Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Creación de juegos serios para el aprendizaje de estructuras dinámicas

Autor:
D. Jorge Marciel Pariente

Tutores:
Dña. Alma María Pisabarro Marrón
Dr. Carlos Enrique Vivaracho Pascual

Resumen

Este trabajo forma parte de un proyecto de gamificación de la asignatura de primer curso del grado en ingeniería informática, *Fundamentos de Programación*, que pretende fomentar el aprendizaje mediante los juegos.

El Trabajo de Fin de Grado ha consistido en el planteamiento y desarrollo de dos videojuegos en *Unity* relacionados con la docencia, de forma que ayuden a comprender mejor los conceptos desarrollados a lo largo de la asignatura, en concreto sobre el tema de estructuras dinámicas. Uno de los juegos trata de representar el concepto de lista enlazada y el otro la diferencia entre pilas y colas. Además del desarrollo de los videojuegos, estos han sido integrados en una plataforma web, desarrollada anteriormente en otro Trabajo de Fin de Grado.

Tabla de Contenidos

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Este proyecto pretende el desarrollo de dos videojuegos mediante *Unity*, relacionados con docencia. En concreto, se van a plantear e implementar dos juegos serios para apoyar el aprendizaje en la asignatura de primero *Fundamentos de Programación*. Este trabajo forma parte de un proyecto global, que pretende desarrollar una serie de minijuegos serios para gamificar completamente la asignatura.

La gamificación en las aulas, se trata de una técnica de aprendizaje que traslada la mecánica de los juegos al ámbito educativo-profesional con el fin de conseguir mejores resultados: sirve para absorber conocimientos, para mejorar alguna habilidad para recompensar acciones concretas... Es un término que ha adquirido una enorme popularidad en los últimos años, sobre todo en entornos digitales y educativos.

En nuestro caso la gamificación se llevará a cabo mediante dos minijuegos desarrollados en *Unity*. El primero, *Apuntados*, que trata de explicar o representar la idea de lista enlazada, las operaciones que se pueden realizar sobre ellas y cómo actúan los punteros de los elementos de estas. El segundo videojuego, *Apilas*, intenta explicar cómo funcionan y la diferencia que existe entre los distintos tipos de estructuras, en concreto entre Pilas y Colas. Son videojuegos con temáticas diferentes, pero muy ligados, ya que ambos están relacionados con las ideas de estructura dinámica y lista enlazada.

Cabe destacar que en este documento se tratarán el desarrollo de los minijuegos en capítulos separados, cada uno con su propios apartados. Además se elaborará un capitulo sobre la integración de los minijuegos con la plataforma, que pretende servir como guía para futuros desarrollos dentro del marco del proyecto.

1.1. Motivación

En algún momento de nuestra vida dejamos de vivir el aprendizaje como algo apasionante y divertido, olvidando que el aprendizaje es en sí una experiencia memorable.

Dinámicas de juego basadas en el reto, la curiosidad, la colaboración o la exploración encajan perfectamente en el aula, pueden llegar a aumentar la motivación del estudiante e incrementar los resultados de éstos. Este tipo de dinámicas promueven además un ambiente donde el alumnado entiende el error como fuente de experiencia y aprendizaje, lo que estimula la creatividad y minimiza el miedo a la participación en el aula.

Existen ciertos conceptos o ideas que son muy difíciles de explicar de forma teórica. Utilizar este tipo de técnicas puede ser de gran ayuda a la hora de explicar estas ideas, permitiendo, entre otras cosas, que se puedan aprender estos conceptos de forma más divertida y motivadora.

Tanto estudiantes como docentes podrían verse beneficiados por esta metodología. Si se consigue un buen resultado, los estudiantes podrían mejorar sus resultados académicos, comprender mejor y con más facilidad los conceptos que se trata de explicar, y visualizar de manera rápida la aplicación de estos conceptos a la práctica.

1.2. Objetivos

El objetivo general de este proyecto es la creación de dos minijuegos en *Unity* para mejorar el aprendizaje de conceptos relacionados con listas enlazadas. Como objetivos concretos se plantean, además:

- Integrar ambos juegos en la plataforma del proyecto.
- Los juegos deberán ser lúdicos, representar un reto que motive al alumno a jugar por diversión.
- Los juegos deberán tener una progresión en los conceptos que representán, de lo mas fácil a lo mas difícil.
- La puntuación obtenida al final de cada nivel deberá representar la habilidad del jugador, pero siempre intentando no desviar el foco del aprendizaje.

1.3. Contenido de la Memoria

El contenido de la memoria se dividirá en 8 capitulos, el resto del documento se estructura de la siguiente forma:

- En el Capítulo 2 se desarrolla el tema de la gamificación como herramienta de aprendizaje y algunos conceptos básicos relacionados con el desarrollo de videojuegos.
- En el Capítulo 3 se describen todas las tecnologías y herramientas mas importantes empleadas para el desarrollo del proyecto.
- En el capitulo 4 se describe la planificación inicial del proyecto junto con su seguimiento, la estimación de costes del mismo y el plan de riesgos.

- En los capítulos 5 y 6 se expone el desarrollo de los videojuegos, de forma separada, cada uno en un capitulo. Para cada videojuego se ha elaborado *GDD*, análisis, diseño y pruebas.
- El capitulo 7 está dedicado a la integración de los minijuegos con la plataforma. Pretende servir como guía para futuros desarrollos dentro del marco del proyecto.
- Por último, en el Capítulo 8 se finaliza esta memoria con unas breves conclusiones, así como posibles futuras mejoras que podrían ser incluidas.

Capítulo 2

Contexto

Los videojuegos llevan con nosotros desde hace décadas, y siempre han sido utilizados tanto por jóvenes como por mayores.

En los últimos años el mundo de los videojuegos ha continuado con su imparable avance, y, actualmente, es una de las mayores industrias del mundo, tanto en cuanto al dinero que maneja como por la cantidad de personas que hacen uso de sus productos. Para muchos jóvenes jugar a videojuegos es parte de su vida, a modo de pasatiempo, diversión u ocio. Sin embargo, en los últimos años los videojuegos también se utilizan como herramienta educativa que permite a los estudiantes desarrollar competencias de manera lúdica en sus procesos de aprendizaje.

Motivar a los alumnos puede suponer un quebradero de cabeza para cualquier docente, nos encontramos ante una tarea nada fácil. Los juegos poseen características que hacen que sean divertidos, lo que los convierte en una poderosa herramienta para aumentar la motivación de los estudiantes cuando se utilizan en un entorno educativo, incluida la enseñanza universitaria.

Se llevan años utilizando técnicas de gamificación en las aulas, aunque las últimas tendencias en juego serios apuntan al uso de juegos digitales y videojuegos como herramientas de adquisición de conocimiento [5].

La ludificación o gamificación [13] se define como el uso de técnicas, elementos y dinámicas propias de los juegos y el ocio en actividades no necesariamente recreativas con el fin de potenciar la motivación, así como de reforzar la conducta para solucionar un problema, mejorar la productividad, obtener un objetivo, activar el aprendizaje y evaluar a individuos concretos.

No todo lo que tenga que ver con el uso del juego en entornos no lúdicos es gamificar, además existen diferentes estrategias o enfoques sobre el tema. En nuestro caso el enfoque es el de juegos serios o educativos, ya que son juegos creados específicamente con fines educativos.

La gamificación supone grandes beneficios [8], entre ellos:

- **Hace que el aprendizaje sea divertido e interactivo:** la gamificación puede ayudar a crear contenidos emocionantes, educativos y entretenidos. Además, gracias a los elementos competitivos puede llegar a convertir el aprendizaje en algo muy divertido.
- **Hace más fácil la retención de conocimientos:** otra ventaja de la gamificación es la sensación de bienestar que nos puede provocar y las consecuencias que tiene sobre el aprendizaje. Cuando ganamos en un juego o conseguimos un logro importante nos sentimos bien y generamos dopamina, algunos estudios apuntan que la dopamina es clave para el aprendizaje, los aprendizajes con carga emocional duran más.
- **Permite ver la utilidad real:** la gamificación permite ver directamente representadas cuales son los resultados de las decisiones tomadas a lo largo del juego y como pueden aplicarse los conceptos explicados en situaciones prácticas. Refuerza el aprendizaje mediante la aplicación de los conocimientos en situaciones prácticas.
- **Proporciona retroalimentación en tiempo real:** la progresión en el juego permite conocer que tal lo estamos haciendo. Además, ayuda a trabajar por conseguir objetivos reales, medibles y significativos.
- **Favorece la socialización:**los alumnos que participan en esta gamificación pueden trabajar juntos, colaborando y compitiendo. Esto fomenta el trabajo en equipo y el pensamiento crítico.

Existen también una serie de problemas o inconvenientes asociados a la gamificación que pueden llegar a darse y debemos intentar combatir en la medida de los posible:

- **Pueden desmotivar al jugador:** los jugadores que no consiguen una buena puntuación pueden desmotivarse, porque no consiguen completar el juego y se sienten más desplazados de la asignatura. Por otra parte, los jugadores que completan con mucha facilidad el juego pueden olvidar los conceptos aprendidos con mucha facilidad. Para intentar combatir estos problemas el diseño de niveles se ha llevado a cabo de forma que completar los niveles sea asequible y conseguir la puntuación máxima del nivel requiera cierta práctica.
- **Puede focalizar al jugador en ganar:** los jugadores pueden llegar a concentrarse en ganar el videojuego, dejando de lado el objetivo de aprender. Para evitar esta situación los juegos cuentan con mecánicas muy sencillas, que intentan desviar lo mínimo posible al jugador del aprendizaje. Además, los juegos apoyan con contenido teórico los conceptos que tratan de explicar.

2.1. Conceptos Relacionado con el Desarrollo de Videojuegos

En esta sección se van a explicar algunos conceptos relacionados con el desarrollo de los videojuegos que se utilizarán a lo largo del documento.

2.1.1. Documento de Diseño del Juego

El *Documento de Diseño del Juego* o GDD (Game Design Document) incluye toda la información de diseño del juego. El que lo lea debe tener una visión clara del juego y de todo lo relacionado con su diseño. Este documento es utilizado como guía durante el proceso de desarrollo del juego.

Diseñar un juego es un proceso iterativo, por lo que la creación del *GDD* también. A pesar de que se considera un documento clave en el desarrollo de un videojuego, este documento no tiene ninguna forma estándar. Es decir, los puntos que se tratán en el documento no son fijos y deben adaptarse en función del videojuego, aunque en esencia todos contienen la misma información organizada de manera distinta.

En nuestro caso vamos a seguir el esquema de *Tracy Fullerton* [14], que consta de:

- **Resumen:** En este apartado se da una visión resumida del juego. Se intenta capturar y trasmitirla al lector la esencia del juego, de la forma mas veraz posible.
- **Audiencia, plataforma y marketing:** Se describe la audiencia objetivo del juego, las plataformas en las que estará disponible y un estudio de mercado. En nuestro caso no los juegos no serán puestos a la venta, por lo que la realización de un estudio de mercado no procede.
- **Mecánicas:** Se describen las mecánicas del videojuego, es decir, como se juega. Además, se puede especificar los controles, niveles o modos de juego.
- **Personajes:** En este apartado se desarrollan los personajes y sus cualidades, si es que existen.
- **Mundo:** Si tu juego incluye una historia, la puedes incluir en este apartado. Al igual que el apartado anterior es opcional.
- **Multimedia:** Se hace una lista de los recursos multimedia que serán necesarios para el desarrollo del videojuego. Dependiendo del juego pueden incluir entornos, personajes, animaciones, música, efectos y muchos mas.

2.1.2. Pruebas Beta

Las pruebas son una parte fundamental de cualquier desarrollo. Nos sirven para observar como responde nuestro sistema y detectar posibles errores.

Como en la mayoría de videojuegos, al estar formados por grandes subsistemas de mecánicas que se relacionan entre sí para generar dinámicas, es muy complicado para un grupo de desarrolladores encontrar todos los posibles errores.

Para identificar el mayor número de errores posibles, los desarrolladores, permiten que los jugadores puedan jugar al juego antes de lanzar al mercado la versión final abierta del videojuego.

Dentro de estas pruebas existen diferentes tipos, entre ellos:

- **Focus Group:** Un grupo de jugadores dando sus opiniones de manera libre y sin guion formal, aunque es guiado por uno o varios moderadores.
- **Prueba de usabilidad:** pruebas sobre la interfaz.
- **Prueba del juego:** Un grupo de personas probando el juego para analizar sus reacciones y ver si la experiencia es la esperada.

Capítulo 3

Herramientas Utilizadas

3.1. *Unity*

El apogeo de los videojuegos ha traído como consecuencia que se interesen cada vez más personas por este mundo. Como consecuencia han nacido los Game Engines que son herramientas cuyo fin es ayudar a las personas que quieran desarrollar un videojuego. Esto es posible gracias a que estas herramientas cuentan con motores, funciones y componentes que logran el renderizado, animación y otros aspectos del desarrollo del videojuego.

El motor utilizado para desarrollar este proyecto es *Unity*, una de las plataformas de creación a tiempo real más utilizadas en el desarrollo de videojuegos de todo el mundo.Existen más herramientas de desarrollo, pero en este caso el uso de Unity es un requisito necesario, ya que la integración con la plataforma del proyecto, desarrollada previamente, así lo requiere.

La idea de la que surge *Unity* es la "democratización de los videojuegos", es decir, hacer más accesible el desarrollo de contenidos interactivos a personas de todo el mundo. Fue creado en 2005 por parte de *Unity* Technologies. Su éxito ha propiciado una gran expansión y como consecuencia de su éxito ha recibido actualizaciones, situándose como una de las plataformas líderes en el mercado.

Unity [16] es un motor multiplataforma para el desarrollo de videojuegos. Es decir, permite desarrollar juegos para distintas consolas y dispositivos desde una misma base, sin tener que crearlo desde cero para cada plataforma. Al ser capaz de compilar para diferentes plataformas, permite crear videojuegos habilitados para web, móviles, consolas, smart TV e incluso dispositivos de realidad aumentada, de manera que es un motor de desarrollo prácticamente universal que ahorra costos significativamente a programadores y desarrolladores independientes.

Unity es una herramienta que no engloba únicamente motores para el renderizado de imágenes, de físicas de 2D/3D, de audio, de animaciones y otros motores, sino que engloba además herramientas de networking para multijugador o soporte de Realidad Virtual.

Actualmente *Unity* es uno de los motores de desarrollo de videojuegos más populares del sector de los videojuegos [15], se ha utilizado para crear multitud de juegos conocidos y otros no tan conocidos. Algunos de los juegos más famosos son: *Monument Valley*, *Ghost of a Tale*, *Hollow Knight* o *Cuphead*. También se ha utilizado para crear experiencias de Realidad Virtual interactivas e incluso miniseries, como *Baymax Dreams*, producida por *Disney* junto con Unity.

