Videojuego de ajedrez con modo de juego contra la máquina

Memoria

Trabajo de Fin de Grado

GRADO EN INGENIERÍA INFORMÁTICA

Junio de 2025

Autor

Óscar Sánchez Rubio

Tutores

Luis Augusto Silva Zendron

Gabriel Villarrubia González

Certificado de los tutores TFG

D. Luis Augusto Silva Zendron y D. Gabriel Villarrubia González, profesores del Departamento de Informática y Automática de la Universidad de Salamanca,

HACEN CONSTAR:

Que el trabajo titulado "*Videojuego de ajedrez con modo de juego contra la máquina*", que se presenta, ha sido realizado por Óscar Sánchez Rubio, con DNI ****7471S y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Ingeniería Informática en esta Universidad.

Salamanca, 3 de Julio de 2025

Fdo.: _____

Fdo.: _____

Resumen

Este trabajo presenta el desarrollo de un motor de ajedrez con inteligencia artificial, centrado en una implementación basada en *bitboards*, búsqueda mediante el algoritmo Minimax con poda alfa-beta, y una función de evaluación clásica, basada en heurística. Se ha desarrollado en el motor Unity y gestionado con Microsoft Project, utilizando un enfoque basado en el Proceso Unificado.

Para la elaboración de este proyecto, se ha tomado como inspiraciones y modelos a seguir, otros motores de ajedrez como es Stockfish. En este trabajo también se mostrarán y explicarán las alternativas que estos otros motores pueden haber utilizado, y su correspondiente razonamiento de por qué haber elegido las de este proyecto en concreto.

También se entrará en profundidad es aspectos relativos a la Ingeniería del Software, los cuáles han sido empleados para conseguir obtener el producto final, la aplicación. Estos aspectos incluyen diagramas y tablas correspondientes a varias disciplinas del Proceso Unificado, además de otros esquemas elaborados para la esquematización de la interfaz.

Durante el desarrollo, se han alcanzado los objetivos principales, entre ellos: la creación de un tablero funcional, la detección correcta de jaque, jaque mate y tablas, los modos de juego contra otro jugador y contra la máquina, la selección de dificultad, el control de tiempo y la carga de posiciones desde cadenas FEN. A pesar de cumplir con la mayoría de las funcionalidades planteadas, el rendimiento de la inteligencia artificial no alcanza un nivel competitivo alto, siendo su mayor debilidad los instantes finales de la partida.

Abstract

This work presents the development of a chess engine with artificial intelligence, focused on an implementation based on bitboards, search using the Minimax algorithm with alpha-beta pruning, and a classical heuristic-based evaluation function. The project was developed using the Unity engine and managed with Microsoft Project, following an approach based on the Unified Process.

In the development of this project, existing chess engines such as Stockfish have been used as inspiration and reference models. This work also presents and explains the alternative techniques used by those engines, along with the reasoning behind the design choices made specifically for this project.

The report also delves into aspects related to Software Engineering, which have been applied to achieve the final product: the application. These aspects include diagrams and tables corresponding to various disciplines of the Unified Process, as well as other schematics designed for the user interface layout.

During development, the main objectives were successfully achieved, including the creation of a functional chessboard, correct detection of check, checkmate and draws, implementation of local multiplayer and AI gameplay modes, difficulty selection, time control, and support for loading positions from FEN strings. Despite fulfilling most of the intended functionalities, the AI's performance does not yet reach a high competitive level, with its greatest weakness appearing in the final stages of the game.

Glosario

- **FEN – *Forsyth–Edwards Notation***: Sistema de notación empleado en ajedrez para representar una posición del tablero mediante una cadena de texto.
- **IA – Inteligencia Artificial**: Campo de la informática dedicado al desarrollo de sistemas y programas capaces de realizar tareas que imiten la inteligencia humana.
- **Motor de ajedrez**: Programa informático diseñado para analizar posiciones de ajedrez y calcular las mejores jugadas posibles mediante algoritmos y técnicas de inteligencia artificial.
- **Movimiento legal**: Jugada que respeta las reglas del ajedrez y no deja al propio rey en jaque. Son los únicos movimientos válidos durante una partida.
- **Movimiento pseudolegal**: Jugada que respeta las reglas de movimiento de las piezas, pero sin considerar si deja al rey en jaque.
- **Heurística**: Conjunto de reglas que determinan un proceso de toma de decisiones.
- **Elo**: Sistema de puntuación utilizado para medir la habilidad relativa de los jugadores en juegos competitivos como el ajedrez.
- **Enroque**: Movimiento en el que el rey se pone en cobertura de alguna de sus dos torres.
- **Derechos de enroque**: Información que indica si cada uno de los bandos puede enrocar de su lado del rey o de la reina.
- **Minimax**: Algoritmo de toma de decisiones empleado en juegos de 2 jugadores.
- **Alfa-beta**: Técnica que mejora la búsqueda del algoritmo Minimax.
- **Transposición**: Es una posición concreta del tablero de ajedrez, la cual puede ser alcanzada por secuencias diferentes de movimientos.
- ***Bitboard***: Representación del tablero de ajedrez usando enteros de 64 bits, donde cada bit corresponde a una casilla.
- ***Zobrist hashing***: Función empleada para generar un *hash* único para cada posición del tablero de ajedrez.
- ***Ply***: Unidad que representa un medio movimiento en ajedrez. Un turno completo equivale a 2 *ply*.
- ***Perft***: Es una función empleada principalmente para la depuración, que calcula el número de nodos explorados por movimiento
- **Promoción**: Movimiento de ajedrez que ocurre cuando un peón alcanza la última fila respecto a su bando.

- **Captura al paso**: Captura que puede hacer un peón sobre otro peón rival que ha movido dos casillas.

- **Ahogado**: Situación del tablero de ajedrez en la que alguno de los dos bandos no tiene ningún movimiento legal posible, pero el rey no se encuentra en jaque

- **Deep Blue**: Supercomputadora desarrollada por IBM y especializada en el juego de ajedrez

- **Stockfish**: Motor de ajedrez de código abierto, considerado el más potente del mundo.

Índice

Contenidos

Ilustraciones

Tablas

1. Introducción

La inteligencia artificial es un concepto amplio que puede referirse a múltiples enfoques, ya que puede considerarse un sistema inteligente aquel que realiza una tarea siguiendo reglas predefinidas. Sin embargo, dentro de la inteligencia artificial se encuentra el *machine learning*, una rama que se enfoca en que las máquinas aprendan a partir de datos, sin necesidad de ser programadas explícitamente para cada tarea. A su vez, el *deep learning* es una técnica avanzada de *machine learning* que utiliza redes neuronales profundas para analizar datos complejos y extraer características de forma automática. Este último método, es al que verdaderamente nos referimos hoy en día cuando hablamos de inteligencia artificial, puesto que es en el que se basan la mayoría de los sistemas inteligentes que se diseñan.

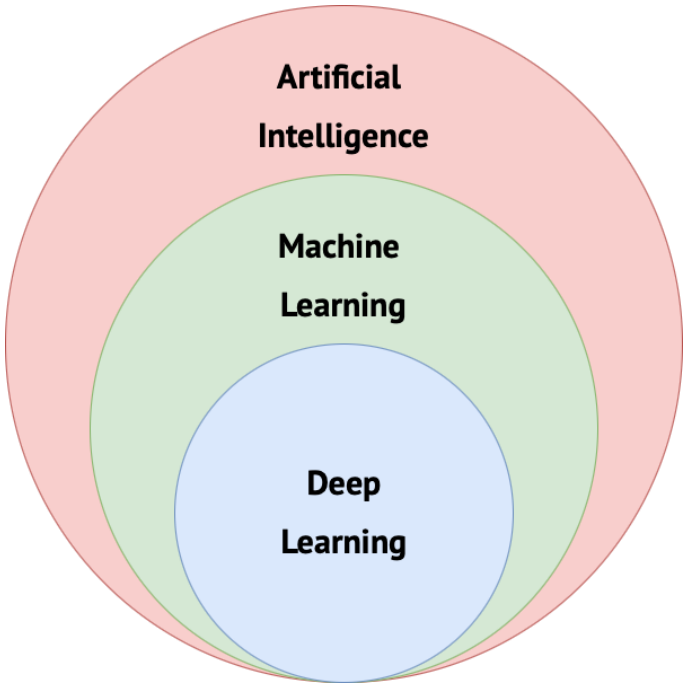


Ilustración 1: Diagrama de los tipos de inteligencia artificial. Fuente: Medium (Byrne, 2021)

Los motores de ajedrez no son una excepción para este suceso. Si bien, en un principio partieron de sistemas inteligentes basados en reglas para elaborar oponentes virtuales, en los últimos años han evolucionado hasta adoptar estas técnicas de *deep learning*. Estos motores de ajedrez requieren de una planificación nada trivial, para poder hallar los movimientos legales de ambos jugadores y saber priorizar el estudio de ciertas posiciones del tablero frente a otras.

El desarrollo de un sistema similar a lo descrito requiere profundizar en muchos temas considerablemente técnicos, tanto a nivel de informática como de ajedrez. Por ello, para poder llevar a cabo un proyecto con este objetivo, se ha optado por centrarse en los métodos más clásicos de la inteligencia artificial, y así poder ahondar lo suficiente como para abarcar la mayor cantidad de temas posibles y crear un oponente virtual que verdaderamente suponga un reto para el usuario.

En esta memoria, se presentarán los objetivos, tanto técnicos como personales, del proyecto, sus antecedentes y una descripción del estado del arte. Además, se explicarán los métodos y herramientas empleados, sus viabilidades y alternativas, y las restricciones y alcance

del proyecto. Por último, que describirá la solución propuesta, se hará un análisis de riesgos y se explicará la organización y gestión aplicada al proyecto.

En cuanto a los anexos, la documentación incluye cinco de ellos:

- **Anexo I: Especificaciones del sistema –** Contendrá la elicitación de requisitos, la definición de los objetivos del sistema y elaboración del diagrama de casos de uso.

- **Anexo II: Análisis y diseño del sistema –** Incluirá los diagramas UML correspondientes a la disciplina de análisis y diseño. Además, se incluirán las actividades relacionadas al diseño de interfaces.

- **Anexo III: Estimación del tamaño y esfuerzo –** Incluirá el cálculo de la estimación del esfuerzo en base a los casos de uso y actores identificados en el primer anexo.

- **Anexo IV: Plan de seguridad** – Se definirán los riesgos y medidas para proteger los activos del proyecto durante su desarrollo.

- **Anexo V: Manual del programador –** Descripción de la estructura del proyecto a nivel de programación.

2. Objeto

En este proyecto se busca alcanzar dos tipos de metas. Por un lado, se encuentran los **objetivos técnicos o funcionales**, los cuales están ligados en gran medida a los requisitos del *software* descritos en el anexo I. Por otro lado, se encuentran los **objetivos personales**, que son aquellos que se impone el alumno para el desarrollo de este trabajo.

En cuanto a los objetivos funcionales, se distinguen los siguientes:

- **Crear un tablero de ajedrez funcional**: El primer paso, y más importante, será la creación de un tablero de ajedrez, en el que se puedan mover las piezas en base a las reglas del ajedrez. También se incluyen aquellos movimientos especiales como las promociones, la captura al paso y el enroque.

- **Gestionar las situaciones de jaque, jaque mate y tablas**: Este objetivo se refiere en esencia a filtrar los movimientos pseudolegales en únicamente movimientos legales. Esto es esencial para la posterior creación de un modo inteligente, puesto que la base para poder saber qué movimiento deberá jugar un sistema inteligente, es otorgarle los posibles movimientos legales de cada posición del tablero.

- **Permitir configurar los parámetros de la partida antes de empezarla**: El usuario debe de ser capaz de ajustar ciertos parámetros, previos a la partida, como la posición inicial del tablero, el color de cada jugador, la duración y el incremento de tiempo de la partida.

- **Modo de juego contra otro jugador:** Debe existir la posibilidad de que el usuario se pueda enfrentar contra otra persona en un modo de juego local.

- **Modo de juego contra la máquina:** Como foco principal del programa, debe existir la posibilidad de enfrentarse con una inteligencia artificial, la cual implementará un algoritmo basa en heurística, en vez de una solución de la mano de las redes neuronales. Esta modalidad debe resultar un reto para el usuario con el que se enfrente.

- **Selección de dificultad de la inteligencia artificial**: Debe existir la posibilidad de poder cambiar la dificultad o, más bien, la certeza con la que jugará la inteligencia artificial en cada partida, pudiendo variar entre fácil, media y difícil.

- **Gestionar el tiempo de partida de los jugadores**: Como se ha mencionado antes, la duración y el incremente de tiempo de la partida debe ser ajustable. Por tanto, durante la partida se deberá decrementar el tiempo a cada jugador, e incrementarlo de forma correspondiente si hay algún incremento. El hecho de que se gestione el tiempo también implica tener en cuenta estos factores para establecer el tiempo que consumirá la inteligencia artificial en cada uno de sus turnos.

- **Cargar partidas en formato FEN**: Debe existir la posibilidad de jugar posiciones del tablero, indicadas a partir de la notación FEN. Esta notación incluye la disposición de las piezas, el turno, enroques disponibles, capturas al paso, número de medias jugadas y número total de jugadas.

Los objetivos personales serían los siguientes:

- **Aprendizaje de los elementos que hacen posible el funcionamiento de un motor de ajedrez**: Aprender sobre los módulos y estructuras que hacen posible el funcionamiento de un motor de ajedrez, desde el tratamiento de los datos y la representación de los mismos, hasta las técnicas que permiten la búsqueda de movimientos y evaluación de los mismos.

- **Creación de un proyecto de escala mediana-grande**: Debido a que, durante el transcurso de la carrera, los proyectos no han sido de una envergadura excesiva, mi propósito es poder terminar un proyecto de gran escala, superando todas las adversidades que eso conlleva.

- **Familiarizarse con las metodologías de la Ingeniería del Software para la elaboración de un proyecto informático**: Aunque las metodologías de la Ingeniería del Software han sido estudias a lo largo la carrera, pretendo adquirir mayor familiaridad con estas.

- **Adquirir experiencia en el uso de un motor de videojuegos**: Los motores de videojuegos son un entorno de desarrollo que me resulta interesante y sobre el cual pretendo aprender más.

- **Mejorar el conocimiento respecto a la programación orientada a objetos**: Profundizar en el manejo y comprensión de la programación orientada a objetos, aprendiendo a estructurar el código en clases, objetos y relaciones entre ellos para desarrollar *software* de forma más organizada y modular.

3. Antecedentes

El ajedrez es un juego con siglos de historia que ha trascendido su carácter de ser un simple entretenimiento para convertirse en un referente intelectual y científico. Su estructura, aunque basada en reglas sencillas, deriva en una complejidad estratégica enorme, por lo que se ha convertido en un modelo ideal para el análisis del pensamiento lógico, la toma de decisiones y la simulación de inteligencia.

Desde los inicios de la inteligencia artificial, el ajedrez ha desempeñado un papel central como uno de los primeros problemas abordados debido a su entorno controlado, con reglas claras, pero con una complejidad combinatoria elevada, habiendo un total de aproximadamente 10^{120} posiciones legales, según el denominado **número de Shannon** (Shannon, 1950).

Ya en el año 1948, Alan Turing y David Champernowne propusieron uno de los primeros —si no el primero— algoritmos capaces de jugar al ajedrez. Puesto que aún no contaban con una máquina para ejecutarlo automáticamente, era un humano el que seguía paso a paso las reglas definidas por el algoritmo y realizaba el proceso (Turing, 1953). Los primeros enfoques, aunque eran muy limitados debido a la baja capacidad de procesamiento de los ordenadores de la época, sentaron las bases de lo que más tarde se conocería como algoritmos de búsqueda y evaluación. A medida que la tecnología avanzó, surgieron programas más sofisticados como Belle, desarrollado en los años 70, que fue de los primeros en incorporar *hardware* especializado para jugar al ajedrez (Condon & Thompson, 1982). Y es que los avances en inteligencias artificiales de ajedrez no se dieron exclusivamente en el campo del *software*, sino que algunos sistemas implementaron una circuitería especial para mejorar su rendimiento en los cálculos que más empleaban

Uno de los hitos más importantes fue la victoria de **Deep Blue**, el supercomputador de IBM, sobre el campeón mundial Garry Kasparov en 1997, demostrando que una máquina podía superar al mejor jugador humano del momento. Esta victoria marcó un antes y un después en la historia de la inteligencia artificial, siendo la primera vez que un ordenador vencía al vigente campeón del mundo (Viana, 2010).

Ilustración 2: Garry Kasparov (negras) contra Deep Blue (blancas) en 1997. Fuente: YOURSTORY (The Day Deep Blue Changed Chess and AI Forever: Kasparov's Historic Defeat, 2023)

Con el tiempo, motores como **Stockfish** han superado ampliamente el nivel de los mejores jugadores humanos, alcanzando puntuaciones Elo superiores a 3500 (Computer Chess Rating List (CCRL)), muy por encima del récord humano de Magnus Carlsen con 2882 (The 7 Most Mindblowing Magnus Carlsen Records, 2024). Stockfish, basado en técnicas de búsqueda clásica mejoradas, incluyendo poda alfa-beta, y sofisticadas funciones de evaluación basadas en redes neuronales, ha marcado el estándar actual en motores de ajedrez de alto rendimiento, cambiando radicalmente la perspectiva en la que los ordenadores abordan el juego del ajedrez; partiendo de una forma programática en los acercamientos más clásicos, y llegando a aproximarse a la forma en la que los humanos razonan en ajedrez.

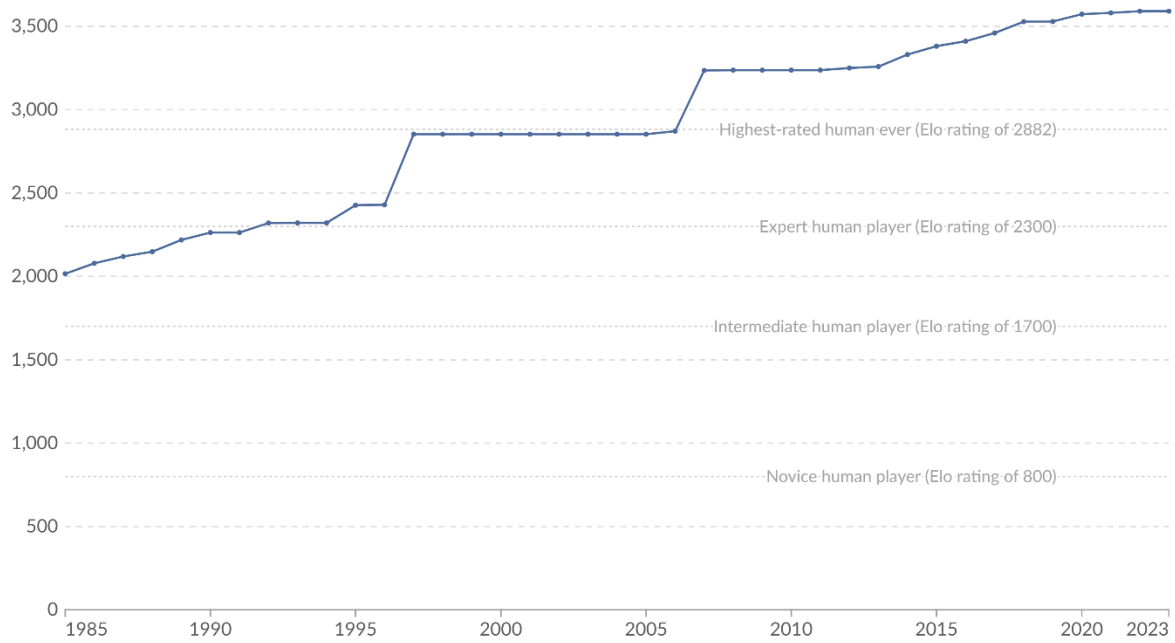


Ilustración 3: Mayor Elo alcanzado por un ordenador a lo largo del tiempo. Fuente: Our World in Data (Highest chess rating ever achieved by computers)

Además de su impacto competitivo, el ajedrez ha adquirido relevancia formativa, siendo una herramienta para enseñar conceptos fundamentales de programación, estructuras de datos y algoritmos de búsqueda. Así, desarrollar un motor de ajedrez ofrece una experiencia práctica para aplicar la teoría en sistemas inteligentes. Por otro lado, el ajedrez puede ser usado como herramienta pedagógica, puesto que más de 300 colegios en España han incorporado el ajedrez como asignatura obligatoria, siguiendo recomendaciones de la UNESCO y del Parlamento Europeo para fomentar el desarrollo cognitivo y emocional de los estudiantes (El ajedrez, ¿una potente herramienta pedagógica?, 2017).

4. Estado del arte

En este apartado se recoge una revisión de los principales avances y desarrollos relacionados con la inteligencia artificial aplicada al ajedrez, con el fin de exponer la distintas alternativas y el camino escogido para el desarrollo de este TFG.

Actualmente, la totalidad de las inteligencias artificiales destinadas al ajedrez, diferencian dos tipos de módulos internos: la **búsqueda** y la **evaluación**. La dinámica que siguen estos módulos es que la búsqueda se encarga de hallar posibles posiciones futuras del tablero, mientras que en la evaluación se asigna una puntuación a cada una de esas posiciones, para determinar qué movimiento derivará en una mejor situación de juego. Para el estudio del estado del arte respecto a los motores de ajedrez, me centraré en hablar de estos dos aspectos, además de indagar en la **representación del tablero**.

4.1. Representación del tablero

Un programa de ajedrez necesita una representación interna del tablero para mantener las posiciones durante su búsqueda, evaluación y desarrollo de partidas. Además de modelar el tablero con la ubicación de sus piezas, se requiere información adicional para especificar completamente una posición, como el lado que debe mover, los derechos de enroque de cada bando, la posible casilla objetivo de captura al paso y el número de movimientos reversibles para seguir la regla de los cincuenta movimientos.

Para empezar, profundizaremos en las estructuras de datos para representar el tablero y la ubicación de sus piezas. Podemos diferenciar representaciones centradas en piezas y casillas, así como soluciones híbridas.

4.1.1. Centrada en las piezas

En la representación centrada se busca mantener un seguimiento de la casilla que ocupada cada pieza aún en el tablero. Este seguimiento se puede realizar, por ejemplo, con listas o vectores de longitud fija (Vučković, 2008).

La representación basada en listas, aunque más sencilla a la hora de implementar, perjudica más el rendimiento del programa, y empezar con una base poco eficiente, podría afectar enormemente a los cálculos más avanzados que se harán posteriormente.

