR” no. Al mirar el resto de los juegos que aparecen al realizar la búsqueda, vemos que aproximadamente la mitad de éstos son únicamente para un jugador y la otra mitad son multijugador. Esto dato nos sorprende ya que la gran mayoría de personas, al realizar un escape room presencialmente, lo hace en un grupo de al menos dos personas.

En cuanto a la interacción que ofrecen estos juegos, la gran mayoría están enfocados en simular las interacciones con las manos. Por ejemplo, coger objetos, abrir puertas, abrir cajones, insertar una llave en una cerradura, etc. Sin embargo, para las extremidades inferiores no existen muchas interacciones, se limitan a simular el movimiento natural que tendría la persona.

Otro aspecto importante sobre los escape rooms es cuán secuenciales son las pruebas. Por lo general, los que están pensados para que participen muchas personas a la vez suelen ser menos secuenciales ya que de esta manera se consigue que todas las personas tengan algo que hacer en todo momento.

2.2. Análisis y selección de las herramientas

Prácticamente todos los juegos que aparecen en Steam están desarrollados con un motor de videojuegos de los que analizamos justo a continuación. Existen bastantes motores de videojuegos, pero los que más relevancia tienen y a los que cualquier persona tiene acceso son Unity [3], Unreal Engine [4] y Godot [5]. Todos ellos son viables para desarrollar una aplicación como la de este TFG. En la Tabla 1 se muestra una comparativa entre estos 3 motores:

	Unity	Unreal Engine	Godot
Precio	Gratuito para uso personal	Gratuito para uso personal	Totalmente gratuito
Documentación	Muy buena	Muy buena	Buena
Lenguaje	C#	C++	C#
Dificultad	Baja / media	Media / alta	Baja / media
Recursos adicionales	Sí	Sí	No

Tabla 1. Comparativa entre los principales motores de videojuegos

Los apartados de la Tabla 1 son los puntos que más impacto tienen sobre la decisión de qué motor usar para el desarrollo de este TFG. En cuanto al precio, los tres son gratuitos para el uso personal.

2.2. Análisis y selección de las herramientas

En cuanto a la dificultad, tanto Unity como Godot tienen una dificultad media, además de utilizar C# [6] como lenguaje de programación. Unreal Engine aparte de ser más complejo de por sí, utiliza un lenguaje que en mi opinión es más complicado [7], o por lo menos es un lenguaje que domino menos. C# es muy similar a Java [8], que lo hemos utilizado bastante durante la carrera. A consecuencia de esto he descartado utilizar Unreal Engine.

Por otro lado, Unity cuenta con muchos paquetes propios que facilitan el desarrollo, así como con un Marketplace (recursos adicionales) donde los usuarios venden u ofrecen de manera gratuita diferentes activos. Godot en cambio no. Además, que la documentación de Unity es bastante más completa que la de Godot.

Con todo esto en mente, hemos decidido utilizar Unity.

Capítulo 3. Gestión del proyecto

A la hora de desarrollar un proyecto es importante definir la manera en que se va a trabajar, así como planificar diferentes áreas de gestión de éste. Gracias a esta planificación inicial tendremos una idea más concisa del trabajo que se ha de realizar.

3.1. Metodología de trabajo

Para este proyecto trabajaremos aplicando una metodología agile [9] [10]. Hay dos motivos principales por los que hemos elegido esta manera de trabajar. El primero es que, pese a que sí hemos definido unos requisitos principales, en función de cómo vaya avanzando el TFG se podrán añadir o eliminar requisitos. También se podrán modificar algunos requisitos si a la hora de implementarlos vemos que no estaban bien planteados en un inicio.

El otro motivo principal es que el tutor no tiene experiencia en Unity y yo solo tengo un poco. Es por esto por lo que no podemos planificar con suficiente precisión cuanto vamos a tardar en realizar cada tarea y, por tanto, es posible que finalmente no podamos hacer todo lo previsto en un inicio.

Trabajaremos en sprints (ciclos) de alrededor de dos/tres semanas. Al final de cada ciclo tendremos, el tutor y yo, una reunión donde le mostraré el progreso para que él me de feedback sobre el incremento realizado en ese ciclo. Esta metodología, dentro de ágil, se conoce como un desarrollo incremental por prototipos.

Antes del primer ciclo definiremos los requisitos principales que se deberán cumplir. A partir de estos requisitos estableceremos tareas genéricas que deberemos desarrollar con tal de acabar cumpliendo con estos requisitos. Estas tareas genéricas formarán lo que se conoce como product backlog. Al iniciar un sprint elegiremos las tareas a implementar en ese sprint. Cada tarea genérica se dividirá en subtareas, que también estarán priorizadas.

Al final de cada sprint valoraremos si se deben añadir, quitar o modificar alguno de los requisitos. Lo que implicaría añadir, quitar o modificar algunas tareas.

Esta manera de trabajar nos ayudará a organizar y realizar las tareas en el orden que creemos que es el correcto, además de guiarnos a la hora de saber si el incremento de ese sprint es suficiente o no.

Con tal de poder organizar el product backlog, el sprint backlog y las subtareas que hemos de hacer en cada sprint, utilizaremos Trello [11].

El tablero Kanban [12] está compuesto por 5 columnas:

- **Product backlog:** tareas genéricas que hemos de hacer para poder desarrollar el proyecto. Las tareas están priorizadas en función del orden en que creemos que se deben realizar.
- **Sprint backlog:** aparecen las tareas del product backlog que hemos de hacer ese sprint. Cada tarea tiene un color que la identifica y siguen la misma priorización que tenían en el product backlog.

3. Gestión del proyecto

- **To Do, Doing, Done:** se dividen las tareas genéricas del backlog en tareas más específicas. Cada tarea específica tiene el mismo color que la tarea general de la que procede. Estas tareas también están ordenadas. Únicamente aparecerán las subtareas de las tareas genéricas de ese sprint.

El Kanban al iniciar el primer sprint queda como se puede ver en la Figura 3.

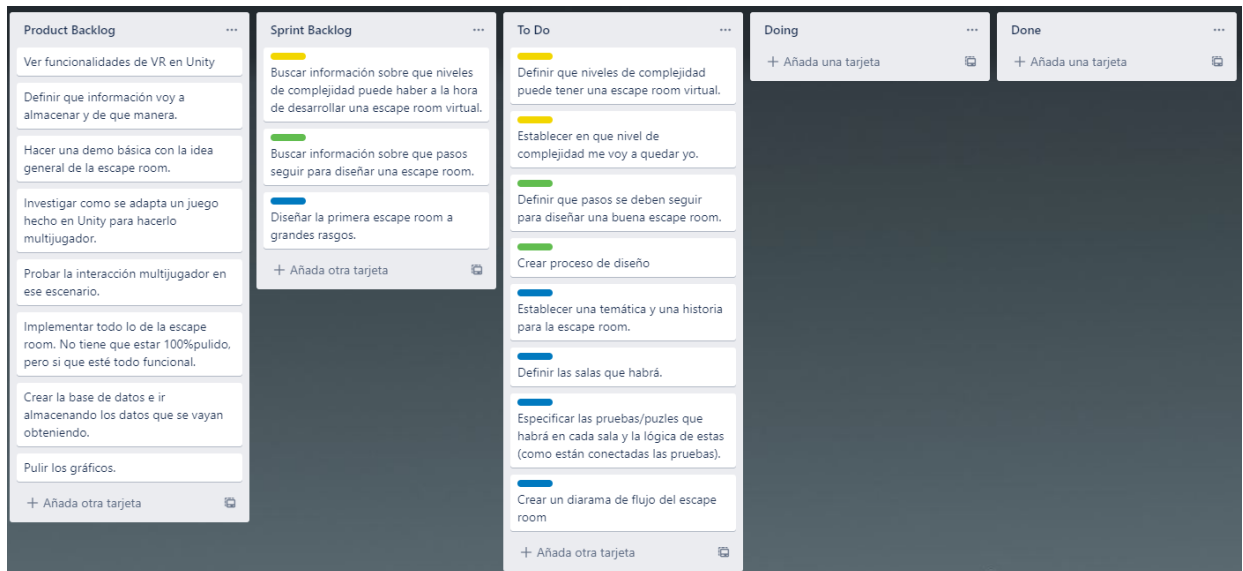


Figura 3. Tablero Kanban

3.2. Planificación temporal

La planificación temporal inicial la hemos hecho sobre los cuatro bloques de trabajo mencionados anteriormente. Hemos empezado con el TFG al iniciar el segundo cuatrimestre (22/02/2021) y pretendemos entregarlo antes de la primera convocatoria (04/07/2021).

El primer bloque de trabajo (hacer el escape room) es el que más trabajo lleva. Principalmente porque es algo nuevo para nosotros y antes de poder empezar a hacer nada tenemos que aprender cómo funcionan las herramientas que vamos a utilizar. El tiempo estimado es de aproximadamente un mes y medio.

El segundo bloque es el de implementar la base de datos. Para este bloque estimamos que tendrá una duración de 2 semanas. En este bloque no se contempla el envío de datos que se utilizaran para el análisis de datos. Únicamente se realizará la gestión de cuentas y las peticiones que se realizan desde el juego.

El multijugador, como hemos comentado en la introducción, es algo totalmente nuevo para mí, y por tanto estimamos que tardaremos unas 3 semanas ya que debo de aprenderlo desde cero.

Para el último bloque, guardar todos los datos del posicionamiento de los jugadores durante las partidas y hacer un pequeño análisis, estimamos que tardaremos alrededor de 2 semanas. Este tiempo es el que más puede variar ya que en función del tiempo que quede haremos un análisis más o menos laborioso.

El resto del tiempo lo utilizaremos para realizar la memoria.

3.3. Gestión de los interesados

Los principales interesados de este proyecto somos el tutor y yo. Es por esto por lo que, como he dicho en el apartado 3.1, tendremos una reunión cada 2-3 semanas para que el tutor vea el avance que he hecho y me dé su opinión respecto al trabajo realizado. De esta manera me aseguro de que el trabajo que voy realizando tiene su visto bueno, lo que implica que no tendré que realizar modificaciones drásticas al final del proyecto.

Capítulo 4. Desarrollo del proyecto

En este capítulo se distinguen tres subapartados principales: el análisis, el diseño y la implementación de los diferentes bloques de trabajo.

4.1. Análisis

A continuación, se definen los usuarios que utilizarán nuestra aplicación, así como sus requisitos funcionales y los requisitos no funcionales de ésta.

4.1.1. Usuarios

Existen 2 perfiles de usuarios que utilizarán la aplicación.

1. **Jugador:** persona que va a jugar al juego.
2. **Analista de datos:** se encarga de estudiar el comportamiento de los jugadores en base a los datos que se recogen. A partir de ese estudio se podrá ir mejorando la aplicación.

4.1.2. Requisitos de usuario

4.1.2.1. Requisitos funcionales

- RUF_1. El jugador debe poder registrar una cuenta e iniciar sesión en ella.
- RUF_2. El jugador debe poder interactuar con los elementos de la sala utilizando sus manos.
- RUF_3. El jugador debe poder moverse tanto utilizando el seguimiento corporal que ofrece el equipo de VR, como con unos controles utilizando los mandos.
- RUF_4. El jugador debe poder ver algún elemento que identifique a los otros jugadores.
- RUF_5. El jugador debe poder elegir en qué escape room quiere jugar la partida.
- RUF_6. El jugador debe poder ver, de cada escape room, el nombre, la dificultad, su mejor tiempo y el número de veces que ha jugado.
- RUF_7. El jugador debe poder elegir entre crear él una partida, o unirse a una partida que ha creado otro jugador.
- RUF_8. El jugador que cree la partida debe poder indicar el momento en el que empieza la partida y, por tanto, empieza a correr el tiempo.
- RUF_9. El analista de datos debe poder consultar los datos de la base de datos.
- RUF_10. El analista de datos debe poder modificar la página web donde se realizan y se muestran los resultados de los análisis.

4.1.2.2. Requisitos no funcionales

- RUNF_1. El juego se debe poder jugar con un equipo de VR.
- RUNF_2. El juego se debe poder jugar multijugador.
- RUNF_3. Las partidas deben durar de media entre 10 y 15 minutos.
- RUNF_4. La gestión de las cuentas debe hacerse de manera segura.
- RUNF_5. Todas las pruebas del escape room las debe poder solucionar un solo jugador.
- RUNF_6. El escape room debe contener algún elemento dinámico.
- RUNF_7. El juego ha de ser funcional para PC.

4. Desarrollo del proyecto

4.2. Diseño

Previo a la implementación es importante realizar el diseño de la aplicación para definir qué componentes la conforman, como se relacionan entre sí e incluso como algunos se van a mostrar de manera gráfica.

4.2.1. Diseño de la arquitectura

En este apartado vamos a explicar los diferentes componentes que tendrá la aplicación y como se comunican entre sí. La arquitectura se puede ver de manera visual en la Figura 4.

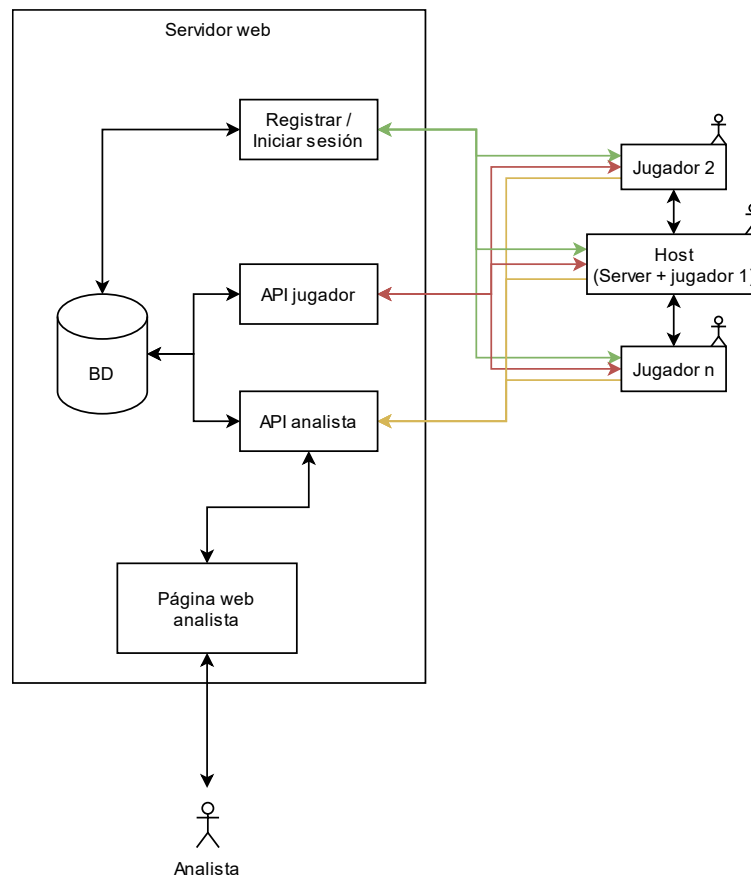


Figura 4. Arquitectura de la aplicación

En la parte del servidor hay varios elementos:

1. La base de datos.
2. Un archivo para registrar usuarios y otro para que éstos puedan iniciar sesión.
3. Una API (API jugador) que recibe las peticiones de los jugadores realizadas desde el videojuego.
4. Una página web dónde se muestran diferentes análisis sobre los datos de los jugadores.
5. Una API (API analista) que recibe peticiones de la página web que muestra los análisis de los datos. Esta API también se encarga de recibir los datos sobre el posicionamiento de los jugadores.

Por otro lado, están los jugadores, dónde en una partida multijugador, uno de ellos hace de host. Esto implica que hace el papel de servidor y de cliente. El resto de los jugadores son simplemente clientes.

Al iniciar el juego, los jugadores deben registrarse si no tienen cuenta e iniciar sesión (relación verde de la Figura 4). Una vez dentro, los jugadores deben elegir si quieren hostear una partida o si se quieren unir a una partida que ha creado otro jugador. Si se quiere jugar solo, se debe crear una partida como host. Si un jugador va a hostear una partida, debe elegir el escape room que quiere jugar. A la hora de elegir escape room, se muestran para cada uno, datos como la dificultad de éste, el mejor tiempo del jugador (si ha jugado anteriormente) y el número de veces que ha jugado.

Por tanto, como todos los jugadores tienen la opción de hostear una partida, deben poder elegir que escape room jugar. Esto implica que todos los jugadores necesitan obtener los datos mencionados anteriormente de cada escape room. Esto se consigue haciendo una petición a “API jugador”. Se representa con la relación roja en la Figura 4.

Al acabar la partida, el host envía al servidor, mediante una petición a “API jugador”, los tiempos en los que se ha resuelto cada prueba y los usuarios que han jugado esa partida. Esta acción también se representa con la relación roja de la Figura 4.

Al final de cada partida todos los jugadores también envían al servidor, mediante una petición a “API analista”, la posición por la que se han ido moviendo. Se representa con la relación amarilla en la Figura 4.

4.2.2. Diseño de las pantallas del juego

Previo al desarrollo de las diferentes pantallas del juego hemos diseñado tanto la lógica de éstas, como el aspecto visual que tendrán. Para representar la lógica hemos hecho un diagrama de estados, donde cada estado representa una pantalla. Este diagrama se puede ver en la Figura 5.

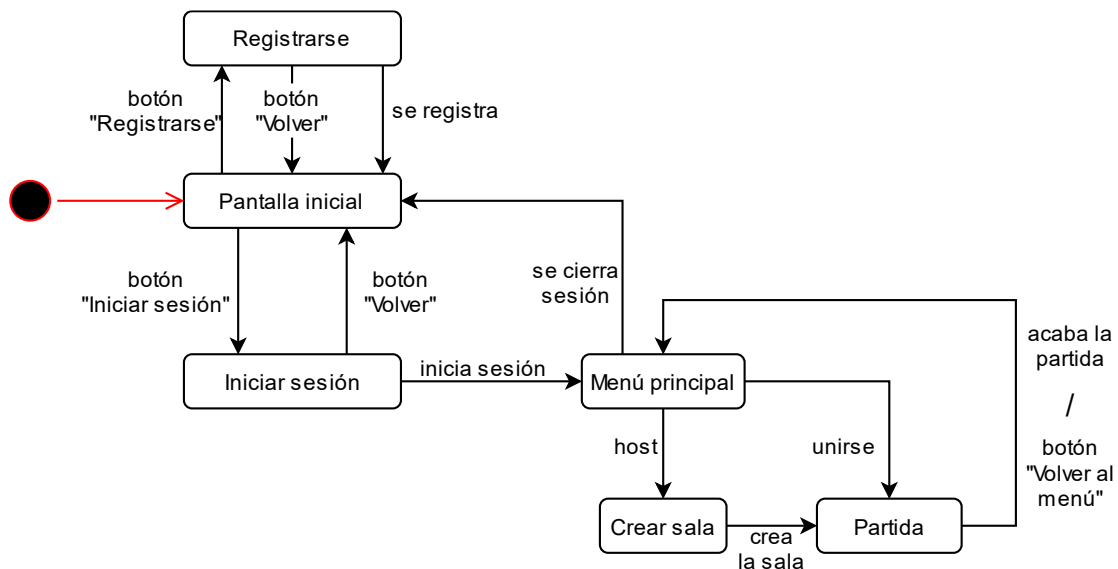


Figura 5. Lógica de las pantallas de juego

A continuación, explicamos en orden las diferentes pantallas, así como un diseño de la interfaz que hemos creado utilizando Adobe XD [13].