Además de lo comentado anteriormente, *Unity* cuenta con algunas ventajas o características clave como:

- **Aprendizaje de Unity:** *Unity* destaca por ser un sistema fácil de usar, las personas se familiarizan rápido con su interfaz intuitiva, ya que muchas acciones se basan en arrastrar y soltar. Esto provoca que el desarrollador se sienta más cómodo a la hora de ajustar y personalizar aquellas partes del juego que lo requieran.
- **Documentación:** *Unity* dispone de una excelente documentación, de las mejores documentaciones de software que existen. En su manual se pueden consultar todos y cada uno de sus distintos apartados, desde cómo actualizar a una versión concreta, hasta guías de expertos para realizar tareas bastante avanzadas. En dicho manual también aparece el historial de las versiones anteriores de su documentación, esto permite no estar limitados a la última versión de *Unity* que exista.
- **La Comunidad:** En la propia web de *Unity* existen una inmensa cantidad de contenido didáctico, pero, además, al ser uno de los motores de desarrollo más utilizados podemos encontrar una enorme cantidad de información en internet. Existe una gran facilidad para la ayuda entre usuarios cuando surgen dudas o necesitamos resolver algún problema.
- **La Asset Store:** La Asset Store es la tienda online de recursos de *Unity*. Aquí pueden encontrarse herramientas, elementos, objetos, mapas y efectos para el desarrollo de videojuegos. Esto facilita el trabajo de los desarrolladores, además de abrir un nuevo mercado para programadores que se dedican a crear y vender recursos específicos para otros creadores.
- **Alta calidad gráfica:** Los gráficos tanto de 2D como en 3D presentan una calidad bastante buena y adecuada a las exigencias que tiene el publico en la actualidad.

3.1.1. *UnityEngine*

UnityEngine es la librería esencial en el desarrollo de juegos en *Unity*. Nos proporciona una serie de recursos básicos a la hora de desarrollar juegos en *Unity*.

En la figura 3.1 podemos ver como se estructuran las clases de esta librería. El diagrama muestra las principales relaciones existentes entre las clases más importantes que conforman el modelo de *Unity*.

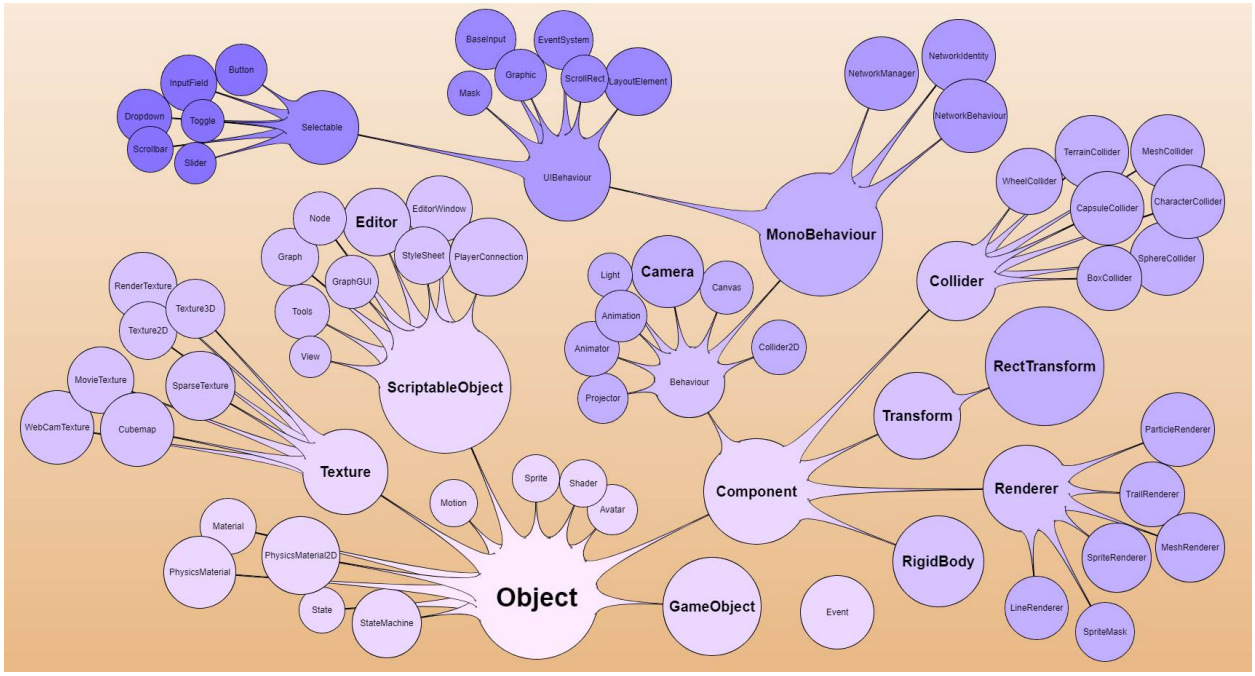

Figura 3.1: Diagrama estructura de clases en *Unity*

En la practica, la estructura principal de un juego son las escenas, que contienen los entornos y menús del juego. Cada escena es como un nivel único. En cada escena se pueden colocar entornos, obstáculos y decoraciones, lo que permite diseñar y construir el juego de forma modular. Dentro de estas escenas podemos añadir *GameObjects*, que se trata del concepto más importante del Desarrollo.

Cada elemento dentro del juego es un *GameObject*, desde personajes hasta luces, cámaras y efectos especiales. Sin embargo, un *GameObject* no puede hacer nada por sí mismo; necesita propiedades para cumplir una función. Para darle a un *GameObject* las propiedades que queremos, debemos agregarle components. Según el tipo de elemento que deseemos crear, podemos agregar diferentes combinaciones de componentes al *GameObject*.

Transform es el componente mas común dentro de *Unity*. Es imposible crear un *GameObject* sin un componente *Transform*. Este componente define la posición del *GameObject*, la rotación y la escala en el mundo y la escena. Los *GameObjects* puede contener otro tipo de componentes. Algunos de los Componentes mas utilizados son *SpriteRenderer*, que se encarga de mostrar un *Sprite* o imagen o el componente *RigidBody*, que se encarga de gestionar las físicas.

Los *Scripts* son componentes programables que nos permiten definir el comportamiento de *GameObjects* y de sus componentes, su uso es un ingrediente esencial en todos los juegos. Incluso el juego más simple necesita scripts para responder a las entradas del jugador y asegurar que los eventos del juego se ejecutan en el momento adecuado. Además, los *Scripts* pueden utilizarse para crear efectos gráficos, controlar el comportamiento físico de objetos o incluso implementar un sistema de inteligencia artificial para los personajes del juego. Estos scripts son también componentes, en concreto MonoBehaviour y nos permiten llevar el desarrollo en *Unity* a otro nivel de personalización.

Además de los eventos desencadenados por el jugador también existen funciones en los scripts que se ejecutan cuando se cumplen ciertas condiciones, la figura 3.2 muestra todos estas funciones, su orden de ejecución y sus condiciones de entrada y salida.

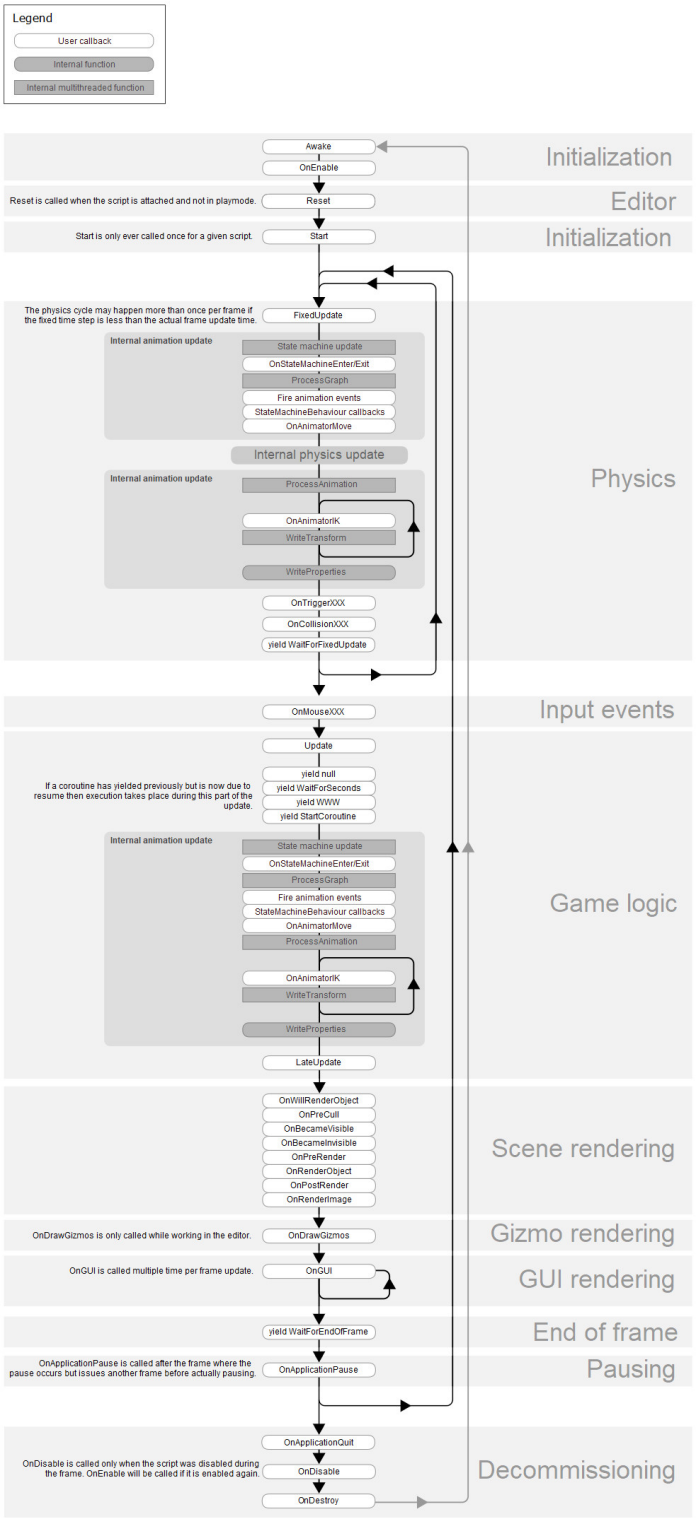

Figura 3.2: Diagrama de flujo de ejecución de funciones en *Unity*

Algunas de las mas importantes son:

- **Start**: Esta función solo se llama una vez por *script* y se ejecuta en el primer fotograma en el que el este se encuentra activo.
- **Update**: Esta función llama cada vez que se muestra un nuevo fotograma, en caso de que el *script* esté activo.
- **OnDestroy**: Esta función es la ultima en ser llamada se llama justo antes de destruir el propio *script*.

3.2. Git

Git [7] es una herramienta gratuita de control de versiones de código abierto. Se define como control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún software. Estas herramientas nos sirven para trabajar en equipo de una manera mucho más simple y optima cuando estamos desarrollando software, pero no solo eso, además nos permiten tener un historial de los cambios realizados y copias de seguridad por si surgen errores durante el desarrollo.

Git se encuentra organizado en repositorios, cada uno de los cuales almacena los archivos que constituyen un determinado proyecto. Permite dividirlo en ramas. Cada una de ellas surge de otra y constituye una copia de todos los archivos y contenidos que en ese momento existieron en todos los puntos del proyecto. Git es un sistema de control de versiones distribuido y permite coordinar repositorios locales y remotos.

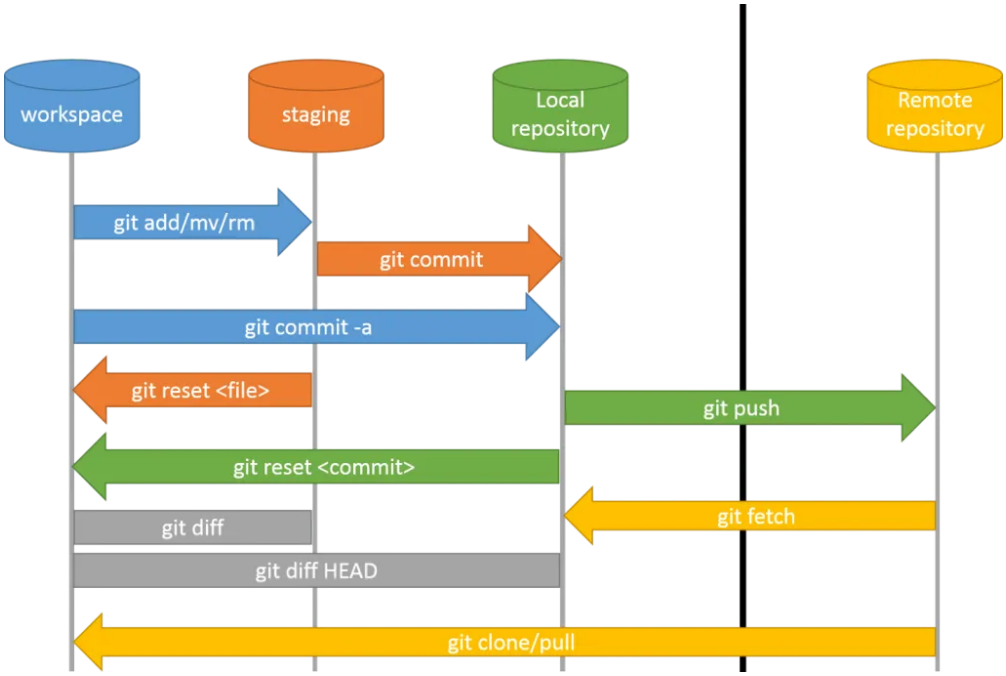

Figura 3.3: Funcionamiento de Git

Con este tipo de herramientas podemos mantener copias de seguridad del proyecto de manera remota a la par que se van realizando los cambios de código pertinentes y se realizan las pruebas correspondientes de manera local. Una vez verificado el trabajo se pueden de nuevo unificar el contenido. En la figura 3.3 puede observarse un esquema de funcionamiento de *Git*.

Además de lo comentado anteriormente GitHub cuenta con un plugin, que permite integrar Github en tu proyecto *Unity*, por lo que facilita mucho el trabajo [4].

El uso de *Git* es posible para juegos pequeños como el nuestro, pero para juegos grandes, el tamaño puede suponer un problema. *Unity* recomienda el uso de *Plastic SCM* [9] como herramienta de control de versiones, que nos permite evitar estos problemas.

Capítulo 4

Plan de Proyecto

4.1. Metodología

Para el desarrollo de este proyecto se ha escogido una metodología de desarrollo ágil que podría clasificarse como *Crystal Clear* [10].

El desarrollo ágil es un enfoque de trabajo que permite a los desarrolladores llevar a cabo proyectos adaptándose a los cambios y evolucionando para cubrir las necesidades reales. Este enfoque se refiere a métodos basados en el desarrollo iterativo e incremental. La figura 4.1 muestra el ciclo de entrega de este tipo de metodologías.

Cada iteración del proceso de desarrollo pretende incrementar el valor del código e incluye sus propias fases de planificación, diseño, codificación, pruebas y documentación.

Figura 4.1: Ciclo de entrega en metodologías ágiles

El conjunto de metodologías *Crystal Clear* tienen como características principales:

- **Participación del usuario:** El equipo de desarrollo tiene acceso a uno o mas usuarios del sistema que están construyendo, para ir asegurarse el cumplimiento de los requisitos del cliente y depurando errores de manera temprana.
- **Entregas frecuentes:** Se debe entregar al usuario funcionalidad "usable" con una frecuencia de entre 2 semanas y no más de un mes.
- **Crecimiento reflexivo:** es necesario que el equipo lleve a cabo reuniones periódicas de reflexión que permitan crecer y hacernos más eficientes.
- **Comunicación constante:** Uno de los pilares de cristal clear es la comunicación. Promueve prácticas como el uso de pizarrones, pizarras y espacios destinados a que los miembros del equipo puedan ver claramente el progreso del trabajo.

