UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Análisis sobre la Gamificación para el trabajo en Soft Skills en la educación primaria: un caso práctico

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Marc Diago Sans

Tutor: Sara Blanc Clavero

Curso 2019-2020

Dedicatoria

Antes que nada, quiero dedicar este proyecto a mi familia, en especial a mis padres, por brindarme la oportunidad de estudiar este grado y apoyarme siempre que lo he necesitado.

También quiero dedicarlo tanto a los profesores que me han dado clase, como a aquellos a los que he conocido gracias a eventos y el día a día en la escuela.

A los amigos del pueblo, que me recuerdan siempre que bajo poco a verlos.

Por último, quiero dedicárselo a los compañeros que he ido conociendo durante todo el grado que han hecho que mi estancia en la escuela sea maravillosa.

Agradecimientos

En primer lugar me gustaría agradecer a mi tutora, Sara Blanc Clavero, por darme la oportunidad de realizar un proyecto en un ámbito que me apasiona y me permite ampliar mis conocimientos en el, por motivarme y por ayudarme a hacerlo posible.

También quiero agradecer a todos los miembros de University Game Dev, en especial a Sergio Rico Alfonso, por compartir sus conocimientos sobre patrones de diseño en videojuegos. A Carlos Longares Alcalá, Josué Navarro Martínez y Daniel Fenollar Onrubia por sus consejos desde la experiencia y ayudarme a acotar objetivos.

Agradezco especialmente la ayuda de los compañeros de la Start.inf probando los diferentes prototipos, ayudando a definir el videojuego y aguantándome todos los días.

A Ángeles Calduch Losa por enseñarme a usar las tildes.

A Iris López Martínez por el apoyo prestado.

Por último, pero no por ello menos importante, a toda la gente que me ha animado y ayudado durante todo este tiempo. Si tuviera que nombrarlos a todos no acabaría la lista.

Resum

Internet i els videojocs s'han convertit en una part del dia a dia dels xiquets, influint en la seua educació. Tots els dispositius mòbils, ordinadors i videoconsoles poden convertir-se en una ferramenta d'aprenentatge per als alumnes d'educació primària i secundària, no obstant això, els *Serious games* no estan completament introduïts en els col·legis, per la qual cosa encara hi ha moltes possibilitats d'explotar el seu potencial.

La primera part de l'estudi es basa a buscar aquells objectius d'aprenentatge que es vol reforçar per mitjà de l'ús del videojoc d'acord al Marc Europeu d'Habilitats Digital[9] i com adaptar-ho per a la majoria dels casos de discapacitats motores i visuals.

La segona part consistix en entrevistes al personal docent i a les professores d'educació especial del Col·legi La Purísima Hnas. Franciscanas, col·legi que s'ha oferit com pilot per a esta experiència, per a determinar que objectius curriculars, competències transversals i requisits d'adaptació curricular estan interessats a implementar dins del videojoc.

I la part final es basa en el desenrotllament d'un prototip del videojoc basant-se en els requisits formulats pel nostre potencial usuari, el col·legi mencionat. El dit prototip es provarà en un context real, amb alumnat del col·legi, integrant en la mostra també a estudiants amb necessitats especials i se'ls farà a tots ells un seguiment per mitjà d'enquestes per a determinar si l'experiment ha sigut efectiu.

Paraules clau: Soft-skills, educació, ludificació, videojocs, Unity

Resumen

Internet y los videojuegos se han convertido en una parte del día a día de los niños, influyendo en su educación. Todos los dispositivos móviles, ordenadores y videoconsolas pueden convertirse en una herramienta de aprendizaje para los alumnos de educación primaria y secundaria; sin embargo, los *Serious games* no están completamente introducidos en los colegios, por lo que aún hay muchas posibilidades de explotar su potencial.

La primera parte del estudio se basa en buscar aquellos objetivos de aprendizaje que se quiere reforzar mediante el uso del videojuego de acuerdo al Marco Europeo de Habilidades Digital [9] y cómo adaptarlo para la mayoría de los casos de discapacidades motoras y visuales.

La segunda parte consiste en entrevistas al personal docente y a las profesoras de educación especial del Colegio La Purísima Hnas. Franciscanas, colegio que se ha ofrecido como piloto para esta experiencia, para determinar qué objetivos curriculares, competencias transversales y requisitos de adaptación curricular están interesados en implementar dentro del videojuego.

Y la parte final se basa en el desarrollo de un prototipo del videojuego en base a los requisitos formulados por nuestro potencial usuario, el colegio mencionado. Dicho prototipo se probará en un contexto real, con alumnado del colegio, integrando en la muestra también a estudiantes con necesidades especiales y a todos ellos se les hará un seguimiento mediante encuestas para determinar si el experimento ha sido efectivo.

Palabras clave: Soft-skills, educación, ludificación, videojuegos, Unity

Abstract

Internet and video games have become a part of children's day to day, influencing their education. All mobile devices, computers and game consoles can become a learning tool for elementary and secondary school students, however, Serious games are not fully introduced in schools, so there are still many possibilities to exploit their potential.

The first part of the study is based on looking for those learning objectives that we want to reinforce by using the videogame according to the European Digital Competence Framework[9] and how to adapt it for most cases of motor and visual disabilities.

The second part consists of interviews with teachers and special education teachers of La Purísima Hnas. Franciscanas school, a school that has offered itself as a pilot for this experience, to determine what curricular objectives, transversal competencies and curricular adaptation requirements are interested in implementing within the videogame.

And the final part is based on the development of a video game prototype based on the requirements formulated by our potential user, the mentioned school. This prototype will be tested in a real context, with students from the school, also integrating students with special needs in the sample and all of them will be monitored through surveys to determine if the experiment has been effective.

Key words: Soft-skills, education, gamification, videogames, Unity

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Desde hace tiempo, la fama de los videojuegos ha sido negativa por parte de la sociedad en la relación a los niños y el aprendizaje, viéndose como algo que les perjudica, ya que les distrae de sus obligaciones y hace que no adquieran conocimientos realmente útiles para su futuro. Incluso se atribuye en general sucesos violentos al uso de los videojuegos, reforzados por periódicos y cadenas de televisión.

En los últimos años se ha visto un creciente interés por desarrollar videojuegos educativos muy simples que se basan en preguntas sobre materias impartidas en la escuela, y que por su simplicidad, no son atractivos para el público objetivo y por consecuencia los resultados no son los esperados. Por ello, los videojuegos educativos se han visto reducidos a simples "minijuegos"divididos por categorías [1]. Actualmente, los niños empiezan a jugar a juegos de grandes presupuestos, también llamados AAA o triple A, y esto supone una pérdida de interés en los videojuegos de tipo Flash, llamados así por el motor gráfico que usan, los cuales ofertan las páginas web para jugarlos en el navegador.

Con la reaparición de la gamificación, también llamada ludificación, por parte de las aerolínias comerciales para hacer programas de fidelización para sus clientes, el mundo académico comenzó a interesarse por esta metodología para ver si podía aplicarse en otros ámbitos. Actualmente, es utilizada para hacer más atractivas las actividades de aprendizaje en las empresas, y que así se mejore la productividad y eficacia de los trabajadores. Esto llevó incluso a la introducción de la gamificación en entrenamientos virtuales para trabajos peligrosos o incluso para entrenamientos del ejército.

La ludificación de la educación infantil se ha llevado a cabo mediante juegos de mesa o actividades físicas que resultaron beneficiosas para los estudiantes, lo que les ayudó a desarrollar competencias y conocimientos de una forma más cómoda. La gamificación de los videojuegos en la educación de los niños puede ser interesante, ya que introduciría el aprendizaje tecnológico del que los métodos tradicionales carecen y que pertenecería a un sector del videojuego que hoy en día se encuentra en auge.

Este análisis pretende desarrollar y prototipar un videojuego que sea atractivo y eficiente para la educación primaria y secundaria, tanto mecánicas, jugabilidad y controles, como tipografía y colores que permitan que incluso estudiantes con discapacidades puedan disfrutar y aprender con él. Aunque el aprendizaje curricular se encuentra dentro de los objetivos de la gamificación, este proyecto no se centra exclusivamente en este apartado, sino que se enfoca al desarrollo de competencias transversales de carácter social, de comportamiento o planificación de acuerdo a *OECD Science, Technology and Innovation Outlook 2018* [1].

[1] https://www.mundoprimaria.com/juegos-educativos

1.1 Motivación

De la misma forma que se ha explicado en la introducción, el motivo del autor es desarrollar un videojuego educativo que sea atractivo y útil para los estudiantes de primaria y secundaria.

El motivo por el que se ha elegido este tema es para dar visibilidad a que no todos los videojuegos son violentos y perjudiciales para los niños, jóvenes o incluso adolescentes; sino que pueden ser de provecho y permitirles adquirir habilidades sociales, de comportamiento y de planificación mediante un medio al que muchos de ellos están acostumbrados.

1.2 Objetivos

El objetivo principal del proyecto es desarrollar un prototipo de gamificación de las *soft skills*, también llamadas competencias, en forma de videojuego que permita a los estudiantes mejorar sus competencias transversales y rendimiento académico.

Como ya se ha explicado, el proyecto consta de tres partes y objetivos: el estudio basado en buscar objetivos de aprendizaje que se quiere reforzar con el uso de videojuegos, las entrevistas al personal docente y estudiantes del Colegio La Purísima Hnas. Franciscanas para determinar los objetivos curriculares, sociales y digitales y, finalmente, el desarrollo de un prototipo en base a los dos puntos anteriores, que será probado por los alumnos del colegio anteriormente indicado.

Como objetivos secundarios de este TFG, uno de los puntos que más interés tiene, es que el prototipo del videojuego sea diseñado para que la mayor cantidad de estudiantes pueda usarlo, teniendo en cuenta la integración del alumnado con necesidades especiales o adaptaciones curriculares. Por ejemplo, se han tenido en cuenta necesidades de accesibilidad para la dislexia o el daltonismo. Además, el juego también puede adaptarse a diferentes materias curriculares o refuerzos no curriculares, por ejemplo, para entrenar la lecto-escritura entre otros.

La configuración del juego se realiza a través de un archivo de un método sencillo que cualquier profesor, aún sin experiencia en informática, pueda modificar los textos dentro del juego y cambiar las preguntas que conforman la base de aprendizaje del videojuego. Además, al ser un texto libre, es abierto a diferentes idiomas, lo que amplía su usabilidad.

Con la finalidad de garantizar que cualquiera pueda hacer uso del videojuego, se siguen el conjunto de buenas prácticas explicadas en la rama de Ingeniería de Software, que incluyen patrones de software para que sea escalable, documentación del código e instrucciones de uso.

1.3 Estructura de la memoria

La memoria está dividida en los siguientes capítulos:

- **Estado de la cuestión:** Se describe la situación de los videojuegos en la sociedad y en especial, de los videojuegos formativos o *Serious games*.
- **Análisis del problema:** Se especifican los casos de uso, actores y los requisitos del sistema, así como la identificación de las posibles soluciones y qué solución ha sido propuesta.

- **Diseño de la solución:** Se presenta el diseño del videojuego y su arquitectura, además de especificar el diseño detallado del sistema.
- **Desarrollo de la solución propuesta:** Se hace una revisión de todas las fases del desarrollo del proyecto, desde el primer análisis hasta el desarrollo del prototipo, prestando atención a los problemas que surgieron durante el proceso.
- **Implantación:** En este capítulo se presenta la etapa en la que se ha realizado el desarrollo del prototipo y se ha llevado a la explotación.
- **Pruebas:** Se detallan las pruebas realizadas para el prototipo del videojuego.
- **Conclusiones:** Conclusión de la memoria, detallando problemas y errores cometidos y confirmando que los objetivos iniciales se han cumplido. También se indica la relación entre este trabajo y las asignaturas cursadas durante el grado.
- **Trabajos futuros:** Listado de sistemas o diseños a mejorar de cara a una futura versión del proyecto.
- **Bibliografía:** Listado de referencias bibliográficas usadas durante la memoria.
- **Apéndice:** Listado de toda la información adicional que se agrega a la memoria, como un listado de los acrónimos o el archivo de configuración.

CAPÍTULO 2
Estado de la cuestión

El sector de los videojuegos es una industria que está en auge. Es raro encontrar a alguna persona que no conozca los nombres *League of Legends*, *Fortnite* o *Minecraft*. Estos videojuegos son parte de la cultura de los jóvenes. Eventos como los torneos mundiales de *League of Legends*, llamados *Worlds*, son una muestra del interés que hay detrás de estos juegos. Los *Worlds* de 2018 sumaron un total de dos millones de espectadores[1], mientras que los *Worlds* 2019, parte de cuya competición se situó en España, sumaron casi cuatro millones de espectadores[2], duplicando la cantidad de espectadores de un año a otro. Y no solo *League of Legends*, la *Fortnite World Cup 2019* alcanzó la suma de más de dos millones de espectadores[3].

Dentro de los videojuegos están los llamados *E-Sport*, los juegos anteriormente mencionados son ejemplos de estos. Estos *E-Sports* son deportes electrónicos cuyos jugadores se están empezando a considerar deportistas de élite en algunos países. Pero no todo son deportes, un debate del día a día es si estos se pueden considerar arte, ya que dentro del catálogo de videojuegos existentes hay un lugar para juegos considerados por la prensa especializada como obras de arte. Juegos como *Gris*, de *Nomada Studio* situados en Barcelona, *Return of the Obra Dinn*, del autor americano Lucas Pope, o *Inside*, del estudio danés *Playdead*, son muy buenos ejemplos de este tipo de juegos.

La sociedad percibe a los videojuegos como algo cuya función es puramente entretenimiento, pero los videojuegos pueden ir más allá de esto. Con la introducción de la gamificación en forma de videojuegos de algunos sectores, nacieron los denominados *Serious games*, juegos serios o también llamados juegos formativos, los cuales son el fruto de gamificar en forma de videojuego cualquier competencia con una finalidad diferente al entretenimiento [5].

Juegos como *Grand Thieft Auto*, conocido por su violencia explícita y contenido para adultos, disponen de sistemas muy pulidos, como el de la conducción, los cuales son prestados por sus creadores, *Rockstar Games*, para su uso en otros ámbitos y la creación de juegos serios [14]. En este caso, dicho sistema de conducción, fue utilizado por autoescuelas e incluso universidades para realizar simulaciones de tráfico desde que la compañía dejó abierto el juego a modificaciones. Otros juegos, como *America's Army* producido por la armada de Estados Unidos, son usados para el entrenamiento militar.

Respecto a la gamificación, ha tenido una rápida expansión en cuanto a iniciativas, en especial la educativa, y nos encontramos en un área relativamente nueva en la que no existe una clasificación para definir qué elementos debe tener un videojuego educativo, aunque un factor común es la importancia de la motivación [6], ya que sin ella, los alum-

[1] https://escharts.com/tournaments/lol/worlds-2018
[2] https://escharts.com/tournaments/lol/2019-world-championship
[3] https://escharts.com/tournaments/fortnite/fortnite-world-cup-finals

nos no aprovecharán ni conseguirán todos los beneficios que el videojuego les intente proveer. *Profesor Layton* de *Level-5* o *Brain Training* de *Nintendo* son dos videojuegos que se enfocan principalmente en enseñar a los jugadores a resolver problemas de lógica. El primero incluye una narrativa, un recurso que utilizan los videojuegos para atraer al jugador mediante una historia o sucesos para que siga jugando e interesándose por resolver más rompecabezas, pero ambos juegos están encerrados en el ámbito de los problemas de lógica.

Recientemente en 2018, *Ubisoft Montreal* reutilizó recursos de su videojuego *Assassin's Creed: Origins* para crear el *Discovery Tour* [15], una herramienta educativa que permite al jugador recorrer una recreación del Antiguo Egipto, aunque no sería correcto definir este modo como videojuego, se puede observar cómo el interés de llevar estos juegos al sector educativo empieza a ser atractivo para las empresas más grandes. Aunque los trabajos realizados por las grandes empresas para la educación son una buena señal para el estado del arte, estos videojuegos educativos suelen estar centrados en enseñar algo. Esto se debe a que la idea y diseño proviene de profesores, pasando por alto el diseño de videojuegos, el cual insiste en que ante todo, deben ser entretenidos o evocar emociones al jugador.

