Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería del Software

Creación de un juego para el aprendizaje de paso de datos por valor y por referencia

---

Alumno:
D. Pablo Gil Alonso

Tutores:
Dña. Alma Pisabarro Marrón
D. Carlos Enrique Vivaracho Pascual

Agradecimientos

A mi familia que ha estado siempre desde el principio para apoyarme.

A todos mis amigos y en especial al grupo Pystacho por todos los momentos juntos que hemos pasado todo este tiempo.

A mis compañeros y profesores de la carrera por ayudarme y motivarme a mejorar cada día.

A mis tutores Alma y Carlos por haberme ayudado todos estos meses para sacar adelante el proyecto.

Resumen

Este Trabajo de Fin de Grado forma parte de un proyecto de gamificación para la asignatura Fundamentos de Programación del primer curso del grado en Ingeniería Informática, que intenta fomentar el aprendizaje de los alumnos mediante el uso de videojuegos.

El proyecto ha consistido en desarrollar un videojuego utilizando *Unity* que ayude a explicar algunos de los conceptos vistos en la asignatura, utilizando distintas técnicas de gamificación.

El juego desarrollado tratará de explicar el funcionamiento del paso de parámetros por valor y referencia y su comportamiento en distintos tipos de funciones y procedimientos.

Abstract

This project is part of a gamification initiative for the course *Fundamentos de programación* of the first year of the degree in Computer Engineering, which aims to promote student learning through the use of video games.

The project consists in developing a video game using Unity which will help to explain some of the concepts seen in the subject using different gamification techniques.

The game developed would try to explain parameter passing using values or references and its behavior in different types of functions and procedures.

Índice general

Agradecimientos **III**

Resumen **V**

Abstract **VII**

Lista de figuras **XIII**

Lista de tablas **XV**

1. Introducción **1**

1.1. Motivación 1

1.2. Objetivos 2

1.3. Contexto 2

2. Plan de Proyecto **5**

2.1. Metodología utilizada 5

2.2. Plan de riesgos 6

2.3. Presupuesto 9

2.4. Seguimiento del proyecto 10

3. Tecnologías Utilizadas **13**

3.1. *Unity* 13

3.2. *C#* 14
3.3. *Visual Studio* 2017 15
3.4. *Git* 15
3.5. *GIMP* 16
3.6. *Audacity* 16

4. Documento de Diseño de Juego (GDD) 17

4.1. Introducción 17
4.2. Audiencia y plataformas 17
4.3. Mecánicas 18
4.4. Niveles 20
4.5. Historia 26
4.6. Multimedia 28
4.7. Mundo del juego 29

5. Análisis 35

5.1. Elicitación de requisitos 35
5.1.1. Requisitos funcionales 35
5.1.2. Requisitos no funcionales 37
5.2. Casos de uso 39

6. Diseño e Implementación 41

6.1. Modelo de Dominio 41
6.2. Patrones de diseño utilizados 45
6.2.1. Patrón Observador 45
6.2.2. Patrón *Singleton* 45
6.3. Implementación de niveles 46
6.3.1. Creación de niveles 46

6.3.2. Carga de niveles 47
6.4. Integración con la plataforma 48
6.4.1. Lectura y escritura de puntuaciones 48
6.4.2. Cierre de la aplicación 50

7. Pruebas **51**
7.1. Pruebas Unitarias 51
7.2. Pruebas de integración 55
7.3. Pruebas Beta 56

8. Conclusiones **57**
8.1. Líneas de trabajo futuras 58

Bibliografía **59**

Lista de Figuras

2.1. Desarrollo en cascada 5

3.1. Logo *Unity* 14
3.2. Logo *C#* 14
3.3. Logo *Visual Studio* 15
3.4. Logo *Git* 15
3.5. Logo *GIMP* 16
3.6. Logo *Audacity* 16

4.1. Boceto de un problema con sus elementos principales 18
4.2. Objetos del juego 26
4.3. Métodos del juego 26
4.4. Objetivos del nivel 27
4.5. Puntuación del nivel 27
4.6. *Sprites* utilizados 28
4.7. Menú inicial 29
4.8. Nivel del juego 30
4.9. Objetos iniciales 30
4.10. Objetivos del nivel 31
4.11. Paso por valor y referencia 31

4.12. Cronometro del nivel 32

4.13. Menú de pausa 32

4.14. Resumen del nivel 33

5.1. Diagrama de casos de uso 39

6.1. Modelo de dominio: Elemento Partida 42

6.2. Modelo de dominio: Método 43

6.3. Modelo de dominio: Nivel 44

6.4. Patrón Observador 45

6.5. Patrón *Singleton* 46

6.6. *Prefabs* de los objetos de un nivel 47

Lista de Tablas

2.1. R-01: Enfermedad del estudiante 7
2.2. R-02: Fallo con el equipo de trabajo 7
2.3. R-03: Perdida de información 7
2.4. R-04: Cambio en los requisitos 8
2.5. R-05: Falta de conocimiento sobre las tecnologías 8
2.6. R-06: Falta de disponibilidad de los tutores 8
2.7. R-07: Retraso en el cumplimiento de la planificación 9
2.8. Duración del proyecto 9
2.9. Duración del proyecto 10
2.10. Seguimiento análisis de requisitos 10
2.11. Diseño del sistema 11
2.12. Implementación 11
2.13. Pruebas 12

4.1. Sistema de puntuación 20
4.2. Problema nivel 0 21
4.3. Problema nivel 1 22
4.4. Problema nivel 2 23
4.5. Problema nivel 3 24
4.6. Problema nivel 4 25

5.1. RF-01: Iniciar juego 35
5.2. RF-02: Salir del juego 35
5.3. RF-03: Iniciar nivel 35
5.4. RF-04: Salir del nivel 36
5.5. RF-05: Reiniciar nivel 36
5.6. RF-06: Pausar nivel 36
5.7. RF-07: Otorgar y almacenar puntuación 36
5.8. RF-08: Iniciar tutorial 36
5.9. RF-09: Crear objeto valor 36
5.10. RF-10: Crear objeto referencia 37
5.11. RF-11: Llamar a método 37
5.12. RF-12: Completar nivel 37
5.13. RF-13: Mostrar funcionamiento de métodos 37
5.14. RF-14: Mostrar ayuda sobre el paso de parámetros 37
5.15. RNF-01: Motor del juego 38
5.16. RNF-02: Lenguaje 38
5.17. RNF-03: Conexión con la plataforma de despliegue 38
5.18. RNF-04: Compatibilidad 38
5.19. RNF-05: Interfaz adaptable 38

7.1. PU-01: Cargar Nivel 52
7.2. PU-02: Crear nuevo valor o referencia 52
7.3. PU-03: Destruir valor 52
7.4. PU-04: Destruir referencia 52
7.5. PU-05: Asignar objeto a método 53
7.6. PU-06: Asignar objeto a solución 53
7.7. PU-07: Ejecutar función 53

7.8. PU-08: Ejecutar procedimiento 53

7.9. PU-09: Completar nivel 54

7.10. PU-10: Pausar nivel 54

7.11. PU-11: Reanudar nivel 54

7.12. PU-12: Reiniciar problema 54

7.13. PU-13: Volver al menú 55

7.14. PU-14: Salir del juego 55

Capítulo 1

Introducción

El objetivo de este proyecto es el desarrollo de un videojuego mediante el motor *Unity*, orientado al mundo educativo. Se tratará de un juego serio que formará parte de una iniciativa que pretende gamificar los conceptos que se ven en la asignatura Fundamentos de Programación. La aplicación de desplegará en una plataforma web ya desarrollada en la que se alojan distintos videojuegos con la misma finalidad.

Mediante el desarrollo de este proyecto se quiere explorar el uso de videojuegos como herramientas de aprendizaje en centros educativos. La gamificación [4] es la utilización de técnicas, elementos y dinámicas típicas de los juegos y actividades de ocio con el fin de fomentar la motivación, mejorar la productividad o ayudar al aprendizaje. Esta técnica pretende aplicar conceptos de los videojuegos a tareas externas a ellos de forma que faciliten el cumplimiento de objetivos, normalmente relacionados con el aprendizaje.

En nuestro caso todo esto se llevará a cabo mediante la creación de un minijuego que se centrará en explicar el paso de parámetros a funciones y procedimientos, más concretamente en la diferencia al pasar estos parámetros como valor o como referencia

1.1. Motivación

El tipo de conceptos que se ven en programación en ocasiones pueden ser demasiado abstractos y difíciles de comprender cuando te estas introduciendo en la materia. Además, los entornos de desarrollo software y las tecnologías utilizadas no suelen ser muy intuitivos y amigables con el usuario. Todos estos elementos pueden hacer que la primera toma de contacto para el alumno con este mundo sea frustrante.

Mediante el uso de videojuegos y técnicas de gamificación para explicar estos conceptos se puede ayudar a los estudiantes a asimilar la información fácilmente, en un entorno mucho más amigable y conocido que hará que las personas a las que va dirigida la explicación estén mucho más motivadas y predispuestas a aprender.

Al planificar y realizar este proyecto se ha intentado tener todo esto en cuenta para hacer del juego una experiencia divertida y educativa para el jugador a la vez que se ayuda al profesor en su labor de enseñanza. Al incluir el minijuego en una plataforma ya existente que recopila otros proyectos similares, se está colaborando a crear una herramienta con la que los profesores podrán realizar un seguimiento del desempeño y progreso de los alumnos mientras se exploran nuevas técnicas para mejorar la experiencia de los alumnos.

1.2. Objetivos

El objetivo principal de este proyecto es la creación de un videojuego en *Unity* sobre paso de parámetros, adicionalmente se han planteado también los siguientes objetivos:

- Integrar el juego en una plataforma orientada a la gamificación
- Crear varios niveles, cada uno de los cuales explique un concepto distinto relacionado con la temática general del juego.
- Crear un sistema de puntuación que incite a los alumnos a jugar y así mejorar sus resultados. Estas puntuaciones servirán además para evaluar el desempeño de los jugadores.
- Crear un juego con una temática y aspecto que motive a los jugadores a jugar y los haga sentir cómodos.

Este trabajo también cuenta con algunos objetivos adicionales fruto del trabajo anterior en la plataforma para la gamificación:

- Realizar modificaciones en el videojuego *Caída de Datos* relacionadas con el tipo de letra utilizada en algunas pantallas. Estos cambios pretenden mejorar la experiencia del usuario al jugar.
- Corregir fallos en la forma que los juegos *Caída de Datos* y *Fiesta Recursiva* realizan la comunicación con la plataforma para guardar las puntuaciones de cada partida.

1.3. Contexto

Actualmente la industria de los videojuegos [17] es una de las industrias más potentes dentro del mundo del entretenimiento. A pesar de la creencia de la gente, los videojuegos hace tiempo que superaron a otros grandes gigantes de este sector. Llegando a facturar en los últimos años más que industrias todopoderosas como son el cine y la música juntos.

Este boom en el sector de los videojuegos se debe entre otras cosas a la gran cantidad de posibilidades que ofrecen para todo el público, abarcando desde sencillos minijuegos para los

más pequeños hasta grandes superproducciones lúdico-narrativas que alcanzan presupuestos superiores a los 250 millones de dólares [8]. Todo esto ha provocado que los videojuegos se conviertan en un fenómeno mundial pasando de ser un simple entretenimiento a un nuevo mundo lleno de posibilidades que sirve como punto de encuentro para millones de personas.

Aprovechando el impacto de los videojuegos en la sociedad, están surgiendo nuevos usos y proyectos de todo tipo con los videojuegos como herramienta principal. Este es el caso de la gamificación en el aula, que hace uso de esta tecnología para entornos educativos, con el objetivo de adquirir conocimientos y desarrollar nuevas habilidades en un entorno más lúdico y diferenciado de la enseñanza tradicional.

Ya se han realizado multitud de proyectos y estudios en este campo y se ha podido ver que la gamificación puede llegar a aportar multitud de ventajas:

- **Motivar al alumno**: la gamificación puede ayudar al alumno a mejorar su motivación, creando experiencias divertidas y emocionantes que se alejan de la rutina y que hacen que el jugador este más predispuesto a colaborar al encontrarse en un entorno más cómodo para el.
- **Recibir retroalimentación al instante**: los videojuegos suelen proporcionar retroalimentación constante al jugador lo que ayuda al alumno a tener una referencia en todo momento de su progreso y aprendizaje.
- **Recompensas**: los videojuegos normalmente proporcionan una recompensa al jugador ya sea en forma de puntos, trofeos, etc. Esto incita al jugador a seguir jugando para mejorar sus puntuaciones y ampliar su lista de logros.
- **Escalado de dificultad**: los videojuegos suelen definir una serie de niveles que el usuario debe ir superando para avanzar al siguiente. Mediante esta mecánica es posible introducir poco a poco al alumno en los conceptos que se quieren explicar facilitando su aprendizaje.
- **Socializar**: un aspecto importante de muchos videojuegos es el aspecto social de los mismos, permitiendo a los distintos jugadores interactuar entre ellos creando una comunidad que los anima a seguir jugando y mejorando.
- **Uso de los conocimientos**: el uso de videojuegos para la enseñanza permite mostrar el uso de los distintos conocimientos y habilidades adquiridos en diferentes entornos y situaciones que permiten al jugador aplicar lo aprendido de manera practica y eficaz.