4.1.1. Roles

Los equipos *Crystal Clear* son pequeños, de menos de ocho personas y pueden contar con usuarios expertos como parte del equipo. Se pueden encontrar roles como:

- **Patrocinador:** es la persona que finalmente permite la realización del proyecto. Esta persona provee el dinero necesario para la ejecución inicial del proyecto.
- **Usuario experto:** son parte de los futuros usuarios del sistema. El sistema en desarrollo debería responder mejor y aportar más valor a la organización, cuando el equipo de desarrollo se encuentra en contacto con usuarios expertos del sistema.
- **Diseñador programador:** En este tipo de desarrollos el concepto de diseñador y programador van unidos, de esta manera se consigue un código de mayor calidad. Se encarga de producir el código necesario para la ejecución del sistema.
- **Coordinador:** El coordinador es responsable de mantener el orden, reducir conflictos y facilitar discusiones. Deberá, como mínimo, tomar nota de cómo va el proyecto, planeando y verificando el estado de cada sesión, combinando toda esta información para posteriormente publicarla.

4.1.2. Adaptación de *Crystal Clear* a Este Proyecto

Crystal clear debe ser adaptado a todos los contextos donde se aplique. En este contexto se requiere una importante adaptación, ya que se trata de proyecto que se realiza por un alumno como trabajo de fin de grado.

En este caso, puesto que el equipo va a estar formado solamente por una persona, el alumno se encargará de ser a la vez Diseñador Programador y Coordinador, por otra parte, los tutores desempeñaran la función de patrocinadores y usuarios expertos.

Se realizarán iteraciones de aproximadamente 3 semanas, finalizada la iteración se realizará una revisión del progreso del proyecto. En esta revisión se marcarán las tareas que han sido completadas y se decidirá que tareas van a formar parte de la siguiente iteración. Si es posible se realizará también una reunión al finalizar cada iteración, en la que se puede incluir también la entrega de software.

Teniendo en cuenta el tiempo del que dispone el alumno para realizar el proyecto, los primeros meses serán dedicados exclusivamente a realizar la planificación y documentación iniciales, plantear la arquitectura y estructura del proyecto, poner en marcha todas las herramientas necesarias y dejar preparada por completo el entorno de desarrollo de la aplicación. De este modo se pretende que las iteraciones, cuya duración total abarcará alrededor de unos 4 meses, sean prácticamente dedicadas sólo a la implementación y las tareas paralelas de documentación. Esta actividad permitirá al alumno profundizar en su documentación al respecto, teniendo claros todos los aspectos relevantes para la correcta realización del proyecto en cuestión.

El Trabajo Fin de Grado del Grado en Ingeniería Informática de la Universidad de Valladolid corresponde a 12 créditos o 300 horas. Dada la disponibilidad del alumno, se emplearán unas 45 horas por iteración, aproximadamente 3 horas al día. En total se estima que el proyecto será desarrollado en 7 iteraciones, sumando un total de 315 horas.

Entre la tercera y cuarta iteración no se trabajará en el proyecto, por vacaciones, por lo que las semanas del 21 de Diciembre al 11 de Enero no formarán parte de la planificación. En la tabla 4.1 podemos observar la distribución de las iteraciones en el tiempo.

iteración	**Comienzo**	**Finalización**
iteración 1	28/08/2020	19/10/2020
iteración 2	19/10/2020	09/11/2020
iteración 3	09/11/2020	30/11/2020
iteración 4	30/11/2020	21/12/2020
iteración 5	11/01/2021	01/02/2021
iteración 6	01/02/2021	22/02/2021
iteración 7	22/02/2021	15/03/2021

Tabla 4.1: planificación inicial de iteraciones

4.2. Plan de Riesgos

Un riesgo [12] es un evento o circunstancia cuya probabilidad de ocurrencia es incierta, pero que, en caso de aparecer, tiene un efecto (positivo o negativo) sobre los objetivos de un proyecto.

Estos riesgos se pueden dividir en varios tipos [6]:

- **Riesgos del proyecto:** Afectan a la planificación temporal, al coste y calidad del proyecto. Identifican problemas de presupuesto, calendario, personal, recursos...

- **Riesgos técnicos:** Amenazan la calidad y la planificación temporal del software que hay que desarrollar. Identifican posibles problemas de diseño, implementación, obsolescencia técnica, interfaz...
- **Riesgos del negocio:** Amenazan la viabilidad del software. Se trata de riesgos de mercado, estratégicos, de ventas o de presupuesto.

Existen una serie de pasos clave para elaborar una planificación y gestión de riesgos:

1. Identificar riesgos.
2. Diseñar planes para compensar el riesgo.
3. Prevenir proporcionalmente al riesgo.
4. Monitorizar estado de los riesgos.

En esta sección se van a detallar los riesgos detectados que se han tenido en cuenta a la hora de realizar el desarrollo del proyecto. Se trata de riesgos de proyecto, que afectan a su correcto desarrollo. Para cada uno de estos riesgos se muestra un título, una breve descripción, su probabilidad (muy baja, baja, media, alta, muy alta), su impacto (muy bajo, bajo, medio, alto, crítico), un plan de mitigación que reduzca su probabilidad y el plan de contingencia que se adoptará en caso de que se materialice. Las tablas 4.2 a 4.6 muestran esta información.

Título	Tiempo de formación insuficiente
Descripción	No se dispone de suficientes conocimientos en el uso de las herramientas o tecnologías utilizadas
Probabilidad	Media
Impacto	Medio
Medidas preventivas	Se ha reservado tiempo para la formación en la fase inicial del desarrollo
Plan de contingencia	Buscar información en la red

Tabla 4.2: Riesgo de tiempo de formación insuficiente

Título	Enfermedad o situaciones excepcionales
Descripción	Se contrae una enfermedad o una situación excepcional que incapacita al desarrollador
Probabilidad	Baja
Impacto	Medio
Medidas preventivas	La planificación del proyecto se ha llevado a cabo contando con márgenes de tiempo suficientes para afrontar estas situaciones, además se procurará mantener un buen estado de salud
Plan de contingencia	Retrasar tareas y si es necesario alargar la duración del proyecto

Tabla 4.3: Riesgo de enfermedad

Título	Cambios en los requisitos del sistema
Descripción	Los requisitos cambian, añadiendo o restando funcionalidad al sistema
Probabilidad	Media
Impacto	Medio
Medidas preventivas	La planificación se ha llevado a cabo siguiendo metodología ágil, que permite cambios en los requisitos con facilidad
Plan de contingencia	Modificar la planificación

Tabla 4.4: Riesgo de cambios en los requisitos

Título	Fallo en el cumplimiento de la planificación temporal
Descripción	Debido a las complicaciones surgidas durante el desarrollo del proyecto no es posible cumplir con la planificación establecida
Probabilidad	Alta
Impacto	Medio
Medidas preventivas	La planificación se ha llevado a cabo contando con márgenes de tiempo suficientes para afrontar estas situaciones
Plan de contingencia	Alargar la duración del proyecto

Tabla 4.5: Riesgo de fallo en la planificación

Título	Fallo o avería en el entorno de desarrollo
Descripción	Durante el desarrollo del proyecto cortes de electricidad, red o averías en la máquina
Probabilidad	Baja
Impacto	Medio
Medidas preventivas	Mantener el entorno en condiciones adecuadas
Plan de contingencia	Reparar la avería o fallo cuanto antes

Tabla 4.6: Riesgo de avería

4.3. Presupuesto

El proyecto contará con una persona como desarrollador, el coste se calculará a partir del sueldo de un desarrollador junior de unos 18.000 €brutos anuales, aproximadamente 9 euros la hora. Para la estimación inicial de 300 horas el coste total del equipo humano sería de unos 2700 €.

Para estimar el coste del hardware necesario utilizaremos el coeficiente máximo lineal. Este coeficiente representa el porcentaje límite al cual las empresas van a amortizar sus bienes. Es decir, se trata de una forma de calcular el desgaste anual de un activo con el fin de reflejar contablemente el gasto por amortización y el valor neto contable del activo en el balance.

Para el desarrollo del proyecto se ha utilizado una maquina valorada en unos 850 €, consultando la tabla de coeficientes de amortización lineal [1] podemos observar que los equipos para procesos de información cuentan con un coeficiente lineal máximo del 25 %. Por lo tanto, el coste de usar el equipo informático durante aproximadamente 4 meses es de 850*(0.25)*(4/12), lo que supone 70,83 € de amortización.

En cuanto al software, para el desarrollo se ha utilizado la herramienta *Unity*, que cuenta con licencia personal gratuita para estudiantes o individuos con ingresos inferiores 100 mil dólares. La creación de diagramas se ha realizado mediante la herramienta astah, que cuenta también con versión gratuita para estudiantes. Para la elaboración de los videotutoriales se ha utilizado la herramienta de edición de video openshot, que es gratuita y de código abierto.

Dentro del apartado multimedia, todos los recursos utilizados cuentan con licencia de uso gratuita o han sido creados con herramientas de dibujo gratuitas.

La suma total asciende a 2770,83 €, como se muestra en la tabla 4.7.

Tipo de coste	**Precio (€)**
Equipo humano	2700
Hardware	70,83
Software	0
Recursos multimedia	0
Total	2770,83

Tabla 4.7: Resumen gastos del proyecto

4.4. Seguimiento del Proyecto

Se ha realizado un seguimiento iteracion a iteracion del proyecto. Para ello, se han planificado reuniones periódicas para la correcta supervisión del proyecto. Los lunes de cada tres semanas se tomarán como final de la correspondiente iteración y comienzo del siguiente. En cada una de las iteraciones se elegirán al principio las tareas a abordar en la misma. Existirán varios tipos de tareas dependiendo de su finalidad:

- **Documentación:** tareas asociadas con la presente documentación.
- **Formación:** tareas asociadas a la formación.
- **Configuración:** tareas asociadas a la configuración o puesta en marcha de ciertos elementos necesarios para la realización del proyecto.
- **implementación:** tareas asociadas a la implementación del proyecto.
- **Bug:** tarea relativa al arreglo de algún bug o error encontrado.

Junto a cada tarea se especifica el tiempo en horas-hombre empleadas en el desarrollo de esta, Además, se especificara el estado de la tarea en concreto. Las tareas qué se encuentran incompletas pasarán a la planificación de la siguiente iteración.

4.4.1. Iteración 1

Como ya se comento anteriormente en la planificación inicial se realiza una primera iteración, con vistas a planificar la estructura del proyecto, formación, poner en marcha todas las herramientas necesarias y la documentación. Sin embargo, la formación sobre *Unity* no fue completada, debido a la disponibilidad horaria y a una mala estimación del peso de esta. En la tabla 4.8 se muestran las tareas planificadas para la primera iteración, en total se emplearon un total de 35 horas para la realización de la misma.

tarea	Tipo	Descripción	Horas	Estado
T-001	Documentación	Redacción de la introducción, contexto y objetivos	5h	completada
T-002	Documentación	Redacción del plan de proyecto	6h	completada
T-003	Formación	Formación sobre Unity	21h	Incompleta
T-004	Configuración	Instalación De Unity y puesta en marcha del entorno de desarrollo	3h	completada
Total			**35h**	

Tabla 4.8: Tareas de la iteración 1

4.4.2. Iteración 2

Esta iteración es dedicada a completar la formación sobre *Unity*, la documentación de *Unity* y el resto de las herramientas utilizadas. Además, se realiza el documento de diseño del juego *Apuntados*. En la tabla 4.9 se muestran las tareas planificadas para la primera iteración.

tarea	Tipo	Descripción	Horas	Estado
T-003	Formación	Formación sobre Unity	20h	Completada
T-005	Documentación	Redacción sobre Unity	12h	completada
T-006	Documentación	Redacción sobre el resto de las herramientas utilizadas	3h	completada
T-007	Documentación	Creación del GDD: Apuntados	10h	completada
Total			**45h**	

Tabla 4.9: Tareas de la iteración 2

4.4.3. Iteración 3

Esta segunda iteración se dedica al desarrollo del juego *Apuntados*. Se modificó el Documento de Diseño del Videojuego entregado en la anterior iteración, a petición de los tutores y se realizarón el modelo de análisis y diseño. Además, se comenzó con la implementación del videojuego. En total se han empleado 43 horas, muy cerca de lo estimado para la realización de esta iteración. Todas las tareas (mostradas en la tabla 4.10) fueron completadas dentro de la iteración, por lo que ninguna pasa a la siguiente.

tarea	Tipo	Descripción	Horas	Estado
T-008	Documentación	modificación documento de diseño del videojuego: Apuntados	7h	Completada
T-009	Documentación	Análisis: Apuntados	10h	completada
T-010	Documentación	Diseño: Apuntados	6h	completada
T-011	Implementación	Creación de menús de la aplicación	10h	completada
Total			**43h**	

Tabla 4.10: Tareas de la iteración 3

4.4.4. Iteración 4

Esta tercera iteración se dedicó a la implementación de *Apuntados*, sus pruebas y la documentación de estos procesos. Se ha sobrepasado la estimación de horas-hombre, pero se encuentra dentro de un margén aceptable. En la tabla 4.11 se puede ver la distribución de tareas de la iteración.

tarea	Tipo	Descripción	Horas	Estado
T-012	Implementación	implementación mecanismo básico de juego: Apuntados	25h	Completada
T-013	Bug	Solución al error que provocaba una fuga de memoria: Apuntados	4h	completada
T-014	Implementación	Pruebas Beta: Apuntados	4h	completada
T-015	Documentación	Pruebas: Apuntados	6h	completada
T-016	Implementación	Mejoras visuales: Apuntados	15h	Completada
Total			**48h**	

Tabla 4.11: Tareas de la iteración 4

4.4.5. Iteración 5

En esta iteración se comenzó el desarrollo de *Apilas*, empezando por el Documento de Diseño del Videojuego y continuando por el Modelo de Análisis y Diseño. Además, se integro *Apuntados* dentro de la plataforma web del proyecto. Se ha sobrepasado de nuevo la estimación horas-hombre, pero dentro de un margén aceptable. En la tabla 4.12 se puede ver la distribución de tareas de la iteración.

tarea	Tipo	Descripción	Horas	Estado
T-017	Documentación	Documento de diseño del videojuego: Apilas	8h	completada
T-018	Implementación	integración en la plataforma: Apuntados	4h	completada
T-019	Configuración	implementación mecanismo básico de juego: Apilas	15h	Completada
T-020	Documentación	Análisis: Apilas	10h	completada
T-021	Documentación	Diseño: Apilas	6h	completada
T-022	Documentación	Pruebas: Apilas	6h	completada
Total			**49h**	

Tabla 4.12: Tareas de la iteración 5

4.4.6. Iteración 6

Esta iteración fue dedicada a mejoras tanto visuales como de jugabilidad en ambos videojuegos. Además, se integro *Apilas* dentro de la plataforma web del proyecto. En la tabla 4.113 se puede ver la distribución de tareas de la iteración.

tarea	Tipo	Descripción	Horas	Estado
T-023	Implementación	Mejoras visuales: Apilas	20h	Completada
T-024	Implementación	integración en la plataforma: Apilas	3h	completada
T-025	Implementación	Cambios en la jugabilidad: Apilas	12h	completada
T-026	Implementación	Cambios en la jugabilidad: Apuntados	6h	incompleta
Total			**41h**	

Tabla 4.13: Tareas de la iteración 6

4.4.7. Iteración 7

Esta iteración fue dedicada a completar la documentación del proyecto, creación de tutoriales y arreglo final de errores. En la tabla 4.14 se puede ver la distribución de tareas de la iteración.

tarea	Tipo	Descripción	Horas	Estado
T-026	Implementación	Cambios en la jugabilidad: Apuntados	6h	incompleta
T-027	Implementación	Tutorial: Apuntados	10h	Completada
T-028	Implementación	Tutorial: Apilas	10h	Completada
T-029	Bug	Solución al error que producía un mal renderizado de ciertos objetos: Apilas	4h	Completada
T-030	Bug	Solución al error que provocaba que no se eliminarán ciertos elementos al salir de pantalla: Apuntados	4h	Completada
T-031	Documentación	Documentación sobre la conclusión	4h	Completada
Total			**38h**	

Tabla 4.14: Tareas de la iteración 7

Capítulo 5

Juego Serio 1: *Apuntados*

5.1. GDD

5.1.1. Introducción

Se trata de un videojuego didáctico sobre estructuras dinámicas en java, en concreto listas enlazadas. El objetivo es explicar mediante la gamificación el funcionamiento de las listas enlazadas y practicar el manejo de punteros. El videojuego consta de abalorios y punteros, los punteros actúan como conexiones entre los abalorios y estos representan elementos de la lista, se pueden conectar unos a otros formando collares, que simbolizan listas enlazadas.

