PRODUCT BACKLOG

Siguiendo el modelo de metodología MoSCoW, en este documento las diferentes historias de usuario implementadas en el programa en desarrollo se han dividido en varias categorías en función de su prioridad:

Elementos imprescindibles (Must have):

En esta categoría se encuentran los requisitos mínimos que nuestro software ha de cumplir. Obviar cualquiera de ellos supondría poner en riesgo la ejecución del programa.

Historia	Estado	Duración
Tablero	**Terminado**	**1 Sprint (3/3/20-17/3/20**
Crear fichas	**Terminado**	**1 Sprint (3/3/20-17/3/20)**
Mover pieza	**Terminado**	**2 Sprints (17/3/20-14/4/20)**
Posición piezas iniciales	**Terminado**	**1 Sprint (3/3/20-17/3/20)**
Comer	**Terminado**	**1 Sprint (31/3/20-14/4/20)**
Muerte rey	**Terminado**	**1 Sprint (14/04-28/04)**
Jugar online	**Terminado**	**2 Sprints (28/04-26/05)**
Programación de la IA	**Terminado**	**1 Sprint (28/04-12/05)**
Posibles Movimientos	**Terminado**	**1 Sprint (17/3/20-31/3/20)**
Interfaz elegir modo	**Terminado**	**1 Sprint (28/04-12/05)**

Tablero

Consiste en una matriz de casillas 8x8. Cada casilla contará con métodos que permitirán editar si la misma está ocupada y qué pieza ocupa dicho lugar y su color, además de dos variables de posición (x,y). Dentro del tablero se implementa el movimiento de las piezas.

Crear fichas

Cada tipo de pieza del juego cuenta con su propia clase, la cual contiene atributos como el color o la posición. Al inicializar el tablero, se crearán las piezas necesarias de cada tipo llamando a sus respectivas clases, y se especificarán su color y posición.

Mover pieza

El propio método se implementa en tablero. Durante la partida, si un jugador toca una de sus piezas y acto seguido selecciona una casilla a la que la misma pueda desplazarse, ha de hacerlo.

Posición piezas iniciales

Al inicializarse el tablero, cada pieza ha de aparecer en su respectiva posición inicial. En el propio main se llama al método initVistaTablero() de tablero, el cual a su vez hace un update de todas las casillas para situar las piezas.

Comer

En el caso de que una pieza se sitúe en la misma casilla que otra del color opuesto, la segunda debe desaparecer. En el caso de existir un banquillo, la ficha eliminada ha de aparecer en él.

Muerte rey

Cuando uno de los dos reyes muera y finalice la partida, el tablero quedará inactivo, pero no los botones de la barra de herramientas. De esta forma, los jugadores podrán optar por jugar una nueva partida o salir del juego.

Jugar online

El programa contará con la aplicación Servidor y la aplicación Cliente. Introduciendo una serie de datos como la dirección IP, dos jugadores serán capaces de enfrentarse de manera online. Mediante la conexión al servidor, se enviarán/recibirán los movimientos realizados en cada turno.

Programación de la IA

El jugador se enfrenta a una Inteligencia Artificial con tres niveles posibles, un nivel fácil donde los movimientos de la IA son aleatorios, un nivel intermedio donde la IA hace también movimientos aleatorios pero detecta cuando está en jaque el rey contrario y lo come, y por último, un nivel más difícil donde la IA lee los movimientos de un fichero que elige el jugador y en caso de no poder realizar ninguno de los movimientos programados para ese turno, realizará uno aleatorio de nivel intermedio.

Posibles movimientos

Un método implementado en cada una de las clases de las piezas ha de devolver todas las casillas del tablero a las que la pieza en cuestión podría desplazarse en el turno. Para ello, se hace un recorrido del tablero en todas las direcciones hacia las que la pieza podría desplazarse, con su casilla como posición inicial.

Interfaz elegir modo

Puesto que existe la posibilidad de jugar contra una IA o contra otro jugador (bien presencialmente o bien online), es necesaria una interfaz que, nada más se lance el programa, nos haga escoger la modalidad de juego.