Una alternativa más viable es emplear un vector de 32 posiciones, que representarían las 16 piezas de cada bando. Este método es mucho más llamativo, principalmente por una razón, y es que podemos dividir los índices del 0 al 15 para que pertenezcan a un bando, y del 16 al 31 para que pertenezcan al contrario, facilitando así el proceso de iterar sobre las piezas a la hora de generar los movimientos, sin tener que buscarlas entre todo el conjunto. Una representación de este estilo usaba el programa de ajedrez MicroChess de 1976:

	PIEZAS BLANCAS	PIEZAS NEGRAS
Rey	0	16
Reina	1	17
Torre Rey	2	18

Torre Reina	3	19
Alfil Rey	4	20
Alfil Reina	5	21
Caballo Rey	6	22
Caballo Reina	7	23
Peón Torre Rey	8	24
Peón Torre Reina	9	25
Peón Caballo Rey	10	26
Peón Caballo Reina	11	27
Peón Alfil Rey	12	28
Peón Alfil Reina	13	29
Peón Rey	14	30
Peón Reina	15	31

Tabla 1: Representación del tablero de MicroChess. Fuente: Chess Programming Wiki (Piece-Lists)

Sin embargo, la opción por la que se decantan la mayoría de los motores de ajedrez son los denominados *bitboards*, los cuales son conjuntos de 64 bits que representan un aspecto del tablero. Este aspecto puede ser, por ejemplo: todas las blancas, todos los peones, los alfiles negros, las piezas que atacan al rey, etc.

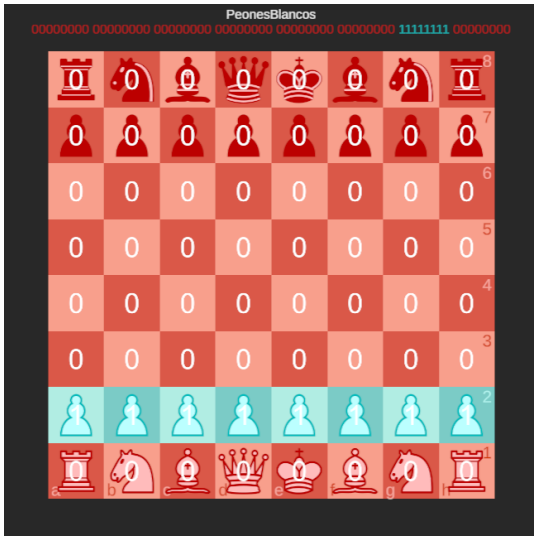


Ilustración 4: Visualización del bitboard de los peones blancos en la posición inicial

El hecho de que sólo puedan representar un aspecto del tablero se debe a la limitación de que cada bit representa una casilla del tablero (ocupada o vacía). Sin embargo, esta debilidad también es su fortaleza, y es que se pueden realizar operaciones a nivel de bit, las cuales son extremadamente rápidas, y otorgarnos información muy valiosa tanto para la generación de movimientos, la búsqueda o la evaluación.

Un ejemplo de la utilidad de los *bitboards* es saber si el rey está en jaque o no. Suponiendo que tenemos un *bitboard* que indica las casillas que atacan todas las piezas del rival, y un *bitboard* con la casilla en la que se encuentra el rey aliado, podemos saber si el rey está siendo atacado:

$$Rey\ atacado = Ataques\ rivales \wedge Rey\ aliado$$

El rey está en jaque si la intersección entre el *bitboard* de ataques rivales y el *bitboard* que representa la posición del rey no es nula:

$$Rey\ atacado \neq 0 \rightarrow El\ rey\ está\ en\ jaque$$

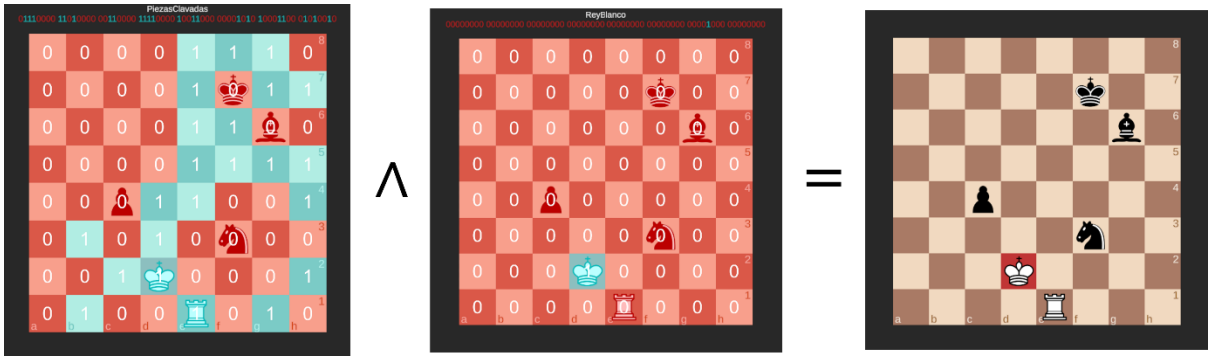


Ilustración 5: Intersección del bitboard de ataques rivales y el del rey aliado

Para el desarrollo del juego de ajedrez que tiene como objetivo este proyecto, emplearé esta representación basada en *bitboards*, puesto que es la que usan la mayor parte de motores de ajedrez más exitosos como Stockfish, y sus ventajas de rendimiento parecen ser muy superiores al del resto de alternativas de representación. En concreto, los *bitboards* que usaré en mi programa serán 12, que estarán dispuestos en un arreglo, y se corresponderán con las siguientes piezas:

0	Rey blanco	6	Rey negro
1	Reinas blancas	7	Reinas negras
2	Torres blancas	8	Torres negras
3	Alfiles blancos	9	Alfiles negros
4	Caballos blancos	10	Caballos negros
5	Peones blancos	11	Peones negros

Tabla 2: Bitboards empleados en el programa

4.1.2. Centrada en las casillas

La representación centrada en las casillas implementa una relación inversa a lo visto en la representación centrada en las piezas: ¿está una casilla vacía o está ocupada por una pieza en particular?

La representación centrada en las casillas más popular es ***mailbox***, y se basa en un arreglo que contiene códigos de piezas para cada casilla, de manera que dependiendo de dicho código se puede saber si la casilla está libre, u ocupada por una determinada pieza (Board representation in chess programming, 1999). Además, en este tipo de representaciones, es común que existan casillas adicionales en el arreglo (más de 64) para indicar aquellas que están fuera del tablero. De este modo, si al mover una pieza te se accede a una casilla que está marcada como inválida, se puede saber rápidamente el alcance de los movimientos, evitando comprobar constantemente si estás dentro de los límites del tablero.

--	--	--	--	--	--	--	--	--	--	--	--
--	--	--	--	--	--	--	--	--	--	--	--
--	--	A8	B8	C8	D8	E8	F8	G8	H8	--	--
--	--	A7	B7	C7	D7	E7	F7	G7	H7	--	--
--	--	A6	B6	C6	D6	E6	F6	G6	H6	--	--
--	--	A5	B5	C5	D5	E5	F5	G5	H5	--	--
--	--	A4	B4	C4	D4	E4	F4	G4	H4	--	--
--	--	A3	B3	C3	D3	E3	F3	G3	H3	--	--
--	--	A2	B2	C2	D2	E2	F2	G2	H2	--	--
--	--	A1	B1	C1	D1	E1	F1	G1	H1	--	--
--	--	--	--	--	--	--	--	--	--	--	--
--	--	--	--	--	--	--	--	--	--	--	--

Tabla 3: Representación del tablero en mailbox

4.2. Búsqueda

4.2.1. Minimax

Si bien antes se ha explicado que los motores de ajedrez han tendido por transformar sus sistemas heurísticos clásicos, en otro basados en redes neuronales, esto no es del todo cierto. El hecho es que aún mantienen un acercamiento clásico cuando hablamos de búsqueda de movimientos. El más conocido y usado de estos métodos clásicos para búsqueda es el algoritmo **Minimax**, el cual es un algoritmo empleado en la toma de decisiones, que busca minimizar la pérdida máxima esperada, es decir, busca la situación de juego que minimice su pérdida, suponiendo que el oponente hará su mejor jugada. Dentro de este algoritmo, la máquina que ejecuta la búsqueda recibe el nombre de Max, puesto que busca minimizar su pérdida o, dicho de otra forma, maximizar su victoria, mientras que el oponente contra el que se enfrenta recibe el nombre de Min, puesto que se pretende minimizar su puntuación.

En el desarrollo de este proyecto, Minimax será la elección escogida para el campo de la búsqueda, debido a ser la implementación más popular en los motores de ajedrez con mayor rendimiento como Stockfish (Stockfish) o Komodo (Komodo Chess).

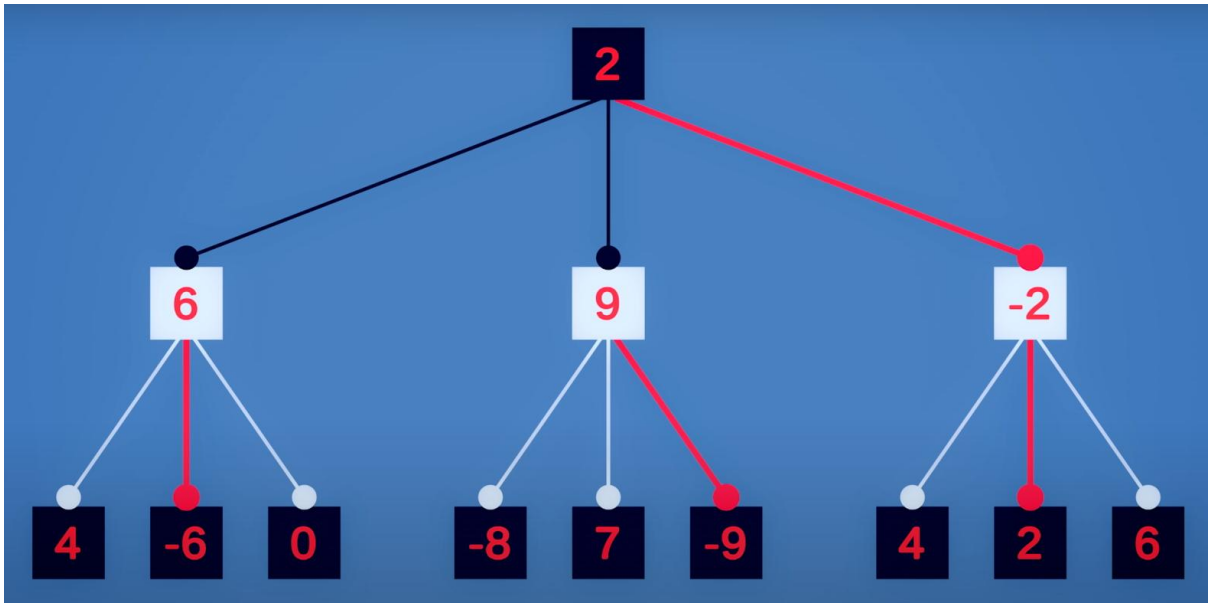


Ilustración 6: Diagrama del funcionamiento del algoritmo Minimax. Fuente: YouTube (Lague, 2021)

Si Minimax busca minimizar la pérdida del jugador que está tomando la decisión, entonces siempre va a escoger la mejor jugada, en cualquier caso. Sin embargo, esto no es cierto del todo, y es que una posición del tablero de ajedrez puede resultar muy buena para la función de evaluación en ese momento exacto, pero un movimiento después puede tornarse en una situación muy desventajosa.

Es por esto, que la clave de Minimax es la profundidad máxima a la que se realiza la búsqueda, pero esto tiene un gran inconveniente, y es que antes se ha visto el elevadísimo número de situaciones legales que existen en el ajedrez, y que según avanzan los turnos respecto a una situación inicial, las posibles jugadas que se pueden realizar aumentan exponencialmente. Como se puede ver la siguiente tabla, partiendo de la situación inicial del tablero, existen 20 movimientos para el turno de las blancas, después, las negras podrán realizar 20 movimientos como respuesta para cada uno de los 20 movimientos que pudieron hacer las blancas, es decir, 400 movimientos, y así sucesivamente:

Profundidad	Nodos	Capturas	E.p.	Enroques	Promociones
0	1	0	0	0	0
1	20	0	0	0	0
2	400	0	0	0	0
3	8,902	34	0	0	0
4	197,281	1576	0	0	0
5	4,865,609	82,719	258	0	0
6	119,060,324	2,812,008	5248	0	0
7	3,195,901,860	108,329,926	319,617	883,453	0
8	84,998,978,956	3,523,740,106	7,187,977	23,605,205	0
9	2,439,530,234,167	125,208,536,153	319,496,827	1,784,356,000	17,334,376

Tabla 4: Número de jugadas posibles según la profundidad de búsqueda. Fuente: Chess Programming Wiki (Perft results)

4.2.2. Minimax con poda alfa-beta

Debido a que la máxima profundidad alcanzada en la búsqueda es crucial para obtener la mejor jugada, existe lo que se conoce como poda **alfa-beta**, la cual es una técnica que reduce el número de nodos evaluados en un árbol de juego por el algoritmo Minimax (Edwards, 1963), lo que nos permite alcanzar una profundidad mayor de búsqueda, en menos tiempo.

El funcionamiento del algoritmo alfa-beta es el siguiente:

1. En la ilustración 7, podemos ver que se han evaluado los tres nodos hoja, correspondientes a la primera rama del árbol de búsqueda. Dadas sus evaluaciones, sabemos que debemos escoger el valor mínimo, es decir, -6, puesto que es lo que haría el rival para perjudicarnos lo máximo posible, minimizar nuestra puntuación.

2. En la misma imagen, descubrimos que en el primer nodo hoja de la segunda rama, el valor de la evaluación es de -8. Sabiendo esto, podemos concluir que de seguir buscando por esta rama, el rival nos dejaría con, al menos, una puntuación de -8, que es menor que la puntuación más baja entre los tres nodos hoja de la primera rama.

3. Sabiendo esto, no merece la pena seguir invirtiendo tiempo el buscar por esta segunda rama y, por lo tanto, debemos de "*podarla*".

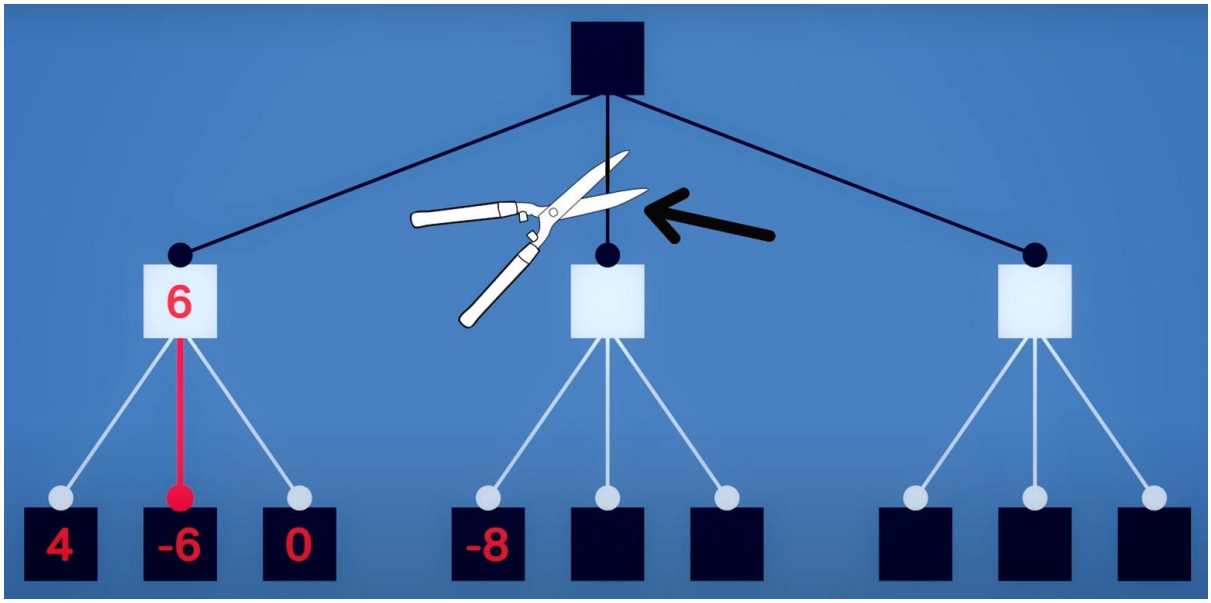


Ilustración 7: Diagrama del funcionamiento de la poda alfa-beta. Fuente: YouTube (Lague, 2021)

4.2.3. Árbol de búsqueda Monte Carlo

Sin embargo, Minimax no es la única base desde la que partir para desarrollar un sistema de búsqueda. El algoritmo de búsqueda Monte Carlo (conocido como *Monte Carlo tree search*) es muy usado en juegos de estrategia, no solamente el ajedrez. Este algoritmo explora un árbol de decisiones simulando multitud de situaciones de tablero de forma aleatoria. A través de estas simulaciones (*playouts* (Coulom, 2007)), va aprendiendo qué caminos son más prometedores y prioriza la búsqueda hacia ellos.

La estrategia del algoritmo consta de cuatro fases, que se repiten mientras aún se considere que haya tiempo de búsqueda:

1. **Selección**: Se recorre el árbol desde el nodo raíz hasta seleccionar un nodo hoja que aún no se ha añadido al árbol.

2. **Expansión**: Se añade el nodo hoja al árbol.

3. **Simulación**: Se realizan movimientos, internamente en el algoritmo, hasta el final de la partida. El resultado puede ser 1, 0 o -1.

4. **Retropropagación**: Se propagan los resultados hacia atrás en el árbol.

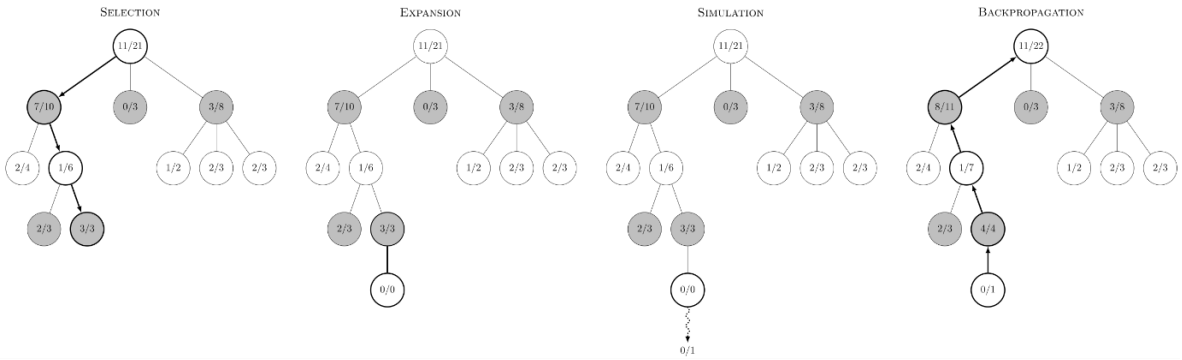


Ilustración 8: Diagrama de las cuatro fases del árbol de búsqueda Monte Carlo. Fuente: Wikipedia (Monte Carlo tree search)

Para manejar el flujo sobre qué nodo explorar, y evitar favorecer jugadas perdedoras, se emplea una fórmula UTC (*Upper Confidence bounds applied to Trees*) (Coulom, 2007).

4.3.Evaluación

Las funciones de evaluación, en el caso del ajedrez, han sido casi siempre una heurística basada en el cálculo del material de cada jugador o en la valoración de la posición de las piezas mediante un mapa que pondera cada casilla en función del tipo de pieza y el color. La realidad en cuanto a los actuales motores de ajedrez es que la mayoría implementan una solución híbrida, en la que la evaluación se hace mediante un algoritmo heurístico clásico, a la vez que son tratadas mediante una red neuronal.

En el caso concreto de Stockfish, estuvo usando una función heurística clásica para la evaluación hasta agosto de 2020, cuando empezó a usar un sistema de evaluación híbrido, junto a una ***efficiently updatable neural network*** (NNUE) (Stockfish). Este tipo de red neuronal fue creada de forma especial para juegos como el ajedrez o, en concreto, el shogi. En julio de 2023, Stockfish finalmente eliminó su función heurística clásica, y transicionó a una evaluación puramente basada en redes neuronales (Stockfish).

Para la realización de este proyecto, se creará una función de evaluación clásica, sin hacer uso de ningún tipo de red neuronal, puesto que he podido investigar que para conseguir una función evaluadora basada en redes neuronales y que realmente suponga un reto en la partida, es necesario un trabajo muy exhaustivo y que requiere de un tiempo considerable para desarrollar (How to Create a Chess Engine with TensorFlow (Python), 2024). Debido a que poseo más experiencia en la creación de funciones heurísticas clásicas, optaré por la primera opción, y poder conseguir así un mejor resultado.

5. Normas y referencias

En este apartado se comentarán los métodos y herramientas empleados para la elaboración de este trabajo, y las áreas del mismo en las que han sido empleados. Además, se comentarán los modelos utilizados, y se hará un repaso de los prototipos elaborados durante el proceso de creación y las métricas abordadas para evaluar el proyecto.

5.1. Métodos

La principal metodología usada para la elaboración de la documentación y la planificación del proyecto ha sido el **Proceso Unificado**. Esto se ve reflejado de manera primordial en los dos primeros anexos de este trabajo, en los que se incluye la documentación relativa a las disciplinas de requisitos, análisis y diseño.

Esta metodología ha sido selecciona por dos grandes motivos:

- Elaborar una documentación correcta para el proyecto, que conlleve a un producto sólido y alineado con los objetivos.

- Se cuenta con experiencia previa en el Proceso Unificado, al haber sido empleado en varias asignaturas durante la carrera.

Es posible que al usar otras metodologías como el método Scrum, se menoscabase la eficacia del resultado, debido a estar más centrado el reducir el tiempo invertido en hacer la documentación. Además, está el hecho de que el proceso unificado es la única metodología de ingeniería del software con la que realmente se ha trabajado, por lo que emplear otro método podría haber llevado a errores en el proyecto.

Las características principales del Proceso Unificado son las siguientes:

- **Iterativo e incremental**: El software se construye en ciclos (iteraciones) que producen versiones parciales pero funcionales del sistema.

- **Dirigido por casos de uso**: El desarrollo se enfoca en construir el sistema para el usuario, siguiendo lo descrito en los casos de uso.

- **Centrado en la arquitectura**: Se pone énfasis en definir una arquitectura robusta desde las primeras etapas del desarrollo.