El objetivo del juego es formar un collar determinado de abalorios a partir de unos abalorios distribuidos aleatoriamente de una forma inicial. Para cumplirlo el jugador puede crear y eliminar conexiones entre los diferentes elementos de forma análoga al funcionamiento de las listas enlazadas. El juego tiene varios niveles, cada uno de ellos centra el aprendizaje en una operación sobre listas enlazadas.

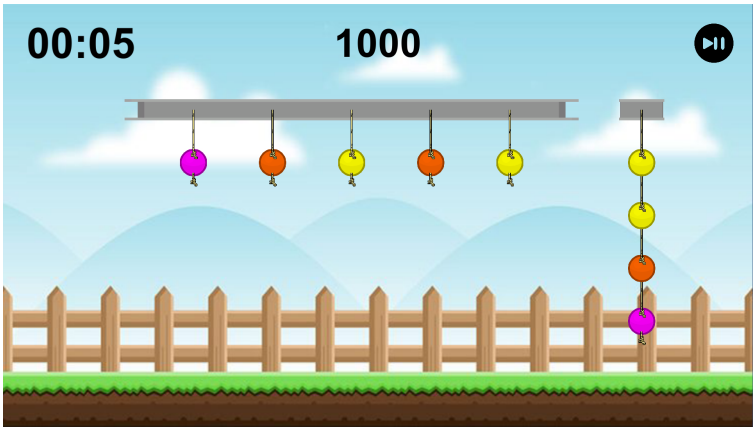

Figura 5.1: Captura de pantalla: Apuntados

En la figura 5.1 podemos ver una imagen del juego, en la parte central se encuentran los punteros en forma de cuerda y colgando de estos punteros las cuentas que representan la lista enlazada. Separado, en la parte derecha se encuentra la lista que representa la solución al nivel. En la parte superior se encuentra el marcador de puntuación, un contador de tiempo y el botón de pausa.

5.1.2. Audiencia y Plataforma

El desarrollo del juego forma parte de un proyecto de gamificación de la asignatura de Fundamentos de programación, que forma parte del primer curso de los estudios del *Grado de Ingeniería Informática* , por lo que la audiencia son los alumnos de esta asignatura. Estos alumnos por lo general se encuentran en un arco de edades de entre 17 y 25 años.

5.1.3. Mecánicas

El collar objetivo aparecerá representado de forma visible en la parte derecha de la pantalla. Todos los abalorios estarán sometidos a una fuerza similar a la gravedad, que los empujará hacia abajo haciéndolos desaparecer del para siempre.

Los punteros en cambio no se pueden mover. Para evitar que los abalorios desaparezcan del escenario deben de estar conectados a un collar conectado a un puntero. El collar solución debe encontrarse conectado a un puntero para que sea tomado como valido y el nivel sea superado.

El jugador podrá realizar las siguientes acciones:

- Crear una nueva conexión desde un abalorio a otro. Para ello el jugador deberá hacer clic izquierdo con el cursor encima del abalorio origen de la conexión y una vez seleccionado el elemento hacer clic izquierdo de nuevo sobre el abalorio destino de la conexión.

  En caso de que el usuario seleccione como destino de la conexión un elemento no valido o el mismo elemento que ha seleccionado en primera instancia se cancelará la acción. Si el primer elemento seleccionado ya tiene una conexión existente hacia otro, la conexión inicial se destruirá.

- Destruir una conexión ya existente, para ello el jugador debe situar el cursor encima de la conexión y hacer clic derecho del ratón.

  La interfaz del juego mostrará el collar objetivo en la parte derecha de la pantalla de forma que destaque del resto de elementos. En el resto de la pantalla se encontrarán los collares iniciales, con los que se podrá interactuar (Figura 5.2).

En caso de que el nivel sea irresoluble, debido a la perdida de abalorios, se mostrará un mensaje de nivel fallido y la posibilidad de repetir el nivel. Al contrario, si el nivel ha sido solucionado de manera correcta se mostrará un mensaje de nivel resuelto y la posibilidad de jugar el siguiente.

La puntuación se calculará partiendo de un valor inicial, con el paso del tiempo y por cada conexión creada o destruida por encima de una cantidad establecida se le restará una cantidad establecida por nivel.

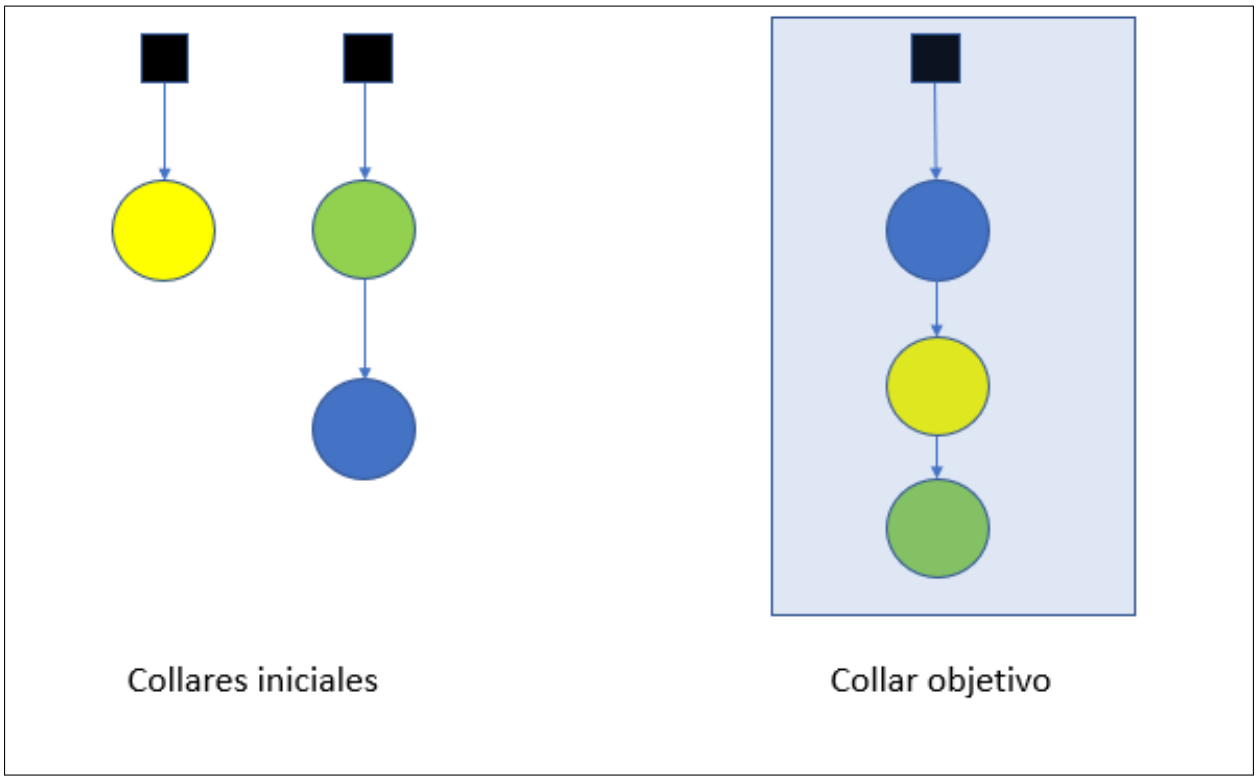

Figura 5.2: Boceto: *Apuntados*

5.1.4. Niveles

El videojuego constará de 4 niveles, cada uno haciendo hincapié en un apartado del manejo de listas enlazadas. Cada nivel contará con una serie de variantes con el fin de evitar que el jugador memorice los niveles y los repita para maximizar su puntuación.

Nivel 1

Objetivos de aprendizaje:

- Practicar el manejo de punteros.
- Comprender la estructura de lista simplemente enlazada.
- Entender la posible pérdida de información por el borrado de referencias.
- Construcción de listas simplemente enlazadas.

Objetivo del juego: Construir un collar de cuentas de colores a partir de varias cuentas. Se dispondrá de 1 puntero adicional. El collar objetivo tiene 4 cuentas de 3 colores distintos. Se construirá a partir de 5 bolas con los colores necesarios y la quinta bola de un color aleatorio, repetido o no.

Nivel 2

Objetivos de aprendizaje:

- Practicar el manejo de punteros.
- Comprender la estructura de lista simplemente enlazada.
- Entender la posible pérdida de información por el borrado de referencias.
- Construcción de listas simplemente enlazadas.

Objetivo del juego: Construir un collar de cuentas de colores a partir de otros collares. Se dispondrá de 1 puntero adicional. El collar objetivo tiene 3 cuentas de 3 colores distintos. Se construirá a partir de 3 collares (2 de longitud 4 y otro de longitud 5), cada uno de ellos con una bola de las necesarias para formar el collar objetivo y el resto del mismo color entre ellas, pero diferente a la bola necesaria de ese collar.

Nivel 3

Objetivos de aprendizaje:

- Practicar el manejo de punteros.
- Comprender la estructura de lista simplemente enlazada.
- Entender la posible pérdida de información por el borrado de referencias.
- Construcción de listas simplemente enlazadas.
- Recorrido y búsqueda de información en una lista.
- Eliminación e inserción de un elemento en una lista.

Objetivo del juego: Construir un collar de cuentas de colores a partir de otros collares sin perder los collares originales (aunque obviamente quedarán de longitud más corta). Se dispondrá de 1 puntero adicional. El collar objetivo tiene 3 cuentas de 3 colores distintos. Se construirá a partir de 3 collares (2 de longitud 4 y otro de longitud 5), cada uno de ellos contendrá una bola del color necesario y el resto de las bolas serán aleatorias.

Nivel 4

Objetivos de aprendizaje:

- Practicar el manejo de punteros.
- Comprender la estructura de lista simplemente enlazada.
- Entender la posible pérdida de información por el borrado de referencias.
- Ordenación de listas simplemente enlazadas.

Objetivo del juego: Construir un collar de cuentas de colores a partir de un único collar. Se dispondrá de 1 puntero adicional.
El collar objetivo tiene 6 cuentas de 5 colores distintos. Se construirá a partir de un collar con las mismas cuentas en un orden distinto.

5.1.5. Multimedia

La multimedia utilizada para el desarrollo del juego será la siguiente:

- **Sprites:** Utilizaremos un Sprite o imagen para cada elemento del juego, uno para los punteros, uno único para todos los abalorios, al que cambiaremos de color para representar los diferentes tipos y un último Sprite para representar las conexiones entre estos elementos.
- **Música y efectos de sonido:** Como efectos de sonido al realizar cualquier acción se reproducirá un sonido y al terminar el nivel el juego habrá un sonido de nivel completado o fallado.
- **Imágenes de fondo:** Para diferenciar los niveles existirán diferentes escenarios. Se utilizarán un total de 5 imágenes de fondo para la ambientación de los niveles una para el menú y otra por cada nivel del juego.

5.1.6. Versiones

Para el desarrollo del videojuego se van a construir las siguientes versiones jugables:

- **Versión 1.0:** Una primera versión con un solo nivel jugable, en el que funcionen las mecánicas del videojuego, sin música ni efectos de sonido.
- **Versión 2.0:** Se añadirá un nuevo nivel jugable, efectos de sonido y corrección de posibles bugs.
- **Versión 3.0:** Se añadirán el resto los niveles jugables.
- **Versión 4.0:** Ultima versión del juego con últimos cambios y sistema de puntuación.

5.2. Análisis

5.2.1. Análisis de Requisitos

La aplicación resultante tiene que satisfacer ciertos requisitos concretos al final de la misma. Estos determinarán la solución finalmente elegida para la implementación del proyecto. Se muestras en las tablas 5.1 a 5.13.

RF-01	Carga del menú principal
Descripción	Al iniciar el juego el sistema deberá cargar un menú principal, que permitirá salir del juego, cargar un nivel en concreto y visualizar un tutorial.

Tabla 5.1: RF-01: Carga del menú principal

RF-02	Iniciar nivel
Descripción	El sistema deberá permitir cargar un nivel, siempre que se haya completado el anterior.

Tabla 5.2: RF-02: Iniciar nivel

RF-03	Mostrar menú de pausa
Descripción	El sistema deberá permitir mostrar y ocultar el menú de pausa, aunque el contador de tiempo debe seguir activo.

Tabla 5.3: RF-03: Mostrar menú de pausa

RF-04	Reiniciar nivel
Descripción	El sistema deberá permitir al usuario reiniciar el nivel en juego.

Tabla 5.4: RF-04: Reiniciar nivel

RF-05	Crear punteros
Descripción	El sistema deberá permitir crear punteros entre un elemento y otro.

Tabla 5.5: RF-05: Crear punteros

RF-06	Eliminar punteros
Descripción	El sistema deberá permitir eliminar un puntero creado anteriormente.

Tabla 5.6: RF-06: Eliminar punteros

RF-07	Sistema de puntuaciones
Descripción	El sistema deberá generar una puntuación en función de la habilidad del jugador.

Tabla 5.7: RF-07: Sistema de puntuaciones

RF-08	Tutorial
Descripción	El sistema deberá contar con un tutorial, que explique las mecánicas básicas del videojuego.

Tabla 5.8: RF-08: Tutorial

RF-09	Salir del juego
Descripción	El sistema deberá permitir cerrar el juego.

Tabla 5.9: RF-09: Salir

RNF-01	Motor de desarrollo
Descripción	El El sistema deberá ser desarrollado usando el motor *Unity3D*.

Tabla 5.10: RNF-01: Motor de desarrollo

RNF-02	Lenguaje de programación
Descripción	Los scripts deberán ser desarrollados usando C#.

Tabla 5.11: RNF-02: Lenguaje de programación

RNF-03	Plataforma objetivo
Descripción	El sistema deberá estar desarrollado y adaptado para funcionar correctamente en WebGL.

Tabla 5.12: RNF-03: Plataforma objetivo

RNF-04	Comunicación con la plataforma
Descripción	El sistema deberá comunicarse con la plataforma del proyecto de forma que cargue y guarde estadísticas.

Tabla 5.13: RNF-04: Comunicación con la plataforma

5.2.2. Casos de Uso

El siguiente diagrama presenta los diferentes casos de uso que pueda realizar un usuario al utilizar nuestra aplicación. El jugador será el único actor del sistema.
Hemos dividido los casos de uso en dos escenarios, el jugador se encuentra en el menu principal (Figura 5.3) o el jugador se encuentra jugando un nivel (Figura 5.4).