Al iniciar el juego, se muestra una pantalla como la de la Figura 6 (izquierda).

4. Desarrollo del proyecto



Figura 6. Diseño de la pantalla de inicio (izquierda), de la pantalla de registro (centro) y de la pantalla de inicio de sesión (derecha)

Si le damos al botón de “Registrarse” aparece una pantalla como la de la Figura 6 (centro). En esta pantalla, si rellenamos los campos y se crea el usuario correctamente, nos redirige a la pantalla de inicio (Figura 6 izquierda).

Si por el contrario pulsamos el botón de “Iniciar sesión” de la pantalla de inicio, aparece una pantalla como la de la Figura 6 (derecha). En esta pantalla, si rellenamos los campos e iniciamos sesión correctamente, nos redirige al menú principal. Este se muestra en la Figura 7.



Figura 7. Diseño de la pantalla de menú

Una vez en el menú, nos aparecen los diversos escape rooms disponibles junto con su dificultad, el mejor tiempo y el número de intentos que hemos hecho. Si le damos al botón de “Cerrar sesión” nos redirige a la pantalla de inicio (Figura 6 izquierda).

Desde el menú, si decidimos hostear una partida (o si queremos jugar solos, que haremos de host igualmente) cargará la escena de Unity del escape room correspondiente y nos aparecerá un panel como el de la Figura 8 (izquierda).

Entonces elegimos una contraseña y le damos al botón de crear sala. En ese instante la pantalla cambiará y aparecerá un botón para empezar la partida como se aprecia en la Figura 8 (derecha). Ahora ya se podrán unir otros jugadores a nuestra partida.

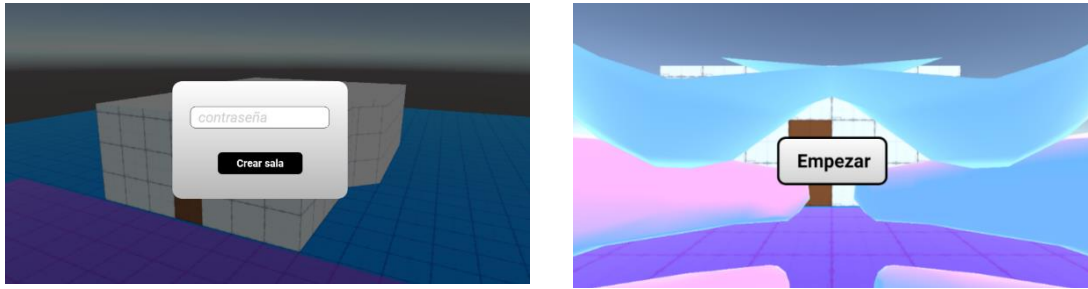


Figura 8. Diseño de la pantalla para crear sala (izquierda) y la pantalla para empezar partida (derecha)

Al darle al botón de “Empezar” comenzará a correr el tiempo y se podrá entrar en la primera sala del escape room.

Si por el contrario queremos unirnos a una partida que va a hostear otro jugador, deberemos especificar la dirección IP del host y la contraseña de la partida desde el panel inferior del menú (Figura 7). Al unirnos a una partida nos saldrá directamente la pantalla de juego con el botón de “salir de la partida” en la parte inferior izquierda de la pantalla.

Tanto si somos host como si somos clientes, al resolver la prueba final volveremos al menú (Figura 7).

4.2.3. Diseño de la base de datos

Para la base de datos utilizaremos una de tipo relacional ya que, aparte de ir bien para esta aplicación en concreto, es la que hemos visto durante la carrera y la que yo personalmente se utilizar.

El modelo de datos necesario para guardar toda la información que se utiliza tanto dentro del juego, como en el apartado de análisis, es el que se muestra en la Figura 9.

Para cada escape room, a parte de su identificador necesitamos guardar su nombre y su dificultad ya que estos datos se mostrarán dentro del juego a la hora de elegir que escape room jugar.

Cada escape room puede tener varias temáticas ya que, por ejemplo, uno podría ser de ciencia ficción y de terror. Y más de un escape room pueden tener la misma temática.

Un escape room está formado por una o varias salas, dónde cada sala puede no contener o contener varias pruebas. Una prueba pertenece a una sola sala, en concreto a la sala en que se resuelve la prueba. Y una sala pertenece a un escape room en concreto.

Por otro lado, de cada escape room se jugarán múltiples partidas. De estas partidas nos interesa saber la fecha en que se realizaron. De cada partida también guardaremos el tiempo que se tarda en realizar cada prueba de las que hay en ese escape room. El tiempo en resolver una prueba será el tiempo que pase desde que empieza la partida hasta que se resuelve la prueba.

4. Desarrollo del proyecto

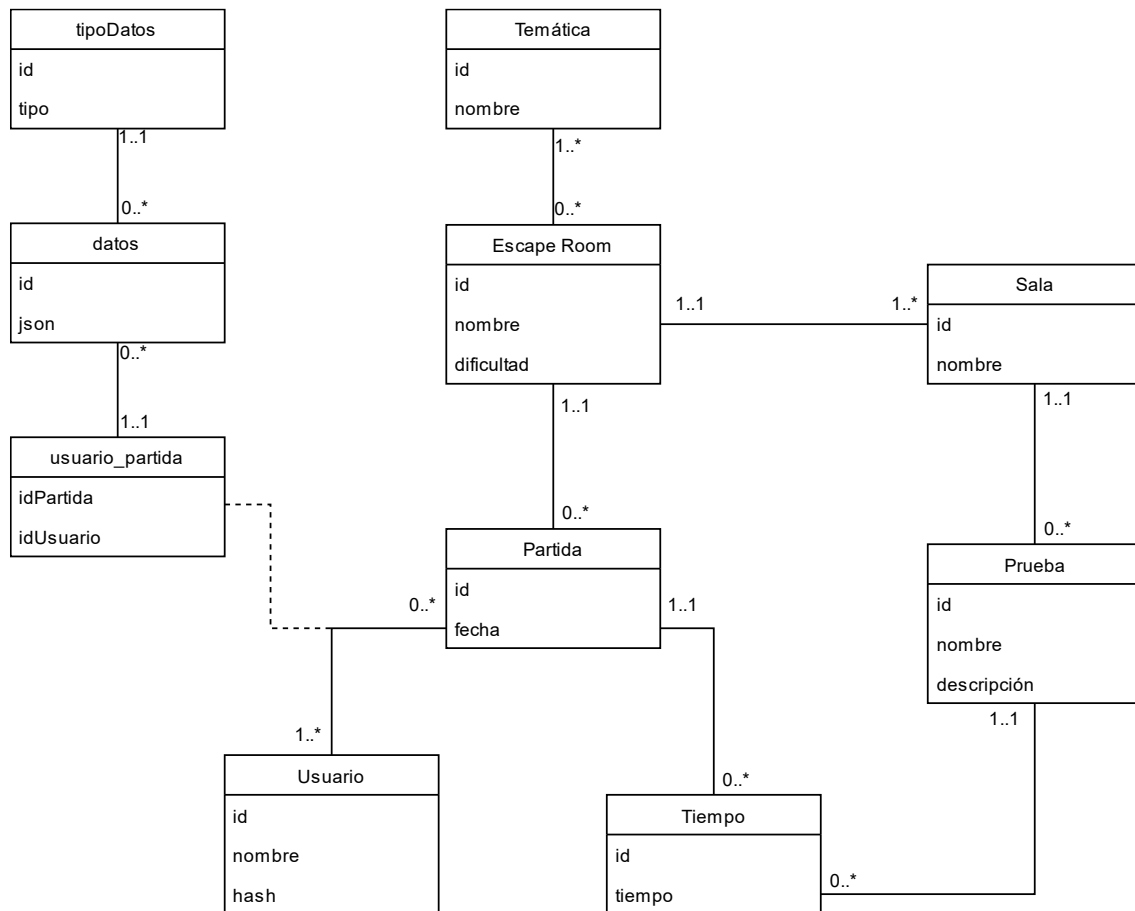


Figura 9. Modelo de la base de datos

De cada partida también nos interesa saber los jugadores que han participado, que pueden ser uno o varios. Esta información se guardará en la relación “usuario_partida”. Para que esto sea posible necesitaremos guardar la información de los jugadores. Para ello tenemos la tabla “usuario” con la que también podremos gestionar el inicio de sesión de los jugadores.

Finalmente, para poder guardar de manera ordenada los datos que se utilizan para hacer los análisis, se necesitan las tablas “datos” y “tipoDatos”. Cada usuario de cada partida podrá guardar varios JSON. Cada JSON es de un tipo en concreto. Nosotros únicamente vamos a enviar un JSON con la posición del jugador cada 250ms, por tanto, solo tendremos un tipo de JSON. Pero si, por ejemplo, se quisiese hacer un análisis de cuando se cogen y sueltan los objetos de la sala, se podría crear otro tipo de datos llamado “interacción con objeto” y crear un JSON diferente con los datos correspondientes.

4.2.4. Diseño del escape room

En este apartado no solo se muestra el diseño de las diferentes pruebas y el flujo del juego, sino que también se definen una serie de conceptos que se aplican al diseño de cualquier escape room, no solo al que nosotros hemos desarrollado.

4.2.4.1. Proceso estandarizado

Pese a que cada escape room es diferente, es importante definir un proceso que se vaya a seguir siempre a la hora de desarrollar uno. De esta manera, si en un futuro se quiere añadir una nueva

sala a la aplicación, habrá unos pasos a seguir establecidos que garanticen, o intenten garantizar, la creación de un escape room de calidad. El proceso que hemos definido está representado en la Figura 10.

Primero de todo es importante establecer una temática y una historia, aunque sea simple, para que el escape room tenga un sentido.

A la hora de elegir las pruebas, hay que tener en cuenta que pueden ser de 5 tipos diferentes:

- **Lógica:** a partir de una información que ha obtenido el jugador, este debe ser capaz de llegar a una conclusión (de manera lógica) que le permita avanzar en el juego.
- **Habilidad:** la dificultad reside en la habilidad psicomotriz del jugador. Por ejemplo, resolver un cubo de Rubik.
- **Conocimiento:** en base a una información que obtiene el jugador, este necesita de unos conocimientos (cultura general) para avanzar en el juego.
- **Matemáticas:** se necesita resolver una operación matemática para avanzar.
- **Buscar:** buscando por sitios escondidos de la sala obtienes objetos, pistas, etc.

Las pruebas se componen de 3 fases.

- **Descubrimiento:** el jugador descubre la existencia de la prueba.
- **Solución:** el jugador soluciona la prueba.
- **Transición:** al resolver la prueba, el jugador obtiene información o algún elemento que le hace descubrir una nueva prueba o que le ayuda a resolver una prueba que ya había descubierto. En algunos casos, la transición de una prueba se podría considerar parte del descubrimiento de otra.

Como se puede apreciar en el diagrama de flujo de la Figura 10, existen pruebas principales y pruebas secundarias.

- **Principales:** conforman la historia que se definió al principio.
- **Secundarias:** son añadidos a las pruebas principales con la intención de que el escape room sea más difícil, tenga una mayor variedad de pruebas, sea más largo, etc.

Teniendo esto en cuenta, es una buena idea primero diseñar y hacer un prototipo de las pruebas principales. De esta manera podemos comprobar si la base del escape room es sólida y estamos conformes con ella. Una vez creamos que las pruebas que conforman la historia principal están bien pensadas y se pueden llevar a cabo, podemos pasar a definir las pruebas secundarias. Habrá más o menos pruebas secundarias en función del tiempo, dificultad, etc. que queramos que tenga la sala.

4. Desarrollo del proyecto

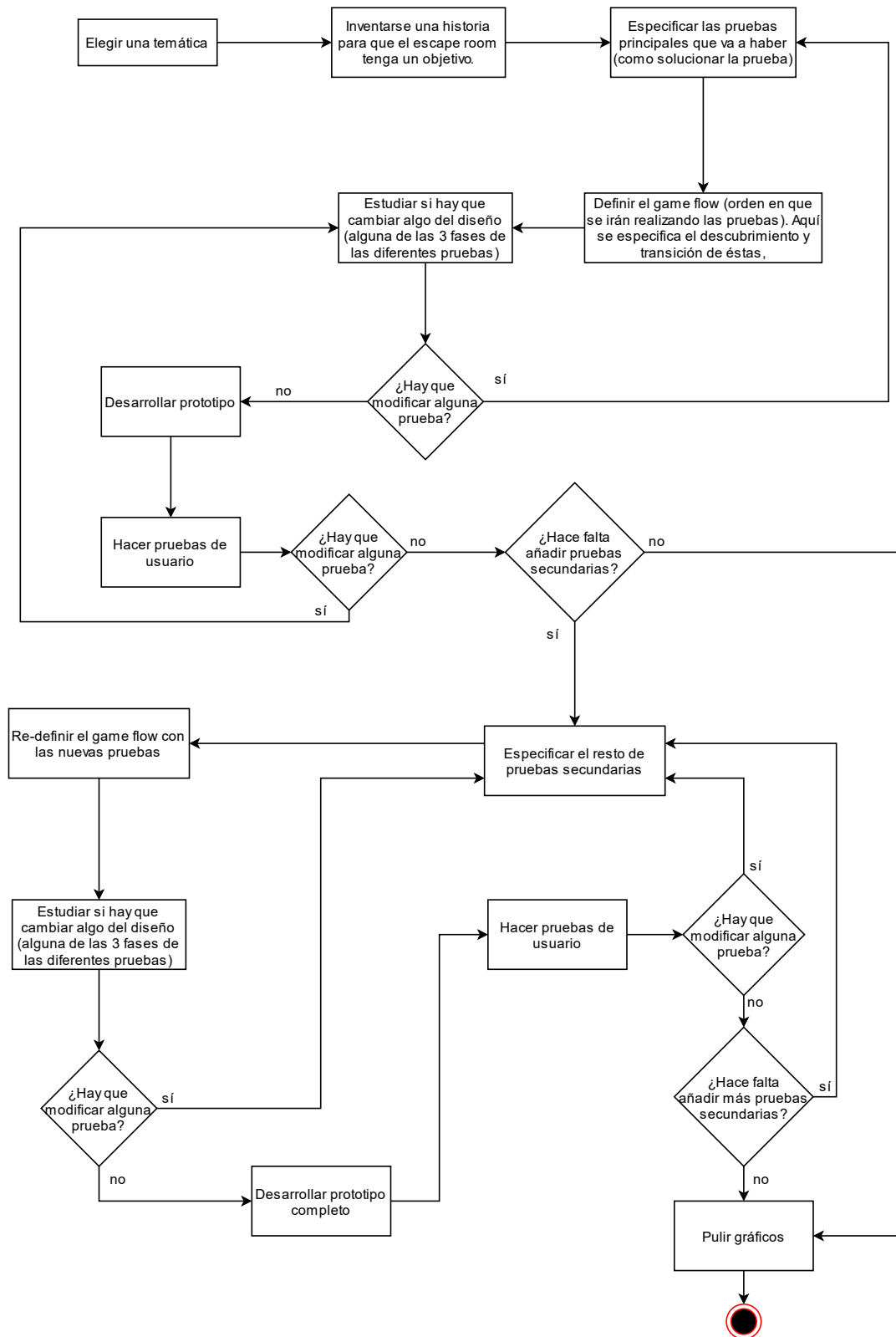


Figura 10. Proceso de diseño de un escape room

4.2.4.2. Niveles de dinamismo

Uno de los requisitos del juego, en concreto el requisito RUNF_6, es que tenga algún componente dinámico para añadirle un poco más de complejidad y hacer algo un poco diferente a lo estándar. De esta manera se puede conseguir que cada partida sea diferente.

Entonces, es importante definir los diferentes niveles de dinamismo que puede tener el juego. Aquí están ordenados de más simple a más complejo:

1. Todos los elementos son estáticos. Por tanto, el escape room es siempre el mismo: mismas habitaciones, mismas pruebas, mismos códigos, etc.
2. Ciertos objetos (que contienen, o no, pistas para completar el juego) pueden estar en diversos sitios de la sala. Pese a que estos posibles sitios están predefinidos, la elección a la hora de iniciar el juego es aleatoria de entre el conjunto definido.
3. Los códigos necesarios para abrir candados se generan de manera aleatoria y algunos objetos tendrán que cambiar en función de los códigos que se van a utilizar en cada partida.
4. Las salas que componen un escape room son aleatorias. Es decir, se crean salas independientes y se escogen de manera aleatoria las salas que habrá en cada partida.
5. Se define conjunto de pruebas relacionadas entre sí, de tal manera que en cada partida se escojan, de manera aleatoria, un subconjunto del total de pruebas.

El cuarto nivel, pese a que posibilita el que un jugador pueda jugar un escape room más de una vez, tiene un gran inconveniente. Si las salas que conforman un escape room se escogen de manera aleatoria, implica que las salas deben ser independientes entre sí. Por tanto, imposibilita (o complica de manera significativa) el tener elementos (objetos, pistas, etc.) en la sala n , que se vayan a utilizar en la sala $n+1$.

Además, parte de la gracia de un escape room es que tenga una temática y una historia y que, por tanto, los pasos que vayan siguiendo los jugadores tengan un sentido/objetivo y esté todo relacionado.

Para el nivel 5 no se requiere la implementación del nivel 4 y por tanto sí que sería posible hacerlo manteniendo el sentido de la historia. El problema es que se debería definir una serie de etapas del juego donde para cada etapa haya un conjunto de pruebas. Entonces, de cada etapa se escogería una prueba la cual debería poder enlazarse con la prueba de la etapa siguiente. Esto supone una carga de trabajo mucho mayor, que no va con el objetivo de este TFG.

Así pues, hemos decidido adoptar el nivel tres. Para el escape room que hemos desarrollado, las salas, pruebas y ambientación general serán elementos estáticos. Pero, tanto los códigos que se requieren para resolver las pruebas, como las pistas a partir de las cuales los jugadores obtienen los códigos, como la posición de algunos elementos necesarios para resolver el juego, serán aleatorios.

4.2.4.3. Pruebas y flujo del juego

Como viene indicado en el proceso de la Figura 10, previa a la definición de las pruebas se debe establecer una temática y una historia. En el caso de nuestro escape room serán:

- **Temática:** la historia transcurre en un laboratorio secreto.

4. Desarrollo del proyecto

- **Historia:** nos ha llegado información de que un equipo de científicos está haciendo pruebas en un laboratorio, de manera ilegal y secreta, con un alien que capturaron en medio de la montaña la semana pasada. Nosotros, como agentes de la CIA, debemos infiltrarnos en el laboratorio, encontrar el alien y llevárnoslo para utilizarlo y experimentar nosotros con él.

