**Plataforma para recreación de estrategia basada en aprendizaje reforzado**

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Logotipo

Descripción generada automáticamente

Julio de 2023

Autor

Erick José Mercado Hernández

Tutor/a

Vidal Moreno Rodilla

# Lista de cambios

|  |  |  |  |
| --- | --- | --- | --- |
| **Número** | **Fecha** | **Versión** | **Autor** |
| 0 | 02-02-2023 | Versión 0.1 (Creación del documento) | Erick José Mercado Hernández |
| 1 |  | Versión 1.0 (Desarrollo del documento) | Erick José Mercado Hernández |

# Resumen

Este proyecto se centra en el diseño e implementación de una plataforma para la recreación de estrategias basada en aprendizaje supervisado, con el objetivo de proporcionar a los desarrolladores y aficionados a los juegos de estrategia una herramienta eficiente para analizar, mejorar y adaptar sus enfoques estratégicos en tiempo real. La inteligencia artificial (IA) y el aprendizaje automático (ML) han experimentado un crecimiento exponencial en los últimos años, demostrando resultados prometedores en la industria del videojuego.

El proyecto incluye una revisión exhaustiva de conceptos teóricos y técnicos relacionados con el aprendizaje supervisado y su aplicación en juegos de estrategia. A partir de esta base teórica, se investigan plataformas y herramientas existentes, identificando áreas de mejora y limitaciones actuales. La plataforma propuesta se diseñará considerando escalabilidad, modularidad y facilidad de uso para los usuarios, ofreciendo módulos y componentes que faciliten la integración de modelos de IA, recolección de datos y la implementación de estrategias basadas en aprendizaje supervisado.

Tras el desarrollo de la plataforma, se realizarán evaluaciones de rendimiento y funcionalidad en diversos escenarios y juegos de estrategia previamente creados como casos de estudio. Se analizará el impacto de la integración del aprendizaje supervisado en la generación y adaptación de estrategias, así como la eficiencia y precisión de los modelos de IA implementados.

En resumen, este TFG busca desarrollar una plataforma innovadora para la recreación de estrategias basadas en aprendizaje supervisado, ofreciendo a los desarrolladores y entusiastas de los juegos de estrategia una herramienta poderosa para mejorar sus habilidades, diseñar estrategias más efectivas y experimentar con modelos de IA de última generación. La plataforma propuesta tiene el potencial de cambiar la forma en que los jugadores abordan los juegos de estrategia y abrir nuevas oportunidades para la investigación y el desarrollo en este campo.

# Abstract

This Final Degree Project (TFG) focuses on the design and implementation of a platform for strategy recreation based on supervised learning, aiming to provide developers and strategy game enthusiasts with an efficient tool to analyse, enhance, and adapt their strategic approaches in real-time. Artificial intelligence (AI) and machine learning (ML) have experienced exponential growth in recent years, and their application in the video game industry has shown promising results.

The project encompasses a comprehensive review of theoretical and technical concepts related to supervised learning and its application in strategy video games. Based on this theoretical foundation, existing platforms and tools are investigated to identify areas for improvement and current limitations. The proposed platform will be designed considering scalability, modularity, and ease of use for users, offering modules and components that facilitate the integration of AI models, data collection, and the implementation of supervised learning-based strategies.

Following the platform's development, performance and functionality evaluations will be conducted in various scenarios and previously created strategy games as case studies. The impact of supervised learning integration on strategy generation and adaptation will be analysed, along with the efficiency and accuracy of the implemented AI models.

In summary, this TFG aims to develop an innovative platform for strategy recreation based on supervised learning, offering developers and strategy game enthusiasts a powerful tool to improve their skills, design more effective strategies, and experiment with innovative AI models. The proposed platform has the potential to change how players approach strategy games and open new opportunities for research and development in this field.

Índice

[1.- Introducción 1](#_Toc138422811)

[2.- Objetivos del proyecto 2](#_Toc138422812)

[3.- Conceptos teóricos 3](#_Toc138422813)

[3.1.- Algoritmo A\* 3](#_Toc138422814)

[3.2.- Algoritmo de Montecarlo 5](#_Toc138422815)

[3.3.- Aprendizaje reforzado 7](#_Toc138422816)

[3.4.- Inteligencia Artificial En TBRPG 8](#_Toc138422817)

[4.- Metodología, técnicas y herramientas 8](#_Toc138422818)

[4.1.- Metodología 9](#_Toc138422819)

[4.2.- Técnicas 13](#_Toc138422820)

[4.2.1.- Patrón Singleton 13](#_Toc138422821)

[4.2.2.- Metodología para la Elicitación de Requisitos de Sistemas de Software de Durán y Bernárdez. 13](#_Toc138422822)

[4.3.- Herramientas 14](#_Toc138422823)

[4.3.1.- Motor gráfico 14](#_Toc138422824)

[4.3.2.- Lenguaje de programación 15](#_Toc138422825)

[4.3.3.- Herramientas auxiliares 18](#_Toc138422826)

[5.- Aspectos relevantes del desarrollo 20](#_Toc138422827)

[5.1.- Proceso de desarrollo 20](#_Toc138422828)

[5.1.1.- Investigación 20](#_Toc138422829)

[5.1.2.- Especificación de requisitos 25](#_Toc138422830)

[5.1.3.- Análisis y diseño 30](#_Toc138422831)

[5.1.4.- Implementación 34](#_Toc138422832)

[6.- Resultados 34](#_Toc138422833)

[6.1.- Resultados y conclusiones 34](#_Toc138422834)

[6.2.- Líneas de trabajo futuras 35](#_Toc138422835)

[Bibliografía 36](#_Toc138422836)

[Tabla 1: Resumen de los requisitos funcionales 26](#_Toc138424126)

[Tabla 2: Resumen de requisitos de información 28](#_Toc138424127)

[Tabla 3: Resumen de requisitos no funcionales 29](#_Toc138424128)

[Ilustración 1: Explicación algoritmo A\* 3](#_Toc138424182)

[Ilustración 2: Representación distancia de Manhattan 4](#_Toc138424183)

[Ilustración 3: Representación de la distancia de Chevyshev 5](#_Toc138424184)

[Ilustración 4: Pseudocódigo de ejemplo del algoritmo de Montecarlo 6](#_Toc138424185)

[Ilustración 5: Esquema de la simulación 6](#_Toc138424186)

[Ilustración 6: Representación del aprendizaje reforzado 7](#_Toc138424187)

[Ilustración 7:Diagrama de Gantt (I) 10](#_Toc138424188)

[Ilustración 8: Diagrama de Gantt (II) 10](#_Toc138424189)

[Ilustración 9: Diagrama de Gantt (III) 11](#_Toc138424190)

[Ilustración 10: Diagrama de Gantt (IV) 11](#_Toc138424191)

[Ilustración 11: Diagrama de Gantt (V) 12](#_Toc138424192)

[Ilustración 12: Diagrama de Gantt (VI) 12](#_Toc138424193)

[Ilustración 13: Pseudocódigo algoritmo de Montecarlo 22](#_Toc138424194)

[Ilustración 14: Ejemplo de simulación del algoritmo 23](#_Toc138424195)

[Ilustración 15: Diagrama de casos de uso 25](#_Toc138424196)

[Ilustración 16: Diagrama de clases I 30](#_Toc138424197)

[Ilustración 17: Diagrama de clases II 31](#_Toc138424198)

[Ilustración 18: Diagrama de clases III 31](#_Toc138424199)

[Ilustración 19: Diagrama de clases IV 32](#_Toc138424200)

# 1.- Introducción

# 2.- Objetivos del proyecto

Este proyecto tiene como propósito diseñar e implementar una plataforma para la recreación de estrategias basada en aprendizaje supervisado en el ámbito de los juegos de estrategia. Los objetivos específicos del proyecto son los siguientes:

* Investigar y analizar los conceptos teóricos y técnicos fundamentales relacionados con el aprendizaje supervisado y su aplicación en el desarrollo de juegos de estrategia, identificando las técnicas y enfoques más relevantes en este campo.
* Estudiar y evaluar las plataformas y herramientas existentes para la recreación y análisis de estrategias en juegos de estrategia, identificando áreas de mejora y limitaciones en las soluciones actuales.
* Diseñar la arquitectura y los componentes de la plataforma propuesta, considerando aspectos clave como la escalabilidad, modularidad y facilidad de uso para los desarrolladores y aficionados a los juegos de estrategia.
* Desarrollar e implementar módulos y componentes que faciliten la integración de modelos de IA basados en aprendizaje supervisado, la recolección de datos y la implementación de estrategias en la plataforma propuesta.
* Realizar pruebas y evaluaciones de rendimiento y funcionalidad de la plataforma en diversos escenarios y juegos de estrategia previamente creados como casos de estudio, analizando el impacto de la integración del aprendizaje supervisado en la generación y adaptación de estrategias.
* Analizar la eficiencia y precisión de los modelos de IA implementados en la plataforma, evaluando su capacidad para mejorar y adaptar estrategias en tiempo real en función de las acciones del jugador.
* Documentar y difundir los resultados del proyecto, proporcionando información detallada sobre la plataforma desarrollada, sus características y funcionalidades, así como las conclusiones y futuras líneas de investigación derivadas del proyecto.