- **Basado en componentes**: Cumple con los requisitos de llevar a cabo un diseño modular, reutilizable y mantenible del sistema.

El ciclo de vida del software en el Proceso Unificado se divide en cuatro fases principales, cada una con objetivos y entregables específicos:

- **Inicio**: Se define el alcance del proyecto y se comprueba la viabilidad de este. Se identifican los actores y los casos de uso clave.

- **Elaboración**: Se establece una arquitectura base para el proyecto y se refinan y especifican los casos de uso.

- **Construcción**: Basándose en la arquitectura del proyecto, se desarrolla el producto completo y funcional.

- **Transición**: Se corrigen errores y se hacen ciertas mejores sobre el producto, y se prepara para ser entregado al cliente.

Por otro lado, en cada fase del Proceso Unificado existen varias disciplinas que pueden ser llevadas a cabo en mayor o en menor medida. Las disciplinas más importantes son: Requisitos, Análisis, Diseño, Implementación, Pruebas y Gestión del proyecto. Cada una de estas disciplinas proporciona un modelo, una representación abstracta, que ayuda que el producto final sea posible, entendiéndolo y diseñando sus partes.

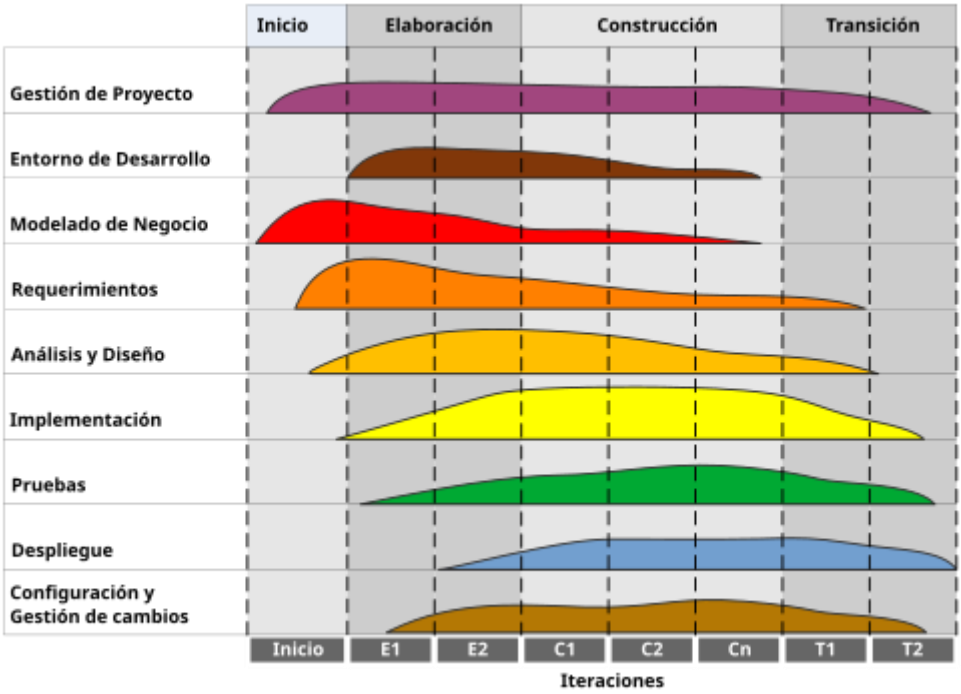


Ilustración 9: Diagrama de las fases y disciplinas del Proceso Unificado. Fuente: Wikipedia (Proceso unificado)

5.2.Herramientas

A continuación, se describe todo el abanico de herramientas utilizadas en la realización del proyecto, las cuales podemos repartir entre tres categorías:

- **Herramientas propias de la implementación**: Unity, Visual Studio Code, Stockfish, Chess FEN Viewer.

- **Herramientas metodológicas**: Visual Paradigm, Microsoft Project, EZEstimate.

- **Herramientas para la elaboración del TFG**: Diagrams.net.

5.2.1. Unity

Como herramienta principal para la creación del programa, está Unity. Se trata de un motor de videojuegos muy versátil; puede ser usado tanto para proyecto en 2D como 3D, además, está soportado en una gran cantidad de plataformas, por lo que es un primer gran llamamiento. Unity tiene un sistema de *scripting* basado en C#.

5.2.2. Visual Studio Code

Es un editor de código ligero y muy intuitivo, que mantiene una interfaz muy amigable y lo hace fácil de usar, sin necesidad de un aprendizaje extra. Aunque, esencialmente es un editor de código, el gran número de extensiones que tiene en su *marketplace*, lo puede hacer actuar de manera similar a un entorno de desarrollo. Sin embargo, esto no ha sido para nada necesario para este desarrollo, puesto que la compilación a formado parte de Unity.

5.2.3. Stockfish

El propio motor de ajedrez, que ha sido mencionado anteriormente, ha servido como herramienta para la depuración del código del programa. En concreto, se ha usado como comparar los resultados de la función ***perft*** (*performance test, move path enumeration*) de Stockfish, con los de mi programa. Este tipo de depuración ha sido muy útil, puesto que es muy común cometer errores en la programación de la generación de movimientos.

```
go perft 5
info string Available processors: 0-15
info string Using 1 thread
info string NNUE evaluation using nn-1c0000000000.nnue (133MiB, (22528, 3072, 15, 32, 1))
info string NNUE evaluation using nn-37f18f62d772.nnue (6MiB, (22528, 128, 15, 32, 1))
a2a3: 181046
b2b3: 215255
c2c3: 222861
d2d3: 328511
e2e3: 402988
f2f3: 178889
g2g3: 217210
h2h3: 181044
a2a4: 217832
b2b4: 216145
c2c4: 240082
d2d4: 361790
e2e4: 405385
f2f4: 198473
g2g4: 214048
h2h4: 218829
b1a3: 198572
b1c3: 234656
g1f3: 233491
g1h3: 198502

Nodes searched: 4865609
```

Ilustración 10: Salida del comando perft a profundidad 5 en Stockfish

5.2.4. Chess FEN Viewer

Se trata de una herramienta *online*, que permite obtener la cadena en formato FEN, dada una posición del tablero que se establezca mediante su interfaz gráfica. Esto, en combinación con Stockfish, ha permitido aprovechar de forma óptima el tiempo de desarrollo, no teniendo que transformar una posición de tablero a FEN manualmente. La notación FEN es esencial, puesto que es la manera en la que puede comunicar las posiciones a Stockfish, o casi cualquier motor de ajedrez.

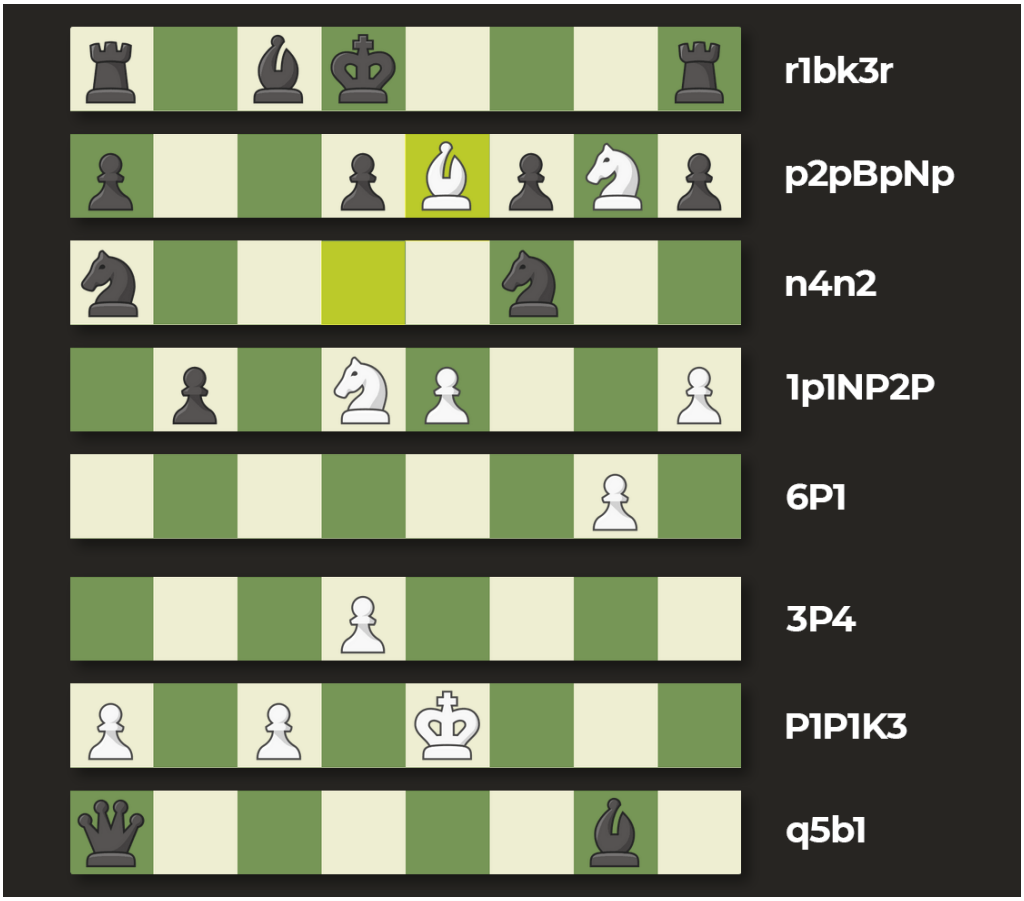


Ilustración 11: Representación de una posición del tablero en formato FEN. Fuente: Chess.com (Forsyth-Edwards Notation (FEN))

5.2.1. Visual Paradigm

Visual Paradigm es una herramienta de modelado visual y gestión de proyectos de software que facilita la creación de diagramas UML (Lenguaje Unificado de Modelado) y otros tipos de representaciones gráficas para el análisis, diseño y documentación de sistemas. Ha sido el *software* empleado durante la carrera, para elaborar todos los diagramas relativos a la Ingeniería del Software, y es por eso que se ha usado para este proyecto con el mismo propósito.

5.2.2. Microsoft Project

Microsoft Project es una herramienta de gestión de proyectos que permite planificar, organizar y controlar proyectos de manera eficiente. De nuevo, ha sido una aplicación usada durante el curso para la organización de proyectos, y es por eso por lo que usará en este trabajo para realizar el mismo objetivo.

5.2.3. EZEstimate

Es una herramienta especializada en la elaboración de presupuestos y estimaciones para proyectos, permitiendo usar el modelo de costes basado en puntos de casos de uso y, por lo tanto, pudiendo dar valor a la complejidad de los casos de uso, la complejidad de los actores, los factores de complejidad técnica y los factores de complejidad del entorno.

5.2.1. Diagrams.net

Antes denominado *Draw.io*, es una herramienta gratuita y en línea, destinada a la creación de diagramas. Al ser en línea, permite usarse fácilmente desde cualquier navegador e incluso vincularla con Google Drive, permitiendo concentrar todos los archivos en un mismo lugar. Ha sido empleada para la elaboración de *wireframes* del diseño de interfaces.

5.3.Modelos

5.3.1. Modelo de decisión

El modelo empleado en el programa de ajedrez para poder elegir el mejor movimiento se conoce como modelo de decisión. Se basa el buscar los posibles movimientos, para después evaluar cada posición del tablero mediante los valores numéricos o puntajes asignados a cada posición resultante, que indican qué tan favorable es cada opción para el jugador

5.4.Prototipos

En este apartado, describiré los prototipos empleados en el diseño de la interfaz. Esta fase del desarrollo fue iniciada, una vez se completaron los diagramas de las disciplinas de Requisitos, Análisis y Diseño.

5.4.1. Mapa del sitio

Para empezar con el diseño de la interfaz, fue necesario empezar con un Mapa del sitio, el cual es un esquema en el que se distinguen las diversas pantallas o interfaces con las que interactuará el usuario. En este proyecto, se han distinguido cinco pantallas:

- **Menú principal**: Será la primera interfaz con la que el usuario se topará al iniciar la aplicación, es por tanto que debe servir como presentación de la misma y ofrecer un conjunto de botones que naveguen a los submenús del programa.

- **Configuraciones previas**: Es el preámbulo a la pantalla de partida, y agrupa todas las funcionalidades que permiten al usuario ajustar los parámetros de la partida.

- **Partida**: Es la interfaz con la que interactuará el usuario mientras juegue, y su elemento principal debe ser el tablero.

- **Promoción**: En el momento en el que un peón promocione o corone, esta pantalla aparecerá para permitir al usuario seleccionar el tipo de pieza al que promocionar. Debe contener un botón para cada uno de los 4 tipos de piezas a la que puede promocionar un peón, y no debe ser excesivamente grande visualmente, puesto que se mostrará encima de la pantalla de partida.

- **Final de partida**: Comunica al usuario que la partida ha concluido, y le muestra el resultado de la misma. También debe permitir regresar al usuario a la pantalla del menú principal.

Habiendo definido esto, el aspecto que tiene el diagrama de sitio es este:

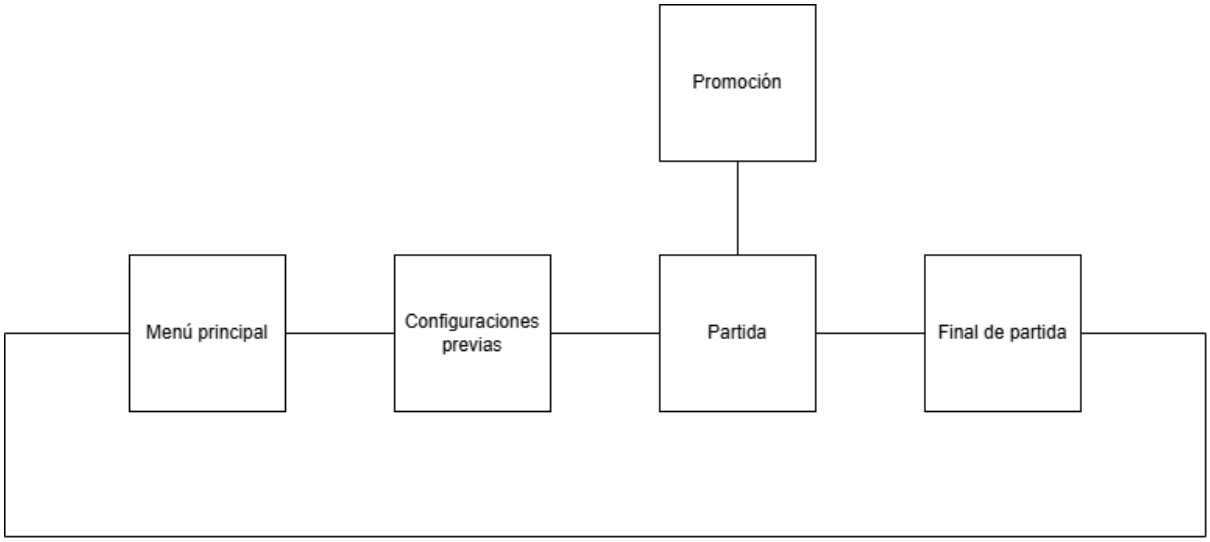


Ilustración 12: Mapa del sitio

5.4.2. Wireframes

Una vez se ha planteado el mapa del sitio, los siguientes prototipos estarán relacionas con la distribución y funcionalidad de cada pantalla. Para realizar este prototipado, se empleará un tipo de esquematización denominada *wireframes*. Los *wireframes* debe actuar como un "*esqueleto*" a la hora de hacer la versión final de la interfaz, y en ningún momento deben profundizar en el funcionamiento de la misma, o en el color o tipografía de sus elementos y texto.

Empezando con el ***wireframe* del menú principal**, este constará de un título o nombre para la aplicación, y un conjunto de botones, los cuales típicamente indicarán: "*jugar*", "*salir*", "*opciones*", etc.

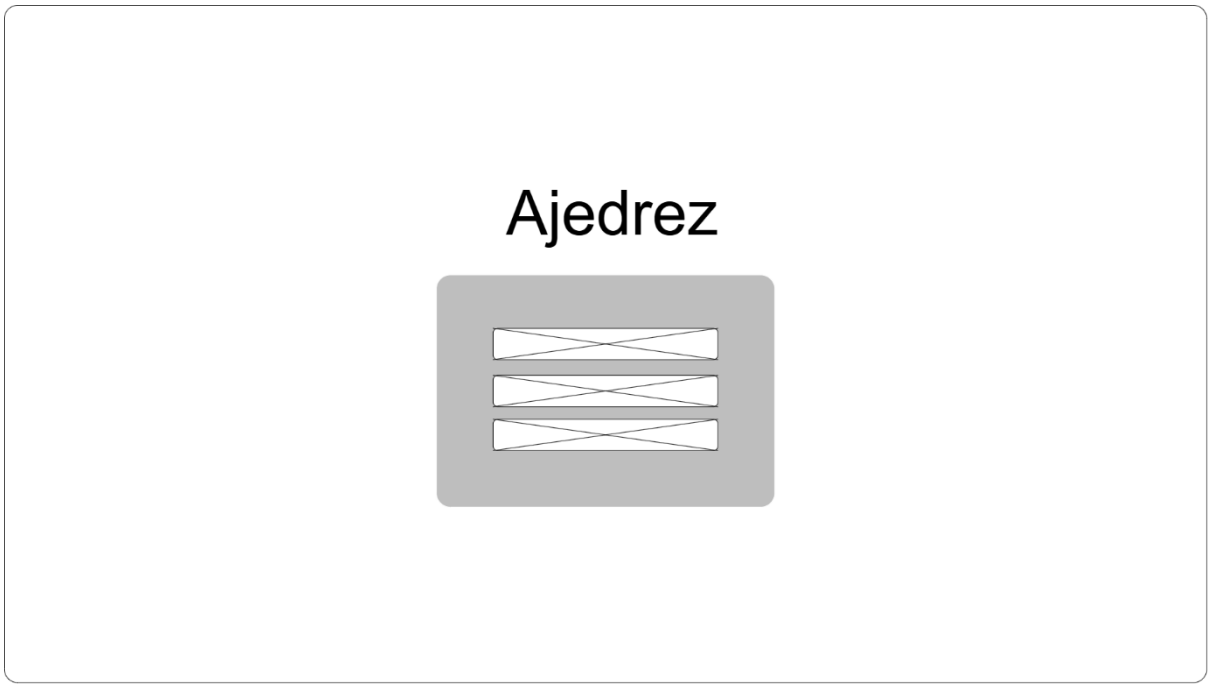


Ilustración 13: Wireframe del menú principal

Para el caso de las **configuraciones previas**, este *wireframe* contendrá todas las opciones definidas en los objetivos del sistema. En el centro de la pantalla, se hallará un tablero en el que se podrá visualizar la posición de partida que se va a jugar, la cual se cargará mediante una cadena de caracteres en notación FEN que se introduzca en el formulario que habrá justo debajo de tablero. A la derecha de este formulario, se encuentra un botón para intercambiar los colores de las piezas de los jugadores, los cuales se indican en el margen izquierdo y derecho de la pantalla, respectivamente.

Debajo de donde se indicar el color de cada jugador, habrá un selector, a la izquierda, del tipo de jugador; humano o inteligencia artificial, el cuál determinará el contenido del bloque de la derecha. Si se elige jugador humano, aparecerá un formulario para introducir el nombre del mismo, y si se elige inteligencia artificial, aparecerá un selector para indicar la dificultad de esta (fácil, media o difícil).

Por último, encima del tablero, se encuentran dos selectores, siendo el de la izquierda para establecer la duración y el de la derecha para establecer el incremento de partida, representado de la forma estándar que se hace en ajedrez: 1+0, 3+5, 15+10, etc.

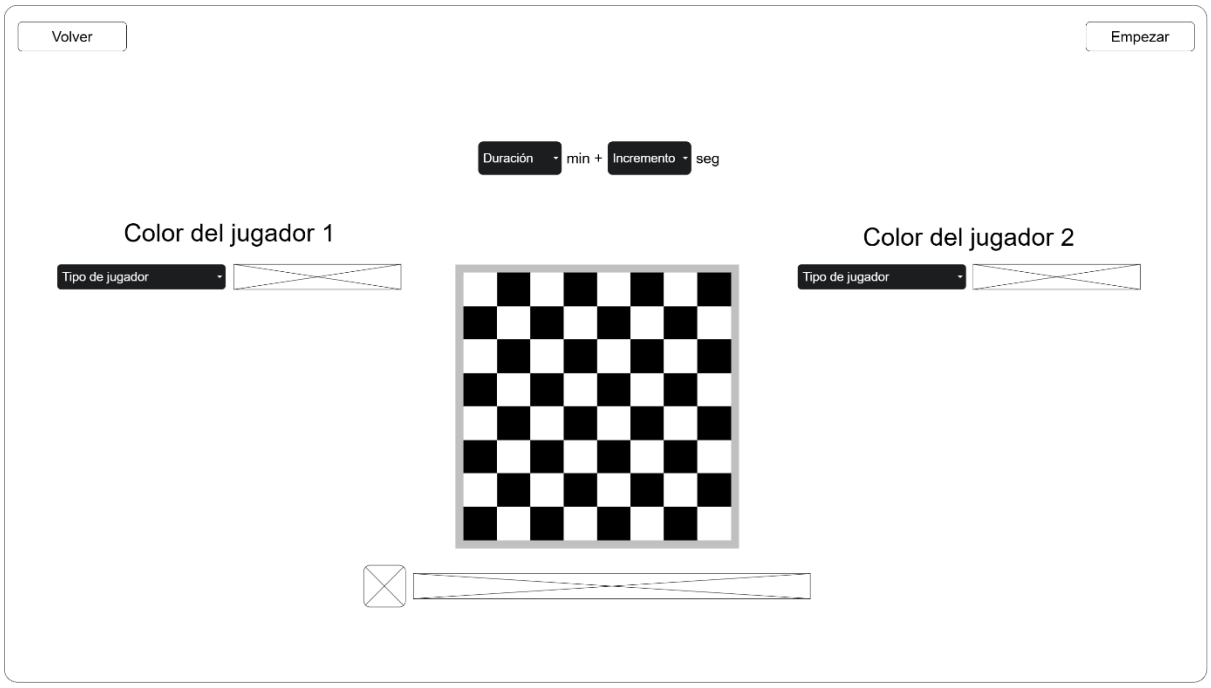


Ilustración 14: Wireframe de las configuraciones previas

En el ***wireframe* de la partida**, como antes se ha mencionado, habrá un tablero en el centro en el que se coloquen las piezas. En el margen derecho está el reloj de cada jugador, indicando su tiempo restante, y en las esquinas inferior derecha y superior izquierda, se encuentran los nombres del jugador 1 y 2, respectivamente. Estos nombres dependerán de los valores introducidos en las configuraciones previas.