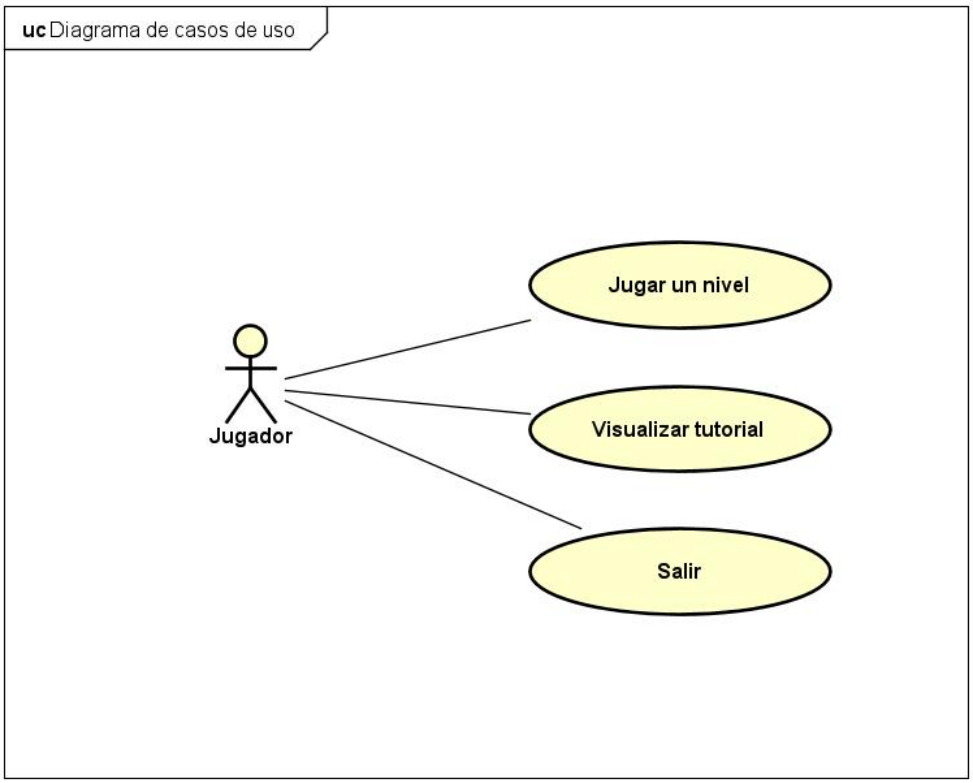

Figura 5.3: Diagrama de casos de uso menú principal: *Apuntados*

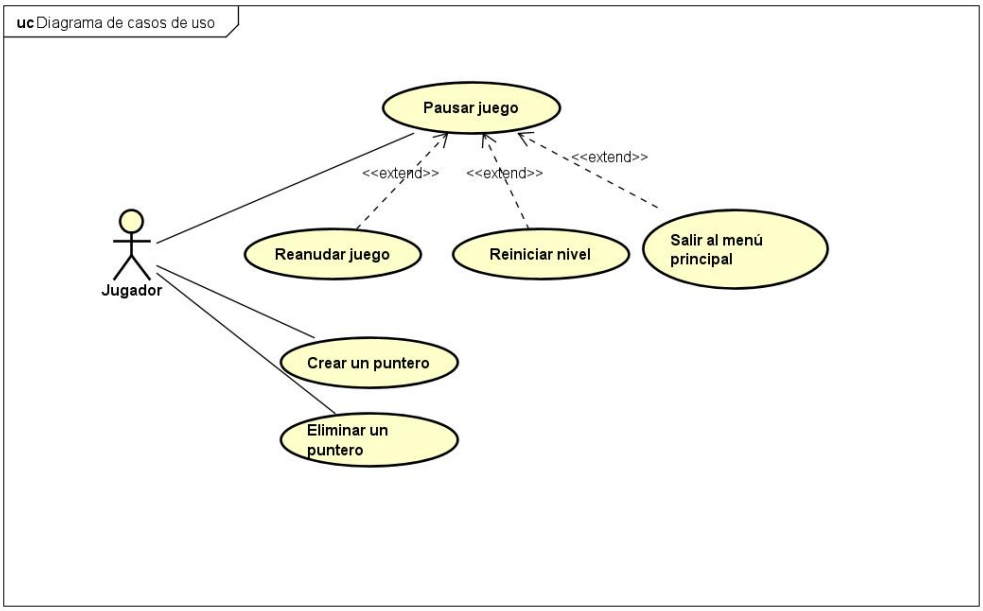

Figura 5.4: Diagrama de casos de uso nivel: *Apuntados*

5.3. Diseño

Como explicamos en el capitulo 3, *Unity* nos brinda una librería de clases, en la que podemos encontrar recursos para el desarrollo de videojuegos, lo que nos lleva a tener que aplicar sus clases a nuestra solución. Por esta razón los diagramas de diseño no se realizarán sobre todas las clases, si no sobre los scripts, es decir, las clases descendientes de *MonoBehaviour*. En la Figura 5.5 se muestra el modelo de dominio.

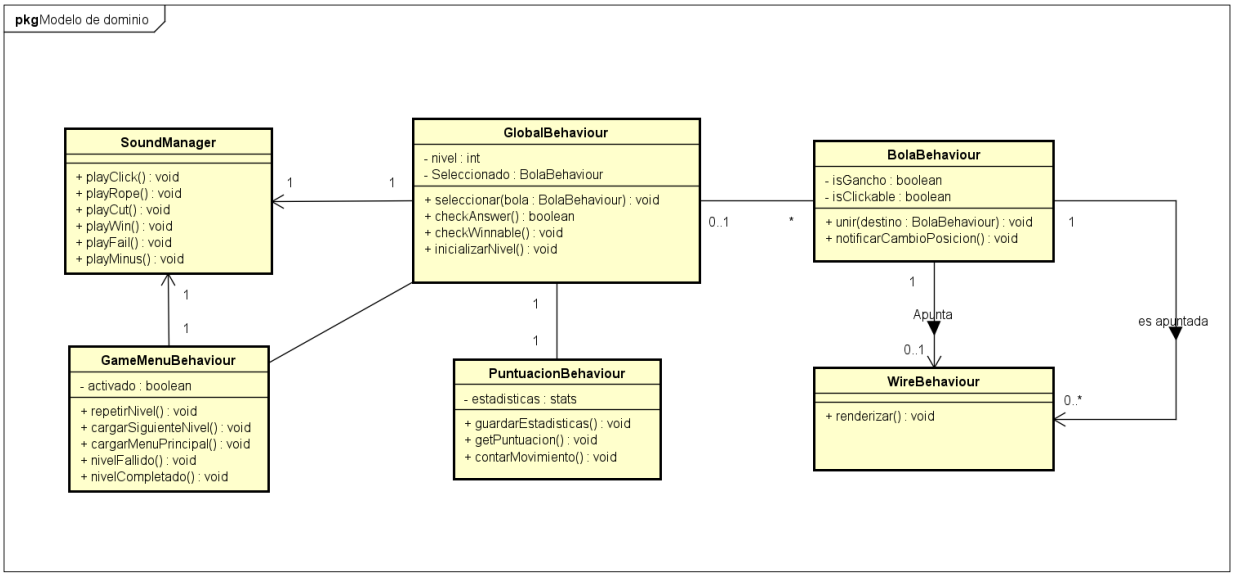

Figura 5.5: Modelo de dominio: *Apuntados*

En esta solución se han decidido aplicar los siguientes patrones de diseño:

- **Patrón observador:** En nuestro caso los abalorios o bolas que se muestrán en pantalla se encuentran sometidos a una fuerza que simula la gravedad, de forma que cambián de posición a partir de componentes propios de *Unity* que se encargan de gestionar las físicas de estos elementos.

  Una posible solución sería utilizar la función update dentro de cada puntero para renderizar la imagen del mismo en la posición correcta. Esto supondría un gran coste de recursos. Para intentar reducir este coste de recursos hemos aplicado el patrón observador (Figura 5.6), de forma que sea cada bola individualmente la que mediante la función update compruebe si su posición ha cambiado respecto el fotograma anterior y notifique a los puntero que le corresponde para que vuelvan a renderizar el puntero.

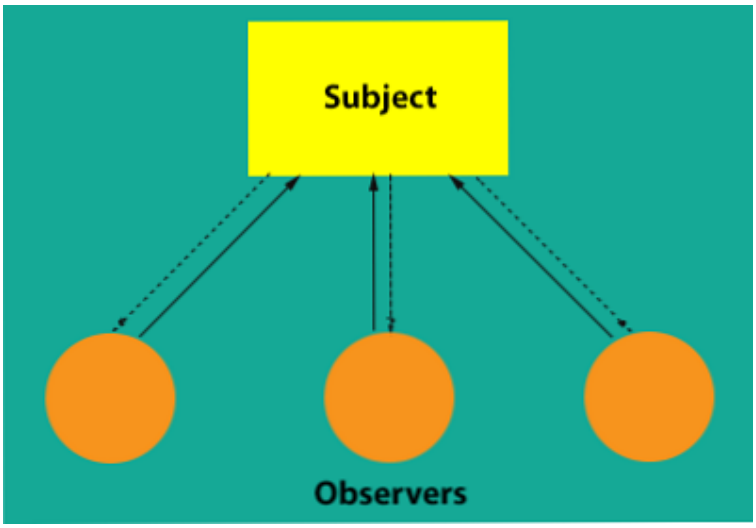

Figura 5.6: Patrón observador

- **Patrón Fachada:** La motivación del patrón Fachada (Figura 5.7) es la necesidad de estructurar los desarrollos y reducir su complejidad, haciendo que las comunicaciones entre jerarquías o paquetes sean las mínimas posibles y que éstas estén centralizadas.
  En nuestro caso hemos utilizado este patrón para la reproducción de sonidos, *SoundManager* es la clase que se encarga de realizar todas las tareas necesarias para reproducir los sonidos que el juego necesite.

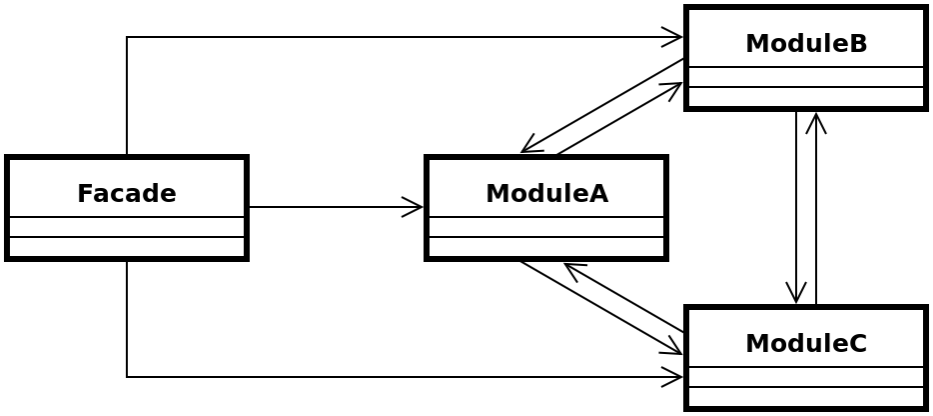

Figura 5.7: Patrón Fachada

5.4. Pruebas

En esta sección expondremos las pruebas realizadas en las diferentes tareas de implementación de nuestro proyecto.

5.4.1. Pruebas Unitarias

El objetivo de estas pruebas es tratar de dar con posibles errores en nuestra aplicación realizando dichas pruebas directamente sobre ella. Esto nos servirá para demostrar su correcta funcionalidad, en caso de que no haya errores. En la tablas 5.14 a 5.24 se muestran las pruebas realizadas y como al final todas han funcionado correctamente.

PU-01	visualizar videotutorial
Descripción	Al pulsar el botón tutorial del menú principal el juego deberá cambiar de escena y mostrar el tutorial.
Resultado	Correcto.

Tabla 5.14: PU-01: visualizar videotutorial

PU-02	Generar niveles
Descripción	Al seleccionar un nivel, se genera un nivel resoluble, aleatorio, dentro de los parámetros establecidos.
Resultado	Correcto.

Tabla 5.15: PU-02: Generar niveles

PU-03	Completar nivel
Descripción	Al cumplir la condición de victoria el nivel se marca como completado y se realizan las acciones correspondientes.
Resultado	Correcto.

Tabla 5.16: PU-03: Completar nivel

PU-04	Fallar nivel - Tiempo agotado
Descripción	Al agotar la puntuación por una penalización de tiempo el nivel se marca como fallado.
Resultado	Correcto.

Tabla 5.17: PU-04: Fallar nivel - Tiempo agotado

PU-05	Fallar nivel - Movimientos agotados
Descripción	Al agotar la puntuación por una penalización de movimientos el nivel se marca como fallado.
Resultado	Correcto.

Tabla 5.18: PU-05: Fallar nivel - Movimientos agotados

PU-06	Reiniciar nivel
Descripción	Al utilizar la opción de reiniciar nivel el nivel se genera de nuevo y las puntuaciones se reinician.
Resultado	Correcto.

Tabla 5.19: PU-06: Reiniciar nivel

PU-07	Salir del juego
Descripción	El juego se cierra correctamente.
Resultado	Correcto.

Tabla 5.20: PU-07: Salir del juego

PU-08	Crear puntero
Descripción	Al hacer click izquierdo en una bola y luego en otra diferente se crea un puntero y en caso de existir uno previamente este se elimina y se crea uno nuevo.
Resultado	Correcto.

Tabla 5.21: PU-08: Crear puntero

PU-09	Eliminar puntero
Descripción	Al hacer click derecho sobre un puntero este se elimina.
Resultado	Correcto.

Tabla 5.22: PU-09: Eliminar puntero

PU-10	Reducir puntuación
Descripción	Al utilizar demasiados movimientos o tiempo la puntuación se reduce.
Resultado	Correcto.

Tabla 5.23: PU-10: Reducir puntuación

PU-11	Perdida de elementos
Descripción	Al salir un elemento fuera de pantalla el nivel se marca como fallado.
Resultado	Correcto.

Tabla 5.24: PU-11: Perdida de elementos

5.4.2. Pruebas Beta

En este caso como se contaba con tiempo y la posibilidad de reunir alumnos de otros cursos para probar el juego se decidió realizar pruebas Beta.

Dentro de las instalaciones de la escuela se realizaron pruebas con alrededor de 10 alumnos, todos ellos habían cursado y superado la asignatura que pretendemos gamificar y conocían los conceptos que intentan explicar los videojuegos.

Como parte de la prueba, antes de dejar a los alumnos probar el videojuego, se les explico el concepto que se trataba de explicar con él, las mecánicas, condiciones de victoria... Después de esto se les dejo probar el videojuego durante unos 10 minutos, todos ellos en maquinas independientes, con una persona encargada de guiar, en el transcurso del videojuego, a los jugadores aclarándoles todas las dudas que pudieran tener sobre la jugabilidad.

Una vez terminado el tiempo se realizo una encuesta a los alumnos, en la cual se preguntaba sobre cuantos niveles habían logrado completar y su puntuación, sus sensaciones al jugarlo, posibles mejoras y si creían que el videojuego ayudaba a comprender el concepto que trata de explicar el videojuego.

Tras la recopilación de las encuestas, se interpretaron los datos obtenidos durante este proceso. Estas pruebas no mostraron ningún error o fallo en los juegos, pero si para realizar cambios que ayudaron a mejorar las mecánicas, apariencia visual, y refinar el concepto del videojuego. En general a los alumnos les gusto el videojuego, consideraron que ayudaba a entender los conceptos y creían que podría ser util como material didáctico.

A partir de la retroalimentación se realizaron los siguientes cambios:

- Cambios gráficos en los punteros.
- Se reestructuraron los niveles, para una mejor sensación de progreso.

Capítulo 6

Juego Serio 2: *Apilas*

Para el desarrollo de este segundo videojuego se han intentado reutilizar la mayor cantidad de soluciones y recursos utilizados en Apuntados y descritos en el capítulo anterior, para que su desarrollo fuera más rápido. Lo importante, recordemos, es que el juego cumpla su objetivo de aprendizaje.