Con la aparición de los motores gráficos gratuitos para el desarrollo de videojuego, el proceso se ha simplificado y ya no se requiere de equipos de más de cincuenta personas para el desarrollo de estos. Motores como Unity 3D o Unreal Engine 5, tienen planes de negocio centrado en estudiantes y equipos pequeños a los cuales no se les cobra una cuota. Esto ha promovido la aparición de muchos grupos de desarrollo pequeños, llamados *Indie* que realizan videojuegos simples y de bajo presupuesto, además de la creación de programas para estudiantes, como *Unity Student Ambassador* que apuestan por tutorizar y realizar talleres para que los alumnos universitarios interesados puedan iniciar sus primeros pasos durante su estancia en los grados.

Gracias a lo comentado anteriormente, nos encontramos en un momento en el que es más fácil conectar a estudiantes con conocimientos de desarrollo de videojuegos y profesores de escuelas de primaria o secundaria y que juntando ambas disciplinas, puedan aparecer videojuegos educativos cuyas competencias curriculares y transversales puedan aprenderse siguiendo unos diseños de videojuegos reales, que no solo harán que los alumnos se entretengan aprendiendo, sino que los profesores podrán ver desde otro punto de vista, toda la psicología detrás de los sistemas para entretener y mantener su interés de los jugadores y aplicarla a sus clases.

2.1 Crítica al estado de la cuestión

Ruth S. publicó en 2016 unos artículos sobre los juegos formativos en la educación [10, 11], indicando sus beneficios y posibilidades y también proponiendo un prototipo, el cual solo se centra en el desarrollo del alumno a nivel individual. Los videojuegos educativos actuales se centran en un único jugador, ignorando la mejora de las competencias sociales como la comunicación, liderazgo o trabajo en equipo. Juegos similares a la propuesta de Ruth, se centran en el apartado de juegos serios que pone el aprendizaje por delante del entretenimiento, y esto es algo que no debería pasar. Los videojuegos deben evocar emociones que atrapen al jugador y, gracias a esta inmersión deseada en el juego, dicho jugador aprenda.

2.2 Propuesta

En este trabajo se propone un videojuego que, respetando la filosofía detrás del desarrollo de videojuegos actual, permita a los jugadores desarrollar sus *soft skills* así como sus competencias curriculares. El videojuego propuesto será de carácter cooperativo, ya que se busca incentivar a los alumnos a trabajar juntos y no a competir, además de inclusivo, buscando unos controles, mecánicas, tipografía y colores que no deje a ningún alumno sin poder participar.

El videojuego busca mejorar los resultados de otros videojuegos educativos existentes, haciendo que la carga de enseñanza y entretenimiento estén a la par y así motivarles a seguir jugando y aprendiendo. No se busca que el videojuego substituya las clases lectivas, pero sí que se use como un apoyo a estas.

El prototipo se ha desarrollado mediante el motor gráfico Unity3D, ya que dispone de unas herramientas que permiten que un juego simple en 2D pueda ser desarrollado sin necesidad de librerías externas. Es un motor que permite realizar videojuegos de forma flexible y con la capacidad de exportarlo a la mayoría de sistemas operativos, por lo que se ha considerado ideal para este proyecto.

CAPÍTULO 3
Análisis del problema

En el desarrollo de videojuegos, al igual que en el desarrollo de cualquier aplicación software, antes de empezar a programar, hay que realizar la especificación de requisitos software. En esta especificación hay que definir una descripción de los objetivos y comportamiento del sistema, evitando así mal interpretaciones de los objetivos y una vez finalizado el desarrollo, hay que comprobar si todos los requisitos funcionales y no funcionales se han cumplido y que todos los casos de uso se satisfacen.

3.1 Casos de uso

En esta sección se detallan los casos de uso que se han definido durante el primer análisis del prototipo. Estos casos de uso definen la funcionalidad que los diferentes actores pueden realizar en el videojuego.

3.1.1. Actores

El actor principal del videojuego es el jugador, ya que es el que va a participar directamente en la aplicación. Pero no hay que olvidar que los profesores se encargan de suministrar preguntas y material al juego, así que, estos también se consideran actores en este contexto y deben tener casos de uso definiendo su funcionalidad.

- *Jugador*: Es el rol de actor por defecto en los videojuegos. Cualquier estudiante, profesor, padre o persona sin relación en el ámbito educativo que participe en el juego, es un jugador. En nuestro caso, el videojuego está enfocado a niñas y niños de trece años, aunque podrían llegar a jugar personas de menor edad, así que nadie debería tener problemas, independientemente del nivel de estudios que tenga.

- *Profesor*: El rol de este actor se centra en la modificación de las preguntas, pistas y textos que contiene el videojuego. La persona que ocupe este rol, deberá tener conocimientos curriculares de la materia o refuerzo que se quiere implementar. A pesar de que el videojuego se ha desarrollado con el fin de mejorar las competencias transversales, sociales y curriculares de los jugadores, el contenido introducido no tiene porqué ser del ámbito curricular, ya que la configuración da libertad a este rol para añadir el contenido que desee. En el anexo B hay información sobre este archivo de configuración.

3.1.2. Especificación de los casos de uso

En este apartado se detalla la tabla utilizada para definir los casos de uso de los actores que intervienen en el videojuego. La tabla definida es la siguiente:

Identificador

Nombre	
Descripción	
Actor	
Precondición	
Flujo	
Postcondición	

Tabla 3.1: Tabla de ejemplo de los casos de uso

El significado de los campos es el siguiente:

- *Identificador*: Código identificativo único de cada caso de uso.
- *Nombre*: Nombre descriptivo del caso de uso.
- *Descripción*: Definición simple y concisa de lo que especifica el caso de uso.
- *Actor*: Rol que realiza el caso de uso.
- *Precondición*: Condicion requerida para poder realizar la funcionalidad especificada.
- *Flujo*: Funcionalidad del caso de uso paso por paso.
- *Postcondición*: Estado del sistema tras ejecutar el caso de uso.

Casos de uso: Jugador

Como se puede observar en la *Figura 3.1*, aquí están definidos los casos de uso que el jugador podrá realizar desde el menú principal del videojuego.

Estas funcionalidades son iniciar partida, salir del juego o modificar los nombres de los jugadores. Se busca un menú simple sin mucha funcionalidad para que sea fácil de comprender.

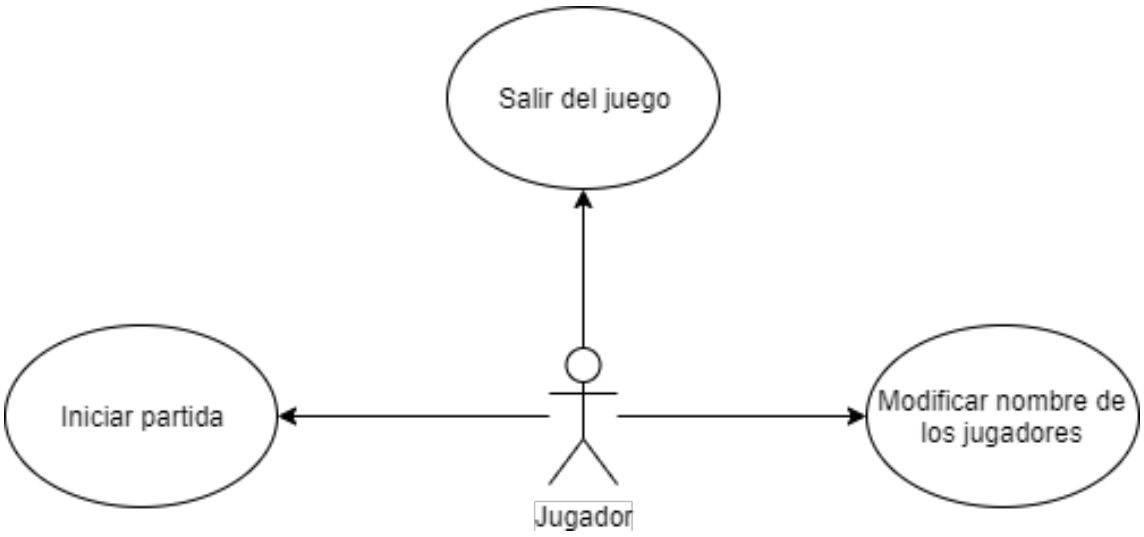

Figura 3.1: Casos de uso: Jugador en menú principal

CU01

Nombre	Iniciar partida.
Descripción	El jugador podrá iniciar partida.
Actor	Jugador.
Precondición	Ninguna.
Flujo	1. El jugador pulsa en el botón *Iniciar partida*.
Postcondición	La partida se inicia, notificando al sistema con los nombres de los personajes.

Tabla 3.2: CU01: Iniciar partida

CU02

Nombre	Salir del juego.
Descripción	El jugador podrá salir del juego en cualquier momento.
Actor	Jugador.
Precondición	Ninguna.
Flujo	1. Pulsar el botón *Salir del juego*. 2. Pulsar *Sí* en la confirmación.
Postcondición	La aplicación se cerrará.

Tabla 3.3: CU02: Salir del juego

CU03

Nombre	Modificar nombre de los jugadores.
Descripción	El jugador podrá modificar el nombre de los personajes.
Actor	Jugador.
Precondición	Ninguna.
Flujo	1. Pulsar en el avatar del jugador que desea cambiar el nombre. 2. Una vez escrito el nuevo nombre, confirmar pulsando la tecla *Enter*.
Postcondición	El nombre del avatar será modificado.

Tabla 3.4: CU03: Modificar nombre de los jugadores

La *Figura 3.2* corresponde a la funcionalidad que el jugador puede realizar durante la partida. Las partidas consisten en asignar dados para que el personaje mueva, usar poderes específicos, decidir qué camino tomará el jugador en una bifurcación, mover la cámara, consultar las pistas si ha caído en una casilla de investigación y realizar rompecabezas en las casillas de rompecabezas. No se va a volver a especificar el caso de uso de *Salir del juego*, ya que es el mismo que en el menú principal.

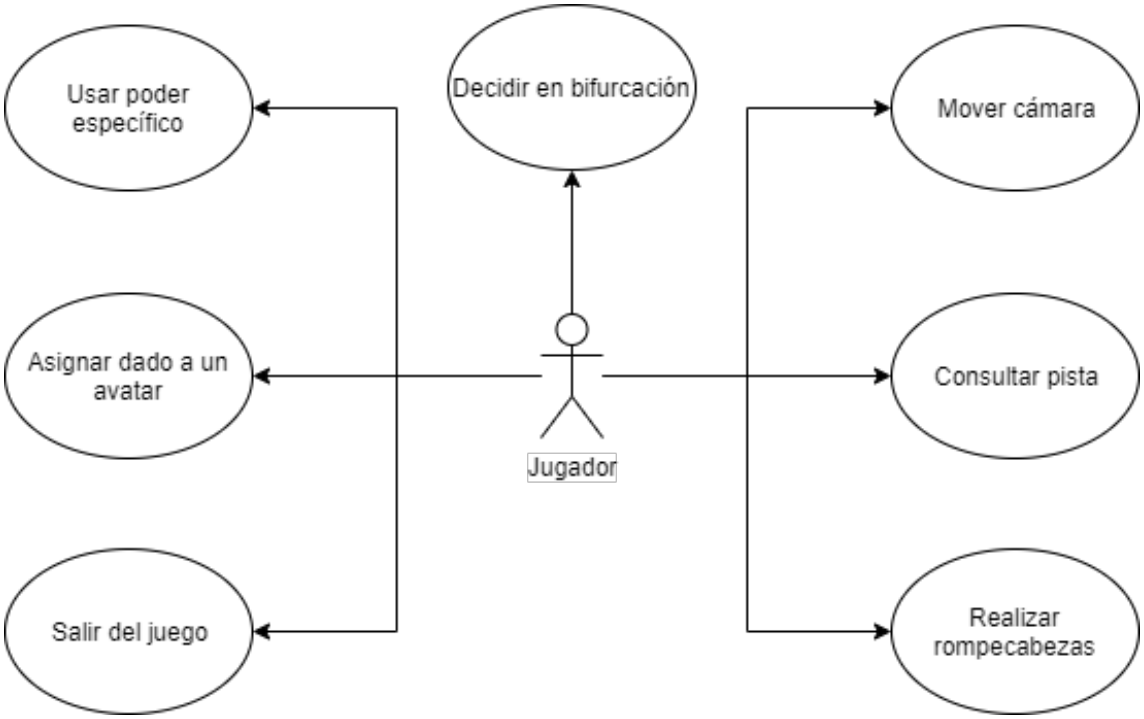

Figura 3.2: Casos de uso: Jugador durante la partida

CU04

Nombre	Asignar dado a un avatar.
Descripción	Un jugador puede arrastrar uno de los dados que hay en la caja de dados a un avatar para que este mueva tantas casillas como valor hay en el dado.
Actor	Jugador.
Precondición	No haya acciones pendientes de otros casos de uso.
Flujo	1. Seleccionar un dado de la caja de dados. 2. Arrastrarlo hasta un avatar.
Postcondición	El personaje asociado al avatar avanzará tantas casillas como valor tenga el dado usado.

Tabla 3.5: CU04: Asignar dado a un avatar

CU05

Nombre	Usar poder específico.
Descripción	Un jugador puede activar el poder específico de cualquier personaje para alterar la partida a su favor.
Actor	Jugador.
Precondición	Hay usos de poderes restantes y no se está realizando ninguna otra acción.
Flujo	1. Pulsar en el botón *Poder* situado junto a los avatares.
Postcondición	La partida se alterará dependiendo del poder usado.

Tabla 3.6: CU05: Usar poder específico

CU06

Nombre	Decidir en bifurcación.
Descripción	Si durante el movimiento el personaje tiene que avanzar por una casilla que tiene dos destinos diferentes, este se detendrá y el jugador deberá decidir qué camino tomar.
Actor	Jugador.
Precondición	Hay movimientos de casilla pendientes y la casilla actual tiene dos destinos distintos.
Flujo	1. El personaje se detiene. 2. Aparecen flechas apuntando a las casillas destino de la actual. 3. El jugador decide qué camino seguir pulsando en la flecha correspondiente.
Postcondición	El personaje sigue sus movimientos por el camino indicado.

Tabla 3.7: CU06: Decidir en bifurcación

CU07

Nombre	Mover cámara.
Descripción	El jugador podrá cambiar el personaje activo para mover la cámara a dicho personaje.
Actor	Jugador.
Precondición	La cámara no se encuentra en el personaje activo.
Flujo	1. El jugador pulsa en un avatar para convertirlo en el personaje activo. 2. La cámara se mueve hasta dicho personaje.
Postcondición	La cámara se posiciona en el personaje activo.

Tabla 3.8: CU07: Mover cámara

CU08

Nombre	Consultar pista.
Descripción	El jugador podrá leer la pista obtenida.
Actor	Jugador.
Precondición	El personaje finaliza su movimiento en una casilla de investigación.
Flujo	1. El personaje cae en una casilla de investigación y se abre una ventana con la pista. 2. El jugador puede leer la pista y cerrar la ventana cuando quiera para seguir el juego.
Postcondición	Ninguna.

Tabla 3.9: CU08: Consultar pista

CU09

Nombre	Realizar rompecabezas.
Descripción	El jugador debe solucionar el rompecabezas.
Actor	Jugador.
Precondición	El personaje ha caído en una casilla de rompecabezas.
Flujo	1. El personaje cae en una casilla de rompecabezas y se abre una ventana con el problema. 2. El jugador puede completarlo o fallar.
Postcondición	▪ Si completa el rompecabezas, se abre un camino junto a dicha casilla y esta se convierte en una casilla normal. ▪ Si no completa el rompecabezas, finaliza su activación y hay que volver a usar los dados.

Tabla 3.10: CU09: Realizar rompecabezas

Casos de uso del profesor

Para finalizar con esta sub-sección, se muestra el casos de uso que corresponde al rol del profesor, el cual es simplemente modificar la configuración.

Figura 3.3: Casos de uso: Profesor

CU10

Nombre	Modificar la configuración.
Descripción	El profesor podrá modificar la configuración del juego para adaptarlo al temario que quiera reforzar.
Actor	Profesor.
Precondición	El juego se ha instalado y se ha generado el archivo de configuración.
Flujo	1. El profesor utiliza la aplicación o un editor de texto para modificar el archivo.
Postcondición	El juego se reconfigura para cargar los nuevos textos y preguntas.