Capítulo 2

Plan de Proyecto

2.1. Metodología utilizada

Para el desarrollo del proyecto se ha decidido utilizar desarrollo en cascada [23]. El desarrollo en cascada es un enfoque metodológico que ordena las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar al que la anterior haya finalizado. Al final de cada una de las etapas de desarrollo se lleva una revisión fina que debe determinar si el proyecto está preparado para comenzar la fase siguiente.

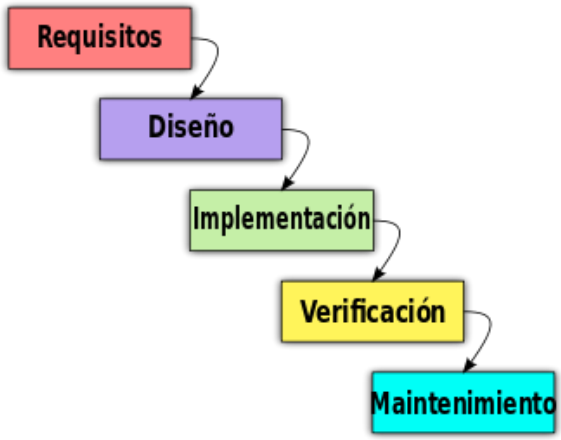

Figura 2.1: Desarrollo en cascada

Existen varios modelos de esta metodología, cada uno con diferentes etapas para el desarrollo del proyecto, sin embargo, hay varias etapas que deben aparecer siempre:

- **Análisis de requisitos**: en esta fase se analizan los requisitos que debe cumplir el software para alcanzar los objetivos marcados, pero sin entrar en detalles internos. Esta etapa es en la que se recogen todos los requisitos de software, alcance del proyecto, análisis de riesgos y la estimación de costes. Es importante que en esta parte se analicen bien todos los detalles y se haga de forma consensuada, ya que a partir de lo que se obtenga se desarrollaran el resto de las etapas del proyecto no pudiéndose realizar cambios a mitad del proceso.

- **Diseño del sistema**: en esta fase se busca una solución especifica en base a los requisitos analizados en la etapa anterior. Los desarrolladores se encargan de diseñar la arquitectura que se va a utilizar para desarrollar el software, así como de crear un plan de diseño detallado. La fase de diseño da como resultado un borrador con los planes para la implementación y pruebas de los distintos componentes de la aplicación.

- **Implementación**: en esta fase se implementa la arquitectura diseñada en la fase anterior, esto incluye la programación del software y la ejecución de pruebas para la solución de posibles errores.

- **Pruebas**: en esta fase los componentes implementados en la fase anterior se unen para componer el sistema final y comprobar que todo funciona de forma correcta. El objetivo es conocer la calidad del software realizando diversas pruebas sobre él, como pueden ser pruebas unitarias, de caja blanca, de aceptación, etc.

- **Mantenimiento**: una vez que ya han concluido todas las fases anteriores y se ha lanzado la primera versión de la aplicación se realiza las labores de mantenimiento necesarias sobre esta.

Los principales motivos por los que se ha elegido esta metodología para el desarrollo del proyecto es que funciona bien para equipos pequeños debido a su forma de trabajar. Se obtienen unas fases de desarrollo claras y bien definidas, los objetivos quedan bien especificados y la carga y costes de trabajo se estiman antes de comenzar cualquier desarrollo. Además, se puede representar de forma cronológica fácilmente.

2.2. Plan de riesgos

Para la planificación del proyecto se ha elaborado un plan de riesgos que deben tenerse en cuenta durante el desarrollo. Es importante realizar este paso en la etapa de análisis, pues cualquier contratiempo que pueda surgir más adelante puede tener consecuencias catastróficas si no se ha previsto con antelación su aparición y una posible respuesta.

Para realizar el plan de riesgos se debe tener en cuenta la probabilidad de que ocurra y el impacto que pueden causar, a partir de esto se prevé un plan de contingencia para cada riesgo identificado.

Riesgo-01	Enfermedad del estudiante
Descripción	El estudiante no puede trabajar en el proyecto debido a causas médicas.
Probabilidad	Baja
Impacto	Medio
Medidas preventivas	Evitar actividades que puedan poner en riesgo la salud.
Plan de contingencia	Realizar la planificación del proyecto incluyendo margenes de tiempo adicional entre tareas importantes.

Tabla 2.1: R-01: Enfermedad del estudiante

Riesgo-02	Fallo con el equipo de trabajo
Descripción	El estudiante no puede trabajar en el proyecto debido problemas técnicos con el equipo de desarrollo.
Probabilidad	Baja
Impacto	Medio
Medidas preventivas	Realizar copias de seguridad y trabajar de forma online cuando sea posible.
Plan de contingencia	Recuperar los datos datos de la copia de seguridad y seguir trabajando desde otro dispositivo.

Tabla 2.2: R-02: Fallo con el equipo de trabajo

Riesgo-03	Perdida de información
Descripción	El estudiante no puede trabajar en el proyecto debido problemas técnicos con el equipo de desarrollo.
Probabilidad	Baja
Impacto	Alto
Medidas preventivas	Realizar copias de seguridad del proyecto.
Plan de contingencia	Realizar la planificación del proyecto incluyendo margenes de tiempo adicional entre tareas importantes.

Tabla 2.3: R-03: Perdida de información

Riesgo-04	Cambio en los requisitos
Descripción	Se realizan cambios en los requisitos analizados en la planificación del proyecto.
Probabilidad	Media
Impacto	Alto
Medidas preventivas	Realizar un buen análisis de requisitos, para que no sea necesario modificarlos en un futuro.
Plan de contingencia	Analizar el cambio propuesto y decidir si es posible implementarlo.

Tabla 2.4: R-04: Cambio en los requisitos

Riesgo-05	Falta de conocimiento sobre las tecnologías
Descripción	No se tiene suficiente experiencia con las tecnologías utilizadas para el desarrollo.
Probabilidad	Media
Impacto	Medio
Medidas preventivas	Formarse sobre las tecnologías que se van a utilizar antes del comienzo del proyecto.
Plan de contingencia	Consultar documentación sobre el tema y volver a planificar tareas si es necesario.

Tabla 2.5: R-05: Falta de conocimiento sobre las tecnologías

Riesgo-06	Falta de disponibilidad de los tutores
Descripción	No se tiene suficiente experiencia con las tecnologías utilizadas para el desarrollo.
Probabilidad	Baja
Impacto	Alto
Medidas preventivas	Planificar reuniones con antelación suficiente.
Plan de contingencia	Realizar tareas que no requieran de la retroalimentación de los tutores.

Tabla 2.6: R-06: Falta de disponibilidad de los tutores

Riesgo-07	Retraso en el cumplimiento de la planificación
Descripción	Debido a retrasos en el desarrollo no se puede cumplir con la planificación.
Probabilidad	Media
Impacto	Alto
Medidas preventivas	Realizar una buena planificación previa de actividades.
Plan de contingencia	Incluir margenes de tiempo al realizar la planificación y replanificar actividades si es necesario.

Tabla 2.7: R-07: Retraso en el cumplimiento de la planificación

2.3. Presupuesto

El coste real del proyecto ha sido de 0€ ya que no ha habido que pagar sueldo de desarrollado al alumno y no ha habido que comprar material adicional. Sin embargo, se simulará un análisis de costes del proyecto como si se hubiese tratado de un desarrollo comercial.

Lo primero que se necesita saber para calcular los costes del proyecto es la duración total planificada del mismo. En las duraciones mostradas en la tabla 2.13 está incluida la duración de cada tarea para las etapas de desarrollo.

Tarea	**Duración**
Desarrollo *scripts* c#	100 horas.
Desarrollo con el motor *Unity*	100 horas.
Creación de documentación	90 horas.
Creación de imágenes y *sprites*	10 horas.
Creación de animaciones	10 horas.
Creación de efectos de sonido	4 horas.
Total	314 horas.

Tabla 2.8: Duración del proyecto

El proyecto contará con un único desarrollador, el sueldo medio para un programador junior en España es de 18.836€ al año [18], es decir, 9€ la hora.

El hardware utilizado consta de un ordenador portátil valorado en 1000€ y un monitor externo utilizado como segunda pantalla valorado en 200€. La vida útil de un ordenador

portátil está estimada entre 2 y 4 años. Para calcular el presupuesto se ha estimado cogeremos el valor intermedio, 3 años, por lo que su coste sería de 27,8€/mes. La vida útil de un monitor está estimada en unos 8 años, por lo que su coste sería de 2,08€/mes.

El software no supone ningún coste en este proyecto. *Unity* cuenta con una licencia gratuita para proyectos con ingresos menores a 100 mil dolares anuales. Para la creación de elementos multimedia también se han utilizado programas con licencia gratuita como son *Audacity* para la edición de audio y *GIMP* para la edición de imágenes.

Todas las imágenes utilizadas en el juego cuentan con licencia de uso gratuita por lo que tampoco han supuesto ningún coste. Además, no se han tenido en cuenta por ser mínimos, costes de electricidad, calefacción, espacio de trabajo, etc.

Tipo de coste	**Precio (€)**
Desarrollador junior	2.826€
Hardware	59,76€
Software	0€
Otros recursos	0€
Total	2885,76€

Tabla 2.9: Duración del proyecto

2.4. Seguimiento del proyecto

A continuación, se muestra de manera detallada el seguimiento del proyecto para cada una de las etapas utilizadas en su desarrollo.

La fase inicial del proyecto trascurrió según lo esperado, no se produjeron grandes retrasos en ninguna de las tareas realizadas y se cumplió con todas las fechas de finalización estimadas dentro de los márgenes estimados.

Tarea	**Horas estimadas**	**Horas reales**	**Fecha inicio**	**Fecha fin**
Formaciones previas	15 horas	15 horas	15-02-2022	17-02-2022
Análisis creativo	4 horas	4 horas	18-02-2022	18-02-2022
Definición de requisitos	3 horas	3 horas	21-02-2022	21-02-2022
Análisis casos de uso	2 horas	2 horas	21-02-2022	21-02-2022
Realización del GDD	8 horas	15 horas	22-02-2022	25-02-2022
Total	32 horas	39 horas	15-02-2022	25-02-2022

Tabla 2.10: Seguimiento análisis de requisitos

En la fase de diseño del videojuego surgieron algunos retrasos en alguna de las tareas debido a la falta de experiencia con las tecnologías, sin embargo, no supuso ningún problema ya que se habían planificado las etapas dejando márgenes de tiempo entre ellas por los retrasos que pudiesen aparecer.

Tarea	**Horas estimadas**	**Horas reales**	**Fecha inicio**	**Fecha fin**
Diseño de la interfaz	5 horas	8 horas	28-02-2022	29-02-2022
Diseño de la arquitectura	6 horas	6 horas	01-03-2022	02-03-2022
Diseño del modelo de dominio	15 horas	15 horas	03-03-2022	07-03-2022
Diseño de niveles	12 horas	13 horas	07-03-2022	11-03-2022
Total	38 horas	42 horas	28-02-2022	11-03-2022

Tabla 2.11: Diseño del sistema

La fase de implementación ha sido la más larga del proyecto y también en la que más retrasos se han producido. Debido a la falta de conocimiento sobre las tecnologías que se iban a utilizar no se pudo estimar correctamente la duración cada tarea.

Tarea	**Horas estimadas**	**Horas reales**	**Fecha inicio**	**Fecha fin**
Configuración del proyecto *Unity*	2 horas	2 horas	14-03-2022	14-03-2022
Creación de *sprites* e imágenes	40 horas	47 horas	14-03-2022	28-03-2022
Implementación de la interfaz	30 horas	31 horas	29-03-2022	05-04-2022
Implementación de *scripts*	80 horas	120 horas	06-04-2022	07-05-2022
Creación de animaciones	35 horas	43 horas	09-05-2022	20-05-2022
Creación de efectos de sonido	8 horas	7 horas	23-05-2022	24-05-2022
Conexión con la plataforma web	12 horas	12 horas	25-05-2022	28-05-2022
Total	207 horas	262 horas	14-03-2022	28-05-2022

Tabla 2.12: Implementación

En la etapa de pruebas se han cumplido todas las estimaciones iniciales ya que solamente se han encontrado pequeños errores con las animaciones y la conexión con la plataforma que no han necesitado mucho tiempo para ser corregidos.