# 3.- Conceptos teóricos

En este apartado se va a explicar los conceptos teóricos de manera que se pueda tener una base sobre el campo en el que se va a desarrollar el proyecto y así poder entender en mayor medida como implementarlo.

## 3.1.- Algoritmo A\*

El algoritmo A\* es un popular algoritmo de búsqueda en grafos utilizado en una variedad de aplicaciones, desde la inteligencia artificial en videojuegos hasta la planificación de rutas en sistemas de GPS. A\* es especialmente conocido por su capacidad para encontrar el camino más corto entre dos nodos o puntos en un mapa o un grafo.

Este algoritmo fue introducido por primera vez por Peter Hart, Nils Nilsson y Bertram Raphael en 1968. A\* es una versión mejorada del algoritmo de Dijkstra y utiliza heurísticas para guiar su búsqueda, lo que lo hace más eficiente en muchos casos.

El funcionamiento de A\* se basa en mantener dos listas: una lista abierta y una lista cerrada. La lista abierta contiene los nodos que están siendo considerados para la búsqueda, mientras que la lista cerrada contiene los nodos que ya han sido visitados.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

Ilustración : Explicación algoritmo A\*

Cada nodo en el grafo tiene tres valores importantes asociados a él:

* g(n): es el costo real para llegar a este nodo desde el nodo de inicio.
* h(n): es la estimación heurística del costo desde este nodo hasta el nodo objetivo.
* f(n): es la suma de g(n) y h(n).

El proceso básico del algoritmo A\* es el siguiente:

1. Comienza con el nodo inicial en la lista abierta.
2. Selecciona el nodo de la lista abierta con el valor f(n) más bajo.
3. Si este nodo es el nodo objetivo, entonces el algoritmo ha encontrado el camino y se detiene.
4. Si no es el nodo objetivo, se mueve a la lista cerrada y se añaden todos sus nodos vecinos a la lista abierta.
5. Para cada nodo vecino, se calcula su valor g(n) y se actualiza su valor f(n).
6. Este proceso se repite volviendo al paso 2, hasta que se encuentre el nodo objetivo o no queden nodos en la lista abierta.

Un aspecto crucial del algoritmo A\* es la elección de la función heurística h(n). Para garantizar que A\* encuentre el camino más corto, la función heurística debe ser admisible, es decir, nunca debe sobrestimar el costo para llegar al objetivo. Un ejemplo común de una función heurística admisible es la distancia en línea recta desde un nodo hasta el nodo objetivo en un mapa 2D.

La distancia Manhattan, también conocida como distancia de ciudad o distancia L1, es una medida de la distancia entre dos puntos en un sistema de coordenadas basado en una cuadrícula (como un tablero de ajedrez o un mapa de la ciudad de Manhattan, de donde toma su nombre).

En el contexto del algoritmo A\*, la distancia Manhattan puede ser utilizada como la función heurística h(n) que mencioné anteriormente. Esto es particularmente útil cuando el movimiento está limitado a una cuadrícula y no se permiten movimientos diagonales.

La distancia Manhattan entre dos puntos es la suma de las diferencias absolutas de sus coordenadas. Por ejemplo, si tienes dos puntos, y , la distancia Manhattan entre estos dos puntos se calcula como:

Imagen que contiene edificio, biombo

Descripción generada automáticamente

Ilustración : Representación distancia de Manhattan

La distancia de Chebyshev se usa a menudo como una de estas heurísticas. Es un sistema de medición que permite los movimientos diagonales, a diferencia de la distancia de Manhattan que solo permite movimientos horizontales y verticales. En un juego de cuadrícula, por ejemplo, si se permite el movimiento en diagonal, la distancia de Chebyshev podría ser una elección más adecuada para la heurística.

La distancia de Chebyshev entre dos puntos y se calcula como:

Esto significa que la distancia de Chebyshev es el mayor valor de la diferencia absoluta entre las coordenadas x o y de los dos puntos. Esto permite movimientos diagonales, ya que un movimiento diagonal implica cambiar tanto las coordenadas x como y al mismo tiempo.

Entonces, en términos de la implementación del algoritmo A\*, la distancia de Chebyshev se usaría para calcular la función heurística h(n), que estima el costo del camino más corto desde el nodo n hasta el objetivo.

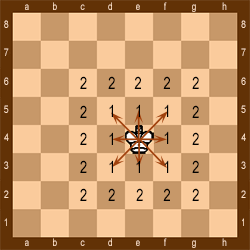


Ilustración : Representación de la distancia de Chevyshev

## 3.2.- Algoritmo de Montecarlo

El método de Montecarlo es una técnica computacional que se basa en el uso de números aleatorios y probabilidades para resolver problemas complejos. El nombre "Montecarlo" se debe al famoso casino en Mónaco, reflejando el elemento aleatorio intrínseco del método.

El método de Montecarlo es útil en una variedad de contextos, especialmente cuando el problema es demasiado complejo para resolverlo con métodos analíticos. Se utiliza en física, matemáticas, economía, inteligencia artificial, juegos, y más.

Este método se basa en la generación de un gran número de resultados aleatorios (o "muestras") y luego se analiza el conjunto de resultados para hacer estimaciones sobre la solución del problema.

El método de Montecarlo también se utiliza para resolver problemas más complejos, como la integración en múltiples dimensiones, la simulación de sistemas físicos, y la planificación y toma de decisiones en IA y juegos. En estos contextos, el método de Montecarlo ofrece una forma de explorar y hacer estimaciones sobre un espacio de posibilidades muy grande y complejo.

En inteligencia artificial y juegos, el método de Montecarlo se utiliza de manera extensiva en la toma de decisiones y planificación. En el juego de Go, por ejemplo, la famosa IA AlphaGo de DeepMind utiliza una variante de este método, llamada Monte Carlo Tree Search (MCTS), para explorar posibles secuencias de movimientos y determinar la mejor acción a seguir. Este enfoque ha demostrado ser muy efectivo, permitiendo a AlphaGo vencer a campeones humanos del juego.

El método de Montecarlo también se utiliza en el campo de la optimización, donde se busca encontrar la solución óptima a un problema en un espacio de soluciones posibles. Por ejemplo, puede ser usado para optimizar los parámetros de un modelo de aprendizaje automático, o para encontrar la mejor ruta en un problema de viajante de comercio.

