



ANÁLISIS Y DESARROLLO DE UNA APLICACIÓN DE AJEDREZ

Rafael Rivas Pérez

Introducción	4
Sobre este proyecto	7
Análisis y requerimientos de la aplicación	8
3.1 Introducción al problema	8
3.2 Antecedentes	8
3.3 Objetivos	9
3.4 Requisitos	9
2.4.1 Requisitos Funcionales	9
2.4.2 Requisitos no funcionales	11
3.5 Recursos	12
2.5.1 Recursos hardware	12
2.5.2 Recursos software	12
Análisis de los aspectos relacionados con el juego	13
4.1 Generador de movimientos legales	13
4.2 Función de búsqueda	14
4.2.1 Algoritmo minimax	15
4.2.2 Poda alpha-beta	16
4.3 Función de evaluación	17
4.4 Sistema de puntuación elo	18
4.5 Otros aspectos a tener en cuenta	18
4.5.1 Base de datos de aperturas	18
4.5.2 Tablas de Nalimov	18
Diseño del software	19
5.1 Arquitectura	19
5.2 Modelados	20
5.2.1 Casos de uso	20
5.2.2 Diagramas de interacción	22
5.2.3 Diagramas de actividad	24
5.3 Base de datos	27
5.4 Prototipo gráfico	29
5.4.1 Pantallas principales	29
5.4.2 Logo	30
5.4.3 Splash Screen	31
Implementación	32
6.1 Servidor	32
6.2 Cliente	34
6.2.1 Paquete game	35
6.2.2 Paquete connectionTCP	37
6.2.3 Paquete dataaccess	37
6.2.4 Paquete session	37
6.2.4 Paquete ui	37
Conclusiones	38
Bibliografía	39

1. Introducción

El Ajedrez ha demostrado ser, hasta nuestros días, una fuente inagotable de retos y aprendizaje, desde quienes simplemente disfrutan del juego y los puzzles que este plantea hasta posteriormente los programadores, quienes aprovechando el potencial de las computadoras han intentado buscar las formas más eficientes de dar soluciones al problema así como intentar replicar la forma de pensar y el aprendizaje humano.

Desde mucho antes de la llegada de las computadoras el afán de conseguir que una máquina fuera capaz de jugar al ajedrez hizo que se desarrollaran ciertas ideas y artilugios. El primer éxito real en este campo fué en el año 1912 cuando el matemático español Leonardo Torres Quevedo construyó un autómata que mediante unos sencillos algoritmos y unos electroimanes bajo un tablero podía resolver cualquier final de rey y torre contra rey, el final más sencillo pero aún así todo un logro dado la época.



El ajedrecista, de Leonardo Torres Quevedo, 1912.

En el año 1950, antes de que aparecieran las primeras computadoras que jugaran al ajedrez, Claude Shannon publicó un artículo donde predijo acertadamente las dos posibles principales formas de búsqueda en cualquier programa, a las que denominó de Tipo A y de Tipo B.

Los programas de tipo A son más rudimentarios y utilizan una búsqueda basada en la fuerza bruta, examinando todas las posibles posiciones en cada rama del árbol de juego utilizando únicamente el algoritmo minimax. Shannon creyó que esto sería muy poco práctico ya que el computador tardaría mucho en analizar apenas 3 o 4 jugadas por delante, y más teniendo en cuenta la velocidad de procesamiento que se podía conseguir en la época.

Por otra parte Shannon sugirió que los programas de tipo B utilizarían una especie de inteligencia artificial capaz de desechar variantes malas sin tener que analizarlas y solo analizando aquellas prometedoras, en las que podría profundizar más al haber menos variantes. Sin embargo no fué hasta los años 90 cuando los programas de tipo B empezaron a ganar en popularidad.

Un hecho trascendental del que ya mucho se ha hablado, fueron los matches en 1996 y 1997 entre el entonces campeón del mundo Gary Kasparov y la supercomputadora Deep Blue, desarrollada por IBM, la derrota del legendario jugador llegó en el segundo match de 1997 y supuso la primera vez que una inteligencia artificial demostraba ser mejor que el humano más cualificado en una tarea compleja y creativa. Y aunque el hecho no estuvo exento de polémica y sospechas de la intervención humana en alguna jugada por parte de IBM, no importaría demasiado puesto que pocos años más tarde los software de ajedrez comenzaron a ser invencibles para los jugadores humanos.

SuperComputadora Deep Blue, IBM.



Gary Kasparov vs Deep Blue, 1996.

Con el paso de los años los programas de ajedrez han ido siendo cada vez más potentes, y aunque las estrategias y algoritmos aplicados son cada vez más precisos la fuerza real de estos software reside principalmente en la capacidad de procesamiento de los cpu, mientras que un jugador humano analiza unas decenas de posiciones por jugada en una partida clásica, un motor de ajedrez actual tiene que analizar millones de posiciones para alcanzar el nivel de un maestro aunque lo haga en mucho menos tiempo, por lo que en este sentido, y solo en este sentido, son ineficientes comparados con nuestra capacidad de aprendizaje y experiencia.

Pero en 2017, con las emergentes redes neuronales y aprendizaje automático se realizó un cambio de enfoque, Deep Mind desarrolló el programa AlphaZero el cual en solo unas horas de autoaprendizaje fué capaz de derrotar en gran medida a los mejores programas de ajedrez del mundo hasta la fecha. Gracias a las redes neuronales AlphaZero es capaz de tomar decisiones beneficiosas a largo plazo sin tener que haber evaluado infinidad de jugadas intermedias, sino basándose en la experiencia de partidas anteriores, lo que hace que se parezca a la forma de juego humano y tome decisiones inéditas para los programas de ajedrez hasta la fecha.

Por otra parte con la democratización de las computadoras personales y más tarde la aparición de internet no solo los programas de ajedrez llegaron a todo el mundo si no que empezaron a surgir sitios web donde poder jugar con otros usuarios de todo el mundo, almacenar las partidas, participar en torneos online, analizar las partidas, estudiar ajedrez y muchas otras aplicaciones.

Los Smartphone son hoy en día la forma más rápida y habitual de acceder a una aplicación casi de cualquier tipo, y las de ajedrez no son la excepción siendo el camino más acertado a la hora de llegar a una mayor cantidad de público gracias a su facilidad de uso y accesibilidad.

2. Sobre este proyecto

2.1 Descripción del proyecto

Este proyecto trata de construir una aplicación móvil para el sistema operativo Android que permita a los usuarios jugar al ajedrez con otros usuarios en tiempo real, así como guardar las partidas jugadas entre estos para poder rescatarlas y revisarlas desde la aplicación móvil posteriormente. También proporcionará un modo de juego offline contra una IA construida desde cero y aplicará un sistema de puntuación que haga más interesante el juego.

El verdadero interés del proyecto se encuentra en su complejidad técnica durante el desarrollo, la aplicación de algoritmia y la búsqueda eficiente de soluciones y no así en sus ideas innovadoras ya que es un terreno ampliamente explorado el de las aplicaciones de ajedrez.

2.2 Control de versiones

Se utilizará el software de control de versiones Git para mantener las diversas versiones tanto del proyecto cliente como del proyecto servidor. También se usará Gitlab como repositorio para ir almacenando los commit que se vayan haciendo en Git para tener el proyecto guardado seguro fuera del ordenador de desarrollo.

3. Análisis y requerimientos de la aplicación

3.1 Introducción al problema

Este proyecto cuenta con dos partes bien diferenciadas, y que deben de hecho considerarse como dos proyectos independientes durante el desarrollo.

La primera de ellas es la aplicación que permite a los usuarios jugar entre ellos, así como también realizar las tareas típicas de cualquier aplicación web dirigidas al público en general como registrarse, loguearse y guardar los datos correspondientes en una base de datos. Para esto es necesario que a su vez esta parte se divida en otras dos partes, la del cliente o aplicación móvil, y la del servidor, encargada de todo el intercambio y guardado de los datos.

La segunda es el software o IA que se encargará específicamente del análisis de las posiciones, denominado motor o módulo de ajedrez, siendo capaz de devolver el mejor movimiento estimado a una posición dada basándose en una serie de parámetros.

Más tarde el módulo de ajedrez será incluido en la aplicación, pasando a ofrecer nuevas funcionalidades para el usuario como el análisis de sus partidas, la consulta sobre posiciones específicas o la posibilidad de jugar contra el módulo.

3.2 Antecedentes

Las plataformas para jugar ajedrez en línea han existido desde los albores de internet, la primera de ellas fué Internet Chess Club en 1992. Actualmente existen tres grandes plataformas que acaparan a los usuarios: Chess.com, Chess24 y Lichess.

La última de ellas, Lichess, es un servidor de ajedrez en Internet, gratuito y de código abierto, gestionado por una organización sin ánimo de lucro del mismo nombre, no tiene publicidad y todas las funciones son gratuitas, ya que el sitio se financia con donaciones de los usuarios. Tiene aplicación para móviles android y en ella está basada mayormente la aplicación que se va a desarrollar en términos de diseño de la interfaz y usabilidad.

En cuanto a la IA de ajedrez han existido muchas implementaciones de software con este propósito a lo largo de los años, aunque actualmente el más importante es Stockfish, que es de código abierto y escrito en c++. Stockfish es el motor utilizado por todas las plataformas anteriormente mencionadas.

En los últimos años también se han desarrollado motores de ajedrez basados en redes neuronales como por ejemplo AlphaZero desarrollado por Google y LeelaZero de los mismos desarrolladores de Stockfish y cuyo desempeño está a la altura de los mejores motores tradicionales.

3.3 Objetivos

Al finalizar el desarrollo del proyecto se deben haber cumplido los siguientes objetivos:

- La aplicación deberá ser capaz de registrar usuarios y tras esto permitirlesloguearse y desloguearse de la aplicación.
- Permitirá al usuario jugar con otros usuarios en tiempo real y en diferentes modalidades de tiempo.
- La aplicación guardará de forma automática las partidas jugadas entre usuarios en una base de datos en un servidor remoto.
- Permitirá al usuario jugar offline contra una IA en diferentes modalidades de tiempo y dificultad.
- Ofrecerá un perfil de usuario donde este podrá ver sus partidas jugadas y reproducirlas sobre el tablero.
- Incluirá un sistema de puntuación de jugadores basada en la puntuación elo de torneos oficiales de ajedrez.

3.4 Requisitos

2.4.1 Requisitos Funcionales

1. Registro y login

- La primera pantalla que se mostrará si no se está logueado en la app será la de login. En esta pantalla habrá un campo para introducir el nombre de usuario y la contraseña, así como un botón para hacer login.
- En esta misma pantalla habrá otro botón más abajo para registrarse, al pulsarlo se abrirá otra pantalla con dos campos donde se podrá introducir el nombre de usuario y contraseña y un botón para registrarse. Al registrarse el usuario entrará a la aplicación logueado automáticamente.

2. Pantalla principal o home

- En esta pantalla habrá dos pestañas, la primera para jugar contra otros usuarios, la segunda para jugar contra la IA.
- En la primera pestaña se mostrarán varios botones cada uno correspondiente a un tiempo de juego diferente y que al pulsar uno de ellos se creará una conexión con el servidor poniéndose a la espera de que este le asigne otro jugador que haya

seleccionado ese mismo tiempo para jugar. Durante la espera se mostrará un mensaje de buscando oponente.