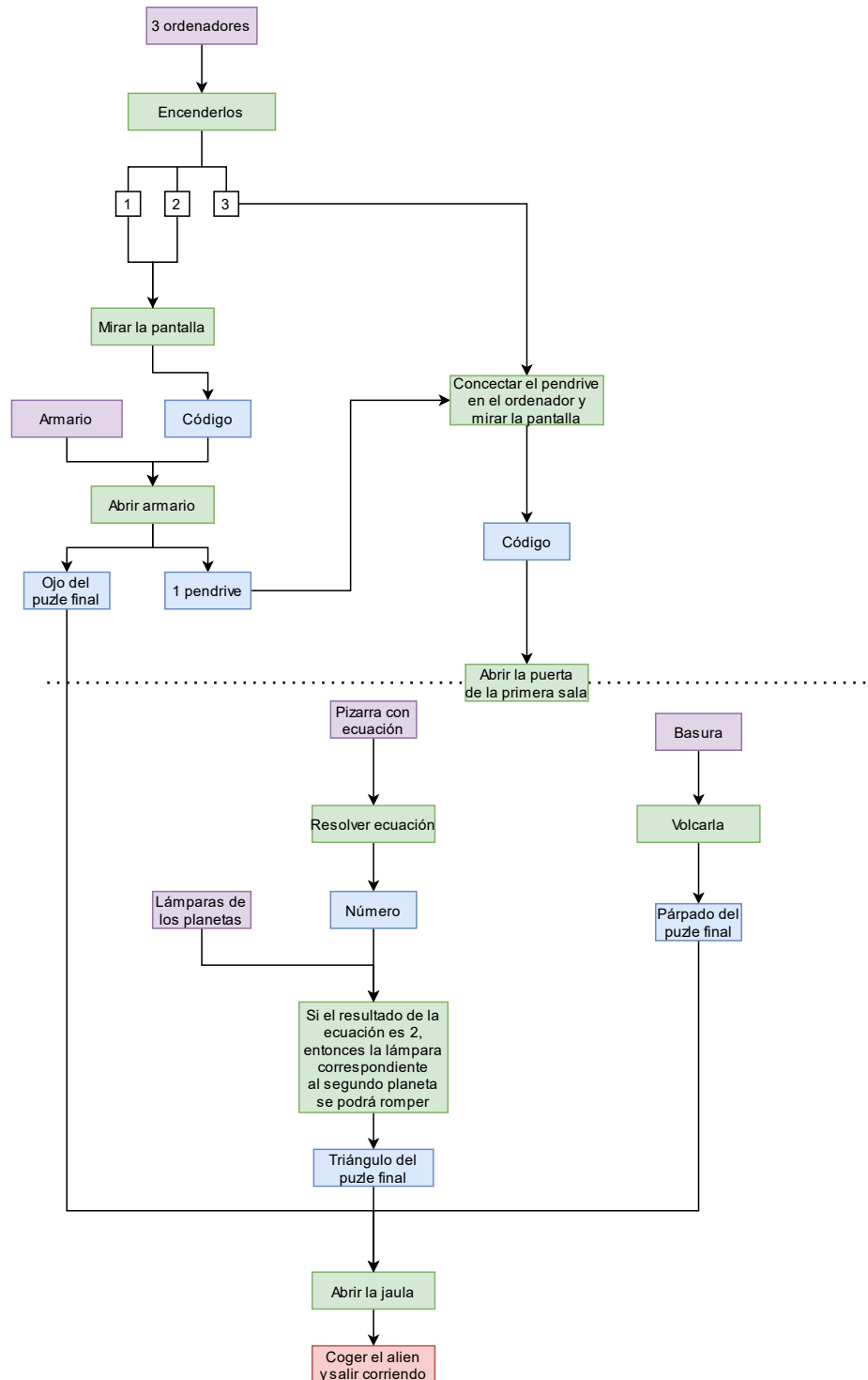
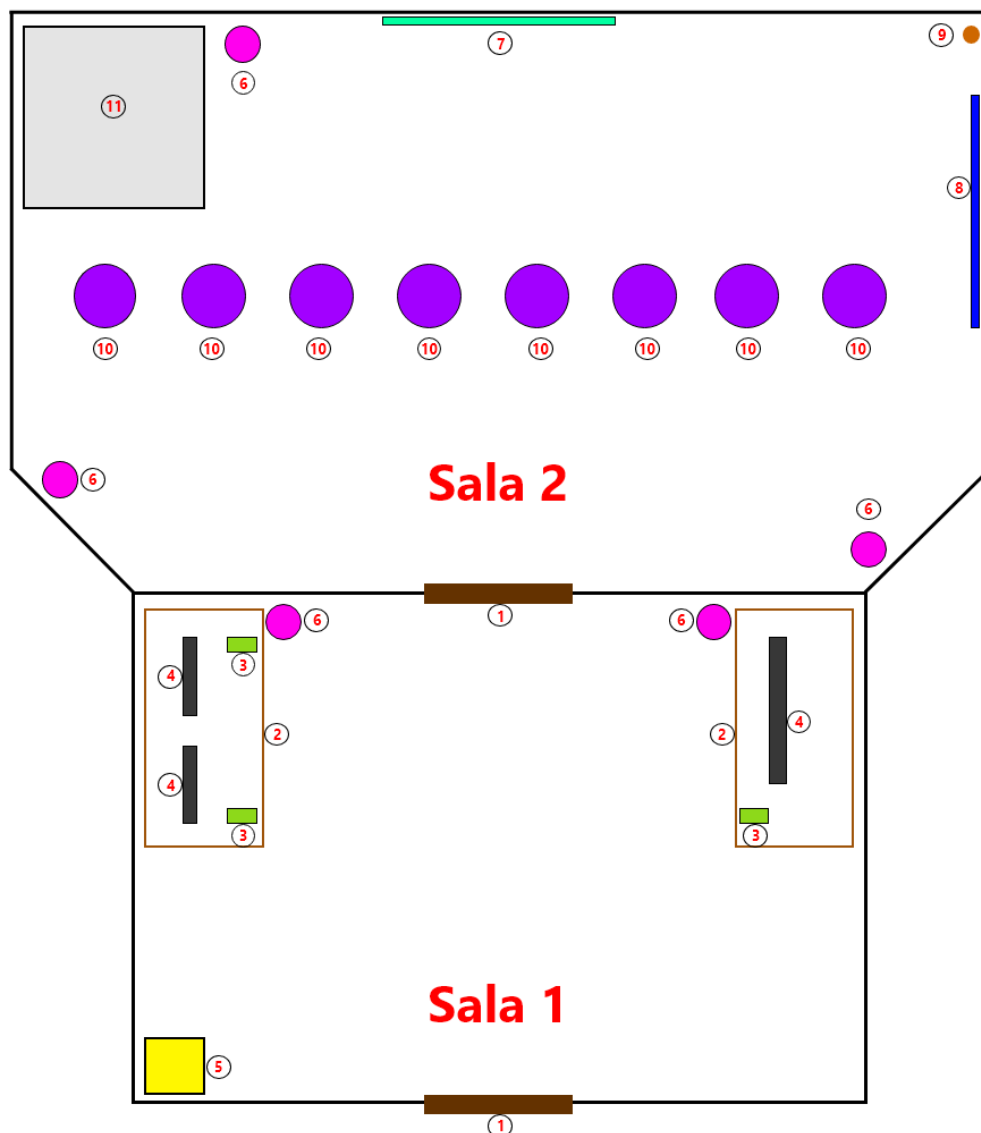


Figura 11. Pruebas y flujo de juego del escape room

En la Figura 12 aparecen todos los objetos necesarios para completar el juego. A continuación, se irán referenciando para explicar las diferentes pruebas. La expresión: (X), siendo “X” un número, hace referencia a un objeto de los que aparecen en la Figura 12.



Este escape room se compone de 4 pruebas:

4. Desarrollo del proyecto

4.2.4.3.1. Prueba 1

El escape room consta de dos salas. Al entrar en la primera sala encontramos, entre otros objetos, tres ordenadores (3). Si los encendemos, en la pantalla (4) de dos de ellos aparece un informe y en otro aparecerá una imagen representando un fondo de escritorio. Los dos informes contienen un “número de código” de dos cifras.


El informe del primer ordenador es como el de la Figura 13 y el informe que aparece en el segundo ordenador es como el de la Figura 14. Juntando los dos “números de código”, que están representados como “XX” en la Figura 13 y en la Figura 14, obtenemos el código para abrir un armario (5) que se encuentra en la primera sala.

El número de 4 cifras que desbloquea el armario se genera de manera aleatoria. Por tanto, los números que aparecen en los informes se deben de poner en función del código del armario. Dentro de éste hay un pendrive y un triángulo. Al desbloquear el armario finaliza la primera prueba y se guarda el tiempo.

Autopsia alien

Nº código	XX
Lugar de la autopsia	Casa del norte, 47°27'07.1"N 107°48'04.3"W
Fecha de la autopsia	25/04/2008

Se ha realizado la autopsia del alien encontrado en el estado de Montana, EE.UU. Posee una estructura interna similar a la del cuerpo humano, pero con un cráneo y cerebro notablemente más grande.



Altura	254 cm
Peso	120 kg
Edad	235 años

Imagen de la autopsia

Figura 13. Imagen que aparece en la pantalla 1

Avistamiento OVNI

Nº código	XX
Lugar del avistamiento	Gran Cañon
Fecha del avistamiento	17/09/1999

El pasado martes 17 las fuerzas aéreas de los Estados Unidos avistaron un OVNI en la parte oeste del Gran Cañón del Colorado.

El objeto viajaba a unos 8.000 kph y parecía propulsarse por cohetes o algún motor de combustión similar. Pese a la velocidad a la que iba tenía una agilidad como si de un dron pequeño se tratase.



Imagen del avistamiento

Figura 14. Imagen que aparece en la pantalla 2

4.2.4.3.2. Prueba 2

La segunda prueba consiste en abrir la puerta para entrar a la segunda sala (1). Para ello, se debe de conectar el pendrive al tercer ordenador. Si el ordenador está encendido se mostrará una

imagen como la de la Figura 15. “X”, “Y”, “Z” y “W” son los números que forman el código para desbloquear la puerta y que también han sido generados de manera aleatoria al empezar la partida.

En la puerta para entrar a la segunda sala hay un grafiti como el de la Figura 16. Esta imagen nos indica en que orden hay que poner los números que aparecen en la Figura 15. Por tanto, si introducimos el número “X Y Z W” se desbloquea la puerta y se guarda el tiempo que se ha tardado en resolver la prueba.

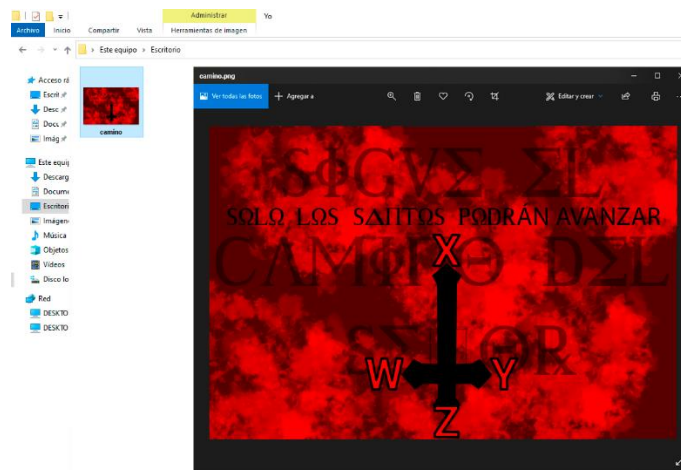


Figura 15. Imagen que contiene el pendrive



Figura 16. Instrucción que aparece en la puerta para entrar en la sala 2

4.2.4.3.3. Prueba 3

En la segunda sala hay 8 lámparas colgando del techo (10), cada una con la textura de un planeta del sistema solar. También hay, entre otras cosas, una pizarra (7), un poster con una imagen del sistema solar (8) y un puntero (9) para señalar la pizarra. En la pizarra aparecerá la frase “Destroy planet X” y justo al lado aparecerá la ecuación $ax + bx + cx - (b+c)x + x = 2((1/2)a)x + \{rnd\ 1 - 8\}$. El último componente de la ecuación será un número del 1 al 8 generado de manera aleatoria. La ecuación está pensada de tal manera que siempre quede $x = \{rnd\ 1 - 8\}$. Entonces solo una de las lámparas se podrá romper en función del número generado de manera aleatoria. Para romper la lámpara es necesario coger el puntero de la pizarra. Al romper la lámpara obtendremos un nuevo objeto, un párpado de un ojo, y se guardará el tiempo que hemos tardado en resolver la prueba.

4. Desarrollo del proyecto

4.2.4.3.4. Prueba 4

Esta es la prueba final. Consiste en colocar sobre el candado de la jaula (11) un triángulo, un parpado y un ojo tal y cómo se indica en la Figura 17. Al realizar la prueba 1 se obtiene el triángulo, al realizar la prueba 3 se obtiene el párpado y para conseguir el ojo, debemos buscar en una de las papeleras (6). Una vez se colocan los tres elementos, se guarda el tiempo final y se termina la partida.

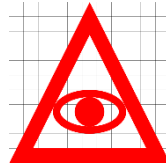


Figura 17. Colocación de los tres elementos en el candado final

4.2.5. Diseño de la página web de los analistas

La página web tiene una interfaz muy simple, como la de la Figura 18.

Análisis del comportamiento de los jugadores

Escape room:

Partida:

Jugador:

mapa de calor

Figura 18. Interfaz de la página web de análisis

Al entrar en esta página web únicamente se puede seleccionar un escape room. Al seleccionarlo se muestra un mapa de calor utilizando las posiciones de todos los jugadores en todas las partidas de ese escape room. Entonces aparece la opción de elegir una partida. Al seleccionar una partida se genera el gráfico de nuevo, pero utilizando únicamente los datos de todos los jugadores de esa partida. También aparece la opción de elegir un jugador de los que han participado en esa partida. Al elegir uno, se actualiza el mapa de calor con los datos de ese jugador en esa partida.

Al ir cambiando los elementos seleccionados se va actualizando la página web de manera dinámica. Además, si por ejemplo se selecciona que no se quiere ver la información de ningún escape room, se ocultan las opciones de elegir partida y jugador. En la Figura 19 se muestra un diagrama de estados para entender mejor como aparecen los elementos de manera dinámica.

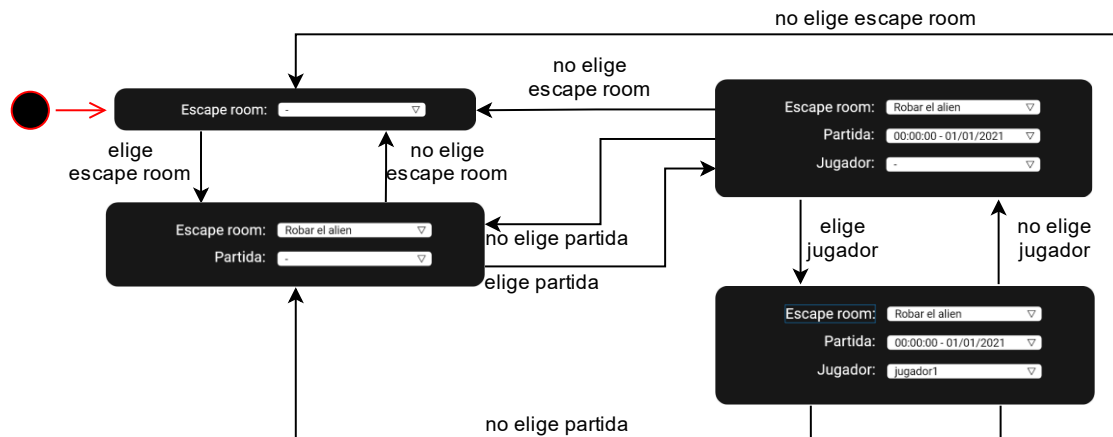


Figura 19. Diagrama de estados del menú de selección de la página web

4.3. Implementación

La explicación del desarrollo de los diferentes componentes de la aplicación la vamos a realizar en el mismo orden que los desarrollamos. Este orden es:

1. Desarrollo del escape room
2. Integración con la base de datos (sin la parte del almacenamiento de los datos para analizar)
3. Multijugador
4. Análisis de los datos (incluye el envío y la gestión de estos en la base de datos)

Previo a esta explicación, vamos a introducir como es el funcionamiento de Unity de manera general con tal de que la posterior explicación técnica se entienda mejor.

Cabe destacar que en este apartado se ha incluido código principalmente de Unity ya que se encuentra muy disperso en diferentes scripts. En la mayoría de los casos, el código de los archivos que se encuentran en el servidor se ha obviado ya que se pueden consultar fácilmente desde el repositorio: https://github.com/gserrrr/Archivos_TFG-Sergio_Garcia_Puertas. En este repositorio también se puede encontrar el código de Unity, pero lo importante de éste aparece en este documento.

4.3.1. Funcionamiento de Unity

A la hora de crear un juego en Unity, se crean diferentes escenas dónde cada escena contiene sus objetos. Los objetos tienen asignado una etiqueta y una máscara que se utilizan para establecer interacciones específicas entre objetos. Estos objetos están formados por componentes. Además, puedes añadirle scripts como componentes con tal de añadirle una funcionalidad o comportamiento extra al objeto.

Los scripts por defecto heredan de la clase "MonoBehaviour" [14]. Esto hace que hereden una serie de métodos cuyo comportamiento ya está definido. Entre los más destacables están el método "Start", que se ejecuta antes del primer fotograma, y el método "Update", que se ejecuta antes de cada fotograma.

En la Figura 20 podemos ver cómo es el editor de Unity. Con el número (1) está indicado el panel donde aparecen todos los objetos que conforman la escena actual. Si cambiásemos de escena, este panel se actualizaría con los objetos presentes en la nueva escena.

4. Desarrollo del proyecto

Con el número (2) se indica la pantalla donde, entre otras, tenemos las pestañas de “Scene” y “Game”. En la pestaña de Scene podemos editar los objetos de manera gráfica. En la pestaña de Game podremos jugar al juego una vez le demos al botón de play.

Al hacer click en un objeto nos aparecerán, en el panel marcado con el número (3), todos los componentes por los que está compuesto ese objeto. Y como hemos mencionado anteriormente, si ese objeto tiene algún script asociado, aparecerá como un componente más. Durante la posterior explicación del desarrollo de las pruebas se explican los diferentes componentes que vayan apareciendo.

Finalmente, en la zona señalada con el número (4) se encuentran todos los activos del proyecto, ya sean materiales, objetos, archivos, etc.