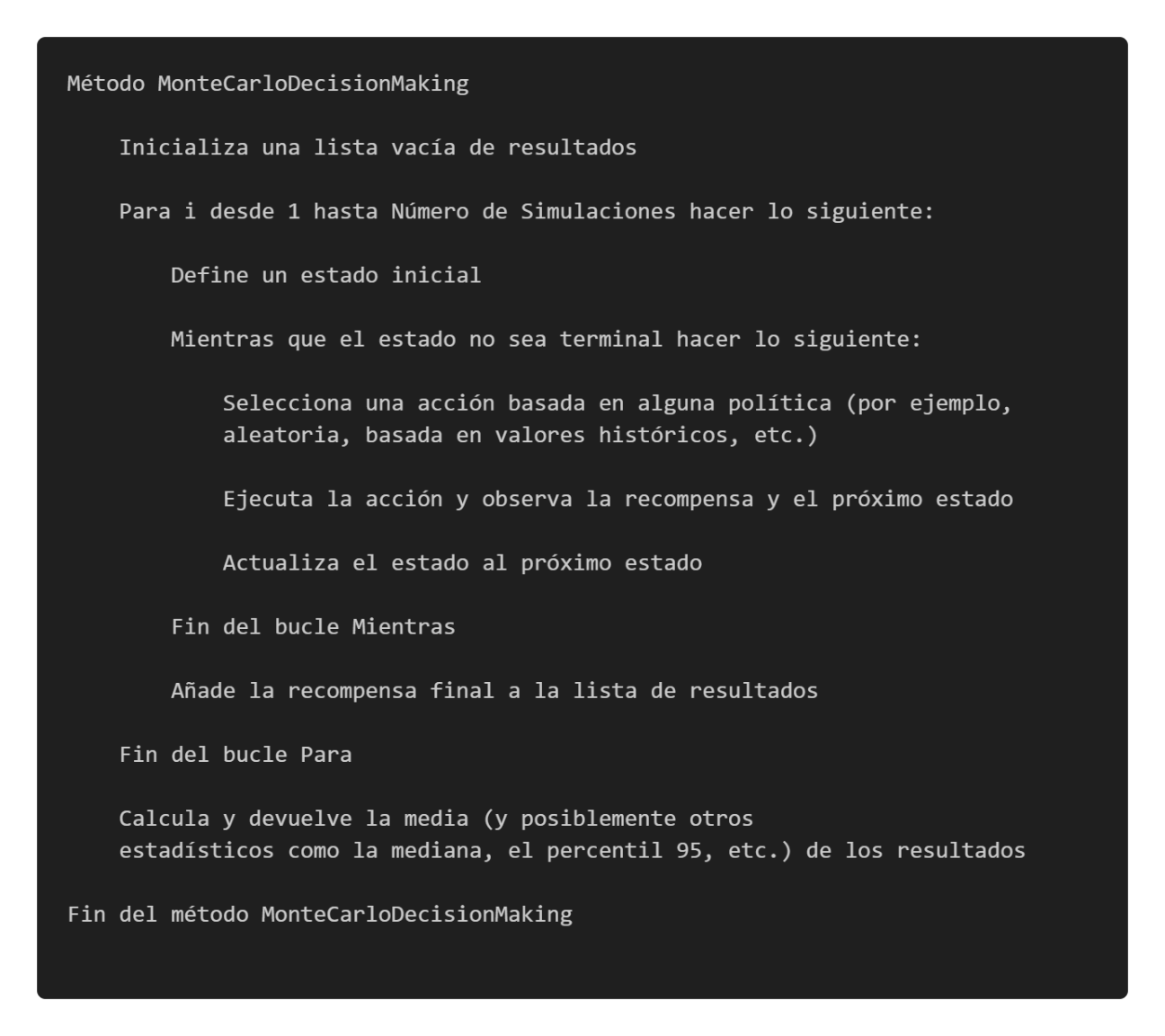


Ilustración : Pseudocódigo de ejemplo del algoritmo de Montecarlo

Diagrama

Descripción generada automáticamente

Ilustración : Esquema de la simulación

## 3.3.- Aprendizaje reforzado

El aprendizaje por refuerzo es una rama de la inteligencia artificial y del aprendizaje automático que se centra en cómo un agente puede aprender a tomar decisiones óptimas en un entorno, dado un sistema de recompensas y castigos. Este tipo de aprendizaje se inspira en cómo los seres vivos aprenden de su entorno y sus interacciones.

En el aprendizaje por refuerzo, un agente toma acciones en un entorno, y por cada acción recibe una recompensa o un castigo (una señal de "refuerzo"). El objetivo del agente es aprender una política, que es una estrategia de toma de decisiones que maximiza la recompensa acumulada a lo largo del tiempo.

Diagrama

Descripción generada automáticamente

Ilustración : Representación del aprendizaje reforzado

El aprendizaje por refuerzo se puede describir con el siguiente ciclo:

1. El agente observa el estado actual del entorno.
2. Basado en esta observación, el agente selecciona y ejecuta una acción.
3. El entorno cambia de estado, y el agente recibe una recompensa o un castigo.
4. El agente utiliza esta retroalimentación para actualizar su política.

Este ciclo se repite, con el agente mejorando continuamente su política a medida que adquiere más experiencia en el entorno.

Un aspecto crucial del aprendizaje por refuerzo es el equilibrio entre la exploración y la explotación. La exploración se refiere a probar acciones que el agente no ha intentado o ha intentado raramente, para ver si conducen a una mayor recompensa. La explotación se refiere a usar la política actual del agente para tomar la acción que, según su experiencia, maximizará la recompensa. Ambas son necesarias para que un agente aprenda una política óptima.

## 3.4.- Inteligencia Artificial En TBRPG

Los juegos de rol tácticos basados en turnos (TBRPG) ofrecen un entorno complejo y desafiante para la inteligencia artificial (IA). En estos juegos, los jugadores y la IA toman turnos para mover personajes por un mapa de cuadrícula y realizar acciones, como atacar o usar objetos. Para jugar de manera eficaz, la IA debe tomar decisiones estratégicas y tácticas basadas en la información del estado del juego y las acciones de los jugadores.

1. **Toma de decisiones:** En un TBRPG, la IA debe tomar decisiones en cada turno sobre qué personajes mover y qué acciones realizar. Esto puede implicar la evaluación de diferentes opciones y la elección de la que maximice un objetivo, como minimizar el daño a los personajes controlados por la IA o maximizar el daño a los personajes del jugador. La IA puede utilizar algoritmos como MiniMax, Alpha-Beta Pruning, o el algoritmo de Montecarlo para explorar el árbol de decisiones y seleccionar la mejor acción.
2. **Planificación de la ruta:** La IA debe calcular las rutas eficientes para mover a sus personajes por el mapa de cuadrícula. El algoritmo A\* es comúnmente utilizado para esto, a menudo con la distancia de Manhattan o la distancia de Chebyshev como heurística cuando los movimientos diagonales están permitidos.
3. **Aprendizaje y adaptación:** La IA puede utilizar técnicas de aprendizaje automático para adaptarse a las tácticas del jugador y mejorar su rendimiento a lo largo del tiempo. Por ejemplo, podría usar el aprendizaje por refuerzo para aprender una política que maximice la recompensa a largo plazo, basándose en una función de recompensa que favorezca los estados de juego en los que la IA está en una posición ventajosa.
4. **Evaluación del estado del juego:** La IA debe evaluar el estado actual del juego para tomar decisiones informadas. Esto puede implicar el uso de funciones de evaluación que asignen un valor a cada estado del juego, basado en factores como la salud de los personajes, la posición estratégica y el número de acciones disponibles.
5. **Generación de contenido procedimental:** En algunos TBRPGs, la IA también puede estar involucrada en la generación de contenido procedimental, como la creación de niveles o la selección de encuentros de enemigos, para proporcionar una experiencia de juego variada y desafiante.

# 4.- Metodología, técnicas y herramientas

En este apartado se van a presentar las metodologías, las técnicas y las herramientas que se han utilizado durante el desarrollo del proyecto.

Para comenzar, se va a explicar la metodología utilizada siguiendo el proceso unificado, se continuará explicando las técnicas y patrones utilizados entre los que se encuentran el patrón Singleton o la metodología para la elicitación de requisitos de Duran y Bernárdez y para finalizar se expondrán las herramientas relevantes usadas, en las que consta el motor grafico de UNITY, el lenguaje de programación y herramientas auxiliares.

## 4.1.- Metodología

El desarrollo del proyecto va a realizarse siguiendo el Proceso Unificado del Desarrollo Software. El proceso unificado del Desarrollo Software es un enfoque para el desarrollo de software que se basa en la iteración y la adaptación. Este se centra en la entrega continua de software funcional y tiene como principal marco un ciclo de vida de desarrollo de software.

En este proceso es un marco de trabajo o extensible que puede ser adaptado a organizaciones o proyectos. En lo respectivo al ciclo de vida, este se puede enmarcar en un ciclo iterativo e incremental, lo cual se refiere a que hay que realizar varias iteraciones las cuales están basadas en las etapas clásicas del Proceso Unificado del Desarrollo de Software. Estas etapas, las cuales podemos definir como modelado del negocio, requisitos, análisis, diseño, implementación, pruebas y despliegue, se van a ir refinando y formando poco a poco para llegar a obtener un producto software completo.

Este tipo de ciclo de vida se ha escogido debido a que el tiempo estimado para el proyecto, que son 8 meses es un poco ajustado, el desarrollo únicamente lo va a implementar una persona y el proyecto contiene diversos módulos con tecnologías muy diferentes que hay que investigar, diseñar e implementar.

Para comenzar, se ha elaborado una planificación temporal que se adapta al ciclo de vida y a las tecnologías que se van a utilizar en cada momento obteniendo las siguientes iteraciones:

* **Iteración inicial**: En esta iteración se va a realizar una investigación del estado del arte de cada una de las funcionalidades del proyecto, observar que tecnologías y arquitecturas proponen soluciones similares y así poder elaborar una primera propuesta simplificada de los requisitos del análisis y del diseño del sistema para así poder establecer mejor que componentes forman parte del proyecto y que van a realizar cada uno.
* **Iteración controles de cámara e iteración con el entorno:** En esta iteración se lleva a cabo el control de la cámara del juego, así como crear todos los componentes que se utilizara para que también se pueda interactuar con el escenario.
* **Iteración construcción del juego:** En esta iteración se procederá con la construcción e implementación propiamente dicha de los algoritmos y componentes que darán vida a nuestro juego, entre los que se encuentran aquellos importantes como el algoritmo de búsqueda del camino más corto A\*, se define su heurística con la distancia de Chevyshev así como el algoritmo de aprendizaje reforzado para la toma de las decisiones de las diferentes unidades conocido como algoritmo o método de Montecarlo.
* **Iteración final:** En esta iteración se refinan los aspectos finales del proyecto tales como la documentación, pulimento de posibles bugs que se han producido durante la fase anterior e implementación de los sistemas faltantes.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Ilustración :Diagrama de Gantt (I)

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ilustración : Diagrama de Gantt (II)

Escala de tiempo

Descripción generada automáticamente

Ilustración : Diagrama de Gantt (III)

Escala de tiempo

Descripción generada automáticamente

Ilustración : Diagrama de Gantt (IV)

Escala de tiempo

Descripción generada automáticamente

Ilustración : Diagrama de Gantt (V)

Escala de tiempo

Descripción generada automáticamente con confianza media

Ilustración : Diagrama de Gantt (VI)

## 4.2.- Técnicas

En este apartado se van a exponer las diferentes técnicas que se han utilizado a lo largo del desarrollo del proyecto.

### 4.2.1.- Patrón Singleton

Singleton es un patrón de diseño de software que garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella (Soni).

La clase es la encargada de crear su propia instancia y de mantener una referencia a ella y esta proporciona un método estático que permite a las demás clases acceder a la instancia sin necesidad de crear una nueva.

El patrón Singleton permite limitar el acceso a los recursos compartidos y garantizar que sólo exista una instancia e esos recursos. También permite una mayor flexibilidad al permitir que la instancia de Singleton sea creada y destruida dinámicamente.