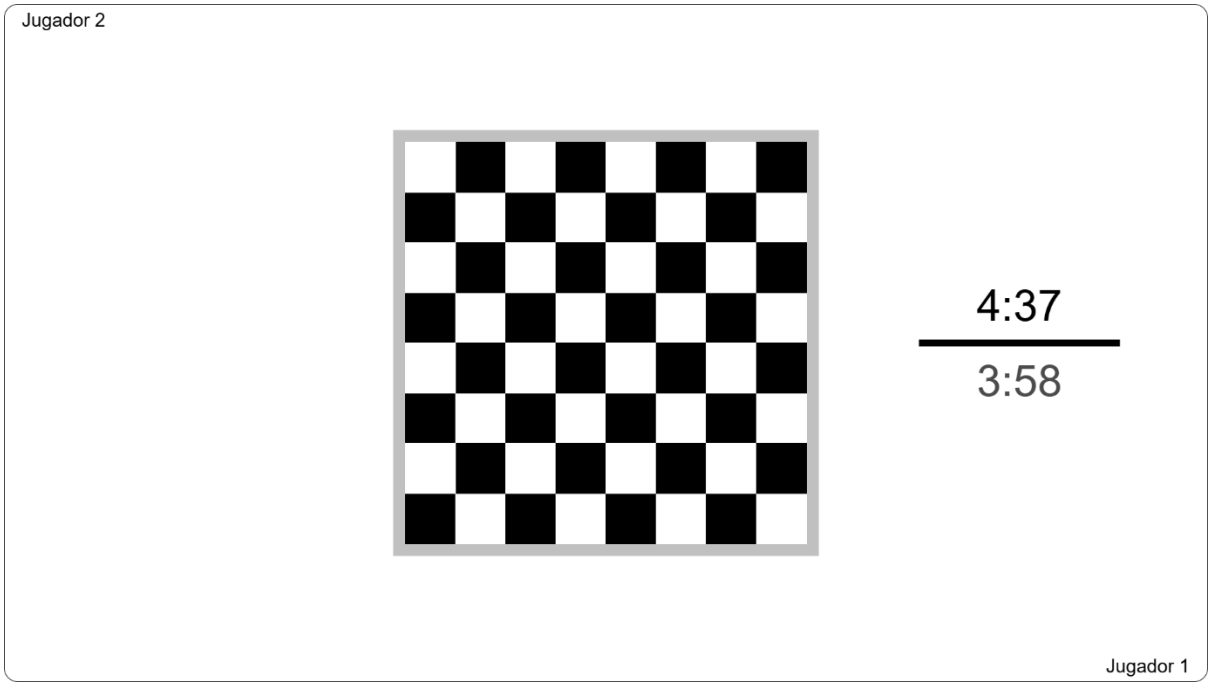


Ilustración 15: Wireframe de la partida

Para el ***wireframe* de promoción**, se debe poder seleccionar el tipo de promoción que desea el usuario, siendo las posibilidades: reina, torre, alfil o caballo. Cada una de estas opciones se representará con un botón en un pequeño menú que aparecerá sobre la pantalla de partida.

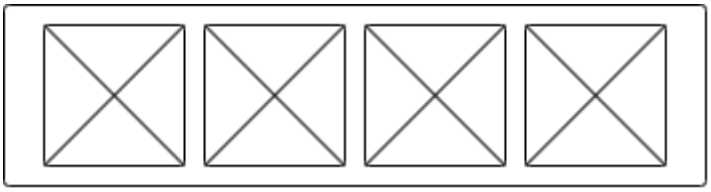


Ilustración 16: Wireframe de promoción

Por último, queda definir cómo será el ***wireframe* de la pantalla de final de partida**, la cual también aparecerá encima de la de partida, e informará del resultado de la misma. En la parte inferior de esta ventana, habrá un botón que permitirá volver a la pantalla del menú principal.

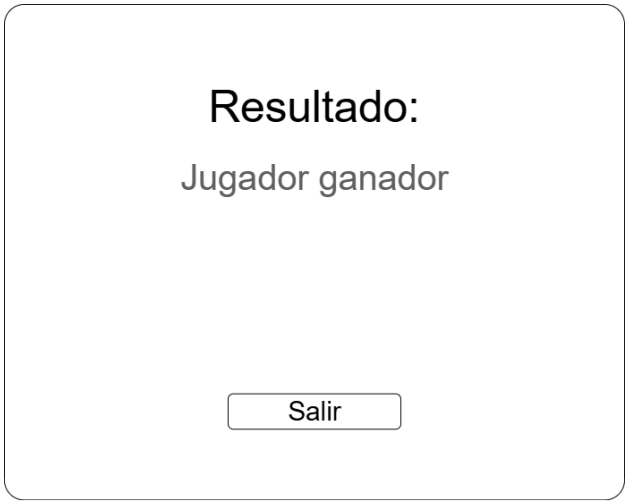


Ilustración 17: Wireframe de final de partida

5.5.Métricas

Para evaluar el rendimiento computacional del motor de ajedrez, se midió principalmente el tiempo de respuesta en diferentes niveles de profundidad de búsqueda. Esta métrica es fundamental para determinar la eficiencia del motor, ya que un tiempo de respuesta adecuado permite que el jugador o usuario reciba jugadas en un plazo razonable, mejorando la experiencia de uso. Además, se analizaron el consumo de recursos del sistema, como el uso de CPU y memoria, para garantizar que el motor pueda funcionar de manera óptima sin afectar el rendimiento general del equipo. Otro indicador importante fue la velocidad de nodos por segundo, que mide cuántas posiciones puede evaluar el motor en un segundo, reflejando su capacidad para explorar el árbol de juego de forma rápida y exhaustiva.

En cuanto a la calidad del juego, se utilizaron métricas como el rendimiento estimado en términos de puntuación ELO, mediante enfrentamientos simulados contra otros motores reconocidos. Esto permite situar el nivel de juego del motor dentro de un contexto competitivo y valorar su fuerza relativa. También se analizó el porcentaje de jugadas óptimas, comparando las decisiones del motor con las recomendaciones de bases de datos de partidas maestras o motores de referencia, lo que ayuda a identificar la precisión y efectividad en la selección de movimientos. Finalmente, se evaluó la tasa de errores, es decir, la frecuencia con la que el motor realiza movimientos claramente inferiores o tácticamente erróneos, para detectar áreas de mejora en su algoritmo de evaluación y toma de decisiones.

6. Requisitos iniciales

En este apartado se presenta un resumen de los principales requisitos del sistema definidos en el Anexo de "Especificaciones del sistema". Estos requisitos garantizar que el producto final cumpla con los objetivos planteados al inicio de esta memoria, y se clasifican en tres tipos:

- **Requisitos de información**
- **Requisitos funcionales**
- **Requisitos no funcionales**

6.1. Requisitos de información

Los requisitos de información son aquellos que definen qué datos necesita el sistema para funcionar correctamente y qué información debe generar, procesar, almacenar o mostrar. La siguiente tabla muestra los requisitos de información identificados para el sistema:

ID	Nombre
IRQ-01	Información sobre las piezas
IRQ-02	Información sobre los movimientos
IRQ-03	Información sobre las partidas
IRQ-04	Información sobre los jugadores
IRQ-05	Información sobre la inteligencia artificial
CRQ-01	Niveles de dificultad de la inteligencia artificial

Tabla 5: Requisitos de información

6.2. Requisitos funcionales

Los requisitos funcionales son aquellas especificaciones que describen las funciones, comportamientos y servicios que debe ofrecer un sistema para cumplir con sus objetivos. Dentro de este apartado, se han realizado varios diagramas y tablas, empezando por el **diagrama de paquetes**. En este diagrama, se establece una organización de los casos de uso, se definiendo paquetes que los agrupan:

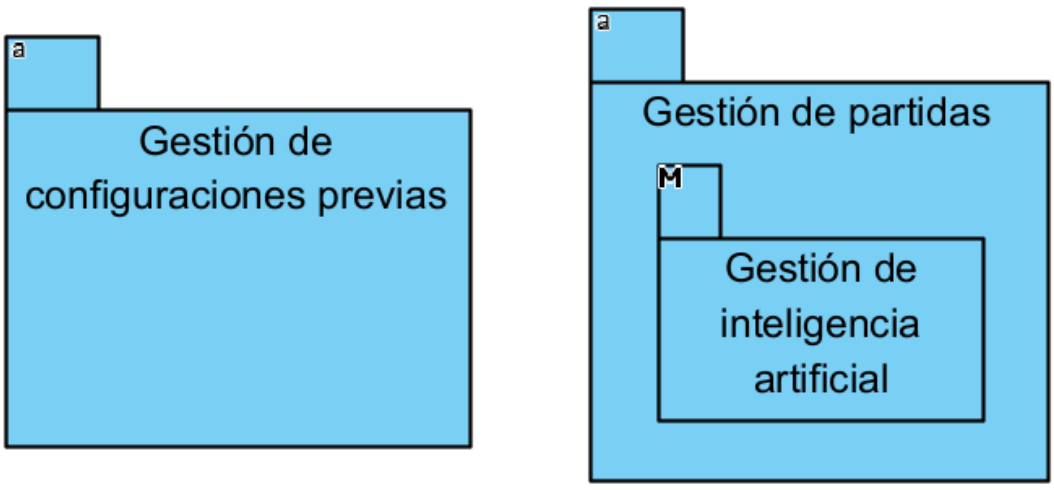


Ilustración 18: Diagrama de paquetes

El siguiente apartado consiste en identificar los **actores del sistema**, en este caso, hay dos tipos de actores:

ID	Nombre
ACT-01	Usuario
ACT-02	Sistema

Tabla 6: Actores del sistema

Una vez se sabe en qué paquetes van a estar agrupados los casos de uso y los actores que van a hacer uso de ellos, es momento de identificarlos y hacer los diagramas de casos de uso de cada paquete. Comenzando con el paquete de **Gestión de configuraciones previas**, su contenido sería el siguiente:

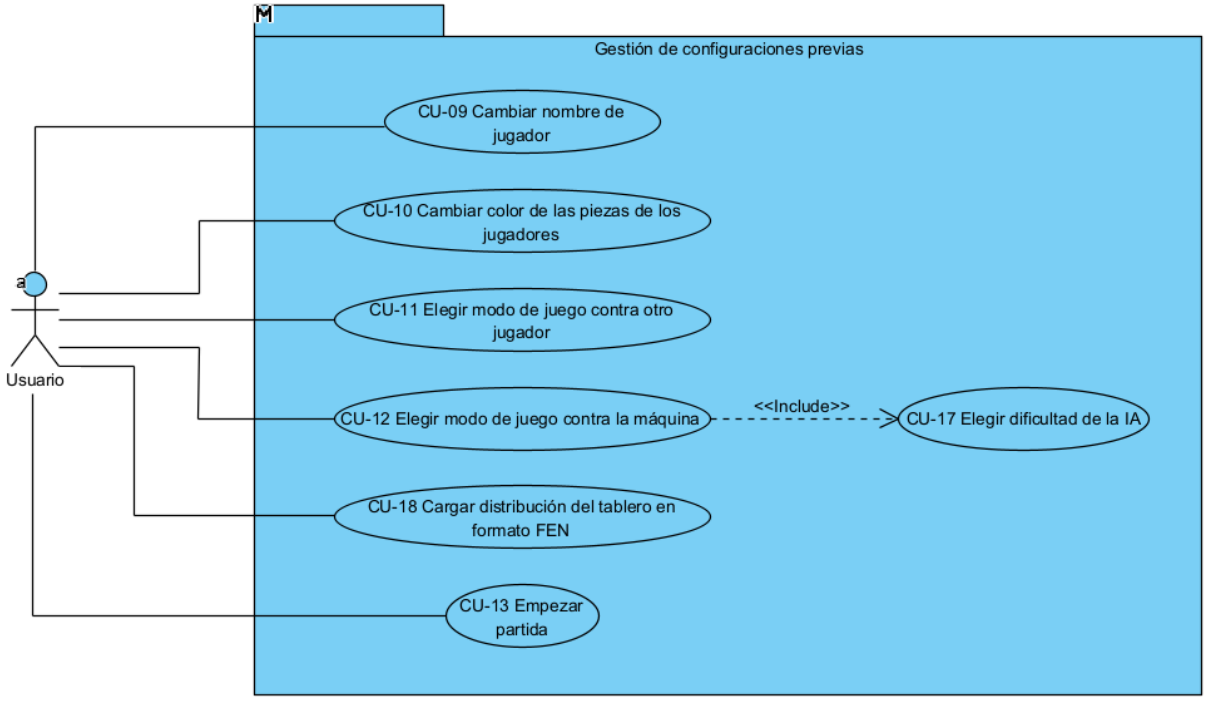


Ilustración 19: Diagrama de casos de uso de Gestión de configuraciones previas

Los otros dos paquetes que conforman el diagrama de casos de uso es el paquete de **Gestión de partida** y su subpaquete **Gestión de Inteligencia Artificial**:

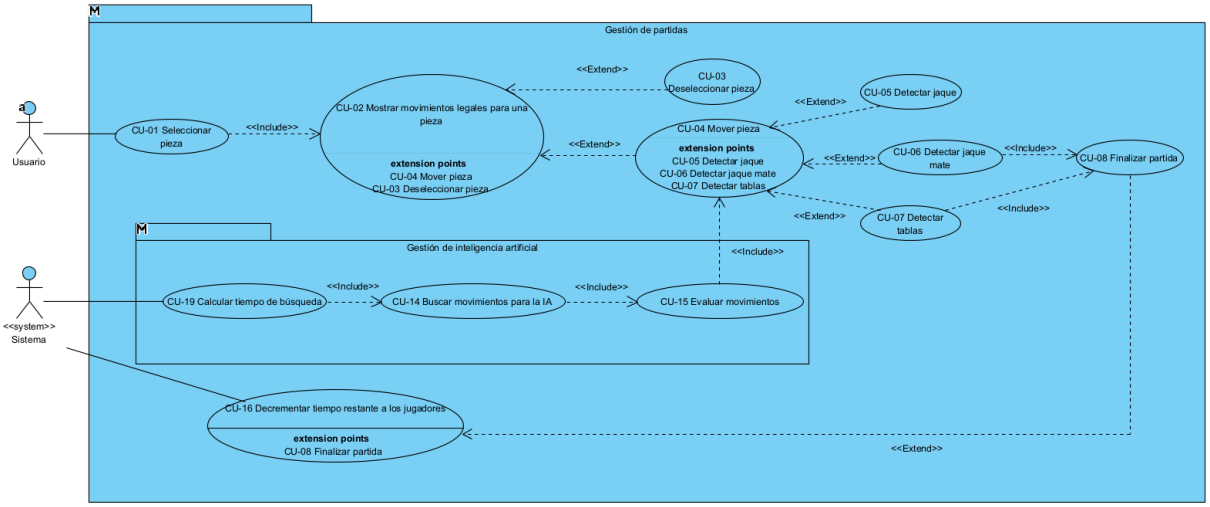


Ilustración 20: Diagrama de casos de uso de Gestión de partidas y Gestión de inteligencia artificial

Por último, en la siguiente tabla se refleja el conjunto de los **casos de uso**, con una breve descripción sobre su funcionalidad:

ID	Nombre	Descripción
CU-01	Seleccionar pieza	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario solicite escoger una pieza del tablero
CU-02	Mostrar movimientos legales para una pieza	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario haya seleccionado una pieza del tablero que desee mover
CU-03	Deseleccionar pieza	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario haya cancelado la selección de una pieza
CU-04	Mover pieza	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando se desee mover una pieza del tablero
CU-05	Detectar jaque	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el sistema detecte una situación de jaque en el tablero
CU-06	Detectar jaque mate	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el sistema detecte una situación de jaque mate en el tablero
CU-07	Detectar tablas	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el sistema detecte una situación de tablas en el tablero
CU-08	Finalizar partida	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando se llegue a una situación de tablero que termine la partida
CU-09	Cambiar nombre de jugador	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario solicite cambiar el nombre de alguno de los jugadores
CU-10	Cambiar color de las piezas de los jugadores	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario solicite cambiar el color de las piezas con las que jugará cada jugador
CU-11	Elegir modo de juego contra otro jugador	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario solicite jugar contra otra persona de forma local
CU-12	Elegir modo de juego contra la máquina	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario solicite jugar contra la inteligencia artificial

CU-13	Empezar partida	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario solicite empezar la partida con las configuraciones establecidas
CU-14	Buscar movimientos para la IA	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el sistema tenga que buscar posibles movimientos que podrá jugar la inteligencia artificial
CU-15	Evaluar movimientos	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el sistema tenga que evaluar los posibles movimientos que puede jugar la inteligencia artificial
CU-16	Decrementar tiempo restante a los jugadores	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el sistema tenga que decrementar el tiempo restante al jugador correspondiente
CU-17	Elegir dificultad de la IA	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario solicite elegir la dificultad de la inteligencia artificial
CU-18	Cargar distribución del tablero en formato FEN	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el usuario solicite cargar una distribución del tablero a partir de una notación FEN
CU-19	Calcular tiempo de búsqueda	El sistema debe comportarse tal como se describe en el siguiente caso de uso, cuando el sistema solicite calcular el tiempo que se invertirá en la siguiente búsqueda de movimientos por parte de la IA

Tabla 7: Casos de uso

6.3. Requisitos no funcionales

En cuanto a los requisitos no funcionales, son aquellos que no describen lo que el sistema hace, sino cómo debe hacerlo, es decir, se centran en las cualidades, restricciones y características del sistema. Para este proyecto en concreto, los requisitos no funcionales han sido los siguientes:

ID	Nombre
NFR-01	Dificultad
NFR-02	Rendimiento

Tabla 8: Requisitos no funcionales

7. Restricciones y alcance

Ya se ha expuesto, en el apartado anterior, cuáles eran los requisitos no funcionales y, ahora, se indagará en ellos en profundidad:

- **Dificultad**: La dificultad es casi un requisito intrínseco por parte de una inteligencia artificial de ajedrez, sin embargo, hay que saber hasta qué nivel de dificultad se va a poder implementar esta inteligencia artificial. El simple hecho hacer que la máquina juegue al ajedrez por sí sola, es un objetivo lo suficientemente alto, pero al añadirle el hecho de mejorar ese algoritmo hasta el punto de que resulte difícil derrotarla, hace que mayor parte de la atención del proyecto, derive en este único módulo, la inteligencia artificial.

- **Rendimiento**: Aunque no es obvio, este requisito va de la mano de la dificultad de la inteligencia artificial. El hecho es que el rendimiento influye directamente en la rapidez de la búsqueda del mejor movimiento; cuanta más rapidez, mayor será la profundidad máxima que se alcance en la búsqueda. Es por esto por lo que debe ser un factor prioritario, y asegurar un mínimo de rendimiento, sobre todo en las funciones más esenciales del programa, que serán llamadas continuamente.

El impacto esperado de este trabajo se centra en ofrecer un producto funcional que permita a usuarios, con distintos niveles de conocimiento de ajedrez, practicar contra una inteligencia artificial ajustable en dificultad. A nivel educativo, un motor puede servir como herramienta de apoyo para introducir conceptos de estrategia, táctica y toma de decisiones. Además, desde un punto de vista técnico, este proyecto demuestra cómo se pueden aplicar algoritmos de inteligencia artificial clásicos de forma eficiente en entornos con recursos limitados, lo que resulta útil para sectores como el desarrollo de *software* educativo.

8. Estudio de alternativas y viabilidad

En este apartado se discutirá sobre la elección de las herramientas descritas anteriormente, y la viabilidad de otras alternativas. Empezando a hablar por **Unity**, la elección de este motor de videojuegos se resume en la familiaridad. Unity es una herramienta que se ha empleado en la asignatura de Animación Digital en la carrera y, aunque no se ha explorado con total profundidad en esta, es el motor de videojuegos del que mejores conocimientos poseo. Además, Unity usa *scripts* en C# para la programación de los proyectos, y C# es un lenguaje que también me resulta más familiar que C++ de Unreal o GDScript de Godot.

En relación con las herramientas metodológicas, estas han sido seleccionadas debido a haber sido usadas durante el grado en diversas asignaturas. Estas herramientas son: **Visual Paradigm**, **Microsoft Project** y **EZEstimate**. Aunque podría haber sido posible haber utilizado otro tipo de herramientas en este aspecto, debido a la necesidad tener que invertir más tiempo en el aprendizaje de esas otras herramientas, se convirtió una idea descartada.

En cuanto a **Diagrams.net**, esta herramienta resultó accesible, gratis y rápida de usar desde cualquier navegador. Los elementos que ofrece para crear los diagramas son sencillos pero muy útiles, y además se ofrece conectividad con Google Drive.

Con relación al lenguaje de programación escogido, **C#** es un lenguaje de alto nivel y muy productivo, lo que permite agilizar el trabajo de los programadores. Además, es un lenguaje orientado a objetos, lo que es perfecto en la relación al tipo de metodologías empleadas en este proyecto. Sin embargo, tiene un principal problema, y es que utiliza recolector de basura, y aunque, en ciertas aplicaciones, esto no resulte un problema en ningún momento, en el caso de los motores de ajedrez sí lo es.

Durante la búsqueda de movimientos para la inteligencia artificial, se instanciarán numerosas clases en memoria, que rápidamente se abandonarán, creando así basura de forma muy veloz. Esto que el recolector de basura se ejecute más veces de lo que sería esperable, ralentizando y perjudicando el rendimiento del sistema. La solución más evidente para esto habría sido emplear un lenguaje mucho mejor destinado para este propósito como es C++, pero debido al riesgo de conseguir incluso peores resultad que con C#, al tener nulos conocimientos previos sobre este otro lenguaje, hizo decantarme por C#.

Finalmente, el último tema a tratar en este apartado será sobre la posibilidad de **paralelizar la búsqueda de movimientos**. Aunque es realmente posible hacer la búsqueda paralela (Marsland, 1980), sucede que algoritmos como alfa-beta son inherentemente secuenciales, puesto que los valores de alfa y beta se actualizan continuamente a lo largo de la búsqueda y los cortes se deciden basándose en estos valores (How do I write a Multi threaded Alpha-Beta Search algorithm?). Por esta razón, obtener mejoras en la velocidad aumentando la cantidad de hilos es muy difícil, y cuantos más hilos se utilicen, menores serán las ganancias. Sin embargo, todavía existen varias formas de hacerlo, la mayoría bastante complicadas y que escalan muy mal conforme aumentas los hilos. El algoritmo tradicionalmente utilizado solía ser el conocido como Young Brothers Wait Concept (Feldmann, 1989), un algoritmo bastante complejo que, por ejemplo, fue usado por Stockfish hasta hace algunos años. No obstante, con la creciente cantidad de núcleos disponibles en los ordenadores modernos, la escalabilidad fue muy pobre y el código demasiado complejo.

