AJEDREZ

Grupo 7
IS

Índice

Estructura

Introducción

Para este proyecto, el equipo ha decidido programar el clásico juego del ajedrez. Aplicando los conocimientos adquiridos en Tecnología de la Programación, el desarrollo de dicho proyecto ha sido llevado a cabo en Java. En el mismo se han implementado una clase Main y Controller, junto con diferentes clases para las piezas e interfaces de acción, además de diversos enumerados. Todo el diseño del programa será explicado más adelante.
Aplicando el modelo Scrum, nuestro equipo cuenta con un Product Owner (Belén García), un Scrum Master (Silvia Egido) y un equipo de desarrollo formado por los 7 integrantes existentes.
El desarrollo del proyecto ha sido llevado a cabo en el entorno de trabajo *Eclipse IDE for Java Developers* 2020-03 y mediante Sprints de dos semanas, tras los cuales tenía lugar una reunión en la que se ponían en común los avances realizados durante esos periodos y se asignaron nuevas tareas a cada uno de los miembros. Sin embargo, y como es evidente, no limitamos la comunicación del equipo a las reuniones. La aportación de nuevas ideas y la resolución de dudas se producían de manera casi diaria. Probablemente, es gracias a esto último que siempre hemos logrado completar las tareas asignadas dentro del tiempo que habíamos establecido, a excepción de un par de casos en los que la carga de trabajo era bastante superior y se necesitó más de un Sprint.
En este documento procedemos a explicar con detalle el diseño del juego, el cual está formado por un controlador y un modelo y las vistas, los cuales interactúan entre sí a través del primero y recurriendo al patrón MVC. El modelo consta de las diferentes clases de las piezas, que heredan de “MovimientoPiezas”, el tablero y las casillas.

Patrones

Introducción

En nuestro proyecto hemos usado varios patrones de entre todos lo que hay, los cuales se encuentran explicados a continuación.

Patrones usados

- **Patrón MVC (Modelo Vista Controlador):** separa la parte de modelo o de lógica de las entradas y salidas de datos.
- **Polimorfismo (piezas):** permite que una misma operación sea aplicada a varios objetos de distintos tipos.
- **Observador:** se usa para notificar el cambio de estado de un objeto determinado a los diferentes objetos que necesitan saber esta información.
- **Creador:** sirve de guía para decir a quién se le asigna la responsabilidad de crear objetos. Su propósito es encontrar un creador que debemos conectar al objeto producido.

Patrón MVC:

La clase Controlador lleva las mecánicas básicas para que el juego pueda ser jugador como el orden de los turnos o las comprobaciones de que cada jugador solo puede mover piezas de su color
El Modelo incluye el tablero con las casillas, las piezas y los jugadores. La parte principal del modelo es la clase Tablero que contiene una matriz de 8x8 Casillas, las cuales contienen las distintas piezas.
La vista consta de dos partes, la vista del tablero que debe ser una matriz de 8x8 y la parte de acciones extra del juego como el historial o botones para rendirse. Ambas partes de la vista de añaden en la clase Tablero y al realizarse cambios en este, se notifica a las vistas para que se actualicen.

Patrón Polimorfismo:

Este patrón se ha usado para generalizar el tratamiento de las piezas a través de la interfaz MovimientoPiezas, la cual obliga a las clases que la implementan el describir sus acciones en base al tipo de pieza que sea, así por ejemplo, para el movimiento cada pieza implementa su propio código para recorrer las casillas a las que se puede mover, su color o si se ha movido alguna vez. Con este patrón se consigue tratar a todas las piezas de igual manera independientemente de su clase y permitiría añadir piezas nuevas de forma sencilla en caso de querer.

Patrón Observador:

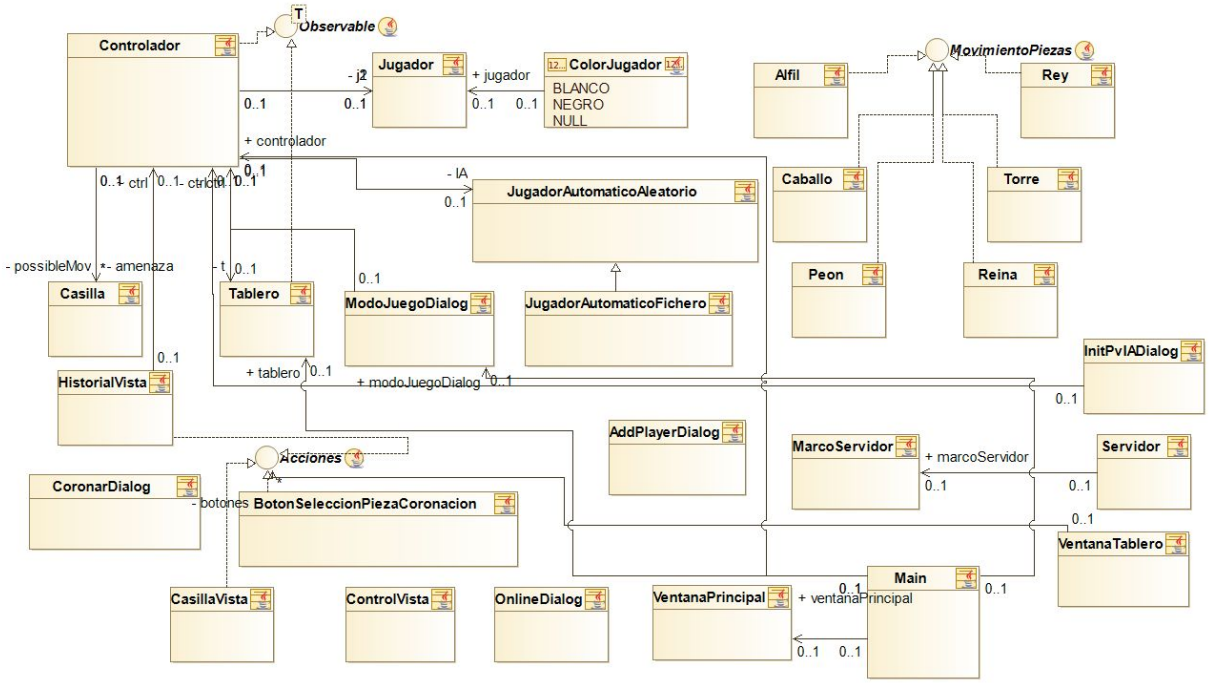
utilizamos este patrón junto al patrón MVC para mejorar y facilitar la interfaz gráfica del juego, registrando así el modelo como observador de las vistas y al generar cambios notificar a todas las posibles vistas para que se actualicen.

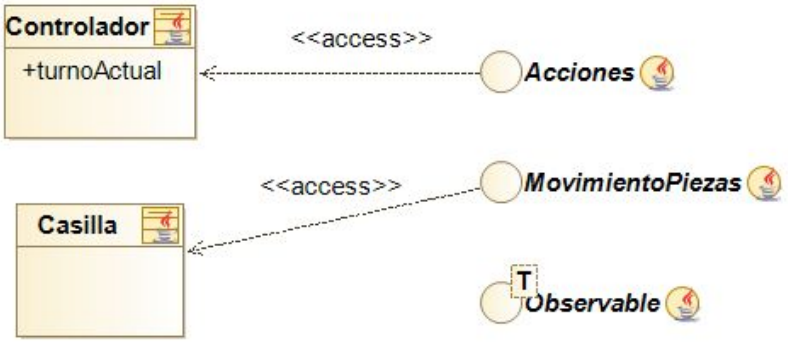
Patrón creador:

Este patrón sirve para asignar responsabilidades, nuestra forma de usarlo ha sido que en el Tablero se crean las Casillas y en estas se crean las Piezas porque necesariamente toda casilla está solo sobre el tablero y toda pieza está solo sobre una casilla, por lo que no tendría sentido crearlas por ejemplo en el Controlador ya que este no debería tener acceso a estas mismas y en el Controlador se crean los jugadores para así poder llevar el turno correctamente.

Clases

Utilizamos distintos tipos de clases en el proyecto: clases anidadas (MarcoServidor) y clases anónimas (Los botones del tablero, Piezas y Casillas), además de enumerados e interfaces (Observable<T>, MovimientoPiezas y Acciones) y clases normales que son el resto.