Elementos importantes (Should have):

Contiene funcionalidades altamente deseables. Son requisitos importantes pero no estrictamente necesarios.

Historia	Estado	Duración
Reiniciar partida	**Terminado**	**2 Sprints (28/04-26/05)**
Elegir blancas y negras	**Terminado**	**2 Sprints (03/03-17/03)**
Jaque	**Terminado**	**1 Sprint (28/04-12/05)**
Movimiento inicial peones	**Terminado**	**1 Sprint (17/03-31/03)**
Coronar	**Terminado**	**1 Sprint (14/04-28/04)**
Pedir y aceptar tablas	**Terminado**	**1 Sprint (31/3/20-14/4/20)**
Rendirse	**Terminado**	**2 Sprint (28/04-26/05)**
Toolbar	**Terminado**	**1 Sprint (14/04-28/04)**
Vista de turno	**Terminado**	**1 Sprint (14/04-28/04)**
Salto del peón	**Terminado**	**1 Sprint (14/04-28/04)**
Enroque	**Terminado**	**1 Sprint (14/04-28/04)**
Colorear casillas posibles	**Terminado**	**1 Sprint (14/4/20-28/4/20)**
Casillas amenazadas	**Terminado**	**1 Sprint (28/04-12/05)**

Turno	Terminado	1 Sprint (28/04-12/05)

Reiniciar Partida

En la barra de herramientas estará disponible en todo momento un botón que permita a los jugadores reiniciar la partida y comenzar a jugar de nuevo.

Elegir blancas y negras

Implementación de una interfaz que permita a los jugadores elegir su color, bien ellos mismos o bien mediante un random.

Jaque

Se produce si un rey está amenazado. Tras mover una pieza, se revisan todos sus posibles movimientos, y en el caso de que en el array de possibleMoves se encuentre la casilla del rey del color opuesto, se producirá jaque.

Movimiento inicial peones

Si un peón no ha sido movido antes, el jugador puede escoger entre desplazarlo una o dos casillas hacia delante. Una vez se haya movido, solo podrá desplazarse una casilla.

Coronar

En el caso de que un peón llegue hasta la respectiva última fila del tablero (fila 8 para las piezas blancas, fila 1 para las negras), el jugador debe poder elegir una figura en la que transformarlo. Dichas figuras serían un alfil, un caballo, una torre o una dama.

Pedir y aceptar tablas

Durante todo el juego será visible un botón “Pedir tablas” que los jugadores podrán pulsar. Su contrincante ha de aceptarlas para que estas se produzcan. En caso contrario, no habrá tablas.

Rendirse

Si bien existen las tablas y la opción de salir del juego, es necesaria una para rendirse. En la barra de herramientas habrá un botón que permitirá que un jugador se rinda, otorgando la victoria a su contrincante.

Toolbar

Consiste en una barra de herramientas siempre visible en la pantalla, la cual contiene diferentes botones a los que los jugadores tendrán acceso, como reiniciar partida o pedir tablas.

Vista de turno

Debe mostrarse en la ventana el turno del jugador, tanto su nombre como su color.

Salto del peón

La clase Peón cuenta con una variable booleana llamada *hasMoved*, la cual se mantendrá en false hasta que un peón abandone su posición inicial. La posibilidad de desplazarse dos casillas hacia delante solo estará disponible mientras *hasMoved* se mantenga en false.

Enroque

En el caso de que el rey y una torre de un jugador aún no se hayan movido, puede escoger desplazar el rey dos casillas a cualquiera de sus lados, siempre y cuando estén vacías, y situar la torre en la casilla adyacente al rey por el lado contrario, siempre y cuando esté libre también. En el caso de que dichas casillas no estén libres, no se podrá realizar el enroque.

Colorear casillas posibles

Al pulsar un jugador una de sus piezas en su turno, deben colorearse todas las casillas libres a las que dicha pieza podría desplazarse. Esto se hace mediante el método “possibleMoves” que cada pieza tiene implementada, que devuelve un array con las casillas disponibles.

Casillas amenazadas

Método que devuelve las casillas, tanto vacías como ocupadas por piezas del jugador del turno, que el jugador adversario podría ocupar/comer en su siguiente turno. En el caso del peón, solo serían las diagonales.