### 4.2.2.- Metodología para la Elicitación de Requisitos de Sistemas de Software de Durán y Bernárdez.

La metodología para la Elicitación de Requisitos de Sistemas de Software de Durán y Bernárdez es una técnica para recopilar y documentar los requisitos funcionales de un sistema de software en el ámbito de la investigación. Esta fue propuesta por Amador Durán Toro y por Beatriz Bernárdez Jiménez, de la Universidad de Sevilla, a principios del año 2000 y desde su publicación hasta el día de hoy se ha convertido en un estándar de la industria del software.

En esta metodología se exponen las siguientes tareas para poder realizar la elicitación y la documentación de los requisitos:

* Definir los objetivos y el alcance del proyecto y establecer el plan de elicitación de requisitos.
* Recopilar los requisitos del sistema utilizando diferentes técnicas.
* Verificar que los requisitos están completos y son consistentes.
* Establecer prioridades.
* Documentar los requisitos de manera clara y precisa para su posterior implementación.

## 4.3.- Herramientas

Para la realización del proyecto se han utilizado las siguientes herramientas que se van a exponer a continuación, de forma que, basándonos en la investigación inicial, eran algunas de las mejores para poder llevar a cabo el desarrollo del sistema.

### 4.3.1.- Motor gráfico

A lo largo de este punto se van a describir el motor grafico que se ha utilizado en el desarrollo del proyecto.

#### 4.3.1.1 Unity

Unity es un motor de videojuegos multiplataforma creado por Unity Technologies. Desde su lanzamiento en 2005, Unity se ha convertido en una de las plataformas de desarrollo de videojuegos más populares y se utiliza tanto en la industria de los videojuegos como en industrias como la arquitectura, la ingeniería y la construcción.

Algunos conceptos teóricos principales de Unity incluyen:

1. **Motor de juego:** Unity es un motor de juego, lo que significa que proporciona un marco para desarrollar juegos de computadora. Esto incluye renderizado de gráficos, física de simulación, sonido, scripting, animación y mucho más.
2. **Multiplataforma:** Unity es conocido por su capacidad para exportar juegos a múltiples plataformas. Estas incluyen Windows, Mac, Linux, Android, iOS, consolas de juegos y más.
3. **Entorno de programación**: Unity utiliza principalmente C# para la programación de comportamientos de juego y características. Esto incluye el control de los personajes, la gestión de la interfaz de usuario, la interacción con bases de datos, la manipulación de gráficos y sonidos, y muchas otras funcionalidades.
4. **Interfaz de Usuario (UI):** La interfaz de usuario de Unity se divide en varias partes, incluyendo la Vista de Escena, donde los desarrolladores pueden manipular objetos en 3D en tiempo real; el Inspector, que permite ver y editar las propiedades de los objetos; y la Vista de Juego, que permite a los desarrolladores ver su juego tal como lo haría un jugador.
5. **Assets y Prefabs**: Los "assets" o "recursos" son cualquier elemento que se utiliza en el juego, como modelos 3D, texturas, sonidos, scripts, etc. Los prefabs, o prefabricados, son instancias de objetos que se pueden reutilizar en varias escenas, permitiendo una mayor eficiencia y consistencia.
6. **Componentes y sistemas de juego**: Unity permite a los desarrolladores construir sus juegos utilizando un enfoque basado en componentes. Esto significa que los objetos en un juego están formados por componentes individuales que definen su comportamiento y características.
7. **Scripting**: Unity utiliza el lenguaje de programación C# para scripting. Los scripts se utilizan para controlar el comportamiento de los objetos en el juego y para implementar la lógica del juego.
8. **Unity y Realidad Virtual/Aumentada**: Unity se ha convertido en una plataforma líder para el desarrollo de aplicaciones de realidad virtual (VR) y realidad aumentada (AR). Unity soporta una variedad de dispositivos de VR/AR y tiene características integradas para ayudar a los desarrolladores a crear experiencias inmersivas.

Unity es una plataforma potente y versátil que permite a los desarrolladores crear desde simples juegos 2D hasta simulaciones 3D completas para una variedad de industrias y aplicaciones. Con un gran número de recursos disponibles en línea y una comunidad activa, Unity es una excelente opción para aquellos interesados en el desarrollo de videojuegos y aplicaciones interactivas.

### 4.3.2.- Lenguaje de programación

En este apartado se van a exponer el lenguaje de programación utilizados en el desarrollo y en el interior de cada lenguaje se van a mostrar los paquetes más relevantes utilizadas en el lenguaje utilizado.

#### 4.3.2.1- C#

C# (pronunciado C Sharp) es un lenguaje de programación orientado a objetos y fuertemente tipado, diseñado por Microsoft en el año 2000 como parte de su plataforma .NET. Aunque se inspira en lenguajes anteriores como Java y C++, tiene sus propias características y convenciones únicas.

Algunos de los aspectos teóricos clave de C#:

1. **Fuertemente Tipado:** C# es un lenguaje fuertemente tipado, lo que significa que el tipo de datos de cada variable y objeto se conoce en tiempo de compilación. Esto ayuda a minimizar errores como operaciones inválidas entre tipos incompatibles.
2. **Orientado a Objetos:** C# es un lenguaje de programación orientado a objetos (OOP), lo que significa que se centra en la creación y manipulación de "objetos", que son instancias de "clases". Las clases son plantillas que definen las propiedades y comportamientos de un objeto.
3. **Interoperabilidad de Plataformas:** C# fue diseñado para la plataforma .NET de Microsoft, pero también se puede usar en una variedad de plataformas a través de .NET Core y Xamarin. También es el principal lenguaje de programación utilizado en el motor de juego Unity.
4. **Gestión Automática de Memoria:** C# usa un recolector de basura automático, lo que significa que los objetos que ya no son necesarios son automáticamente eliminados de la memoria, liberando así recursos.
5. **Excepciones y manejo de errores:** C# incluye un sólido sistema de manejo de errores basado en excepciones, lo que permite a los programadores manejar errores en tiempo de ejecución y prevenir el bloqueo de aplicaciones.
6. **Seguridad de Tipos:** C# proporciona características de seguridad de tipos, lo que significa que previene operaciones que no son seguras en cuanto a tipos, como conversiones de tipos inseguras, operaciones aritméticas desbordadas, acceso a memoria no asignada, etc.
7. **Soporte para programación asíncrona**: C# proporciona soporte incorporado para la programación asíncrona, que es fundamental para el desarrollo de aplicaciones modernas que requieren tareas no bloqueantes y eficientes en la gestión de recursos.
8. **Lenguaje de alto nivel:** C# es un lenguaje de alto nivel, lo que significa que su sintaxis está diseñada para ser fácilmente entendible por los humanos en lugar de las máquinas. Esto hace que el lenguaje sea más fácil de leer y escribir.
9. **Sintaxis y estructura del lenguaje:** La sintaxis de C# es similar a otros lenguajes C-like, con funciones, variables, operadores, ciclos, declaraciones condicionales, clases y más. C# también soporta la sobrecarga de operadores, los genéricos, los delegados, los eventos, y tiene un soporte robusto para la manipulación de cadenas y expresiones regulares.

#### 4.3.2.1.1.- UnityEngine.UI

Este es un sistema de IU (Interfaz de Usuario) para Unity, y viene con el motor de Unity. Se utiliza para crear interfaces de usuario en tus juegos.

Es un paquete de Unity que proporciona a los desarrolladores las herramientas necesarias para crear y manipular interfaces de usuario en sus juegos. Estas herramientas permiten a los desarrolladores crear menús de juego, pantallas de pausa, indicadores de vida, indicadores de puntuación, diálogos, inventarios, y prácticamente cualquier elemento de interfaz de usuario que se pueda ver en un videojuego.

Algunos componentes claves que `UnityEngine.UI` ofrece:

* **Canvas**: Este es el área donde se dibujan todos los elementos de la UI. Puedes tener varios Canvas en una escena y cada Canvas puede tener su propia cámara de renderización.
* **Panel**: Los paneles se utilizan para organizar elementos de la UI en un grupo, lo que facilita moverlos o manipularlos todos a la vez. Los paneles también pueden tener un fondo, lo que los hace útiles para crear ventanas de diálogo o menús de pausa.
* **Text**: Es un componente que se utiliza para mostrar texto en la UI. Puedes cambiar el tamaño, el color, la fuente y otros atributos del texto.
* **Button**: Los botones son una parte integral de cualquier UI. Puedes asignar acciones específicas a los botones, que se ejecutan cuando el jugador hace clic en el botón.
* **Slider**: Los deslizadores son útiles para cualquier tipo de control de la UI que requiera un rango de valores. Son comúnmente utilizados para cosas como controlar el volumen de la música o el brillo de la pantalla.
* **Image/Sprite**: Estos componentes permiten mostrar imágenes o sprites en la UI, útiles para iconos de elementos, avatares de personajes, indicadores de vida y mucho más.

Todos estos componentes de la UI se pueden personalizar ampliamente en cuanto a su aspecto y comportamiento, y pueden interactuar con scripts escritos en C# para crear UIs interactivas y dinámicas.