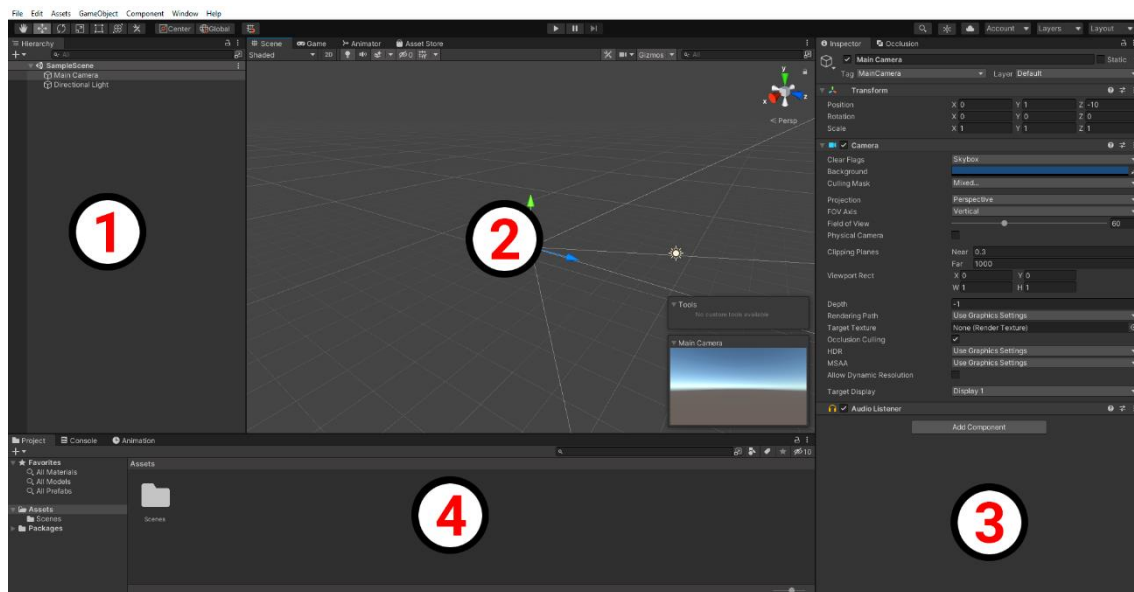


Figura 20. Editor de Unity

4.3.2. Selección de herramientas

El requisito RUNF_1 define que se debe poder jugar con un equipo de realidad virtual. Las dos opciones más populares que hay para implementar la realidad virtual en Unity son el paquete **SteamVR** [15] externo de Unity y el paquete **XR Interaction Toolkit** [16] propio de Unity.

Ambos paquetes tienen las funcionalidades básicas y necesarias para el desarrollo de este juego, que son el coger los inputs del jugador (mediante el equipo de VR) y el manejar las interacciones entre los diferentes objetos del juego.

La principal ventaja que tiene el paquete de SteamVR es que se lleva desarrollando desde hace bastante tiempo y por tanto está ya en una versión avanzada y estable. Por otro lado, el paquete propio de Unity va por la versión 0.10.0, con lo que aún es propenso a recibir cambios importantes en un futuro o a tener fallos que serán arreglados en un futuro.

En cualquier caso, como he dicho anteriormente, ambos cuentan con las funcionalidades básicas y necesarias para realizar la aplicación de este TFG. Pese a que SteamVR es más completo y a priori esto es una ventaja, yo nunca había utilizado un equipo de realidad virtual y menos desarrollado una aplicación con éste. Es por esto por lo que empezar a utilizar el paquete propio de Unity resulta más fácil que empezar con el de SteamVR.

Por esta razón, junto a que personalmente prefiero aprender a utilizar una herramienta que es más probable que vaya a utilizar en un futuro, hemos decidido utilizar el paquete XR Interaction Toolkit de Unity.

Otro aspecto a tener en cuenta a la hora de desarrollar el juego es la herramienta que se va a utilizar para construir el escenario. Tanto los elementos puramente decorativos, como los elementos con los que interaccionan los jugadores. En este caso también existen dos opciones principales, que son el software libre de modelado 3D **Blender** [17], y el paquete de Unity **ProBuilder** [18].

Para este juego, los gráficos no son una parte muy importante ya que el objetivo de este TFG reside en crear un juego con una arquitectura similar a la que tendría un juego que vaya a salir al mercado. Con Blender se pueden conseguir unos detalles superiores a los que se pueden conseguir con ProBuilder. Pero al igual que ocurría en la comparativa anterior, pese a ser la opción más completa, también tiene una barrera de entrada superior a la que se necesita para utilizar ProBuilder. Además, al realizar un modelo en Blender, debes exportarlo a Unity y por tanto estar constantemente trabajando con los dos programas por separado.

Por estos motivos hemos decido utilizar únicamente ProBuilder como herramienta de modelado 3D. En cualquier caso, en el Marketplace de Unity podemos encontrar recursos gratuitos que nos ayuden a ambientar nuestro juego.

4.3.2.1. XR Interaction Toolkit

Para posteriormente entender la explicación de las pruebas y de cómo es posible jugar con el equipo de VR, es necesario explicar varios conceptos sobre este paquete de Trabajo.

Lo primero que es necesario entender es que hay dos tipos de objetos (de los que tienen algún componente de VR). Estos son los objetos denominados **interactors** y los denominados **interactables**.

Los interactors son objetos capaces de interactuar con objetos interactables. En la Figura 21 podemos ver 3 objetos: la mesa, las manos y el cubo. En este caso la mesa no tiene ningún componente de VR, es decir, no se puede interactuar con ella. Por tanto, no es ni un interactor ni un interactable.

Las manos, como se aprecia en la Figura 22 (izquierda), tienen el componente **XR Direct Interactor**. El cubo, como se puede ver en la Figura 22 (derecha), tiene el componente **XR Grab Interactable**. Entonces, en este ejemplo, las manos (interactors) son capaces de interactuar con el cubo (interactable).

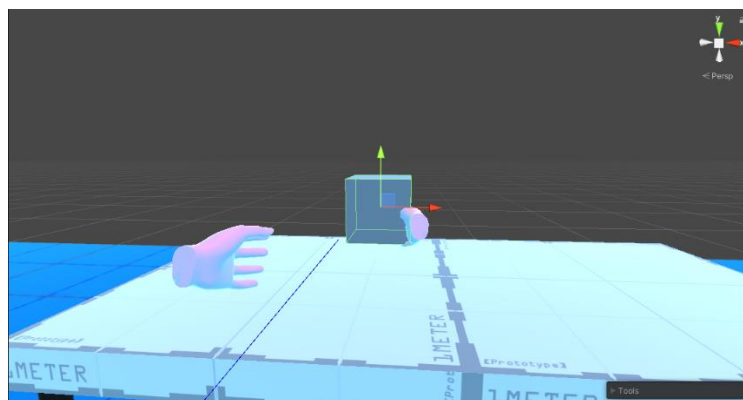


Figura 21. Ejemplo de interactor e interactable

4. Desarrollo del proyecto

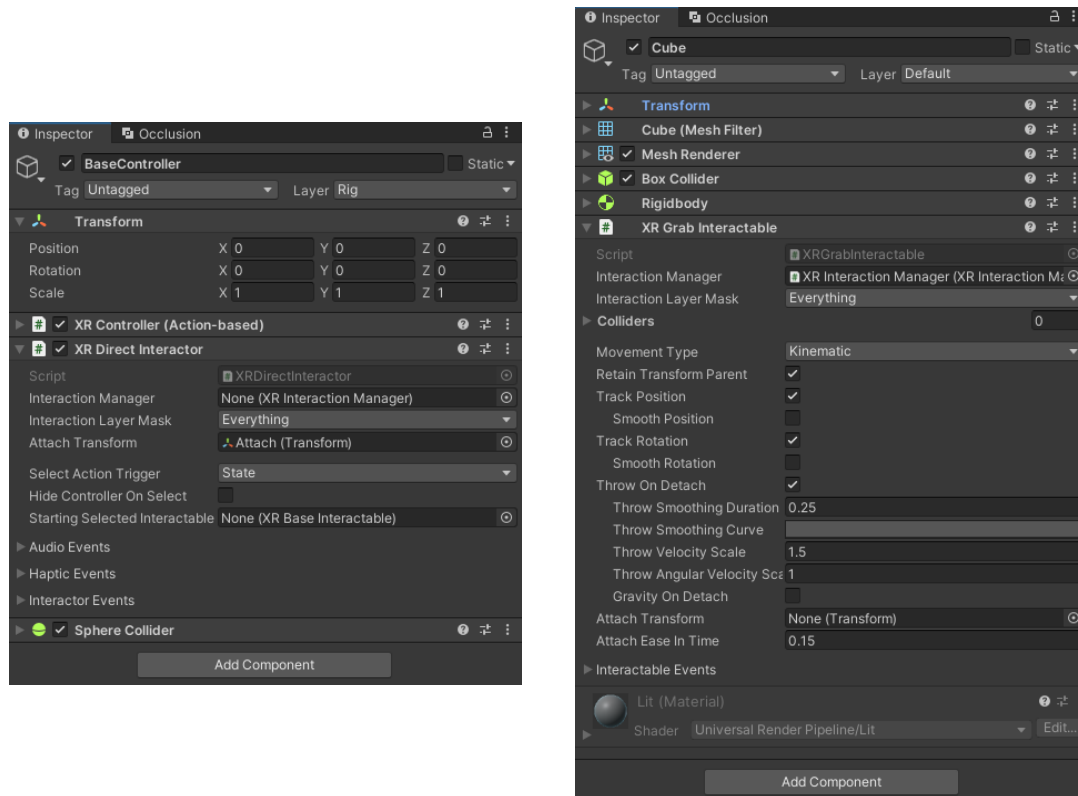


Figura 22. Componentes "XR Direct Interactor" (izquierda) y "XR Grab Interactable" (Derecha)

Los interactables pueden estar en 3 estados:

- **Hover:** se entra en este estado cuando su hitbox entra en contacto con la hitbox de un interactor. Es decir, si el interactor quisiese interactuar en ese instante, podría.
- **Select:** se entra en este estado cuando estando en el estado de hover, el interactor acciona el botón asignado para seleccionar el objeto. Por defecto el botón asignado es el de "Grip". Aparece en la Figura 23.
- **Activate:** se entra en este estado cuando estando en el estado de select, el interactor acciona el botón asignado para activar el objeto. Por defecto el botón asignado es el de "Trigger". Aparece en la Figura 23.

Los interactores pueden estar en 2 estados:

- **Hover:** se entra en este estado cuando su hitbox entra en contacto con la hitbox de un interactable.
- **Select:** se entra en este estado cuando el interactor, estando en el estado de hover, acciona el botón asignado para seleccionar un interactable.



Figura 23. Botones de los mandos de Oculus

4.3.3. Explicación técnica de cada prueba

Varios elementos de estas pruebas se inicializan de manera dinámica. Todas estas inicializaciones se hacen en el script “Inicialización.cs”, el cual se ejecuta al empezar la partida. Sin embargo, como las pruebas se explican por separado, iremos haciendo referencia a las diferentes partes de código de este archivo.

4.3.3.1. Prueba 1

La prueba 1 consta de los elementos que aparecen en la Figura 24: una mesa con dos ordenadores y un armario con un candado numérico.

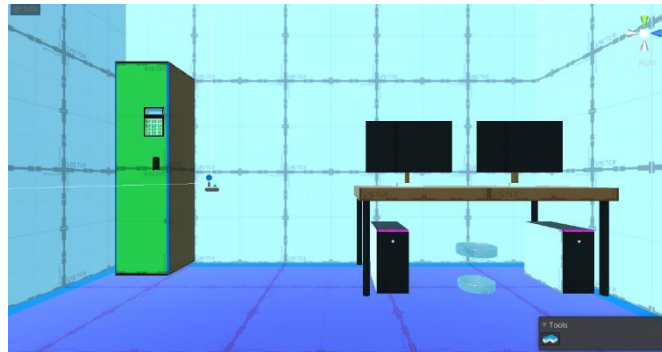


Figura 24. Elementos que componen la prueba 1

Para estos elementos se necesita generar un número aleatorio de 4 cifras, el cual será el código del candado del armario y que a su vez aparecerá como pista en los informes de los ordenadores. Esto se realiza con el siguiente código del script “Inicialización.cs”:

```
//generamos el número aleatorio para el candado del armario
for (int i = 0; i<4; i++){
    int num = UnityEngine.Random.Range(0, 10);
    codigoArmarioCreado += num.ToString();
}

//le asignamos este número tanto al candado del armario, como a las pistas
//que aparecen en los monitores
```

4. Desarrollo del proyecto

```
codigoArmario.setCodigoBueno(codigoArmarioCreado);  
numeroMonitorIzq1.text = codigoArmarioCreado.Substring(0, 2);  
numeroMonitorIzq2.text = codigoArmarioCreado.Substring(2, 2);
```

Ambos ordenadores cuentan con un botón el cual tiene un componente de tipo “Collider”. El botón se puede apreciar mejor en la Figura 25 (izquierda). A su vez, el dedo índice de la mano derecha del jugador tiene otro componente de tipo “Collider”, como se puede ver en la Figura 25 (derecha). Cuando el collider del dedo colisiona con el collider del botón del ordenador, se activan los siguientes eventos de tipo Trigger del script “OrdenadorEncendido.cs”:

```
private void OnTriggerEnter(Collider other){  
    if (other.CompareTag("Dedo") & !siendoPulsada){  
        siendoPulsada = true;  
        if (!estaEncendido){  
            estaEncendido = true;  
            seEnciende.Invoke();  
        }else{  
            estaEncendido = false;  
            seApaga.Invoke();  
        }  
    }  
}  
  
private void OnTriggerExit(Collider other){  
    if (other.CompareTag("Dedo")){  
        siendoPulsada = false;  
    }  
}
```

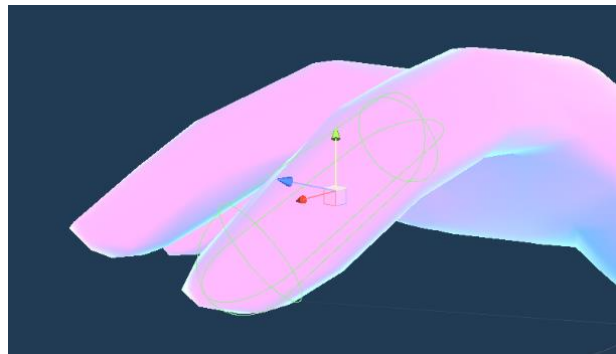
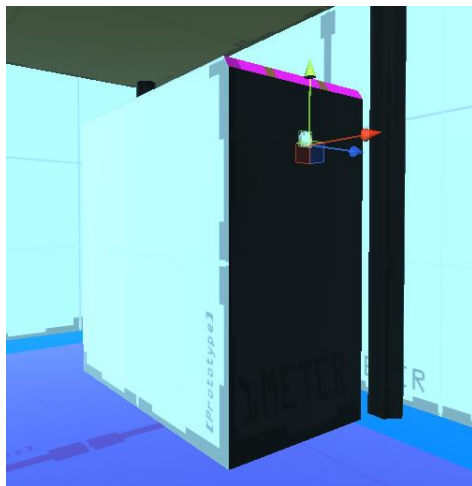


Figura 25. Botón para encender el PC (izquierda) y collider del dedo índice (derecha)

Usamos la variable “estaEncendido” para evitar que detecte una colisión cuando ya está colisionando. “seEnciende.Invoke” y “seApaga.Invoke” son otros eventos que hemos creado al cual se le pueden asignar acciones desde el editor de Unity, cómo se puede apreciar en la Figura 26.

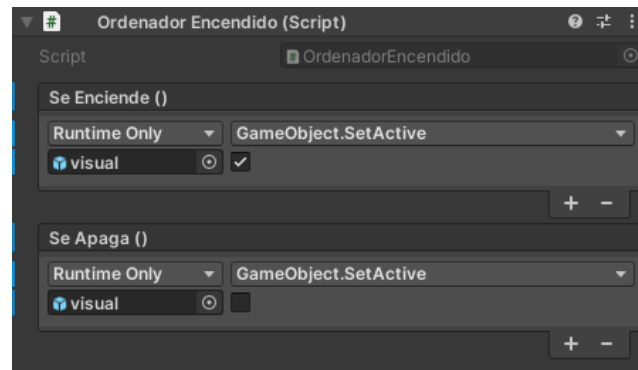


Figura 26. Componente "OrdenadorEncendido"

Al encender el ordenador de la izquierda, por ejemplo, en realidad estamos haciendo visible un panel el cual contiene una imagen del informe de la Figura 13, y en vez de contener "XX" como número de código, contiene un texto cuyo valor se establece en base al número aleatorio del candado del armario. Lo mismo sucede con el ordenador de la derecha.

Al encender los dos ordenadores el jugador obtiene el código completo del candado. Ahora pasaremos a explicar la lógica del armario.

En el armario podemos diferenciar 2 partes principales: la puerta (incluido el candado) y el resto de la estructura. El resto de la estructura no tiene ninguna interacción, sin embargo, la puerta sí. Ésta tiene, entre otros, los componentes "XR Grab Interactable" y "Hinge Joint". El componente "XR Grab Interactable" nos permite interactuar con este objeto utilizando nuestras manos, que son un interactador. Para hacer que la interacción sea como en la realidad, se utiliza el componente "Hinge Joint" para obligar al objeto a moverse de una manera específica. En este caso como una puerta. Para darle más realismo, la puerta únicamente se puede coger por el pomo.

Además, la puerta no se debe poder abrir en una primera instancia. Para conseguir esto, hemos hecho que por defecto no interaccione con ninguna máscara. Es decir, que no interaccione con ningún objeto y, por tanto, que no interaccione con nuestras manos. Al introducir la contraseña correcta y darle al botón "enter" le asignamos que interaccione con todas las máscaras y por tanto ya se pueda abrir la puerta.

Ahora pasaremos a explicar el funcionamiento del candado. Se puede ver una imagen más de cerca en la Figura 27.

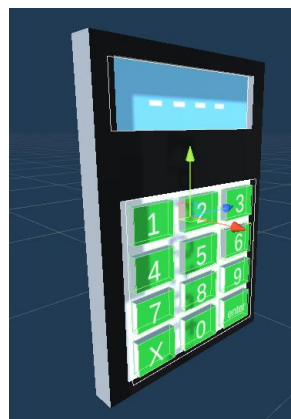


Figura 27. Candado numérico

4. Desarrollo del proyecto

En la parte superior hay un texto el cual se va editando en función de los botones que se pulsen. Cada uno de los botones tiene un componente de tipo “Collider” y el script “teclaPulsada.cs”. Este script es muy simple:

```
public UnityEvent pulsada;  
bool siendoPulsada = false;  
  
private void OnTriggerEnter(Collider other){  
    if (other.CompareTag("Dedo") & !siendoPulsada){  
        pulsada.Invoke();  
        siendoPulsada = true;  
    }  
}  
  
private void OnTriggerExit(Collider other){  
    if (other.CompareTag("Dedo")){  
        siendoPulsada = false;  
    }  
}
```

Utilizamos una variable booleana para controlar las colisiones al igual que con el botón del ordenador. Este código se ejecutará cuando el collider del dedo de la mano colisione con el collider del botón del candado.

Entonces, cuando se pulsa un botón correspondiente a un valor del 0 al 9, se invoca el evento “pulsada” al cual se le asignan acciones desde el editor. Cada uno de los botones llama al método “botonNumericoPulsado()” del script “KeypadLogic.cs” pasándole el valor correspondiente por parámetro. La llamada a esta función para el botón “1”, por ejemplo, se haría como se muestra en la Figura 28 mediante el editor de Unity.