Tabla 3.11: CU10: Modificar la configuración

3.2 Especificación de requisitos

A continuación se definen los requisitos software del videojuego. Estos requisitos se han generado a partir de reuniones con la tutora del proyecto y los profesores del Colegio La Purísima Hnas. Franciscanas.

Los requisitos se representarán mediante la siguiente tabla:

Identificador	
Nombre	
Descripción	
Prioridad	

Tabla 3.12: Tabla ejemplo requisitos funcionales

Cuyo significado de los campos es el siguiente:

- *Identificador*: Código identificativo único de cada requisito.
- *Nombre*: Nombre descriptivo del requisito.
- *Descripción*: Definición simple y concisa del requisito especificado.
- *Prioridad*: Urgencia para realizar el requisito.

Estos requisitos se separan en dos tipos, para facilitar la comprensión de estos. Los tipos son los siguientes:

- *Requisitos funcionales*: Define un comportamiento de una función que contiene el software.
- *Requisitos no funcionales*: Propiedades o cualidades que la aplicación software debe cumplir.

3.2.1. Requisitos funcionales

Las siguientes tablas corresponden a los requisitos funcionales del videojuego:

RF-01	
Nombre	Iniciar nueva partida.
Descripción	El juego deberá permitir iniciar una nueva partida.
Prioridad	Alta.

Tabla 3.13: RF-01: Iniciar nueva partida

RF-02	
Nombre	Salir del juego.
Descripción	El juego ofrecerá la posibilidad de salir de ella cuando el jugador lo desee.
Prioridad	Alta.

Tabla 3.14: RF-02: Salir del juego

RF-03

Nombre	Modificar nombre de personaje.
Descripción	El jugador podrá modificar el nombre del personaje desde el menú principal, antes de iniciar una partida. Este nombre se mantendrá para la partida completa.
Prioridad	Media.

Tabla 3.15: RF-03: Modificar nombre de personaje

RF-04

Nombre	Leer archivo de configuración.
Descripción	El sistema debe ser capaz de leer el archivo de configuración compuesto de los siguientes parámetros: ▪ Número de jugadores máximo ▪ Enunciado de las preguntas ▪ Respuestas correctas y incorrectas a las preguntas ▪ Textos y pistas del videojuego
Prioridad	Alta.

Tabla 3.16: RF-04: Leer archivo de configuración

RF-05

Nombre	Generar dados.
Descripción	El sistema debe generar tantos dados como jugadores en partida cuando la caja de dados esté vacía.
Prioridad	Alta.

Tabla 3.17: RF-05: Generar dados

RF-06

Nombre	Arrastrar dados.
Descripción	El jugador debe ser capaz de mover los dados y sacarlos de la caja de dados. Si un dado es soltado en una posición que no es válida, volverá a su posición original en la caja.
Prioridad	Alta.

Tabla 3.18: RF-06: Arrastrar dados

RF-07

Nombre	Asignar movimiento a personaje
Descripción	Soltar un dado en un avatar destruirá dicho dado y asignará ese movimiento al personaje asignado a ese avatar.
Prioridad	Alta.

Tabla 3.19: RF-07: Asignar movimiento a personaje

RF-08

Nombre	Activar poderes.
Descripción	El jugador deberá activar el poder correspondiente a su personaje para modificar el estado del juego al asignar el siguiente dado. Los poderes son los siguientes: ▪ Ignorar casillas de peligro. ▪ Mover hasta una casilla de rompecabezas. ▪ El valor del dado usado se incrementa en uno. ▪ La casilla destino será de tipo investigación además de su tipo original.
Prioridad	Media.

Tabla 3.20: RF-08: Activar poderes

RF-09

Nombre	Mover personaje.
Descripción	El sistema moverá el personaje tanto lógica como visualmente por el tablero, cuando este reciba una cantidad de movimiento. Mientras el personaje se mueve, el jugador no puede realizar acciones hasta que encuentre una bifurcación o acabe.
Prioridad	Alto.

Tabla 3.21: RF-09: Mover personaje

RF-10

Nombre	Decidir en bifurcación.
Descripción	Cuando un personaje vaya a abandonar una casilla que tiene dos o más casillas destino, este se bloqueará y se generarán flechas que apuntarán a esos destinos. El jugador deberá pulsar en la flecha que marque el camino al cual se quiere dirigir. Una vez pulsado, se vuelve al movimiento normal.
Prioridad	Alta.

Tabla 3.22: RF-10: Decidir en bifurcación

RF-11

Nombre	Mover cámara a un personaje.
Descripción	Cuando un personaje se mueve o se pulsa en su avatar, el sistema debe asignarlo como *personaje activo*, lo cual centrará y seguirá sus movimientos hasta que otro sea elegido.
Prioridad	Alta.

Tabla 3.23: RF-11: Mover cámara a un personaje

RF-12

Nombre	Abrir ventana de pistas.
Descripción	Cuando un personaje cae en una casilla de investigación, se abrirá una ventana mostrando una pista al azar para que el jugador pueda leerla.
Prioridad	Media.

Tabla 3.24: RF-12: Abrir ventana de pistas

RF-13

Nombre	Caer en una casilla de peligro.
Descripción	Cuando un personaje cae sobre una casilla de peligro, esta moverá al personaje hasta otra casilla que la casilla de peligro tenga enlazada. Este movimiento no disparael efecto de la casilla donde cae.
Prioridad	Media.

Tabla 3.25: RF-13: Caer en una casilla de peligro

RF-14

Nombre	Mostrar rompecabezas.
Descripción	Si un jugador cae en una casilla de rompecabezas, esta cargará la pregunta que tiene asignada. Se leerá del archivo de configuración el enunciado y las respuestas y se mostrarán mediante un texto y botones para elegir.
Prioridad	Alta.

Tabla 3.26: RF-14: Mostrar rompecabezas

RF-15

Nombre	Comprobar respuesta.
Descripción	Cuando un jugador seleccione una respuesta, el sistema debe comprobar si es correcta. De acertar, la casilla rompecabezas se convertirá en una normal sin efecto y se enlazará a una casilla nueva que abrirá otra parte del tablero. Si falla, no ocurre nada.
Prioridad	Alta.

Tabla 3.27: RF-15: Comprobar respuesta

RF-16

Nombre	Menú principal.
Descripción	El videojuego dispondrá un menú principal al entrar al juego desde el se podrá iniciar una nueva partida, modificar los nombres o salir.
Prioridad	Alta.

Tabla 3.28: RF-16: Menú principal

RF-17

Nombre	Menú de pausa.
Descripción	Dentro de la partida, los jugadores podrán pulsar la tecla *Escape* para acceder a este menú que les permitirá volver al menú principal o salir del juego.
Prioridad	Alta.

Tabla 3.29: RF-17: Menú de pausa

RF-18

Nombre	Volver al menú principal.
Descripción	Cuando un jugador pulse en el botón de *Volver al menú principal*, el sistema finalizará la partida, perdiendo todo el progreso y volviendo al menú principal.
Prioridad	Media.

Tabla 3.30: RF-18: Volver al menú principal

RF-19

Nombre	Comprobar final de partida.
Descripción	El juego comprobará si todos los personajes están en la casilla de meta para comprobar si el juego se ha acabado.
Prioridad	Media.

Tabla 3.31: RF-19: Comprobar final de partida

RF-20

Nombre	Música y efectos de sonido.
Descripción	El sistema deberá usar los métodos nativos del motor gráfico para reproducir efectos de sonido y música.
Prioridad	Bajo.

Tabla 3.32: RF-20: Música y efectos de sonido

RF-21

Nombre	Modificar contenido.
Descripción	El profesor podrá modificar el contenido del juego y el sistema deberá mostrar los datos de acuerdo a la configuración especificada.
Prioridad	Alta.

Tabla 3.33: RF-21: Modificar contenido

Requisitos no funcionales

Las siguientes tablas corresponden a los requisitos no funcionales del videojuego:

RNF-01

Nombre	Compatibilidad con sistemas operativos.
Descripción	El videojuego debe ser compatible con los sistemas operativos Windows y Lliurex.
Prioridad	Alta.

Tabla 3.34: RNF-01: Compatibilidad con sistemas operativos

RNF-02

Nombre	Compatibilidad de resoluciones.
Descripción	El videojuego debe ser capaz de adaptarse a cualquier resolución, desde 800x600 o superior.
Prioridad	Alta.

Tabla 3.35: RNF-02: Compatibilidad de resoluciones

RNF-03

Nombre	Compatibilidad con formato de audios.
Descripción	El sistema debe permitir el uso de los principales formatos de audio: MP3, WAV, MIDI, OGG y ACC.
Prioridad	Media.

Tabla 3.36: RNF-03: Compatibilidad con formato de audios

RNF-04

Nombre	Textos legibles y escalables.
Descripción	Para evitar que jugadores con problemas visuales tengan dificultades para leer, el juego usará la fuente *verdana* y los jugadores podrán modificar su tamaño.
Prioridad	Alta.

Tabla 3.37: RNF-04: Textos legibles y escalables

RNF-05

Nombre	Interfaz simple e intuitiva.
Descripción	La interfaz debe ser simple e intuitiva para que ningún jugador requiera de conocimientos de videojuegos o tecnológicos para poder usarla.
Prioridad	Alta.

Tabla 3.38: RNF-05: Interfaz simple e intuitiva

RNF-06

Nombre	Controles simplificados.
Descripción	El videojuego será jugado con unos controles reducidos para permitir que jugadores con problemas motrices puedan participar sin problemas.
Prioridad	Alta.

Tabla 3.39: RNF-06: Controles simplificados

RNF-07

Nombre	Paleta de colores.
Descripción	La paleta de colores usada en el videojuego debe ser una que permita que cualquier nivel de daltonismo pueda usarla sin confundir colores.
Prioridad	Alta.

Tabla 3.40: RNF-07: Paleta de colores

RNF-08

Nombre	Instrucciones del juego en símbolos.
Descripción	Para evitar bloques de texto extensos y poco atractivos, las instrucciones del juego deberán ser presentados con símbolos e imágenes.
Prioridad	Media.

Tabla 3.41: RNF-08: Instrucciones del juego en símbolos

RNF-09

Nombre	Control de errores.
Descripción	El videojuego deberá controlar los errores de forma que de encontrar uno, el juego se recupere por sí solo y no se interrumpa la partida actual.
Prioridad	Alta.

Tabla 3.42: RNF-09: Control de errores

3.3 Identificación y análisis de soluciones posibles

Antes de empezar siquiera a estudiar y analizar cómo debería ser el prototipo final o reunirse con los profesores y alumnos del Colegio La Purísima Hnas. Franciscanas, se hizo un repaso a diversos juegos para intentar identificar cómo enfocar el diseño del videojuego final. Se evaluaron los costes de tiempo y esfuerzo de diferentes estilos. Es muy común en los videojuegos partir de unas ideas existente y alterarlas para diferenciarlas del resto, lo que se llama definir el *género del videojuego*.

Se planteó en un inicio intentar seguir la estructura del videojuego nombrado en el estado del arte, *Profesor Layton*, que consiste en la resolución repetitiva de rompecabezas, pero enfocándolo a adaptar dichos rompecabezas a diferentes competencias curriculares.

A su vez, se consideró basar la mecánica en el videojuego *Wario Ware*, desarrollado por *Nintendo R&D1*, el cual consiste en que el jugador se enfrente a una cantidad infinita de rompecabezas de diez segundos de duración sin fallar y que cada diez rompecabezas realizados, se redujera en medio segundo el tiempo para realizarlos. Una vez el jugador falle, el juego muestra una puntuación y permite volver a empezar.

Ambas propuestas fueron descartadas porque requerían la generación de una cantidad muy elevada de rompecabezas cuyo esfuerzo de desarrollo era demasiado alto, además que seguían siendo videojuegos de un solo jugador, lo cual no permitía mejorar las competencias sociales. Esto hizo que se cambiara el enfoque y se buscara juegos multijugador, preferiblemente no online, al cual se pudiera modificar con rompecabezas.

Se propuso realizar una adaptación de un juego de *Búsqueda del tesoro* a videojuegos. Se intentó iterar esta idea para ver si podía combinarse con las dos propuestas anteriores. Finalmente, se desecharon completamente las ideas previas y se decidió basar el juego en *Mario Party*, desarrollado por *Hudson Soft* y *Nd Cube*, el cual consiste en que varios jugadores compitan viajando por un tablero para ver quién es el primero en llegar a la meta. Durante el juego, los jugadores deben tomar decisiones y participar en minijuegos y se pensó que era un buen punto de partida para preparar una solución posible, ya que partimos de que varios alumnos van a tener que jugar a la vez, tomando decisiones por su propia cuenta e implementando rompecabezas curriculares mientras todos avanzan para llegar a la meta.

3.4 Solución propuesta

La solución propuesta para este proyecto se ha dividido en tres fases, para distribuir y organizar el trabajo. Dichas fases son:

1. Análisis y estudio de metodologías de educación y cómo adaptarlas a un videojuego.
2. Reuniones y revisiones de la propuesta con los profesores y alumnos del Colegio La Purísima Hnas. Franciscanas.
3. Desarrollo del prototipo.

La primera fase consiste en la generación de un documento de diseño de videojuego[1], también llamado *GDD* o *Game Design Document*, donde se detalla exactamente cómo funcionaría el videojuego. Este documento se trata siempre de una versión temprana, la cual no tiene porqué reflejar el producto final, simplemente se usa como una guía de la

[1] https://en.wikipedia.org/wiki/Game_design_document

cual partir. La idea base consiste en un videojuego cooperativo donde los alumnos tienen un avatar que va moviéndose por un tablero para que trabajando todos juntos, puedan encontrar la meta.

La segunda y tercera fase se van turnando durante el desarrollo del proyecto. En la primera reunión con los profesores del colegio, se presenta el *GDD* y mediante una entrevista se revisa el contenido para que cumpla con los requisitos de enseñanza que los profesores requieran y se les solicita que preparen preguntas y/o información de los contenidos curriculares que desean reforzar. Después de la reunión, se realiza un prototipo básico con las mecánicas mínimas para funcionar, el cual una vez realizado, se vuelve a presentar a los profesores para que le den el visto bueno y que aporten las preguntas y/o información que se les solicitó que prepararan, para así incluirlas en el videojuego y poder prepararlo para la puesta en marcha y probarlo con estudiantes de dicha escuela.

3.4.1. Competencias

Las competencias se trabajan usando dos enfoques, el directo y el indirecto. A la hora de las competencias transversales, la cohesión entre el juego y el curriculum permiten que la competencia matemática y científica se entrenen directamente, ya que este curriculum vendría dado por la persona que asume el rol del profesor.

Figura 3.4: Niveles de resolución de problemas

En la Figura 3.4 se pueden ver los diferentes niveles para la resolución de problemas. El prototipo propuesto alcanza hasta el nivel intermedio, puesto que unos niveles superiores aumentaría demasiado la complejidad del sistema. Aplicando el enfoque indirecto de la resolución de los rompecabezas a partir de distinta información en forma de pistas, los estudiantes entrenan y refuerzan su competencia de resolución de problemas. Cuando hablamos de un aprendizaje colaborativo, hablamos de un aprendizaje social y participativo que, combinado con la estrategia del juego, hace que el estudiante centre su atención y active su capacidad de razonar y recordar para resolver problemas de diferente nivel, favoreciendo el aprendizaje profundo.

El nivel fundamental es muy sencillo, y funciona de forma que las pistas contienen las respuestas directamente.

Como se ha dicho anteriormente, el nivel intermedio, aplicando pistas y razonamiento deductivo, trabaja el proceso cognitivo que se utiliza para llegar a una deducción a partir de lo que consideramos globalmente cierto [8]. El juego no llega al nivel Avanzado, ya que las preguntas no llegan a que los jugadores tengan que usar un razonamiento hipotetico-deductivo. Dicho razonamiento se basa en la comprobación de premisas falseables y que para comprobar su veracidad deben contrastarse.