Además de estos componentes básicos, `UnityEngine.UI` también incluye sistemas para manejar la navegación de la UI (como el movimiento entre botones utilizando el teclado o el controlador), y para manejar animaciones de la UI.

#### 4.3.2.1.2.- TextMeshPro

TextMeshPro es una solución avanzada para la visualización de texto en Unity. Se trata de un sistema de representación y disposición de texto que proporciona un mayor control sobre la estética y el formato del texto en comparación con la funcionalidad estándar de texto de Unity. TextMeshPro permite a los desarrolladores utilizar una amplia variedad de estilos y efectos en sus elementos de texto.

Algunas características y capacidades clave de TextMeshPro son:

* **Calidad de renderizado**: TextMeshPro proporciona un renderizado de texto de alta calidad y fidelidad. A diferencia del componente estándar de texto de Unity, TextMeshPro utiliza Signed Distance Field (SDF) para renderizar los glifos del texto, lo que permite un escalado y zoom suave sin pixelación o borrosidad.
* **Soporte para fuentes**: TextMeshPro permite el uso de fuentes TrueType y OpenType, y proporciona una herramienta para generar "fuentes de atlas" a partir de estas, que son más eficientes para su renderizado.
* **Estilos de texto y decoraciones**: Con TextMeshPro, puedes aplicar estilos de texto como negrita, cursiva, subrayado y tachado. Además, es posible ajustar el espaciado entre letras, palabras, líneas y párrafos.
* **Efectos de texto**: TextMeshPro ofrece una variedad de efectos para mejorar la apariencia del texto. Esto incluye sombras, contornos, degradados, resplandor, y otros.
* **Manejo de texto 3D**: A diferencia del componente de texto estándar de Unity, TextMeshPro puede trabajar con texto en un entorno 3D, lo que permite la creación de texto tridimensional con profundidad y perspectiva.
* **Interacción con texto**: TextMeshPro permite la interacción con el texto en formas más avanzadas, como el reconocimiento de enlaces y el resaltado de texto al pasar el cursor sobre él.

#### 4.3.2.1.3.- Cinemachine:

Cinemachine es una suite de herramientas cinematográficas para Unity que mejora y simplifica la manera en que los desarrolladores pueden manipular y controlar cámaras en sus juegos. Cinemachine puede ayudar a los desarrolladores a crear cámaras dinámicas, de seguimiento, de tercera persona, de vista superior, y muchos otros tipos de cámaras, con una gran cantidad de opciones de personalización.

Algunas características clave de Cinemachine son:

* **Virtual Cameras**: Las cámaras virtuales son una característica central de Cinemachine. Son entidades que no renderizan por sí mismas, sino que describen cómo se debe mover y comportar una cámara. Puedes tener varias cámaras virtuales y cambiar entre ellas, lo que permite crear fácilmente cortes y transiciones de cámara.
* **Cinemachine Brain**: Este es el componente que se adjunta a una cámara de Unity. Gestiona todas las cámaras virtuales en la escena y decide cuál de ellas está controlando la cámara física en cualquier momento. También se encarga de las transiciones y mezclas entre diferentes cámaras virtuales.
* **Blend System**: El sistema de mezcla de Cinemachine te permite definir cómo se realiza la transición entre diferentes cámaras virtuales. Puedes controlar la duración y el estilo de la mezcla para crear transiciones de cámara suaves.
* **Noise and Shake**: Cinemachine proporciona sistemas de ruido y vibración que puedes utilizar para crear efectos de cámara como temblores o movimientos aleatorios, que son útiles para escenas de acción o para simular el movimiento de una cámara handheld.
* **Camera Tracking**: Cinemachine puede seguir objetos, orientarse hacia ellos y reencuadrarlos automáticamente. Esto es útil para las cámaras de seguimiento, que se mueven y giran para seguir a un personaje o a otro objeto en movimiento.
* **Cinemachine Impulse**: Este sistema permite a los desarrolladores generar y propagar eventos de fuerza física que pueden influir en las cámaras. Por ejemplo, una explosión en el juego podría hacer que la cámara tiemble.

### 4.3.3.- Herramientas auxiliares

En este apartado se van a describir las principales herramientas que han ayudado en el desarrollo del proyecto, tanto para la creación del código como para la realización de la documentación.

#### 4.3.3.1.- Git, GitHub

Git es un sistema de control de versiones de código abierto utilizado para llevar un seguimiento de los cambios en archivos y carpetas y para coordinar el trabajo en proyectos de software. El sistema de control de versiones permite a los desarrolladores trabajar de forma independiente y hacer commit de sus cambios en su propio repositorio local, lo que facilita la colaboración y el trabajo en equipo (Guervós, 2017).

GitHub es una plataforma en línea de código abierto que proporciona alojamiento y gestión de proyectos de software utilizando el sistema de control de versiones Git. La plataforma se utiliza para compartir y colaborar en proyectos de software. Ofrece la posibilidad de hacer seguimiento de errores y solicitudes de características, la integración con diferentes servicios de integración continua y el soporte para la revisión de código (Tsitoara, 2020).

#### 4.3.3.2.- EZ Estimate

EZ Estimate es una herramienta CASE (Kendall, 2005) utilizada para generar estimaciones de costos de proyectos. Esta estima dicho coste a partir del modelo de requisitos y la complejidad asociada a ellos. Nos da un resultado del coste del tiempo que va a tomar en horas de persona.

#### 4.3.3.3.- Microsoft Project

Microsoft Project es una herramienta de software de gestión de proyectos utilizada para planificar, seguir y analizar proyectos. La herramienta se utiliza para organizar y llevar a cabo tareas y actividades necesarias para completar un proyecto dentro de un plazo determinado. Además, se pueden crear y asignar tareas a diferentes miembros del equipo, establecer dependencias entre tareas y seguir el progreso de un proyecto. (Biafore, 2013)

Ofrece una amplia variedad de herramientas de análisis y visualización, como la posibilidad de generar informes y gráficos que muestren el progreso del proyecto y el uso de recursos, entre los que se encuentra el diagrama de Gantt el cual ayuda a visualizar las tareas y principales hitos de una mejor forma.

#### 4.3.3.4.- Visual Paradim

Visual Paradigm es una herramienta de software de modelado y diseño utilizada para crear y gestionar modelos de sistemas (Henderi, 2021). La herramienta se utiliza para representar y documentar de manera visual diferentes aspectos de un sistema, como su estructura, funciones, flujo de trabajo y relaciones entre diferentes elementos.

Con Visual Paradigm, los usuarios pueden crear y gestionar modelos utilizando diferentes lenguajes de modelado, como UML (Fowler, UML gota a gota, 1999). La herramienta también permite a los usuarios crear diferentes tipos de modelos, como diagramas de clases, diagramas de actividad y diagramas de secuencia, etc.

#### 4.3.3.5.- Doxygen

Doxygen es una herramienta de documentación que se utiliza para generar documentación a partir de comentarios en código fuente. Aunque no es específica de Unity (y de hecho se utiliza en una amplia gama de proyectos de software), puede ser útil en proyectos de Unity para ayudar a documentar y entender los scripts de C#.

Algunas características clave de Doxygen:

* **Compatibilidad con múltiples lenguajes:** Doxygen puede procesar código fuente escrito en varios lenguajes de programación, incluyendo C#, el lenguaje utilizado para escribir scripts en Unity. También es compatible con C++, C, Objective-C, Python, Java, PHP, entre otros.
* **Generación de documentación automática:** Doxygen genera documentación a partir de comentarios en el código fuente. Puedes estructurar estos comentarios de una manera específica para indicar qué partes del comentario deben utilizarse para qué partes de la documentación.
* **Varias salidas de documentación:** Doxygen puede generar documentación en varios formatos, incluyendo HTML (que puede ser vista en un navegador web) y LaTeX (que puede ser convertido a PDF para una documentación impresa).
* **Diagramas de clases:** Si usas ciertas etiquetas en tus comentarios, Doxygen puede generar diagramas de clases a partir de tu código, lo que te ayuda a visualizar las relaciones entre diferentes clases y métodos.
* **Integración con IDEs:** Muchos entornos de desarrollo integrados (IDEs) tienen algún tipo de soporte para Doxygen, lo que te permite ver la documentación generada directamente en tu IDE.

# 5.- Aspectos relevantes del desarrollo

En este apartado se van a exponer los aspectos relevantes del desarrollo. Se va a comenzar explicando el proceso de desarrollo desde el punto de vista de la ingeniería del software y se van a poder observar aspectos como el modelo de requisitos, el modelo de análisis y el modelo del diseño.

Posteriormente, se va a mostrar una visión del sistema a alto nivel, explicando la arquitectura del sistema y todos sus componentes.

Se continuará con cada una de las partes del sistema por separado, explicando la investigación realizada en lo relevante a ellas, el funcionamiento, tecnologías y también mostrando el diseño que se ha elegido para cada apartado.

Por último, se va a explicar los aspectos que tienen que ver con el despliegue.

## 5.1.- Proceso de desarrollo

Para el proceso de desarrollo se ha seguido la metodología del proceso unificado, el cual ya se ha presentado en el apartado de metodología. El proceso unificado presenta el ciclo de vida de un proyecto el cual es iterativo e incremental.