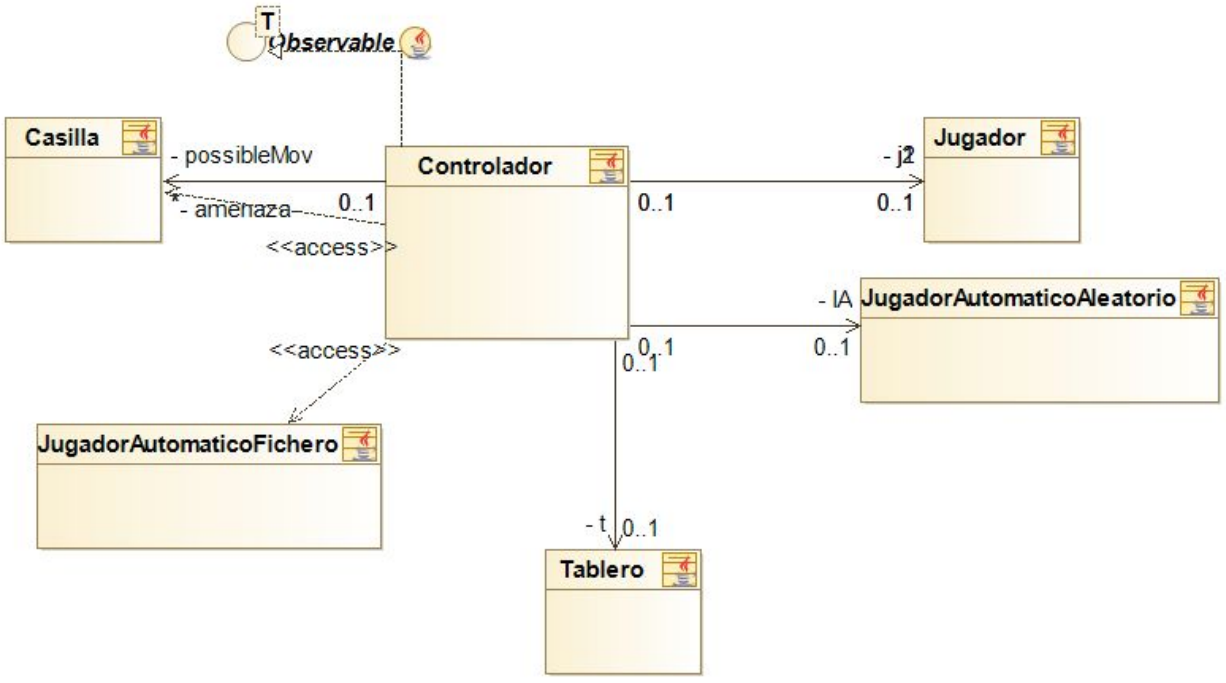




Controlador

El controlador contiene los jugadores además del turno.
Siguiendo el patrón MVC, es el encargado de unir las vistas y el modelo y se encarga de diversos parámetros del funcionamiento del juego como:

- Control de turnos a través del método controlTurnos(int x, int y) el cual dependiendo de si es PvP o PvIA ejecuta una acción u otra.
- Modos de juegos usando los métodos turnoJugadorvsIA(int x, int y), turnosPvP(int x, int y) y run() para el online, donde recibe los datos de los movimientos del otro jugador.
- Crear los jugadores con crearJugadorOnline(String nombre, ColorJugador color), addPlayers(String nombre1, String nombre2) y addPlayerAndIA(String nombreJ, ColorJugador colorJ, ColorJugador colorIA, String mIA).
- Comprobar que los movimientos son correctos, seleccionarPieza(int x, int y) recibe una lista con todas las posibles casillas a las que puede moverse la pieza y moverPieza(int x, int y) comprueba si la casilla de destino es válida y en caso de serla, llama a un método del tablero que mueve la pieza y si está en modo online manda los datos al servidor.