Tarea	**Horas estimadas**	**Horas reales**	**Fecha inicio**	**Fecha fin**
Pruebas unitarias	16 horas	16 horas	29-05-2022	02-06-2022
Pruebas de integración	13 horas	13 horas	04-06-2022	06-06-2022
Pruebas beta	8 horas	8 horas	07-06-2022	08-06-2022
Total	200 horas	256 horas	29-05-2022	08-06-2022

Tabla 2.13: Pruebas

Capítulo 3

Tecnologías Utilizadas

En este capítulo se presentan las tecnologías y herramientas utilizadas para el desarrollo del proyecto, así como para la gestión de versiones y el seguimiento del mismo.

3.1. *Unity*

Se ha utilizado *Unity* [21] como software principal para el desarrollo del proyecto, más concretamente la versión 2020.3.27, ya que proporciona de forma gratuita múltiples herramientas para el desarrollo de videojuegos, además de contar con actualizaciones periódicas que mantienen el software compatible con las últimas tecnologías.

Unity es un motor de videojuego multiplataforma creado por *Unity Technologies*. Un motor de videojuego es un software que recoge una serie de rutinas de programación que facilitan el diseño, desarrollo y funcionamiento de un videojuego, algunas de las funcionalidades que proporcionan estos motores y que también incluye *Unity* son:

- Renderizado de gráficos 2D y 3D
- Gestión de Animaciones
- Motor físico
- Detección de colisiones
- *Scripting*
- Inteligencia artificial
- Sonido

Una de las principales características de este motor es su capacidad de desarrollo multiplataforma. No solo permite elegir desde que sistema se desea desarrollar el proyecto, ya que está disponible para *Windows*, *MacOS* y *Linux*, sino que además permite la creación de software para distintos dispositivos a partir de un mismo proyecto, siendo capaz de exportar el juego creado a más de 25 plataformas distintas. Algunas de las más populares que se encuentran en esta lista son son: PC *Windows*, *MacOS* y *Linux*, dispositivos móviles *iOS* y *Android*, consolas *Play Station*, *Xbox* y *Nintendo* y plataformas web.

Todo esto sumado a la gran comunidad de desarrolladores con la que cuenta y a su licencia gratuita [20], la convierten en una de las herramientas para el desarrollo de videojuegos más populares.

Figura 3.1: Logo *Unity*

3.2. *C#*

C# [2] es un lenguaje de programación orientado a objetos y desarrollado por *Microsoft* dentro del conjunto de tecnologías *.NET*, se desarrolló como una evolución a *C* y *C++* y su sintaxis es similar a *Java*.

Unity solamente permite utilizar C# para los distintos scripts que controlan el comportamiento de los objetos dentro del juego, por lo que este es el lenguaje que se ha utilizado para ello. Esto es debido a que las clases que se utilizan para controlar aspectos básicos de cualquier juego, como pueden ser los eventos, animaciones o colisiones, están programados en este lenguaje dentro de la librería *UnityEngine* [19].

Figura 3.2: Logo *C#*

3.3. *Visual Studio* 2017

Visual Studio [22] es un IDE (Entorno de Desarrollo Integrado) compatible con múltiples lenguajes de programación, tales como *C++*, *C#*, *Visual Basic*, *.NET*, *F#*, *Java*, *Python*, *Ruby* y *PHP*.

Esta desarrollado por *Microsoft* y optimizado para las tecnologías dentro del framework *.NET*. Además, es la plataforma de desarollo recomendada por *Unity* ya que permite depurar de forma sencilla el videojuego, por lo que ha sido el IDE elegido para este proyecto .

Figura 3.3: Logo *Visual Studio*

3.4. *Git*

Git [6] es un software de control de versiones gratuito y de código abierto, pensado para el mantenimiento de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en los distintos archivos de un proyecto y coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

Una de las principales ventajas de *Git* además del control de versiones y copias de seguridad ya mencionadas, es su sistema de trabajo en ramas. Las ramas están destinadas a provocar proyectos divergentes de un proyecto principal, para hacer experimentos o para probar nuevas funcionalidades, creando una línea de progreso diferente de la rama principal donde desarrollar código para después hacer una fusión al proyecto principal.

Figura 3.4: Logo *Git*

Para el desarrollo de este proyecto se ha utilizado un repositorio *Git* alojado en *GitHub* [7] que es una de las plataformas de gestión de repositorios más populares y completas que se pueden encontrar.

3.5. *GIMP*

GIMP [5] (GNU Image Manipulation Program) es un programa, libre y gratuito, de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Forma parte del proyecto GNU y está disponible bajo la Licencia pública general de *GNU* y *GNU Lesser General Public License*.

Figura 3.5: Logo *GIMP*

Se ha utilizado este programa para la creación y edición de todas las imágenes del videojuego, tanto las utilizadas para crear las interfaces como las de los distintos objetos y elementos de los diferentes niveles.

3.6. *Audacity*

Audacity [1] es un software de edición de audio y grabación de sonido digital, completamente gratuito, creado en 1999 por Dominic Mazzoni y Roger Dannenberg que fue lanzado al mercado de forma oficial en mayo de 2000 en la versión 0.8. Este software de código abierto está disponible para *Windows*, *MacOS* , *Linux* y otros sistemas operativos similares.

Este programa se ha utilizado para editar la música de fondo de cada nivel y los distintos efectos de sonido que aparecen en el videojuego.

Figura 3.6: Logo *Audacity*

Capítulo 4

Documento de Diseño de Juego (GDD)

4.1. Introducción

La aplicación que se va a desarrollar en este trabajo es un videojuego educativo que ejemplifica en el paso de parámetros en programación, concretamente en el paso de parámetros por valor y por referencia. La finalidad del juego es mostrar y explicar de manera lúdica cómo funciona el paso de parámetros y sus diferentes características.

El objetivo del juego es obtener a partir de un objeto u objetos iniciales, el objeto u objetos finales requeridos en cada nivel. Para conseguirlo el jugador debe hacer uso de distintos métodos, que modificarán el estado del objeto o devolverán uno nuevo a partir del de entrada. El jugador deberá encontrar la secuencia de llamadas que en el menor número de pasos posibles genere la solución del nivel.

4.2. Audiencia y plataformas

El juego forma parte de un proyecto de gamificación de la asignatura Fundamentos de programación del primer curso del Grado en Ingeniería Informática y Grado en Estadística por lo que la audiencia del juego serán los alumnos que cursan dicha asignatura.

La plataforma sobre la que se ejecutan los juegos será una página web, de manera que serán accesibles y podrán ser jugados mediante un navegador.

4.3. Mecánicas

La mecánica básica del juego es conseguir un estado final en uno o varios objetos, usando para ello la secuencia correcta de métodos y/o funciones. El estado inicial de los objetos y el final se le da al jugador, y este tiene que pasar de uno al otro utilizando de la manera más efectiva y correcta posible los métodos y/o funciones que se le proporcionan.

El juego comenzará en cuanto se muestre por pantalla el nivel con el problema a resolver. Los distintos problemas estarán formados por los tres elementos comentados: un objeto u objetos iniciales que le serán proporcionados al jugador informándole de su estado y características (figura 4.1 arriba a la izquierda), varios métodos que recibirán esos objetos como parámetros (figura 4.1 en el centro abajo) y un objeto u objetos finales buscado y sus características (figura 4.1 arriba a la derecha).

Figura 4.1: Boceto de un problema con sus elementos principales

Para poder llegar al estado solución el jugador deberá usar los métodos disponibles, funciones a las que solo podrá pasar los objetos como valor y no cambiarán su estado, sino que devolverán otros nuevos, y procedimientos, estos últimos tendrán diferentes efectos sobre el objeto dependiendo de cómo se haya pasado el parámetro ya sea, valor o referencia, y será el jugador el encargado de decidir en qué momento se utiliza una estrategia u otra en función de lo que crea más conveniente.

Funcionamiento de los distintos métodos:

- Funciones: solo admiten parámetros pasados por valor. A partir de los objetos introducidos crea un nuevo objeto local a la función que el jugador puede utilizar para pasarlo como parámetro a otro método. Al terminar de ejecutarse los objetos de entrada desaparecen.

- Procedimientos: admite parámetros pasados por valor o por referencia, que pueden ser de entrada, salida o entrada-salida. Al terminar la ejecución se modifica el estado de los parámetros que sean de salida, si el parámetro estaba pasado como valor desaparece y si era una referencia, se modificara también el estado del objeto original.

El jugador cuenta con un número ilimitado de intentos para encontrar una secuencia que complete el nivel. Para obtener la puntuación máxima deberá encontrar esa secuencia en el menor tiempo y utilizando el menor número de pasos posibles. Este número se establece al inicio del nivel. Por cada paso de más que utilice el jugador se restará una cantidad de puntos que varía en función de la dificultad y las características del nivel.

Acciones del jugador durante el desarrollo del juego:

- Pasar un objeto como parámetro a cualquiera de las funciones y procedimientos. Esto se puede hacer de dos formas, por valor o por referencia, cuando el nivel lo permita. En ese caso el jugador contará con un botón en la pantalla en el que podrá seleccionar el tipo de paso de parámetro (valor o referencia) mientras esté activada una de las opciones todos los objetos que se utilicen en ese periodo se comportarán de la forma indicada por el jugador. Cabe destacar que en algunos niveles el botón puede aparecer desactivado obligando al jugador a utilizar los objetos de una manera determinada.

- Ver el comportamiento de los métodos. Al lado de cada método se mostrará una imagen que, al clicar sobre ella, desplegará una pequeña descripción que aclarará brevemente la acción que realiza ese método y el efecto que tiene sobre cada objeto.

- Entregar un objeto como solución. Una vez que un objeto se encuentre en el mismo estado que el objeto solución pedido el jugador podrá entregar el objeto a la lista de soluciones, esto hará que ese objeto no pueda volver a usarse en ese problema. Una vez que se han entregado todas las soluciones el nivel se completa y se deja de jugar

Para calcular la puntuación lo primero que se tiene en cuenta es el tiempo empleado. Solo se podrá obtener la puntuación máxima en un nivel si se resuelve por debajo del límite de tiempo y si se utiliza el menor número de pasos posibles. De esta forma, además de valorar la rapidez se tiene en cuenta la eficiencia.

Existe una cantidad máxima de puntos que podrán ser descontados, de forma que el jugador siempre obtenga puntos al superar el nivel sin importar la cantidad de pasos o de tiempo que haya empleado.

Se puede ver un ejemplo de cómo funciona el sistema de puntuación en la tabla 4.1

Solución óptima: 3 pasos y 30 segundos o menos		
Tiempo	Pasos	Puntos
00:30	3	2000
	4	1750
	5	1600
01:00	3	1500
	4	1250
	5	1100

Tabla 4.1: Sistema de puntuación

Los datos que se pueden ver en la tabla anterior son valores de ejemplo que no hacen referencia a ningún nivel en concreto. Todos los límites de tiempo y número de pasos mínimos es distinto para cada problema para poder adaptarse a la complejidad de cada uno y conseguir un juego equilibrado.

4.4. Niveles

El juego contara con 5 niveles, cada uno de ellos se centrará en un aspecto del paso parámetros que será introducido al jugador mediante las distintas mecánicas del juego. Además, los niveles podrán utilizar los conceptos de niveles anteriores ya que tendrán dificultad incremental.

El juego también contará con un tutorial en el que se le explicará al jugador las reglas y mecánicas básicas del juego.

Nivel 0

El primer nivel se centrará en el paso de parámetros con funciones. Al usuario se le mostrará una única función, uno o dos objetos iniciales y solo se podrá utilizar el paso por valor en todas las llamadas a la función. Para completar los problemas de este nivel bastará con utilizar la función de cada problema y entregar la solución para superar el nivel.

El resultado de este nivel no se tendrá en cuenta para calcular la puntuación total de la partida y solo será necesario completar un problema para superar el nivel, ya que el objetivo de éste, además de entender el uso de funciones, es que el jugador comprenda las distintas mecánicas del juego y se adapte a los controles y menús del mismo.

En la tabla 4.2 se puede ver un ejemplo de problema para el nivel 0.

Objetos de partida:	Taza vacía
Objetivos:	Taza llena de café
Métodos	
- func llenarTaza() Recibe una taza vacía y devuelve una nueva taza llena de café	
Solución	
Llamar a la función 'llenarTaza' pasando la taza vacía de partida por valor, se recibe una taza llena y se arrastra a la solución para completar el nivel.	

Tabla 4.2: Problema nivel 0

Nivel 1

El segundo nivel es similar al anterior. Los problemas solamente contienen funciones y paso de parámetros por valor sin añadir ningún nuevo concepto. Las diferencias de este nivel con el anterior es que ahora se incluyen varias funciones y es el jugador el que debe decidir cuáles de ellas utiliza para obtener la solución óptima. Alguna de las funciones pueden necesitar dos parámetros de entrada diferentes y además puede ser necesario hacer varias llamadas a las funciones hasta conseguir la solución.

Para superar este nivel será necesario completar tres problemas y la puntuación máxima que se podrá obtener por superar todos ellos serán 1000 puntos.

El objetivo de este nivel es que el jugador aprenda a utilizar de forma más fluida los controles del juego e intente obtener las soluciones utilizando el menor número de pasos posibles.