Turno

Para controlar el cambio de turno durante el desarrollo del juego, se ha de implementar un enumerado el cual contendrá “Blanco” y “Negro”.

Elementos interesantes(Could have) :

También pueden denominarse como *nice to have*. Esta categoría está compuesta por las funcionalidades que podrían añadirse en el caso de haber tiempo. Son requisitos que podrían valorarse en el caso de que no supongan un esfuerzo extra.

Historia	Estado	Duración
Mensajes de error	**Terminado**	**1 Sprint (14/04-28/04)**
Reglas Básicas	**Terminado**	**1 Sprint (12/05-26/05)**
Banquillo	**Terminado**	**1 Sprint (12/05-26/05)**
Historial de jugadas	**Terminado**	**1 Sprint (14/04-28/04)**
Salir del juego	**Terminado**	**1 Sprint (31/3/20-14/4/20)**

Mensajes de error

Cada vez que se intente hacer un movimiento imposible, el programa ha de indicarlo mediante mensajes de error.

Reglas básicas

Implementación de un botón “Reglas” que muestre el reglamento básico del juego al ser pulsado.

Banquillo

Lista de piezas “comidas” durante el juego. Se divide en dos listas, la de las piezas blancas y la de las negras.

Historial de jugadas

Los movimientos quedan registrados en un historial junto al tablero, especificando el color del jugador, la pieza, y las casillas de origen y destino.

Salir del juego

Consiste en la implementación de un botón “Exit Game” que ha de estar siempre disponible y cierra el programa al ser pulsado.

Lista de requisitos en orden de prioridad:

1. Debe ser funcional

Nuestro cliente requiere un juego funcional, por lo que hemos desarrollado diferentes historias de usuario que nos permitan cumplir con dicho requerimiento.

Desde un principio determinamos qué apartados íbamos a realizar y cuáles se descartaban o se dejaban para realizar en el último momento como extra.
Para poder tener un ajedrez jugable, necesitábamos desde un inicio tener una base, por eso comenzamos a programar apartados como el tablero, el movimiento de las piezas, y que estas pudieran comer, entre otras cosas (*Historia de usuario [1] "Tablero", Historia de usuario [4] "Mover pieza" e Historia de usuario [6] "Comer"*).

Una vez terminada la parte principal y comprobar que funciona correctamente, hemos ido añadiendo elementos importantes para que el programa se pareciera lo máximo posible al juego original. Decidimos añadir algunas funcionalidades como el historial de jugadas, el cual sirve de guía para mantener un seguimiento de la partida; colorear las casillas posibles, para facilitar el juego y hacerlo más intuitivo; tablas o rendirse. Todas las decisiones fueron tomadas en equipo.

Procuramos revisar frecuentemente fallos que puedan ir surgiendo a medida que se modifica el código, además de ir añadiendo actualizaciones para depurarlo.

2. Debe tener una interfaz intuitiva

Para facilitar el desarrollo de las partidas para los jugadores, se ha implementado una interfaz intuitiva a petición del cliente. Para ello, hemos implementado un tablero de ajedrez (*Historia de usuario [1] "Tablero"*), el cual contiene todas las figuras con sus respectivas imágenes (blancas y negras)(*Historia de usuario [2] "Crear fichas"* e *Historia de usuario [5] "Posición de piezas iniciales"*). Al iniciar una partida, a la derecha de la ventana aparece el nombre del jugador(Historia 23) que deberá mover pieza, al igual que un historial de movimientos (*Historia de usuario [19] "Historial de jugadas"*) que irá creciendo a lo largo de la partida.

En cuanto al desarrollo de un turno y para facilitar el mismo, el jugador, al seleccionar una de sus piezas, verá marcadas todas las casillas a las que podrá desplazarse (*Historia de usuario [20] "Colorear casillas posibles"*). En el caso de seleccionar una casilla a la cual no puede moverse, saltará un mensaje de error (*Historia de usuario [21] "Mensaje de error"*) en la esquina inferior derecha que le explicará el por qué de dicha imposibilidad.