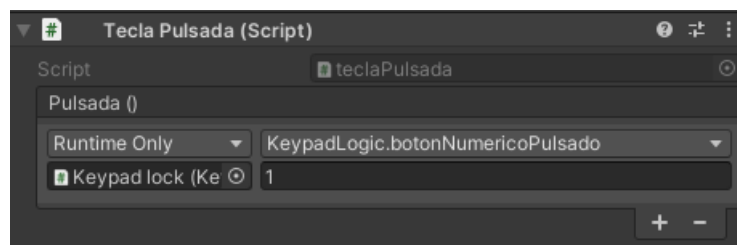


Figura 28. Botón "1" pulsado

Si se pulsa el botón cuyo símbolo es una “X”, se llama al método “borrarUltimo()” del script “KeypadLogic.cs”. Y si se pulsa el botón de “enter” se llama al método “comprobarNumero()” del script “KeypadLogic.cs”. Cabe mencionar que el script “KeypadLogic.cs” es un componente del objeto de Unity que representa el candado.

A continuación, se explican los métodos mencionados anteriormente para entender su funcionamiento:

```
public void botonNumericoPulsado(int num){  
    codigoActual.text = codigoActual.text.Substring(1) + num.ToString();  
}
```

Cuando se pulsa un botón numérico actualizamos el texto que representa la combinación actual del candado.

```
public void borrarUltimo(){  
    codigoActual.text = "-" + codigoActual.text.Substring(0, 3);  
}
```


Cuando se pulsa el botón para borrar el último número, se cambia el texto sustituyendo el último número introducido por un “-”.

```
public void comprobarNumero(){
    if (codigoActual.text.Equals(codigoBueno) && !completada){
        abrirPuerta.Invoke();
        completada = true;
        PeticionesApi.Prueba pruebaCompletada = new
        PeticionesApi.Prueba(idPruebaCompletada,temporizador.tiempoActual);
        peticiones.tiempos.Add(pruebaCompletada);
    }
}
```

A la hora de comprobar el número, se compara el valor actual del código con el valor aleatorio que se le asignó en un principio. Además, como hemos mencionado anteriormente, se cambia la máscara con que interactúa la puerta de tal manera que ahora interactúa con las manos. También se añade a una lista el tiempo que ha transcurrido desde el inicio de la partida hasta que se ha introducido el código correcto. Esa lista va guardando pares de (idPrueba, tiempo) que posteriormente se guardarán en la base de datos. En esto se entra más en profundidad en el apartado de la integración del juego con la base de datos.

Al abrir el armario, se obtienen un triángulo y un pendrive que se requieren para completar las siguientes pruebas.

4.3.3.2. Prueba 2

Los elementos que conforman la prueba 2 están en la otra parte de la habitación. Estos son un ordenador, una pantalla y una puerta con un candado como los de la Figura 29.

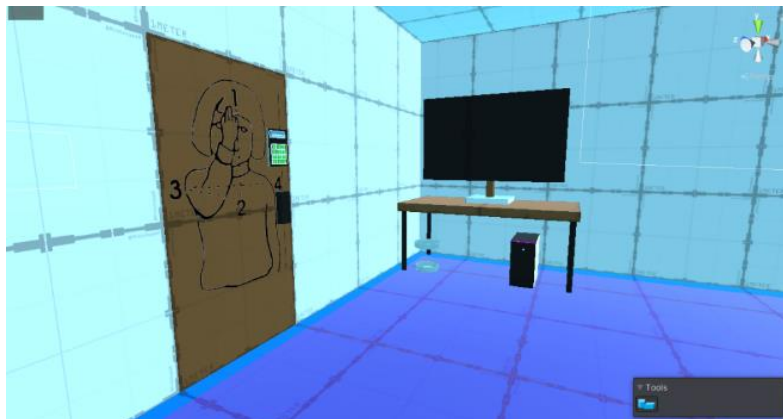


Figura 29. Elementos que componen la prueba 2

Lo primero que debe hacer el jugador es encender el ordenador de la misma manera en que se encienden los ordenadores de la prueba 1. La diferencia es que al encender este, únicamente se mostrará un fondo de escritorio. Para avanzar, el jugador debe conectar el pendrive al ordenador.

Para que esto sea posible el pendrive tiene el componente “XR Grab Interactable”. Para simular que el pendrive se “conecta” al ordenador, necesitamos utilizar el componente “XR Socket Interactor”. Este componente actúa como interactor, al igual que nuestras manos, por tanto, es capaz de interactuar con un objeto interactable como lo es el pendrive.

Este ordenador tiene dos hendiduras que simulan las entradas de un puerto usb. Lo que hemos hecho ha sido crear dos objetos vacíos y añadirles un collider y el componente “XR Socket

4. Desarrollo del proyecto

Interactor”. Cuando el collider del pendrive entra en contacto con el del socket interactor, ambos objetos entran en estado de hover. En ese momento aparece la silueta de un pendrive para indicarle al jugador que lo debe soltar.

En el momento en que el jugador suelta el pendrive, el socket interactor pasa de estado hover a estado select y el pendrive se mantiene en una posición estática, como si estuviese conectado. Aprovechando los eventos definidos por el paquete XR Interaction Toolkit, que se disparan al cambiar de estado, se gestiona el que aparezca la silueta del pendrive y una vez se “conecta”, aparezca algo más en la pantalla. En la Figura 30 se aprecia el collider del socket interactor y la silueta verde que aparece cuando ambos colliders colisionan.

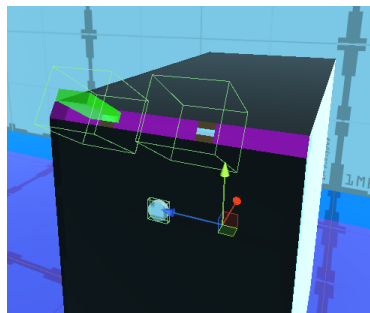


Figura 30. Socket interactor para conectar el pendrive al PC

Por defecto el componente “XR Socket Interactor” interacciona con cualquier objeto interactable. Para conseguir que solo interactúe con el pendrive le hemos asignado una capa diferente a éste y hemos especificado que solo interactúe con esa capa. En Figura 31 se muestran todas las acciones que se llevan a cabo durante los diferentes cambios de estado.

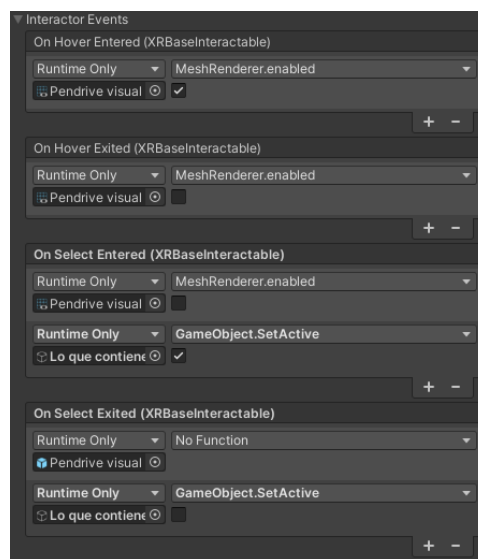


Figura 31. Eventos del socket interactor del PC

Para esta prueba también se genera un número aleatorio para el candado de la puerta. Este número también se debe mostrar en la pantalla a modo de pista. Recordemos que en la pantalla aparecerá una imagen como la de la Figura 15, donde “X”, “Y”, “Z” y “W” son textos cuyos valores corresponderán a cada uno de los números que conforman el código del candado de la puerta. Esto se asigna en el script “Inicialización.cs” con las siguientes instrucciones:

4.3. Implementación

```
//generamos el código random que aparecerá en el monitor de la derecha
for (int i = 0; i < 4; i++){
    int num = UnityEngine.Random.Range(0, 10);
    codigoPendrive += num.ToString();
}

//le asignamos los valores a los textos de la imagen
numeroMonitorDch1.text = codigoPendrive[0].ToString();
numeroMonitorDch2.text = codigoPendrive[1].ToString();
numeroMonitorDch3.text = codigoPendrive[2].ToString();
numeroMonitorDch4.text = codigoPendrive[3].ToString();

//asignamos el código al candado de la puerta
codigoPuertaSala2.setCodigoBueno(codigoPendrive);
```

El candado lo hemos hecho de tal manera que se pueda utilizar en cualquier objeto. Al poner el código correcto se activa un evento al cual se le añaden acciones específicas desde el editor de Unity. Así el código es siempre el mismo, pero se puede personalizar cada vez que se use.

La puerta tiene el mismo mecanismo que la puerta del armario anterior. En una primera instancia no se puede interactuar con ella, pero al introducir la contraseña correcta sí. También utiliza los mismos componentes que la puerta del armario. Al pasar esta puerta el jugador entra en la segunda sala. Al igual que en la primera prueba, al introducir el código correcto se guarda el tiempo transcurrido de partida hasta que se ha completado la prueba.

4.3.3.3. Prueba 3

Para la tercera prueba se genera un número aleatorio entre el 1 y el 8 (ambos incluidos) en el script “Inicialización.cs”. A partir de este número ocurren dos cosas:

1. Se completa una ecuación que aparece escrita en una pizarra. Como se menciona en el apartado del diseño, esta ecuación siempre acaba siendo “ $x = \text{random}(1,8)$ ”. Se puede ver cómo queda en la Figura 32. Esto se consigue con el siguiente código:

```
int r = UnityEngine.Random.Range(1, cantPlanetas + 1);
numeroPlaneta.text += r.ToString();
```

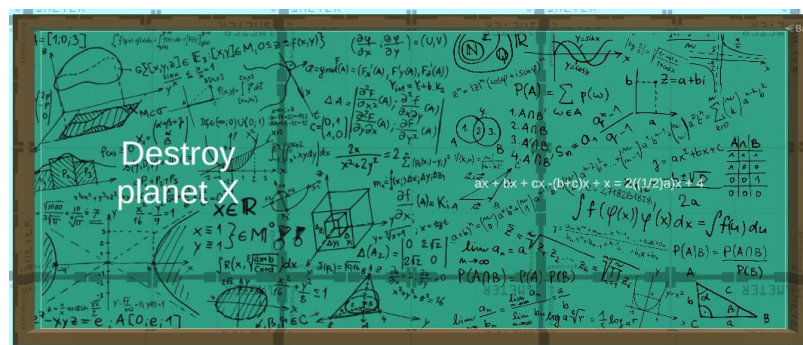


Figura 32. Pizarra que contiene la ecuación necesaria para resolver la prueba 3

La variable “numeroPlaneta” representa el texto que contiene la ecuación completa.

2. Se crean de manera dinámica 8 lámparas con la textura de los diferentes planetas del sistema solar, pero solo una de ellas se puede romper. Por ejemplo, si el número aleatorio generado es el 2, únicamente se podrá romper la segunda lámpara. En la Figura 33 se puede ver como quedan las lámparas al inicializarse la sala.

4. Desarrollo del proyecto

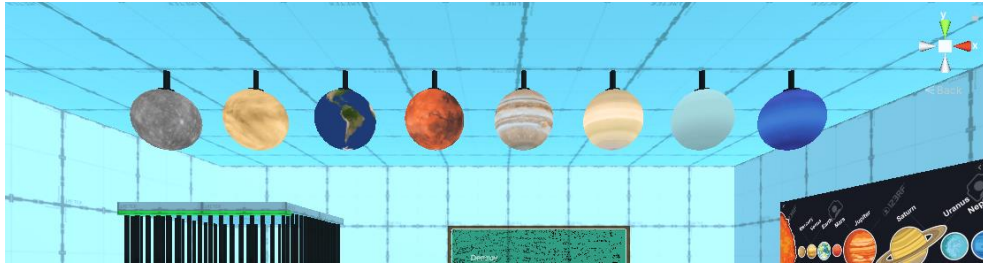


Figura 33. Planetas instanciados necesarios para resolver la prueba 3

Como los planetas se deben instanciar de manera dinámica, es necesario tener guardadas las posiciones en que se quieren instanciar. Estas están guardadas en un fichero JSON con la siguiente estructura:

```
{
  "lamparas":[
    {
      "position":{"x":77.0,"y":2.85,"z":39.75},
      "rotation":{"x":0.0,"y":0.0,"z":0.0,"w":1.0},
      "scale":{"x":0.48,"y":0.48,"z":0.48}
    },
    . . .
  ]
}
```

Con la siguiente función, donde se pasa por parámetro el número del planeta generado, se crean las lámparas:

```
void SpawnLamparasClientRpc(int n) {
    //leemos el JSON con las 8 ubicaciones de las lámparas
    string jsonStringPath =
    File.ReadAllText("Assets/JSON/lamparas.json");
    JObject lamparas = JObject.Parse(jsonStringPath);

    for (int i = 0; i < cantPlanetas; i++) {
        GameObject planeta;
        if ((i + 1) == n) { //es el planeta que se tiene que destruir
            planeta = Instantiate(prefabPlanetaDestruible,
                new Vector3((float)lamparas["lamparas"][i]["position"]["x"],
                    (float)lamparas["lamparas"][i]["position"]["y"],
                    (float)lamparas["lamparas"][i]["position"]["z"]),
                transform.rotation);
        } else { //el planeta no se tiene que poder destruir
            planeta = Instantiate(prefabPlanetaNoDestruible, new
                Vector3((float)lamparas["lamparas"][i]["position"]["x"],
                    (float)lamparas["lamparas"][i]["position"]["y"],
                    (float)lamparas["lamparas"][i]["position"]["z"]),
                transform.rotation);
        }
        planeta.GetComponent<TexturaPlanetas>().ponerTexturas(i);
    }
}
```

En resumen, se tienen dos lámparas diferentes como activos del proyecto. Un activo (la lámpara que no se rompe) tiene únicamente el componente gráfico. El otro activo (la lámpara que si se puede romper) tiene además el script "ControlPlaneta.cs". El código anterior instancia, mediante un bucle, 8 lámparas y en función de si es la que se debe romper o no, instancia un activo u otro.

El script “ControlPlaneta.cs” utiliza el método “OnCollisionEnter()” y mira si el objeto que ha colisionado es un puntero como el de la Figura 34, que se encuentra en la esquina de la sala. Está hecho de tal manera que para romper la lámpara haya que darle con este puntero, el cual tiene el componente “XR Grab Interactable” para que los jugadores lo puedan coger.

Si es el caso, al romperse la lámpara se instancia un objeto con forma de párpado, que cae al suelo por la gravedad. Al romper la lámpara se da por finalizada la prueba y se guarda el tiempo.



Figura 34. Puntero para destruir el planeta correspondiente

4.3.3.4. Prueba 4

En este punto el jugador ya tiene un objeto con forma de triángulo y un objeto con forma de párpado. Únicamente le queda por conseguir un objeto con forma de ojo. Este objeto se instancia de manera aleatoria en una de las papeleras que hay distribuidas por las dos salas. Esto también se hace en el script “Inicialización.cs”. Para facilitar esta tarea se han agrupado todas las papeleras de tal manera que, teniendo la referencia a un objeto de la escena, se pueda acceder a cada una de las papeleras. Con el siguiente código se instancia el ojo:

```
//de todas las papeleras que hay, cogemos una random y spawnamos ahí el ojo
r = UnityEngine.Random.Range(0, basuras.transform.childCount - 1);
GameObject hijo = basuras.transform.GetChild(r).gameObject;
GameObject g = Instantiate(prefabOjo,hijo.transform.GetChild(1).position,
hijo.transform.GetChild(1).rotation);
```

Tanto el triángulo, como el párpado, como el ojo tienen el componente “XR Grab Interactable” de tal manera que el jugador los puede coger. Para completar la prueba, el jugador debe coger cada uno de ellos y ponerlos en el “candado” de la jaula final que se puede ver en la Figura 35 (izquierda).

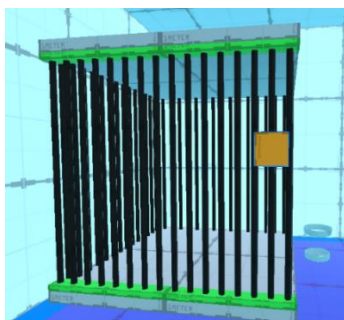


Figura 35. Jaula que contiene el candado final (izquierda) y Socket Interactor del candado final (derecha)

4. Desarrollo del proyecto

Este candado funciona utilizando socket interactors, al igual que el ordenador de la segunda prueba. Cuando acercas lo suficiente uno de los tres objetos, se ilumina una silueta. Si el jugador lo suelta cuando la silueta está activa, el objeto se queda fijo en una posición. En la Figura 35 (derecha) el jugador ya ha posicionado el ojo y está sujetando el párpado de tal manera que se ha activado la silueta.

La lógica del socket es similar a la explicada en la prueba 2, pero un poco más compleja ya que se ven involucrados más elementos. Cada vez que se pone un objeto se comprueba si están los 3 puestos. Si es el caso se guarda el tiempo que se ha tardado en completar la prueba y se finaliza la partida. En la Figura 36 se pueden ver las acciones que se llevan a cabo para la gestión del socket que controla únicamente el triángulo.

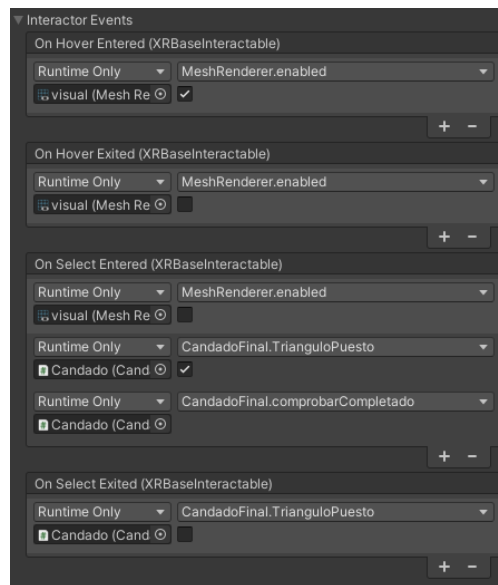


Figura 36. Eventos del candado final

4.3.4. Integración con la base de datos

A continuación, se explican las herramientas que se utilizan para la base de datos y para el servidor donde ésta reside y los diferentes archivos que hay en el servidor que la consultan (excluyendo los referentes al bloque de análisis).

4.3.4.1. Selección de herramientas

En este apartado hay 2 elementos para los que se debe elegir una herramienta.

El primer elemento es la base de datos como tal. Existen bases de datos relacionales [19] y no relacionales [20]. Durante la carrera solo hemos utilizado bases de datos relacionales, y ese ha sido el motivo principal que nos ha llevado a tomar la decisión de usar una base de datos relacional. Además, una base de datos relacional encaja muy bien con los tipos de datos que necesitamos guardar. En concreto hemos utilizado MySQL [21], que es con la que más experiencia tenemos.

La otra decisión que hay que tomar es qué servidor web utilizamos. Las opciones que he empleado yo durante los estudios han sido: un servidor Apache [22] utilizando php [23] como lenguaje en varias asignaturas, y un servidor Glashfish [24] utilizando java [8] como lenguaje en una parte de una asignatura. Más adelante, cuando ya estábamos en una fase más avanzada del TFG utilicé por primera vez un servidor node [25]. Pero a la hora en que tomamos esta decisión

aún no lo había usado, por tanto, esta opción la descartamos. En cuanto a las dos opciones que ya había utilizado yo, tengo más experiencia y trabajo de manera más eficiente utilizando Apache como servidor y php como lenguaje. Por tanto, esa es la opción que escogimos.