Esta dinámica de partida y estrategia permite desarrollar el trabajo en equipo y apoyarse entre los estudiantes [3]. Este fue uno de los motivos por el cual se decidió un videojuego colaborativo donde todos los jugadores tengan iguales oportunidades.

La dinámica de partida propuesta permite desarrollar el trabajo en equipo. Supervisado por el profesor y en base a las competencias de respeto e inclusión que rigen cualquier actividad curricular en el contexto escolar donde se aplique, el videojuego se puede considerar un espacio inclusivo.

Por eso, en cuanto a las competencias sociales, se introducen los poderes. Los personajes tienen poderes únicos, lo que va más allá de un simple diseño del videojuego. Usando diseños de personajes, se intenta promover la inclusividad y el respeto mutuo, ya que indiferentemente de la persona que encarne al personaje, todos tienen un motivo y un beneficio que aportar a su equipo. Además, estos poderes equilibrados entre ellos, permiten que cualquier estudiante se encuentra en una igualdad de oportunidades que no dejará a nadie fuera. Este sería el caso de incluir en el equipo un miembro con alguna necesidad específica de refuerzo. Podría tener un poder aventajado haciendo que sea necesario en el equipo y verse así, reforzado positivamente y no aislado.

Respecto al pensamiento crítico, este se encuentra entre las competencias transversales y las sociales, pues los jugadores necesitarán ponerse de acuerdo para contestar a las preguntas. El juego solo acepta una respuesta y los participantes deben convencerse entre ellos. El juego también está pensado para ser usado en grupos de pedagogía terapéutica, sirviendo como refuerzo del aprendizaje.

El motivo de crear un videojuego entretenido y educativo a la vez, es para permitir que los estudiantes alcancen el *flow state*, también llamado *la zona* o *el flujo*. Esto es un estado mental operativo en el cual una persona se queda totalmente inmersa en una actividad. Es uno de los principales retos de los videojuegos. Esta zona permite que los estudiantes desarrollen su *mindfulness*, o atención plena, la cual consiste en estar atento de forma intencional sin juzgar al resto de compañeros y sin preocuparse de los problemas o los fallos que se encuentren, mitigando así el estrés por los estudios y permitiendoles adquirir conocimientos de una forma distinta o incluso divertida.

El marco digital de competencias europeo [9] define una guía de competencias digitales. De todas las competencias enunciadas en el marco, el proyecto permite mejorar y entrenar en dos de ellas. En base a la colaboración y la comunicación, el juego permite que los jugadores colaboren mediante tecnologías digitales para la co-construcción y co-creación de recursos y conocimientos a partir de las pistas que el juego va proporcionando a los jugadores a medida que van avanzando por él. Como se ha explicado en esta sección, el proyecto incide en la resolución de problemas, dicha resolución de problemas se ejercita mediante tecnologías digitales, comprometiendo individual y colectivamente a los estudiantes en un proceso cognitivo para entender y resolver problemas y situaciones en un entorno digital, como es en este caso, el videojuego.

En los tres tipos de competencia se habla de la colaboración, ya que esta es la principal competencia del videojuego. La colaboración engloba y complementa otras competencias

ya comentadas [2], como el trabajo en equipo, el respeto mutuo o la creatividad mediante tecnologías digitales. La colaboración consiste en el trabajo efectivo con otras personas para alcanzar un objetivo común, la cual implica que deben tomar decisiones colectivas basadas en el consenso, negociar desacuerdos o apoyar y valorar los esfuerzos de los compañeros. Por este motivo se ha buscado trabajar en grupos pequeños, ya que a menor tamaño se permite generar un ambiente entre ellos para que se animen entre sí y celebren mutuamente sus éxitos.

3.5 Plan de trabajo

Como ya se ha hablado en la solución propuesta, el proyecto se ha dividido en tres fases, y a cada una de estas fases se le ha establecido una duración aproximada en semanas.

Para la primera fase de análisis y estudio de metodologías se aproximaron seis semanas de trabajo, debido a que es la base donde el videojuego debe funcionar. Para la segunda fase de reuniones y revisiones se planificaron días concretos, los cuales se realizarían en el colegio asociado. Y para terminar, la fase de desarrollo se dividió en cinco semanas de trabajo hasta la primera reunión y posteriormente tres semanas para preparar el prototipo para la prueba de validación con el alumnado.

Tras definir el plan de trabajo, han surgido contratiempos en la planificación inicial debido a que por motivos de la escuela, ciertas reuniones se tuvieron que retrasar. Estas reuniones demoraron el desarrollo del videojuego, retrasando la tercera fase una semana adicional.

Durante las reuniones y revisiones, como se explicará más adelante en la memoria en la sección *Pruebas funcionales*, las valoraciones y críticas constructivas supusieron la revisión de diferentes sistemas del juego, lo que retrasó otra semana de desarrollo adicional.

3.6 Presupuesto

En este apartado se va a detallar el presupuesto del proyecto, en el que se detallan los gastos de personal y software generado durante el desarrollo del proyecto.

Se ha invertido un total aproximado de 360 horas de trabajo realizados en jornadas de cinco horas al día durante las dieciséis semanas que ha durado el proyecto.

El coste del personal se ha detallado en la siguiente tabla:

Puesto	**Horas**	**Coste (Euros) por hora**	**Total (Euros)**
Analista	60	18	1.080
Diseñador de videojuegos	80	18	1.440
Programador	250	18	4.500
Total	390	7.020	

Tabla 3.43: Coste del personal

Se han utilizado licencias de Creative Commons gratuitas para el arte del videojuego. En caso de requerir imágenes y *assets* gráficos para una versión futura, se tendría que tener en cuenta a la hora de calcular el coste del personal. En el caso del analista y el diseñador de videojuegos, al haberse realizado las tareas a la vez por la misma persona, se ha abaratado el coste de horas de dichas tareas.

En cuanto el coste del software y de las licencias usadas durante la elaboración del proyecto, se resumen en la siguiente tabla:

Nombre del producto	**Coste (Euros)**
Microsoft Windows 10 Pro	259
Unity3D	0
Adobe Photoshop	96
Microsoft Visual Studio	0
Total	355

Tabla 3.44: Coste del software

El precio de Unity3D es gratuito siempre y cuando se use en proyectos con beneficios inferiores a 200.000 euros anuales y con el logo de Unity al arranque del juego. Si se desea eliminar el logo del arranque o se supera ese nivel de beneficios, se aplicaría una licencia de Unity PRO, la cual tiene un coste de 500 euros adicional.

Durante el desarrollo del proyecto, se han utilizado licencias gratuitas de estudiantes de los productos anteriormente listados, por lo que han supuesto un coste de 0 euros. La tabla sirve para un fin ilustrativo del precio que supondría realizar el videojuego de no disponer de tales licencias educativas.

Para terminar este apartado, se muestra un resumen de todos los costes:

Descripción	**Coste (Euros)**
Personal	7020
Software	355
Total	7375

Tabla 3.45: Coste del proyecto

CAPÍTULO 4
Diseño de la solución

En esta sección se detalla el diseño de la solución propuesta. En primer lugar se habla del diseño del videojuego prestando atención a las decisiones tomadas a partir del primer análisis sobre los videojuegos formativos y las teorías de diseño para videojuegos de entretenimiento. Luego se detallarán apartados como la arquitectura del sistema y el diseño detallado del programa, finalizando la sección con la explicación de la tecnología utilizada.

4.1 Diseño del videojuego

Una vez definida la solución propuesta, se llevó a cabo el primer documento de diseño del videojuego en el cual se especifica y se detalla todos los aspectos de este. Como se ha explicado en la introducción de la sección, se detallarán las decisiones tomadas sobre el diseño y se explicará el motivo por el cual fueron decididas.

4.1.1. Análisis del videojuego

Antes que nada, se define una descripción general sobre el juego, indicando el género, las plataformas donde va a funcionar y cuál es su audiencia objetivo.

Descripción	El juego consta en la Aventura de 3-6 exploradores que intentarán llevarse el oculto Tesoro del templo perdido. Moviéndose por un tablero y separándose por caminos individuales, tendrán que decidir cómo ayudarse para sortear diversos rompecabezas, decidiendo si hacerlo en grupo o tomar el desafío por separado.
Género	Tablero y rompecabezas.
Plataformas	*Windows* y *Lliurex*.
Audiencia objetivo	El juego está enfocado a alumnos de primaria para que jueguen en una asignatura o en el tiempo libre para impartir unos temarios concretos y complementar a las clases teóricas.

Tabla 4.1: Análisis del videojuego

El juego está planteado para que participen activamente en la partida entre tres jugadores y seis jugadores. El motivo de esto es que al darle total libertad a un equipo para jugar, los estudiantes tendrán que comunicarse entre ellos para planificar qué desean hacer. Está previsto que también promueva el liderazgo dentro del grupo, por ese motivo se ha buscado un tamaño de equipo medio y que así los jugadores se vean comprometidos

a trabajar juntos. A pesar de ser un videojuego multijugador, solo puede ser jugado utilizando una única pantalla, para que sus participantes tengan que estar juntos a la hora de jugar.

Se ha dado libertad de configuración de las preguntas y textos para que no sea específico de una asignatura. Podrán juntar sus propuestas uno o varios profesores y crear una partida que evalúe los conocimientos de los alumnos como ellos crean conveniente. Es interesante la opción de que el juego pueda usarse como parte de un contexto de juego mayor, en el cual las pistas puedan ser usadas para, por ejemplo, buscar objetos por el aula o el colegio.

4.1.2. Jugabilidad

El juego está basado en un sistema de tablero como el usado por *Mario Party* o *Wii Resort*. Los jugadores empiezan desde una casilla inicial, llamada *Inicio* y deberán separarse por diversos caminos para explorar la mazmorra y encontrar el tesoro. El juego permite que todos los jugadores vayan por el mismo camino, castigando a los jugadores que vayan solos con menos ayudas para superar los rompezabezas.

Los jugadores disponen de una caja de dados común, la cual genera tantos dados como jugadores hay en la partida. Los jugadores deberán decidir y repartir los dados entre todos los personajes para asignarles movimientos, sin haber límites en cuántos dados puede asignarse a cada jugador. Cuando un dado es asignado a un jugador, este se moverá tantas casillas como valor tenga el dado, y de pasar por una casilla que tenga una bifurcación, el juego parará hasta que el jugador elija qué dirección quiere tomar.

Cuando finaliza el movimiento de un jugador, se aplicará el efecto especifico de la casilla. Una vez aplicado el efecto, podrá volverse a asignar un dado a un jugador. Si la casilla es de investigación, una ventana emergente aparecerá con una pista que podrán memorizar, dicha pista no tiene porqué ser de algún rompecabezas que se encuentre en la partida, así los jugadores tendrán que tomar apuntes de datos aunque ellos no sepan que les serán útiles en el futuro del juego. Estas pistas solo pueden ser consultadas en el momento en el que se cae en la casilla, de forma que es vital que los jugadores tomen nota o las memoricen para su posible uso posterior.

Las casillas se distribuyen por salas, y cada sala dispone de una casilla de rompecabezas. Si un jugador cae en ella, saltará una pregunta que el profesor haya propuesto en el archivo de configuración. Si la respuesta es correcta, se abrirá una puerta en la sala permitiendo avanzar por la mazmorra, y la casilla rompecabezas se convertirá en una normal. Si la respuesta es incorrecta, no ocurrirá nada y el jugador podrá seguir moviéndose si un dado le es asignado.

Adicionalmente, cada personaje dispone de un poder específico que podrá utilizar antes de asignar un dado. Hay unos usos limitados a cuántos poderes puedes usar. Caer en una casilla de investigación proporcionará un uso adicional a los poderes. Esto dará un incentivo para que los jugadores exploren el mapa y recojan todas las pistas posibles.

4.1.3. Mecánicas

Cada jugador encarnará a un explorador que se adentra en la búsqueda del tesoro y este personaje no podrá repetirse. Cada personaje tiene un nombre por defecto que el alumno podrá cambiar al principio de la partida. Cada explorador tiene un poder especial que le permitirá cambiar el rumbo de la partida para ayudarles en su aventura.

Se ha buscado que cada personaje sea diferente tanto estéticamente como en su función, para darle a los jugadores una sensación de ser únicos y de tener un propósito exclusivo durante la partida. Estos personajes se moverán a través de unas casillas situadas en un tablero con el fin de superarlo. Los personajes han sido diseñados por Kenney[1], un artista que publica sus obras bajo una licencia de Creative Commons Universal, la cual permite usarlas para cualquier proyecto, sea comercial o no.

Las casillas se ordenan en grupos llamados salas, las cuales tienen cómo mínimo una casilla de investigación y siempre una casilla rompecabezas. Las salas son bucles en las que, una vez el jugador entra, no puede salir y solo al completar la casilla rompecabezas se le permitirá avanzar en el tablero.

 **Figura 4.1:** Marta	La habilidad de Marta es hacer que en el siguiente dado ignoren las casillas de peligro sin que cuenten para el movimiento.	 **Figura 4.2:** Carol	Carol tiene la habilidad de mover a un jugador hasta la casilla rompecabezas más cercana y le permite usarla.
 **Figura 4.3:** Tomás	Tomás puede hacer que el valor de todos los dados tenga +1 en el próximo uso.	 **Figura 4.4:** José	Al usar su poder, José descubre una pista al azar que no esté asignada a ninguna casilla de investigación.

Tabla 4.2: Personajes del juego

Se ha ocultado el tipo de las casillas con efecto para que la primera vez que los jugadores se enfrenten a una sala, no tengan claro cuál es su objetivo y tengan que explorar. Siempre se busca que los participantes interactúen entre ellos y hagan debates sobre cómo enfrentarse a las salas y a sus casillas, haciendo un uso lógico de los dados que disponen.

El color de las casillas ha sido seleccionado después del estudio de varias páginas web y artículos que permitieron definir unos colores, apoyados por unos símbolos que permitan a cualquier persona con daltonismo diferenciar el tipo de casilla que tiene el tablero.

[1] https://www.kenney.nl/

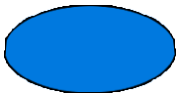 **Figura 4.5:** Casilla normal	Casilla más común en el tablero que no tiene efecto cuando caes en ella.	 **Figura 4.6:** Casilla peligro	Casilla que tiene otra casilla vinculada. Cuando un personaje caiga en ella, lo transportará a la casilla vinculada.
 **Figura 4.7:** Casilla investigación	Las casillas de investigación hacen que se muestre una pista al azar al jugador que ha caído en ella.	 **Figura 4.8:** Casilla rompecabezas	Cuando un jugador caiga en una casilla rompecabezas, el juego le pedirá que responda a una pregunta para poder superar la sala donde esta casilla se encuentre.
 **Figura 4.9:** Casilla secreto	La casilla secreto es el estado original de las casillas rompecabezas, investigación y peligro. La primera vez que un jugador cae en esta casilla se revelará qué casilla hay abajo.		

Tabla 4.3: Casillas del juego

En la Figura 4.10 se puede observar parte del tablero del juego. El tablero es una representación visual de todo lo explicado anteriormente, donde se pueden ver las salas conectadas mediante puertas y pasillos, y los jugadores colocados encima de la casilla donde se encuentran en ese momento. El tablero se ha creado gracias al arte 2D que Dragosha[2] ofrece en su web, el cual tiene una licencia de Creative Commons Internacional, que permite su uso en proyectos comerciales o no-comerciales siempre que se le dé crédito al artista original.

En esta sala en concreto, se pueden observar varias casillas secreto en las cuales aún no ha caído ningún jugador, por lo que su tipo sigue oculto. En la parte más norte del mapa podemos encontrar la casilla de investigación que ya está descubierta, al igual que las casillas peligro alrededor de la hoguera y la casilla rompecabezas junto a la puerta.