Además, se ha añadido una barra de herramientas (*Historia de usuario [22] "Toolbar"*) la cual contiene una serie de botones que permitirán al jugador reiniciar la partida (*Historia de usuario [30] "Reiniciar partida"*), pedir tablas (*Historia de usuario [14] "Tablas"* e *Historia de usuario [15] "Pedir y aceptar tablas"*) , rendirse (Historia 16) y salir del juego (Historia 32), cada uno con su función bien indicada. También podrá seleccionar que se muestren las reglas básicas del juego(*Historia de usuario [17] "Mostrar reglas básicas"*).

3. Debe tener una IA contra la que jugar

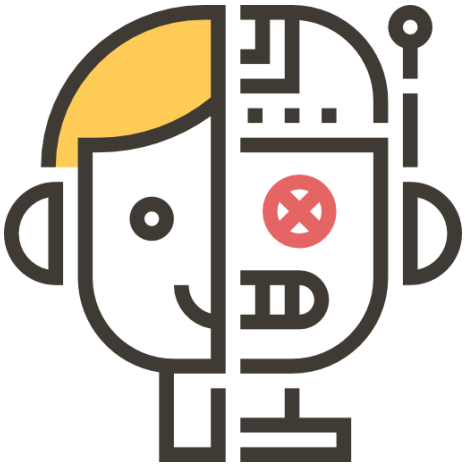
Se supone el caso de que no se quiera jugar contra otro jugador físico, y para ello, el cliente nos pide crear una Inteligencia Artificial (IA) que permita jugar una partida en solitario.
En nuestro caso, tras una reunión de equipo e investigación sobre el tema, hemos decidido realizar la programación de esta IA de dos formas (*Historia de usuario [28] "Programación de la IA"*):

Primero, queremos establecer unas jugadas predeterminadas que se programen manualmente por uno o varios miembros del equipo de desarrollo. De esta forma, se le da un punto de dificultad e inteligencia añadida al jugador rival. Esta clase hereda de la otra clase para que en caso de quedarse sin movimientos posibles pueda realizar movimientos aleatorios y no se acabe la partida sin motivo.

Por otra parte, vamos a incorporar determinados movimientos aleatorios, siempre con figuras que puedan moverse, y a casillas que estén disponibles.

Ambas estrategias van a ser utilizadas simultáneamente en la partida, alternando unas y otras o dependiendo de cuál sea más recomendable para cada situación. Siempre teniendo en cuenta que debe ser el turno del jugador rival, y siempre y cuando pueda realizar algún movimiento (*Historia de usuario [10] "Posibles movimientos"*). Las reglas básicas del ajedrez (*Historia de usuario [17] "Mostrar reglas básicas"*) no se van a ver modificadas en este modo.

Las pruebas del jugador preprogramado consisten en comprobar que lee el fichero JSON, que almacena correctamente los datos, que los movimientos se realizan atendiendo al formato establecido y que en caso de no tener se efectúa un movimiento aleatorio. Por otra parte, las pruebas del jugador aleatorio consisten en comprobar que los movimientos son válidos y que no elige valores fuera del arraylist con las piezas.



4. Debe poder jugarse online

Se supone el caso de que se quiera jugar contra otro jugador pero sin que éste esté físicamente disponible para recurrir a la versión del juego 1vs1. Para esto, se ha implementado una versión online del propio juego (*Historia de Usuario [27] "Jugar online")*.

El programa lleva incorporadas una aplicación servidor y otra cliente, mediante las cuales se enviarán los movimientos realizados en cada turno de un ordenador a otro en la misma red doméstica. Para que esto pueda realizarse con éxito, es necesario conocer las direcciones IP privadas del otro jugador y la dirección IP donde esté alojada la aplicación Servidor.

Bugs encontrados:

- Los peones pueden moverse dos casillas en cualquier momento *(Solucionado)*

Datos a investigar:

Qué investigar	Estado	Duración
JUnit	**Terminado**	**5h**
Aplicación Servidor	**Terminado**	**7h**
Aplicación Cliente	**Terminado**	**5h**
Serialización	**Terminado**	**2h**
JSON	**Terminado**	**2h**