4.3.4.2. Registrar usuario

Recordemos que, al iniciar el juego, nos aparecerá una pantalla como la de la Figura 6 (izquierda). Si clicamos en el botón de “Registrarse” cambiaremos al menú que se muestra en la Figura 6 (centro). Una vez completamos los campos y le demos al botón de “Registrarse” se ejecutará la siguiente función:

```
string registerURL = "https://alumnos-ltim.uib.es/escaperoom/register.php";

IEnumerator Registrarse(){
    WWWForm form = new WWWForm();
    form.AddField("usuario", campoUsuario.text);
    form.AddField("contraseña", campoContraseña.text);
    UnityWebRequest www = UnityWebRequest.Post(registerURL, form);
    yield return www.SendWebRequest();
    if (www.downloadHandler.text != "0") {
        Debug.Log(www.downloadHandler.text);
    } else {
        Debug.Log("Te has registrado correctamente");
    }
}
```

Como vemos, se envía una petición post al archivo “register.php” al cual se le pasa un formulario con el nombre de usuario y la contraseña especificados por el usuario desde el menú. La contraseña, pese a que parezca que viaja en texto plano, no es así ya que al utilizar https se crea un canal por dónde la información pasa cifrada [26].

El archivo “register.php” lo primero que hace es conectarse a la base de datos, que se encuentra en el mismo servidor. Si no ha habido ningún error comprueba que el nombre de usuario no exista ya que, pese a no ser llave primaria, serán únicos. Finalmente, si el nombre aún no ha sido utilizado, crea un hash y guarda la información en la base de datos. Si todo ha ido bien devuelve un “0”. Esto se hace con el siguiente trozo de código:

```
//establecemos la conexión
$con = mysqli_connect($URL,$USUARIO,$CONTRASEÑA,$DB,$PUERTO);
//miramos si se ha podido establecer la conexión
if(mysqli_connect_errno()){
    echo("1: conexión fallida");
    exit();
}

$user = $_POST["usuario"];
$pass = $_POST["contraseña"];

//miramos si el nombre existe
$consulta = "SELECT nombre FROM usuario WHERE nombre = '" . $user . "'";
$resultado = mysqli_query($con,$consulta) or die("2: query fallida");

if(mysqli_num_rows($resultado) > 0){
    echo("3: el nombre ya existe");
    exit();
}

//añadir el usuario a la BD
$hash = password_hash($pass,PASSWORD_DEFAULT);
```


4. Desarrollo del proyecto

```
$insertUserQuery = "INSERT INTO usuario(nombre, hash) VALUES  
('".$user."', '".$hash."');";  
mysqli_query($con,$insertUserQuery) or die("4: error al insertar usuario");  
  
echo("0");
```

Según la documentación de PHP [27], la manera correcta de almacenar la información de cuenta del usuario es utilizando la función “password_hash” [28], la cual crea un hash a partir de una sal [29] que genera de manera automática php. De esta manera únicamente debemos almacenar el hash que genera la función.

Si todo ha ido bien, se redirige de manera automática a la pantalla de la Figura 6 (izquierda).

4.3.4.3. Iniciar sesión

Estando en el menú inicial, si clicamos en el botón “Iniciar sesión” cambiará el menú que se muestra en la Figura 6 (derecha). Una vez rellenados los campos, si clicamos en el botón “Iniciar sesión” se ejecutará la siguiente función:

```
string loginURL = "https://alumnos-ltim.uib.es/escaperoom/login.php";  
IEnumerator Login() {  
    WWWForm form = new WWWForm();  
    form.AddField("usuario", campoUsuario.text);  
    form.AddField("contraseña", campoContraseña.text);  
    UnityWebRequest www = UnityWebRequest.Post(loginURL, form);  
    yield return www.SendWebRequest();  
    if (www.downloadHandler.text.StartsWith("ERROR")) {  
        Debug.Log(www.downloadHandler.text);  
    } else {  
        DBManager.loggedIn = true;  
        DBManager.usuario = campoUsuario.text;  
        DBManager.id = www.downloadHandler.text;  
        SceneManager.LoadScene(menu);  
    }  
}
```

De manera similar a cuando se registra un usuario, se envía una petición post al archivo “login.php” del servidor enviándole un formulario con el nombre de usuario y contraseña introducidos. Si se ha podido iniciar sesión correctamente (la petición devuelve un texto que no empieza por “ERROR”) se guarda el nombre de usuario e id en una clase estática llamada DBManager. Esta información se utiliza posteriormente para hacer las peticiones a la API. Al iniciar sesión cambia de escena al menú de selección de sala (Figura 7).

A continuación, se muestra el trozo de código dónde se comprueba si la contraseña es correcta:

```
//miramos si el nombre existe  
$consulta = "SELECT id, nombre, hash  
FROM usuario WHERE nombre = '" . $user . "'";  
$resultado = mysqli_query($con,$consulta) or die("ERROR 2: query fallida");  
//cogemos la información de la consulta  
$loginInfo = mysqli_fetch_array($resultado);  
$hash = $loginInfo["hash"];  
//si el hash aplicado a la contraseña NO coincide con el hash guardado  
if(!password_verify($pass,$hash)){  
    echo("ERROR 6: contraseña incorrecta");  
    exit();  
}  
//si se ha iniciado sesión correctamente devolvemos el id de usuario  
echo($loginInfo["id"]);
```


Según la documentación de PHP, la manera correcta de comprobar si una contraseña corresponde a un hash es utilizando la función “password_verify()” [30]. Lo que hace el servidor es hacer una consulta a la base de datos para coger el id y el hash correspondiente al nombre de usuario que ha especificado el jugador (si es que el nombre existe) y, posteriormente, hace la comparación. El método “password_verify()” ya tiene en cuenta la sal que se utilizó al generar el hash.

Si la contraseña no es correcta devuelve un error, y si lo es devuelve el id del usuario.

4.3.4.4. API cliente

Para obtener/guardar datos desde el juego se envían peticiones al archivo “api.php” del servidor. Este archivo puede realizar 4 funciones diferentes. A la hora de enviar la petición se especifica mediante el parámetro “opción”.

En función del valor de este parámetro, utilizando un switch se puede:

1. Dado el nombre de un jugador, obtener un json donde se indica el número de escape rooms que existen (se obtiene al hacer “(int)salas[“cantSalas”]”) y de cada uno se obtienen su nombre, dificultad, el mejor tiempo del jugador y el número de intentos. Esta consulta se realiza al cargar la escena de la Figura 7. Esta escena a la hora de implementarla se ve como en la Figura 37. En esta figura se muestra la interfaz desde el editor, es decir, sin cargar las tarjetas de las salas.

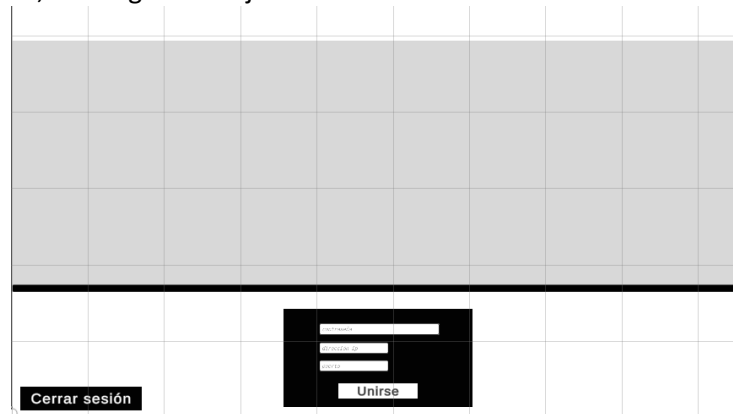


Figura 37. Menú visto desde el editor

Esto se consigue con el siguiente código:

```
UnityWebRequest www = UnityWebRequest.Get(apiURL + "?opcion=1&usuario="
+ DBManager.usuario);
yield return www.SendWebRequest();

if (www.downloadHandler.text.StartsWith("ERROR")) {
    Debug.Log("7: consulta de los mejores tiempos fallida");
} else {
    JObject salas = JObject.Parse(www.downloadHandler.text);
    for (int i = 0; i < (int)salas["cantSalas"]; i++) {
        GameObject g = Instantiate(panel);
        g.transform.SetParent(gameObject.transform, false);
        g.GetComponent<CrearPanel>().Crear((JObject)salas["partidas"][i]);
    }
}
```

4. Desarrollo del proyecto

Entonces, para cada escape room se instancia un objeto panel (como el de la Figura 38) y se llama al método “Crear()” de su script “CrearPanel.cs”.

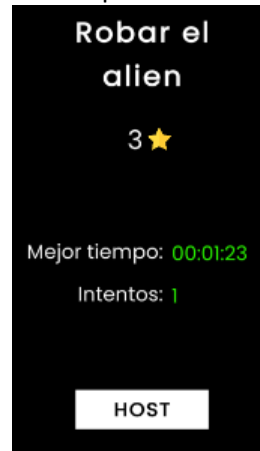


Figura 38. Panel del menú

Este método le asigna los valores que le llegan por parámetro a los diferentes componentes por los que está formado el panel. Este es el código de la clase “CrearPanel” y su método “Crear()”:

```
[SerializeField] TMP_Text titulo, dificultad, mejorTiempo, numIntentos;
[SerializeField] Button host;

public void Crear(JObject datosPartida) {
    titulo.text = datosPartida["escaperoom_nombre"].ToString();
    dificultad.text = datosPartida["escaperoom_dificultad"].ToString();
    //si ha jugado mínimo una partida (por tanto tenemos los datos)
    if (datosPartida["num_intentos"].ToString() != "") {
        mejorTiempo.text = datosPartida["mejor_tiempo"].ToString();
        numIntentos.text = datosPartida["num_intentos"].ToString();
    } else {
        mejorTiempo.text = "-";
        numIntentos.text = "0";
    }
    host.onClick.AddListener(CambiarEscena);
}

void CambiarEscena() {
    SceneManager.LoadScene(titulo.text);
}
```

2. Añadir una nueva partida de un escape room especificado y devolver su ID en la tabla.
3. Dado un id de una partida, un id de una prueba y un tiempo, inserta una fila en la tabla “tiempo”.
4. Dado un id de usuario y un id de partida, inserta una fila en la tabla “usuario_partida”.

El uso de los últimos tres casos se realiza desde el script “PeticionesApi.cs”. En este archivo hay definidas dos listas:

```
public List<Prueba> tiempos = new List<Prueba>();
public List<string> jugadores = new List<string>();
```

Cada vez que se complete una prueba se irán guardando objetos de tipo “prueba” en la lista “tiempos”, los cuales tienen la siguiente estructura:

```
public class Prueba {

    public int num;
    public float tiempo;

    public Prueba(int n, float t) {
        this.num = n;
        this.tiempo = t;
    }
}
```

En la lista “jugadores” se van guardando los identificadores de los diferentes jugadores de la partida. Inicialmente, cuando no estaba implementado el multijugador, esto no hacía falta, pero una vez implementado es necesario para saber quién ha jugado en que partidas.

Una vez se realiza la última prueba se ejecuta el método “Guardar()”, el cual emplea las 3 consultas anteriores:

```
IEnumerator Guardar() {
    //Consulta #2 -> añadir una nueva partida y devolver el ID
    WWWForm form = new WWWForm();
    form.AddField("opcion", 2);
    form.AddField("idEscRoom", idEscapeRoom);
    UnityWebRequest www = UnityWebRequest.Post(apiURL, form);
    yield return www.SendWebRequest();
    string idPartida = www.downloadHandler.text;

    //Consulta #3 -> añadir un nuevo tiempo a una partida
    for (int i = 0; i < tiempos.Count; i++) {
        form = new WWWForm();
        form.AddField("opcion", 3);
        form.AddField("idPartida", idPartida);
        form.AddField("idPrueba", tiempos[i].num);
        form.AddField("tiempo", tiempos[i].tiempo.ToString().Replace(',', '.'));
        StartCoroutine(PonerTiempos(form));
    }

    //Consulta #4 -> las relaciones entre los jugadores y la partida
    for (int i = 0; i < jugadores.Count; i++) {
        form = new WWWForm();
        form.AddField("opcion", 4);
        form.AddField("usuario", jugadores[i]);
        form.AddField("idPartida", idPartida);
        StartCoroutine(PonerUsuarioPartida(form));
    }
}

IEnumerator PonerTiempos(WWWForm form) {
    UnityWebRequest request = UnityWebRequest.Post(apiURL, form);
    yield return request.SendWebRequest();
}

IEnumerator PonerUsuarioPartida(WWWForm form) {
    UnityWebRequest request = UnityWebRequest.Post(apiURL, form);
    yield return request.SendWebRequest();
}
```

4. Desarrollo del proyecto

4.3.5. Multijugador

Al implementar el multijugador, no solo se han añadido elementos nuevos, sino que también se han modificado algunos de los existentes. Lo más importante de este apartado es entender los elementos y la lógica que se emplea para gestionar el multijugador.

4.3.5.1. Selección de herramientas

Hay tres opciones principales para añadirle multijugador a un juego hecho en Unity. Estas son Photon [31], Mirror [32] y MLAPI [33]. Con las tres opciones se puede implementar el multijugador. Mirror y MLAPI son completamente gratuitos, sin embargo, en Photon solo es gratis el plan básico, que limita en 20 a los jugadores por partida. De cualquier manera, es suficiente para el juego ya que lo ideal es que las partidas sean de entre 1 y 8 jugadores, pese a que este no es un requisito.

El bloque de multijugador lo hemos añadido a este TFG porque yo personalmente quería aprender cómo se gestiona el multijugador en un juego, que como hemos mencionado en la introducción es algo que no hemos visto durante la carrera. De las tres posibilidades, Photon y Mirror están en una versión avanzada mientras que MLAPI está en beta todavía. A pesar de esto, MLAPI contiene las funcionalidades principales de las otras dos opciones, incluyendo las necesarias para realizar esta aplicación. Photon y Mirror son aplicaciones de terceros mientras que MLAPI es la solución oficial de Unity. Personalmente me interesa más aprender la solución propia de Unity, ya que, si en un futuro me propongo desarrollar un videojuego con multijugador, probablemente utilizaré MLAPI.

4.3.5.2. Explicación

A la hora de implementar MLAPI, se distinguen dos tipos de objetos: los objetos normales (como los que hay sin multijugador) y los denominados “NetworkObjects” [34]. La diferencia es que, en un sistema multijugador donde hay un servidor que se comunica con los diferentes clientes, los objetos normales están instanciados en el juego de cada cliente y una modificación en ellos no se ve reflejado en el resto de los clientes. Por otro lado, un NetworkObject es un objeto instanciado en el servidor y replicado por los clientes. De tal manera que, si se produce una modificación en este objeto, se ve reflejado en el resto de los clientes. Los NetworkObjects tienen asociado un propietario, que por defecto es el servidor. Únicamente el propietario de un objeto puede modificarlo.

Otro concepto importante a tener en cuenta es que hay dos maneras de implementar el multijugador. Una es teniendo un servidor el cual cree salas y los jugadores se unan a estas salas. Otra es que uno de los jugadores haga de servidor. En este caso el jugador que hace de cliente y de servidor se denomina “host”. Hemos optado por implementar la segunda opción ya que es un poco más sencilla de implementar, pero la gestión dentro de una partida es prácticamente la misma que con la otra opción.

Antes de empezar con la implementación más concretamente, hemos de introducir un concepto más. Este concepto es el de la llamada a un procedimiento remoto [35], conocido en inglés como Remote Procedure Call (RPC). Hay de dos tipos: el ClientRpc es un método que es llamado por el servidor, pero que se ejecuta en todos los clientes. Por otro lado está el ServerRpc, el cual es llamado por un cliente, pero se ejecuta en el servidor.

El diseño de la aplicación se ha hecho de tal manera que en el menú de la Figura 7, el jugador pueda elegir si crear una partida y hacer de host o unirse a una partida como cliente.

Al darle al botón de host el juego cambia a la escena de la Figura 8 (izquierda). Una vez introducimos la contraseña de la sala se ejecuta el siguiente código:

```
tuIp.text += LocalIPAddress();
tuIp.gameObject.SetActive(true);
NetworkManager.Singleton.ConnectionApprovalCallback += ApprovalCheck;
NetworkManager.Singleton.StartHost(new Vector3(0, 0, 0), new Quaternion(), true);
```

Este código hace que se muestre por pantalla tu ip para que el resto de los jugadores sepan la ip a la que se deben conectar. También se añade la función “ApprovalCheck()”, que se ejecutará cuando se conecte un jugador:

```
private void ApprovalCheck(byte[] connectionData, ulong clientID,
NetworkManager.ConnectionApprovedDelegate callback) {
    string contraseña = Encoding.ASCII.GetString(connectionData);
    bool conexionAprobada = contraseña == contraseñaInput.text;
    callback(true, null, conexionAprobada, new Vector3(0,0,0),
    new Quaternion());
}
```

Finalmente se llama a la función “StartHost()” propia de Unity la cual hace que el jugador que ha creado la sala haga de servidor y por tanto los usuarios ya se puedan conectar. Estas dos últimas funciones son de la clase “NetworkManager”, la cual permite al programador utilizar funciones de medio y alto nivel. MLAPI también utiliza otro componente llamado “UNetTransport”, el cual maneja los componentes de la capa de transporte y, por tanto, trabaja a más bajo nivel.

En este punto, ya se ha iniciado el “servidor” (recordemos que el host hace de cliente y de servidor) y por tanto el resto de los jugadores ya pueden utilizar la opción de “Unirse” del menú de la Figura 7. Al rellenar los campos y darle al botón para unirse se ejecuta el siguiente código:

```
networkM.GetComponent<UNetTransport>().ConnectAddress = ip.text;
networkM.GetComponent<UNetTransport>().ConnectPort = int.Parse(puerto.text);
NetworkManager.Singleton.NetworkConfig.ConnectionData =
    Encoding.ASCII.GetBytes(contraseñaInput.text);
NetworkManager.Singleton.StartClient();
```

Se modifica la ip y el puerto de la clase “UNetTransport” y se guarda en la variable “ConnectionData” la contraseña que se ha introducido. La función “StartClient()” es propia de Unity y se encarga de unirse al servidor cuya ip y puerto son los especificados. Además cuando ejecute el método “ApprovalCheck()” mencionado anteriormente, usará el valor de “ConnectionData” para ver si la contraseña es la correcta.

Cuando un cliente se une a un host, el cliente cambia a la misma escena en la que está el host. De esta manera, si el host está en el escape room X, al cliente le cargará la escena correspondiente al escape room X automáticamente.

Cuando todos los jugadores están listos, el host le deberá dar al botón de empezar que aparece en la Figura 8 (derecha). Al darle, los jugadores podrán entrar en la primera sala y empezará a correr el tiempo.

A continuación, se explican los ajustes que se han debido de hacer al juego básico para adaptarlo al multijugador.