[2]http://dragosha.com/

Figura 4.10: Muestra del tablero

Cada sala tiene una estética diferente, el visto en la Figura anterior tiene como nombre identificador *Exterior*. A pesar de que las pistas y rompecabezas se asignan a casillas al azar, es posible asignar contenido concreto para cada sala, usando su nombre identificador en el archivo de configuración en lugar de la etiqueta *p_Número*. Por ejemplo:

```
{
  "preguntas": {
    "Exterior": {
      "pista": {
        "pis_1": "La respuesta correcta es Correcto.",
        "pis_2": "La respuesta incorrecta es Falso."
      },
      "enunciado": "Este es el enunciado de una pregunta?",
      "opcion_1": "Correcto",
      "opcion_2": "Falso",
      "opcion_3": "Falso",
      "opcion_4": "Falso"
    }
  }
}
```

4.2 Arquitectura del sistema

En esta sección se resume, en un formato general, la arquitectura del sistema que se le ha dado a la solución propuesta. El prototipo es una primera iteración del juego, por lo que está sujeto a cambios y debe poder ser escalado en cualquier momento. Ese fue el motivo para representar tanto a las casillas como a los jugadores mediante interfaces. Estas interfaces disponen de todos los métodos que dichos jugadores y casillas deben tener, de forma que a la hora de implementar un componente nuevo de cualquiera de las dos interfaces, se puede hacer de forma rápida y simple.

El sistema parte de una clase principal llamado *GameManager*, dicha clase es llamada así ya que es un estándar en el desarrollo de videojuegos. *GameManager* es el encargado de verificar la configuración y de leer los nombres del menú principal y de esta forma, al pulsar *Iniciar Partida*, crear un *LevelManager* con todos los datos necesarios para que el juego funcione.

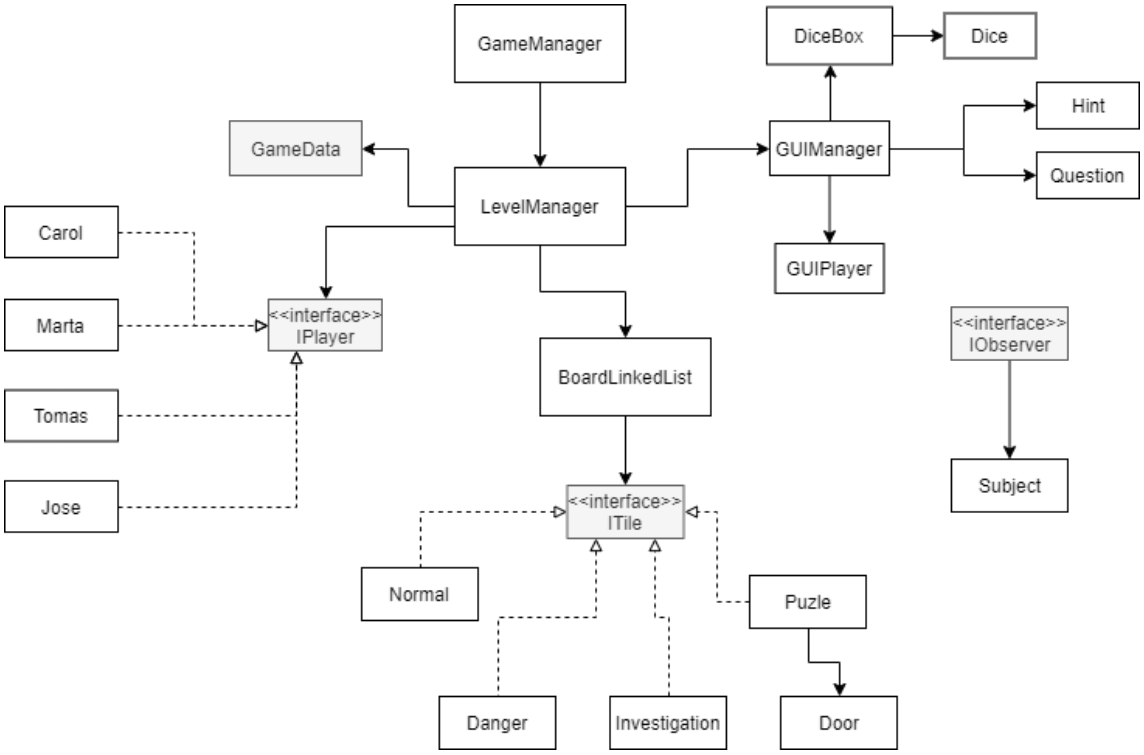

Figura 4.11: Diagrama de clases

LevelManager funciona como un *Singleton* o instancia única, el cual garantiza una única instancia para dicha clase, además de estar disponible a nivel global. Este *LevelManager* será el que haga de unión a todos los sistemas que componen la partida, desde el *GameData* hasta al *GUIManager*, como se puede ver en la Figura 4.11. Esta decisión se ha tomado para aislar los componentes y evitar que haya dependencias entre ellos, de forma que el juego pueda ser probado aunque el resto de sistemas no estén funcionando. Además, este aislamiento permite que, a la hora de controlar errores, solo el *LevelManager* tiene la necesidad de controlarlos y darles solución sin necesidad de que intervengan otras clases.

Para el tablero, se ha creado *BoardLinkedList*, un rediseño de la lista enlazada que utiliza como valores *ITile*, una interfaz de casillas. *BoardLinkedList* implementa todos los métodos necesarios para gestionar la lista de *ITile* y poder añadir y eliminar nodos. El rediseño de la lista enlazada no se limita al valor enlazado, sino que está modificado para que la lista sea unilateral y que cada nodo pueda ser enlazado con uno o más nodos. Esta última modificación es necesaria para crear la bifurcación de caminos y poder diseñar un tablero con varias rutas y posibilidades.

GUIManager es otro *Singleton* cuya función es gestionar todos los elementos gráficos del juego, como la cámara, los personajes, la caja de dados y las ventanas emergentes de pistas y rompecabezas. La existencia de este *manager* es para juntar todas las llamadas a la *GUI*, la *Graphic User Interface* o interfaz gráfica del usuario, ya que está compuesta por varias clases y, al igual que en el *LevelManager*, se prefiere que la comunicación entre los objetos pase por un *manager* a que se comuniquen entre ellos.

Igual que con las casillas, existe una interfaz *IPlayer* con los métodos y atributos base de los personajes. Esta interfaz es implementada por los diferentes tipos de personaje, ya que cada uno define el método *Power()* con la funcionalidad correspondiente a los casos de uso. La interfaz *ITile* funciona del mismo modo, en su caso, el método a implementar es el *Effect()*. La única casilla con un comportamiento especial es *Puzle*, la cual tiene una referencia a la clase *Door*. La clase *Door* va asociada a un elemento visual del juego. Cuan-

do se completa una casilla de rompecabezas, esta avisa a su referencia de que tiene que desbloquearse.

Todas las clases u objetos que se usan en el motor *Unity* heredan de *MonoBehaviour*, una clase abstracta que define todos los métodos relacionados con la ejecución, desde controlar los gráficos a modificar los tiempos de refresco o crear hilos. Sin embargo, *Unity* permite la creación de clases o objetos que no hereden de dicha clase, en nuestro caso son las clases cuyo fondo esta en gris en la Figura 4.11. Estas clases son las interfaces *IPlayer*, *IObserver* y *ITile* y por otro lado, *GameData*.

GameData es la clase que guarda toda la información del sistema, en vez de estar almacenada en *LevelManager*. Esto se debe a que las clases que heredan *MonoBehaviour* no se puede serializar. Al aislar *GameData* de *MonoBehaviour* se consigue que el contenido de la clase pueda extraerse o integrarse a través de archivos externos a la aplicación, por si fuera necesario sacar información de la partida para la generación de estadísticas.

En el diagrama se han omitido clases o archivos que más adelante se detallan y que son meramente enumeraciones o métodos de utilidad para el motor gráfico.

Por último, el sistema tiene implementado un patrón de diseño *Observer*, cuya función es definir una relación uno a muchos entre objetos, de forma que los objetos o clases que sean *Subject* podrán usar *Notify()* para avisar del cambio de estado y enviar un evento a los objetos o clases *IObserver*.El motor *Unity* ya tiene integrado este patrón, el cual es llamado por el editor como *Listener*, pero es muy complejo y pesado y esto ralentizaba y aumentaba el coste de memoria del juego, así que se decidió implementar el patrón de cero.

4.3 Diseño detallado

En esta sección se entrará en detalle en cada una de los patrones y estructuras de datos vista en la sección anterior, especificando los atributos y métodos que dispone cada clase y el porqué de estos. La sección se divide en apartados para cada patrón o estructura para organizar mejor la información.

En cuanto al directorio, *Unity* genera una estructura de carpetas, dentro de la cual, nos vamos a centrar en el directorio *Scripts*, que es donde se almacenan todos los archivos relacionados con la programación del juego.

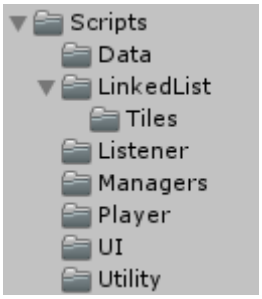

Figura 4.12: Directorio Scripts

Los contenidos de cada carpeta son los siguientes:

- *Data*: Contiene *GameData*, y en caso de ser necesario, todo lo que necesitemos para gestionar los datos del juego.
- *LinkedList*: Contiene *BoardLinkedList* y la carpeta Tiles, donde se encuentran *ITile*, *Normal*, *Danger*, *Investigation*, *Puzle* y *Door*.

- *Listener*: Contiene *IObserver*, *Subject* y *ClassEvent*. Esta última clase es una enumeración de los tipos de eventos que *Subject* puede notificar.
- *Managers*: Aquí se encuentran todos los *Managers* de la aplicación: *GameManager*, *LevelManager* y *GUIManager*.
- *Player*: Contiene los archivos correspondientes a los personajes: *IPlayer*, *Marta*, *Carol*, *Tomas* y *Jose*.
- *UI*: Aquí están los métodos encargados de gestionar la *GUI*, como *GUIPlayer*, *GUIPortrait*, *DiceBox*, *Dice*, *Hint*, *Question* y *Camera*.
- *Utility*: Aquí se encuentran las clases que son requeridas para que el sistema funcione correctamente, como *AspectUtility* que modifica la resolución del juego para que las texturas no se deformen, o *DepthSortByY*, el cual determina la profundidad de los objetos visuales en base a su posición en Y.

4.3.1. GameManager, LevelManager y GameData

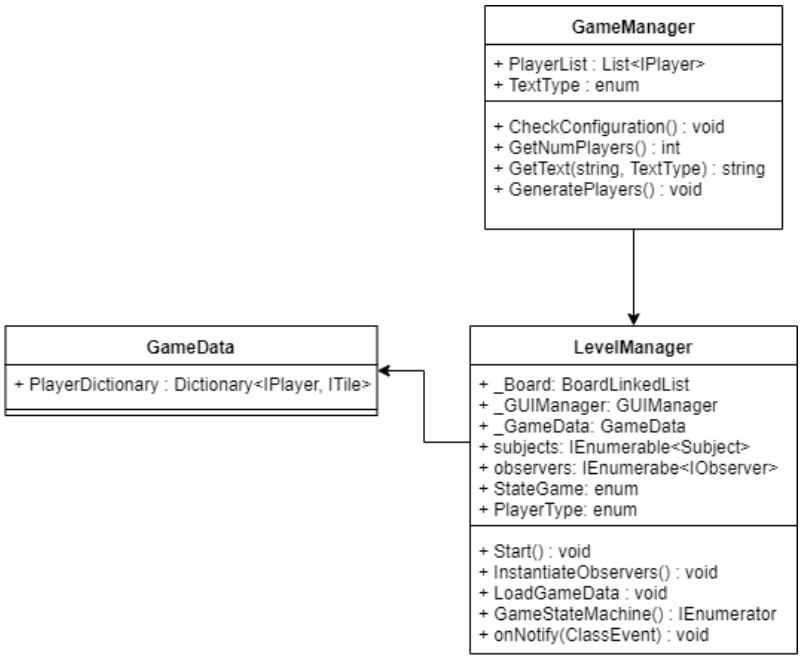

Figura 4.13: GameManager, LevelManager y GameData

En este apartado se detalla la funcionalidad de las clases principales del programa, *GameManager*, *LevelManager* y *GameData*. Se les llama clases principales porque son las que sirven de unión para el resto de clases y las que mantienen un estado para dirigir el juego de principio a final.

GameManager, como se ha explicado anteriormente, es la clase que inicia el juego, podría considerarse la clase *main*. Nada más iniciar el juego, *GameManager* revisa el archivo de configuración mediante *CheckConfiguration()*. Si la lectura del archivo falla, el juego muestra un error de lectura por pantalla y da las opciones de generar un archivo de configuración nuevo en base a una plantilla y continuar con los valores por defecto o de cerrar el juego, indicando la ruta de la carpeta donde esté el archivo, ver ANEXO B. Una vez la lectura ha sido satisfactoria, el juego genera los jugadores que son almacenados en una lista *PlayerList*. Tanto esta lista, como *GetNumPlayer()* y *GetText(string, TextType)* son públicos, de forma que *LevelManager* puede acceder a ellos cuando quiera. Una vez

la revisión de la configuración y la generación de personajes ha sido realizada con éxito, *GameManager* carga la escena de *Unity*. Cabe destacar que para que el *GameManager* no sea destruido entre el cambio de escenas, lleva la directiva *DontDestroyOnLoad*.

El *GameManager* se crea durante el menú principal y mediante una llamada por la *GUI* de *Unity* mediante el botón *Iniciar Partida*, es cuando ocurre dicha carga de escena. Aquí también es donde se obtienen los nombres de los jugadores. Las clases *Button* y *FilleableTextMesh* son clases propias de *Unity*, las cuales no están definidas en los diagramas porque no son propias del proyecto.

En cuanto al *LevelManager*, una vez la escena se ha cargado, ejecuta el método *Start()*, que es un método de *Unity* predefinido que siempre se lanza cuando se crea un objeto. Dentro de este *Start()* se crean instancias a todos los *manager*, se analiza la escena y asigna los *IObservers* a todos los *Subject* mediante *InstantiateObservers()* y se procede a la creación del *GameData* mediante *LoadGameData()*, el cual extrae sus datos del *GameManager* para ser creado.

Durante la creación del *GameData*, *LevelManager* también llama al método *InstantiatePlayers(List<IPlayer>)* de *GUIManager*, pero este se detallará más adelante. Una vez el método *Start()* finaliza, la clase crea una corrutina, *coroutine*, hilo o *thread* en otros lenguajes, llamada *GameStateMachine()* la cual genera una máquina de estados que lo gestiona hasta que el juego finaliza, y va dando prioridad a animaciones y acciones para dirigir la partida.

LevelManager es *Subject* y *IObserver*, por lo que desde la corrutina se actualiza el estado del resto de objetos mediante *Notify(ClassEvent)*. Para recibir la información del resto de clases, utiliza el *onNotify(ClassEvent)*, el cual actualiza los atributos internos de la máquina de estado para identificar en qué punto de la partida se encuentra el juego.

En el momento en el que se carga la información para crear *GameData*, este crea un diccionario con *IPlayer* y *ITile*, donde cada jugador es asignado a la casilla *Head* del *BoardLinkedList*. De esta forma, todos los jugadores empezarán en la casilla inicial cuando empiece la partida. La única función de esta clase es mantener el diccionario con las posiciones de cada jugador para que *LevelManager* sepa dónde están en todo momento.

4.3.2. IPlayer y personajes

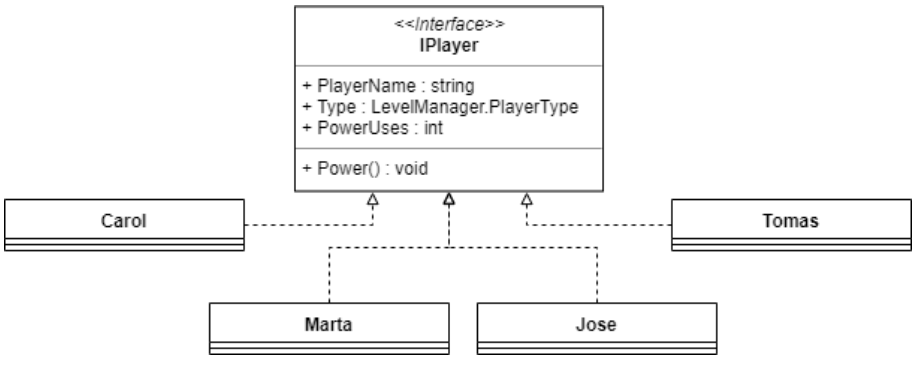

Figura 4.14: IPlayer y personajes

La interfaz *IPlayer* se ha creado para facilitar la implementación de personajes adicionales al videojuego. Dispone de un *PlayerName* el cual es el nombre dado al jugador, un *Type* qué indica que tipo de personaje es y un *PowerUses* que avisa al *LevelManager* de los usos restantes de poderes que le quedan al personaje. Estos atributos, excepto *PowerUses* que tiene un valor por defecto de 3, son dados en la creación del objeto durante *Genera-*

tePlayers() en el *GameManager* y dicha lista es transmitida a *LevelManager* para que este la gestione.