- En la segunda pestaña habrá diferentes botones (spinner) que al pulsarlos desplegarán un menú. El primero será para elegir con qué color se quiere jugar, el segundo el tiempo en minutos de la partida por jugador, el tercero el incremento de tiempo por jugada en segundos, y el último la dificultad a la que jugará la IA. Debajo de estos campos habrá un botón para comenzar la partida.
- En la parte superior derecha de la pantalla se encontrará un botón de menú que desplegará un menú lateral.
- En el menú lateral estará la opción de entrar en el perfil de ese usuario, cerrar la sesión, que llevará a la pantalla de logueo, y ajustes.

3. Juego

- En la pantalla de juego se especificará el nombre y la puntuación de cada jugador.
- Habrá un reloj en cada lado de cada jugador que mostrará el tiempo que le queda.
- Habrá una vista con scroll donde se irán mostrando las jugadas que se van haciendo en notación algebraica.
- El jugador solo podrá seleccionar las piezas durante su turno.
- Cuando se acabe la partida, por jaque mate o por tiempo, se mostrará un cuadro de diálogo con las opciones salir, que devolverá al jugador a la pantalla home, jugar, que buscará otro jugador para iniciar una nueva partida y revisar partida que abrirá la pantalla con un tablero para reproducir la partida.
- Al finalizar una partida entre dos usuarios esta se guardará automáticamente en la bd y se realizará el cálculo correspondiente a la variación de puntuaciones de cada usuario para guardarse también.

4. Perfil

- En la pantalla de perfil se mostrará el nombre y puntuación elo del jugador.
- También se mostrará un lista con scroll con los detalles de todas las partidas jugadas por ese jugador, al pulsar sobre cualquiera de ellas se abrirá la pantalla para revisar y reproducir esa partida.

5. Reproducción de partidas jugadas

- Esta pantalla será igual que la de juego, pero en lugar de poder seleccionar las piezas habrá dos botones de flechas para ir avanzando o retrocediendo jugadas en la partida.

2.4.2 Requisitos no funcionales

En este apartado se comentarán los requisitos no funcionales que tienen alguna relevancia específica para un funcionamiento deseado del software así como para su mantenibilidad y futuros desarrollos de funcionalidades.

1. Rendimiento

- Tiempo de respuesta del servidor durante una partida:

El envío de información de un cliente a otro pasando por el servidor durante la partida debe ser lo más corto posible dado que un factor del juego es el tiempo. Para esto la comunicación será por sockets con el servidor consiguiendo una latencia casi nula y la información enviada será pequeña.

- Tiempo de respuesta de la IA durante una partida:

El tiempo que requiere la evaluación de una posición por parte de la IA construida no debe ser demasiado largo, no más de 2 segundos, en caso contrario, si la búsqueda fuera muy profunda y llevara mucho tiempo se debería avisar al usuario antes de la partida. El tiempo de respuesta tampoco puede ser demasiado corto ya que parecería que el movimiento se realiza a la vez que el movimiento del jugador perdiendo la sensación de secuencia de turnos y generando cierta incomodidad en el usuario.

2. Seguridad

- El único dato violable en la aplicación es la contraseña de acceso, esta debe encontrarse encriptada en la base de datos.

3. Facilidad de uso

- Se debe poder acceder a lo que ofrece la aplicación de la forma más directa posible. En esta aplicación el usuario podrá estar jugando con solo pulsar un botón en el menú tras abrir la app.

4. Escalabilidad

- La aplicación que se va a desarrollar es muy susceptible de sufrir mejoras y agregar nuevas funcionalidades en un futuro ya que con lo desarrollado en el actual trabajo solo tiene las funcionalidades básicas, por lo tanto debe dejarse un código modularizado y abstracto en función de lo posible con el objetivo de poder agregar y refactorizar código de una manera simple.

5. Reusabilidad

- Como ya se ha mencionado el código está modularizado y las partes son fácilmente extraíbles para utilizarlas en proyectos futuros, por ejemplo el motor de análisis está en un paquete propio y se puede extraer directamente sin dependencias.

3.5 Recursos

2.5.1 Recursos hardware

- Ordenador para el desarrollo
- Smartphone con sistema operativo android
- Servidor remoto que contendrá la aplicación servidor

2.5.2 Recursos software

- Java 17: Última versión LTS (Soporte a largo plazo) de java a fecha de junio de 2022.
- IntelliJ Idea: Entorno de desarrollo para el desarrollo de la aplicación servidor.
- Android Studio: Entorno de desarrollo basado en IntelliJ Idea para el desarrollo de la aplicación cliente.
- Spring Boot: Framework de Java para el desarrollo de aplicaciones web con el que se desarrollará la API REST del servidor.
- Postman: Software para hacer peticiones web de una forma sencilla con el que se harán las pruebas necesarias para comprobar que la API funciona bien.
- MySQL: Sistema de gestión de bases de datos relacional que se utilizará para el almacenamiento y gestión de los datos.
- MySQL Workbench: Interfaz gráfica para utilizar fácilmente mysql .
- Adobe XD para el prototipado gráfico de la aplicación.
- Dia: Software para realizar diagramas UML.

4. Análisis de los aspectos relacionados con el juego

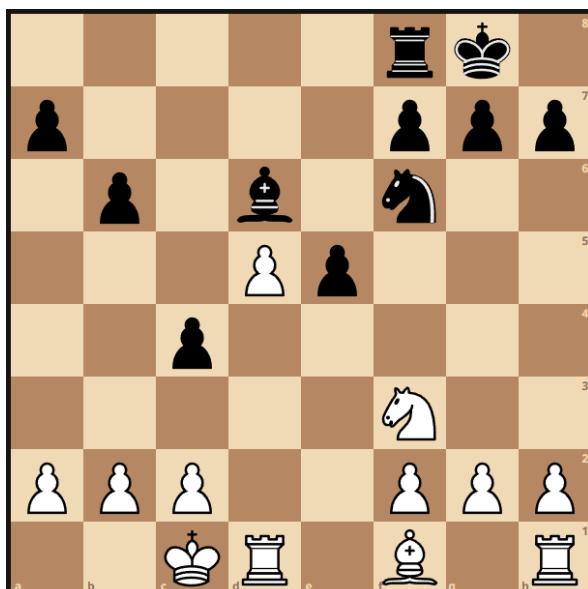
Aunque este proyecto no trata de profundizar en todas las diferentes técnicas que se pueden aplicar para el desarrollo de un motor de ajedrez y que daría para un proyecto aparte en sí mismo, sí que es conveniente analizar los métodos que se van aplicar en la actual aplicación para que el juego pueda realizar tareas tales como encontrar las jugadas que son legales en una determinada posición, identificar cuando hay jaque, jaque mate o ahogados, o realizar una búsqueda en el árbol de juego y evaluar las posiciones resultantes para encontrar la mejor jugada a una cierta profundidad.

En un programa o motor de ajedrez hay tres partes fundamentales: el generador de movimientos, que se encarga de encontrar los movimientos que son posibles según las reglas en una posición, la función de búsqueda, encargada de recorrer el árbol de juego encontrando las posiciones futuras, y la función de evaluación, encargada de evaluar cómo de buena o mala es una posición determinada para un jugador. A continuación se analizará cada una de estas partes.

4.1 Generador de movimientos legales

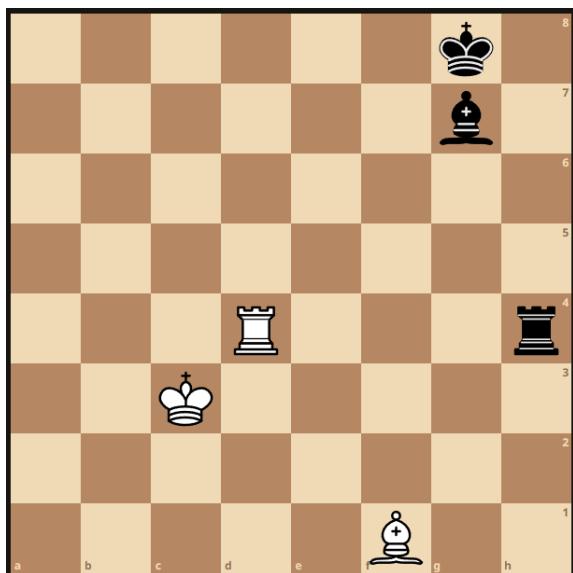
Los movimientos legales de una posición son aquellos que un jugador puede realizar en una determinada posición aplicando las reglas del juego.

El programa primero debe ir comprobando según los movimientos de cada pieza si una pieza determinada puede estar en una casilla determinada, por ejemplo si una pieza según su movimiento puede moverse una casilla al frente habrá que comprobar si esa casilla ya está ocupada o no, y en caso de estarlo si está ocupada por una pieza de su color o del color contrario.



Por ejemplo, para el alfil blanco de casillas blancas el programa comprobará si puede colocarse en g2, y verá que hay una pieza de su color así que no seguirá comprobando en esa dirección, luego comprobará si puede colocarse en e2 y verá que está vacía, así que agrega esa casilla a movimientos legales, lo mismo con d3, y en c4 verá que hay una pieza del color contrario, esta también la agrega a movimientos legales pero no seguirá comprobando más allá puesto que no puede saltar a la pieza enemiga. También debe comprobar los bordes del tablero para no salirse.

Se debe comprobar también si un movimiento genera jaque al propio rey, y si es así eliminar ese movimiento de las jugadas legales. Por ejemplo:



Si es el turno de las blancas, la torre no podrá moverse puesto que entonces el alfil negro dará jaque al rey blanco (Se dice que la torre blanca está clavada).

Por último también debe comprobarse si el rey está en jaque o jaque mate. Si el rey está en jaque sólo serán movimientos legales aquellos en los que en la posición resultante el rey ya no esté en jaque. En cuanto al jaque mate se detectará cuando el número de movimientos legales sea 0 y el rey esté en jaque, y por último el ahogado se detectará cuando el número de movimientos legales sea 0 y el rey no esté en jaque.

4.2 Función de búsqueda

La función de búsqueda consiste en recorrer el árbol de juego partiendo desde una posición hasta una determinada profundidad, o cantidad de movimientos, tomando las posiciones resultantes para evaluarlas y quedarse con la mejor puntuada para uno de los jugadores.

Un árbol de juego es un grafo en forma de árbol cuyos nodos representan posiciones en juego y cuyas aristas representan movimientos de los jugadores. Cualquier sucesión de jugadas puede representarse por un camino conexo dentro del árbol de juego.

Existen una multitud de algoritmos que se aplican en los programas de ajedrez y se combinan entre ellos para intentar optimizar la búsqueda de posiciones prometedoras, intentando reducir el número de posiciones a analizar y eliminar ramas completas del árbol donde se sabe que no se va a encontrar ninguna jugada buena. Entre todos ellos aquí analizaremos dos, el más simple e ineficiente pero a la vez base de todos los demás, el algoritmo minimax, y una mejora de este, la poda alpha-beta.

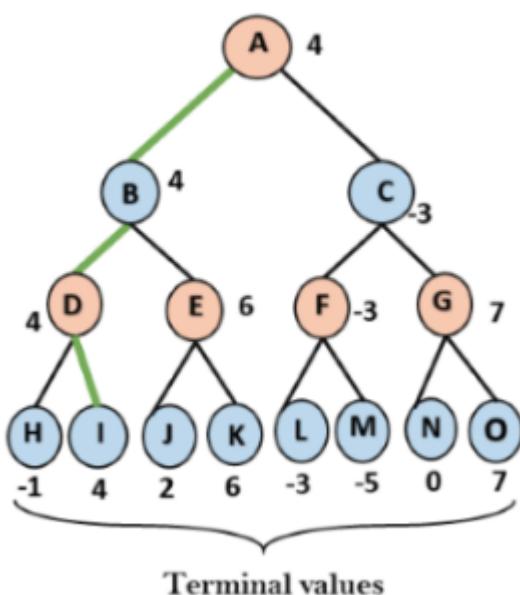
4.2.1 Algoritmo minimax

El ajedrez es un juego de suma cero, esto significa que la cantidad de ventaja que consigue un jugador supone la misma cantidad de desventaja para el otro jugador. En teoría de juegos el algoritmo minimax demuestra que para juegos de suma cero con información perfecta existe una única solución óptima.