9. Descripción de la solución propuesta

Para hacer la descripción de la solución propuesta, se subdividirá en varios aparatados. Se empezará hablando sobre cada una de las pantallas que se han implementado, partiendo del mapa del sitio y de los *wireframes*, comentando sus aspectos finales más relevantes. Además, habrá un apartado dedicado al funcionamiento y estructura de la inteligencia artificial que se ha desarrollado.

Para poder probar la aplicación final, he adjuntado un enlace a una carpeta compartida de Google Drive en la que se encuentra el ejecutable y todos los archivos necesarios para su funcionamiento. El enlace es el siguiente: https://drive.google.com/drive/folders/1aP4Lllp8D-qImKcmI6b6IFNkZMKKR2ux?usp=sharing

9.1. Menú principal

Esta pantalla ha resultado ser la menos elaborada debido a haber concentrado el tiempo dedicado en mejorar la inteligencia artificial y ciertos aspectos de la jugabilidad. Su interfaz cuanta con unos simples botones que permiten jugar o salir de la aplicación, y un fondo procedente de la página Unplash (Unpht).

Ilustración 21: Menú principal

9.2. Configuraciones previas

El usuario, a través del menú principal y pulsando en "*Jugar*", accederá a esta pantalla de configuraciones previas. A primera vista, lo que más destaca es la previsualización de la posición del tablero, la cual, como anteriormente se ha explicado, se podrá cambiar a gusto del usuario.

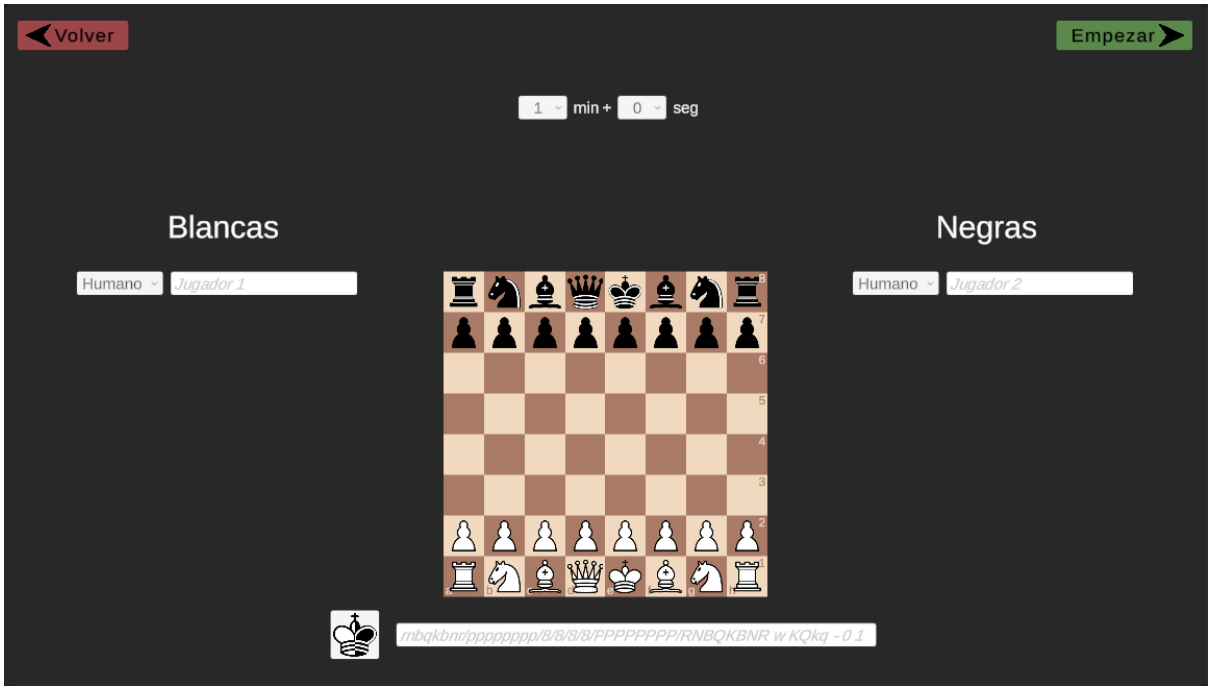


Ilustración 22: Configuraciones previas

Nada más abrir esta pantalla, la posición inicial que aparece en el tablero es el inicio de partida del ajedrez, identificado por su cadena en formato FEN: "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1", como se puede ver en el *placeholder* del formulario debajo del tablero. Si se introduce una cadena de texto que no se corresponde con ninguna posición del tablero en formato FEN, se desactivará la interactuabilidad con el botón "*Empezar*", se vaciará el tablero de piezas y aparecerá un mensaje informando de ello, como se puede ver en la siguiente imagen:

Ilustración 23: Configuraciones previas (posición FEN incorrecta)

Si, en el formulario, se introduce una posición FEN válida, diferente a la posición inicial del ajedrez, el aspecto de la interfaz será el siguiente:

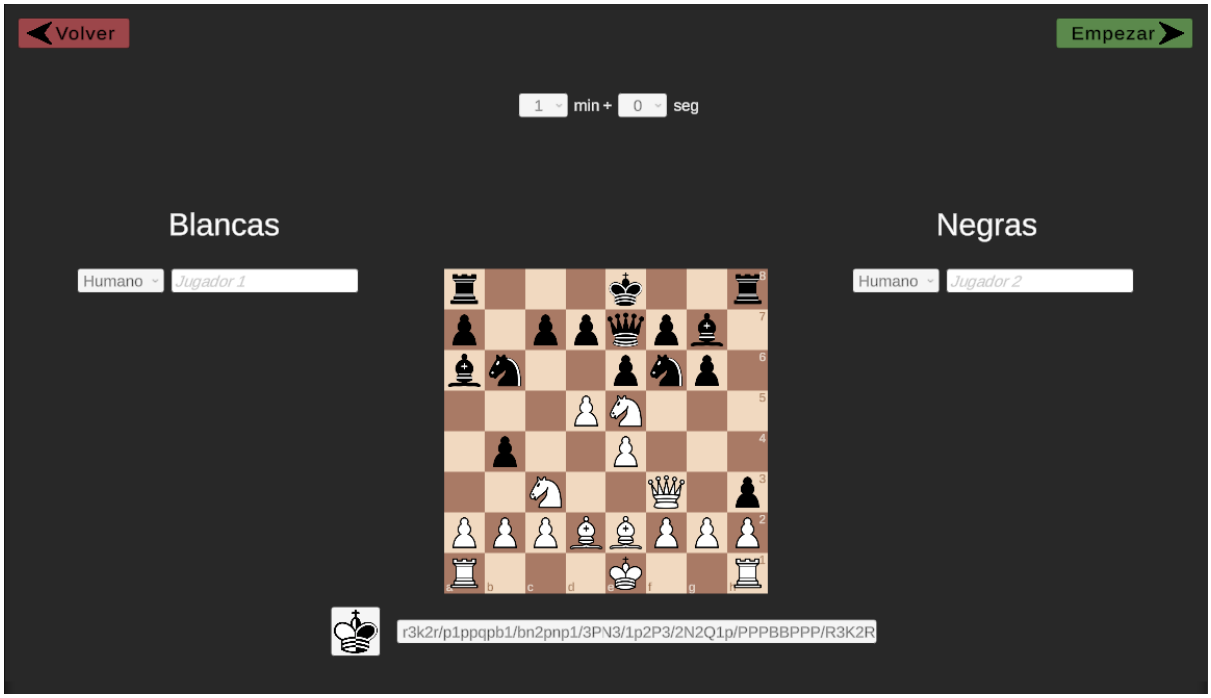


Ilustración 24: Configuraciones previas (posición FEN: "r3k2r/p1ppqpb1/bn2pnp1/3PN3/1p2P3/2N2Q1p/PPPBBPPP/R3K2R w KQkq – 0 1")

A la izquierda del formulario para introducir la posición del tablero, se encuentra el botón para intercambiar el color de los jugadores del tablero. Al pulsar dicho botón, la previsualización del tablero también cambiará, y siempre mantendrá una perspectiva en la que las piezas del jugador 1 se muestren abajo, y las del jugador 2 arriba (al menos en su distribución de la situación inicial de partida):

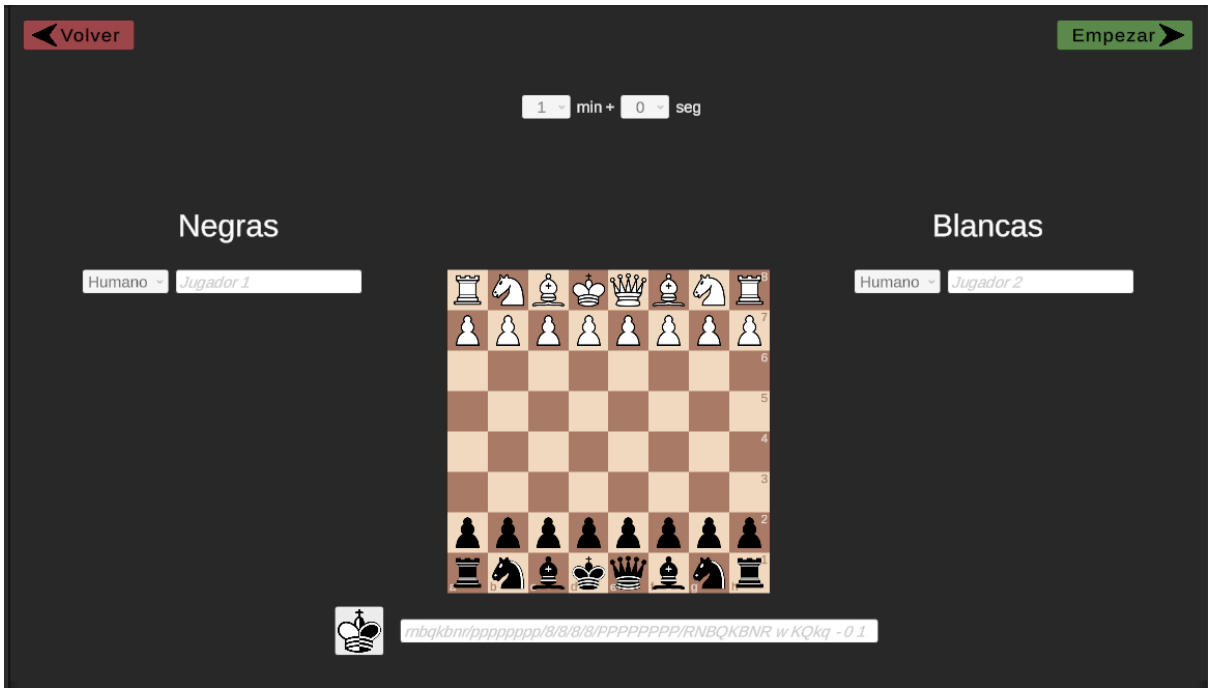


Ilustración 25: Configuraciones previas (colores de los jugadores intercambiados)

En los márgenes izquierdo y derecho, nos encontramos con toda la configuración relativa a los jugadores. Para empezar, existe un selector para cada jugador, en el que se puede especificar el tipo de jugador de cada bando, sea humano o inteligencia artificial:

Ilustración 26: Selector del tipo de jugador

Cambiar el tipo de jugador, altera las opciones que se le muestran al usuario:

- **Humano**: A la derecha del selector, se mostrará un formulario que permite cambiar el nombre que tendrá dicho jugador dentro de la partida, siendo de manera predeterminada jugador 1 o 2, en función de cual sea. Esto se ve indicado en el *placeholder* de ese formulario.

Ilustración 27: Formulario para el nombre del jugador 1

- **Inteligencia artificial**: De nuevo, a la derecha del selector del tipo de jugador, aparecerá otro selector para elegir la dificultad que tendrá la inteligencia artificial durante la partida, pudiendo variar entre: fácil, medio o difícil.

Ilustración 28: Selector de dificultad de la IA

Por último, respecto a esta pantalla de configuraciones previas, se encuentran las opciones relativas al tiempo. Los parámetros de tiempo configurables para la partida son la duración de la misma y el incremento de tiempo, pudiendo establecerse mediante dos selectores, uno para cada parámetro:

Ilustración 29: Selector de duración de partida

Ilustración 30: Selector de incremento de partida

Finalmente, es las esquinas superior izquierda y superior derecha, tenemos los botones de "*Volver*" y "*Empezar*", respectivamente. Permitiendo, tanto volver al usuario a la pantalla del menú principal, como avanzar a la pantalla de partida:

Ilustración 31: Botón de "Volver"

Ilustración 32: Botón de "Empezar"

9.3. Partida

Una vez que el usuario haya establecido unos parámetros válidos en la configuración previa, y haya pulsado el botón de "*Empezar*", aparecerá la ventana de partida, y comenzará el juego en base dichas configuraciones previas. En el caso de la siguiente ilustración, las configuraciones han sido las predeterminadas (jugador 1 – blancas, jugador 2 – negras, duración – 1 minuto, incremento – 0 segundos):

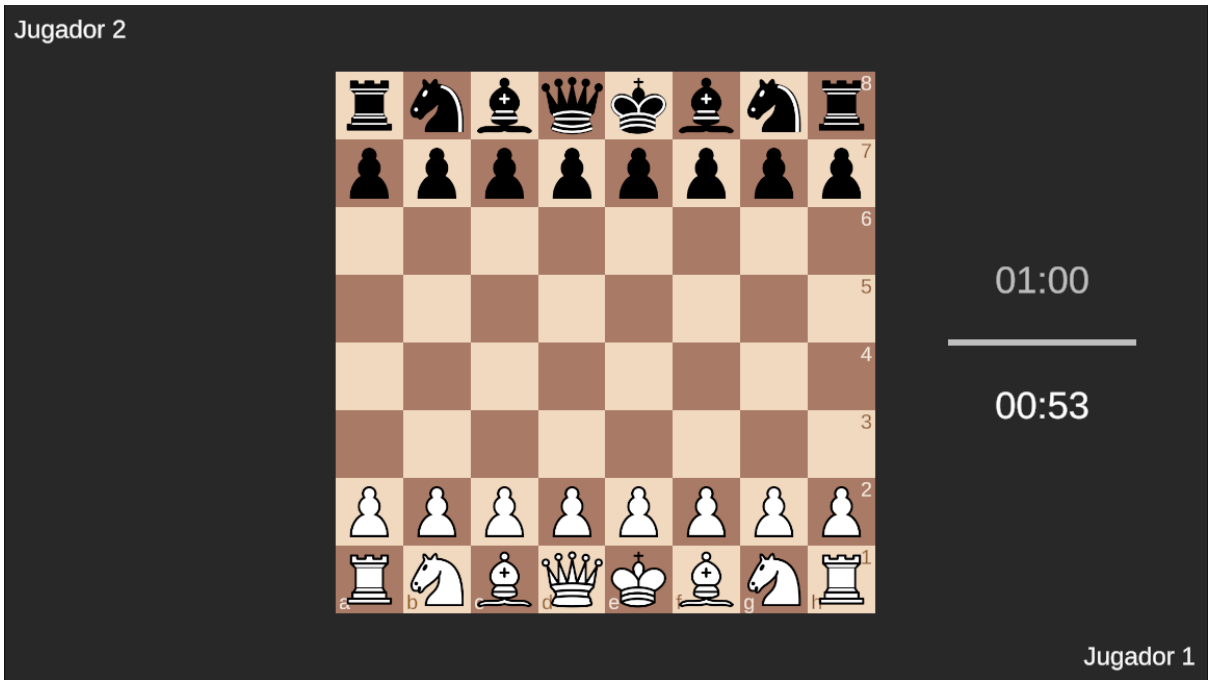


Ilustración 33: Partida con configuraciones predeterminadas

Para la creación del estilo visual del tablero, ha servido de inspiración el diseño del tablero y piezas (SVG chess pieces) de la página lichess.org (lichess.org). Como aspectos destacables a primera vista, se encuentra el tiempo restante de cada jugador a la derecha (el del jugador 1 abajo y el del jugador 2 arriba), siendo separado por una línea divisoria blanca. Los nombres de los jugadores 1 y 2, también aparecen en la esquina inferior derecha y en la esquina superior izquierda, respectivamente, pero al tratarse de una partida con configuraciones predeterminadas, no aparece ningún nombre característico.

Las piezas, son seleccionadas manteniendo pulsado el *click* izquierdo del ratón sobre ellas, y si mientras se hace eso, se mueve el cursor por la pantalla, dicha pieza lo seguirá. Además, se mostrarán los movimientos legales para esa determinada pieza, en forma de casillas azules hacia las que se puede desplazar, y la casilla desde la que se hace el movimiento de color amarillo:

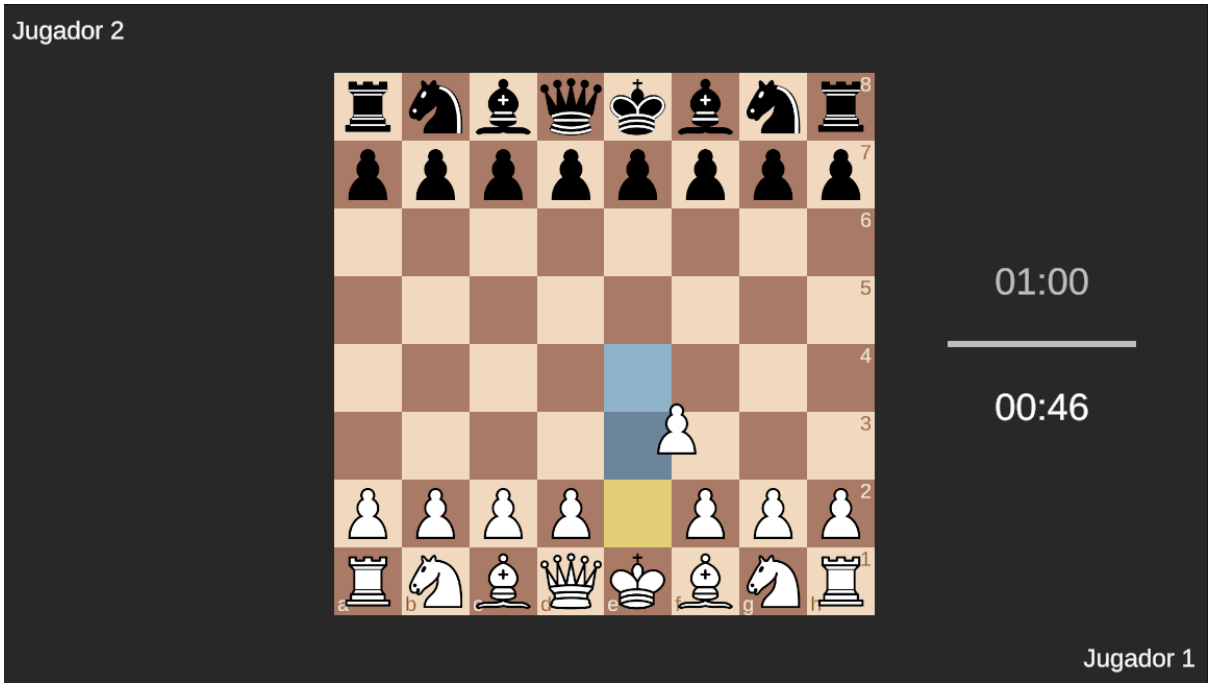


Ilustración 34: Partida (selección de una pieza)

Tras haber realizado un movimiento, se señalarán en el tablero las casillas en las que a acontecido el último movimiento:

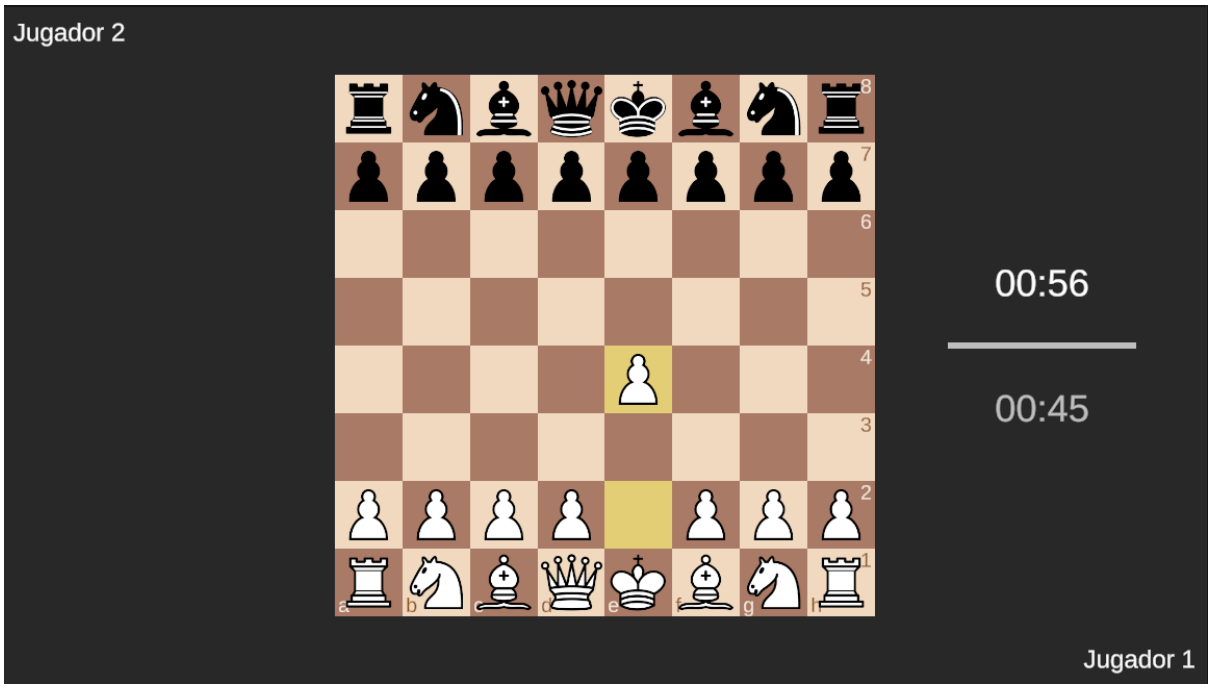


Ilustración 35: Partida (pieza movida)