Ahora los objetos que tienen el componente “XR Grab Interactable” tienen que ser NetworkObjects ya que son objetos con los que se puede interaccionar y por tanto el resto de los jugadores deben ver como estos objetos se mueven. Para convertir un objeto normal en un

4. Desarrollo del proyecto

NetworkObject basta con añadirle el componente “NetworkObject”. Pero, para que todos los usuarios vean los cambios en su posición, rotación y escala se necesita del componente “NetworkTransform”. Por tanto, todos los objetos que se pueden coger también tienen este componente.

La implementación de los NetworkObjects con los que pueden interactuar todos los jugadores tiene una problemática. Estos objetos tienen un propietario (por defecto el servidor) y por tanto solo el propietario puede modificarlos. Esto significa que ninguno de los jugadores podrá interactuar con los objetos excepto el host, que recordemos hace de servidor. Para solventar este problema hemos creado un script llamado “CambiarPropietario.cs”, que está como componente de todos los objetos que deben poder coger los jugadores. El código es el siguiente:

```
private void OnTriggerEnter(Collider other) {
    if(other.gameObject.CompareTag("Coger")) {
        if (gameObject.GetComponent<NetworkObject>().OwnerClientId ==
            NetworkManager.LocalClientId) {
            ObtenerOwnershipServerRpc(other.gameObject.GetComponentInParent
                <NetworkObject>().OwnerClientId);
        }
    }
}

[ServerRpc]
void ObtenerOwnershipServerRpc(ulong clientId) {
    gameObject.GetComponent<NetworkObject>().ChangeOwnership(clientId);
}
```

La problemática reside en que un objeto solo se puede cambiar de propietario desde el servidor, y este cambio solo lo puede solicitar el jugador que es propietario. Es decir, si el propietario de un objeto “O” es la jugadora “Alice”, no puede venir el jugador “Bob” y convertirse en el propietario de “O” por voluntad propia. Sino que debe ser Alice quien le diga al servidor que el nuevo propietario de “O” será Bob.

El script anterior reside en el objeto “O”. Cada jugador tiene en su juego un objeto instanciado para cada uno de los jugadores (conocido como “player object”). Cada jugador es el propietario de su objeto jugador y por tanto un jugador solo puede mover su objeto jugador (el resto de los objetos jugador ignoran los inputs). Entonces si Bob toca el objeto “O”, se activará el método “OnTriggerEnter()” en su juego, pero también se activará el mismo método en el juego de Alice. Esto se debe a que en ambos juegos una mano ha tocado el objeto “O”.

Cuando se ejecuta el método “OnTriggerEnter()”, la primera condición que se mira es si ha sido tocado por un objeto con el tag “Coger”. Este tag solo lo tiene un objeto de los que forman las manos. A continuación, mira (con el siguiente condicional) si el propietario del objeto es uno mismo (recordemos que el código se está ejecutando para el objeto “O” en el juego de cada jugador). De esta manera conseguimos que la instrucción “ObtenerOwnershipServerRpc()” la ejecute solo el propietario del objeto. A esta función se le pasa por parámetro el identificador (de dentro de la partida, no el de la base de datos) del jugador que quiere ser el nuevo propietario (el que ha tocado el objeto). Como es un ServerRpc este código se ejecutará en el servidor y por tanto éste es capaz de ejecutar la función “ChangeOwnership()”.

Una vez implementado el multijugador, el funcionamiento del script “Inicialización.cs” es un poco diferente. Cuando solo había un jugador, este podía ejecutar todo el código y generar él todos los valores aleatorios. Ahora, al haber varios jugadores, si todos ejecutan ese código, cada uno generará unos números diferentes y por tanto los jugadores no estarán sincronizados.

Ahora lo que hacemos es generar las variables aleatorias en el servidor y utilizar un ClientRpc pasándole por parámetro los valores que se han generado:

```
[ClientRpc]
private void SincronizarClientRpc(string codigoArmarioLocal, string codigoPen,
string ecuacion) {

    //le asignamos este número tanto al candado del armario,
    //como a las pistas de los monitores
    codigoArmario.setCodigoBueno(codigoArmarioLocal);
    numeroMonitorIzq1.text = codigoArmarioLocal.Substring(0, 2);
    numeroMonitorIzq2.text = codigoArmarioLocal.Substring(2, 2);

    //asignamos los números a la imagen que contiene el pendrive
    numeroMonitorDch1.text = codigoPen[0].ToString();
    numeroMonitorDch2.text = codigoPen[1].ToString();
    numeroMonitorDch3.text = codigoPen[2].ToString();
    numeroMonitorDch4.text = codigoPen[3].ToString();

    //asignamos el código que se obtiene a la puerta
    codigoPuertaSala2.setCodigoBueno(codigoPen);
    numeroPlaneta.text = ecuacion;
}
```

De esta manera cada jugador le asigna los mismos valores, de manera local, a sus variables correspondientes. Otro cambio es que al instanciar el objeto con forma de ojo y el objeto con forma de párpado, se deben hacer desde el servidor y de la siguiente manera:

```
GameObject g = Instantiate(prefab, position, rotation);
g.GetComponent<NetworkObject>().Spawn();
```

Con el método “Instantiate()” se instancia de manera local, para que este objeto se replique en los juegos del resto de jugadores se necesita utilizar el método “Spawn()”.

Otro cambio importante a tener en cuenta es que cuando solo había un jugador, este guardaba los datos y hacía las peticiones a la API. Ahora estas peticiones las hace únicamente el host. Esto se consigue poniendo un condicional a la hora de llamar al método:

```
if (IsHost) {
    StartCoroutine(Guardar());
}
```

Es importante entender que si Alice, por ejemplo, está moviendo la mano de su personaje, esa mano existe en el juego de cada jugador y por tanto se está moviendo en todos los juegos, incluido en el de Bob, que es el host. Por tanto, si Alice pulsa una tecla de un candado, esa tecla también se va a pulsar en el juego de Bob. Y lo mismo ocurre al completar una prueba. Si Alice le da al “Enter” en un candado y tenía marcado el código correcto, lo mismo ha sucedido en el juego de Bob, y por tanto éste también es capaz de registrar que se ha completado la prueba. Es por esto por lo que independientemente de que jugador complete una prueba, va a suceder lo mismo en todos los juegos de los diferentes jugadores y, por tanto, todos van a registrar los tiempos. Incluido el host, que es quien hace las peticiones para guardar los datos de la partida.

Finalmente, al acabar la partida o si alguien se sale en mitad de la partida, entre otras acciones, el host deberá utilizar la función “NetworkManager.StopHost()” y los clientes la función “NetworkManager.StopClient()”. Si no, no serán capaces de crear una nueva partida o de unirse a una nueva partida.

4. Desarrollo del proyecto

4.3.6. Análisis de los datos

Este bloque se puede dividir en dos partes: el envío de los datos por partes de los jugadores, y el desarrollo de la web donde se incluye el algoritmo para crear el mapa de calor. Previa a la explicación de estas dos partes vamos a explicar que tecnologías hemos empleado.

4.3.6.1. Selección de herramientas

La página web que hemos realizado, en cuanto a su estructura, es muy simple. Únicamente contiene un título, un formulario y un contenedor donde se pinta un mapa de calor. Por tanto, no es necesario utilizar ningún framework para controlar la organización de los elementos. Por este motivo hemos utilizado html [36] y css [37] puro, sin utilizar ningún framework ni ninguna librería. Por otro lado, para manejar los cambios en el formulario y para hacer peticiones de manera asíncrona al servidor hemos decidido utilizar jQuery [38].

Además de lo mencionado anteriormente, necesitamos una herramienta que nos permita dibujar un mapa de calor. Para esto hemos utilizado la librería “heatmap.js” [39]. Más adelante se explican los métodos utilizados y los datos necesarios para utilizarla.

4.3.6.2. Envío de datos y creación de la página web

Al objeto que representa cada jugador se le ha añadido el script “Posición_JSON.cs”. En el método “Start()” llama a la siguiente corrutina [40]:

```
IEnumerator GuardarPosicion() {
    while (true) {
        jsonPosicion += "{\"x\":\"" + (Math.Truncate(10 * cam.transform.position.x) /
        10).ToString().Replace(',', '.')) + "\",\"y\":\"" + (Math.Truncate(10 *
        cam.transform.position.z) /10).ToString().Replace(',', '.')) + "\",\"z\":\"";
        yield return new WaitForSeconds((float)0.25);
    }
}
```

Esta corrutina guarda cada 250ms la posición del jugador, en formato JSON, con ninguno o un decimal. Hemos decidido hacerlo así para que a la hora de representarlo gráficamente haya varias entradas de la misma posición.

Recordemos que, al acabar una partida, el host envía mediante el script “PeticionesApi.cs” los datos a “API cliente”. Ahora, una vez se han realizado todas las peticiones explicadas anteriormente, también llama a la función “PosicionesJugadoresClientRpc(idPartida)”. Como se puede observar esta llamada es un ClientRpc y le pasa por parámetro el identificador de la base de datos de esa partida, que ha sido insertada previamente. Este es el código del método mencionado:

```
[ClientRpc]
void PosicionesJugadoresClientRpc(string idPartida) {
    GameObject[] jugadores = GameObject.FindGameObjectsWithTag("Jugador");
    for (int i=0; i < jugadores.Length; i++) {
        if (jugadores[i].GetComponent<NetworkObject>().IsLocalPlayer) {
            jugadores[i].GetComponent<Posicion_JSON>().EnviarJson(idPartida);
        }
    }
}
```

Lo que hacen los clientes es obtener una referencia al objeto jugador que les representa y llamar al método “EnviarJson(idPartida)”. Es necesario pasarle el id de la partida ya que se envía como parámetro en la petición. Así pues, los clientes ejecutan el siguiente código:


```

public void EnviarJson(string idPartida) {
    //Arreglamos el json
    StopCoroutine(GuardarPosicion());
    jsonPosicion = jsonPosicion.Substring(0, jsonPosicion.Length - 1);
    jsonPosicion += "]";

    //Enviamos el formulario
    WWWForm formulario = new WWWForm();
    formulario.AddField("opcion", 1);
    formulario.AddField("idPartida", idPartida);
    formulario.AddField("idUsuario", DBManager.id);
    formulario.AddField("tipoDatos", 1); //tipoDatos -> posición
    formulario.AddField("datos", jsonPosicion);
    Debug.Log(jsonPosicion);
    StartCoroutine(Enviar(formulario));
}

IEnumerator Enviar(WWWForm f) {
    UnityWebRequest www = UnityWebRequest.Post(urlApiAnalisis, f);
    yield return www.SendWebRequest();
    Debug.Log(www.downloadHandler.text);
}

```

Primero se arregla el JSON para que tenga el formato correcto y después se envía una petición a "API análisis" especificándole "opción=1".

Esta petición la recibe el servidor, y luego de establecer la conexión con la base de datos, ejecuta el siguiente código:

```

$jsonDatos = $request['datos'];
$idPartida = $request['idPartida'];
$idUsuario = $request['idUsuario'];
$tipoDatos = $request['tipoDatos'];

$consulta = "INSERT INTO datos(json, idPartida, idUsuario, idTipodatos) VALUES ('".
$jsonDatos."', ".$idPartida.", ".$idUsuario.", ".$tipoDatos.")";

mysqli_query($con,$consulta) or die ("ERROR 2: fallo en la consulta #1");
echo("La consulta 1 ha ido bien");

```

Se inserta un JSON en la tabla datos, para un jugador en una partida en concreto. Si no ocurre ningún error, en este punto ya tenemos almacenados los datos necesarios para hacer el análisis.

Ahora pasaremos a explicar el funcionamiento de la página web. Inicialmente esta únicamente contiene, aparte de diversos contenedores para estructurar bien los elementos, tres párrafos y tres combobox. Se utiliza un párrafo y un combobox para seleccionar un escape room, otros para seleccionar partida y otros para seleccionar jugador. Los elementos quedan tal y cómo se diseñó en la Figura 18.

Como los elementos aparecen de manera dinámica, la opción de elegir jugador y partida están inicialmente ocultas ya que primero se debe seleccionar un escape room. Al cargar la página, se envía una petición al servidor para obtener el nombre e identificador de todos los escape rooms. Una vez obtenidos, se añaden al primer combobox.

En este punto, la página web se ve como en la Figura 39.

Análisis del comportamiento de los jugadores

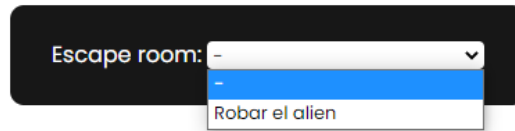


Figura 39. Elementos iniciales de la página web

Si clicamos en un escape room (en este caso solo hay uno, pero si se añaden más a la base de datos, aparecerán aquí) se ejecuta la función “cambioComobox1()”.

Esta función se compone de dos partes. En la primera se hacen los ajustes visuales de tal manera que, si se ha seleccionado un escape room, nos aparezca la opción de elegir una partida. Y si no se ha seleccionado ningún escape room, se quita la opción de elegir partida.

En caso de no haber seleccionado ninguno, se llama a la función “pintarMapaCalor(JSON)” pasándole un JSON vacío, de tal manera que únicamente pinte el mapa de fondo. Esta función se explica más adelante. Pero de manera resumida, dado un JSON con coordenadas de Unity, crea un mapa de calor sobre un plano del escape room.

Si por el contrario sí se ha seleccionado un escape room, también se ejecuta el siguiente código:

```
var form = new FormData();
form.append('opcion', 3);
form.append('idEscRoom', $("#e1_form").val());
$.ajax({
    url: 'https://alumnos-ltim.uib.es/escaperoom/apiAnalisis.php',
    data: form,
    method: 'post',
    processData: false, // tell jQuery not to process the data
    contentType: false, // tell jQuery not to set contentType
    enctype: 'multipart/form-data',
    success: function(result){
        var json = JSON.parse(result);
        var display = json.display;
        var posiciones = json.posiciones;
        if(display.length > 0){
            $("#e2_form").find("option").remove();
            $("#e2_form").append("<option value='- '>-</option>");
            for(let i=0; i < display.length; i++){
                $("#e2_form").append("<option value="+ display[i].id +">"+
                    display[i].tiempo + " - " + display[i].fecha + "</option>");
            }
        }
        pintarMapaCalor(posiciones);
    }
});
```

Aquí se envía una petición a “API análisis” especificando “opción=3” y pasándole el identificador del escape room seleccionado. Esta petición devuelve un JSON con la siguiente estructura:

```
{
  "display": [
    {
      "id": "...",
      "fecha": "...",
```

```

        "tiempo": "...",
        },
        {...}
    ],
    "posiciones": [
        {
            "x": "...",
            "y": "...",
        },
        {...}
    ]
}

```

Entonces con los datos del objeto “display” añadimos las opciones al combobox de selección de partida, y con los datos del objeto “posiciones” se llama a la función “pintarMapaCalor(JSON)”. De esta manera se pinta un mapa de calor, con las posiciones de todos los jugadores en todas las partidas en ese escape room, como el que se ve en la Figura 40.

Análisis del comportamiento de los jugadores

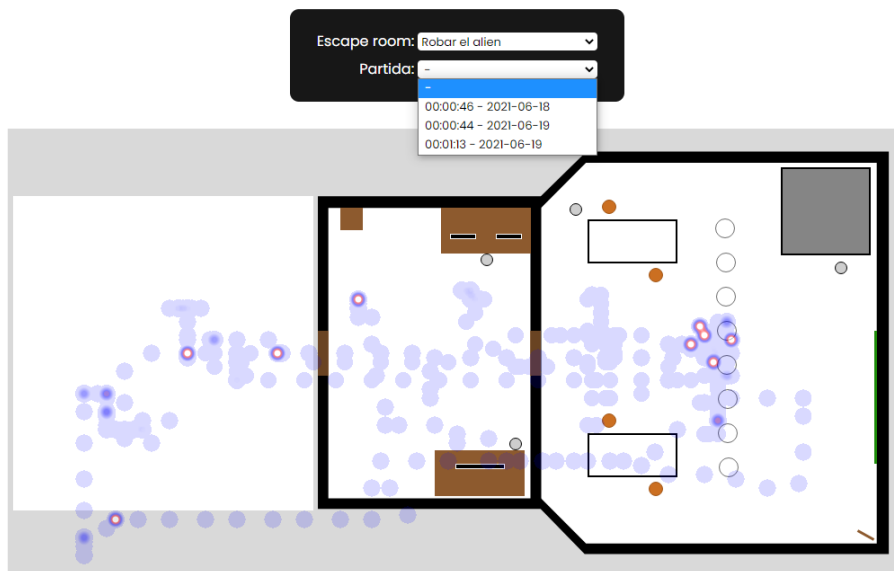


Figura 40. Selección de sala - página web

Si seleccionamos una partida, se ejecuta el método “cambioCombobox2()”. Este método tiene una estructura similar al anterior. Primero se modifica el formulario acorde a si hemos seleccionado una partida o no, y si ha sido el caso se actualizan las opciones del combobox para elegir jugador y se pinta el mapa de calor con las posiciones de todos los jugadores en esa partida.

Entonces nos aparecerá la opción de elegir un jugador en concreto. Al elegirlo se ejecuta la función “cambioCombobox3()” que esta vez es un poco más simple que los dos anteriores, aunque conserva la misma estructura.

Finalmente, vamos a explicar el funcionamiento del método “pintarMapaCalor()”. Para realizar esta función hemos utilizado la librería “heatmap.js”. Las dos funciones más importantes son “create(config)”, que necesita unos parámetros de configuración, y “setData(data)”. Ésta última necesita un array de JSON, donde cada uno tiene el siguiente formato:

4. Desarrollo del proyecto

```
{
  "x":a,
  "y":b,
  "value":c
}
```

“x” e “y” hacen referencia a una coordenada, en píxeles, del contenedor donde se va a pintar el mapa. “value” representa la intensidad de ese punto. Este es el código completo:

```
function pintarMapaCalor(jsonPosiciones){
  var jsonBueno = [];
  for (let i = 0; i < jsonPosiciones.length; i++) {
    var coordenadas = jsonPosiciones[i];
    var aux = {
      "x":coordenadas.y,
      "y":coordenadas.x
    }
    aux.x = (aux.x/dimensionesMapas[$("#e1_form").val()].anchoJuego) *
    anchoCanvas;
    aux.y = (aux.y/dimensionesMapas[$("#e1_form").val()].altoJuego) *
    altoCanvas;

    var existe = false;
    for(let j=0; j < jsonBueno.length; j++){
      var aux2 = jsonBueno[j];
      //miramos si ya está esa posición guardada
      if(aux2.x == aux.x && aux2.y == aux.y){
        existe = true;
        jsonBueno[j].value = jsonBueno[j].value + 1;
        break;
      }
    }
    if(!existe){
      var nuevo = {
        "x":aux.x,
        "y":aux.y,
        "value":1
      }
      jsonBueno.push(nuevo);
    }
  }

  //creamos el mapa de calor
  if( $('#heatmapContainer').length){
    $('#heatmapContainer').remove();
  }
  $("body").append("<div id='heatmapContainer'></div>");
  var config = {
    container: document.getElementById('heatmapContainer'),
    radius: 10,
    maxOpacity: .5,
    minOpacity: .15,
    blur: .75,
    gradient: {
      '.5': 'blue',
      '.8': 'red',
      '.95': 'white'
    }
  };
  var heatmapInstance = h337.create(config);
  var data = {
    max:50,
```

```
        min:0,  
        data:jsonBueno  
    };  
    heatmapInstance.setData(data);  
}
```

Previo a usar la función “create()” y “setData()” es necesario crear un JSON con el formato correcto. Así pues, para cada coordenada del JSON original (están en coordenadas de Unity):

1. Se convierte a coordenadas del contenedor que se va a utilizar.
2. Se mira si esas coordenadas han aparecido anteriormente. Si es el caso simplemente se suma “1” a “value”. Si no, se añade un nuevo JSON al array de coordenadas nueva.