Otro aspecto que se tiene en cuenta en este algoritmo es que se tratará de encontrar la jugada más ventajosa para el jugador pero teniendo en cuenta que el rival responderá con las jugadas más ventajosas para él, así pues en cada movimiento dentro de una búsqueda se tratará de buscar la jugada más beneficiosa para el jugador del que es el turno en ese movimiento.

Por tanto el programa partiendo de la posición a analizar hará una secuencia de movimientos hasta la profundidad especificada, la posición resultante se evaluará con un valor numérico, positivo si es a favor de las blancas y negativo si es a favor de las negras, y continuará evaluando todas las posiciones que se pueden dar para el último movimiento de esa primera variante, una vez terminado se quedará con el valor más favorable para el color que realiza ese último movimiento. Cuando se tienen los valores para el último nodo de todas las ramas del árbol que se está analizando, se va propagando hacia arriba del árbol esos valores cogiendo en cada capa el valor más favorable para el color del que es el turno hasta llegar arriba donde solo queda un valor acompañado del movimiento más óptimo.

La representación de este algoritmo como esquema de árbol es el siguiente:



Aquí tendríamos una búsqueda a profundidad tres, el color naranja serían las posiciones donde le toca a las blancas y el azul a las negras. Siempre que le toque a las blancas el programa se quedará con el valor más alto de los nodos inferiores y siempre que le toque a las negras con el valor más bajo, consiguiendo el mejor movimiento para la posición inicial si siempre se escogen los mejores movimientos para cada bando en posiciones posteriores.

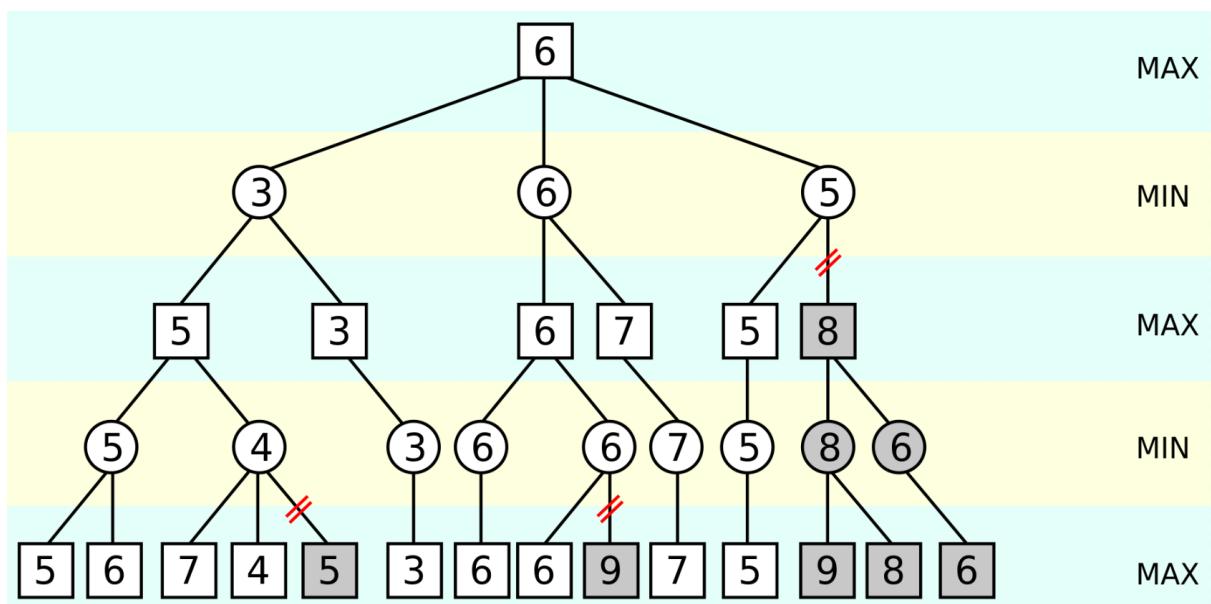
4.2.2 Poda alpha-beta

La poda alpha-beta es una mejora del algoritmo minimax y tiene como objetivo reducir el número de posiciones o nodos a analizar.

Alpha es el valor de la mejor opción hasta el momento a lo largo del camino para MAX, esto implicará por lo tanto la elección del valor más alto. Beta es el valor de la mejor opción hasta el momento a lo largo del camino para MIN, esto implicará por lo tanto la elección del valor más bajo.

Esta búsqueda alfa-beta va actualizando el valor de los parámetros según se recorre el árbol. El método realizará la poda de las ramas restantes, eliminándolas, cuando el valor actual que se está examinando sea peor que el valor actual de α o β para MAX o MIN, respectivamente.

En el siguiente ejemplo se explica con más claridad:



Si nos fijamos en la poda más grande a la derecha del esquema MIN encuentra un valor de 5 en la primera rama que analiza, este número nunca va a crecer, solo puede decrecer dado que se está escogiendo número más pequeño porque estamos en MIN, y como ya hay un 6 en la otra rama y sabemos que el nodo superior MAX va a escoger el número más alto este nodo nunca puede ser escogido por más posiciones que se analicen porque nunca va a ser superior a 5 por lo tanto lo podemos descartar.

4.3 Función de evaluación

La función de evaluación es la que decide si una posición es favorable para un bando u otro y en qué cantidad, y es en esta parte donde más variabilidad hay a la hora de diseñar un motor de ajedrez, puesto que hay una infinidad de parámetros que se pueden tener en cuenta y que se pueden ajustar para darle mayor o menor relevancia.

Entre los parámetros más importantes a la hora de evaluar una posición encontramos los siguientes:

- Material: Cuenta la cantidad de piezas sobre el tablero de un bando y otro, otorgándole un valor a cada una según su tipo. Esta es la forma más básica de evaluar una posición y por si sola no es fiable.
- Movilidad: Cada pieza recibe un pequeño bonus a su valor mientras más casillas controle.
- Control del centro: Controlar el centro del tablero suele ser ventajoso, por lo que tener piezas apuntando a ellas aumenta el valor de la posición, aunque esto depende de más factores.
- Espacio: Disponer de más espacio o tener las piezas más adelantadas hace que el jugador tenga más opciones de disponer sus piezas, mientras que el oponente se queda con pocos movimientos posibles reduciendo su disponibilidad de buenos movimientos.
- Seguridad del rey: Tener el rey rodeado de otras piezas de su mismo color hace que sea más difícil atacar por parte del rival.
- Estructura de peones: Los peones doblados o que acaban en la misma columna penalizan la posición porque no pueden protegerse entre ellos y son fácilmente atacables, mientras que cadenas de peones bien estructuradas crean una posición sólida.
- Peones a punto de coronar: Los peones cerca de la coronación aumentan considerablemente su valor.
- Piezas defendidas: Las piezas que no están defendidas tienen penalización porque son vulnerables a trucos como descubiertas, tenedores etc.

4.4 Sistema de puntuación elo

Este no es un aspecto intrínseco del juego sino algo externo pero ampliamente aplicado en el mundo del ajedrez. La puntuación que se asigna a los jugadores va en función de si se gana, pierde o entabla y de la puntuación del oponente. Tras una partida se aplica la siguiente fórmula para averiguar la nueva puntuación elo de un jugador:

$$RA_1 = RA + k (SA - EA)$$

Siendo:

RA = La puntuación elo que tiene el jugador en la partida.

k = factor que depende del nivel de ajedrez o de si se es nuevo pero que vale 20 en la mayoría de casos.

SA = Resultado de la partida. 0 si perdió, 1 si ganó o 0.5 si fueron tablas.

EA = Resultado esperado. Su valor se consulta en la tabla de Arpad Elo mirando el valor adjudicado a la diferencia de elo entre el jugador del que se está midiendo la puntuación menos el elo del oponente.

4.5 Otros aspectos a tener en cuenta

4.5.1 Base de datos de aperturas

A día de hoy la mayoría de las aperturas están muy estudiadas y es complicado para un motor de ajedrez realizar bien una apertura solo con el cálculo y la evaluación de posiciones, ya que estas contienen un valor estratégico a largo plazo difícil de llegar a evaluar para el motor, por lo que suele darse una base de datos para que sirva de guía durante la apertura, mejorando el nivel de juego durante esta fase y reduciendo la cantidad de computo.

4.5.2 Tablas de Nalimov

Las tablas de Nalimov son las tablas de todos los finales posibles, hoy en día conseguido hasta finales con 7 piezas. Aunque lleven el nombre del matemático Nalimov, ha habido muchos contribuyentes a lo largo de la historia en estas tablas, empezando por Richard Bellman y Ken Thompson en los años 70.

Estas tablas hacen que un motor pueda jugar finales perfectos hasta con 7 piezas, pero el tamaño es demasiado grande, 100 TB en su versión más completa, aunque se puede reducir hasta a 16.7 TB aplicando ciertas reglas para eliminar variantes innecesarias.

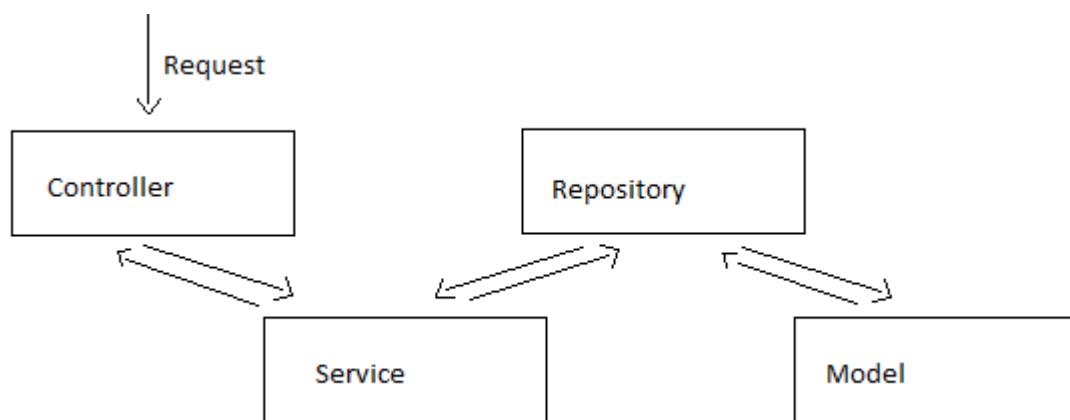
5. Diseño del software

5.1 Arquitectura

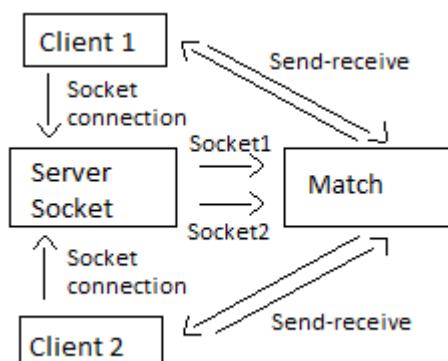
La arquitectura utilizada será un cliente-servidor donde cualquier dato que tenga que guardarse o recuperarse de la base de datos o transferirse a otro cliente pasará por el servidor.