6.1. GDD

6.1.1. Introducción

Se trata de un videojuego didáctico sobre listas enlazadas. El objetivo del proyecto es explicar mediante la gamificación el funcionamiento de diferentes tipos de listas enlazadas, en concreto se representará el funcionamiento de pilas y colas. El videojuego consta de recipientes y bolas dentro de estos recipientes, los recipientes representan estructuras, ya sea pilas o colas y las bolas representan elementos contenidos dentro de las mismas.

El objetivo del juego es organizar las bolas por colores, de forma que en una estructura se encuentren todas las bolas del mismo color. Para cumplir este objetivo el jugador puede mover las bolas que se encuentran en el interior de los recipientes. Los recipientes imitan el comportamiento que tienen las estructuras que representan. El juego cuenta con varios niveles, los dos primeros muestrán el comportamiento de un tipo de estructura aislada y los dos últimos el comportamiento de los diferentes tipos de estructuras entre ellas.

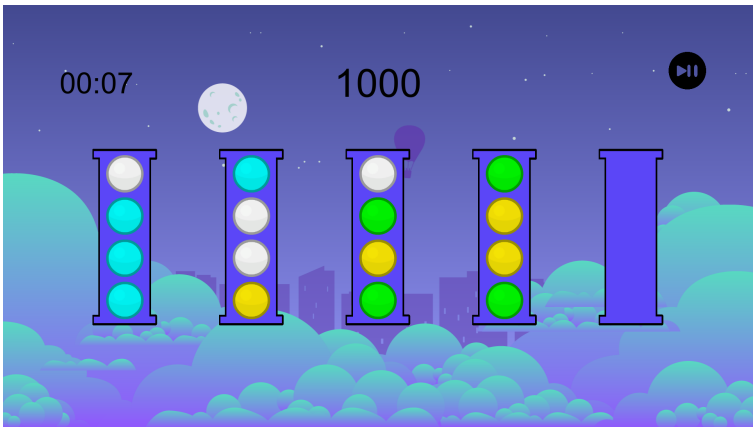

Figura 6.1: Captura de pantalla: Apilas

En la figura 6.1 podemos ver una imagen del juego, en la parte central se encuentran las estructuras que componen el nivel y dentro de esas estructuras las bolas. En la parte superior se encuentra el marcador de puntuación, un contador de tiempo y el botón de pausa.

Este juego está inspirado en algunos juegos de puzzles de ordenar bolas [2].

6.1.2. Audiencia y Plataforma

El desarrollo del juego forma parte de un proyecto de gamificación de la asignatura de Fundamentos de programación, que forma parte del primer curso de los estudios del *Grado de Ingeniería Informática*, por lo que la audiencia son los alumnos de esta asignatura. Estos alumnos por lo general se encuentran en un arco de edades de entre 17 y 25 años.

6.1.3. Mecánicas

Cada nivel estará formado por un conjunto de bolas, en total cada nivel contará con cuatro bolas de cuatro colores diferentes, distribuidos a lo largo de cinco estructuras, cada una de ellas con capacidad para cuatro bolas. Cuatro de estas cinco estructuras se encontrarán inicialmente llenas de bolas y la otra se encontrará vacía.

Para conseguir ordenar las bolas por colores el jugador puede mover las bolas entre los diferentes recipientes. Para ello hará click sobre la estructura de la que desea sacar la bola, una vez seleccionado el primer recipiente el jugador puede realizar click izquierdo sobre el recipiente destino del movimiento, desencadenando el movimiento. Una vez seleccionado el primer recipiente se mostrará resaltada sobre el resto la bola que se moverá al seleccionar el recipiente destino. No podemos seleccionar la bola que va a salir del recipiente, solamente los recipientes entre los que se va a producir el intercambio de bolas.

La parte inferior de los recipientes representa la posición 0 de la estructura. Por lo que el comportamiento de las listas es el siguiente:

- Pila: Las pilas se representarán por una probeta con un borde en la parte superior. Las bolas entran y salen por la parte superior y se agrupan en la parte inferior. Representa una estructura LIFO (Last in, First out) por el cual la última bola introducida en el recipiente será la primera en salir del mismo.
- Cola: Las colas se representarán por una probeta con bordes en la parte superior e inferior. Las bolas entran por la parte superior, salen y se agrupan en la parte inferior. Representa una estructura FIFO (First in, First out) por el cual la primera bola introducida en el recipiente será la primera en salir del mismo.

Los movimientos entre recipientes solo se realizarán si existe el espacio suficiente. Si se trata de una cola seleccionar el mismo recipiente como origen y destino del movimiento producirá que el elemento de la posición inferior se situé en la posición superior, en el caso de las pilas no producirá ningún resultado. Para cancelar un movimiento una vez seleccionada la bola que se quiere mover se puede cancelar haciendo click en cualquier posición que no forme parte de ninguna estructura.

La puntuación se calculará partiendo de un valor inicial, con el paso del tiempo y por cada movimiento realizado por encima de una cantidad establecida se le restará una cantidad establecida por nivel. En caso de que la puntuación se encuentre por debajo de cero el nivel se considerará como fallido.

6.1.4. Niveles

El videojuego constará de 4 niveles, los dos primeros muestrán el comportamiento de un tipo de estructura aislada y los dos últimos el comportamiento de los diferentes tipos de estructura entre ellos.

- El objetivo de aprendizaje de este nivel es comprender como se intercambian información entre diferentes Colas. Todas las estructuras del nivel son colas. En la figura 6.2 se puede observar la distribución inicial del nivel.

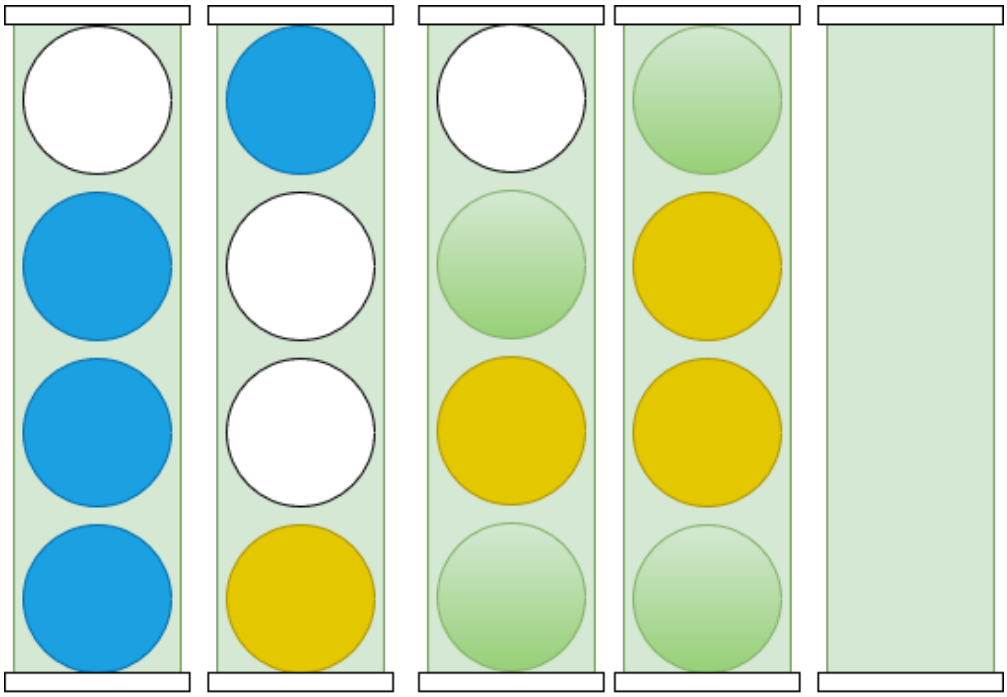

Figura 6.2: Distribución inicial nivel 1

- El objetivo de aprendizaje de este nivel es comprender como se intercambian información entre diferentes Pilas. Todas las estructuras del nivel son colas. En la figura 6.3 se puede observar la distribución inicial del nivel.

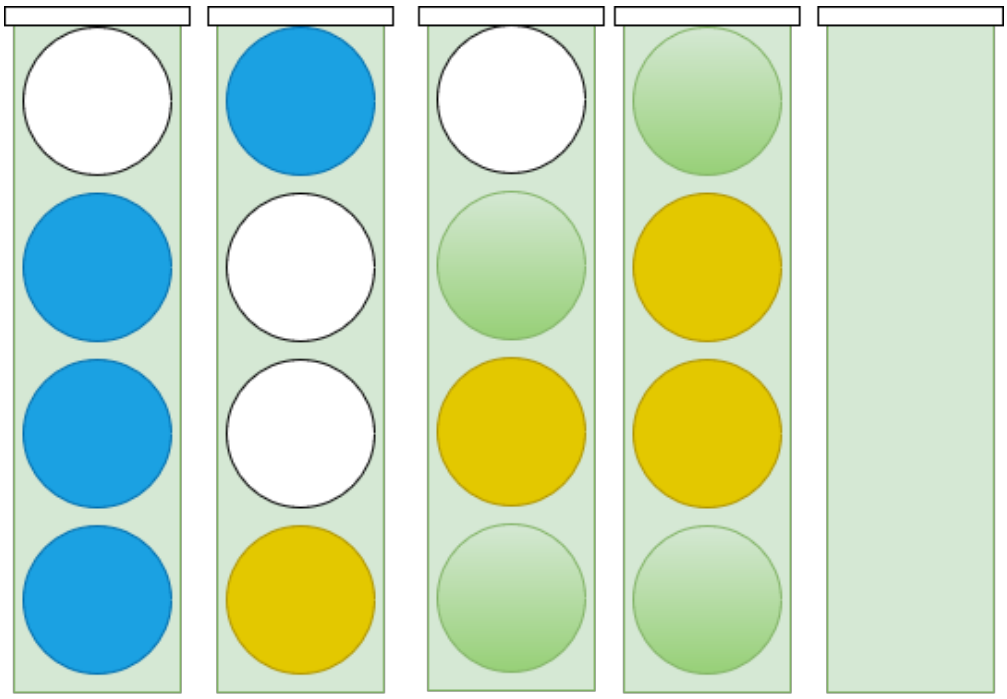

Figura 6.3: Distribución inicial nivel 2

- El objetivo de aprendizaje de este nivel es comprender como se intercambian información las Pilas y colas entre ellas. Este nivel esta formado tanto por pilas como por colas. En la figura 6.4 se puede observar la distribución inicial del nivel.

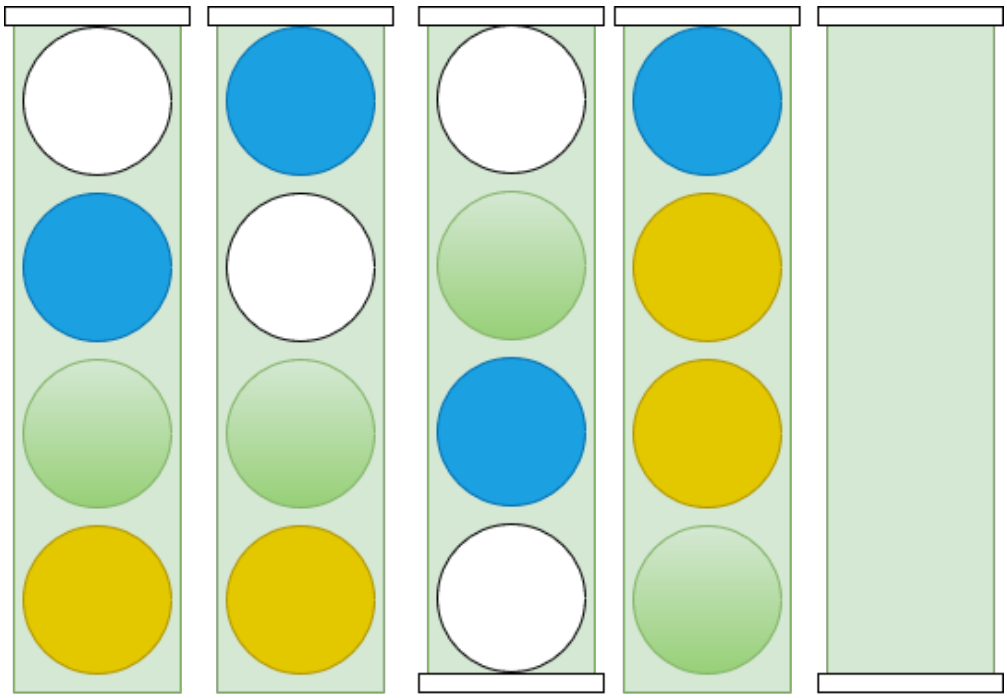

Figura 6.4: Distribución inicial nivel 3

- El objetivo de aprendizaje de este nivel es el mismo que el anterior. Se encuentra formado tanto por pilas como por colas, su única diferencia es la dificultad del mismo. En la figura 6.5 se puede observar la distribución inicial del nivel.

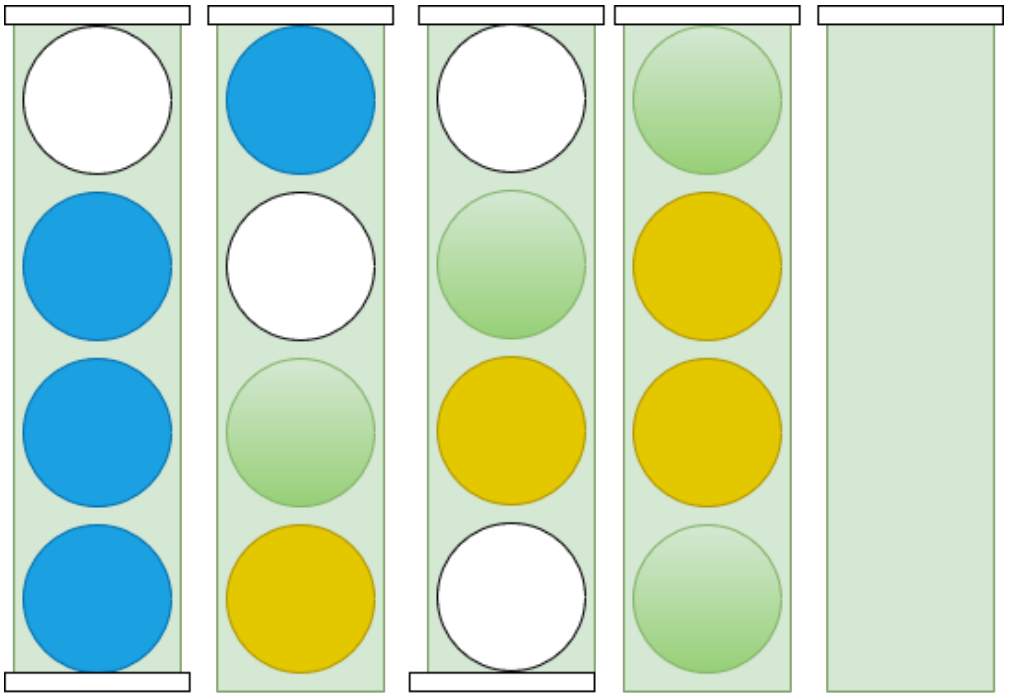

Figura 6.5: Distribución inicial nivel 4

6.1.5. Multimedia

La multimedia utilizada para el desarrollo del juego será la siguiente:

- **Sprites:** Utilizaremos un sprite o imagen para cada elemento del juego, uno para las bolas y otro para cada tipo de recipiente.
- **Música y efectos de sonido:** Como efectos de sonido al realizar cualquier acción se reproducirá un sonido y al terminar el nivel el juego habrá un sonido de nivel completado o fallado.
- **Imágenes de fondo:** Para diferenciar los niveles existirán diferentes escenarios. Se utilizarán un total de 5 imágenes de fondo para la ambientación de los niveles una para el menú y otra por cada nivel del juego.