Cada uno de los personajes *Carol*, *Jose*, *Marta* y *Tomas*, implementan y definen *Power()* de forma diferente, satisfaciendo la especificación dada de cada uno de los poderes. Cuando un poder es activado, se notifica a *LevelManager* para poder cambiar el estado del juego y que este tenga efecto en la partida. Un ejemplo de la implementación de *Power()* en el caso de *Marta* es el siguiente:

```
1   public void Power()
2   {
3       if(PowerUses > 0)
4       {
5           Notify(new ClassEvent(this, ClassEvent.EventType.IgnoreDanger));
6           PowerUses--;
7       }
8       else
9       {
10          Notify(new ClassEvent(this, ClassEvent.EventType.NoUses));
11      }
12  }
```

Se puede ver cómo *Marta* notifica siempre, hayan usos o no, y en caso de no haber usos restantes, el *LevelManager* debe saberlo para notificar al *GUIManager* de esto y que emita un sonido de error.

4.3.3. BoardLinkedList y casillas

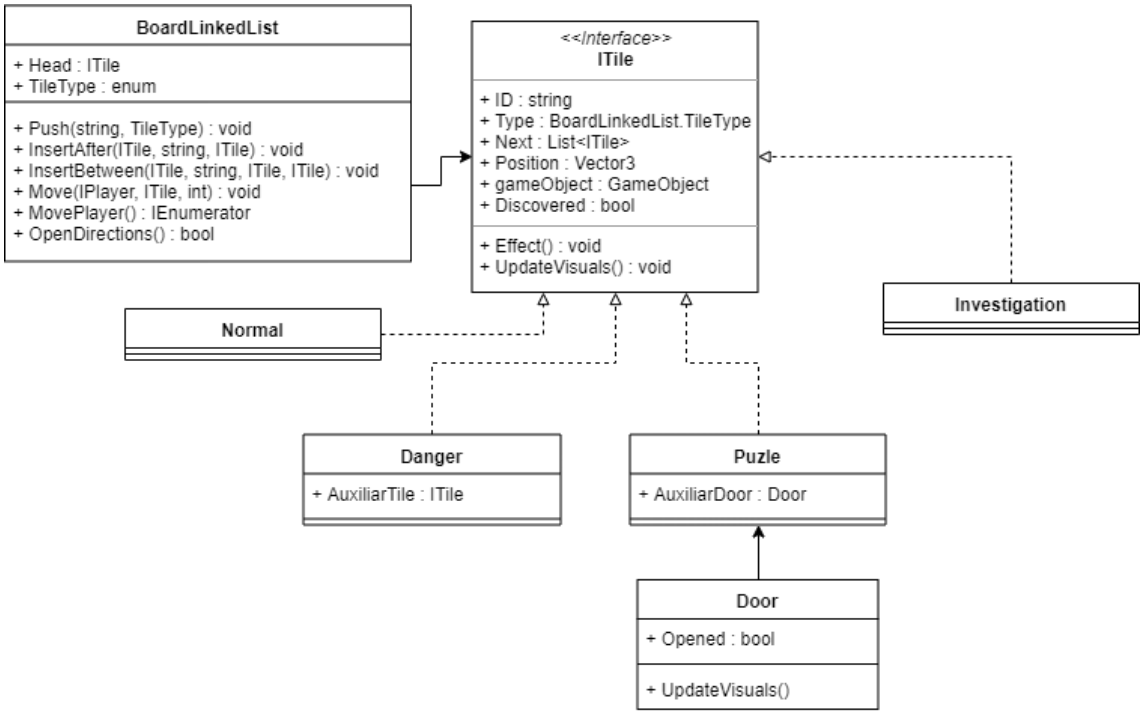

Figura 4.15: BoardLinkedList y casillas

BoardLinkedList es la clase encargada de definir el tablero del juego. Dispone del atributo *Head*, del tipo es una ITile, la cual marca la casilla de inicio. A parte de este atributo, se definen los tipos de casilla que hay en el juego y los métodos para insertar casillas al tablero. No se requiere un método de *Delete* ya que la creación del tablero se hace desde

el motor en modo visual y a la hora de borrar un nodo, esto se puede realizar desde la ventana del editor.

```
IEnumerator MovePlayer(IPlayer PL, ITileBoard Origin, int movements)
    {
        while (movements != 0)
        {
            if (Origin.Next == null)
            {
                Debug.LogError("There is not a next Tile to move");
                yield break;
            }
            else if (Origin.Next.Count == 1)
            {
                ChangePlayerPosition(PL, Origin.Next[0]);
                movements--;
                Origin = Origin.Next[0];
            }
            else
            {
                yield return new WaitUntil(() => OpenDirectionOptions());
                ChangePlayerPosition(PL, destiny);
                movements--;
                Origin = destiny;
            }
            yield return new WaitForSeconds(0.25f);
        }
        yield break;
    }
```

El método *Move(IPlayer, ITile, int)* sirve para que, dado un personaje y su casilla, extraídos y notificados a *BoardLinkedList* desde el *LevelManager*, calcule la casilla donde el jugador estará después de mover tantas casillas como el valor del parámetro *int*. Este método no devuelve nada, en su lugar, crea una corrutina *MovePlayer(IPlayer, ITile, int)* que se dedicará a mover al jugador por las casillas, notificando mediante *Notify(ClassEvent)* a *LevelManager* de cada una de las casillas por las que pasa, para que este a su vez notifique a *GUIManager* para lanzar la animación del jugador para ir avanzando por el tablero. Esta corrutina también permite que el código se pare mediante la directriz `yield return new WaitUnitl(function)` y así poder gestionar las bifurcaciones.

Cuando *MovePlayer()* llega a un punto de *WaitUntil(function)* se lanza *OpenDirections()*, la cual genera en el tablero flechas para indicar los posibles caminos que el jugador puede tomar. Una vez se ha tomado una dirección, *OpenDirections()* devuelve un *true* y permite que la corrutina se siga ejecutando.

Las casillas implementan todas *ITile* y contienen todos los atributos y métodos de la interfaz. Algunas como *Danger* o *Puzle* necesitan un atributo auxiliar, ya que su método *Effect()* requiere de un dato adicional para funcionar. En el caso de *Puzle* este tiene una referencia a un objeto de tipo *Door* para identificar cuál es la puerta asignada a la resolución del rompecabezas. El método *UpdateVisuals()* sirve para actualizar los gráficos de la casilla una vez ha sido descubierta y el valor de *Discovery* pasa a ser true. Debido a que las interfaces no pueden heredar de *MonoBehaviour*, no disponen de los atributos *gameObject* y *Position* así que se han definido en la interfaz cómo propiedades para que las casillas puedan implementar el *get* para devolver el valor correspondiente.

```
public Vector3 Position { get => gameObject.transform.position; }
public GameObject gameObject { get => gameObject; }
```

4.3.4. GUIManager y visuales

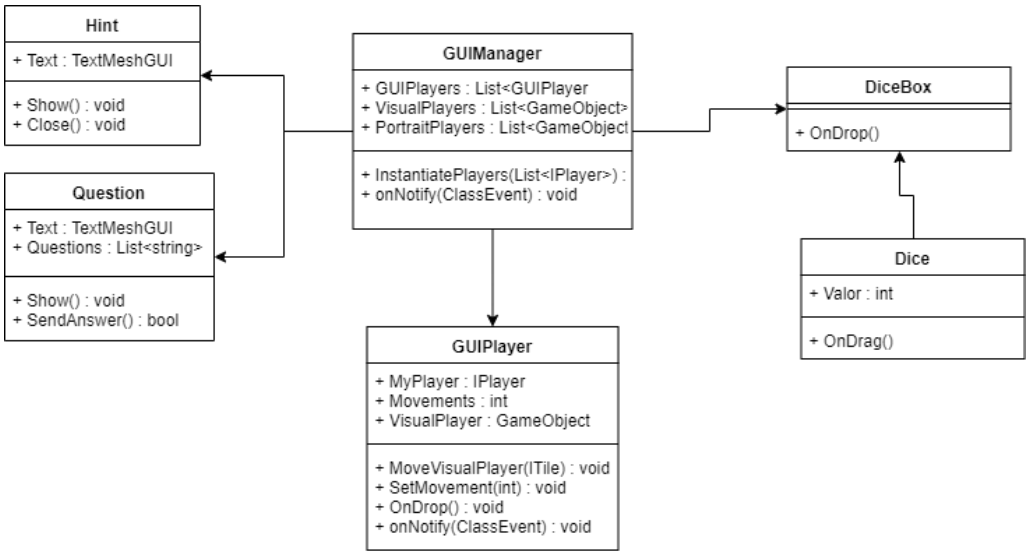

Figura 4.16: GUIManager y visuales

GUIManager es el *manager* encargado de gestionar la parte de la interfaz gráfica y los personajes. Este *manager* dispone de listas de todos los objetos que debe controlar. Es el encargado de crear las instancias de los personajes al inicio de la partida, mediante *InstantiatePlayers(List<IPlayer>)* el cual, mediante el método *Instantiate(gameObject, position)* permite a *Unity* crear objetos visuales prefabricados y colocarlos en un punto del juego. De forma detallada, este método para instanciar a los jugadores crea un *GUIPlayer* por cada uno de los *IPlayer* de la lista dada. Estos *GUIPlayer* tienen asociado un *VisualPlayer* y un *PortraitPlayer*, los cuales son unos objetos visuales de *Unity* también llamados *GameObject*. Cada vez que un jugador es instanciado, *GUIManager* guarda la referencia de cada objeto creado en sus listas.

Al igual que *LevelManager*, dispone del método *onNotify(ClassEvent)* para actualizar todos los elementos visuales cuando reciba una notificación determinada.

La clase *GUIPlayer* es la encargada de mover a los jugadores por el tablero cuando *GUIManager* reciba la notificación de hacerlo. Tiene una referencia a su *IPlayer* asociado, llamado *MyPlayer*, lo cual le servirá cuando reciba a través del *onNotify(ClassEvent)* para saber si se está llamado a su personaje y no a otro. *MoveVisualPlayer(ITile)* es llamado cuando se reciba una notificación de mover, y cargará la animación para desplazar al personaje visual o *VisualPlayer* hasta la posición de la *ITile* del parámetro.

Para el sistema de dados, tanto *DiceBox*, como *GUIPlayer*, disponen del método *OnDrop()*, el cual es un rediseño del que viene por defecto en *Unity* para detectar cuándo un objeto *Draggeable*, o arrastrable, es soltado encima de él. Por otro lado, *Dice* es un objeto arrastrable, el cual implementa el método *OnDrag()* que define dicho comportamiento. *Dice* por su parte, dispone de un atributo *int* llamado *Valor*, el cual contiene el valor del dado. La *DiceBox* siempre es rellenada por el *LevelManager* cuando se queda vacía.

Por último, las clases *Hint* y *Question* están asociadas a ventanas emergentes, las cuales utilizan *TextMeshGUI*, que es un objeto visual de *Unity* para mostrar texto. En el caso de *Hint*, contiene los métodos *Show()* y *Close()* que permiten abrir y cerrar dicha ventana respectivamente, y *Question* dispone del mismo método *Show()*, pero usa *SendAnswer()* para cerrar la ventana y además devolver un *bool* para saber si el rompecabezas se ha respondido satisfactoriamente. Para hacer la carga de preguntas, dispone de una lista de *string* llamada *Question*, la cual es suministrada durante la creación de la pregunta.

4.3.5. IObserver, Subject y ClassEvent

Esta implementación de sistema *Listener* de *Unity* es lo que permite que todo el sistema pueda comunicarse entre él. Está compuesto por tres clases.

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

public interface IObserver
{
    void onNotify(ClassEvent _event);
}
```

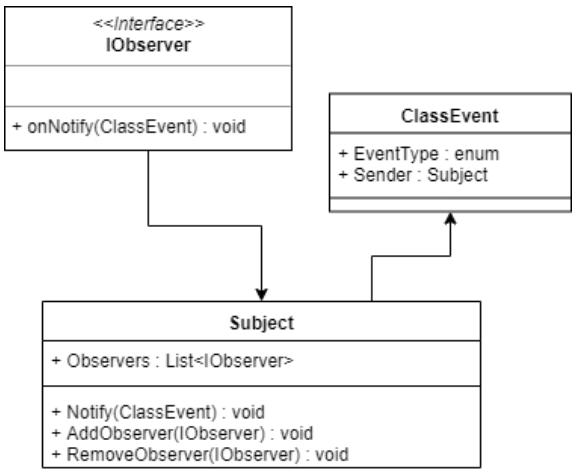

Figura 4.17: IObserver, Subject y ClassEvent

Como puede verse en el código anterior, la interfaz *IObserver* es muy simple. Esta interfaz es implementada por gran parte de las clases del sistema y permite que puedan comunicarse entre ellas. Esto es posible gracias a la siguiente definición contenida en *Subject*:

```
protected List<IObserver> Observers = new List<IObserver>();

protected void Notify(ClassEvent _event)
{
    foreach (IObserver o in Observers)
        o.onNotify(_event);
}
```

Cuando un sistema usa un *Notify(ClassEvent)* lanza el método *onNotify(ClassEvent)* de todos los *IObserver* de la escena. De esta forma, cada clase podrá reaccionar y actualizar su estado en base al método que le llegue. Esto obligó a usar una clase de creación de eventos llamada *ClassEvent*, ya que *Unity* tiene reservada la palabra *Event* para sus propios eventos internos. En la clase *ClassEvent* se define un constructor de eventos y una enumeración de tipos para poder indicar quién crea el evento y de qué tipo se trata. La clase *ClassEvent* es la siguiente:

4.3.6. Clases de utilidad

Existen otras clases en el juego que son simplemente un arreglo de utilidad para el motor. Como *AspectUtility*,que modifica el tamaño del *render* de *Unity* para que las imágenes no se distorsionen al cambiar de resolución, o *DepthSortByY* el cual permite construir

escenarios isométricos para que los *sprites*[3] no se superpongan entre ellos. Estas clases de utilidad están enfocadas a solucionar posibles problemas que puedan ocurrir con los sistemas de renderizado o visuales del motor y tienden a modificar las características de todos los objetos en pantalla.

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

[ExecuteInEditMode]
[RequireComponent(typeof(Renderer))]
public class DepthSortByY : MonoBehaviour
{

    private const int IsometricRangePerYUnit = 100;

    void Update()
    {
        Renderer renderer = GetComponent<Renderer>();
        renderer.sortingOrder = -(int)(transform.position.y *
            IsometricRangePerYUnit);
    }
}
```

En este caso, DepthSortByY requiere componentes específicos e incluso se le asigna una directriz para que se ejecute desde el editor en vez de en modo *play* como es normal. El método *Update()* heredado de *MonoBehaviour* es un bucle que se ejecuta a cada ciclo de procesador, por lo que esta clase siempre se estará ejecutando, garantizando que la profundidad y las capas de objetos sean correctas en cualquier momento.

4.4 Tecnología utilizada

Se ha decidido utilizar el motor *Unity3D*, ya que es un motor muy versátil y que da mucha libertad al desarrollador a la hora de crear contenido. Creado por la compañía *Unity Technologies* en 2005, se ha convertido en uno de los motores más utilizados de los últimos tiempos. Actualmente va por la versión 2019.1 y se hacen revisiones en cada trimestre, además de tener una comunidad grande y un servicio técnico muy eficiente y rápido.