El servidor consta de dos partes, cada una dedicada a un propósito diferente:

1. La primera es la API REST que se encargará de recibir todas las peticiones HTTP, consultar la base de datos y devolver los datos requeridos al cliente. La API está construida de la forma típica, un controlador que recibe las peticiones en una ruta determinada, un repositorio que contiene los métodos que consultan la base de datos, un servicio que comunica el controlador y el repositorio, y un modelo que contiene las clases que representan a las entidades de la base de datos.

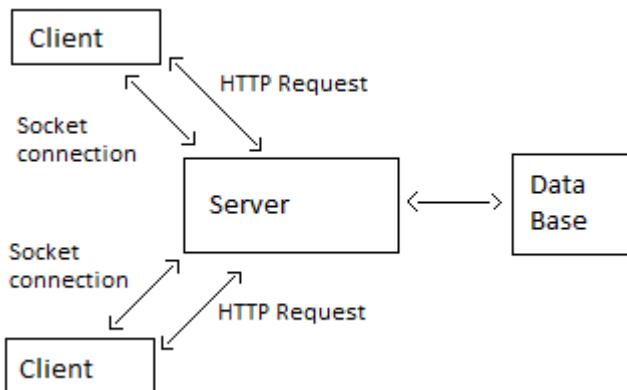


2. La segunda es la encargada de crear conexiones persistentes por medio de sockets con los clientes y recibir y enviar información por estas desde un cliente a otro. Esto servirá a la hora de enviar las jugadas en una partida haciendo que sea mucho más rápido y seguro el envío de información.



Por otra parte tenemos al cliente móvil, este es el que tendrá la mayor carga de trabajo puesto que el servidor quizás tenga que manejar muchas peticiones y sockets a la vez. En el cliente se ejecutará toda la lógica del juego y el motor de análisis, además de lógicamente todas las acciones del usuario y la interfaz gráfica que se detallarán en los siguientes apartados.

El siguiente esquema refleja una vista general de todo el sistema:

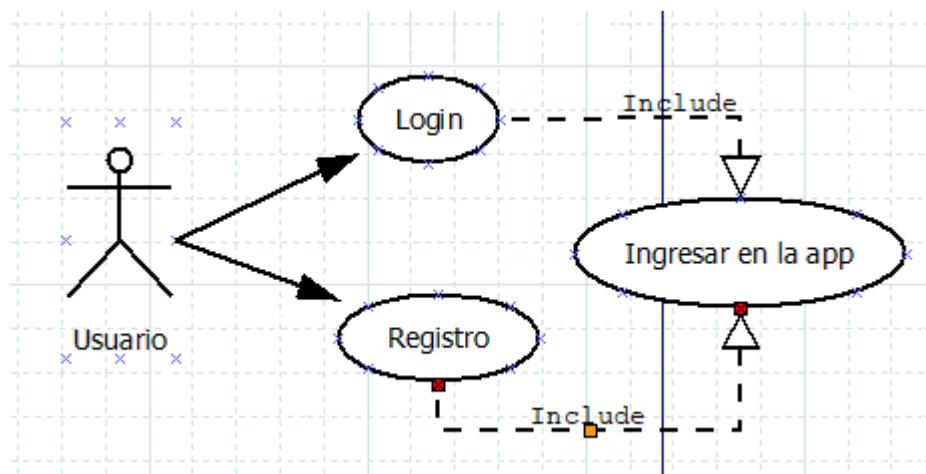


5.2 Modelados

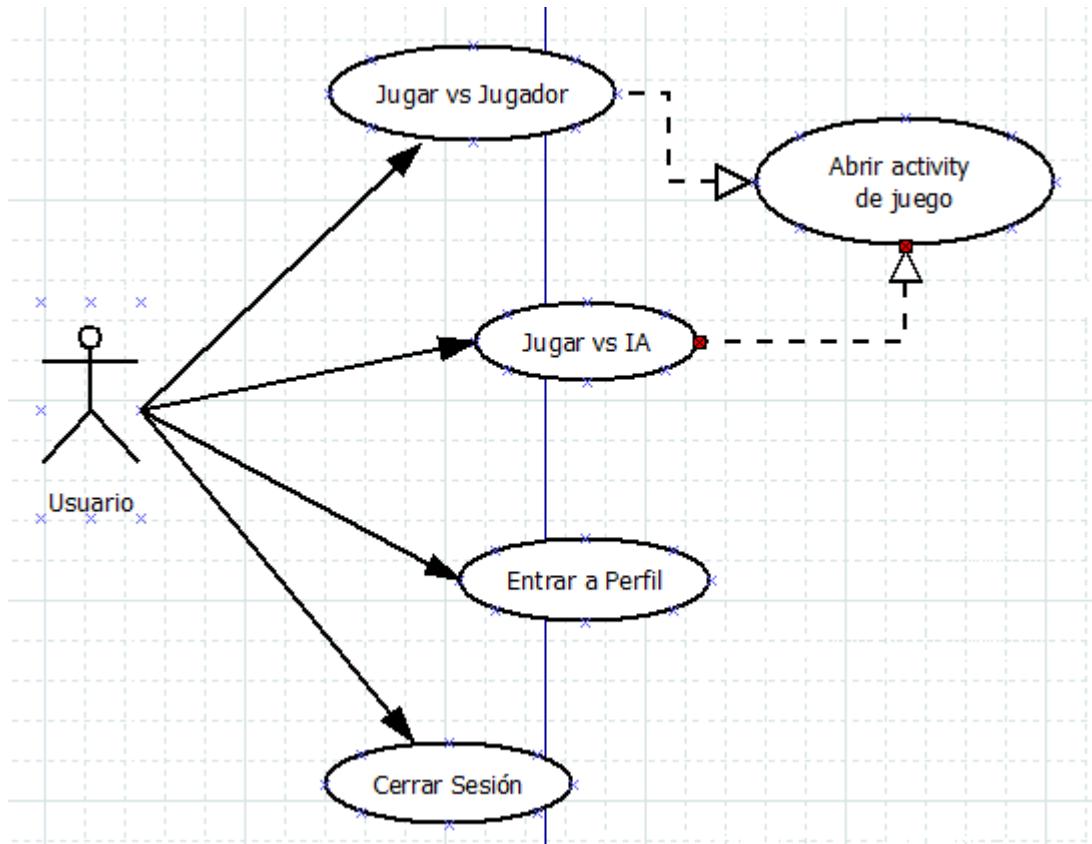
En este apartado se mostrarán y explicarán los diagramas diseñados UML utilizados para el diseño de la aplicación. Son diagramas que reflejan partes de la aplicación que tienen un sentido por sí mismas y no diagramas de la aplicación al completo para no hacer esquemas demasiado extensos y complicados.