6.1.6. Versiones

Para el desarrollo del videojuego se van a construir las siguientes versiones jugables:

- **Versión 1.0:** Una primera versión con un solo nivel jugable, en el que funcionen las mecánicas del videojuego, sin música ni efectos de sonido.
- **Versión 2.0:** Se añadirá un nuevo nivel jugable, efectos de sonido y corrección de posibles bugs.
- **Versión 3.0:** Se añadirán el resto los niveles jugables.
- **Versión 4.0:** Ultima versión del juego con últimos cambios y sistema de puntuación.

6.2. Análisis

6.2.1. Análisis de Requisitos

La aplicación resultante tiene que satisfacer ciertos requisitos concretos al final de la misma. Estos determinarán la solución finalmente elegida para la implementación del proyecto. Se muestras en las tablas 6.1 a 6.12.

RF-01	Carga del menú principal
Descripción	Al iniciar el juego el sistema deberá cargar un menú principal, que permitirá salir del juego, cargar un nivel en concreto y visualizar un tutorial.

Tabla 6.1: RF-01: Carga del menú principal

RF-02	Iniciar nivel
Descripción	El sistema deberá permitir cargar un nivel, siempre que se haya completado el anterior.

Tabla 6.2: RF-02: Iniciar nivel

RF-03	Mostrar menú de pausa
Descripción	El sistema deberá permitir mostrar y ocultar el menú de pausa, aunque el contador de tiempo debe seguir activo.

Tabla 6.3: RF-03: Mostrar menú de pausa

RF-04	Reiniciar nivel
Descripción	El sistema deberá permitir al usuario reiniciar el nivel en juego.

Tabla 6.4: RF-04: Reiniciar nivel

RF-05	Mover bolas
Descripción	El sistema deberá permitir mover bolas entre recipientes.

Tabla 6.5: RF-05: Mover bolas

RF-06	Salir del juego
Descripción	El sistema deberá permitir cerrar el juego.

Tabla 6.6: RF-06: Salir

RF-07	Sistema de puntuaciones
Descripción	El sistema deberá generar una puntuación en función de la habilidad del jugador.

Tabla 6.7: RF-07: Sistema de puntuaciones

RF-08	Tutorial
Descripción	El sistema deberá contar con un tutorial, que explique las mecánicas básicas del videojuego.

Tabla 6.8: RF-08: Tutorial

RNF-01	Motor de desarrollo
Descripción	El El sistema deberá ser desarrollado usando el motor *Unity3D*.

Tabla 6.9: RNF-01: Motor de desarrollo

RNF-02	Lenguaje de programación
Descripción	Los scripts deberán ser desarrollados usando C#.

Tabla 6.10: RNF-02: Lenguaje de programación

RNF-03	Plataforma objetivo
Descripción	El sistema deberá estar desarrollado y adaptado para funcionar correctamente en WebGL.

Tabla 6.11: RNF-03: Plataforma objetivo

RNF-04	Comunicación con la plataforma
Descripción	El sistema deberá comunicarse con la plataforma del proyecto de forma que cargue y guarde estadísticas.

Tabla 6.12: RNF-04: Comunicación con la plataforma

6.2.2. Casos de Uso

El siguiente diagrama presenta los diferentes casos de uso que pueda realizar un usuario al utilizar nuestra aplicación. El jugador será el único actor del sistema.
Hemos dividido los casos de uso en dos escenarios, el jugador se encuentra en el menu principal (Figura 6.6) o el jugador se encuentra jugando un nivel (Figura 6.7).

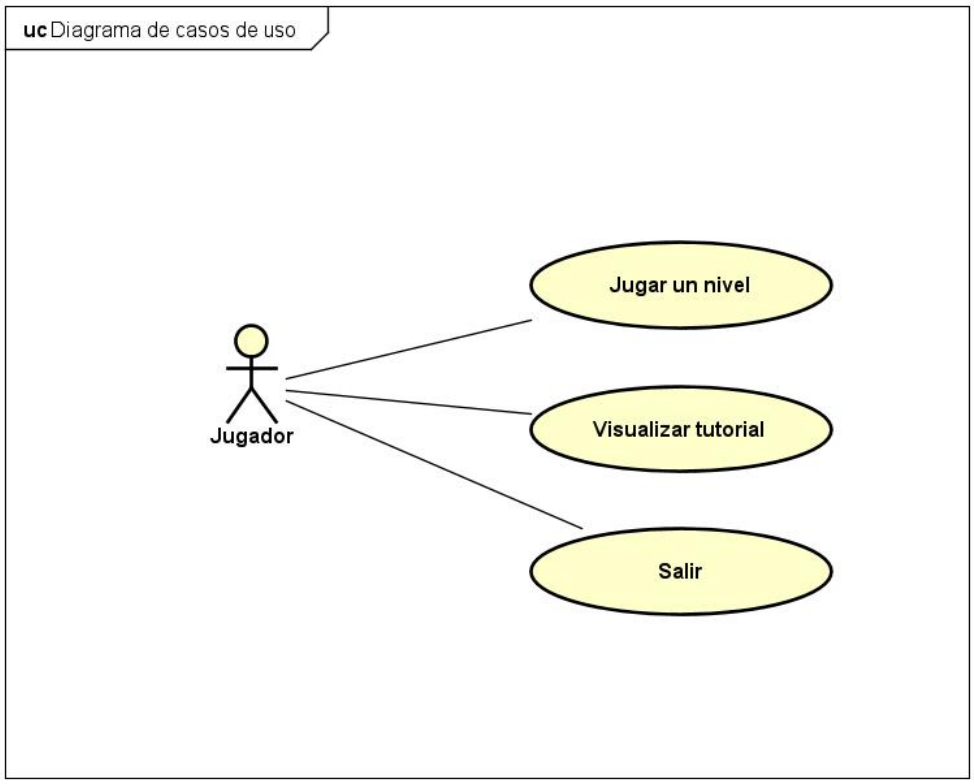

Figura 6.6: Diagrama de casos de uso menú principal: *Apilas*

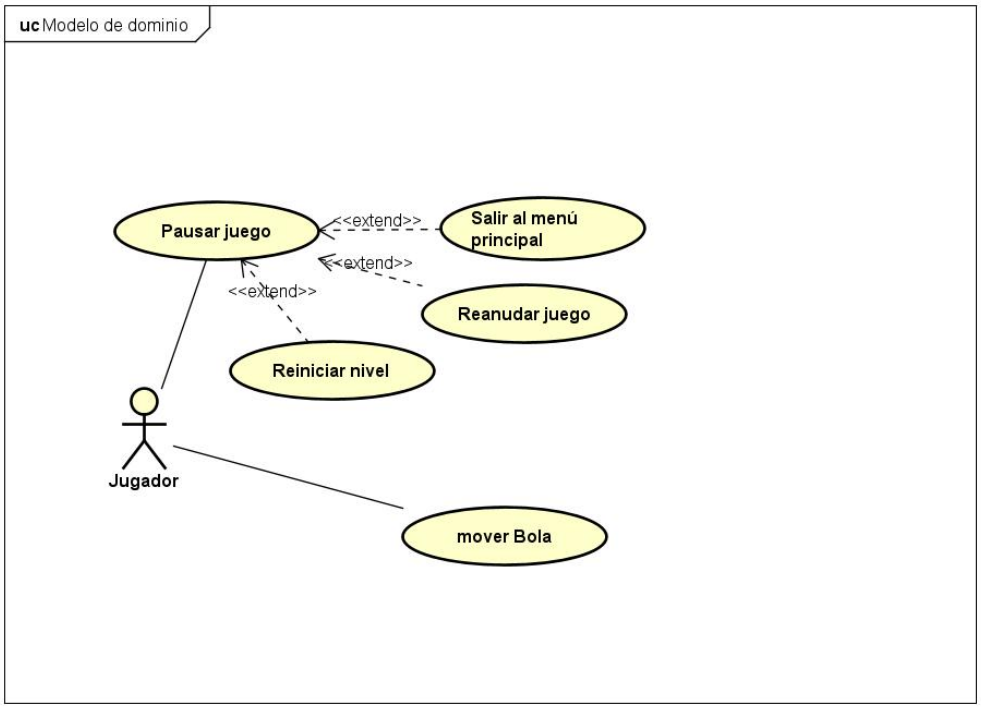

Figura 6.7: Diagrama de casos de uso nivel: *Apilas*

6.3. Diseño

como se indicó en el juego anterior, aquí solo se muestran las clases de nuestra solución (Tabla 6.8), es decir los *Scripts*.

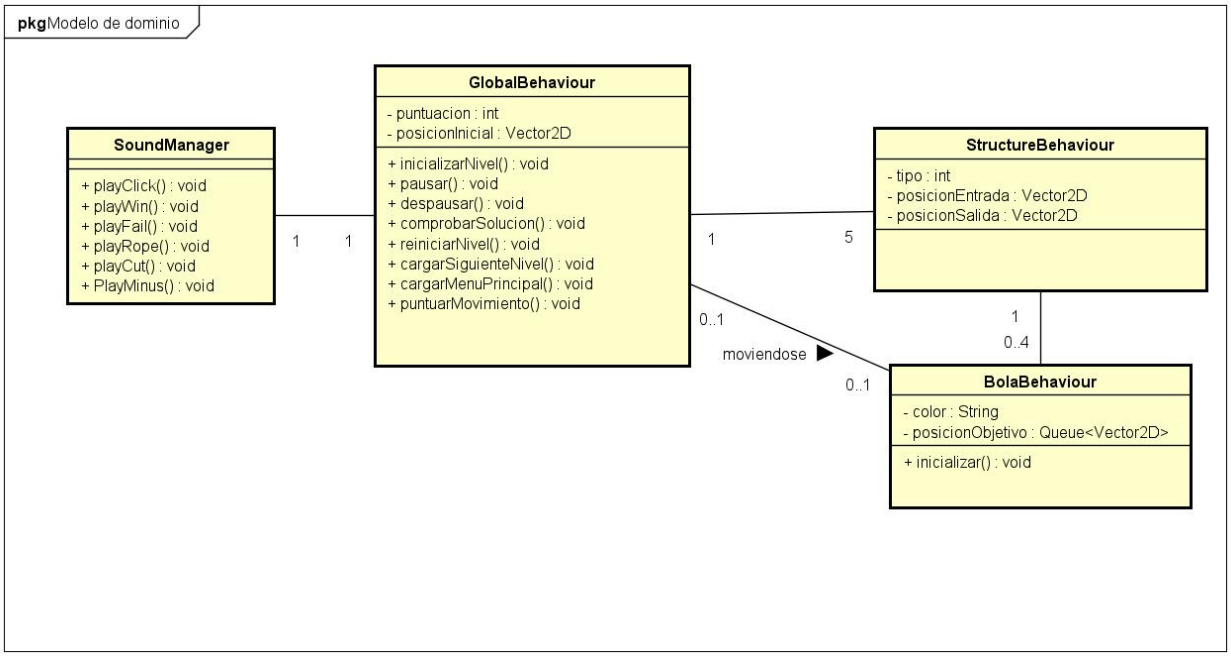

Figura 6.8: Modelo de dominio: *Apilas*

En esta solución se han decidido aplicar el patrón fachada, de la misma forma que se utilizó en el capitulo anterior.

6.4. Pruebas

En esta sección expondremos las pruebas realizadas en las diferentes tareas de implementación de nuestro proyecto. Por escasez de tiempo, en esta ocasión, no se han podido realizar *Pruebas Beta*.

6.4.1. Pruebas Unitarias

Las tablas 6.13 a 6.23 muestrán las pruebas realizadas sobre el minijuego.

PU-01	visualizar videotutorial
Descripción	Al pulsar el botón tutorial del menú principal el juego deberá cambiar de escena y mostrar el tutorial.
Resultado	Correcto.

Tabla 6.13: PU-01: visualizar videotutorial

PU-02	Generar niveles
Descripción	Al seleccionar un nivel, se genera un nivel de forma correcta.
Resultado	Correcto.

Tabla 6.14: PU-02: Generar niveles

PU-03	Completar nivel
Descripción	Al cumplir la condición de victoria el nivel se marca como completado y se realizan las acciones correspondientes.
Resultado	Correcto.

Tabla 6.15: PU-03: Completar nivel

PU-04	Fallar nivel - Tiempo agotado
Descripción	Al agotar la puntuación por una penalización de tiempo el nivel se marca como fallado.
Resultado	Correcto.

Tabla 6.16: PU-04: Fallar nivel - Tiempo agotado

PU-05	Fallar nivel - Movimientos agotados
Descripción	Al agotar la puntuación por una penalización de movimientos el nivel se marca como fallado.
Resultado	Correcto.

Tabla 6.17: PU-05: Fallar nivel - Movimientos agotados

PU-06	Reiniciar nivel
Descripción	Al utilizar la opción de reiniciar nivel el nivel se genera de nuevo y las puntuaciones se reinician.
Resultado	Correcto.

Tabla 6.18: PU-06: Reiniciar nivel

PU-07	Salir del juego
Descripción	El juego se cierra correctamente.
Resultado	Correcto.

Tabla 6.19: PU-07: Salir del juego

PU-08	Mover Bola
Descripción	Al hacer click izquierdo en un recipiente y luego en otro diferente la bola se cambia de recipiente.
Resultado	Correcto.

Tabla 6.20: PU-09: Mover bola

PU-09	Ciclar bola
Descripción	Al hacer click izquierdo sobre un recipiente de tipo cola y volver a hacer click izquierdo de nuevo sobre la misma estructura la bola se mueve desde la parte de abajo a la parte de arriba.
Resultado	Correcto.

Tabla 6.21: PU-09: Ciclar bola

PU-09	Cancelar movimiento
Descripción	Al hacer click derecho sobre un recipiente y después hacer click izquierdo o derecho sobre una superficie que no sea un recipiente se cancela la acción inicial.
Resultado	Correcto.

Tabla 6.22: PU-09: Cancelar movimiento

PU-10	Reducir puntuación
Descripción	Al utilizar demasiados movimientos o tiempo la puntuación se reduce.
Resultado	Correcto.

Tabla 6.23: PU-10: Reducir puntuación

Capítulo 7

Integración con la Plataforma

Este capítulo de la memoria se desarrolla no solamente como parte de esta, sino también como guía de integración en la plataforma para futuros proyectos que lo requieran.

Una vez desarrollados los minijuegos se realizó una labor de integración con la plataforma, se trata de una plataforma web desarrollada en un proyecto anterior [3], que se encarga de centralizar el acceso a los minijuegos, de esta forma se puede acceder a todos ellos de forma rápida y uniforme. La plataforma se encarga de gestionar el control de acceso de los usuarios con distintos roles de entrada, guardara información de todas las actividades realizadas por los usuarios (incluida su puntuación), permitirá subir, cargar, alojar y ejecutar minijuegos.

Cabe destacar que la plataforma está basada en WebGL [11] y solamente admite juegos en ese estándar. Por lo tanto cualquier juego que queramos añadir tiene que estar exportado a ese formato, *Unity* cuenta con un plugin que permite hacerlo. Para construir el proyecto solo debemos acceder a la opción archivo y opciones de Construcción, una vez dentro podemos ver todas la gran variedad de opciones y formatos para los que podemos construir el proyecto. Seleccionamos, la plataforma WebGL (Figura 7.1) y nos pedirá que instalemos un plugin. Una vez instalado el plugin ya podemos construir el proyecto.