El proceso de desarrollo del sistema software se ha divido en 4 iteraciones. Para cumplimentar dichas iteraciones se van a seguir los siguientes pasos que definen también el ciclo de vida de un proyecto software, los cuales son investigación, requisitos, análisis, diseño, implementación y pruebas.

### 5.1.1.- Investigación

El proceso de investigación, el cual se describe a lo largo del Anexo II, se ha realizado durante todo el transcurso de las diferentes iteraciones que conforma el proyecto. Esto se debe a que así se focaliza toda la atención de esa iteración en específico o en la generación de algo designado para dicha iteración.

En base a lo expuesto anteriormente, se va a presentar resumidamente el objetivo de la investigación tratado en cada iteración del proyecto y se van a exponer los diferentes resultados obtenidos.

* **Iteración inicial**:
  + Para comenzar, se lleva a cabo una investigación de los casos y problemáticas más importantes que se den a lo largo del desarrollo de un proyecto de este tipo. De las que se ha concluido que debe desarrollarse de tal manera que se pueda usar todo el potencial del motor gráfico, así como la comunicación de los diferentes componentes que conforman las diferentes escenas del proyecto.
  + Una vez conocida la problemática, se va a llevar a cabo una investigación de otros proyectos similares para investigar sus mecánicas y como se han llevado a cabo las diferentes soluciones. En cuanto a este tipo de soluciones se ha encontrado bastantes ejemplos dentro del mismo género de videojuego en el que me apoyo para realizar este proyecto se encuentran juegos como “XCOM” o “Fire Emblem”, que, si bien tienen enfoques y mecánicas diferentes entre sí, tienen en común el uso de “celdas o baldosas” para mover a la unidad lo que se podría obtener de hay el algoritmo de búsqueda del camino más corto como lo es el A\*.
  + Por último, se ha llevado a cabo una investigación sobre que técnicas, lenguaje de programación, software, etc. utilizar para realizar el desarrollo del proyecto. Llegando a la conclusión de que se usara el motor grafico de UNITY y el lenguaje que este utiliza C# para crear los diferentes scripts que nos permitirán ejecutar los diferentes componentes del proyecto.

Para poder desarrollar una IA se ha optado por usar un algoritmo de aprendizaje reforzado conocido como “Algoritmo de Montecarlo” que permite desarrollar una IA contrincante lo suficientemente capaz de ser desafiante para el jugador y elegir las acciones mediante castigo o recompensa.

* **Iteración controles de cámara e interacción con el entorno:**
  + Primero en esta parte de la iteración se va a investigar sobre cómo se utilizarán los controles de cámara ya que esta es importante para el resto del juego. Todo este desarrollo se realizará mediante script de C# y usando un paquete de Unity llamado Cinemachine que permite el uso de una cámara virtual que puede ser controlada por el jugador. Una vez implementado el GameObject llamado CameraController, añadimos a nuestra VirtualCamera el “transform” del GameObject para que este pueda seguirlo y poder interaccionar con ella. Ahora ya tenemos una cámara funcional que puede rotar sobre el eje Y ademas se puede desplazar sobre los ejes X y Z.
  + Posteriormente, se va a investigar cómo interaccionar con el entorno, para ello será necesario crear varios GameObject y scripts de C# para controlar la dinámica. El control de interacción con el mundo de nuestro juego usaremos el ratón, que podrá interactuar con la interfaz.
* **Iteración construcción del juego:**
  + Primero que se realizara en este apartado es investigar toda la parte de los algoritmos necesarios tanto de encaminamiento como de parte de la IA.

Para el encaminamiento se ha usado el algoritmo de búsqueda del camino más corto A\*. Para la heurística de este apartado se tomará el algoritmo de Chevyshev que permite el desplazamiento diagonal dentro de los ejes X y Z. Este algoritmo se implementará dentro del GameObject Pathfinding que utilizará otro GameObject llamado Grid que será el encargado de generar las casillas por la que se moverán las unidades.

Para que una unidad tenga movimiento se usara implementara la acción la cual es una clase heredada de la clase abstracta llamada BaseAction. Dicha acción llamada MoveAction es la que permite mover la posición de una unidad dentro de las casillas que se han creado anteriormente en el Grid.

* + Se continua con la investigando de los modelos 3D de nuestras unidades. Para ello se ha usado diversos paquetes de assets que hemos importado al proyecto entre los que se encuentran los más importantes “PolygonDungeonRealm”, “Medieval Village Building Pack” y “PoligonFantasyHeroCharacters” entre otros. Para realizar las animaciones para las unidades se ha utilizado la utilidad de Unity llamada “animator” y se han importado modelos de animación de la página web perteneciente a “Adobe” llamada “Mixamo”.
  + Una vez creados los modelos se continúa conformando la unidad o personajes con los que jugaremos dentro del juego y que serán nuestros contrincantes, para ello implementamos todas las acciones de ataque, implementamos un sistema de vida, y el script más importante llamado Unit.

Dentro de este apartado incluimos también todo lo relativo a la inteligencia artificial. Existen dos scripts llamados EnemyAI y EnemyAIAction que permiten implementar un cerebro a cada unidad. Dentro de EnemyAI se encuentra el método que llamara al algoritmo de Montecarlo dentro de la clase BaseAction, el cual elegirá la acción a ejecutar por la unidad contrincante en base a una serie de simulaciones donde se obtendrá que acción tendrá un peso mayor para que la unidad la realice.

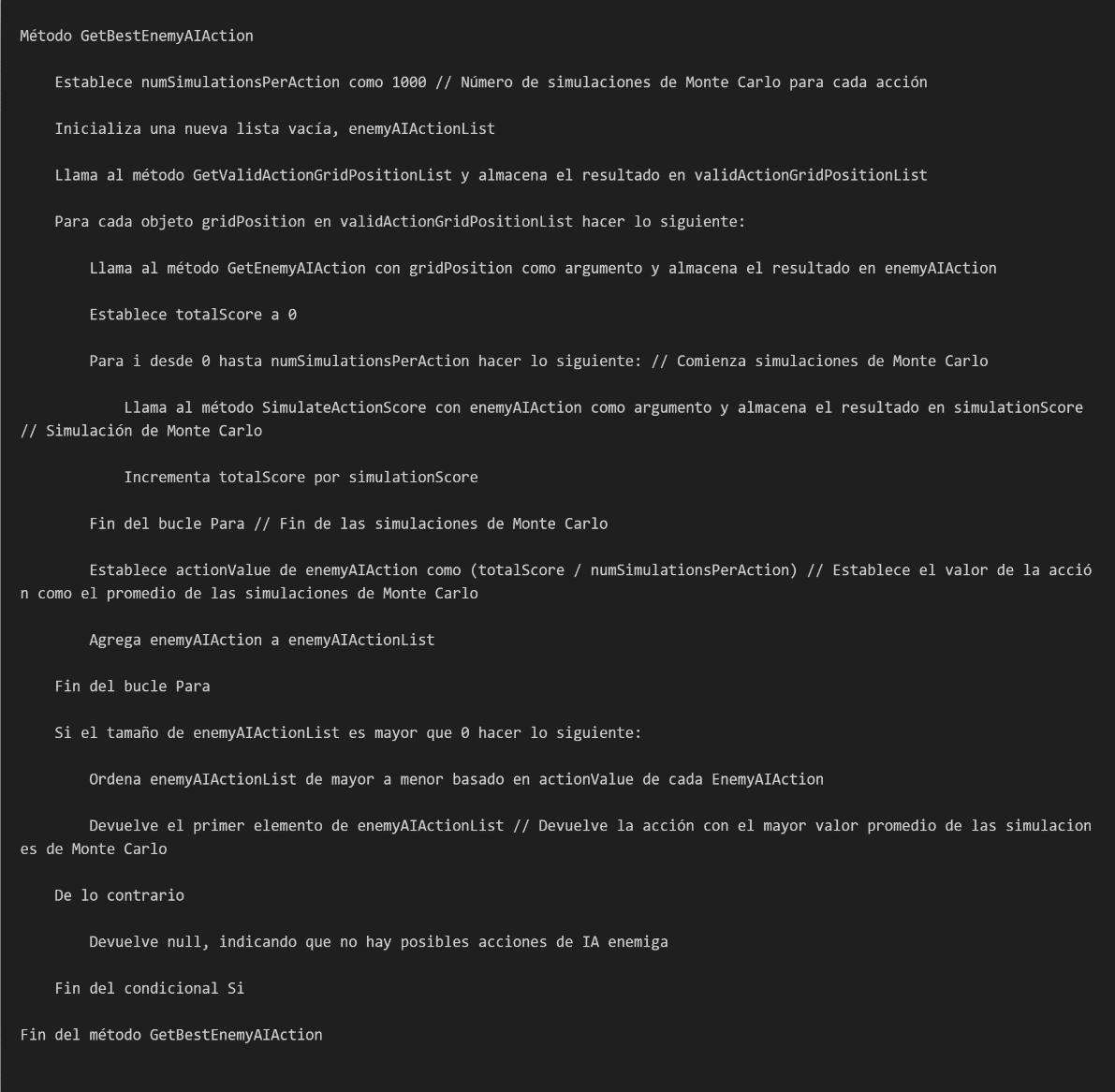


Ilustración : Pseudocódigo algoritmo de Montecarlo

Para elegir que acción ejecutar se simularan diferentes escenarios por cada acción a ejecutar, en cada simulación se plantea una recompensa o castigo. Un ejemplo es el caso de mover a la unidad, ya que esta decidirá moverse a la unidad enemiga más cercana, si la unidad es de rango, su máxima distancia que puede acercarse al objetivo es de dos celdas para poder realizar un ataque.