En la tabla 4.3 se puede ver un ejemplo de problema para el nivel 1.

Objetos de partida:	Taza vacía y media cafetera
Objetivos:	Taza llena de café
Métodos	
- func llenarTaza() Recibe una taza vacía y una cafetera con café y devuelve una nueva taza llena de café - func vaciarTaza() Recibe una taza llena y devuelve una nueva taza vacía - func llenarCafetera() Recibe una cafetera y devuelve una nueva cafetera llena	
Solución	
Llamar a función 'llenarTaza' pasando la taza vacía de partida y la cafetera por valor, se recibe una taza llena y se arrastra a la solución para completar el nivel.	

Tabla 4.3: Problema nivel 1

Nivel 2

En este tercer nivel se introducen los procedimientos y el paso por referencia. Los problemas son similares a los del nivel anteriores con la misma cantidad de objetos y métodos, pero en este caso con procedimientos. En este nivel solamente se podrá usar el paso de parámetros por referencia.

Para superar este nivel será necesario completar tres problemas y la puntuación máxima que se podrá obtener por superar todos ellos serán 1500 puntos.

El objetivo de este nivel es que el jugador entienda el paso de parámetros por referencia y el funcionamiento de los procedimientos.

En la tabla 4.4 se puede ver un ejemplo de problema para el nivel 2.

Objetos de partida:	Vaso vacío y jarra vacía
Objetivos:	Vaso lleno de zumo
Métodos	
- proc llenarJarra() Recibe una jarra vacía y llena la jarra de zumo - proc llenarVaso() Recibe un vaso vacío y una jarra y llena el vaso con la mitad del contenido de la jarra	
Solución	
Llamar al procedimiento 'llenarJarra' pasando la jarra vacía de partida por referencia, una vez la jarra se ha llenado, llamar a 'llenarVaso' pasando la jarra llena y el vaso vacío por referencia para llenar el vaso y completar el nivel.	

Tabla 4.4: Problema nivel 2

Nivel 3

En el cuarto nivel se sigue trabajando con el uso de procedimientos, incluyendo ahora también el paso por valor para los distintos objetos. En estos problemas ya se le da al jugador la opción de elegir el tipo de paso de parámetro (valor o referencia) al utilizar los objetos y procedimientos. También se aumenta un poco la dificultad de los problemas incluyendo más métodos, objetos y número de pasos necesarios para obtener la solución.

Para superar este nivel será necesario completar dos problemas y la puntuación máxima que se podrá obtener por superar todos ellos serán 2000 puntos.

El objetivo de este nivel es que el jugador aprenda a diferenciar el comportamiento del paso por valor y referencia de un mismo objeto para un determinado procedimiento.

En la tabla 4.5 se puede ver un ejemplo de problema para el nivel 3.

Objetos de partida:	Taza llena y cafetera llena
Objetivos:	Taza vacía y cafetera vacía
Métodos	
- proc vaciarTaza() Recibe una taza llena y la vacía - proc llenarTaza() Recibe una taza vacía y una cafetera y llena la taza con la mitad del contenido de la cafetera	
Solución	
Llamar al procedimiento 'vaciarTaza' pasando la taza llena de partida por referencia, una vez la taza se ha vaciado, llamar dos veces a 'llenarTaza' pasando la cafetera llena por referencia y la taza vacía por valor para vaciar la cafetera sin llenar la taza y poder completar el nivel.	

Tabla 4.5: Problema nivel 3

Nivel 4

Este último nivel es un recopilatorio de todo lo visto en los niveles anteriores. Varios objetos de partida y varios objetos solución buscados, ahora aparecerán procedimientos y funciones indistintamente y el jugador decide el tipo de paso de parámetro. También se incluyen funciones y procedimientos que aparentemente tienen el mismo funcionamiento, por lo que el jugador tendrá que decidir que estrategia seguir y que métodos utilizar para poder superar el nivel y obtener la máxima puntuación haciendo las llamadas de la manera más eficiente.

Para superar este nivel será necesario completar dos problemas y la puntuación máxima que se podrá obtener por superar todos ellos serán 3000 puntos.

El objetivo de este nivel es que el alumno utilice todos los conceptos vistos en los anteriores niveles demostrando que es capaz de entenderlos y utilizarlos de forma correcta para superar los problemas planteados.

En la tabla 4.6 se puede ver un ejemplo de problema para el nivel 4.

Objetos de partida:	Vaso vacío y jarra vacía
Objetivos:	Vaso lleno y jarra vacía
Métodos	
- proc llenarJarra() Recibe una jarra vacía y la llena de zumo - func llenarJarra() Recibe una jarra vacía y devuelve una nueva jarra llena de zumo - proc llenarVaso() Recibe un vaso vacío y una jarra y llena el vaso con la mitad del contenido de la jarra	
Solución	
Llamar a la función 'llenarJarra' pasando la jarra vacía por valor, con la jarra llena que devuelve se llama a 'llenarVaso' pasando el vaso por referencia, de esta forma se llena el vaso sin modificar el estado inicial de la jarra y se completa el nivel.	

Tabla 4.6: Problema nivel 4

4.5. Historia

El juego no cuenta con una historia o trasfondo concreto, pero su diseño tiene como temática una cafetería. Los menús y pantallas del juego se ajustan también a esta temática para dar sensación de unidad y cohesión entre todos los elementos. Añadiéndole una temática al juego se consigue además hacerlo más atractivo para los jugadores y alumnos.

Todos los objetos del juego hacen referencia a objetos típicos de una cafetería: jarras, tazas, cafeteras, vasos, etc.

Figura 4.2: Objetos del juego

Los métodos que modifican los objetos también se ajustan a esta temática y representan actividades comunes de una cafetería:

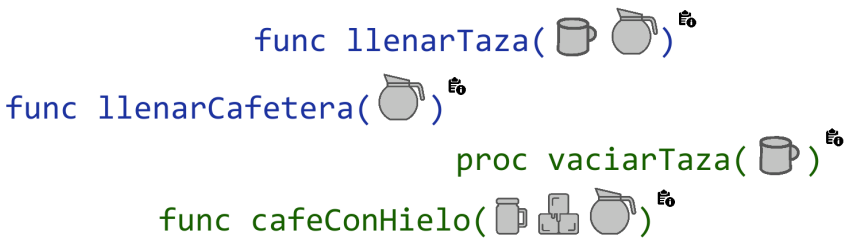

Figura 4.3: Métodos del juego

El objetivo final de cada nivel se representa como un pedido y la puntuación se muestra en el ticket de la comanda:

Figura 4.4: Objetivos del nivel

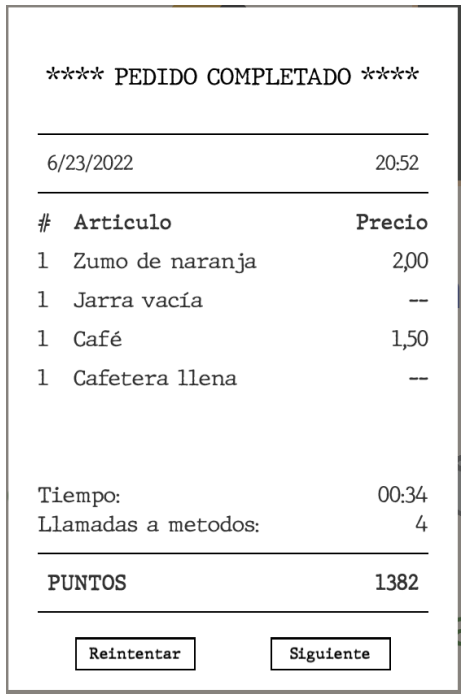

Figura 4.5: Puntuación del nivel

4.6. Multimedia

Para la creación de todos los escenarios se han utilizado varios elementos multimedia: imágenes para los distintos objetos del juego, formas geométricas para crear las pizarras y estanterías y botones y diferentes efectos de sonido para distintos eventos que ocurren durante el juego.

La mayoría de las imágenes utilizadas para los sprites se han descargado de una página de iconos vectoriales [3] que cuentan con licencia gratuita y que posteriormente han sido editados para que se ajusten a los requisitos del proyecto.

Figura 4.6: *Sprites* utilizados

Todos los efectos de sonido se han descargado de una página [9] que cuenta con una base de datos de diferentes efectos de sonido con licencia gratuita. Algunos de ellos se han tenido que editar para ajustarlos a las necesidades del proyecto.

Para la música que se reproduce de fondo durante todo el juego se ha utilizado música [11] sin derechos de autor y que no necesita licencia para ser usada.

4.7. Mundo del juego

El juego está formado por distintos escenarios por los que puede navegar libremente el jugador.

La primera pantalla con la que se encuentra el jugador es el menú principal como se ve en la figura 4.7. Desde aquí el alumno puede seleccionar el nivel al que quiere jugar (Entrantes, Menú I, Menú II, Menú III y Menú IV), también tiene la opción de ver el tutorial en el que se explican las mecánicas del juego o salir al menú de la plataforma desde el botón *exit*.

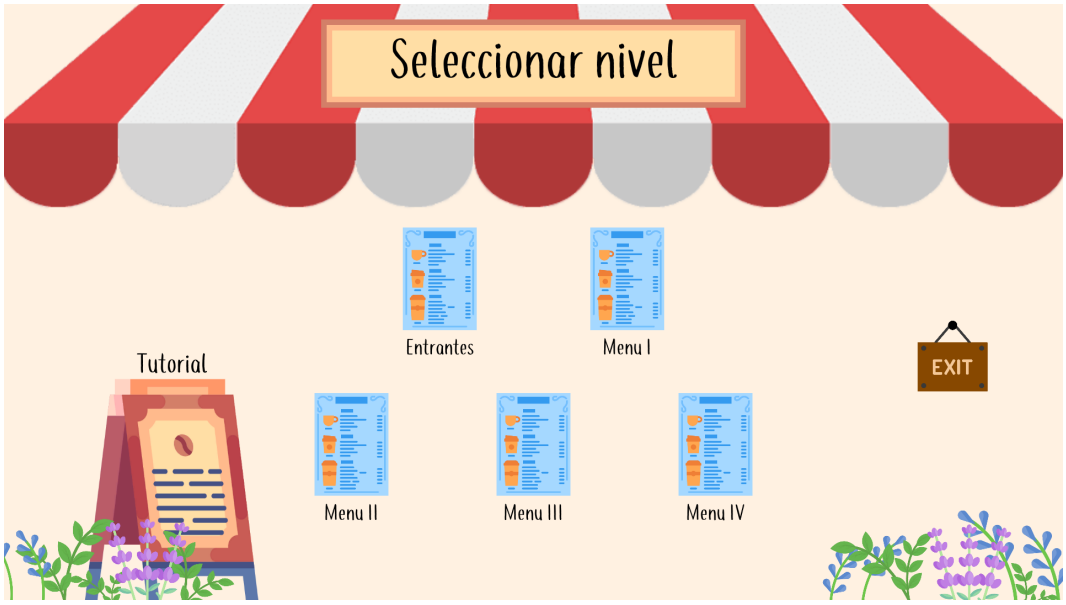

Figura 4.7: Menú inicial

Al seleccionar uno de los menús se carga la pantalla del nivel (figura 6.3) correspondiente en la que se podrá ver arriba a la izquierda en la estantería los objetos iniciales (figura 4.9), arriba a la derecha los objetivos (figura 4.10) y en el centro los métodos necesarios para completar el nivel.