Cuando sucede una situación de jaque, esto se indica mediante un texto que aparece en el lugar de la línea divisoria y se colorear de rojo la casilla en la que se encuentra el rey en jaque:

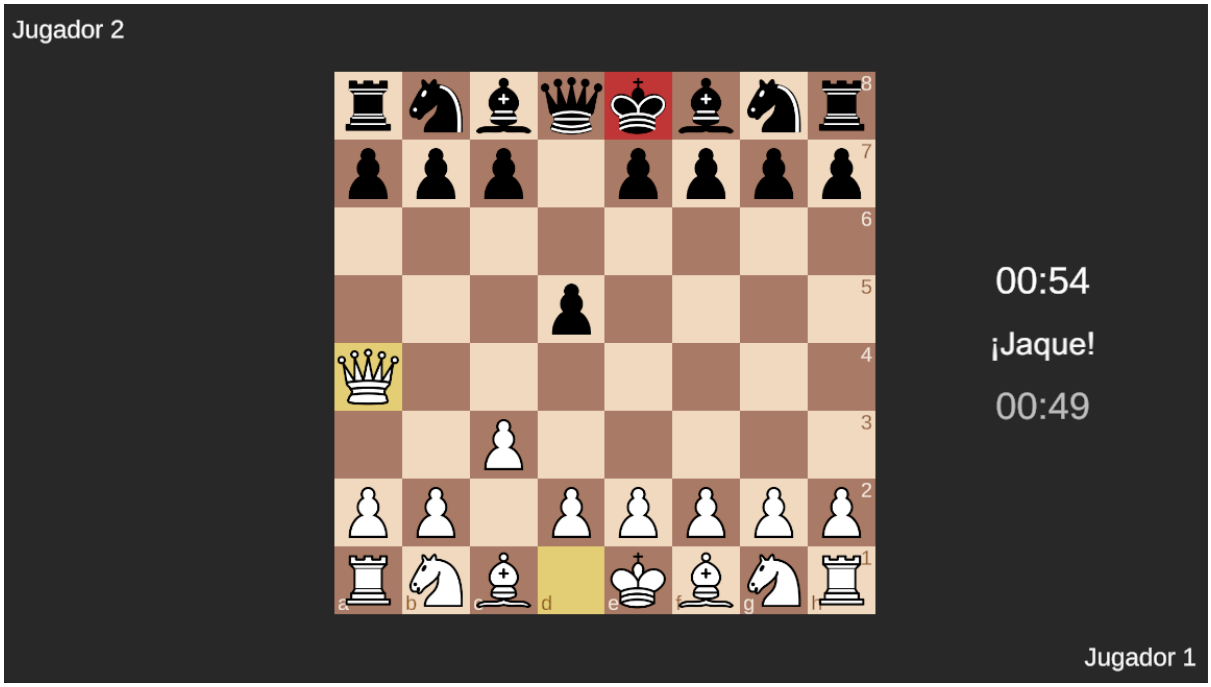


Ilustración 36: Partida (situación de jaque)

Otras situaciones de partida en las que aparece texto en lugar de la línea divisoria, es en el momento en el que ocurre un jaque mate, se agota el tiempo de alguno de los jugadores o hay tablas en la partida:

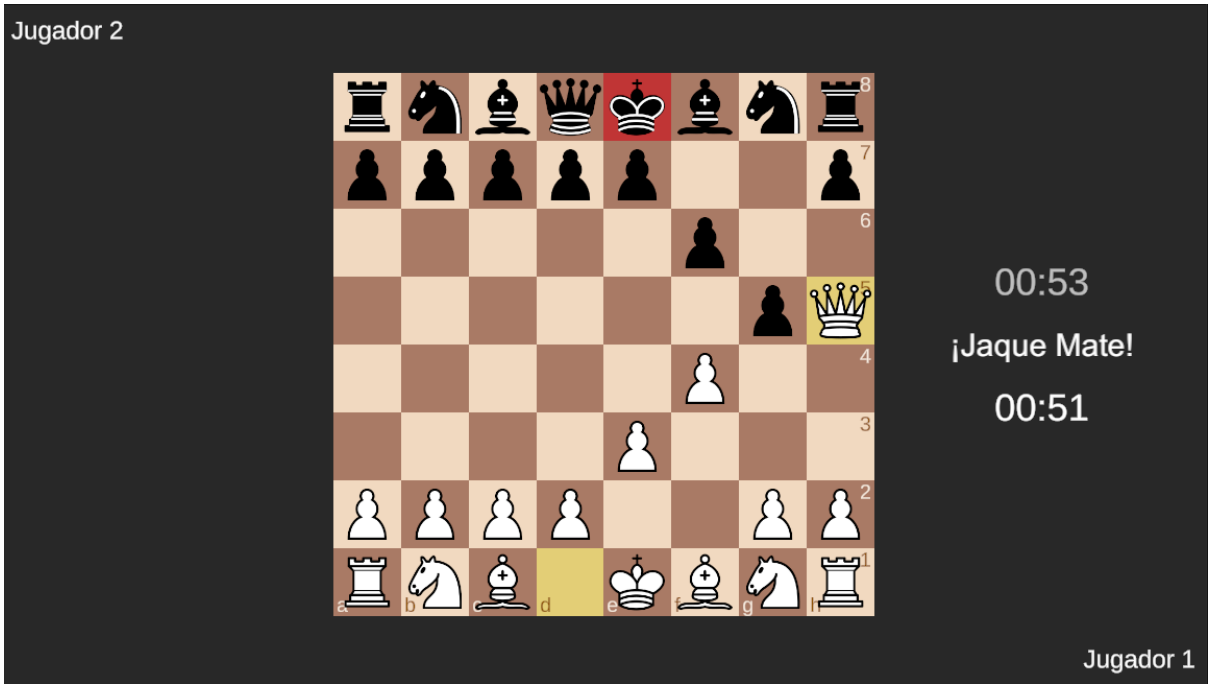


Ilustración 37: Partida (situación de jaque mate)

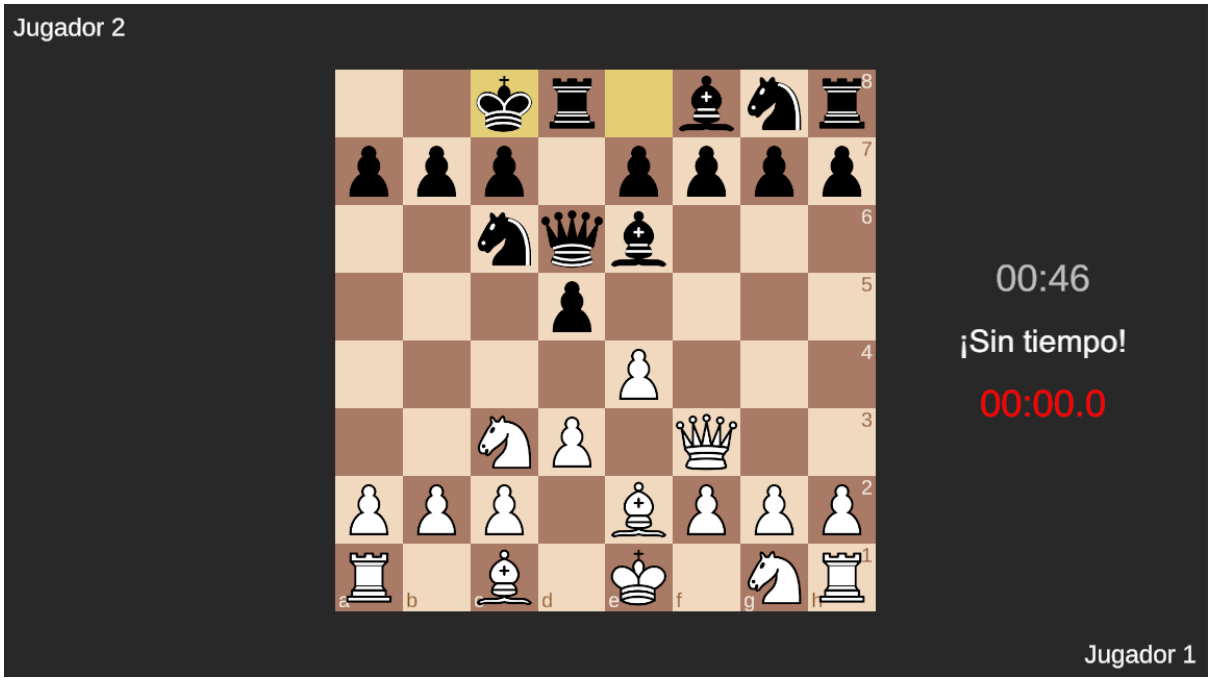
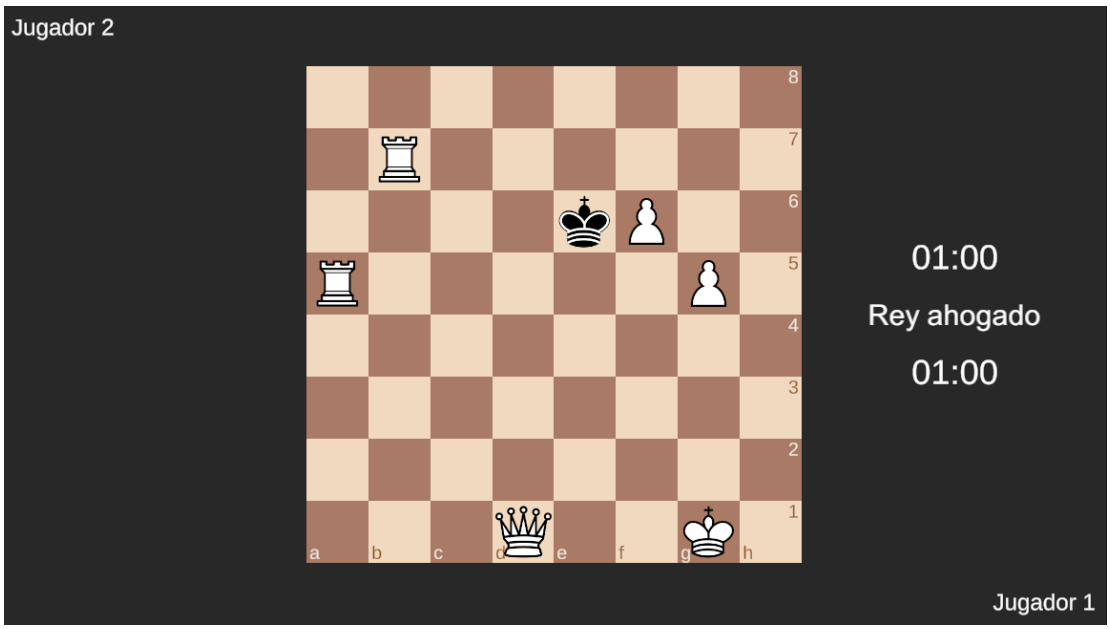


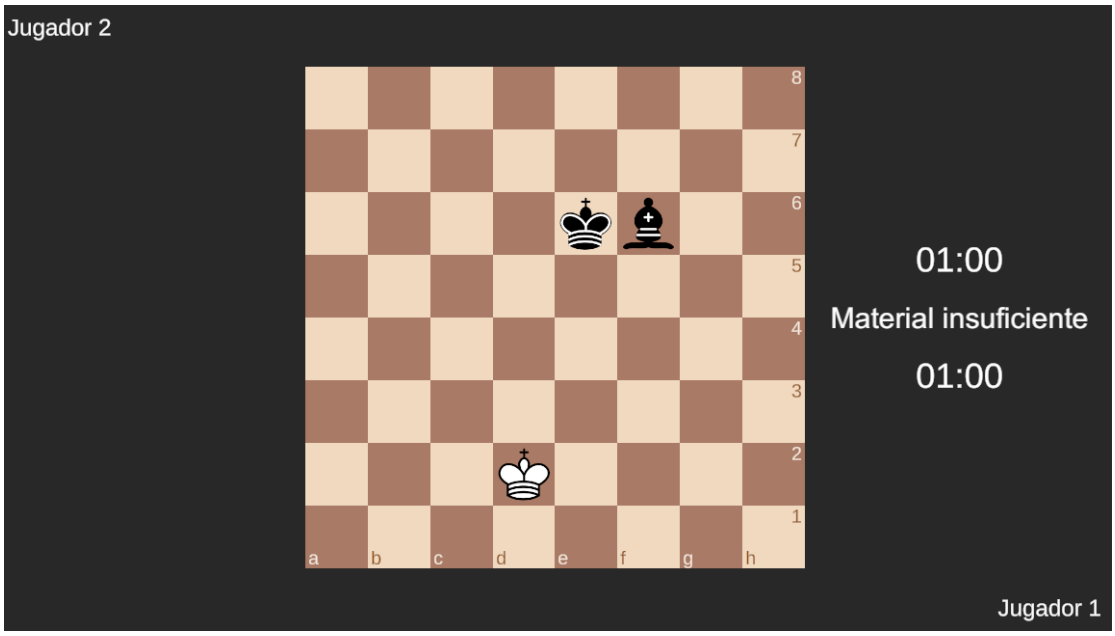
Ilustración 38: Partida (situación de tiempo agotado)

En concreto con las tablas, hay varias situaciones de juego que pueden ser consideradas como tablas o empate:

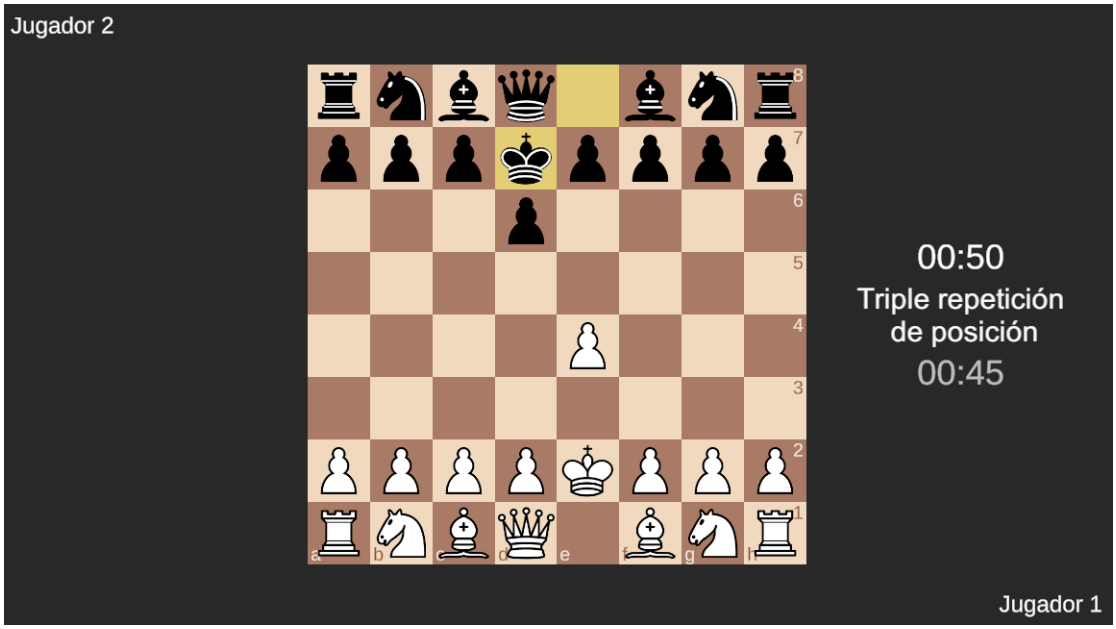
- **Rey ahogado**: Ocurre cuando un jugador no está en jaque, pero no tiene ningún movimiento legal disponible.



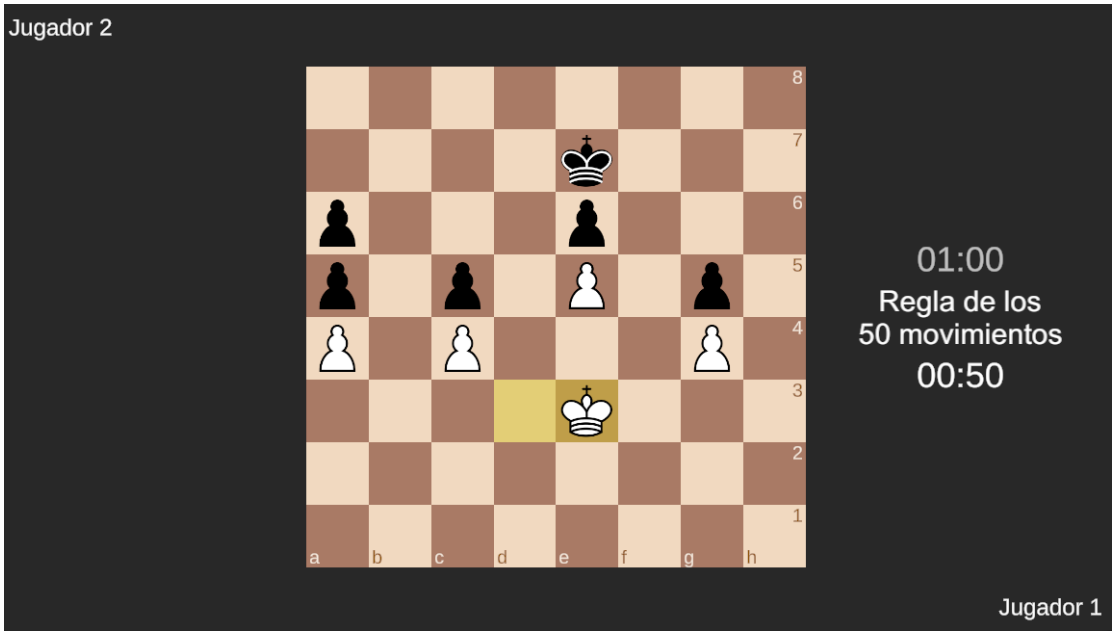
- **Insuficiencia de material**: Se produce cuando ninguno de los dos jugadores tiene suficiente material para dar jaque mate al adversario, sin importar cómo jueguen. Los ejemplos más comunes son: rey contra rey, rey y alfil contra rey, o rey y caballo contra rey.



- **Triple repetición de posición**: Ocurre si se repite una posición al menos por tres veces en la partida.



- **Regla de los 50 movimientos**: Si durante 50 movimientos consecutivos no se ha movido ningún peón ni se ha capturado ninguna pieza, se declara la partida como tablas.



9.3.1. Promoción

En el momento en el que un jugador mueve un peón de su bando hacia la última fila del lado contrario, ocurre una promoción. Esta se ve reflejada en el programa con la aparición de una pequeña ventana que permite seleccionar entre la pieza en la que se quiere promocionar el peón:

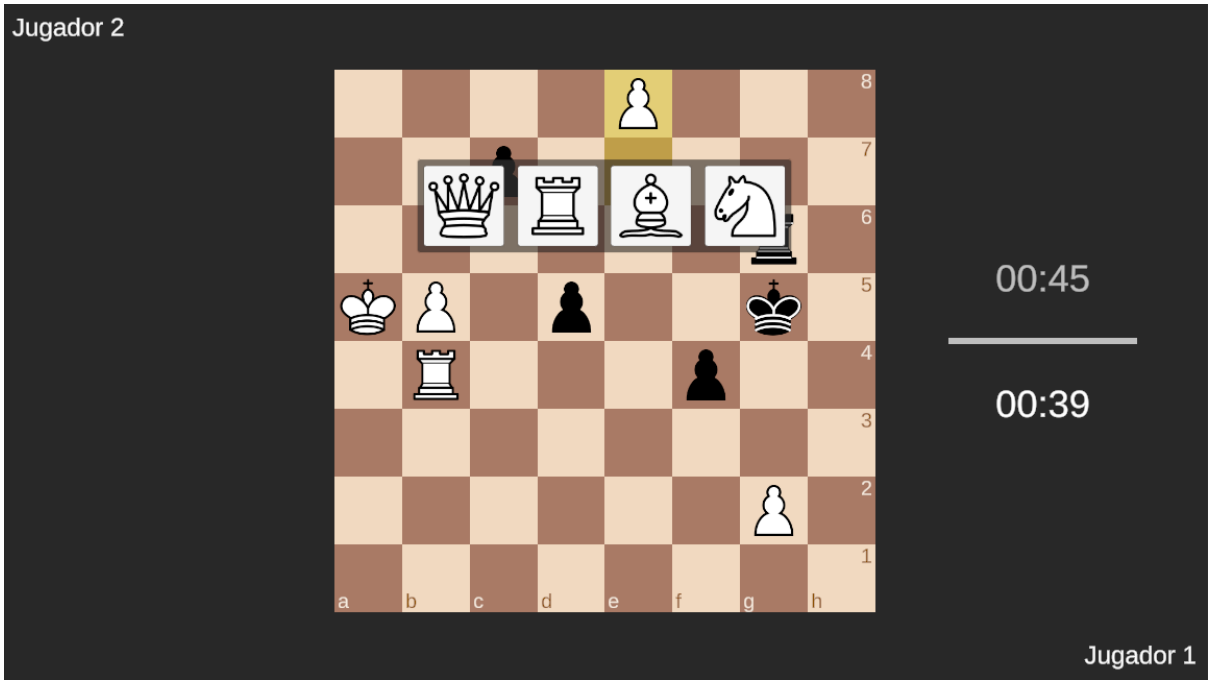


Ilustración 39: Panel de promoción de las piezas blancas

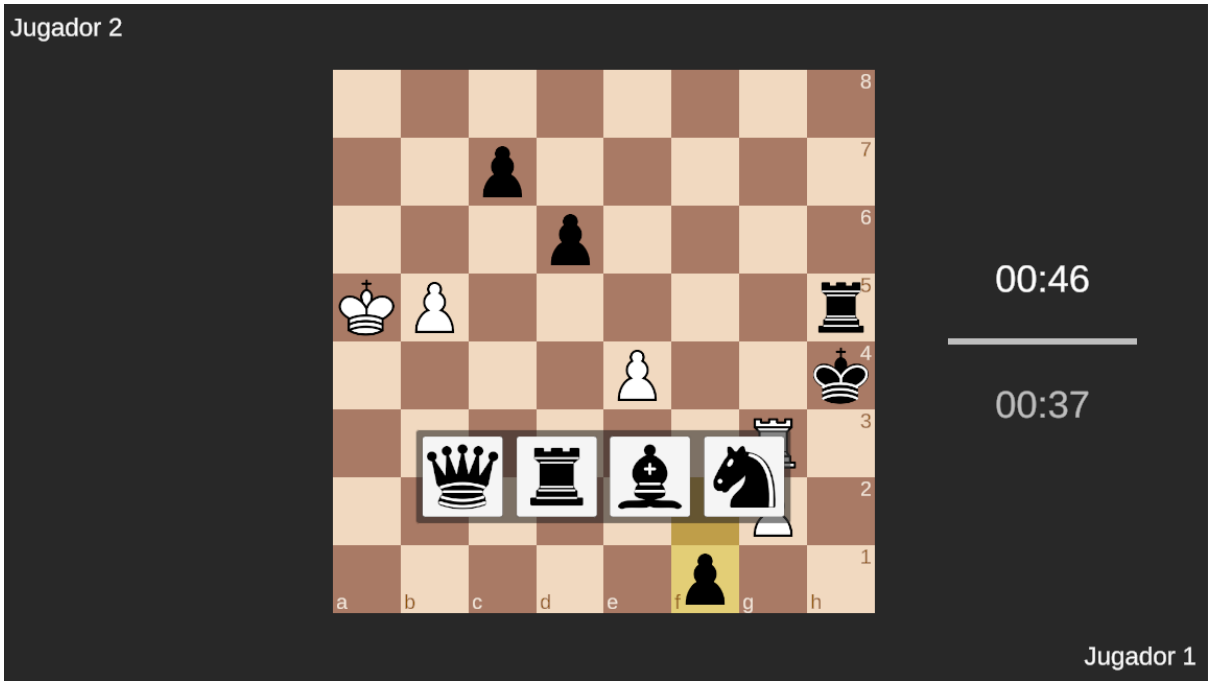


Ilustración 40: Panel de promoción de las piezas negras

9.3.2. Final de partida

Por último, en cuanto a la partida, se encuentra la ventana que indica que el juego ha finalizado, además de mostrar el tipo de resultado (victoria o tablas) y el jugador ganador:

Ilustración 41: Panel de final de partida

9.4.Inteligencia artificial

La inteligencia artificial implementada en esta aplicación se basa en una combinación eficiente de técnicas clásicas de búsqueda y evaluación, cuya elección ha sido expuesta con anterioridad en esta memoria.