5.2.1 Casos de uso

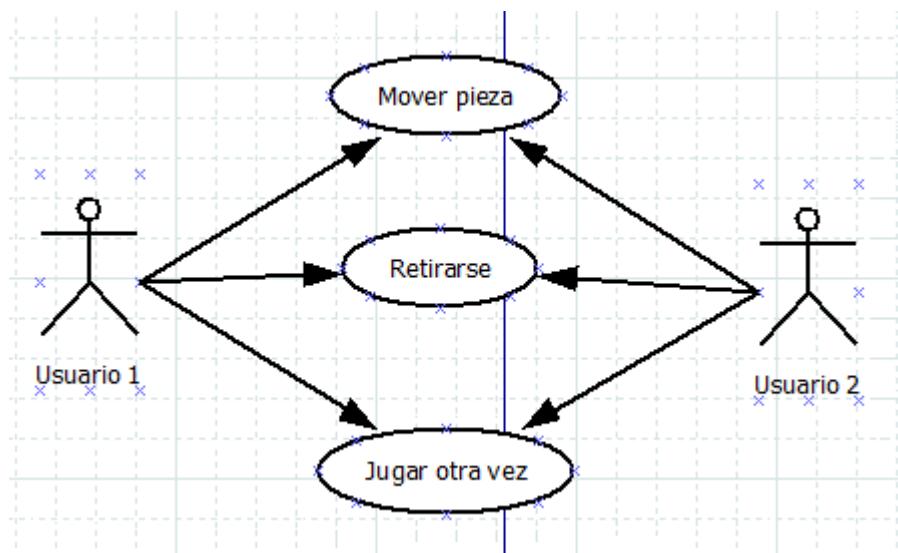
- **Login**



- Desde el Home de la app

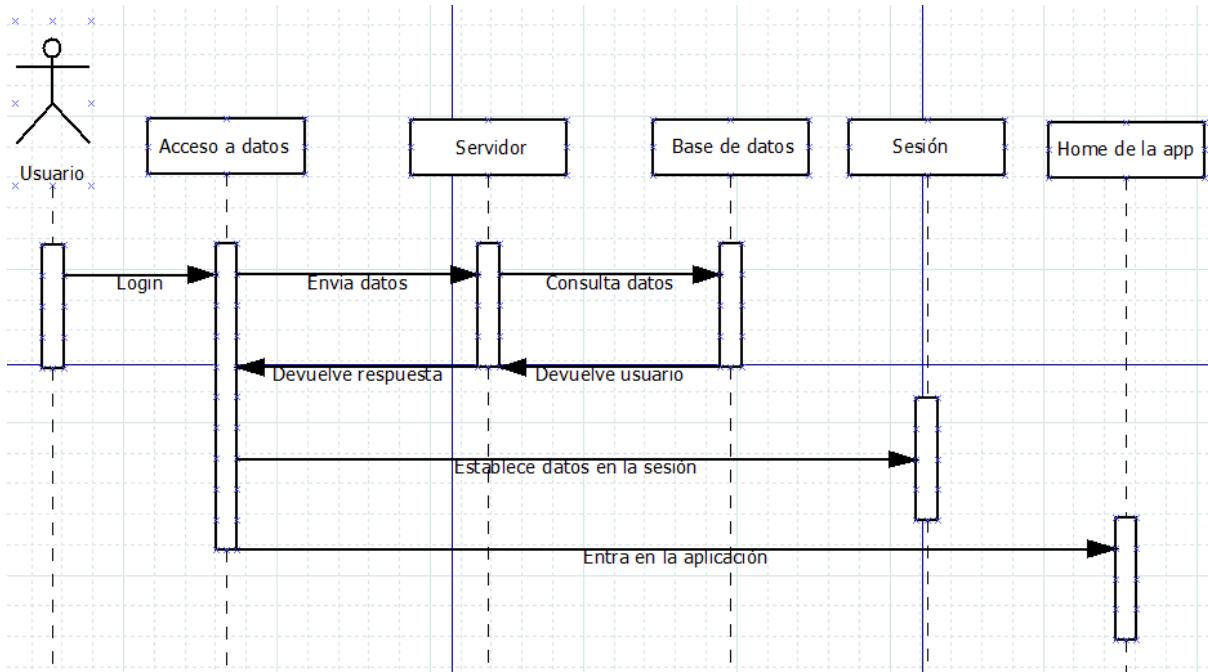


- Durante un juego con otro usuario

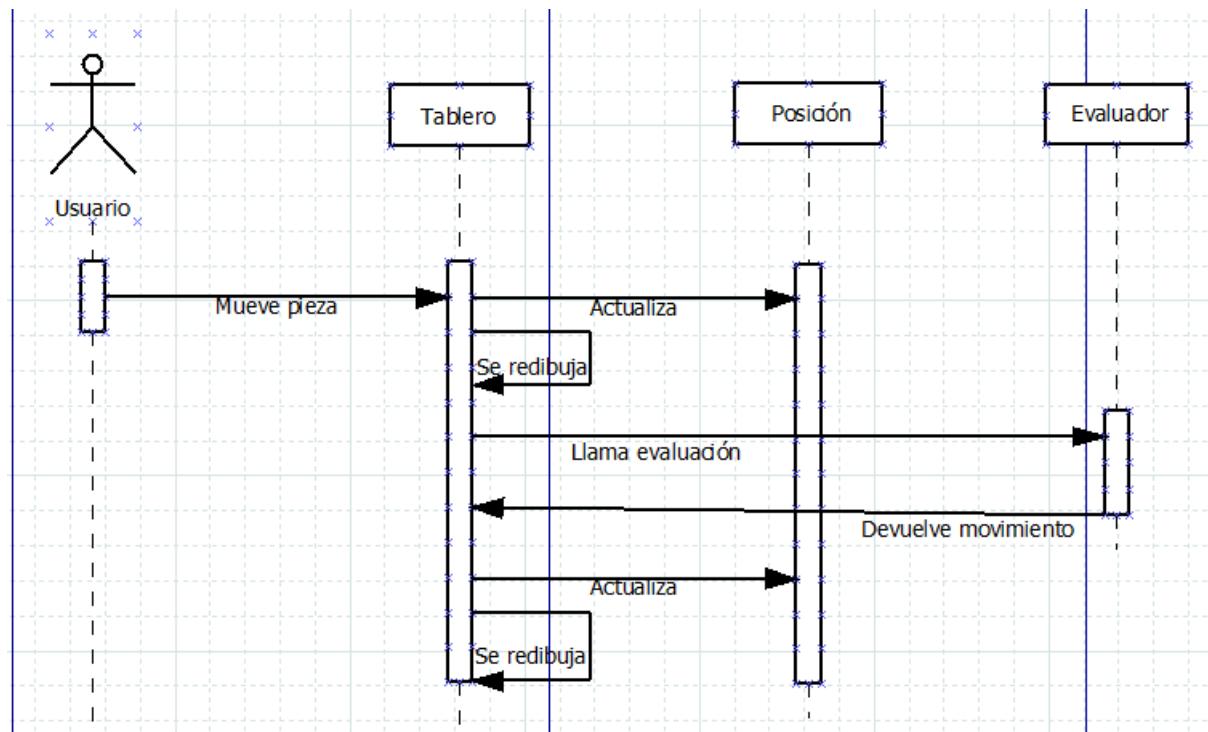


5.2.2 Diagramas de interacción

- Login

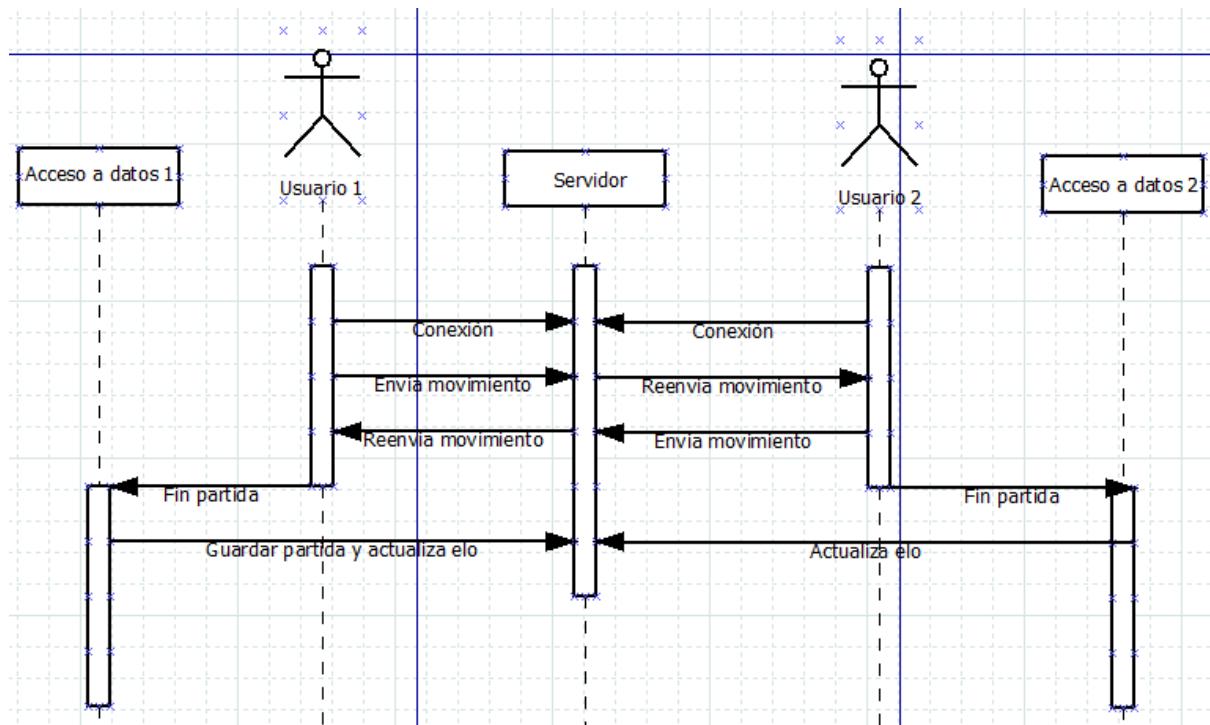


- Interacciones principales durante un juego contra la IA



El tablero es el componente gráfico dentro de la interfaz de usuario donde el usuario puede realizar el movimiento durante su turno, este envía las coordenadas de la pieza movida a la posición que es el que almacena la información de la partida, el tablero entonces se redibuja con la nueva posición y envía esta al evaluador que será el encargado de evaluarla y devolver el mejor movimiento estimado para el turno del ordenador, por último el tablero actualiza otra vez la posición con ese movimiento y se redibuja.

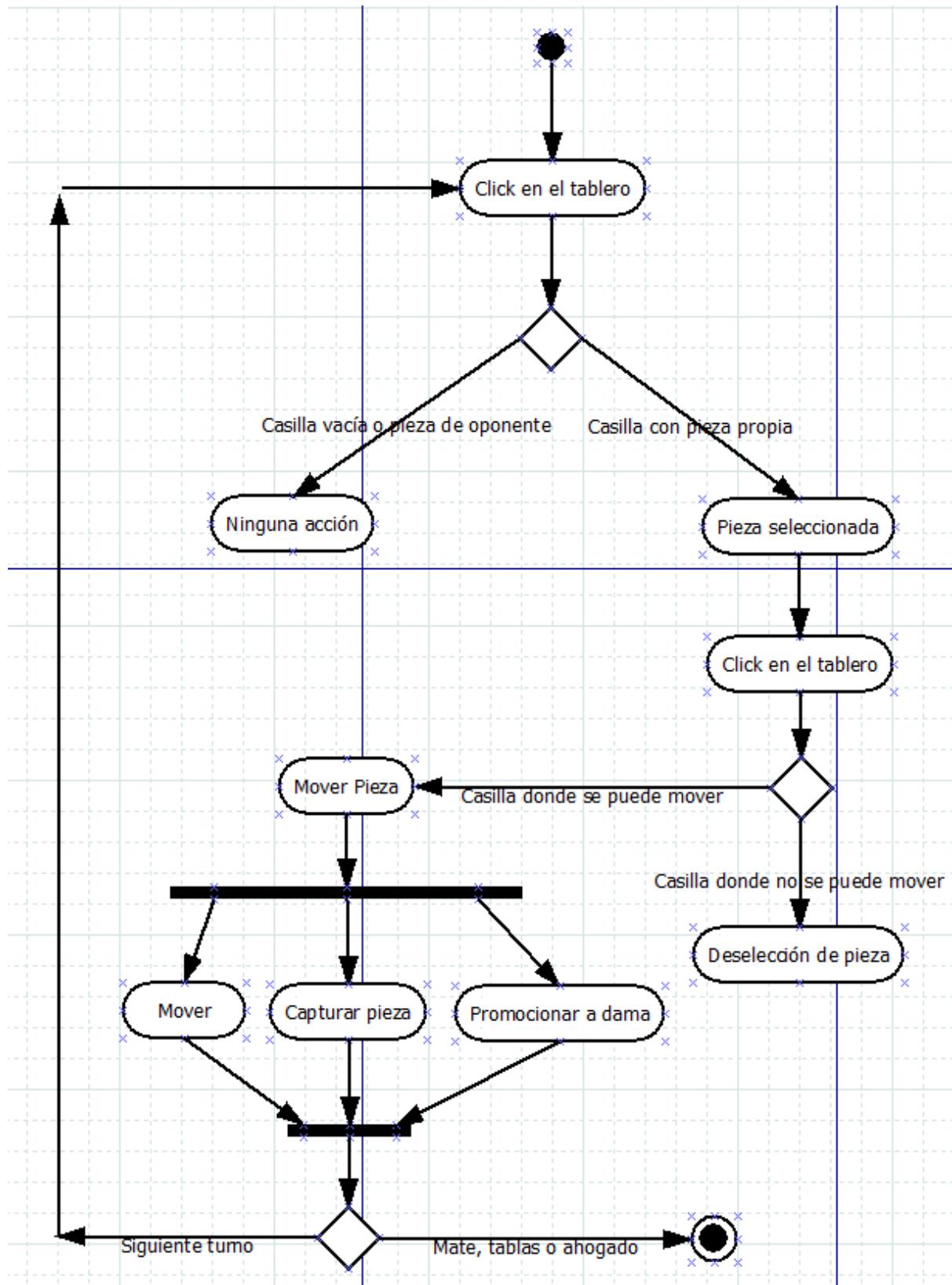
- Interacciones clientes y servidor durante una partida



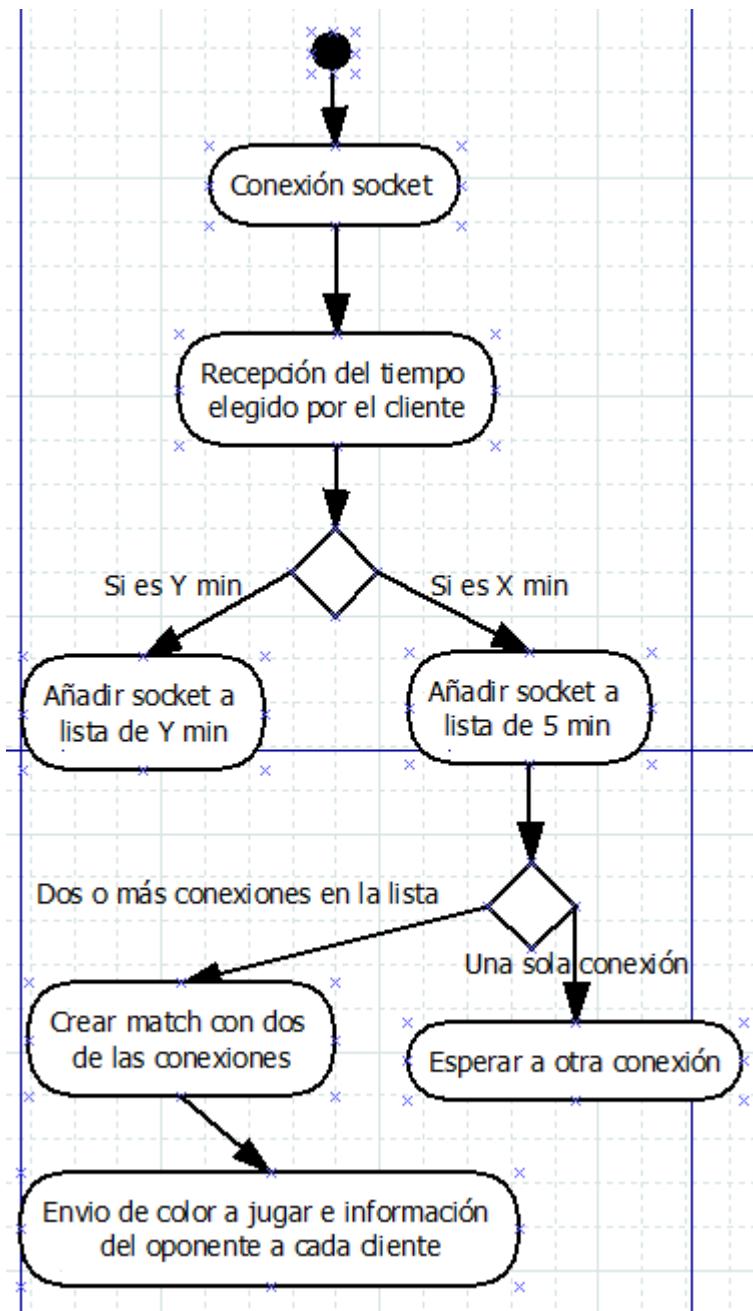
Tras la conexión socket de los clientes con el servidor este genera un match entre ambos que se encargará de reenviar los movimientos entre un jugador y otro. Una vez finalizada la partida el jugador ganador creará una conexión HTTP para guardar la partida y actualizar sus datos, mientras que el jugador perdedor sólo actualizará sus datos, esto es así porque solo uno debe ser el que guarda la partida para no crear duplicados en la bd y debe ser el que gana ya que el que pierde podría haber perdido por desconexión y por lo tanto la partida no se guardaría, tampoco es el servidor quien la guarda ya que hay que quitarle todo el trabajo posible a este para que el envío de los datos de movimientos sea lo más rápido posible para todas las partidas que haya en juego. En caso de tablas serán las blancas quienes guarden la partida.