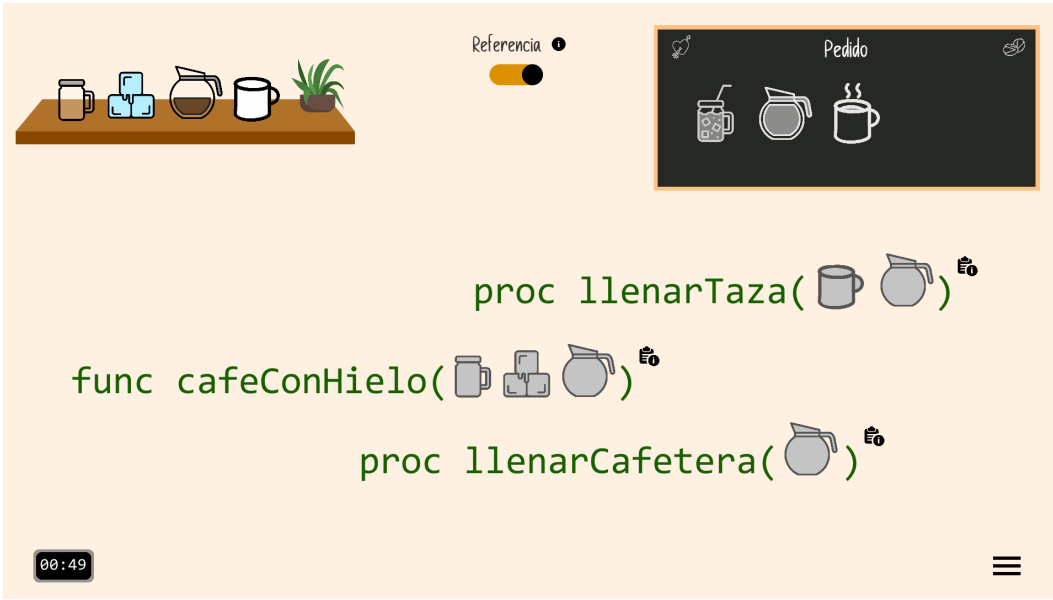

Figura 4.8: Nivel del juego

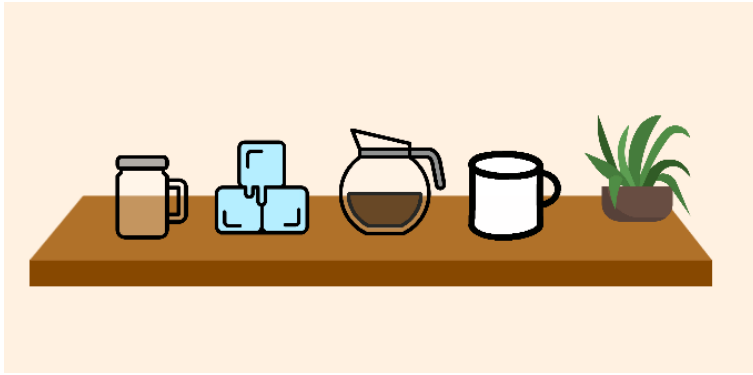

Figura 4.9: Objetos iniciales

Figura 4.10: Objetivos del nivel

Para completar un nivel el jugador deberá arrastrar los objetos de la estantería a los distintos métodos para cambiar su estado, al hacerlo podrá elegir si los utiliza como parámetros o referencia utilizando el botón que aparece arriba en el centro de la pantalla (figura 4.11). Una vez haya conseguido el objeto buscado debe arrastrarlos hasta el pedido para completar el nivel.

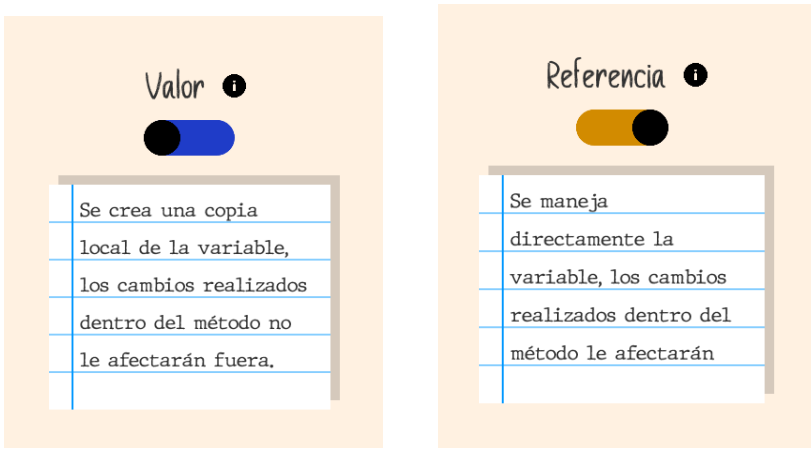

Figura 4.11: Paso por valor y referencia

En todo momento durante la partida se puede ver en la esquina inferior izquierda un reloj que indica el tiempo transcurrido, ese tiempo será el que se utilice después para calcular la

puntuación (figura 4.12).

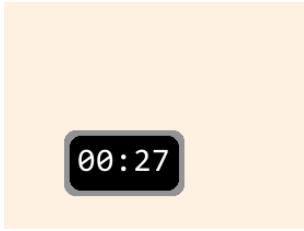

Figura 4.12: Cronometro del nivel

El jugador también puede pausar la partida en cualquier momento utilizando el botón que aparece en la esquina inferior derecha. Esto mostrará un menú en el que aparecerán las opciones de continuar con el nivel, reiniciarlo o volver al menú principal (figura 4.13).

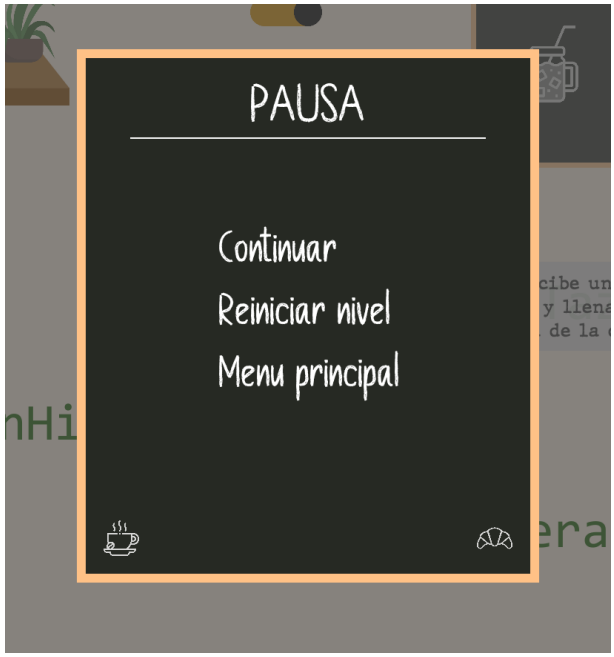

Figura 4.13: Menú de pausa

Una vez el jugador haya completado el nivel se mostrará un resumen del mismo (figura 4.14) en el que se indica el número de pasos y tiempo empleado para la resolución además de los puntos obtenidos. En la parte inferior se mostrarán dos botones para poder reiniciar el

nivel o pasar el siguiente, ambos cargarán una escena igual a la actual en la que se mostrará un problema distinto a resolver con otros objetos iniciales, finales y métodos.

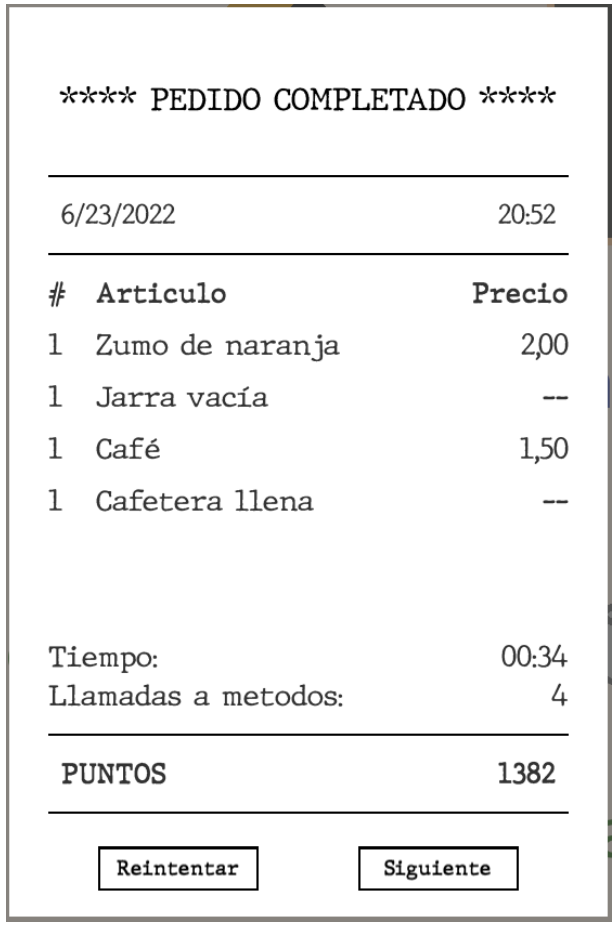

Figura 4.14: Resumen del nivel

Capítulo 5

Análisis

5.1. Elicitación de requisitos

5.1.1. Requisitos funcionales

En esta sección de la tabla 5.1 a la 5.14 se establece la lista de requisitos funcionales que debe satisfacer el juego una vez haya finalizado el desarrollo del proyecto:

RF-01	Iniciar juego
Descripción	Al iniciar el juego este deberá cargar el menú principal mostrando los niveles disponibles con sus puntuaciones, el tutorial y el botón para salir.

Tabla 5.1: RF-01: Iniciar juego

RF-02	Salir del juego
Descripción	El sistema deberá permitir al usuario salir del juego desde el menú principal.

Tabla 5.2: RF-02: Salir del juego

RF-03	Iniciar nivel
Descripción	El sistema deberá permitir al usuario seleccionar y cargar un nivel.

Tabla 5.3: RF-03: Iniciar nivel

RF-04	Salir del nivel
Descripción	El sistema deberá permitir al usuario salir de un nivel en cualquier momento de la partida.

Tabla 5.4: RF-04: Salir del nivel

RF-05	Reiniciar nivel
Descripción	El sistema deberá permitir al usuario reiniciar el nivel en el que se encuentre.

Tabla 5.5: RF-05: Reiniciar nivel

RF-06	Pausar nivel
Descripción	El sistema deberá permitir al jugador pausar un nivel, mostrando en pantalla las opciones de continuar, reiniciar el nivel o volver al menú.

Tabla 5.6: RF-06: Pausar nivel

RF-07	Otorgar y almacenar puntuación
Descripción	El sistema deberá otorgar y almacenar una puntuación por cada nivel superado por el jugador.

Tabla 5.7: RF-07: Otorgar y almacenar puntuación

RF-08	Iniciar tutorial
Descripción	El sistema deberá permitir al jugador iniciar el tutorial del juego.

Tabla 5.8: RF-08: Iniciar tutorial

RF-09	Crear objeto valor
Descripción	El sistema deberá permitir al jugador crear un objeto valor de cualquiera de los objetos de partida de un nivel.

Tabla 5.9: RF-09: Crear objeto valor

RF-10	Crear objeto referencia
Descripción	El sistema deberá permitir al jugador crear un objeto referencia a cualquiera de los objetos de partida de un nivel.

Tabla 5.10: RF-10: Crear objeto referencia

RF-11	Llamar a método
Descripción	El sistema deberá permitir al jugador utilizar los objetos creados, con los distintos métodos del nivel.

Tabla 5.11: RF-11: Llamar a método

RF-12	Completar nivel
Descripción	El sistema deberá permitir al jugador marcar los distintos objetos como completados una vez que tengan el estado deseado para superar el nivel.

Tabla 5.12: RF-12: Completar nivel

RF-13	Mostrar funcionamiento de métodos
Descripción	El sistema deberá permitir al jugador mostrar un texto explicando el funcionamiento de cada método.

Tabla 5.13: RF-13: Mostrar funcionamiento de métodos

RF-14	Mostrar ayuda sobre el paso de parámetros
Descripción	El sistema deberá permitir al jugador mostrar un texto de ayuda en el que se explique el paso por valor y referencia.

Tabla 5.14: RF-14: Mostrar ayuda sobre el paso de parámetros

5.1.2. Requisitos no funcionales

En esta sección de la tabla 5.15 a la 5.19 se establece la lista de requisitos no funcionales que debe satisfacer el juego una vez haya finalizado el desarrollo del proyecto:

RNF-01	Motor del juego
Descripción	El sistema deberá estar desarrollado utilizando el motor de juegos *Unity*.

Tabla 5.15: RNF-01: Motor del juego

RNF-02	Lenguaje
Descripción	El sistema deberá estar desarrollado utilizando el lenguaje de programación *C#*.

Tabla 5.16: RNF-02: Lenguaje

RNF-03	Conexión con la plataforma de despliegue
Descripción	El sistema deberá comunicarse con la plataforma sobre la que será desplegado con la finalidad de guardar y leer estadísticas de cada jugador.

Tabla 5.17: RNF-03: Conexión con la plataforma de despliegue

RNF-04	Compatibilidad
Descripción	El sistema deberá estar desarrollado para ejecutarse en cualquier plataforma compatible con WebGL.

Tabla 5.18: RNF-04: Compatibilidad

RNF-05	Interfaz adaptable
Descripción	El sistema deberá ser capaz de adaptarse a distintas resoluciones y tamaños de pantalla mostrando su interfaz correctamente en todo momento.

Tabla 5.19: RNF-05: Interfaz adaptable

5.2. Casos de uso

En el siguiente diagrama se indican los distintos casos de uso que puede realizar el actor jugador en el sistema.