9.4.1. Bitboards

La aplicación utiliza *bitboards* como estructura principal para representar el estado del tablero, lo cual mejora significativamente la eficiencia en:

- Generación de movimientos legales.
- Cálculo de ataques y defensas.
- Aplicación de máscaras para control de columnas, filas, diagonales, etc.

Este tipo de representación no solo ha ayudado a incrementar la rapidez de la búsqueda de la inteligencia artificial, si no que permite efectuar ciertas evaluaciones forma casi instantánea (con operaciones a nivel de bit), además de poder aprovechar la información que otorgan los *bitboards* para muchos otros tipos de evaluaciones que no se han implementado en este proyecto.

9.4.2. Minimax con poda alfa-beta

Para la toma de decisiones, la inteligencia artificial emplea el algoritmo Minimax con poda alfa-beta, además de emplear una técnica de **ordenación de movimientos**, permitiendo incrementar la efectividad de la poda alfa-beta, ya que los mejores movimientos se evalúan primero, maximizando las oportunidades de cortar ramas.

Esta técnica de ordenación se basa en otorgar una puntuación para cada movimiento generado para un nodo del árbol de búsqueda, en función de lo bueno que sea. Aunque no podemos saber la puntuación exacta del movimiento, previa a la evaluación, sí que podemos suponer o estimar dicha puntuación. Los aspectos que se han tenido en cuenta para puntuar los movimientos en la ordenación han sido los siguientes:

- Priorizar la captura de piezas más valiosas que aquella que realiza el movimiento.
- Priorizar los movimientos que sean promociones de peones.
- Penalizar el mover hacia una casilla atacada por un peón rival.

La ordenación de movimientos por sí sola, no tiene un gran papel en la búsqueda, pero en combinación con la poda alfa-beta, supone un decremento considerable del tiempo invertido el buscar movimientos.

El segundo aspecto muy importante en la búsqueda son las denominadas **tablas de transposición**. El hecho es que, durante la búsqueda, nos toparemos en multitud de ocasiones con situaciones de tablero que ya se han evaluado con anterioridad en esa, u otra, búsqueda. A esto se le llama transposición, es decir, una posición del tablero de ajedrez que puede ser alcanzada por secuencias diferentes de movimientos. Para evitar volver a evaluar posiciones

que ya se han evaluado en el pasado y poder ahorrar tiempo de búsqueda, es necesario mantener un seguimiento de las posiciones evaluadas, a estos es a lo que nos referimos con tabla de transposición.

El funcionamiento de la tabla se basa esencialmente en guardar un identificador de cada posición y un valor de su correspondiente evaluación. Como identificador, se podría emplear las cadenas de texto en formato FEN, sin embargo, estas son relativamente lentas de generar en comparación con otros métodos. En su lugar, la alternativa elegida se conoce como **Zobrist *hashing***, una función *hash* especializada en la generación rápida de resúmenes para situaciones de juegos de tablero. Sin embargo, este método no es perfecto, y es que las colisiones de claves o errores de tipo 1 son inherentes al uso de claves Zobrist, con muchos menos bits de los necesarios para codificar todas las posiciones posibles alcanzables en el ajedrez. Es un riesgo que asumir si queremos la suficiente rapidez en las tablas de transposiciones, como para que resulten atractivas. Además, la probabilidad de que ocurra una colisión es lo suficientemente baja como para que no repercuta prácticamente nunca en el funcionamiento de la inteligencia artificial. En concreto, con un *hash* de 64 bits, se puede esperar una colisión después de aproximadamente 2^{32}, es decir, 4 mil millones de posiciones (Zobrist, 1970). Estos *hashes* se generar a partir de los *bitboards* del tablero, y mediante operaciones de OR exclusivo, lo que permite hacer una actualización incremental de los mismos, en vez de tener que calcularlos desde cero para cada posición.

9.4.3. Evaluación

La evaluación de cada posición se realiza mediante una función heurística compuesta por los siguientes módulos:

1. **Valor del material**: Asigna puntuaciones fijas a cada pieza (peones, caballos, alfiles, torres y dama), considerando la diferencia entre el material propio y el del oponente como una base numérica de ventaja. Para este caso, los valores del material empleados en la aplicación han sido los siguientes:

Tipo de pieza	Valor
Peón	100
Caballo	300
Alfil	300
Torre	500
Reina	900

Tabla 9: Valor del material

2. ***Piece-Square Tables* (PST)**: Se aplican tablas de bonificación o penalización a cada pieza, según se ubicación en el tablero y el tipo de pieza que sea. Además, algunas piezas, como los peones o los reyes, cuentan con tablas de bonificación que varían en función de la fase de la partida (temprana o tardía).

En concreto, se han utilizado las siguientes tablas de bonificación para las piezas:

0	0	0	0	0	0	0	0
50	50	50	50	50	50	50	50
10	10	20	30	30	20	10	10
5	5	10	25	25	10	5	5
0	0	0	20	20	0	0	0
5	-5	-10	0	0	-10	-5	5
5	10	10	-20	-20	10	10	5
0	0	0	0	0	0	0	0

Ilustración 42: Piece-square table peón temprano

0	0	0	0	0	0	0	0
80	80	80	80	80	80	80	80
50	50	50	50	50	50	50	50
30	30	30	30	30	30	30	30
20	20	20	20	20	20	20	20
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
0	0	0	0	0	0	0	0

Ilustración 43: Piece-square table peón tardío

-50	-40	-30	-30	-30	-30	-40	-50
-40	-20	0	0	0	0	-20	-40
-30	0	10	15	15	10	0	-30
-30	5	15	20	20	15	5	-30
-30	0	15	20	20	15	0	-30
-30	5	10	15	15	10	5	-30
-40	-20	0	5	5	0	-20	-40
-50	-40	-30	-30	-30	-30	-40	-50

Ilustración 44: Piece-square table caballo

-20	-10	-10	-10	-10	-10	-10	-20
-10	0	0	0	0	0	0	-10
-10	0	5	10	10	5	0	-10
-10	5	5	10	10	5	5	-10
-10	0	10	10	10	10	0	-10
-10	10	10	10	10	10	10	-10
-10	5	0	0	0	0	5	-10
-20	-10	-10	-10	-10	-10	-10	-20

Ilustración 45: Piece-square table alfil

0	0	0	0	0	0	0	0
5	10	10	10	10	10	10	5
-5	0	0	0	0	0	0	-5
-5	0	0	0	0	0	0	-5
-5	0	0	0	0	0	0	-5
-5	0	0	0	0	0	0	-5
-5	0	0	0	0	0	0	-5
0	0	0	5	5	0	0	0

Ilustración 46: Piece-square table torre

-20	-10	-10	-5	-5	-10	-10	-20
-10	0	0	0	0	0	0	-10
-10	0	5	5	5	5	0	-10
-5	0	5	5	5	5	0	-5
-5	0	5	5	5	5	0	0
-10	0	5	5	5	5	5	-10
-10	0	0	0	0	5	0	-10
-20	-10	-10	-5	-5	-10	-10	-20

Ilustración 47: Piece-square table reina

-80	-70	-70	-70	-70	-70	-70	-80
-60	-60	-60	-60	-60	-60	-60	-60
-40	-50	-50	-60	-60	-50	-50	-40
-30	-40	-40	-50	-50	-40	-40	-30
-20	-30	-30	-40	-40	-30	-30	-20
-10	-20	-20	-20	-20	-20	-20	-10
20	20	-5	-5	-5	-5	20	20
20	30	10	0	0	10	30	20

Ilustración 48: Piece-square table rey temprano

-20	-10	-10	-10	-10	-10	-10	-20
-5	0	5	5	5	5	0	-5
-10	-5	20	30	30	20	-5	-10
-15	-10	35	45	45	35	-10	-15
-20	-15	30	40	40	30	-15	-20
-25	-20	20	25	25	20	-20	-25
-30	-25	0	0	0	0	-25	-30
-50	-30	-30	-30	-30	-30	-30	-50

Ilustración 49: Piece-square table rey tardío

3. **Evaluación *mop-up***: Esta evaluación adicional se activa en función de la fase de la partida, especialmente en el final de esta, y fomenta el control del centro del tablero, por parte del rey aliado, y el acorralamiento del rey enemigo.

10. Análisis de riesgos

Este apartado tiene como objetivo identificar y evaluar los posibles riesgos asociados al desarrollo del motor de ajedrez, centrándose especialmente en aquellos de naturaleza técnica y computacional, dada la orientación del proyecto. Se ha utilizado una **tabla de riesgos** como herramienta principal para facilitar la identificación de los riesgos y estructurar el análisis.

Nº	Descripción del riesgo	Prioridad	Complejidad	Medidas correctoras/preventivas
R1	Evaluaciones erróneas almacenadas en las tablas de transposición	Alta	Media	Refinar el código encargado de almacenar y recuperar información de las tablas de transposición
R2	Eficiencia no sufiente en la generación de movimientos	Media	Alta	Optimización de la generación de movimientos mediante los denominados "magic bitboards"
R3	Ordenación de los movimientos ineficiente	Baja	Baja	Emplear algoritmos de ordenación más veloces como "quick sort"
R4	Debilidad de la IA en etapas de juego tardías	Alta	Media	Mejora del módulo de evaluación y/o de la tabla de transposición
R5	Creación de basura excesiva al instanciar clases	Media	Alta	Emplear ciertos métodos de C# para manipular mejor la memoria, como "Span<T>"

Tabla 10: Tabla de riesgos

11. Organización y gestión del proyecto

11.1. Organización

Como se ha explicado antes, el desarrollo del proyecto se ha llevado a cabo siguiendo el modelo del Proceso Unificado, una metodología iterativa e incremental estructurada en cuatro fases principales: Inicio, Elaboración, Construcción y Transición. A continuación, se explicarán las tareas abordadas durante cada una de las 4 fases del Proceso Unificado, explicando a su vez cada una de sus iteraciones:

1. **Inicio**

    a. **Iteración 1**: Durante esta primera fase, se definió el alcance del proyecto y se identificaron los objetivos que están detallados en el anexo I. Todo esto acompañado de reuniones con los tutores para obtener la idea general de cómo debía ser el sistema a desarrollar. Relativo al modelo de negocio, se realizó una investigación sobre el funcionamiento de los programas de ajedrez, estudiando algunas implementaciones en concreto. Se descargaron y prepararon las herramientas para el desarrollo del proyecto, en especial, Unity.

2. **Elaboración**

    a. **Iteración 1**: Se realizó una comparación de los algoritmos de búsqueda investigados en la fase anterior, para poder optar por el mejor. También se hizo lo mismo para las formas de representación del tablero. Se identificaron los actores del sistema, los requisitos de información, y los paquetes funcionales en los que se agruparon los casos de uso, todo esto, de nuevo, se puede encontrar en el anexo I.

    b. **Iteración 2**: Se identificaron y documentaron los casos de uso, de forma detallada, correspondientes a los paquetes funcionales de Gestión de configuraciones previas, Gestión de partida y Gestión de inteligencia artificial. Se implementó un primer prototipo muy básico, elaborado el Python y sin ningún tipo de interfaz gráfica.

    c. **Iteración 3**: Esta iteración estuvo destinada a la identificación de todas las clases de control y las clases interfaz del sistema, propias del anexo II. Además, se empezarán a identificar los primeros subsistemas.

3. **Construcción**

    a. **Iteración 1**: Se llevó a cabo la realización de casos de uso a nivel de análisis y se identificaron las clases de diseño que formarán parte de los subsistemas. Se empezará trabajando en la interfaz de la aplicación que se corresponde con la partida (la funcionalidad de las piezas y el tablero). A su vez, se irán haciendo pruebas respecto a la función *perft*, con el objetivo de pulir la generación de movimientos al máximo y evitar que haya errores para cuando se empiece a implementar la inteligencia artificial.

b. **Iteración 2**: Se elaboraron los casos de uso a nivel de diseño y se procedió a la creación de los esquemas propios del diseño de interfaces, es decir, el mapa del sitio y los *wireframes*. Con esto se habría finalizado el anexo II. En cuanto a la implementación, en esta iteración, el foco principal fue la inteligencia artificial, desarrollando los componentes de búsqueda y evaluación.

c. **Iteración 3**: Por último, se implementaron el resto de las pantallas que se habían definido antes en los *wireframes*, y se añadieron las últimas mejoras a la inteligencia artificial.

4. **Transición**

a. **Iteración 1**: Se realizaron pruebas de eficiencia en cuanto a la generación de movimientos y se comprobó la fiabilidad y certeza de los movimientos de la inteligencia artificial. Además, se realizó una serie de enfrentamientos entre versiones previas de esta misma inteligencia artificial e incluso con otros motores como Stockfish.

11.2. Gestión del Proyecto

Durante la elaboración de este proyecto, el estudiante ha sido el único desarrollador y creador de los productos resultantes, si bien es que ha sido aconsejado por lo tutores y demás personas del entorno de la universidad. En cuanto a los recursos materiales empleados, todo han sido materiales personales.

Para complementar la organización del proyecto, se ha utilizado la herramienta Microsoft Project con el objetivo de definir y visualizar la planificación temporal del proyecto. A través de un diagrama de Gantt, se ha representado la distribución de tareas en el tiempo, estableciendo fases claras (Inicio, Elaboración, Construcción y Transición), así como hitos, dependencias y fechas estimadas.

	Task Mode	Nombre de tarea	Duration	Start	Finish	Predecessors	Resource
1		**Inicio**	**7 days?**	**Thu 23/01/25**	**Fri 31/01/25**		
2		**Iteración 1**	**7 days?**	**Thu 23/01/25**	**Fri 31/01/25**		
3		**Modelado de negocio**	**3 days**	**Thu 23/01/25**	**Mon 27/01/25**		
4		Reunión con los tutores	1 day	Thu 23/01/25	Thu 23/01/25		Óscar
5		Investigación sobre el funcionamiento de los motores de ajedrez	2 days	Fri 24/01/25	Mon 27/01/25	4	Óscar
6		**Requisitos**	**1 day**	**Tue 28/01/25**	**Tue 28/01/25**		
7		Identificación de objetivos y alcance del proyecto	1 day	Tue 28/01/25	Tue 28/01/25	5	Óscar
8		**Análisis**	**1 day**	**Wed 29/01/25**	**Wed 29/01/25**		
9		Modelo de dominio muy preliminar	1 day	Wed 29/01/25	Wed 29/01/25	7	Óscar
10		**Diseño**	**1 day**	**Thu 30/01/25**	**Thu 30/01/25**		
11		Prototipo de distribución de subsistemas	1 day	Thu 30/01/25	Thu 30/01/25	9	Óscar
12		**Implementación**	**1 day?**	**Fri 31/01/25**	**Fri 31/01/25**		
13		Descarga y preparación de elementos de desarrollo	1 day?	Fri 31/01/25	Fri 31/01/25	11	Óscar
14		**Hito fin iteración 1**	**0 days**	**Fri 31/01/25**	**Fri 31/01/25**	**2**	
15		**Hito fin inicio**	**0 days**	**Fri 31/01/25**	**Fri 31/01/25**	**1**	

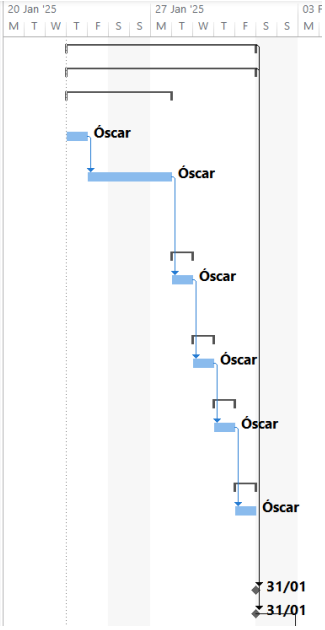


Ilustración 50: Diagrama de Gantt (1)

	Task Mode	Nombre de tarea	Duration	Start	Finish	Predecessors	Resource
16		**Elaboración**	**37 days**	**Mon 03/02/25**	**Tue 25/03/25**	**15**	
17		**Iteración 1**	**13 days**	**Mon 03/02/25**	**Wed 19/02/25**		
18		**Modelado de negocio**	**5 days**	**Mon 03/02/25**	**Fri 07/02/25**		
19		2ª reunión con los tutores	1 day	Mon 03/02/25	Mon 03/02/25		Óscar
20		Comparación de los algoritmos de búsqueda	2 days	Tue 04/02/25	Wed 05/02/25	19	Óscar
21		Comparación de las representaciones del tablero	2 days	Thu 06/02/25	Fri 07/02/25	20	Óscar
22		**Requisitos**	**3 days**	**Mon 10/02/25**	**Wed 12/02/25**		
23		Identificación de actores del sistema	1 day	Mon 10/02/25	Mon 10/02/25	21	Óscar
24		Identificación de requisitos de información	1 day	Tue 11/02/25	Tue 11/02/25	23	Óscar
25		Identificación de paquetes funcionales	1 day	Wed 12/02/25	Wed 12/02/25	24	Óscar
26		**Análisis**	**2 days**	**Thu 13/02/25**	**Fri 14/02/25**		
27		Clases entidad provisionales	2 days	Thu 13/02/25	Fri 14/02/25	25	Óscar
28		**Diseño**	**1 day**	**Mon 17/02/25**	**Mon 17/02/25**		
29		Boceto de interfaces	1 day	Mon 17/02/25	Mon 17/02/25	27	Óscar
30		**Implementación**	**1 day**	**Tue 18/02/25**	**Tue 18/02/25**		

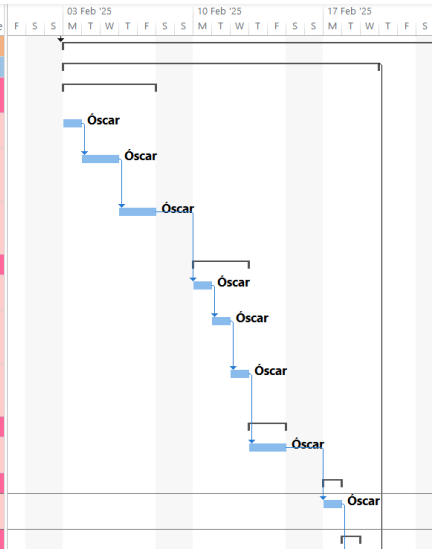


Ilustración 51: Diagrama de Gantt (2)

	Task Mode	Nombre de tarea	Duration	Start	Finish	Predecessors	Resource
31		Creación del proyecto en Unity	1 day	Tue 18/02/25	Tue 18/02/25	29	Óscar
32		**Pruebas**	**1 day**	**Wed 19/02/25**	**Wed 19/02/25**		
33		Se realizarán pruebas con otros motores de ajedrez	1 day	Wed 19/02/25	Wed 19/02/25	31	Óscar
34		**Hito fin iteración 1**	**0 days**	**Wed 19/02/25**	**Wed 19/02/25**	**17**	
35		**Iteración 2**	**13 days**	**Thu 20/02/25**	**Mon 10/03/25**	**34**	
36		**Modelado de negocio**	**1 day**	**Thu 20/02/25**	**Thu 20/02/25**		
37		Reunión con los tutores	1 day	Thu 20/02/25	Thu 20/02/25		Óscar
38		**Requisitos**	**7 days**	**Fri 21/02/25**	**Mon 03/03/25**		
39		CU Gestion de partida	3 days	Fri 21/02/25	Tue 25/02/25	37	Óscar
40		CU Gestion de inteligencia artificial	2 days	Wed 26/02/25	Thu 27/02/25	39	Óscar
41		CU Gestión de configuraciones previas	2 days	Fri 28/02/25	Mon 03/03/25	40	Óscar
42		**Análisis**	**1 day**	**Tue 04/03/25**	**Tue 04/03/25**		
43		Revisión de las clases entidad	1 day	Tue 04/03/25	Tue 04/03/25	41	Óscar
44		**Diseño**	**1 day**	**Wed 05/03/25**	**Wed 05/03/25**		
45		Revisión de los bocetos de las interfaces	1 day	Wed 05/03/25	Wed 05/03/25	43	Óscar

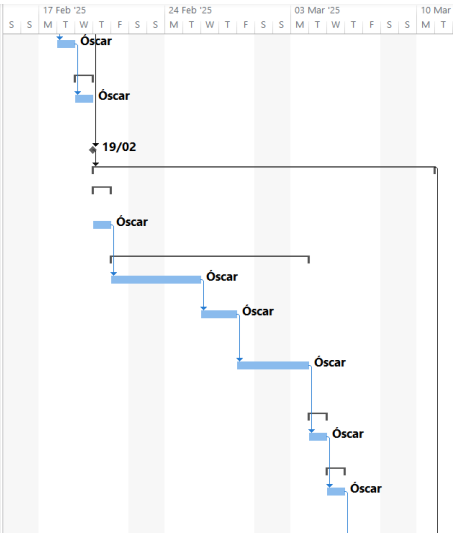


Ilustración 52: Diagrama de Gantt (3)