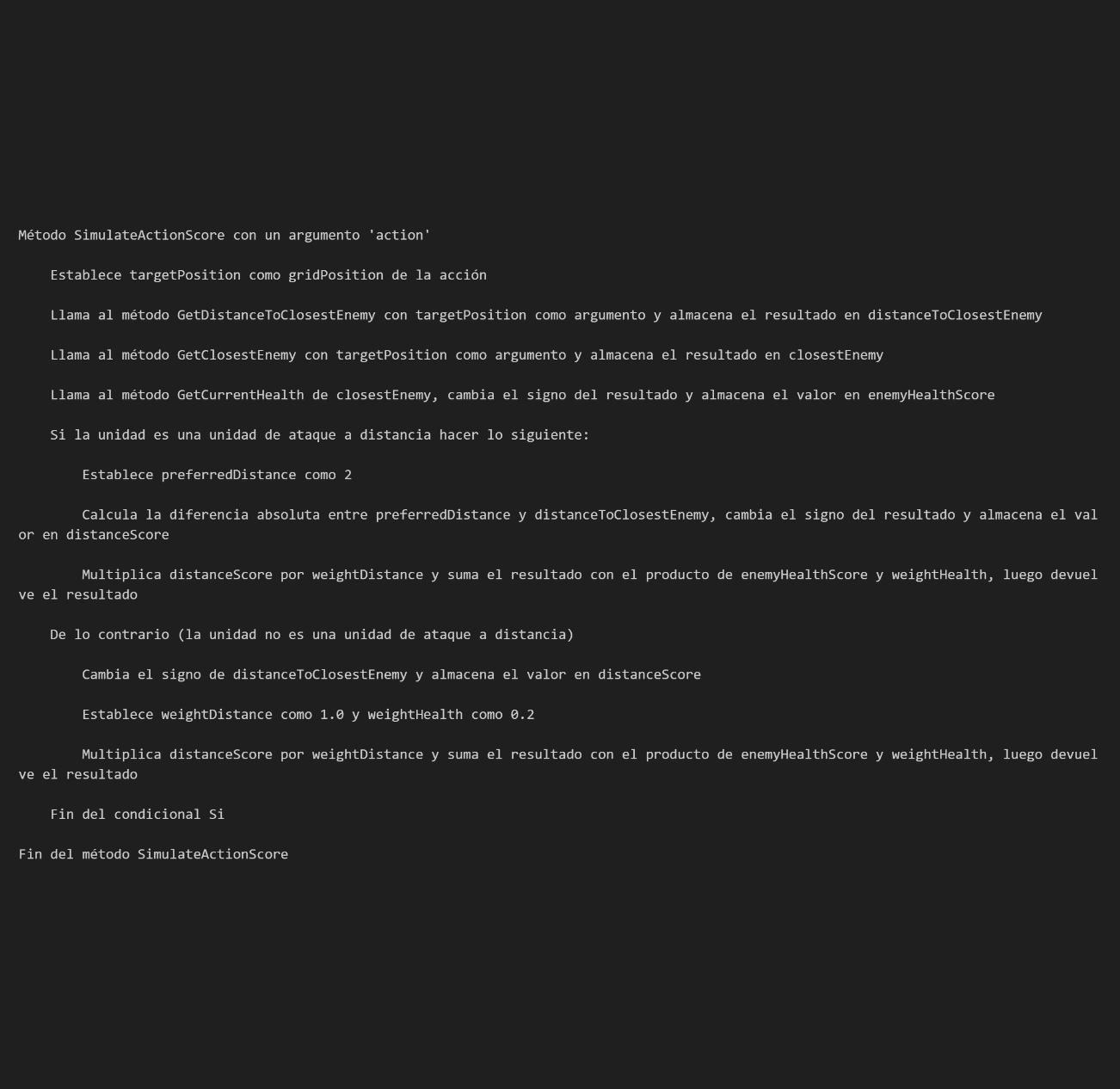


Ilustración : Ejemplo de simulación del algoritmo

* Iteración final:
  + En esta iteración se refinarán la documentación generada a lo largo del proyecto, se corregirán los diferentes bugs que se pueden encontrar dentro del juego, así como se obtendrán datos de la experiencia de usuario mediante una demo entregada a un número reducido de personas.
  + Primero se corrigen los bugs visuales que quedan en el proyecto, sobre todo aquellos referentes a los diferentes al motor de colisiones y algunos efectos especiales.
  + Se continua con el refinamiento del diseño del juego, añadiendo todas las funciones no vitales para dar una experiencia más cercana a un rpg.

Se implementa todo lo referente al sistema de audio y los últimos requisitos que faltan, como lo son el poder modificar las preferencias del usuario, sea nivel de sonido, resolución de pantalla y si se lanzara en formato de ventana o pantalla completa.

* + Se recogen los datos de la experiencia de usuario en la cual se hace hincapié en la necesidad de mostrar más información en el combate, en el coste de cada acción que una unidad puede realizar. El resto de experiencia de usuario es satisfactoria, los botones son bastante visibles, de estilo minimalista lo cual concuerda con el resto del juego.

### 5.1.2.- Especificación de requisitos

Basándome en la propuesta realizada para este Trabajo de Fin de Grado, más la investigación documentada previamente y la planificación temporal en la que se había realizado la funcionalidad que va a ofrecer la plataforma, se van a definir los objetivos principales de la aplicación.

Después de esto, se ha realizado un primer borrador de los requisitos mediante la metodología para la Elicitación de Requisitos de Durán y Bernárdez en la que se ha establecido las principales funcionalidades del sistema que se corresponden con las primeras funcionalidades.

Posteriormente durante el transcurso de iteraciones el modelo de requisitos se fue refinando hasta el resultado definitivo, el cual se va a poder observar a continuación.

Diagrama, Esquemático

Descripción generada automáticamente

Ilustración : Diagrama de casos de uso

En el modelo de requisitos final se han identificado, un total de diecinueve requisitos funcionales, ocho requisitos de información y nueve requisitos no funcionales que van a ayudar a cumplir con el objetivo inicial el cual era la creación de una plataforma de desarrollo de estrategia basado en aprendizaje reforzado. Todos estos requisitos se pueden resumir en las siguientes tablas y si se desea conocer este apartado más en detalle se recomienda consultar el Anexo II.

Tabla : Resumen de los requisitos funcionales

|  |  |  |
| --- | --- | --- |
| **Referencia** | **Nombre** | **Descripción** |
| **CU-01** | Iniciar juego | El jugador inicia el juego desde el menú principal |
| **CU-02** | Elegir escenario | El jugador selecciona un escenario o nivel para comenzar a jugar. |
| **CU-03** | Seleccionar unidad | El jugador selecciona un personaje en pantalla para realizar acciones |
| **CU-04** | Mover unidad | El jugador selecciona una unidad y la mueve a una ubicación del mapa |
| **CU-05** | Ataque cuerpo a cuerpo | El jugador selecciona una unidad con arma cuerpo a cuerpo para atacar a una unidad enemiga |
| **CU-06** | Ataque a distancia | El jugador selecciona una unidad con arma a distancia para atacar a una unidad enemiga |
| **CU-07** | Curar aliado | El jugador selecciona una unidad con la capacidad de curar a sus aliados |
| **CU-08** | Ataque con magia | El jugador selecciona una unidad con acciones mágicas para dañar a las unidades enemigas |
| **CU-09** | Finalizar turno | El jugador finaliza su turno, permitiendo que las unidades controladas por la IA tomen sus acciones. |
| **CU-10** | Configurar opciones del juego | El jugador ajusta las opciones del juego, como gráficos, sonidos y controles. |
| **CU-11** | Ganar o perder escenario | El jugador gana o pierde un escenario basado en el estado del equipo. |
| **CU-12** | Reiniciar escenario | El jugador puede comenzar de nuevo el escenario. |
| **CU-13** | Salir del juego | El jugador puede cerrar el juego |
| **CU-14** | Pausar el juego | El jugador puede acceder al menú de pausa donde aparecerán diferentes opciones |
| **CU-15** | Guardar partida | El jugador guarda su progreso en el juego, incluyendo el estado actual de las unidades y la posición en el escenario o nivel. |
| **CU-16** | Cargar partida | El jugador carga una partida guardada previamente, retomando el progreso desde el punto en que se guardó. |
| **CU-17** | Consultar tutorial/ayuda | El jugador accede a una sección de ayuda o tutorial que proporciona información sobre cómo jugar y consejos para el juego. |
| **CU-18** | Interactuar escenario | Este caso de uso representa la capacidad del jugador o personaje para interactuar con el escenario o entorno en un juego. Puede incluir la interacción con objetos, elementos del entorno, manipulación de objetos, activación de mecanismos y otras interacciones similares que tienen lugar dentro del mundo del juego. |
| **CU-19** | Destruir objeto | Este caso de uso representa la capacidad del jugador o personaje para destruir objetos en el entorno del juego. Esto puede incluir romper objetos, derribar estructuras o eliminar elementos del escenario. |