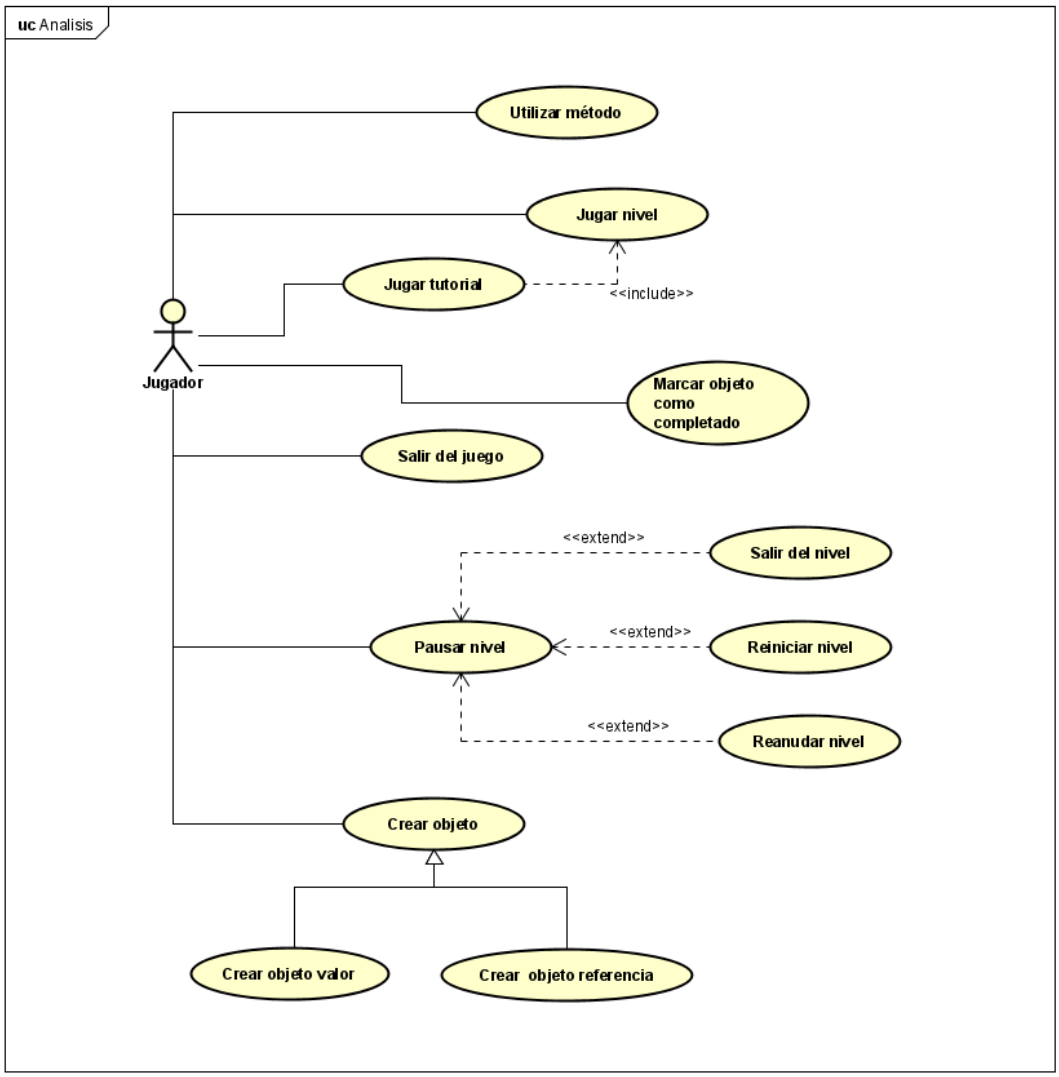

Figura 5.1: Diagrama de casos de uso

Capítulo 6

Diseño e Implementación

6.1. Modelo de Dominio

Como se menciona brevemente en el capítulo **Tecnologías Utilizadas** 3, *Unity* proporciona una serie de clases en la librería *UnityEngine* necesarias para el desarrollo de videojuegos con este motor. Esto conduce a tener que utilizar muchas de sus clases para implementar este proyecto, la principal de estas clases es *MonoBehaviour* [10] y es de la que deben heredar todas las clases que se quiera que interactúen con el motor interno de *Unity*

MonoBehaviour permite al desarrollador, entre otra cosas, el manejo de corrutinas que se ejecutan de manera asíncrona con el resto del código y la gestión de eventos dentro de la partida mediante varios métodos. Los dos principales son:

- *Start* Este método es llamado por el motor cuando el objeto al que pertenece es instanciado en la partida. Podría decirse que funciona a modo de constructor.
- *Update* Este método es llamado por el motor en cada *frame* del juego.

Ha continuación en las figuras 6.1, 6.2 y 6.3 se muestran los diagramas del modelo de dominio de las principales clases de un nivel. Se ha dividido en tres partes para poder ver más en detalle cada elemento:

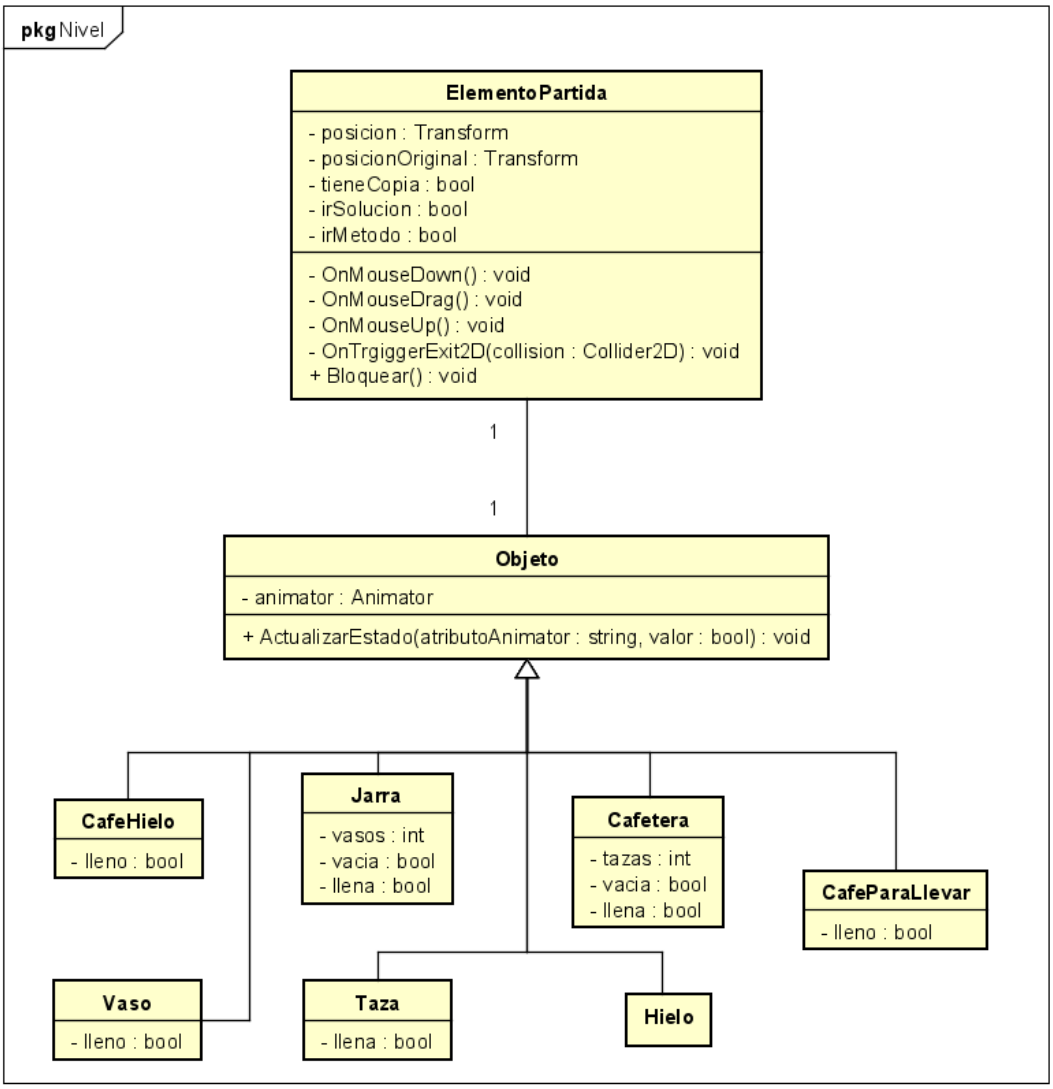

Figura 6.1: Modelo de dominio: Elemento Partida

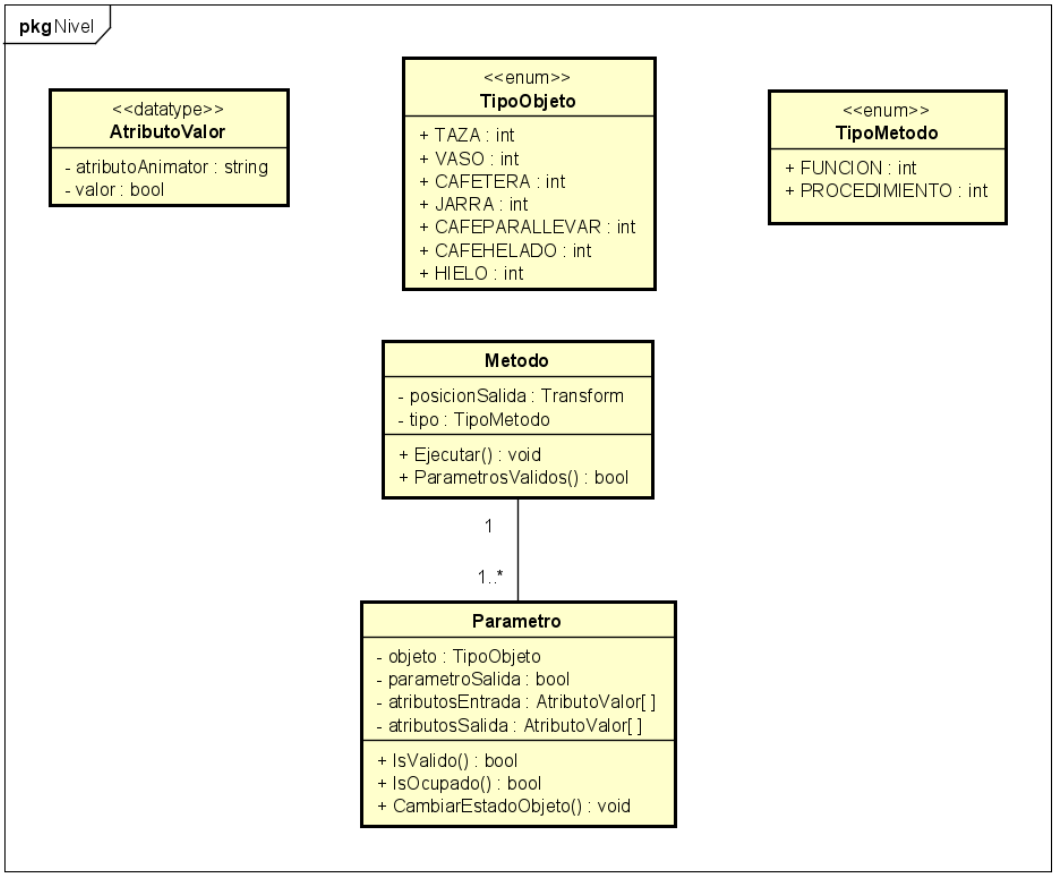

Figura 6.2: Modelo de dominio: Método

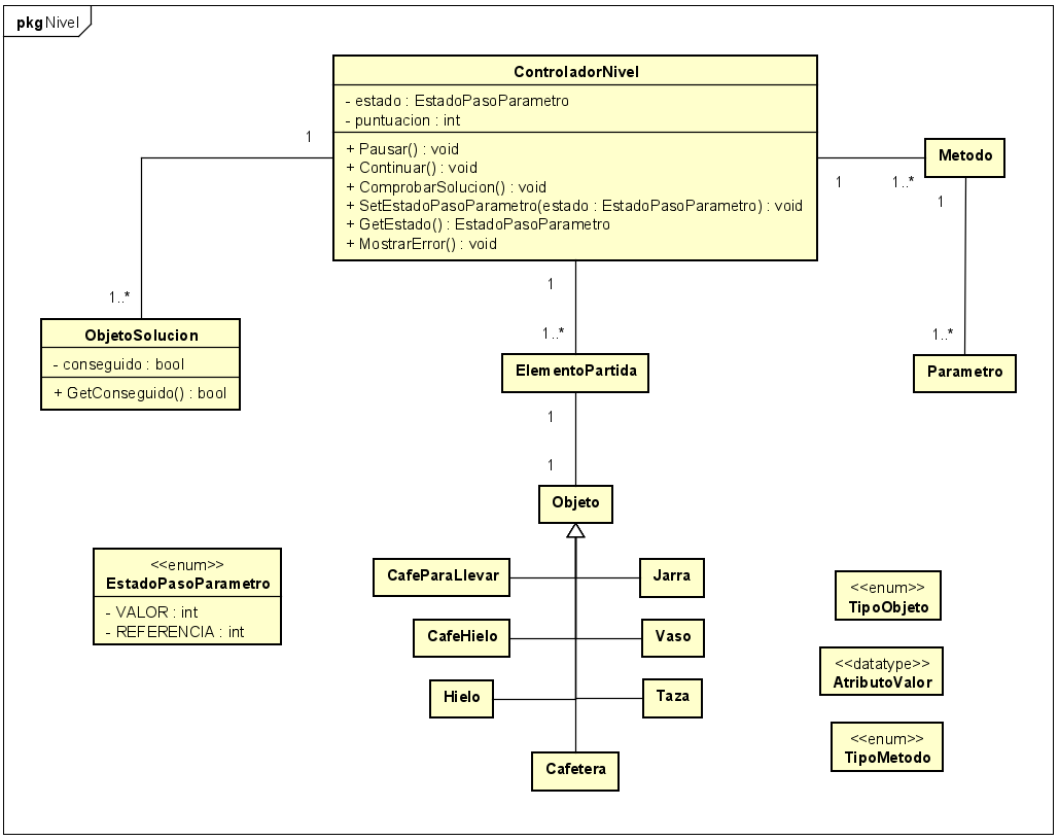

Figura 6.3: Modelo de dominio: Nivel

6.2. Patrones de diseño utilizados

6.2.1. Patrón Observador

Observador [12] es un patrón de diseño de software que define una dependencia entre un objeto y un conjunto de ellos, de modo que los cambios en el primero se vean reflejados en el resto. En el juego este patrón se utiliza para actualizar el estado de los objetos cuando se pasan por referencia.