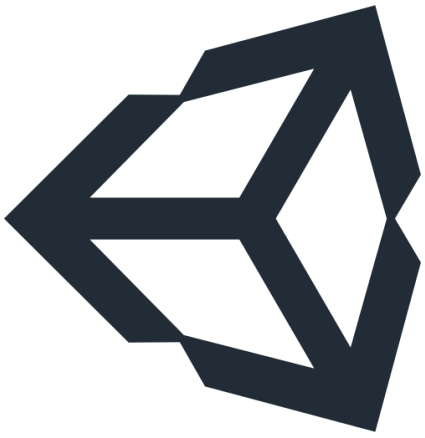

Figura 4.18: Logo de Unity3D

Una propiedad de *Unity* por la cual muchos desarrolladores noveles empiezan a usarlo es la simpleza de su interfaz de trabajo y las pocas herramientas de terceros que incluye. Dispone de componentes y librerías adicionales que se le pueden añadir, pero estas deben ser incluidas de forma manual, para así evitar que el sistema disponga de funciones que no se usen y eso reduzca su rendimiento.

Es un motor ligero que puede funcionar en casi cualquier ordenador de gama baja-media y tiene versiones para *Windows*, *GNU/Linux* y *Mac*. Además de compatibilidad para trabajar en distintos sistemas operativos, dispone de una herramienta de compilación que le permite ajustar el sistema de *Inputs*, o de dispositivos de entrada, para que

[3] https://es.wikipedia.org/wiki/Sprite_(videojuegos)

estos se adapten al sistema objetivo. Así pues, en la versión *Unity3D 2019*, la usada para el desarrollo de este proyecto, es capaz de compilar programas para las plataformas *Windows*, *GNU/Linux*, *OS X*, *SteamOS*, *Android*, *iOS*, *Windows Phone*, *Samsung Smart TV*, *WebGL*, todas las consolas actuales del mercado, desde *Nintendo 3DS* hasta *Playstation 4* y a dispositivos de realidad extendida.

El motor es compatible junto a los programas de creación de arte más usados del mercado, como *Adobe Photoshop* o *Blender*. En cuanto al sistema de *scripts*, está basado en *Mono*, la implementación de código abierto de *.NET Framework*. Este sistema puede ser escrito en tres lenguajes: *UnityScript*, una personalización de la sintaxis de *JavaScript*, *C#* o *Boo*, que dispone de una sintaxis inspirada en *Python*.

En los últimos años, *Unity* ha intentado promover la creación de videojuegos en la educación, creando licencias de carácter educativo y buscando voluntarios que quieran participar para animar a la gente al desarrollo de videojuegos mediante la iniciativa *Unity Ambassador Student*. Adicional a la licencia educativa, *Unity* dispone de otras dos versiones, *Unity Personal* y *Unity Professional*, también llamada *Unity PRO*.

A excepción de *Unity PRO*, el resto de licencias son de uso gratuito y puedes publicar los productos realizados sin ningún tipo de coste. Esto se debe a que hay una cantidad muy grande de estudios sin o con poco presupuesto que usan este motor, y cobrar una membresía podría suponer que algunos equipos no pudieran afrontar el lanzamiento, por lo que deciden no tener cuota para así promover la creación de videojuegos. Solo aquellos juegos que superen cierto umbral de beneficios están obligados a pagar unos *royalties* por el uso del motor.

Debido a las posibilidades que el motor ofrece y la facilidad que es añadirle componentes para adaptar el editor a las necesidades, se ha elegido *Unity 2019* como el motor para desarrollar el prototipo de este videojuego.

Figura 4.19: Logo de Microsoft VisualStudio 2019

En cuanto al *IDE*, *Integrated Development Environment* o Entorno de desarrollo integrado, *Unity* únicamente es compatible con dos, *MonoDevelop* y *Microsoft VisualStudio*.

VisualStudio es un *IDE* desarrollado por *Microsoft* compatible con múltiples lenguajes de programación como *Java*, *Python*, *C#*, *C++*, así como para desarrollo web. *Unity* dispone de un paquete de componentes que hacen que *VisualStudio* sea compatible con el motor y poder enlazar herramientas como del *Debugger* para la realización de pruebas.

Debido a que durante el grado varias asignaturas se han impartido utilizando este *IDE*, se ha preferido su uso frente a *MonoDevelop*. Además, desde la versión *Unity3D 2017* se utiliza *VisualStudio* debido a que, aunque es más pesado que *MonoDevelop*, proporciona muchas herramientas útiles para el motor y todos los tutoriales y guías se adaptaron para el uso de este IDE.

CAPÍTULO 5
Desarrollo de la solución propuesta

Esta sección detalla el desarrollo del videojuego desde el inicio de este proyecto y cómo se ha llegado a implementar de esta forma, incluyendo los inconvenientes o problemas que acarrearon algún cambio en el prototipo.

Cuando se inició el desarrollo de el prototipo, un punto importante fue decidir el motor gráfico, se contemplaron varias opciones como *Unreal Engine* o *GameMaker*, pero estas estaban limitadas por los conocimientos necesarios para poder usarlas correctamente o por las plataformas a las que podían compilar sin requerir hacer cambios drásticos en la configuración, por lo que *Unity* satisfacía las necesidades que teníamos para el proyecto.

Se empezaron desarrollando distintos tipos de minijuegos para las casillas de rompecabezas, pero estos minijuegos fueron desplazados a una futura versión del juego, ya que iba a ser difícil adaptarlo a distintas competencias curriculares y serían necesarias diferentes reuniones para poder ser diseñados correctamente. Es por ello que para una primera iteración del sistema de preguntas y respuestas, este generaba muchas oportunidades a los profesores para inventar formas de repasar conceptos específicos o incluso de usar este videojuego como un suplemento para un juego mayor, como una yincana.

Durante una reunión para probar un prototipo no funcional y ver si los ordenadores del Colegio La Purísima Hnas. Franciscanas podían soportar los requisitos del juego, no se tuvo en cuenta que la mayoría de colegios usan *Lliurex* como sistema operativo, con lo cual, el *GameManager* encontraba errores al intentar acceder y crear el archivo de configuración, ya que estaba pensado para que accediera a un sistema de archivos *Windows*.

```
#if UNITY_STANDALONE_WIN
    path = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.
        MyDocuments),"TFG");
#endif
#if UNITY_STANDALONE_LINUX
    path = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.
        MyDocuments),"/home/TFG");
#endif
file = File.ReadAllText(Path.Combine(path, "Configuration.json"));
ConfigFile = JObject.Parse(file);
```

Esto hizo que se tuviera que implementar un sistema de directivas para activar el modo de compilación dependiente de plataformas, como puede verse en el bloque de código anterior. Esto permitió que a la hora de compilar el juego para una determinada plataforma, este solo compilara la parte del código de acuerdo al sistema al que se estaba compilando. De esta forma, la parte que *Unity* no podía resolver con su sistema de compilado fue resuelta. Este sistema de directivas es muy útil a la hora de hacer pruebas, ya que mediante la directiva `UNITY_EDITOR` se pudo definir un método que generaba flechas de dirección entre las casillas.

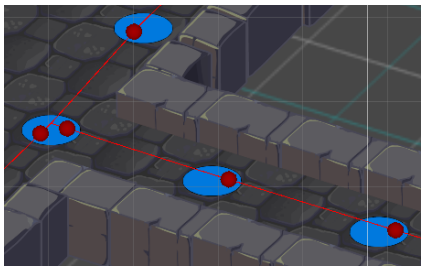

Figura 5.1: Dirección de las casillas desde el editor

Este método llamado *OnDrawGizmos()* es un método interno de *Unity* que permite dibujar figuras geométricas a partir de vectores de dirección y tamaños. Esta forma de representar las direcciones es muy útil a la hora de ir diseñando el tablero, ya que puedes ver que todas vayan en la dirección correspondiente, pero a la hora de compilar el juego y probarlo, su calidad puede ser bastante baja. Gracias a la directiva anterior, se puede probar y modificar este *OnDrawGizmos()* para hacer pruebas entre las casillas sin preocuparse de borrarlas antes de compilar el juego para cualquier plataforma.

En el momento de implementar los personajes, se llegaron a diseñar seis en total, dos más adicionales a Carol, Marta, José y Tomás. Estos eran un Robot y un Zombie, pero por tiempos y complejidad de sus poderes fueron aplazados, igual que los minijuegos, a iteraciones posteriores. Estos recortes de diseño para aplazar desarrollos a iteraciones posteriores se le conoce comúnmente como *Downgrade*.

Durante el desarrollo de todas las clases e interfaces, se acabó optando por el patrón Observador, ya que el diagrama tenía tantas clases enlazadas que en caso de ocurrir un error, todo el sistema se quedaba bloqueado. De esta forma, se realizó una refactorización para empezar a aislar sistemas del juego y centralizar toda la funcionalidad en *LevelManager*, de forma que todo se comunicara mediante *Subject* y *IObserver*, simplificando así las llamadas entre clases y las dependencias entre ellas.

Finalmente, se decidió extraer todos los datos de *LevelManager* y juntarlos en *GameData* mientras se hacía *serializable* para poder exportar e importar datos del juego, por si esto fuera necesario en un futuro. Una clase u objeto *serializable* es aquel que puede ser convertido a texto y viceversa, para exportar o importar información. Todas las interfaces, el patrón Observador y lo acabado de explicar del *GameData* se ha realizado para garantizar que el proyecto sea escalable y que pueda incluir mucha más funcionalidad de cara al futuro.

CAPÍTULO 6
Implantación

En este capítulo se presenta la etapa en la que el desarrollo del prototipo se ha realizado y se ha llevado a la explotación.

El videojuego se ha compilado para los sistemas operativos *Windows* y *Lliurex*, cumpliéndose los requisitos no funcionales *RNF-01* y *RNF-02* adaptándose a cualquier resolución gracias a la clase de utilidad *AspectRatio.cs*. Durante el desarrollo se instaló una máquina virtual con *Lliurex* con recursos similares para hacer pruebas de rendimiento, las cuales fueron exitosas.

El videojuego, tal y como lo compila *Unity* no requiere de instalación. Se genera el directorio de la Figura 6.1, el cual para iniciarlo solo se requiere de ejecutar *TFG.exe*. Para el caso de *Lliurex* se crea un lanzador que ejecuta el fichero equivalente.

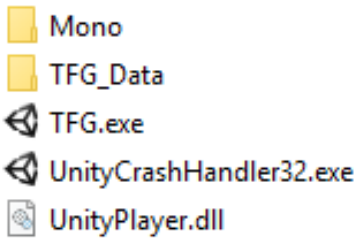

Figura 6.1: Directorio del juego

La primera vez que el juego se ejecuta, este creará el directorio necesario dentro de *Mis Documentos* en *Windows* o en */home/* en el caso de *Lliurex*, donde a la vez, el archivo *Configuration.json* será generado con los valores por defecto. Una vez exista archivo junto a su directorio, los profesores podrán modificarlo para añadir el contenido que ellos deseen impartir.

Para la primera partida jugada por alumnos, los profesores del colegio mandaron un documento con las preguntas propuestas. A partir de estas preguntas, se decidirán cuáles se adaptarán al sistema para la prueba con estudiantes que se realizará en un futuro.

A la hora de jugar, únicamente se necesita el ratón, por lo que la adaptación en un futuro a dispositivos táctiles, si se deseara, sería más fácil.

CAPÍTULO 7
Pruebas

En este apartado se detallan el tipo de pruebas que se han realizado para garantizar que se cumplen tanto los requisitos funcionales, cómo los requisitos no funcionales y los casos de uso.

7.1 Playtesting

La principal metodología de pruebas usada para este proyecto es *Playtesting*. Esta metodología es muy utilizada por los estudios de desarrollo de videojuegos, ya que permiten identificar como los usuarios interactúan y juegan con el juego. Este proceso puede ser realizado durante diferentes fases del desarrollo de la aplicación. Realizar estas pruebas en fases tempranas o intermedias del desarrollo permiten analizar la experiencia de usuario, también llamado *user experience* o *UX*, e identificar qué sistemas no funcionan correctamente, para así cambiarlos o replantear partes del diseño del juego. En fases avanzadas o incluso con el producto ya finalizado permiten identificar errores muy poco frecuentes o situaciones no deseadas que se hayan pasado por alto. Hay que tener en cuenta que los videojuegos son sistemas muy complejos y a mayor escala, más puntos críticos tiene.

Esta metodología de pruebas tiene la ventaja de que al estar el moderador delante de los jugadores viéndole jugar, pero sin intervenir, puede observar cada uno de los movimientos, lo cual permite que recrear un problema encontrado sea más simple. Además, los resultados obtenidos pueden combinarse a la perfección con otros métodos de análisis, como una entrevista.

Las grandes empresas de videojuegos disponen de departamentos que se dedican a realizar pruebas exclusivamente, siendo el *playtesting* una de las más frecuentes. Estos departamentos se llaman departamentos de aseguramiento de calidad, *quality assurance department* o *QA department*. Dado que no se dispone de un equipo de tal tamaño para esta función, se han realizado pruebas de *playtesting* primero a compañeros de la universidad para comprobar que el juego no sufría errores que le impidieran funcionar.

Los primeros participantes fueron seis compañeros de la *Start.inf* de la *Escola Tècnica Superior d'Enginyeria Informàtica* (ETSINF), varios de los cuales son desarrolladores de videojuegos, y pueden aportar una valoración técnica del videojuego. De acuerdo con Nielsen, sólo se necesitan cinco usuarios para realizar pruebas de *playtesting* [12]. Casi todos los problemas que tenía el prototipo en ese momento fueron identificados, y posteriormente solucionados, como por ejemplo, los jugadores no modificaban su nombre si este contenía espacios, o si soltabas un dado encima de otro, el de abajo se borraba.

Posteriormente, ocho miembros de *University Game Devs*, una asociación de alumnos pertenecientes a la ETSINF enfocada al desarrollo de los videojuegos, participaron en pruebas de *playtesting* para comprobar sistemas aislados a lo largo del desarrollo, como el movimiento de los dados o el movimiento de cámara, para comprobar que la respuesta a estos fuera positiva. Faulkner afirma que, a diferencia de lo que dice Nielsen, el número de participantes debería estar alrededor de diez [13]. La cantidad de errores existentes era baja, pero se consiguieron identificar varios problemas que no ocurrieron en las partidas de otros de los participantes, fueron sucesos aislados los cuales se pudieron replicar y solucionar. Uno de estos problemas encontrados y solucionados fue que al mover la cámara entre todos los jugadores rápidamente, esta dejaba de funcionar.

Queda pendiente una prueba que realizar con estudiantes en los ordenadores de la escuela. Durante esta prueba, el juego implementará una encuesta final para valorar la experiencia y satisfacción con el prototipo.

7.2 Pruebas de rendimiento

Para asegurar que el juego puede funcionar en cualquier ordenador, se preparó una máquina virtual para hacer pruebas con un sistema operativo *Lliurex*, con un total de 4GB de memoria RAM. Dado que es un juego usando el componente *Unity2D*, permite reducir el coste de renderizado gráfico y computacional, no se estableció una frecuencia de procesador ni memoria gráfica mínima para funcionar, siempre que estos valores fueran aceptables. Estas pruebas se van a realizar gracias a la herramienta del *Profiler* que *Unity* trae integrado, el cual analiza el consumo de recursos del sistema.

Figura 7.1: Prueba de rendimiento: RAM

Cómo puede verse en la Figura 7.1, durante 20 minutos de partida, *Profiler* ha calculado que el total de la memoria RAM usada por el juego son 1,16GB, lo cual no es un problema para nuestra máquina virtual. Esto nos garantiza que el juego puede funcionar en cualquier ordenador de un centro educativo con más de 2GB de RAM. Por la parte de la memoria gráfica, como se puede observar en la Figura 7.2, el *Profiler* recomienda usar una tarjeta gráfica con una memoria superior a los 16MB, ya que al estar hecho a base de imágenes de tamaño reducido y *sprites*, no supone ningún problema tampoco para cualquier tarjeta gráfica integrada en placas base de hoy en día.