Una vez tenemos el JSON correcto, ya se puede pintar el mapa.

Capítulo 5. Resultados

Durante los apartados anteriores hemos mostrado diversas imágenes, pero la mayoría de éstas son o del diseño o de una fase del escape room en la que no se habían cuidado los gráficos aún.

En la Figura 41, Figura 42 y Figura 43 se puede ver como ha quedado el menú. Pese a no ser idéntico a la interfaz diseñada, los elementos si lo son.

En la Figura 44, Figura 45, Figura 46, Figura 47, Figura 48 se puede ver como ha quedado finalmente el escape room. Gracias a los activos gratuitos que hemos encontrado en el Marketplace de Unity hemos conseguido decorar y ambientar bien la sala.

Finalmente, en la Figura 49, Figura 50, Figura 51 se muestran unos ejemplos de la página web donde se encuentra el mapa de calor. Estas imágenes no han sido tomadas a partir de una partida real, es por esto por lo que aparecen pocas posiciones por partida.



Figura 41. Resultado: menú inicial (izquierda), menú de registro (centro) y menú de inicio de sesión (Derecha)

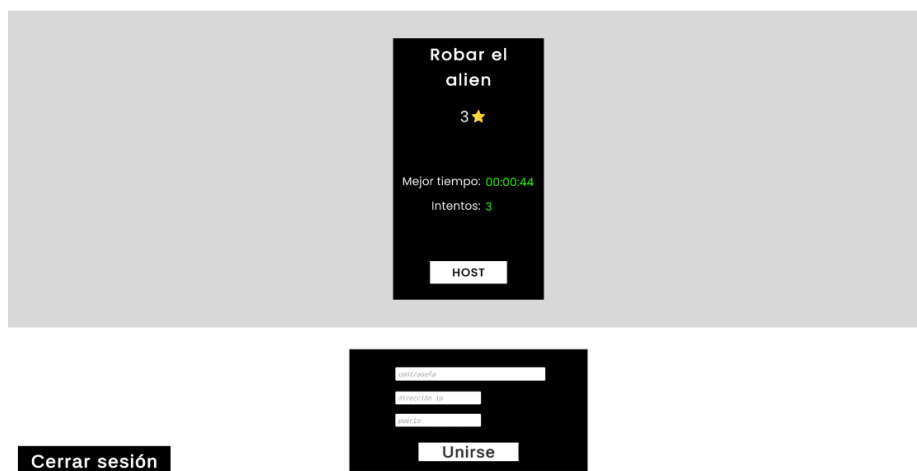


Figura 42. Resultado - Menú principal

5. Resultados



Figura 43. Resultado - Crear sala



Figura 44. Escape room visual 1



Figura 45. Escape room visual 2



Figura 46. Escape room visual 3



Figura 47. Escape room visual 4

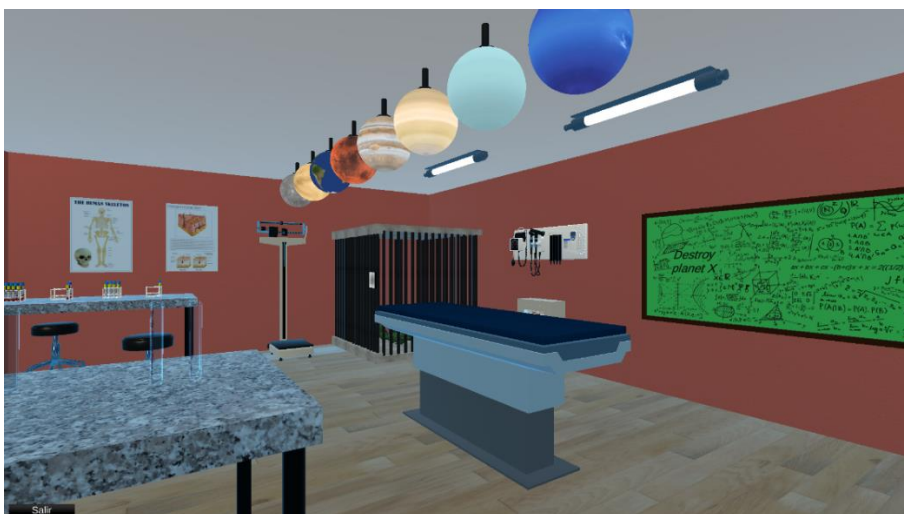


Figura 48. Escape room visual 5

Análisis del comportamiento de los jugadores

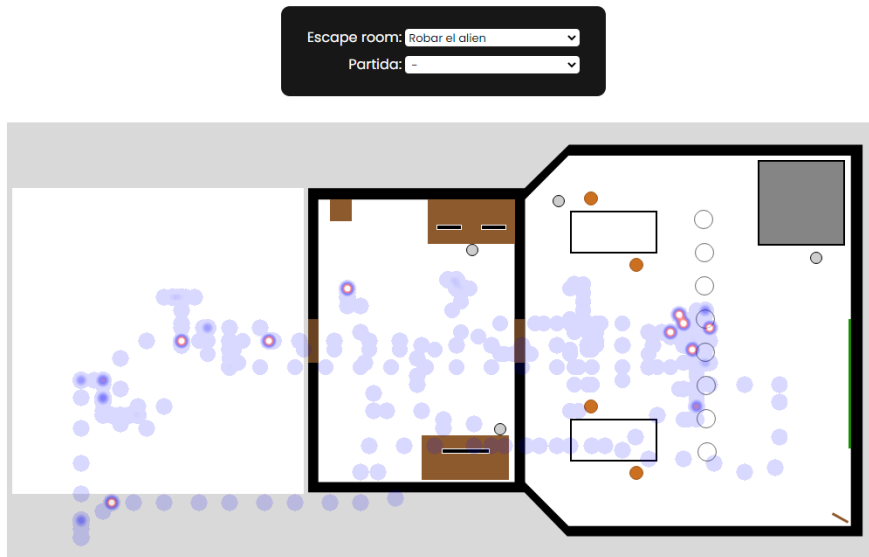


Figura 49. Resultado - Mapa de calor de un escape room

Análisis del comportamiento de los jugadores

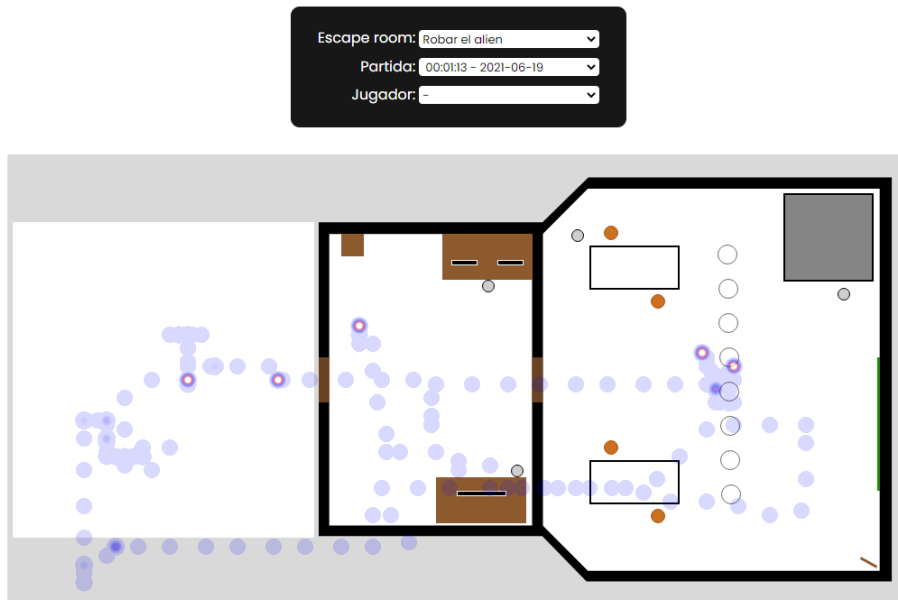


Figura 50. Resultado - Mapa de calor de una partida

Análisis del comportamiento de los jugadores

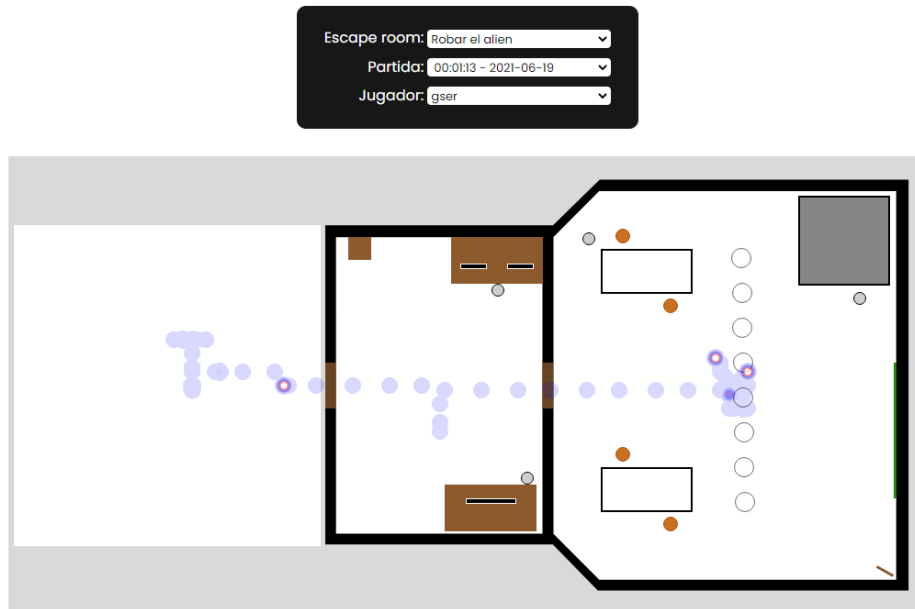


Figura 51. Resultado - Mapa de calor de un jugador en una partida

Capítulo 6. Conclusiones

La primera idea que sacamos en claro tras realizar este proyecto es que realizar un videojuego lleva mucho más tiempo del que puede parecer. Este tiempo se dispara aún más cuando has de utilizar una herramienta por primera vez, ya que previo a utilizarla te has de informar sobre cómo funciona. Y esto es algo que consume mucho tiempo.

En nuestro caso, por no saber con mucha precisión cuanto tiempo nos llevaría cada bloque de trabajo, no empezamos a desarrollar el juego teniendo en cuenta el multijugador, ya que en caso de ir muy mal de tiempo se podría no haber realizado este módulo. Sino que creamos el juego para jugar de manera individual y luego hicimos los cambios pertinentes para convertirlo en multijugador. Esto hace que te encuentres con problemáticas para las que tienes que improvisar una solución, que puede no ser la manera más correcta. Por tanto, algo que hemos aprendido es que, si se quiere implementar el multijugador, se debe diseñar el juego teniendo esto en mente desde un principio.

En cuanto al juego, para añadirle un punto extra de complejidad, decidimos que ciertos elementos fuesen dinámicos, como por ejemplo los códigos que desbloquean los candados y por tanto también algunos de los elementos que se utilizan como pistas. Si bien es cierto que el nivel de dinamismo que hemos aplicado nosotros no permite que la experiencia de un mismo escape room sea diferente al jugar más de una partida, sí que se podría conseguir aplicando un nivel de dinamismo mayor. Pero para realizar esto es necesario mucho más tiempo y un mayor conocimiento sobre las herramientas utilizadas.

Por otro lado, hasta que no estaban todos los bloques implementados no nos enfocamos en pulir los gráficos del juego y en ambientar correctamente el escape room. Pese a que ya comentamos en la introducción lo importante que es la ambientación en estos juegos, uno es mucho más consciente cuando lo está desarrollando y ha de ir haciendo pruebas únicamente con los elementos básicos.

Dicho esto, el objetivo era simular la arquitectura de un juego como los que salen hoy en día al mercado y yo personalmente aprender a utilizar las herramientas que se emplean para desarrollarlos. Creemos que hemos cumplido el objetivo y estamos satisfechos con el trabajo que hemos realizado.

Bibliografía

- [1] «AAA,» [En línea]. Available: [https://es.wikipedia.org/wiki/AAA_\(industria_del_videojuego\)](https://es.wikipedia.org/wiki/AAA_(industria_del_videojuego)). [Último acceso: 24 Junio 2021].
- [2] «Steam,» [En línea]. Available: <https://store.steampowered.com/?l=spanish>. [Último acceso: 10 Junio 2021].
- [3] «Unity,» [En línea]. Available: <https://unity.com/es>. [Último acceso: 14 Junio 2021].
- [4] «Unreal Engine,» [En línea]. Available: <https://www.unrealengine.com/en-US/>. [Último acceso: 14 Junio 2021].
- [5] «Godot,» [En línea]. Available: <https://godotengine.org/>. [Último acceso: 18 Junio 2021].
- [6] «C#,» [En línea]. Available: <https://docs.microsoft.com/es-es/dotnet/csharp/>. [Último acceso: 18 Junio 2021].
- [7] «C++,» [En línea]. Available: <https://www.cplusplus.com/>. [Último acceso: 18 Junio 2021].
- [8] «Java,» [En línea]. Available: <https://docs.oracle.com/javase/7/docs/technotes/guides/language/>. [Último acceso: 18 Junio 2021].
- [9] «Desarrollo ágil,» [En línea]. Available: https://ad.uib.es/estudis2021/pluginfile.php/215587/mod_resource/content/0/Gu%C3%ADa%20Pr%C3%A1ctica%20de%20C3%81gil%20-%20Cap%C3%ADtols%201-3.pdf.
- [10] «Desarrollo ágil,» [En línea]. Available: https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software. [Último acceso: 10 Junio 2021].
- [11] «Trello,» [En línea]. Available: <https://trello.com/>. [Último acceso: 10 Junio 2021].
- [12] «Kanban,» [En línea]. Available: <https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban>. [Último acceso: 10 Junio 2021].
- [13] «Adobe XD,» [En línea]. Available: <https://www.adobe.com/es/products/xd.html>. [Último acceso: 18 Junio 2021].

Bibliografía

- [14] «MonoBehaviour,» [En línea]. Available: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>. [Último acceso: 18 Junio 2021].
- [15] «SteamVR,» [En línea]. Available: https://valvesoftware.github.io/steamvr_unity_plugin/articles/intro.html. [Último acceso: 14 Junio 2021].
- [16] «XR Interaction Toolkit,» [En línea]. Available: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.10/manual/index.html>. [Último acceso: 18 Junio 2021].
- [17] «Blender,» [En línea]. Available: <https://www.blender.org/>. [Último acceso: 18 Junio 2021].
- [18] «ProBuilder,» [En línea]. Available: <https://unity3d.com/es/unity/features/worldbuilding/probuilder>. [Último acceso: 18 Junio 2021].
- [19] «BD relacional,» [En línea]. Available: https://es.wikipedia.org/wiki/Base_de_datos_relacional. [Último acceso: 18 Junio 2021].
- [20] «BD no relacional,» [En línea]. Available: <https://es.wikipedia.org/wiki/NoSQL>. [Último acceso: 18 Junio 2021].
- [21] «MySQL,» [En línea]. Available: <https://www.mysql.com/>. [Último acceso: 18 Junio 2021].
- [22] «Apache,» [En línea]. Available: <https://httpd.apache.org/>. [Último acceso: 18 Junio 2021].
- [23] «PHP,» [En línea]. Available: <https://www.php.net/manual/es/intro-what-is.php>. [Último acceso: 18 Junio 2021].
- [24] «Glassfish,» [En línea]. Available: <https://www.oracle.com/middleware/technologies/glassfish-server.html>. [Último acceso: 18 Junio 2021].
- [25] «Node.js,» [En línea]. Available: <https://nodejs.org/es/>. [Último acceso: 14 Junio 2021].
- [26] «HTTPS,» [En línea]. Available: https://es.wikipedia.org/wiki/Protocolo_seguro_de_transferencia_de_hipertexto. [Último acceso: 18 Junio 2021].
- [27] «Contraseñas php,» [En línea]. Available: <https://www.php.net/manual/es/faq.passwords.php>. [Último acceso: 24 Junio 2021].

-
- [28] «Password_hash(),» [En línea]. Available: <https://www.php.net/manual/es/function.password-hash.php>. [Último acceso: 24 Junio 2021].
- [29] «Sal,» [En línea]. Available: [https://es.wikipedia.org/wiki/Sal_\(criptograf%C3%ADa\)](https://es.wikipedia.org/wiki/Sal_(criptograf%C3%ADa)). [Último acceso: 24 Junio 2021].
- [30] «Password_verify(),» [En línea]. Available: <https://www.php.net/manual/es/function.password-verify.php>. [Último acceso: 24 Junio 2021].
- [31] «Photon,» [En línea]. Available: <https://www.photonengine.com/pun>. [Último acceso: 18 Junio 2021].
- [32] «Mirror,» [En línea]. Available: <https://mirror-networking.com/>. [Último acceso: 18 Junio 2021].
- [33] «MLAPI,» [En línea]. Available: <https://docs-multiplayer.unity3d.com/docs/getting-started/about-mlapi/index.html>. [Último acceso: 18 Junio 2021].
- [34] «NetworkObject,» [En línea]. Available: <https://docs-multiplayer.unity3d.com/docs/mlapi-basics/networkobject>. [Último acceso: 18 Junio 2021].
- [35] «Rpc,» [En línea]. Available: https://es.wikipedia.org/wiki/Llamada_a_procedimiento_remoto. [Último acceso: 18 Junio 2021].
- [36] «HTML,» [En línea]. Available: <https://developer.mozilla.org/es/docs/Web/HTML>. [Último acceso: 18 Junio 2021].
- [37] «CSS,» [En línea]. Available: <https://developer.mozilla.org/es/docs/Web/CSS>. [Último acceso: 24 Junio 2021].
- [38] «jQuery,» [En línea]. Available: <https://jquery.com/>. [Último acceso: 24 Junio 2021].
- [39] «Heatmap.js,» [En línea]. Available: <https://www.patrick-wied.at/static/heatmapsjs/>. [Último acceso: 24 Junio 2021].
- [40] «Corrutina,» [En línea]. Available: <https://es.wikipedia.org/wiki/Corrutina>. [Último acceso: 14 Junio 2021].
- [41] «Escape room,» [En línea]. Available: https://es.wikipedia.org/wiki/Escape_room. [Último acceso: 14 Junio 2021].

Bibliografía

- [42] «Motor de videojuego,» [En línea]. Available: https://es.wikipedia.org/wiki/Motor_de_videojuego. [Último acceso: 18 Junio 2021].