**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](#_Toc132959430)

[2. Objetivos del proyecto 2](#_Toc132959431)

[3. Conceptos teóricos 2](#_Toc132959432)

[4. Metodología, técnicas y herramientas 2](#_Toc132959433)

[5. Aspectos relevantes del desarrollo 2](#_Toc132959434)

[6. Resultados 2](#_Toc132959435)

[7. Bibliografía 3](#_Toc132959436)

# 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.

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:

## 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.

## 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.

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.

## 3.5.- 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.

## 3.6.- 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.

C# es una excelente opción para el desarrollo de aplicaciones de escritorio, juegos, aplicaciones web y servicios, así como para el desarrollo de aplicaciones móviles multiplataforma.

# 4.- Metodología, técnicas y herramientas

## 4.1.- Algoritmo de encaminamiento

## 4.2.- Algoritmo de aprendizaje reforzado

# 5.- Aspectos relevantes del desarrollo

# 6.- Resultados

# 7.- Bibliografía