Cuando un procedimiento se ejecuta habiendo recibido un objeto por referencia, este actualiza el estado únicamente del objeto recibido, ya que es el único que conoce. En ese momento el objeto referencia que se encuentra en el método toma el papel de observador y envía el mensaje a los objetos observados, es decir, el objeto original que se encuentra en la estantería y el resto de referencias al objeto que puedan existir en el nivel, para que actualicen su estado y muestren los cambios en la pantalla.

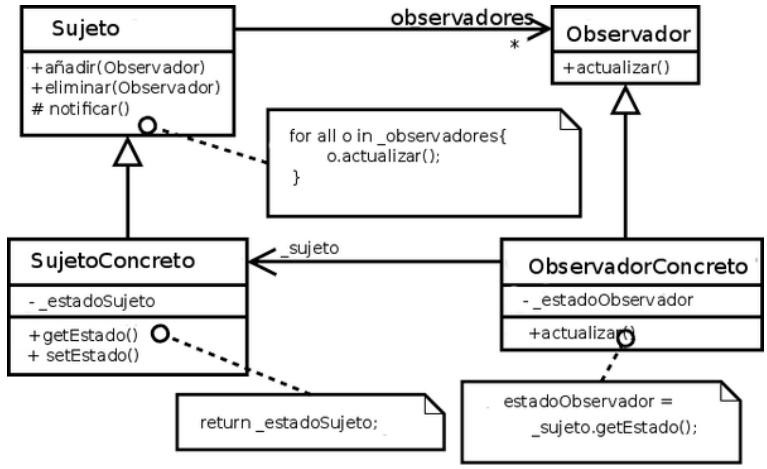

Figura 6.4: Patrón Observador

6.2.2. Patrón *Singleton*

Singleton [13] es un patrón de diseño de software que sirve para asegurarse de que una clase tiene una sola instancia y proporciona un único punto de acceso a ella. Este patrón suele utilizarse cuando se necesita definir una única instancia de cierta clase de modo que sea accesible desde múltiples objetos del sistema y además sea imposible crear de manera accidental varias instancias de la clase accediendo siempre a la misma.

En el juego *Singleton* se utiliza para la clase encargada de controlar todo lo relacionado con los niveles y sus datos comunes. La clase se encarga de manejar los cambios entre escenas

de diferentes niveles y de gestionar los puntos conseguidos por el jugador, también es la clase que se encarga de comunicarse con la plataforma en la que se aloja el juego para que almacene estas puntuaciones.

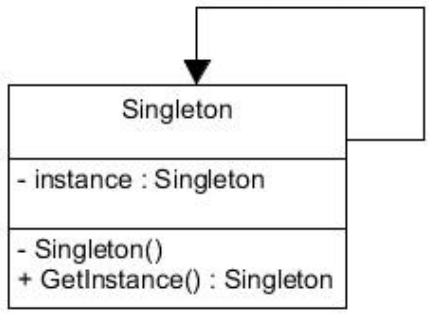

Figura 6.5: Patrón *Singleton*

6.3. Implementación de niveles

6.3.1. Creación de niveles

Todos los niveles del juego están predefinidos en escenas independientes unos de otros, por lo que cada vez que se quiere jugar un nuevo nivel se deberá cargar una nueva escena y cerrar la anterior.

A nivel de eficiencia, esta no es la mejor solución posible, ya que por cada escena dentro del juego este debe almacenar toda la información necesaria para cargar esa escena (*GameObjects*, cámara, gestor de eventos, etc ...). Al tener elementos comunes a cada nivel, como pueden ser elementos del escenario o botones de la interfaz, estos se deberían almacenar varias veces en memoria, resultando en un proyecto demasiado pesado y poco eficiente.

Para solucionar este problema se ha optado por trabajar con *Prefabs* para todos aquellos objetos comunes a cada nivel. De esta forma basta con que el juego almacene una referencia a ese *Prefab* por cada vez que se utilice. El *Prefab* se almacenará una única vez , suponiendo un gran ahorro de memoria y una considerable mejora de rendimiento a la hora de cargar

los recursos de una escena.

Figura 6.6: *Prefabs* de los objetos de un nivel

La principal ventaja de utilizar esta solución frente a otro tipo de implementaciones, como podría ser una creación procedural de los niveles, es la facilidad para crear nuevos problemas para el juego, ya que un código que crease los distintos problemas sería demasiado complejo y difícil de depurar. Con la solución que se ha decidido utilizar basta con crear una nueva escena y añadir los *Prefabs* que se necesiten para crear un nuevo nivel, además de ajustar algún parámetro del nivel como puede ser el límite de tiempo o de pasos para completarlo.

6.3.2. Carga de niveles

Cada nivel está formado por seis problemas, sin embargo, solo es necesario resolver dos o tres de ellos para superar el nivel, de esta forma el jugador puede volver a jugar varias veces cada nivel sin encontrarse las mismas escenas.

Para seleccionar que problema se va a cargar, la clase encargada de los cambios de escena lee todas las escenas disponibles para un nivel que se encuentren en el proyecto y almacena sus nombres en una lista como se ve en el método *ReadScenes*.

```
1     private List<string> ReadScenes(string level)
2     {
3         int sceneCount = SceneManager.sceneCountInBuildSettings;
4         List<string> scenes = new List<string>();
5
6         for (int i = 0; i < sceneCount; i++)
7         {
8             string a = Regex.Replace(SceneUtility.GetScenePathByBuildIndex(i), ".*/",
              ↪  "");
9             a = Regex.Replace(a, "\\.[a-z]+", "");
10            if (a.Contains(level))
11            {
12                scenes.Add(a);
13            }
14        }
15
16        return scenes;
17    }
```

Una vez ha leído y almacenado todas las escenas, se mezclan los elementos dentro de la lista de manera que queden ordenados de forma aleatoria. Cuando el juego necesite cargar un problema de alguno de los niveles buscará la lista que los contenga y cargará el primer elemento, cuando necesite cargar otro problema cogerá el siguiente elemento de la lista y así sucesivamente hasta llegar al último problema, en ese momento se volverá a ordenar la lista de manera aleatoria y la clase volverá a cargar los problemas desde el principio.

De esta forma, además de conseguir que el orden de los problemas sea aleatorio, se asegura que el jugador no vuelva a resolver un problema ya resuelto hasta que no haya jugado todos los que se encuentren disponibles.

6.4. Integración con la plataforma

La plataforma en la que se va a integrar el juego será la encargada de gestionar el arranque y cierre de la aplicación y además guardará un registro de las puntuaciones que obtenga cada jugador en nuestro juego.

Para poder hacer esto correctamente, el juego debe comunicarse con la plataforma mediante llamadas externas. Con estas llamadas externas desde el juego se puede cerrar la aplicación, guardar puntuaciones y cargar y leer cualquier otra información de la que se necesite guardar un registro, como pueden ser niveles desbloqueados o puntuaciones máximas de cada nivel.

6.4.1. Lectura y escritura de puntuaciones

Para guardar y leer datos de la plataforma existen dos métodos: *setCampoLibre* para guardar datos pasando el valor que queramos almacenar (en este caso se ha decidido almacenar las puntuaciones de cada nivel y el ultimo nivel desbloqueado en formato JSON) y para leer los datos guardados se utiliza *getCampoLibre* que devolverá lo último que se haya guardado en el campo libre.

```
1    Application.ExternalCall("setCampoLibre", jsonPoints);
```

```
1    Application.ExternalCall("getCampoLibre");
```

Si se desea guardar la puntuación total que se utiliza para crear los rankings entre jugadores se debe usar el método *guardar*. Al llamarlo se le debe pasar como parámetro un array de dos enteros en el que se indicará en la primera posición la puntuación total del juego sumando todos los niveles y en la segunda posición un 1 si ha superado los 4000 puntos para obtener una estrella en el juego o un 0 en caso de que no lo haya hecho.

```
    int[] data = { total, estrella};

    Application.ExternalCall("guardar", data);
```

Durante una partida el juego va almacenando en una variable local la puntuación obtenida por el jugador y al completar un nivel, si el jugador a mejorado su puntuación máxima, esta se actualiza en la plataforma.

Lo primero que se hace al completar un nivel es consultar las puntuaciones almacenadas, para ello se hace una consulta mediante una llamada externa al método *getCampoLibre*. Esta llamada es asíncrona por lo que no se sabe cuándo va a responder la plataforma, para poder gestionarlo hay que crear un GameObject *OpenField* con una clase llamada también *OpenField*, en la que se implementará el método *getCampoLibre* que será el encargado de recibir la información de respuesta.

```
    public class OpenField : MonoBehaviour
    {
        public void getCampoLibre(string openField)
        {
            /////
        }
    }
```

Una vez se ha recibido la información almacenada, se calcula la puntuación total obtenida en la partida y se compara con la máxima. En caso de que el jugador haya mejorado su puntuación, se almacenará esta nueva puntuación máxima en el campo libre y se actualizará la puntuación total del juego.

Al hacer esto es importante tener en cuenta que si es la primera vez que se guarda una puntuación la llamada *getCampoLibre* lanzará una excepción que hay que capturar.

```
    public void SendPoints(string pointsOpenFile)
    {
        int totOpenFile = 0;
        int totActual = 0;

        try
        {
            LevelPoints p = JsonUtility.FromJson<LevelPoints>(pointsOpenFile);

            for (int i = 0; i < 5; i++)
            {
                totOpenFile += p.points[i];
            }

            for (int i = 0; i < 5; i++)
            {
                totActual += points.points[i];
            }


```

```
            if (points.lastLevel < p.lastLevel)
            {
                points.lastLevel = p.lastLevel;
            }
        }

        catch (Exception e)
        {
            Debug.Log("Datos no validos" + e);
        }

        int total = 0;
        int star = 0;

        for (int i = 1; i < points.points.Length; i++)
        {
            total += points.points[i];
        }

        if (total >= 4000) star = 1;

        int[] data = { total, star };


        if (totActual > totOpenFile)
        {
            string jsonPoints = JsonUtility.ToJson(points);
            Application.ExternalCall("setCampoLibre", jsonPoints);
            Application.ExternalCall("guardar", data);
        }
    }
```

6.4.2. Cierre de la aplicación

Al cerrar la aplicación hay que devolver el control a la plataforma, para que pueda volver a mostrarnos el menú con todos los videojuegos. Esto se hace con la llamada *exit*. También hay que asegurarse de que no quede ningún objeto de *Unity* de fondo eliminándolos con *Destroy*, ya que puede que si no se cierra todo correctamente haya alguna clase que permanezca ejecutándose de fondo.

```
    Application.ExternalCall("salir");

    Application.Quit();
    Destroy(gameObject);
```

Capítulo 7

Pruebas

Además del diseño y la implementación del juego se han hecho diversas pruebas para testear el código del proyecto, verificar su correcto funcionamiento y usabilidad. Se han realizado varios test de unitarios, tests de integración y por último varios tests de usabilidad.

7.1. Pruebas Unitarias

Las pruebas unitarias [16] son una forma efectiva de comprobar el correcto funcionamiento de porciones pequeñas de código. Generalmente se ejecutan sobre las funciones o procedimientos del programa y si se quieren abarcar partes más grandes del código, sobre las distintas clases del proyecto en diseño orientado a objetos. El objetivo de este tipo de pruebas es asegurar que cada unidad funcione de manera correcta y eficiente. Además de verificar que el código efectivamente funciona como se espera, se suelen usar para verificar también los nombres y tipos de los parámetros, tipos que se devuelven y los estados iniciales y finales de una ejecución.

La idea principal de realizar test unitarios, además de comprobar el correcto funcionamiento como ya se ha mencionado, es probar porciones de código no triviales de forma independiente para después con las pruebas de integración asegurar el correcto funcionamiento del programa al completo.

Sobre este proyecto se han realizado las siguientes pruebas unitarias:

PU-01	Cargar Nivel
Descripción	Al seleccionar uno de los niveles en el menú principal, se elige de forma aleatoria y se carga uno de los problemas pertenecientes a ese nivel.
Resultado	Correcto.