	Task Mode	Nombre de tarea	Duration	Start	Finish	Predecessors	Resource
46		**Implementación**	**2 days**	**Thu 06/03/25**	**Fri 07/03/25**		
47		Elaboración de un prototipo de Python	2 days	Thu 06/03/25	Fri 07/03/25	45	Óscar
48		**Pruebas**	**1 day**	**Mon 10/03/25**	**Mon 10/03/25**		
49		Definición de posibles escenarios de prueba para la aplicación	1 day	Mon 10/03/25	Mon 10/03/25	47	Óscar
50		**Hito fin iteración 2**	**0 days**	**Mon 10/03/25**	**Mon 10/03/25**	**35**	
51		**Iteración 3**	**11 days**	**Tue 11/03/25**	**Tue 25/03/25**	**50**	
52		**Modelado de negocio**	**2 days**	**Tue 11/03/25**	**Wed 12/03/25**		
53		Estimación del esfuerzo	2 days	Tue 11/03/25	Wed 12/03/25		Óscar
54		**Requisitos**	**2 days**	**Thu 13/03/25**	**Fri 14/03/25**		
55		Revisión de los casos de uso	1 day	Thu 13/03/25	Thu 13/03/25	53	Óscar
56		Identificación de los requisitos no funcionales	1 day	Fri 14/03/25	Fri 14/03/25	55	Óscar
57		**Análisis**	**2 days**	**Mon 17/03/25**	**Tue 18/03/25**		
58		Identificación del resto de clases a nivel de análisis	2 days	Mon 17/03/25	Tue 18/03/25	56	Óscar
59		**Diseño**	**2 days**	**Wed 19/03/25**	**Thu 20/03/25**		

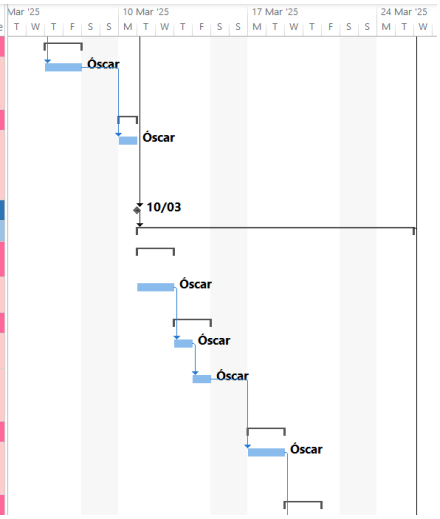


Ilustración 53: Diagrama de Gantt (4)

	Task Mode	Nombre de tarea	Duration	Start	Finish	Predecessors	Resource
60		Identificación preliminar de los subsistemas	2 days	Wed 19/03/25	Thu 20/03/25	58	Óscar
61		**Implementación**	**2 days**	**Fri 21/03/25**	**Mon 24/03/25**		
62		Implementación de aplicación sobre bitboards	2 days	Fri 21/03/25	Mon 24/03/25	60	Óscar
63		**Pruebas**	**1 day**	**Tue 25/03/25**	**Tue 25/03/25**		
64		Pruebas sobre el prototipo	1 day	Tue 25/03/25	Tue 25/03/25	62	Óscar
65		**Hito fin iteración 3**	**0 days**	**Tue 25/03/25**	**Tue 25/03/25**	**51**	
66		**Hito fin elaboración**	**0 days**	**Tue 25/03/25**	**Tue 25/03/25**	**16**	
67		**Construcción**	**44 days**	**Wed 26/03/25**	**Tue 24/06/25**	**66**	
68		**Iteración 1**	**19 days**	**Wed 26/03/25**	**Mon 05/05/25**		
69		**Modelado de negocio**	**1 day**	**Wed 26/03/25**	**Wed 26/03/25**		
70		3ª reunión con los tutores	1 day	Wed 26/03/25	Wed 26/03/25		Óscar
71		**Requisitos**	**2 days**	**Thu 27/03/25**	**Fri 28/03/25**		
72		Validación de los requisitos	2 days	Thu 27/03/25	Fri 28/03/25	70	Óscar
73		**Análisis**	**4 days**	**Mon 31/03/25**	**Thu 03/04/25**		
74		Realización de casos de uso a nivel de análisis	4 days	Mon 31/03/25	Thu 03/04/25	72	Óscar
75		**Diseño**	**2 days**	**Fri 04/04/25**	**Mon 07/04/25**		

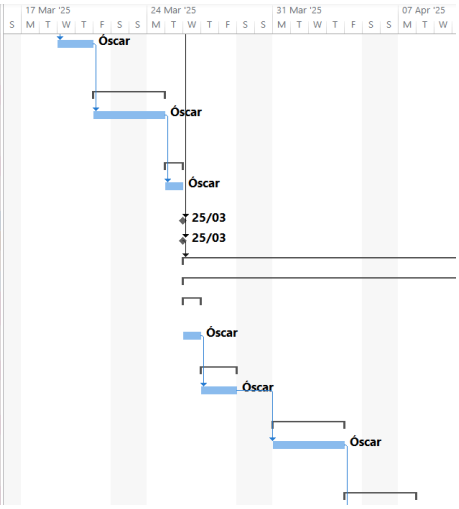


Ilustración 54: Diagrama de Gantt (5)

	Task Mode	Nombre de tarea	Duration	Start	Finish	Predecessors	Resource
76		Identificación de las clases de diseño	2 days	Fri 04/04/25	Mon 07/04/25	74	Óscar
77		**Implementación**	**5 days**	**Tue 08/04/25**	**Thu 24/04/25**		
78		Implementación de la funcionalidad de las piezas y el tablero	5 days	Tue 08/04/25	Thu 24/04/25	76	Óscar[50
79		**Pruebas**	**5 days**	**Fri 25/04/25**	**Mon 05/05/25**		
80		Depuración mediante perft	5 days	Fri 25/04/25	Mon 05/05/25	78	Óscar[50
81		**Hito fin iteración 1**	**0 days**	**Mon 05/05/25**	**Mon 05/05/25**	**68**	
82		**Iteración 2**	**18 days**	**Tue 06/05/25**	**Fri 13/06/25**	**81**	
83		**Modelado de negocio**	**1 day**	**Tue 06/05/25**	**Tue 06/05/25**		
84		4ª Reunión con los tutores	1 day	Tue 06/05/25	Tue 06/05/25		Óscar
85		**Requisitos**	**1 day**	**Wed 07/05/25**	**Wed 07/05/25**		
86		Validación de los requisitos	1 day	Wed 07/05/25	Wed 07/05/25	84	Óscar
87		**Análisis**	**1 day**	**Thu 08/05/25**	**Thu 08/05/25**		
88		Revisar el modelo de análisis	1 day	Thu 08/05/25	Thu 08/05/25	86	Óscar
89		**Diseño**	**5 days**	**Fri 09/05/25**	**Thu 15/05/25**		
90		Realización de los casos de uso a nivel de diseño	3 days	Fri 09/05/25	Tue 13/05/25	88	Óscar
91		Mapa del sitio	2 days	Wed 14/05/25	Thu 15/05/25	90	Óscar[50
92		Wireframes	2 days	Wed 14/05/25	Thu 15/05/25	91SS	Óscar[50

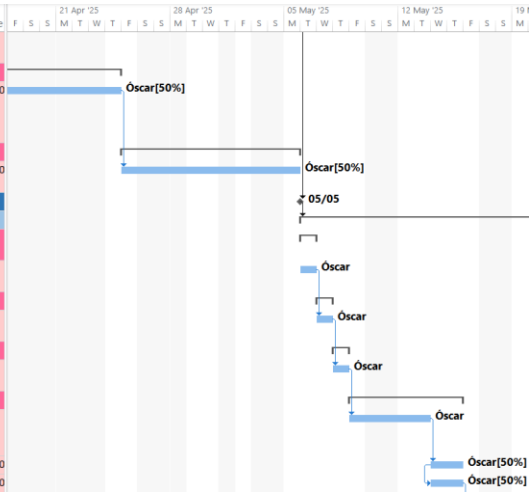


Ilustración 55: Diagrama de Gantt (6)

	Task Mode	Nombre de tarea	Duration	Start	Finish	Predecessors	Resource
93		**Implementación**	**5 days**	**Fri 16/05/25**	**Thu 22/05/25**		
94		Implementación del paquete funcional de Gestión de inteligencia artificial	5 days	Fri 16/05/25	Thu 22/05/25	92	Óscar
95		**Pruebas**	**5 days**	**Fri 23/05/25**	**Fri 13/06/25**		
96		Pruebas al la inteligencia artificial	5 days	Fri 23/05/25	Fri 13/06/25	94	Óscar
97		**Hito fin iteracion 2**	**0 days**	**Fri 13/06/25**	**Fri 13/06/25**	**82**	
98		**Iteración 3**	**7 days**	**Mon 16/06/25**	**Tue 24/06/25**	**97**	
99		**Modelado de negocio**	**1 day**	**Mon 16/06/25**	**Mon 16/06/25**		
100		5ª Reunión con los tutores	1 day	Mon 16/06/25	Mon 16/06/25		Óscar
101		**Requisitos**	**1 day**	**Tue 17/06/25**	**Tue 17/06/25**		
102		Matriz de rastreabilidad	1 day	Tue 17/06/25	Tue 17/06/25	100	Óscar
103		**Análisis**	**1 day**	**Wed 18/06/25**	**Wed 18/06/25**		
104		Verificar análisis	1 day	Wed 18/06/25	Wed 18/06/25	102	Óscar
105		**Diseño**	**1 day**	**Thu 19/06/25**	**Thu 19/06/25**		
106		Verificación del diseño	1 day	Thu 19/06/25	Thu 19/06/25	104	Óscar
107		**Implementación**	**2 days**	**Fri 20/06/25**	**Mon 23/06/25**		

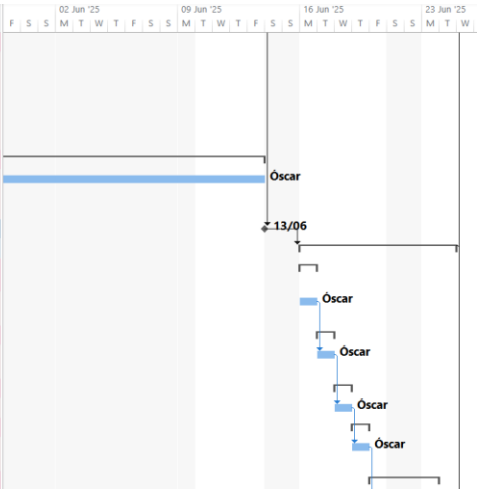


Ilustración 56: Diagrama de Gantt (7)

	Task Mode	Nombre de tarea	Duration	Start	Finish	Predecessors	Resource
108		Implementación del resto de pantallas	2 days	Fri 20/06/25	Mon 23/06/25	106	Óscar
109		**Pruebas**	**1 day**	**Tue 24/06/25**	**Tue 24/06/25**		
110		Pruebas a las nuevas pantallas añadidas	1 day	Tue 24/06/25	Tue 24/06/25	108	Óscar
111		**Hito fin iteración 3**	**0 days**	**Tue 24/06/25**	**Tue 24/06/25**	**98**	
112		**Hito fin contrucción**	**0 days**	**Tue 24/06/25**	**Tue 24/06/25**	**67**	
113		**Transición**	**7 days**	**Wed 25/06/25**	**Thu 03/07/25**	**112**	
114		**Iteración 1**	**7 days**	**Wed 25/06/25**	**Thu 03/07/25**		
115		**Modelado de negocio**	**1 day**	**Wed 25/06/25**	**Wed 25/06/25**		
116		Reunión con el cliente	1 day	Wed 25/06/25	Wed 25/06/25		Óscar
117		**Requisitos**	**1 day**	**Thu 26/06/25**	**Thu 26/06/25**		
118		Verificación final	1 day	Thu 26/06/25	Thu 26/06/25	116	Óscar
119		**Análisis**	**1 day**	**Fri 27/06/25**	**Fri 27/06/25**		
120		Verificación final	1 day	Fri 27/06/25	Fri 27/06/25	118	Óscar
121		**Diseño**	**1 day**	**Mon 30/06/25**	**Mon 30/06/25**		
122		Verificación Final	1 day	Mon 30/06/25	Mon 30/06/25	120	Óscar
123		**Implementación**	**1 day**	**Tue 01/07/25**	**Tue 01/07/25**		
124		Se obtendrá el ejecutable final de programa	1 day	Tue 01/07/25	Tue 01/07/25	122	Óscar
125		**Pruebas**	**2 days**	**Wed 02/07/25**	**Thu 03/07/25**		

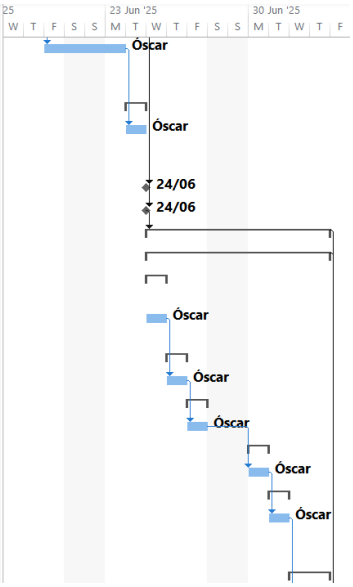


Ilustración 57: Diagrama de Gantt (8)

126		Pruebas de eficacia a la inteligencia artificial	1 day	Wed 02/07/25	Wed 02/07/25	124	Óscar
127		Pruebas de rendimiento en la generación de movimientos	1 day	Thu 03/07/25	Thu 03/07/25	126	Óscar
128		**Hito fin iteración 1**	**0 days**	**Thu 03/07/25**	**Thu 03/07/25**	**114**	
129		**Hito fin transición**	**0 days**	**Thu 03/07/25**	**Thu 03/07/25**	**113**	

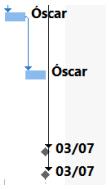


Ilustración 58: Diagrama de Gantt (9)

12. Conclusiones y trabajo futuro

El desarrollo de este motor de ajedrez ha supuesto un desafío considerable a nivel de planificación, debido a tener dudas a la hora de dividir el proyecto en partes modelares, y a nivel de programación, especialmente por la complejidad inherente a los algoritmos de búsqueda, la representación eficiente del tablero y el diseño de una función de evaluación competitiva. A pesar de estas dificultades, el resultado obtenido puede considerarse satisfactorio, teniendo en cuenta el alcance del proyecto.

La evolución de la inteligencia artificial desde su primera implementación es evidente, y es que las implementaciones iban mejorando la eficacia de los movimientos de esta, sin embargo, no al nivel suficiente. Actualmente, su poder competitivo se queda muy lejos de siquiera suponer un reto para un buen jugador, y ni hablar de los mejores motores de ajedrez.

La mayor debilidad notable de esta inteligencia artificial es muy posiblemente el final de partida, y es que, numerosas situaciones en las que la victoria es clara para la máquina, esta simplemente remueve sus piezas del tablero en movimientos repetitivos hasta que se declaran las tablas por triple repetición. Esto me lleva a pensar que puede existir un problema con las tablas de transposición, en la que se guardan evaluaciones que aparentemente son buenas, cuando realmente llevan a tablas y hacer perder toda la ventaja a la máquina. Incluso obviando el hecho de las tablas, es fácil pensar que hay algún problema con la evaluación *mop-up*, puesto que los movimientos de la máquina se centran demasiado poco en el rey, llego el final de partida.

Haciendo un repaso sobre el cumplimiento de los requisitos, tenemos lo siguiente:

- **Crear un tablero de ajedrez funcional**: El tablero de ajedrez es funcional al completo, al fin y al cabo, era la base para el resto de los objetivos.

- **Gestionar las situaciones de jaque, jaque mate y tablas**: El jaque mate y el jaque son correctamente tratados, evitando que surjan movimientos pseudolegales. En cuanto a las tablas, todas las situaciones de tablas son detectadas correctamente.

- **Permitir configurar los parámetros de la partida antes de empezarla**: El programa permite seleccionar y ajustar diversos parámetros para la partida.

- **Modo de juego contra otro jugador:** Existe la posibilidad de enfrentarse a un jugador en local.

- **Modo de juego contra la máquina:** Existe la posibilidad de enfrentar contra la máquina, aunque, seguramente, no con la dificultad esperada.

- **Selección de dificultad de la inteligencia artificial**: La dificultad es intercambiable entre tres modos de dificultad.

- **Gestionar el tiempo de partida de los jugadores**: Los jugadores se ven afectados por el tiempo de la partida, al igual que la inteligencia artificial, lo que permite establecer un tiempo límite en su búsqueda.

- **Cargar partidas en formato FEN**: Cualquier cadena de texto en formato FEN es cargada correctamente en el tablero.

Elaborar este trabajo me ha permitido expandir mi conocimiento muy significativamente respecto al funcionamiento de estos motores de ajedrez y respecto a la creación y desarrollo de proyectos software a gran escala. Gracias a esto, he podido saber lo importante que es la organización en la elaboración de un trabajo similar, y lo importante que es saber investigar e informase de manera autónoma. Además, he podido adquirir bastantes nociones sobre cómo trabajar con motores de videojuegos, en concreto, con el que estado usando para este proyecto, Unity.

Si hablamos de futuras líneas de desarrollo, estas deberían centrarse en dos aspectos fundamentales para la mejora de rendimiento del programa:

- La mejora de la evaluación llegado el final de partida.

- Mejorar la eficiencia de la generación de movimientos, posiblemente de la mano de los llamados "*magic bitboards*".

En cuanto a funcionalidad, hubiera sido interesante elaborar un modo multijugador en línea, o incluso la posibilidad de tener un historial de partidas. Sin embargo, debido a centrarse en el desarrollo de la inteligencia artificial, estas funcionalidades no han podido florecer.

13. Bibliografía

(n.d.). Unplash. Retrieved from https://images.unsplash.com/photo-1721490571166-356691cfd62d?fm=jpg&q=60&w=3000&ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxzZWFyY2h8M3x8Zm9uZG8lMjBkZSUyMHBhbnRhbGxhJTIwZGUlMjBhamVkcmV6fGVufDB8fDB8fHww

Board representation in chess programming. (1999). Retrieved from University of Alabama at Birmingham: https://web.archive.org/web/20130212063528/http://www.cis.uab.edu/hyatt/boardrep.html

Byrne, J. (2021). An Overview of Natural Language Processing and Speech Technologies with Common Methods. Medium. Retrieved from https://ai.plainenglish.io/how-will-natural-language-processing-nlp-change-the-world-823399d48bd5

Computer Chess Rating List (CCRL). (n.d.). Retrieved from https://computerchess.org.uk/ccrl/4040/

Condon, J., & Thompson, K. (1982). Belle Chess Hardware. *Advances in Computer Chess 3*.

Coulom, R. (2007). Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games* (pp. 72–83). Springer.

Edwards, D. J. (1963). *The alpha-beta heuristic.* Artificial Intelligence Project, MIT.

El ajedrez, ¿una potente herramienta pedagógica? (2017). EduCaixa. Retrieved from https://educaixa.org/es/-/el-ajedrez-una-potente-herramienta-pedagogica

Feldmann, R. M. (1989). Distributed game-tree search. *ICGA Journal*.

Forsyth-Edwards Notation (FEN). (n.d.). Chess.com. Retrieved from https://www.chess.com/terms/fen-chess

Highest chess rating ever achieved by computers. (n.d.). Our World in Data. Retrieved from https://ourworldindata.org/grapher/computer-chess-ability

How do I write a Multi threaded Alpha-Beta Search algorithm? (n.d.). Stackoverflow. Retrieved from https://stackoverflow.com/questions/71546186/how-do-i-write-a-multi-threaded-alpha-beta-search-algorithm

How to Create a Chess Engine with TensorFlow (Python). (2024). YouTube. Retrieved from https://www.youtube.com/watch?v=v3jMr0Ppd9Y

Komodo Chess. (n.d.). Wikipedia. Retrieved from https://komodochess.com/store/pages.php?cmsid=14

Lague, S. (2021). Coding Adventure: Chess. YouTube. Retrieved from https://www.youtube.com/watch?v=U4ogK0MIzqk

lichess.org. (n.d.). Retrieved from https://lichess.org/

Marsland, T. C. (1980). *Parallel search of game trees.* Department of Computing Science, University of Alberta. Retrieved from https://webdocs.cs.ualberta.ca/~tony/TechnicalReports/TR80-7.pdf

Monte Carlo tree search. (n.d.). Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

Perft results. (n.d.). Chess Programming Wiki. Retrieved from https://www.chessprogramming.org/Perft_Results

Piece-Lists. (n.d.). Chess Programming Wiki. Retrieved from https://www.chessprogramming.org/Piece-Lists

Proceso unificado. (n.d.). Wikipedia. Retrieved from https://es.wikipedia.org/wiki/Proceso_unificado

Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine*, 256–275.

Stockfish. (n.d.). Chess Programming Wiki. Retrieved from https://www.chessprogramming.org/Stockfish

SVG chess pieces. (n.d.). Wikimedia Commons. Retrieved from https://commons.wikimedia.org/wiki/Category:SVG_chess_pieces

The 7 Most Mindblowing Magnus Carlsen Records. (2024). Retrieved from Chess.com: https://www.chess.com/article/view/7-most-mindblowing-magnus-carlsen-records

The Day Deep Blue Changed Chess and AI Forever: Kasparov's Historic Defeat. (2023). YOURSTORY. Retrieved from https://yourstory.com/2023/05/deep-blue-defeats-kasparov-historic-ai-milestone

Turing, A. (1953). Chess. In B. V. Bowden (Ed.), *Faster Than Thought: A Symposium on Digital Computing Machines* (pp. 256–275). Londres.

Viana, I. (2010). *Deep Blue, la máquina que desafió la inteligencia humana*. Retrieved from ABC: https://www.abc.es/historia/abci-deep-blue-maquina-desafio-inteligencia-humana-201002110300-1133709000633_noticia.html

Vučković, V. (2008). Compact Chessboard Representation. *ICGA Journal*.

Zobrist, A. L. (1970). *A new hashing method with application for game playing.* Department of Computer Sciences, University of Wisconsin. Retrieved from https://www.cs.wisc.edu/techreports/1970/TR88.pdf