5.2.3 Diagramas de actividad

- Pasos llevados a cabo durante los movimientos de una partida

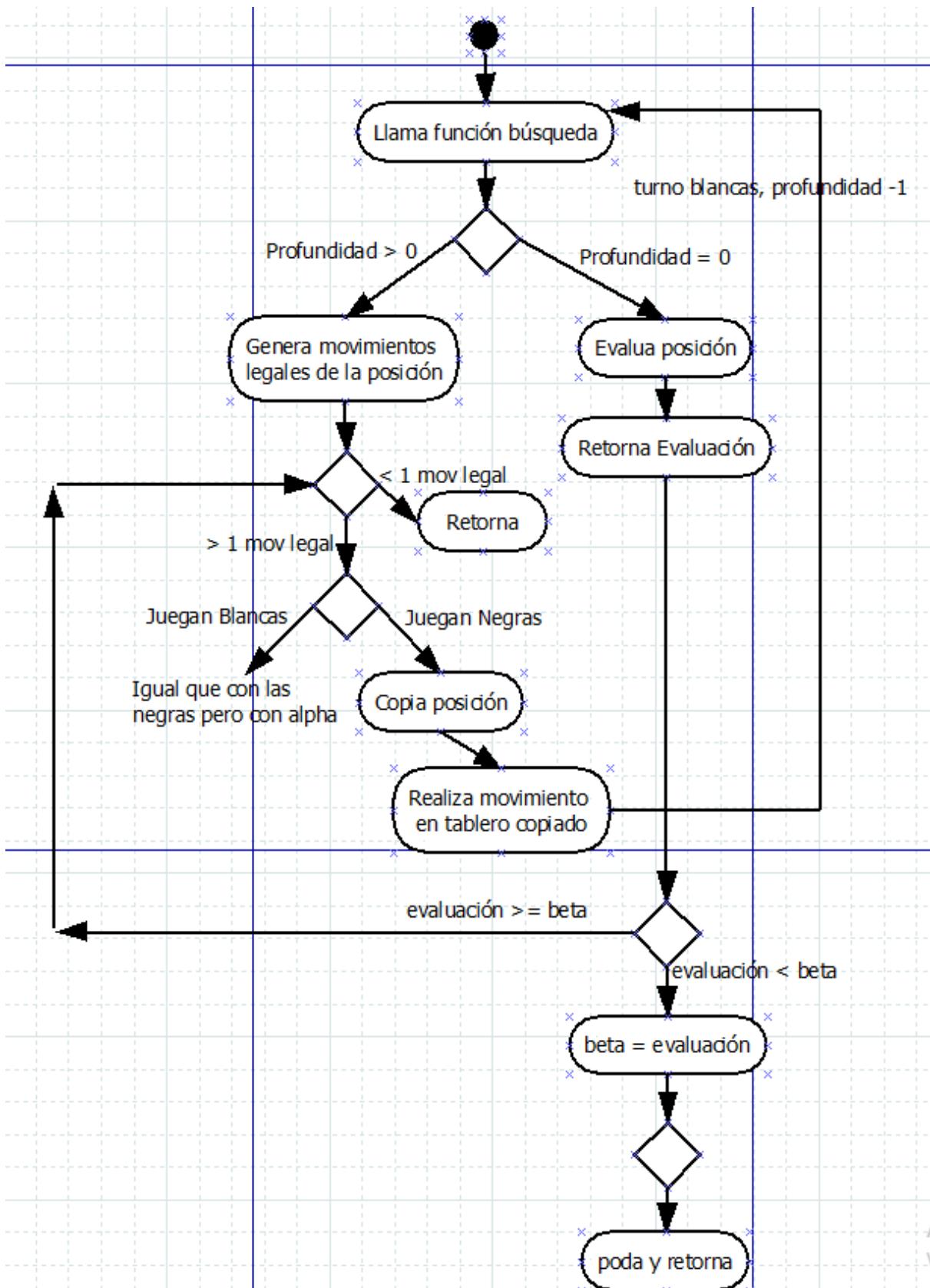


- Manejo de las conexiones persistentes del servidor con los clientes y generación de matches



El servidor está a la escucha de las conexiones entrantes, cuando un cliente inicia una conexión el servidor la acepta, creando una conexión persistente de tipo socket cliente-servidor, tras ello el cliente manda al servidor los datos correspondientes al tiempo escogido para la partida y en función a este tiempo guarda el socket en una lista u otra. Si en dicha lista hay más de una conexión con clientes entonces el servidor toma dos de esas conexiones, las saca de la lista y crea un match con las dos en un hilo nuevo donde se reenviará la información entre un cliente y otro pasando siempre por el servidor.

- Algoritmo MiniMax con poda alpha-beta



5.3 Base de datos

La aplicación requiere de una base de datos sencilla, que debe tener tanto los registros de los usuarios como de las partidas jugadas por estos para que puedan ser recuperadas. La base de datos entonces consta de dos tablas, la tabla User y la tabla Game.

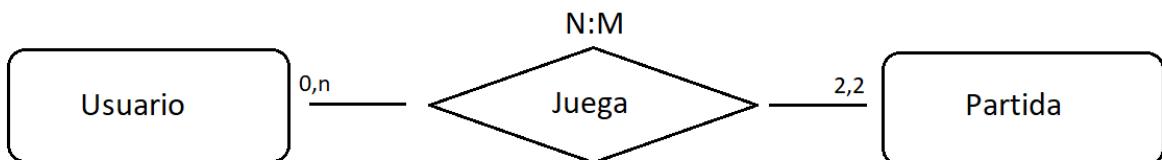
Campos que requiere la tabla User:

- id : Será un campo de tipo long autoincremental.
- nickname : Nombre escogido por el usuario y que no podrá ser repetido por dos usuarios diferentes.
- password : Campo de tipo Varchar como contraseña del usuario.
- elo : Campo de tipo entero que corresponde a la puntuación elo del usuario.

Campos que requiere la tabla Game:

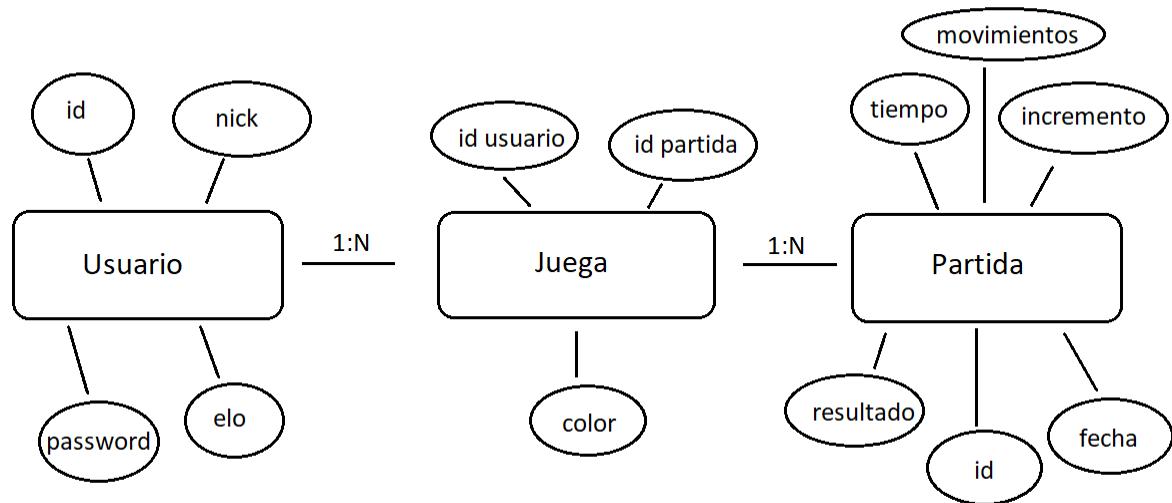
- id : Será un campo de tipo long autoincremental.
- date : Campo de tipo date que guardará la fecha en que se jugó la partida.
- whiteplayer : Guardará el nickname del jugador de piezas blancas.
- blackplayer : Guardará el nickname del jugador de piezas negras.
- time : Campo de tipo entero que guardará el número de minutos al que se jugaba la partida.
- whiteelo : Puntuación elo del jugador blanco en esa partida como campo entero.
- blackelo : Puntuación elo del jugador negro en esa partida como campo entero.
- increment : Campo de tipo entero en el que se guardará el incremento de tiempo con el que se jugó la partida.
- moves : Campo de tipo String que guardará los movimientos realizados en la partida.
- result : Campo de tipo entero que guardará el resultado de la partida con 1 si ganaron las blancas, -1 si ganaron las negras y 0 si fueron tablas.

En cuanto a la relación entre las tablas sería de muchos a muchos ya que un jugador puede jugar entre 0 y N partidas y una partida es jugada por 2 jugadores.