Tabla 7.1: PU-01: Cargar Nivel

PU-02	Crear nuevo valor o referencia
Descripción	Al clicar sobre un objeto de partida y arrastrarlo fuera de ella se deberá crear un nuevo objeto en pantalla con las mismas características que el original pero siendo un nuevo valor o referencia de este en función de la opción que haya seleccionado el jugador.
Resultado	Correcto.

Tabla 7.2: PU-02: Crear nuevo valor o referencia

PU-03	Destruir valor
Descripción	Al soltar un objeto valor en la pantalla, si este no se encuentra suficientemente cerca de un método o solución al que pueda ser asignado, este se destruirá en el mismo punto en el que haya sido soltado.
Resultado	Correcto.

Tabla 7.3: PU-03: Destruir valor

PU-04	Destruir referencia
Descripción	Al soltar un objeto referencia en la pantalla, si este no se encuentra suficientemente cerca de un método o solución al que pueda ser asignado, este se desplazara hasta el objeto original al que hace referencia y al llegar a el se destruirá la referencia.
Resultado	Correcto.

Tabla 7.4: PU-04: Destruir referencia

PU-05	Asignar objeto a método
Descripción	Al soltar un objeto en la pantalla, si este se encuentra suficientemente cerca de un método que requiera un objeto con las mismas características que el, se desplazara hasta esa posición y permanecerá a la espera de que el método se ejecute.
Resultado	Correcto.

Tabla 7.5: PU-05: Asignar objeto a método

PU-06	Asignar objeto a solución
Descripción	Al soltar un objeto en la pantalla, si este se encuentra suficientemente cerca de un objeto solución con las mismas características que el, se desplazara hasta la posición de la solución, al llegar la solución pasará a estado completado, el objeto original al que pertenecia el valor o la referencia se bloqueará para no poder volver a ser usado y el objeto que se ha soltado se destruirá.
Resultado	Correcto.

Tabla 7.6: PU-06: Asignar objeto a solución

PU-07	Ejecutar función
Descripción	Cuando a una función le hayan sido asignados todos los parámetros requeridos, se ejecutará destruyendo los objetos recibidos y generando un nuevo objeto que aparecerá al lado de la función esperando a ser utilizado por el jugador.
Resultado	Correcto.

Tabla 7.7: PU-07: Ejecutar función

PU-08	Ejecutar procedimiento
Descripción	Cuando a un procedimiento le hayan sido asignados todos los parámetros requeridos, se ejecutará cambiando el estado de los objetos recibidos que lo necesiten. Devolverá los parámetros pasados por referencia a su posición original y destruirá los objetos pasados por valor.
Resultado	Correcto.

Tabla 7.8: PU-08: Ejecutar procedimiento

PU-09	Completar nivel
Descripción	Al completar todos los objetivos de un problema, este se marcará como completado, calculando y almacenando la puntuación obtenida.
Resultado	Correcto.

Tabla 7.9: PU-09: Completar nivel

PU-10	Pausar nivel
Descripción	Al clicar el botón de pausa se parará el tiempo del nivel, se bloquearán todos los objetos para no poder ser utilizados y se mostrará el menú de pausa con todas sus opciones.
Resultado	Correcto.

Tabla 7.10: PU-10: Pausar nivel

PU-11	Reanudar nivel
Descripción	Al seleccionar la opción de reanudar nivel se ocultará el menú de pausa volviendo a poner en marcha el tiempo y desbloqueando de nuevo todos los objetos.
Resultado	Correcto.

Tabla 7.11: PU-11: Reanudar nivel

PU-12	Reiniciar problema
Descripción	Al seleccionar la opción de reiniciar problema se reiniciarán todos los puntos obtenidos en ese problema hasta el momento y se cargará de forma aleatoria otro de los problemas de el nivel en el que se encuentre el jugador.
Resultado	Correcto.

Tabla 7.12: PU-12: Reiniciar problema

PU-13	Volver al menú
Descripción	Al seleccionar la opción de volver al menú se cargará la escena del menú principal perdiendo todo el progreso del nivel e el que se encontrase el jugador en ese momento.
Resultado	Correcto.

Tabla 7.13: PU-13: Volver al menú

PU-14	Salir del juego
Descripción	Al seleccionar la opción salir, el juego se cerrará por completo devolviendo el control a la plataforma.
Resultado	Correcto.

Tabla 7.14: PU-14: Salir del juego

7.2. Pruebas de integración

Las pruebas de integración [15] son aquellas que se realizan una vez se han completado y aprobado las pruebas unitarias. El objetivo de estas pruebas es asegurarse que los elementos unitarios, que ya se ha comprobado que funcionan de manera independiente, lo hagan también cuando se ejecutan en conjunto.

Las pruebas de integración se centran principalmente en las comunicaciones entre métodos y procesos dentro de un mismo programa. Estos test también se utilizan para comprobar que el sistema se conecta de forma correcta cuando cualquier elemento externo ya sea software o hardware del que requiera el programa para su funcionamiento.

Una vez finalizadas y aprobadas las pruebas unitarias de la sección anterior 7.1, se han realizado diversas pruebas de integración del proyecto:

- **Niveles**: se han jugado niveles completos para comprobar que todos los aspectos relacionados con los mismos funcionaban correctamente al ponerse en conjunto, creación de objetos, paso de parámetros a métodos, completar objetivos, etc ...
- **Navegación por los menús**: se ha navegado por los distintos menús para comprobar que las distintas escenas del juego se cargaban correctamente, así como los distintos niveles. También se ha comprobado que todas las opciones de pausar, reanudar y reiniciar un nivel funcionasen según lo esperado.
- **Conexión con la plataforma web**: se han realizado diversas pruebas una vez que el juego se encontraba desplegado en la plataforma web final para comprobar que las conexiones necesarias para almacenar y leer puntuaciones funcionase correctamente.

7.3. Pruebas Beta

Las pruebas beta [14] se realizan cuando el programa está completo y pasa a ejecutarse en un entorno real. Por muy bueno que haya sido el desarrollo de un proyecto, siempre aparecerán errores que no han sido descubiertos por los desarrolladores ni por el equipo de pruebas. El objetivo de las pruebas beta es encontrara estos errores para corregirlos antes de lanzar la versión final.

Este tipo de pruebas deberían ser realizadas por usuarios finales, ya que serán ellos los que utilicen la versión final del software por lo que pueden proporcionar una mejor retroalimentación a los desarrolladores.

Debido a las fechas en las que se ha realizado el proyecto ha sido imposible realizar este tipo de pruebas con los usuarios a los que va dirigido el videojuego, los alumnos de la asignatura de *Fundamentos de Programación*, sin embargo se han realizado pruebas con otros alumnos del grado en ingeniería informática que se han prestado voluntariamente a probar el sistema.

Las pruebas se han hecho, proporcionando una pequeña introducción previa sobre el funcionamiento de la aplicación a los usuarios. Durante la ejecución se han anotado los distintos resultados obtenidos por cada usuario y todas las dificultades que se han encontrado al usar el programa, con el fin de corregir posibles errores y mejorar la interacción con los usuarios.

Capítulo 8

Conclusiones

Una vez finalizado el proyecto, podemos concluir diciendo que se ha completado de manera exitosa, cumpliendo todos los objetivos y requisitos propuestos al inicio del mismo.

Se ha podido crear un videojuego que gamifica parte del contenido de la asignatura *Fundamentos de Programación* y que servirá para facilitar en un futuro el aprendizaje de los alumnos del grado. El juego ha sido integrado en una plataforma que recoge varios proyectos con las misma finalidad que este, lo que facilita aún más el aprendizaje y fomenta el uso del juego.

También se han realizado las modificaciones, descritas en los objetivos, para los juegos *Fiesta recursiva* y *Caída de Datos*, que ya se encontraban en explotación para la plataforma. Los cambios se han realizado en base a los comentarios recibidos por los alumnos a los que van dirigidos estos juegos.

En cuanto al aprendizaje personal:

- He podido llevar a cabo un proyecto completo de desarrollo de software de principio a fin, pasando por todas las fases que conlleva un desarrollo de estas características.
- He desarrollado una gran cantidad de código en c# aprendiendo un nuevo lenguaje que no había visto durante el grado, aunque si que he podido aplicar muchos conocimientos vistos sobre lenguajes orientados a objetos y diseño de software vistos en algunas asignaturas.
- He seguido buenas prácticas, tanto en el diseño como en la implementación del proyecto, además de realizar los test pertinentes para asegurar la calidad de todos los elementos del proyecto.
- He podido utilizar y aprender sobre nuevas tecnologías como son el motor *Unity* y algunas de las herramientas utilizadas para la parte multimedia del proyecto para la edición de imágenes y audio.

8.1. Líneas de trabajo futuras

Pese a que el resultado final es bastante completo y se ajusta al resultado esperado, durante el desarrollo del proyecto se han observado ciertos aspectos que se podrían mejorar o en los que se podría seguir trabajando en un futuro para crear una aplicación de mayor calidad.

- Aumento en la cantidad de niveles, ampliando y mejorando el contenido ya existente. Pese a que el juego cuenta con un total de 26 niveles cuando se juega varias veces puede llegar a sentirse repetitivo y nuevos niveles podrían solucionar parte de este problema.
- Exportar el juego a distintas plataformas como pueden ser dispositivos móviles, realizando los cambios necesarios en los controles y eventos del juego para adaptarse a los nuevos sistemas.
- Introducir nuevos objetos en los problemas del juego que añadan distintas mecánicas como puede ser el uso de constantes.
- Mejorar el apartado multimedia añadiendo nuevos efectos de sonido o mejorando las animaciones y transiciones existentes.

Bibliografía

[1] Audacity. https://audacity.es/. Accessed: 10-6-2022.

[2] C. https://es.wikipedia.org/wiki/C_Sharp. Accessed: 21-6-2022.

[3] Flaticon. https://www.flaticon.es. Accessed: 12-3-2022.

[4] Gamificación. https://www.educaciontrespuntocero.com/noticias/gamificacion-que-es-objetivos/. Accessed: 15-5-2022.

[5] Gimp. http://www.gimp.org.es/descargar-gimp.html. Accessed: 27-5-2022.

[6] Git. https://git-scm.com/. Accessed: 12-6-2022.

[7] Github. https://github.com/. Accessed: 12-6-2022.

[8] Las 17 superproducciones más caras de la historia de los videojuegos. https://www.vidaextra.com/listas/videojuegos-caros-historia. Accessed: 21-6-2022.

[9] mixkit. https://mixkit.co/. Accessed: 12-3-2022.

[10] Monobehaviour. https://docs.unity3d.com/2020.3/Documentation/Manual/class-MonoBehaviour.html. Accessed: 15-3-2022.

[11] Música ambiente. https://www.youtube.com/watch?v=xXl7HEVnNiY. Accessed: 12-3-2022.

[12] Patrón observador. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiZs5j9vLr4AhUF-YUKHScOBkUQFnoECAwQAQ&url=https%3A%2F%2Fwww.infor.uva.es%2F~felix%2Fdatos%2Fpriii%2Ftr_patrones-2x4.pdf&usg=AOvVaw2xkR4PyDPR0vVGUgWPs07i. Accessed: 19-6-2022.

[13] Patrón singleton. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiZs5j9vLr4AhUF-YUKHScOBkUQFnoECAwQAQ&url=https%3A%2F%2Fwww.infor.uva.es%2F~felix%2Fdatos%2Fpriii%2Ftr_patrones-2x4.pdf&usg=AOvVaw2xkR4PyDPR0vVGUgWPs07i. Accessed: 19-6-2022.

[14] Pruebas beta. https://es.wikipedia.org/wiki/Pruebas_beta. Accessed: 07-6-2022.

[15] Pruebas de integración. https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/integracion/. Accessed: 07-6-2022.

[16] Pruebas unitarias. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-papel-del-unit-test-en-el-desarrollo-de-software/. Accessed: 07-6-2022.

[17] Sector de los videojuegos. https://www.eleconomista.es/ecoaula/noticias/11832022/06/22/El-sector-del-videojuego-se-prepara-para-entrar-en-el-mundo-de-la-e
html. Accessed: 21-6-2022.

[18] Sueldo medio programador junior. https://www.glassdoor.es/Sueldos/programador-junior-sueldo-SRCH_KO0,18.htm. Accessed: 27-2-2022.

[19] Unity manual. https://docs.unity3d.com/2020.3/Documentation/Manual/. Accessed: 2-4-2022.

[20] Unity plans and pricing for developers. https://store.unity.com/#plans-individual. Accessed: 15-3-2022.

[21] Unity technologies. https://unity.com/es. Accessed: 15-3-2022.

[22] Visual studio. https://es.wikipedia.org/wiki/Microsoft_Visual_Studio. Accessed: 12-6-2022.

[23] Bob Hughes and Mike Cotterell. *Software Project Management*. The McGraw-Hill Companies, 2009.