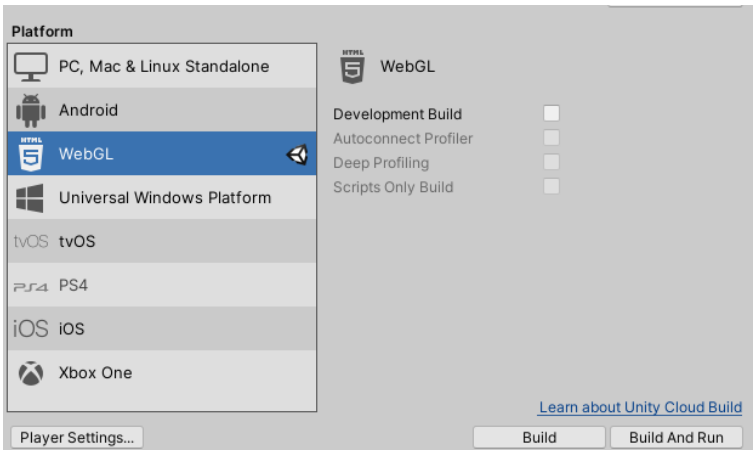

Figura 7.1: Construcción del proyecto como WebGL

Antes de construir el proyecto, debemos crear el minijuego dentro de la plataforma y conseguir el ID asociado al minijuego. Es importante que a la hora de construir el proyecto seleccionemos como nombre de carpeta el identificador que nos ha proporcionado la plataforma. Una vez construido el proyecto podemos subirlo a la plataforma y estará listo para jugar, aunque permanecerá invisible para los usuarios no desarrolladores, hasta que el administrador lo modifique. Inicialmente solo está disponible para que lo prueba el desarrollador y/o administrador, para comprobar que todo está correcto, antes de hacerlo disponible a los alumnos.

Todas las comunicaciones que se llevan a cabo con la plataforma se realizarán a través de llamadas externas, están obsoletas en Unity, pero es la única forma conocida de hacerlo. Para evitar las advertencias que lanza el entorno de desarrollo podemos añadir las anotaciones "pragma warning", como se muestra en los siguientes apartados. Por suerte, Unity mantiene una alta compatibilidad con versiones pasadas, por lo que no se prevé que esta opción sea eliminada en el futuro.

Podemos realizar estas llamadas cuando lo requiera el minijuego, desde cualquier *GameObject*, aunque es recomendable realizar estas llamadas siempre desde un GameObject que encargue de gestionarlas.

7.1. Finalizar el Juego

Al cerrar la aplicación porque se haya finalizado de jugar, tenemos que devolver el control a la plataforma. Si no lo hacemos el juego se cerrará y el jugador no podrá volver al menú de selección de minijuego.
Para devolver el control a la plataforma basta con que añadamos la llamada externa *salir* (Figura 7.2), en el momento en el que el jugador cierre el juego, por ejemplo, pulsando el botón de salir.

```
1  #pragma warning disable CS0618
2  Application.ExternalCall("salir");
3  #pragma warning restore CS0618
4  Application.Quit();
```

Figura 7.2: Llamada a salir

7.2. Carga y Actualización de Puntuaciones

La plataforma se encarga de gestionar las puntuaciones por minijuego y jugador. Desde el minijuego podemos modificar tres campos. Dos de ellos se utilizan para mostrar en el menú de selección de minijuego si este está completado o no y su puntuación, respectivamente. El tercero es un campo libre, que permite almacenar cadenas de caracteres y utilizarlas para guardar información especifica de nuestro minijuego.

Para enviar los datos a la plataforma, simplemente tenemos que añadir una llamada externa. La llamada de tipo *guardar* nos permite enviar un array como primer elemento la puntuación total obtenida en el videojuego y como segundo elemento un entero, 0 o 1, que indica si el nivel ha sido completado o no. En la Figura 7.3 se muestra un ejemplo de cómo hacerlo.

```
1    int superado = 0;
2    if (puntuacionTotal >= 4000) superado = 1;
3    int[] datos = { puntuacionTotal, superado };
4    #pragma warning disable CS0618
5    Application.ExternalCall("guardar", datos);
6    #pragma warning restore CS0618
```

Figura 7.3: Llamada a guardar

Podemos utilizar esta llamada una vez el jugador haya completado un nivel para actualizar las puntuaciones si es necesario.

También podemos dar un valor al campo libre y guardar cualquier cosa que queramos. Ahora el tipo de llamada externa debe ser *setCampoLibre*. En la figura 7.4 se muestra como se guarda un objeto con un array de enteros, que representa las puntuaciones máximas por nivel que ha obtenido el jugador. Para convertir este objeto en una cadena de texto y a la inversa hemos utilizado la clase JsonUtility.

```
1    string jsonStats = JsonUtility.ToJson(stats);
2    #pragma warning disable CS0618
3    Application.ExternalCall("setCampoLibre", jsonStats);
4    #pragma warning restore CS0618
```

Figura 7.4: Llamada a setCampoLibre

Al igual que la llamada *guardar* podemos utilizar esta llamada una vez el jugador haya completado un nivel para actualizar el campo libre con nuevos valores dependiendo de la puntuación obtenida.

Además de enviar datos también podemos solicitar a la plataforma el valor del campo libre, que hemos guardado anteriormente. Para realizar la petición de los datos a la plataforma deberemos añadir la llamada que se muestra en la Figura 7.5. Esta es una llamada asíncrona, por lo que para acceder a la información devuelta habrá que proceder como se indica a continuación.

```
1    #pragma warning disable CS0618
2    Application.ExternalCall("getCampoLibre");
3    #pragma warning restore CS0618
4
```

Figura 7.5: Llamada a getCampoLibre

Como la llamada es asíncrona, no sabemos el momento en el que la plataforma nos ha enviado la información que queremos. Una vez nuestro juego la haya recibido se ejecutará automáticamente una función en un *GameObject* en concreto. Debemos añadir a la escena este *GameObject*, que debe llamarse *OpenField* (Figura 7.6) y debe tener como componente un script llamado *Open Field* (Figura 7.7).

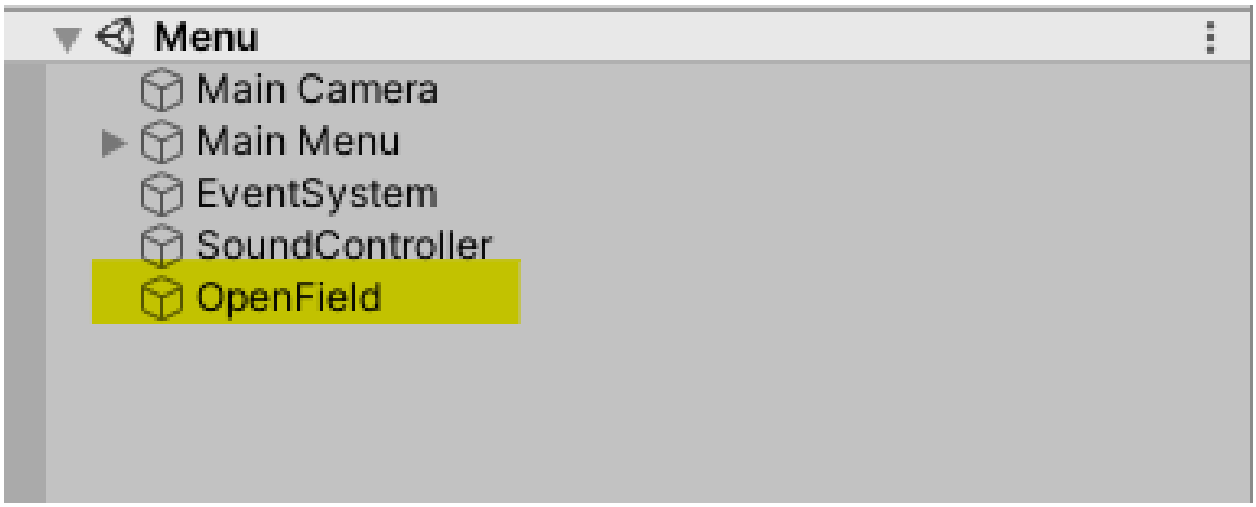

Figura 7.6: OpenField dentro de la escena

Dentro del script debemos crear la función *getCampoLibre*, con un parámetro de tipo string, esta función es la que se ejecutará una vez la plataforma nos haya devuelto los datos, que se encontrarán en ese parámetro. En este método podemos agregar la funcionalidad que necesitemos.

```
1 public class OpenField : MonoBehaviour {
2     public void getCampoLibre(string campoLibre) {
3         ...
4     }
5 }
```

Figura 7.7: Script OpenField

Cabe destacar que al recuperar el campo libre si el juego no ha guardado por primera vez un valor en este campo la plataforma no retornará un valor nulo o cadena vacía. Intentar acceder a ese valor lanzará una excepción que debemos capturar.

Cada vez que se quiera recuperar el valor del campo libre debemos realizar la llamada *getCampoLibre*. Una vez los datos sean recuperados el *Gameobject OpenField* ejecutará su función *getCampoLibre* con el valor obtenido.

Podemos utilizar esta llamada al cargar el menú principal para mostrar como desbloqueados ciertos niveles dependiendo del valor del campo libre almacenado.

En la figura 7.8 se muestra un ejemplo de una función que se encarga de gestionar los datos recibidos por la llamada *getCampoLibre*. Se podría utilizar la llamada a *getCampoLibre* desde el *Script* que se encargue de guardar la estadísticas. Posteriormente, añadir dentro de la función *getCampoLibre* del *Gameobject OpenField* una llamada a la función de la Figura 7.8.

```
1  public void guardarEstadisticas (string campoLibre) {
2      Estadisticas stats;
3      try {
4          if (campoLibre == "" || campoLibre == null) {
5              stats = new Estadisticas();
6              stats.puntuacion = new int[4];
7          } else {
8
9              stats = JsonUtility.FromJson<Estadisticas>(campoLibre);
10         }
11
12         if (stats.puntuacion[SceneManager.GetActiveScene().buildIndex -1] < puntuacion) {
13             stats.puntuacion[SceneManager.GetActiveScene().buildIndex -1] = puntuacion;
14         }
15     }
16     catch (System.Exception) {
17         stats = new Estadisticas();
18         stats.puntuacion = new int[4];
19         stats.puntuacion[SceneManager.GetActiveScene().buildIndex-1] = puntuacion;
20     }
21     string jsonStats = JsonUtility.ToJson(stats);
22     #pragma warning disable CS0618
23     Application.ExternalCall("setCampoLibre", jsonStats);
24     #pragma warning restore CS0618
25
26     int puntuacionTotal = 0;
27     for (int i = 0; i < 4; i++) {
28         puntuacionTotal += stats.puntuacion[i];
29     }
30     int superado = 0;
31     if (puntuacionTotal >= 4000) superado = 1;
32     int[] datos = { puntuacionTotal, superado };
33     #pragma warning disable CS0618
34     Application.ExternalCall("guardar", datos);
35     #pragma warning restore CS0618
36 }
37
38 private class Estadisticas {
39     public int[] puntuacion;
40 }
```

Figura 7.8: Ejemplo de uso de las Estadisticas

7.3. consideraciones Adicionales

Es importante tener en cuenta que al incrustar el juego en la plataforma la relación de aspecto no se conserva correctamente, por lo que hay que tener cuidado con la colocación de los elementos del menú y del escenario. Los elementos colocados en la parte superior de la pantalla pueden llegar a no aparecer. El formato de pantalla resultante es parecido a 16:10.

También es importante destacar que al añadir un video al proyecto y construir el proyecto en WebGL debemos seleccionar como fuente una url y cargar el video como se muestra en la figura 7.9.

```
1    video = GetComponent<VideoPlayer>();
2    string url = Application.streamingAssetsPath + "/Tutorial.mp4";
3    video.url = url;
```

Figura 7.9: Carga de video por URL

Capítulo 8

Conclusión

Una vez concluido el proyecto, podemos decir que este ha cumplido de manera satisfactoria con los objetivos que se planteaban al comienzo.

Se han construido dos juegos serios independientes que gamifican partes del temario de la asignatura de fundamentos de programación y que pueden llegar a facilitar el aprendizaje de los estudiantes que cursen la asignatura. Además, se han adaptado los juegos a la plataforma del proyecto de manera que se pueden jugar desde la misma y las puntuaciones se actualicen.

Considero que tanto la metodología aplicada como las tecnologías utilizadas han sido las adecuadas. El desarrollo iterativo e incremental me ha permitido hacer frente a las variaciones surgidas durante el desarrollo del proyecto, que al tratarse de algo tan volátil como un videojuego, han sido constantes. Por otra parte, *Unity* es una herramienta que facilita mucho desarrollo de videojuegos sobre todo para equipos de desarrollo pequeños.

Cada vez más docentes deciden adaptar su metodología y dar respuesta con ella las nuevas necesidades que surgen de la integración de las nuevas tecnologías en su día a día y en el aula. Creo que la gamificación es una buena técnica de aprendizaje, que aporta grandes beneficios y aplicada en las aulas puede suponer una evolución en los métodos de docencia clásicos.

8.1. Mejoras Futuras

Durante el desarrollo del proyecto se han observado determinadas mejoras que podrían implementarse en un futuro para aportar más valor a los juegos:

- Un aumento de la cantidad de niveles, incluyendo más contenidos de la asignatura fundamentos de programación, además de una revisión de los existentes.
- Adaptar el juego a diferentes plataformas, a partir del proyecto *Unity* permite construir el juego para diferentes plataformas, pero para que funcionen de forma adecuada requieren ajustes adicionales, como configurar los controles.

- Mejorar el apartado multimedia, añadir más y mejores animaciones, transiciones, sonidos e imágenes.
- Crear una historia o trasfondo de los juegos, que los englobe y permita crear narrativas dentro de los mismos.

Referencias

[1] Agencia tributaria. https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml. [Online; accessed Mar-2021].

[2] Ball sort puzzle. https://play.google.com/store/apps/details?id=com.GMA.Ball.Sort.Puzzle&hl=es&gl=US.

[3] Creación de una plataforma para la gamificación de una asignatura. http://uvadoc.uva.es/handle/10324/44410. [Online; accessed Mar-2021].

[4] Extensiones e integraciones de github. https://docs.github.com/es/github/customizing-your-github-workflow/github-extensions-and-integrations. [Online; accessed Mar-2021].

[5] Gamificación en el aula: gincana de programación. http://www.aenui.net/ojs/index.php?journal=revision&page=article&op=view&path%5B%5D=402&path%5B%5D=593. [Online; accessed Mar-2021].

[6] Gestión de riesgos. http://www.lsi.us.es/docencia/get.php?id=473. [Online; accessed Mar-2021].

[7] Git. https://www.git-scm.com/. [Online; accessed Mar-2021].

[8] Las 5 ventajas principales de la gamificación en el aprendizaje. https://insights.learnlight.com/es/articulos/5-ventajas-gamificacion-aprendizaje/. [Online; accessed Mar-2021].

[9] Less work in progress. https://www.plasticscm.com/. [Online; accessed Mar-2021].

[10] Metodología de desarrollo crystal. https://www.tecnologias-informacion.com/metodologia-crystal.html. [Online; accessed Mar-2021].

[11] Webgl. https://es.wikipedia.org/wiki/WebGL. [Online; accessed Mar-2021].

[12] *A guide to the project management body of knowledge*. Project Management Institute, 2004.

[13] Ludificación. https://es.wikipedia.org/wiki/Ludificaci%C3%B3n, Feb 2021. [Online; accessed Mar-2021].

[14] TRACY FULLERTON. *GAME DESIGN WORKSHOP: a playcentric approach to creating innovative games, third edition*. CRC Press, 2017.

[15] David Erosa García. Qué es unity y características principales. https://openwebinars.net/blog/que-es-unity/, Jul 2020. [Online; accessed Mar-2021].

[16] Unity Technologies. https://unity.com/es. [Online; accessed Mar-2021].