Aunque para la aplicación que se va a construir ni siquiera es necesario relacionar las tablas, porque nunca se va a necesitar cruzar datos entre ellas, en vista a escalar la

aplicación en un futuro y necesitar más datos lo correcto sería relacionarlas y crear una tabla intermedia ya que la relación es de muchos a muchos, por lo que quedaría de la siguiente manera:



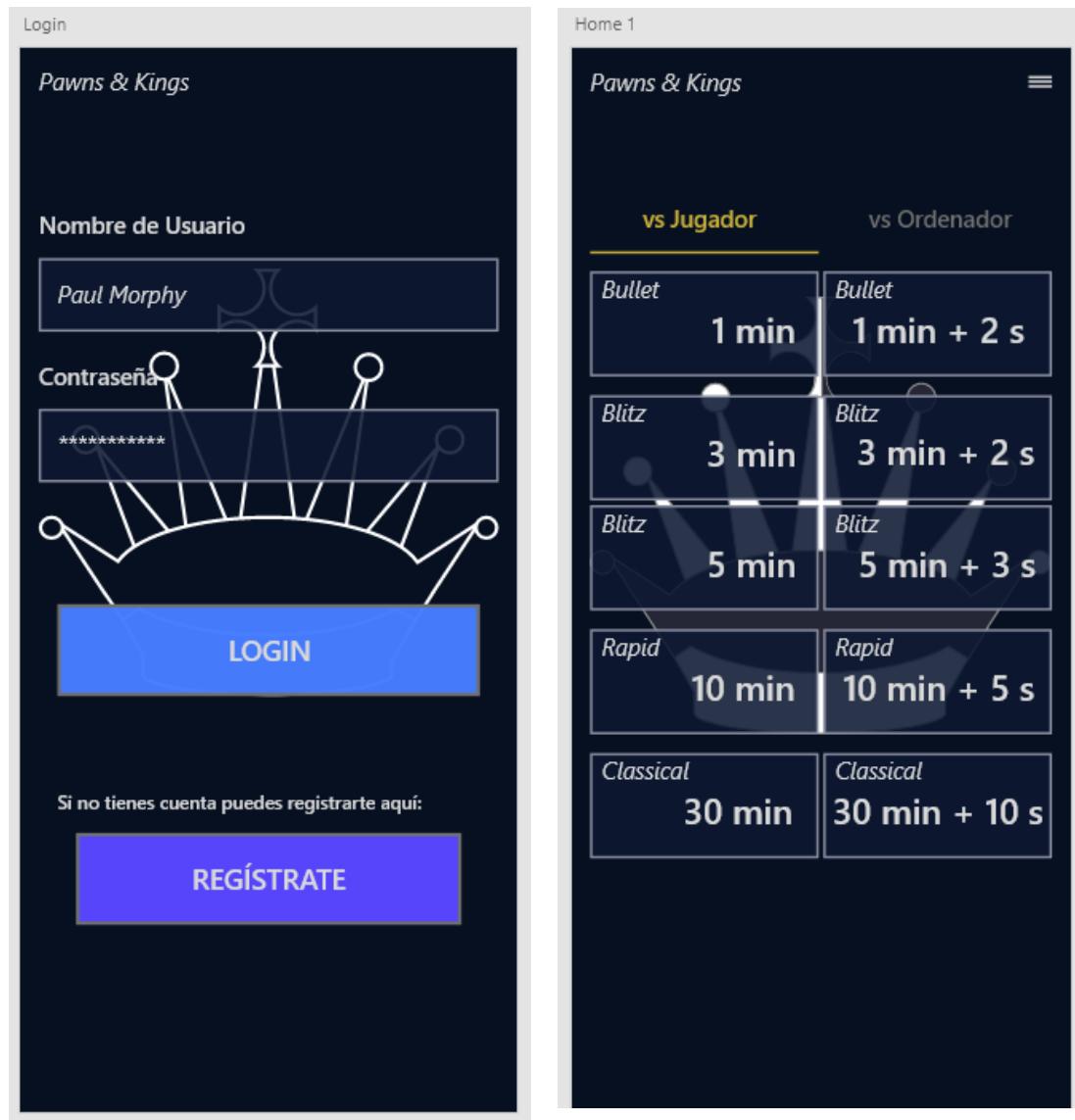
La tabla intermedia tendría como clave foránea los id del usuario y de la partida y con qué color jugó ese usuario.

5.4 Prototipo gráfico

El diseño gráfico se ha realizado con el programa Adobe XD. Se ha buscado un diseño intuitivo y directo para el usuario y con colores oscuros que permitan la concentración en el juego sin molestar la vista.

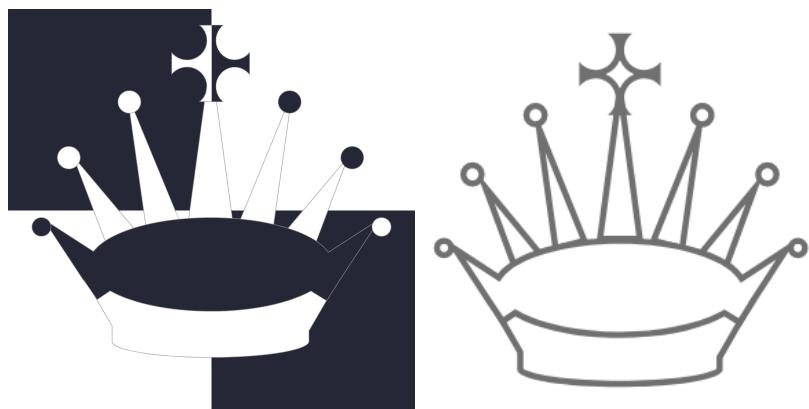
5.4.1 Pantallas principales

A continuación se presenta el diseño de las pantallas principales, no están todas pero el resto son variaciones basadas en ellas.



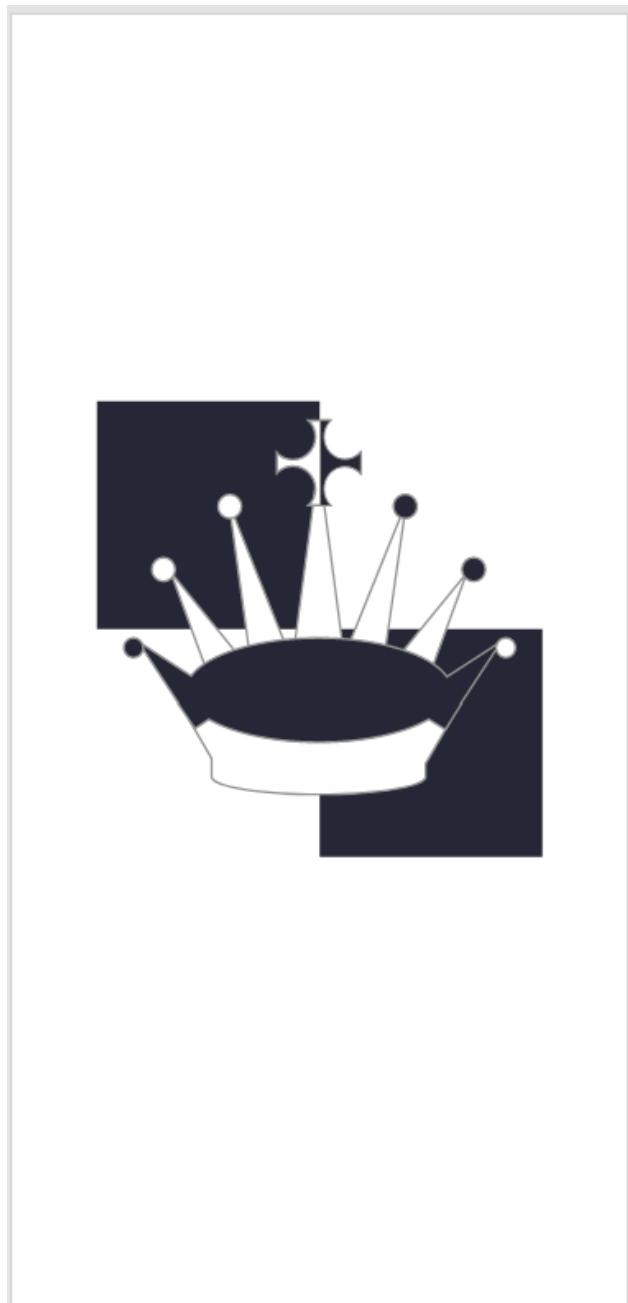
5.4.2 Logo

El logo ha sido diseñado en base a una pieza de ajedrez e intentando conseguir un estilo icónico. La herramienta utilizada ha sido también Adobe XD.



5.4.3 Splash Screen

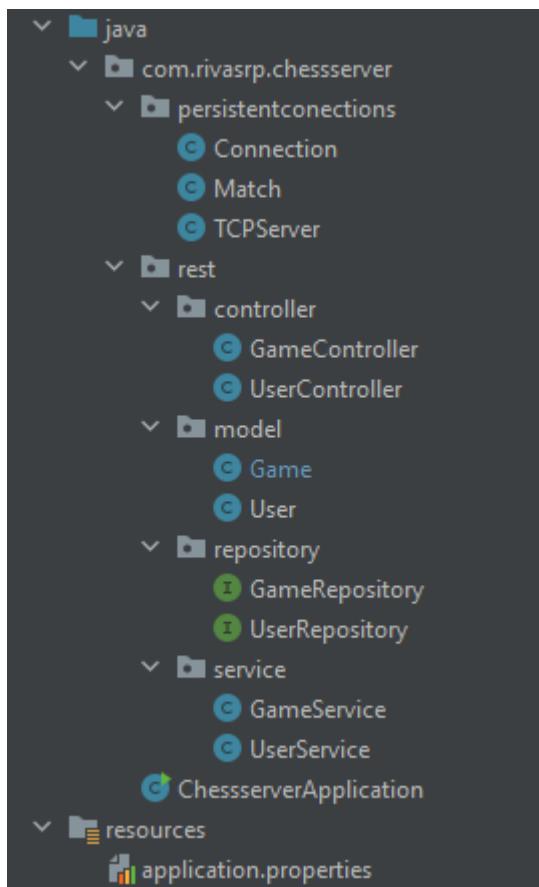
Como pantalla de presentación de la aplicación simplemente se mostrará el logo con un fondo blanco durante unos 2 segundos.



6. Implementación

6.1 Servidor

Organización de paquetes y clases que conforman el servidor:



- Paquete resources

Este paquete contiene un único fichero de Spring Boot en que se especifican una serie de datos correspondientes a la base de datos, para que Spring pueda crear las tablas, añadir campos etc. según se vaya modificando el código.

```
spring.jpa.database=MYSQL
spring.jpa.show-sql=true
spring.jpa..hibernate.ddl-auto=update
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/chessdb
spring.datasource.username=root
spring.datasource.password=
```

- **Paquete rest**

Es el paquete que contiene la API REST con una arquitectura típica de esta. Con la utilización de las anotaciones de Spring Boot se construye de forma muy fácil y rápida sin tener que tocar nada de sql.