Tabla : Resumen de requisitos de información

|  |  |  |
| --- | --- | --- |
| **Referencia** | **Nombre** | **Descripción** |
| **IRQ-01** | Mecánicas de juego | Establecer las reglas del juego, como los sistemas de combate por turnos, la progresión del personaje, las habilidades y talentos, y la gestión de recursos e inventario. |
| **IRQ-02** | Unidades | Establece las unidades jugables y enemigas en el juego, incluyendo sus atributos, habilidades, animaciones y comportamientos. |
| **IRQ-03** | Interfaz de usuario | Diseña y desarrolla la interfaz de usuario del juego, incluyendo menús, pantallas de información, indicadores en el juego y elementos de control. |
| **IRQ-04** | Sonido y música | Implementa efectos de sonido, música y ambientes sonoros en el juego para mejorar la experiencia del jugador y la inmersión en el mundo del juego. |
| **IRQ-05** | Inteligencia artificial | Diseña y desarrolla la inteligencia artificial de las unidades enemigas y no jugables, incluyendo sus tácticas de combate, comportamientos y toma de decisiones. |
| **IRQ-06** | Pruebas y depuración | Realiza pruebas en el juego para identificar y solucionar errores, problemas de rendimiento y otros problemas potenciales. |
| **IRQ-07** | Optimización y rendimiento | Trabaja en la optimización del juego para garantizar que funcione de manera eficiente en diferentes dispositivos y plataformas, y cumpla con los Requisitos No Funcionales de rendimiento. |
| **IRQ-08** | Gráficos y efectos visuales | Crea y optimiza los gráficos y efectos visuales del juego, incluyendo personajes, entornos, animaciones y efectos especiales. |

Tabla : Resumen de requisitos no funcionales

|  |  |  |
| --- | --- | --- |
| **Referencia** | **Nombre** | **Descripción** |
| **NFR-01** | Rendimiento | La velocidad y eficiencia con la que el juego se ejecuta, incluyendo el tiempo de carga, la fluidez de las animaciones y la velocidad de respuesta del juego a las acciones del jugador. |
| **NFR-02** | Escalabilidad | La capacidad del juego para manejar un aumento en la cantidad de niveles, unidades, objetos y otros elementos sin comprometer su rendimiento o funcionalidad. |
| **NFR-03** | Portabilidad | La facilidad con la que el juego puede ser adaptado y ejecutado en diferentes plataformas (por ejemplo, PC, consolas, dispositivos móviles) y sistemas operativos. |
| **NFR-04** | Usabilidad | La facilidad de uso del juego, incluyendo la interfaz de usuario, el diseño de los menús, la claridad de las instrucciones y la capacidad de los jugadores para aprender y dominar el juego rápidamente. |
| **NFR-05** | Accesibilidad | La capacidad del juego para ser disfrutado por personas con discapacidades, incluyendo opciones de subtítulos, ajustes de contraste de colores y soporte para dispositivos de entrada alternativos. |
| **NFR-06** | Estabilidad | La solidez y fiabilidad del juego, evitando bloqueos, errores y problemas de rendimiento que puedan afectar negativamente la experiencia del jugador. |
| **NFR-07** | Personalización | La capacidad de los jugadores para personalizar aspectos del juego, como la apariencia de los personajes, la configuración de las teclas de acceso rápido y las opciones de dificultad. |
| **NFR-08** | Estética y diseño | La calidad visual y artística del juego, incluyendo gráficos, animaciones, diseño de niveles y estilo general. |
| **NFR-09** | Mantenibilidad | La facilidad con la que el juego puede ser actualizado, corregido y mejorado a lo largo del tiempo, así como la capacidad de los desarrolladores para agregar nuevos contenidos y características. |

### 5.1.3.- Análisis y diseño

Al igual que en el caso del apartado anterior, debido a que el modelo de ciclo de vida de este proyecto es iterativo e incremental, al principio se han realizado unos modelos iniciales y a medida que se han ido pasando por las diferentes iteraciones se han ido refinando para obtener unos requisitos acordes a la funcionalidad de este proyecto. Se va a presentar los aspectos relevantes al análisis y el diseño y para consultar más detalles de este apartado se debe consultar el anexo III.

Captura de pantalla con letras y números

Descripción generada automáticamente con confianza media

Ilustración : Diagrama de clases I

Diagrama

Descripción generada automáticamente

Ilustración : Diagrama de clases II

Diagrama

Descripción generada automáticamente

Ilustración : Diagrama de clases III

Diagrama

Descripción generada automáticamente

Ilustración : Diagrama de clases IV

Las clases que se muestran en las ilustraciones anteriores se explicaran en una tabla cada una de las más importantes en profundidad. Estas clases se van a ver ordenadas siguiendo el orden mostrado en las ilustraciones.

Tabla : Detalle de clases importantes

|  |  |
| --- | --- |
| **Clase** | **Descripción** |
| BaseAction | Es la clase de la que heredan el resto de las acciones, se trata de una clase abstracta que representa cada una de las acciones que tiene disponible una unidad, se encarga también de implementar el algoritmo de Montecarlo que simula los escenarios posibles para seleccionar la acción a ejecutar por la IA contrincante. |
| HealAction | Esta clase se encarga de ejecutar la acción de curar a un aliado, hereda diferentes métodos de su clase principal. |
| RangeAction | Esta clase se encarga de ejecutar la acción de realizar un ataque a distancia, hereda varios métodos de su clase principal |
| MeleeAction | Esta clase se encarga de ejecutar los ataques cuerpo a cuerpo de la unidad, hereda varios métodos de su clase principal. |
| MoveAction | Esta clase se encarga de mover a la unidad por el mapa, siguiendo un camino de celdas. |
| FireballAction | Esta clase instancia otro objeto llamado Fireball que realizará daño dentro de un área de explosión. |
| UnitAnimator | En esta clase se implementa todo lo referente a las animaciones que puede ejecutar la unidad |
| Unit | Esta clase es la principal que designa un GameObject como una unidad, permite la creación de las estadísticas del personaje, cargar el sistema de vida de la unidad, etc. |
| GridSystem | Esta clase contiene el sistema de celdas por las que se moverán las unidades, así mismo se encarga de comprobar si esta esta ocupada por un obstáculo o no. |
| Pathfinding | Es la clase encargada de implementar el algoritmo A\*, en esta clase se calcula el camino más corto que puede recorrer una unidad, comprobar si existen obstáculos o si la casilla ya está ocupada. |
| LevelGrid | Esta clase se encarga de generar las casillas por las que se moverán las unidades, así mismo se encarga de comprobar si esta está ocupada por un obstáculo o no. |
| EnemyAI | Esta clase implementa el cerebro de la IA contrincante, se encarga de llamar al algoritmo de aprendizaje reforzado para poder emplear la mejor acción en el estado que se encuentra la unidad. |
| UnitManager | Se encarga de controlar cuantas unidades existe en la escena, diferenciando también entre enemigas y las del jugador. |
| TurnSystem | Se encarga de controlar los turnos entre el jugador y la IA contrincante. |
| SceneSelector | Se encarga de cambiar entre escenas. |
| MouseWorld | Se encarga de controlar todo lo referente a la posición del ratón dentro de las escenas que permiten interactuar con el entorno. |
| CameraController | Se encarga de controlar todo lo referente a la cámara, tanto el desplazamiento como la rotación el zoom. |
| UnitRagdoll | Se encarga de controlar el “Ragdoll” de la unidad que se crea cuando la vida de esta llega a cero. |
| HealthSystem | Se encarga de controlar la vida que dispone la unidad en cada momento. |
| GameManager | Se encarga de controlar el estado del juego y de guiar los diferentes componentes que lo conforman. |

En cuanto al diseño procedimental, fue definido en la iteración inicial para que posteriormente a medida que iban pasando las demás iteraciones se fueran refinando como se ha ido explicando a lo largo de esta sección.

En cuanto al diseño arquitectónico, se ha definido mediante el diagrama de clases del diseño que, aunque se va a mostrar a continuación, para consultarlo en mayor profundidad se puede consultar el Anexo III.

En cuanto a la arquitectura a alto nivel se puede plantear en una etapa ya que todo nuestro proyecto se realiza usando el motor Unity Engine que se encarga de englobar todo lo necesario para la realización del proyecto.

### 5.1.4.- Implementación

La implementación del sistema se ha realizado en un único lenguaje ya que el sistema de scripts que utiliza Unity es único siendo este C#.

La implementación no se puede resumir en el lenguaje expuesto ya que dentro de este se han utilizado librerías y paquetes tanto de C# básico como implementados por Unity.

# 6.- Resultados

En este apartado se van a exponer los diferentes resultados que se han obtenido de la realización del proyecto y las posibles líneas de trabajo futuro que se pueden vislumbrar tras la finalización de este.

## 6.1.- Resultados y conclusiones

## 6.2.- Líneas de trabajo futuras

# Bibliografía

Microsoft. (15 de 02 de 2023). *learn.microsoft.com*. Obtenido de https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/