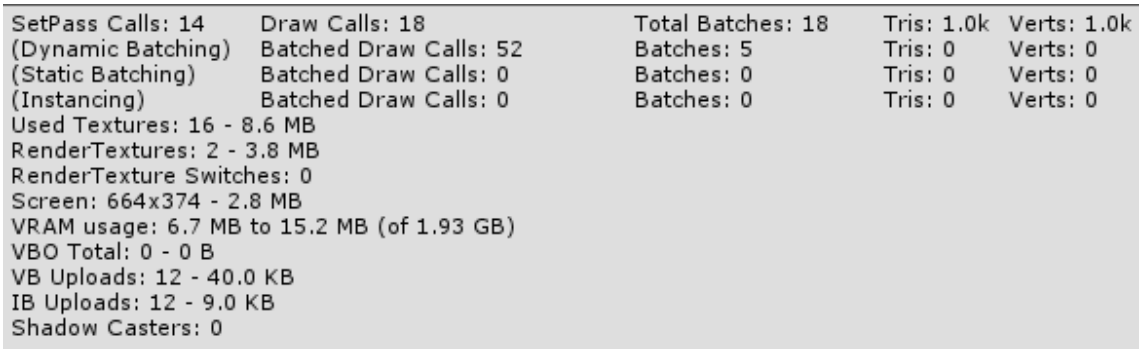

Figura 7.2: Prueba de rendimiento: VRAM

El resto de valores están en unos intervalos aceptables, por lo que se puede garantizar que va a funcionar óptimamente tanto en sistemas *Windows* como *Lluirex*, lo cual era uno de los requisitos no funcionales del prototipo.

7.3 Pruebas funcionales

Para garantizar que el videojuego cumple con los requisitos funcionales y no funcionales, también se realizaron una serie de pruebas funcionales con la escuela asociada.

Se realizó una primera reunión con el profesor de tecnología, las profesoras de educación especial y la directora del colegio. En esta reunión se les mostró capturas de pantalla a la vez que el diseño conceptual del tablero y los personajes. Posteriormente, se detalló el funcionamiento y la mecánica que tiene.

Durante esta reunión, se tomaron apuntes de los comentarios y la crítica constructiva al sistema. Las profesoras de educación especial insistieron en el uso de la fuente *verdana* para facilitar la lectura a cualquier estudiante, esta característica creó el *RNF-04*. También se llegó a la conclusión de que el color amarillo no debería estar.

Se hizo hincapié en recordar de forma positiva la importancia de que los personajes jueguen en equipo, evitando individualismos. Esto causó que los poderes afectaran a los dados y no al personaje que los usa para que así, los jugadores puedan usar sus características especiales para ayudar al resto.

Se propusieron otras mejoras puramente visuales y de mecánicas muy secundarias como recordatorios. Estas mejoras se han incluido en el apartado de trabajos futuro, ya que dentro de la planificación no hay tiempo para hacerlo.

La siguiente prueba funcional a realizar es con el alumnado. Este alumnado estará compuesto por alumnos de primero de la ESO, y la prueba está planeada para realizarse en los próximos meses. Para esta prueba se incluirá una encuesta de satisfacción dentro del videojuego para evaluar la opinión que los estudiantes tienen de este.

CAPÍTULO 8
Conclusiones

En este capítulo se detallan las conclusiones a las que se ha llegado tras la realización de este proyecto, así como la relación del trabajo desarrollado con los estudios cursados.

La realización de este proyecto ha ayudado a reforzar los conocimientos de *Unity* adquiridos durante el grado, en especial, el uso de herramientas internas cómo el *Profiler* o las directrices de compilación por plataformas para crear código específico para diversos sistemas operativos. Incluso la implementación de *GUIManager*, el cual hace de intermediario entre el código y la *GUI* del motor y así evitar que el juego dependa en todo momento de *Unity* y esto facilite la forma de desarrollar.

Analizar el problema y buscar patrones de diseño que se adaptaran al sistema ayudó a la forma en la que se realiza la ingeniería del software en un proyecto más ambicioso. Rediseñar sistemas propios del motor para crear unos que se adaptaran a las necesidades del problema ayudó a cambiar la forma de pensar, y que a veces es mejor hacer tus propias implementaciones a usar las que un lenguaje te proporciona hechas, cómo el *Listener* de *Unity* o las listas enlazadas.

Haber analizado diversos artículos, proyectos y libros sobre la educación y los videojuegos ha ayudado a tener una visión más real del estado en el que están en la sociedad, lo fundamental que es explorar este campo que aún está dando sus primeros pasos.

La tecnología está avanzando rápidamente hasta el punto de que cualquier niño ha llegado a jugar a un videojuego, es algo que está siendo parte de sus vidas, y es importante que esa parte también evolucione junto a la sociedad. Es necesario que los *Serious games* enfocados a la educación estén a la par de atractivos como los juegos de puro entretenimiento. Además, ver que las grandes empresas empiezan a interesarse por estos juegos es una buena señal para que otros se animen y apoyen un sector tan criticado por los medios.

El principal contratiempo ha sido encontrar el diseño del videojuego. Había que tener en cuenta muchos factores que hacían que el diseño cambiara constantemente en las primeras etapas del desarrollo. Factores que iban apareciendo sobre la marcha al darnos cuenta de pasábamos por alto cosas que no debíamos, como que los dados indicaran el valor con puntos y no con números.

En cuanto al resultado obtenido, es un simple prototipo funcional de un juego mucho más grande con más sistemas que aún no existen. Es lo que se llamaría el *core*, o núcleo, del videojuego. El balance final es positivo, aunque hay aspectos que se podrían mejorar, o incluso cambiar, si en futuras iteraciones dejan de tener una función práctica en el juego.

El archivo de configuración está lejos de ser perfecto, ya que depende demasiado del formato. Esto puede suponer problemas de funcionalidad si algún elemento no está

insertado correctamente, como no poner el texto entre comillas o la incompatibilidad de introducir ecuaciones en modo texto.

Consideramos los objetivos iniciales cumplidos al haber desarrollado un prototipo de videojuego que sea capaz de entrenar y mejorar las *soft skills* en los estudiantes de secundaria, mediante un sistema escalable y personalizable.

Finalmente, añadir que los plazos se han llegado a cumplir, aunque la disponibilidad de los colaboradores realentizó el desarrollo de las partes de preguntas, pistas y configuración, pero ese problema se resolvió priorizando tareas que no eran tan prioritarias para no retrasar el desarrollo del proyecto.

8.1 Relación del trabajo desarrollado con los estudios cursados

El trabajo realizado hace uso de tecnologías que han sido impartidas por las asignaturas estudiadas durante el grado. Las asignaturas que tienen especial relación son las siguientes:

- **Diseño de software**: Se ha puesto en práctica patrones de diseño aprendidos durante la asignatura, como el *Singleton* o el *Observer*.
- **Ingeniería del software**: Durante esta asignatura se aprendió a diseñar software mediante diagramas, además de utilizar el *IDE* utilizado durante el proyecto, el *Microsoft Visual Studio*.
- **Introducción a la programación de videojuegos**: Esta asignatura introduce una gran cantidad de aspectos sobre el desarrollo de videojuegos, como la generación de un *GDD*, así como decidir qué mecánicas introducir en un juego o qué sistemas evitar para no generar situaciones críticas en el sistema.
- **Análisis y especificación de requisitos**: Los conocimientos de esta asignatura han permitido la generación de casos de uso, actores y requisitos, tanto funcionales como no funcionales.
- **Calidad de software**: Asignatura que imparte conocimientos referente a la calidad de los sistemas software, para identificar la escalabilidad, usabilidad y otras características.

8.2 Relación del trabajo desarrollado con las competencias transversales UPV

Ya que se ha hablado durante todo el proyecto sobre las *soft skills*, o competencias transversales, se ha incluido las competencias que se han reforzado tras la realización del proyecto:

- **Comprensión e integración**: Al comprender e integrar conocimientos de videojuegos y aplicarlos sobre la educación, creando así un *serious game*.
- **Aplicación y pensamiento práctico**: Al aplicar los conocimientos vistos en las distintas asignaturas del grado comentadas en el apartado anterior y aplicarlas para crear un prototipo de un videojuego.
- **Análisis y solución de problemas**: Al realizar toda la primera fase del proyecto y diseñar un prototipo en base a los resultados obtenidos.

- **Innovación, creatividad y emprendimiento**: Al diseñar un videojuego que busca cambiar la forma en la que los estudiantes aprenden.
- **Diseño y proyecto**: Al llevar el videojuego desde un diseño preliminar a un prototipo funcional.
- **Trabajo en equipo y liderazgo**: Al realizar reuniones y conseguir unos objetivos comunes con las profesoras y profesores del colegio Purísima Hnas. Franciscanas.
- **Responsabilidad ética, medioambiental y profesional**: Al hacer el que el juego sea inclusivo y no discriminar a ningún estudiante.
- **Comunicación efectiva**: Al explicar el proyecto a la directora del colegio asociado y conseguir que quisieran participar con nosotros.
- **Pensamiento crítico**: Al decidir la solución propuesta de entre todas las soluciones posibles.
- **Conocimiento de problemas contemporáneos** : Al entender la valoración pública de los videojuegos y proponer cambios para mostrar su utilidad mas allá del entretenimiento.
- **Aprendizaje permanente**: Al adaptar patrones y estructuras definidas en el motor de juego para generar nuevas soluciones y evitar reproducir soluciones ya conocidas.
- **Planificación y gestión del tiempo**: Al programar el tiempo dedicado a cada fase del proyecto y alcanzar los objetivos en el plazo.
- **Instrumental específica**: Al utilizar *Unity* para la realización del prototipo, siendo específico para la realización de videojuegos.

CAPÍTULO 9
Trabajos futuros

En este capítulo se describe las posibles mejoras o trabajos futuros en base al proyecto realizado.

Debido al gran avance de los videojuegos en los últimos años y de que actualmente prácticamente todo el mundo dispone de un ordenador o *tablet* en casa, esto puede beneficiar al desarrollo de videojuegos de carácter educativo y en especial a este proyecto.

Esto es un simple prototipo de un juego completo más complejo, es una base en la cual construir un sistema que pueda beneficiar aún mas a aquellos profesores o incluso padres que busquen una forma diferente de enseñar. Dentro de las mejoras posibles se destacan las siguientes:

- Añadir más tipos de rompecabezas a las casillas de rompecabezas, como problemas matemáticos, juegos de rellenar frases o incluso de escuchar un audio y realizar algo en base a lo escuchado. El juego está preparado para aceptar cualquier tipo de minijuego sin alterar su diseño interno. Incluso se podrían diseñar rombecabezas que se deban completar por parejas.
- Refinar el archivo de configuración o incluso implementar una aplicación externa que permita una modificación mediante un intérprete externo al juego. Es interesante que el juego no solo sea beneficioso para el estudiante, sino que también sea fácil de configurar para la persona que lleve el rol de profesor.
- Personalización de los personajes. Los estudios muestran que los jóvenes dan valor a poder personalizar su avatar en el juego. Una personalización de estos podría hacer que su inmersión fuera mayor y por ello que los resultados sean mejores.
- Se podría implementar un sistema de puntuación en base al trabajo en equipo y a la participación entre ellos mediante acciones dentro del juego, para hacer ver que pueden obtener mejores resultados trabajando juntos. Actualmente, el juego está planteado a que si juegas solo y por tu parte, tengas menos ayudas y sea más complicado.
- Aprovechando el avance de los teléfonos y *tablets*, se podría realizar una versión *Android* o *iOS* del videojuego a la hora de ampliar las opciones que tienen los colegios para implementar el juego.
- Permitir la creación de paquetes de contenido de forma que se puedan compartir fácilmente, de forma que se pueda aprovechar las preguntas de otras escuelas o compañeros para usarlas en otros centros o incluso mejorar dichos paquetes.
- Generación de elementos visuales para recordar de forma positiva y guiar a los jugadores por el tablero, mediante bocadillos de dialogo o iconos descriptivos.

Bibliografía

[1] OECD (2018) *OECD Science, Technology and Innovation Outlook 2018: Adapting to Technological and Societal Disruption.* OECD Publishing, Paris.

[2] OECD (2017) *PISA 2015 Results (Volume V): Collaborative Problem Solving.* PISA, OECD Publishing, Paris, p 139

[3] OECD (2005) *The definition and selection of key competencies: Executive summary* PISA, OECD Publishing, Paris, p 12

[4] Rello, L. (2014) *DysWebxia. A text Accessibility Model for People with Dyslexia.* PhD Thesis. Barcelona: Departament of Information and Comunication Technologies.

[5] Stokes, B. (2005) *Videogames have changed: time to consider 'Serious Games'?* Development Education Journal, vol. 11, p. 12

[6] Contreras Espinosa, R. y Eguia, L. (editores) *Experiencias de gamificación en aulas* Institut de la Comunicació (InCom-UAB) Publicacions, Institut de la Comunicació, Universitat Autònoma de Barcelona, Barcelona, 2017

[7] Fuentes García, N.M., Gutiérrez Vela, F. L., Paderewski Rodríguez, P., López Arcos, R., y Padilla Zea, N. (2015) *Teaching Emotions to Children by Using Video Games* Proceedings of the XVI International Conference on Human Computer Interaction, Article 20, p. 3

[8] Higueras, B. y Muñoz, J.J. (2012) *Psicología Básica.* Manual CEDE de Preparación PIR, vol. 08, p 337

[9] Kluzer S., Pujol Priego L. *DigComp into Action - Get inspired, make it happen*. Publications Office of the European Union, Luxembourg, 2018. ISBN 978-92-79-79901-3

[10] Contreras Espinosa, Ruth S. (2016) *Juegos digitales y gamificación aplicados en el ámbito de la educación.* RIED. Revista Iberoamericana de Educación a Distancia, vol. 19, núm.2, p. 27-33

[11] Contreras-Espinosa, Ruth S. y Eguia-Gómez, José Luis y Solano Albajes, Lluis (2016) *Investigación-acción como metodología para el diseño de un serious game.* RIED. Revista Iberoamericana de Educación a Distancia, vol. 19, num.2, p. 71-90

[12] Nielsen, Jakob *Why You Only Need to Test with 5 Users.* Nielsen Norman Group, March 19, 2000

[13] Faulkner, L. (2003) *Beyond the five-user assumption: Benefits of increased sample sizes in usability testing* Behavior Research Methods, Instruments, & Computers, vol 35, num. 3, p. 379-383

[14] Grand Theft Auto V: The Rise And Fall Of The DIY Self-Driving Car Lab Consultado el 4 de Octubre de 2019 en https://www.forbes.com/sites/aarontilley/2017/10/04/grand-theft-auto-v-the-rise-and-fall-of-the-diy-self-driving-car-lab/#75f384be7d7a

[15] Modo Descubrimiento de Assassin's Creed: Origins. Consultado el 8 de Octubre de 2019 en https://support.ubi.com/es-es/Faqs/000031846/Discovery-Tour-Mode-of-Assassin-s-Creed-Origins-ACO

APÉNDICE A
Acrónimos

- **CU**: Casos de Uso.
- **GDD**: *Game Design Document* o Documento de diseño del videojuego.
- **GUI**: *Graphic User's Interface* o Interfaz gráfica de usuario.
- **IDE**: *Integrated Development Environment* o Entorno de desarrollo integrado.
- **OECD**: Organización para la Cooperación y el Desarrollo Económicos.
- **QA department**: *Quality Assurance department* o Departamento de aseguramiento de calidad.
- **RF**: Requisito funcional.
- **RNF**: Requisito no funcional.
- **UX**: *User's experience* o Experiencia del usuario.

APÉNDICE B
Configuración del videojuego

El videojuego consiste en una carpeta con distintos archivos. Para su instalación es necesario ejecutar el archivo *.exe*, el cual generará un archivo *Configuration.json* que se almacenará en la carpeta *Documentos/SoftSkillsTFG* en el caso de Windows o en *\Home\SoftSkillsTFG* en el caso de Lliurex o Ubuntu.

B.1 Configuration.json

```
{
  "config": {
    "njugadores": 4,
    "preguntas": {
      "p_1": {
        "pista": {
          "pis_1": "La respuesta correcta es Correcto.",
          "pis_2": "La respuesta incorrecta es Falso."
        },
        "enunciado": "Este es el enunciado de una pregunta?",
        "opcion_1": "Correcto",
        "opcion_2": "Falso",
        "opcion_3": "Falso",
        "opcion_4": "Falso"
      },
      "p_2": {
        "pista": {
          "pis_1": "",
          "pis_2": ""
        },
        "enunciado": "",
        "opcion_1": "",
        "opcion_2": "",
        "opcion_3": "",
        "opcion_4": ""
      }
    }
  }
}
```