Inicio de la clase controladora de Usuario:

```
@RestController
@RequestMapping("user")
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping("/getall")
    public List<User> getall() { return userService.getAllUsers(); }
```

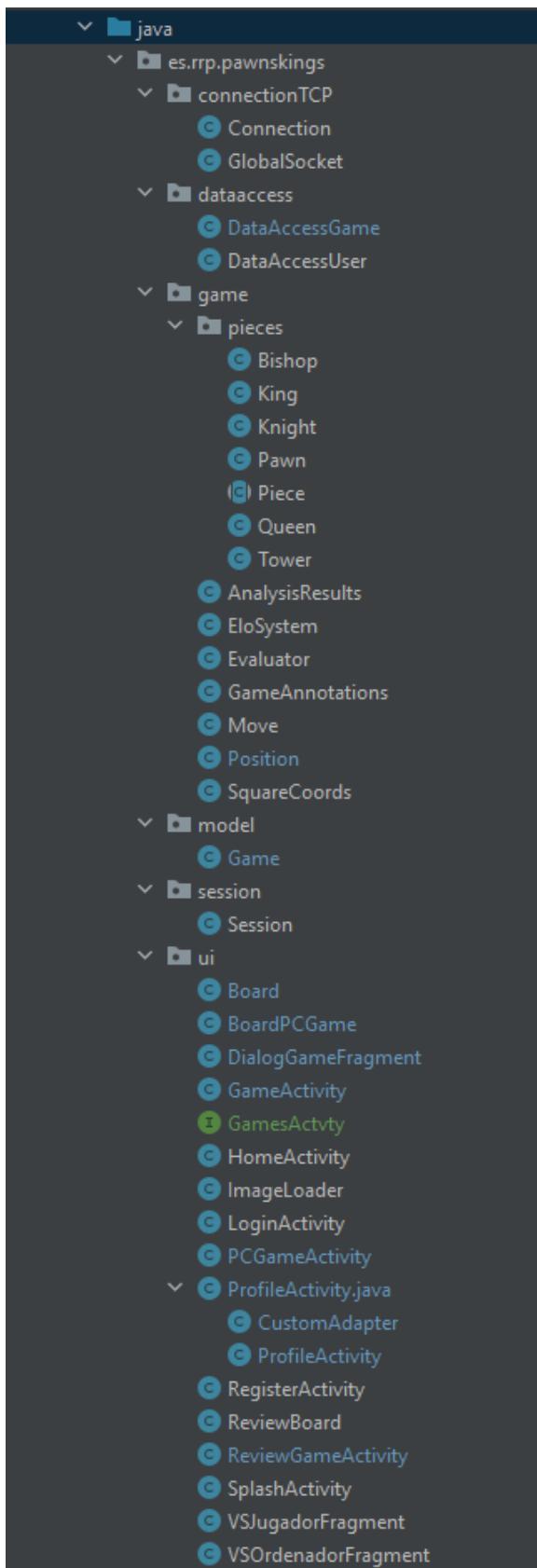
- **Paquete persistentconections**

Aquí se encuentra lo necesario para crear conexiones persistentes entre el servidor y los clientes. Desde la clase principal ChessserverApplication se lanza un hilo independiente con un objeto de TCPServer cuando se lanza la aplicación, debe ser en otro hilo porque el hilo principal se encargará de escuchar las llamadas HTTP, mientras que este debe estar escuchando las conexiones socket entrantes. El objeto TCPServer entonces crea un objeto Connection que será el que se bloquee esperando conexiones entrantes.

Siempre que haya dos o más conexiones que hayan escogido el mismo tiempo de juego entonces se creará un objeto Match con esos dos socket en un hilo nuevo donde transcurrirá el traspaso de información durante una partida entre esos dos clientes. Cabe decir que el método que guarda el socket en el array a la espera de un match y el que comprueba si hay suficientes conexiones para crear un match y lo crea cuando es posible deben ejecutarse en dos hilos diferentes pero coordinándose para poder acceder de manera ordenada a los arrays de sockets.

6.2 Cliente

Organización de paquetes y clases que conforman el cliente:



6.2.1 Paquete game

Este es el código más importante y más complejo de la aplicación, corresponde al motor de ajedrez y todo lo que tiene que ver con las reglas del juego.

Dentro encontramos el paquete pieces, cuyas clases se encargan principalmente de encontrar las jugadas posibles de cada una de las piezas a una posición dada.

También encontramos una clase muy importante, la cual es la piedra angular del motor, la clase Position, que almacena en un array de dos dimensiones la posición actual de una partida y contiene numerosos métodos que aplican las reglas del juego para esa posición o para otra posición dada en forma de array de dos dimensiones. Estos métodos son:

- **setStartPosition()**: Coloca la posición inicial de ajedrez.
- **getPossibleWhiteMoves()**: Encuentra los movimientos posibles para cada pieza blanca apoyándose en instancias de pieces.
- **getPossibleBlacMoves()**: Encuentra los movimientos posibles para cada pieza negra apoyándose en instancias de pieces.
- **controlledSquares()**: Encuentra todas las casillas controladas por las piezas de un color para una posición dada.
- **isKingInCheck()**: Comprueba si el rey del color que es el turno está en jaque en una posición dada, utilizando el método controlledSquares() para saber que casillas amenazan las piezas del color contrario.
- **getKingPosition()**: Obtiene la casilla donde está el rey del color que es el turno en una posición dada.
- **wouldBeCheck()**: Comprueba si de entre los movimientos posibles de un color alguno haría que su propio rey estuviera en jaque y si es así lo elimina de los movimientos posibles.
- **promoteToQueen()**: Comprueba tras un movimiento si es un peón el que movió y si ha llegado a promocionar.
- **ifItsCastle()**: comprueba si el movimiento fué enroque, y si lo fue coloca la torre detrás del rey en su posición tras el enroque.
- **move()**: Coloca una pieza en la casilla indicada, llama a promoteToQueen() y a ifItsCastle() para hacer las comprobaciones pertinentes tras el movimiento, establece la anotación del movimiento y llama a findLegalMoves() descrito a continuación.
- **findLegalMoves()**: Llama a getPossibleWhiteMoves() y getPossibleBlacMoves() para saber cuales son los movimientos posibles para el turno y con esto luego llama a wouldBeCheck(), tras esto comprueba hay 0 movimientos posibles para saber si es jaque mate o ahogado.

Adicionalmente contiene otras variaciones de estos métodos utilizados para el análisis de posiciones y la reproducción de partidas.

Las clases Squarecoords y Move sirven únicamente para almacenar coordenadas de las casillas. Squarecoords almacena una coordenada x y otra y,y a su vez Move almacena dos Squarecoords, siendo casilla de partida y casilla final.

La clase evaluator es la encargada de aplicar el algoritmo de búsqueda y evaluar las posiciones resultantes asignandole un valor numérico. El algoritmo minimax con poda alpha-beta queda así en código Java:

```

public AnalysisResults evaluateAlphaBeta(char[][] position, int depth, boolean colorTurn, float alpha, float beta) {
    if (depth == 0) {
        float evaluation = evaluateMaterial(position);
        AnalysisResults analysisResults = new AnalysisResults(evaluation);
        return analysisResults;
    }

    HashMap<SquareCoords, ArrayList<SquareCoords>> legalMoves = positionObj.getLegalMovesFor(position, colorTurn);
    SquareCoords startPosition = new SquareCoords( row: 0, column: 0 );
    SquareCoords finalPosition = new SquareCoords( row: 0, column: 0 );

    if (colorTurn) {
        boolean cutoff = false;
        for (SquareCoords actualPosition : legalMoves.keySet()) {
            for (SquareCoords square : legalMoves.get(actualPosition)) {

                char[][] clonePosition = copyPosition(position);
                positionObj.rawMove(clonePosition, actualPosition.getRow(), actualPosition.getColumn(), square.getRow(), square.getColumn());
                AnalysisResults analysis = evaluateAlphaBeta(clonePosition, depth: depth - 1, colorTurn: false, alpha, beta);
                if (analysis.getEvaluation() > alpha) {
                    alpha = analysis.getEvaluation();
                    startPosition = actualPosition;
                    finalPosition = square;
                    if (alpha >= beta) {
                        cutoff = true;
                        break;
                    }
                }
            }
            if (cutoff) { break; }
        }
        AnalysisResults analysisResults = new AnalysisResults(alpha, startPosition, finalPosition);
        return analysisResults;
    }
    else {
        boolean cutoff = false;
        for (SquareCoords actualPosition : legalMoves.keySet()) {
            for (SquareCoords square : legalMoves.get(actualPosition)) {

                char[][] clonePosition = copyPosition(position);
                positionObj.rawMove(clonePosition, actualPosition.getRow(), actualPosition.getColumn(), square.getRow(), square.getColumn());
                AnalysisResults analysis = evaluateAlphaBeta(clonePosition, depth: depth - 1, colorTurn: true, alpha, beta);
                if (analysis.getEvaluation() < beta) {
                    beta = analysis.getEvaluation();
                    startPosition = actualPosition;
                    finalPosition = square;
                    if (alpha >= beta) {
                        cutoff = true;
                        break;
                    }
                }
            }
            if (cutoff) { break; }
        }
        AnalysisResults analysisResults = new AnalysisResults(beta, startPosition, finalPosition);
        return analysisResults;
    }
}

```

La clase AnalysisResult contiene dos datos, un valor numérico de la evaluación y el movimiento que lleva a esa evaluación.

La clase GameAnnotations cumple varias funciones, preparar la anotaciones para guardar la partida en la base de datos en forma de string, preparar las anotaciones dándole el formato de notación algebraica para presentarla en pantalla durante una partida y también poder buscar en las jugadas anteriores durante la partida que se movió, por ejemplo para saber si el rey y una torre se movieron anteriormente para saber si se pueden enrocar o no.

Por último la clase EloSystem realiza los cálculos necesarios para establecer la nueva puntuación elo a un jugador tras una partida.

6.2.2 Paquete connectionTCP

Está formado por las clases Connection, encargada de crear una conexión socket con el servidor y mandarle los datos del jugador y del tiempo de la partida y GlobalSocket que es la clase utilizada por connection para almacenar un socket e implementado con un patrón de diseño singleton.

6.2.3 Paquete dataaccess

Encargado de las conexiones HTTP para el envío y recuperación de datos de la base de datos a través de la API del servidor.

6.2.4 Paquete session

Contiene la clase Session encargada de guardar o borrar en el sistema operativo android los datos de la sesión del usuario para poder mantener la sesión abierta si la aplicación se cierra.

6.2.4 Paquete ui

Contiene las clases encargadas de la interfaz de usuario, sus componentes y su funcionamiento tales como activities y fragments. Cabe destacar las clases Board y BoardPCGame y ReviewBoard que son componentes heredados de View y Canvas donde se dibuja el tablero de juego, contiene los eventos necesarios para el movimiento de las piezas y es el encargado de relacionarse con la clase Position para enviarle las jugadas hechas por el usuario y dibujar la posición resultante. También es el encargado de enviar las jugadas al servidor mediante el socket abierto al comenzar la partida.

7. Conclusiones

En cuanto a los objetivos principales propuestos para el desarrollo de esta aplicación:

- Se ha creado una aplicación capaz de registrar usuarios y guardar y recuperar los datos necesarios en una base de datos.
- Se ha conseguido que los usuarios puedan jugar online desde su smartphone con otros usuarios de la aplicación en tiempo real.
- Permite al usuario revisar partidas jugadas por él en el pasado.
- Ofrece un sistema de puntuación de jugadores basado en el sistema oficial elo.
- Se ha conseguido crear un motor de ajedrez capaz de analizar las posiciones y evaluar cual es la mejor jugada.
- Ofrece la posibilidad de jugar contra el motor de ajedrez programado.

Como aplicación, más allá de los objetivos propuestos, tiene los pilares básicos para ser una aplicación de entretenimiento para jugadores de ajedrez, aunque muy sencilla dadas las grandes plataformas con innumerables opciones que se ofrecen hoy en día, por lo que se le deberían añadir algunas funcionalidades más y mejorar las que ya hay antes de ofrecerla al público esperando competir con otras aplicaciones más grandes.

Una de las cosas más interesantes de este proyecto es que se ha explorado el funcionamiento de los motores de ajedrez y como construir uno desde cero, y aunque el implementado en esta aplicación tiene una eficiencia en el juego muy mejorable, ya que construir uno realmente eficiente es un proceso largo y muy complejo, se han aprendido las bases con las que los programadores han estado lidiando desde que la informática entró en el mundo del ajedrez.

8. Bibliografía

https://es.wikipedia.org/wiki/Ajedrez_por_computadora#Cronolog%C3%A3a_de_las_computadoras_de_ajedrez

<https://artsandculture.google.com/story/CQURBJHKlccLIA?hl=es>

<https://neelshelar.com/minimax-algorithm-alpha-beta-pruning-adversarial-search/>

https://addi.ehu.es/bitstream/handle/10810/43118/Memoria_TFG_Daniel_Ochoa.pdf?sequence=1

https://es.wikipedia.org/wiki/Poda_alfa-beta

<https://www.youtube.com/watch?v=otGXjZZRQv8>

<https://www.guapaweb.es/tablas-de-finales-en-ajedrez-todo-lo-que-hay-que-saber/>