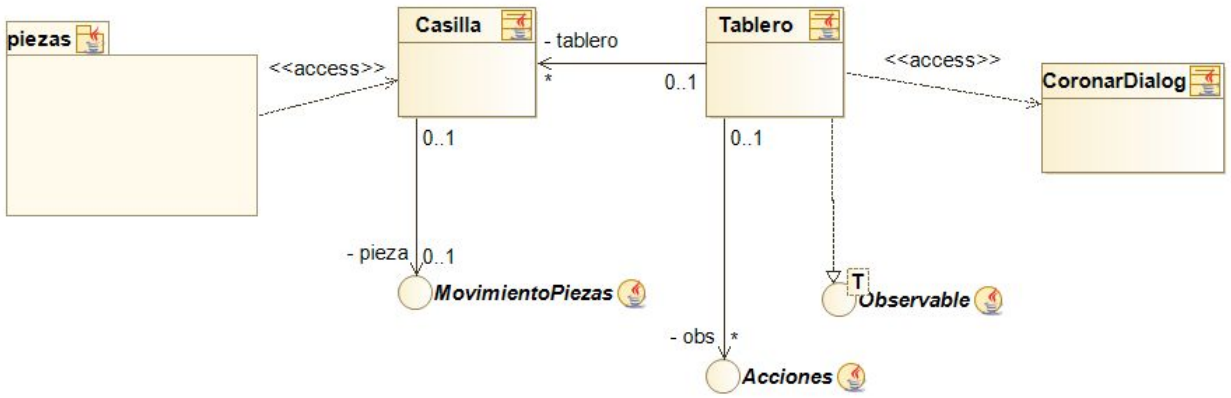
Tablero

Tablero es una clase que implementa a la interfaz Observable<T>. El tablero incluye una matriz de 8x8 Casillas que se inicializa colocando las piezas con la posición inicial del ajedrez y arrayLists con todos los tipos de vista. Esta clase realiza todas las acciones manejadas por el controlador:

Analiza las casillas que tiene para poder ver las casillas amenazadas listaAmenazadas(turnoActual turno).

Controla los movimientos de las piezas (si es enroque, salto del peón, coronación etc) moverPieza(int origenX, int origenY, int destinoX, int destinoY, Controlador c) y las mueve movimiento(int origenX, int origenY, int destinoX, int destinoY).

Llama a actualizar la vista de las casillas updateVistaCasilla(int origenX, int origenY, int destinoX, int destinoY) y actualizarColoresFondocasilla(ArrayList<Casilla> c, boolean marcar).



Casilla

La clase casilla contiene las coordenadas que ocupa dentro del tablero, así como su color y si contiene piezas.

private final int posX;
private final int posY;
private MovimientoPiezas pieza;
private boolean isOcupied;

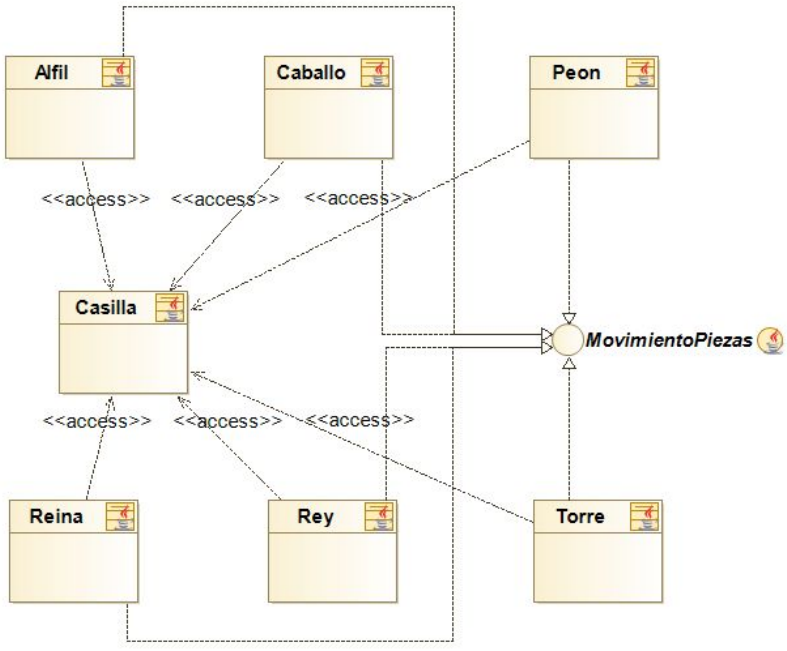
Además esta puede comprobar si la misma está siendo ocupada por alguna pieza y por cuál en concreto. Estas casillas pueden tener cualquiera de las piezas del ajedrez que implementan MovimientoPiezas y se actualizan según los movimientos realizados por el jugador a través de la vista y contienen métodos para notificar a las vistas las actualizaciones tras los movimientos.

Piezas

Todas las piezas implementan a la interfaz MovimientoPiezas que indica los métodos necesarios que deben tener todas las piezas para poder usarse en el tablero, haciendo uso del patrón polimorfismo.
Las piezas tienen como atributo un enumerado ColorJugador con el color de la misma.
Todas las piezas sobrescriben los métodos de la interfaz, y mediante el método de possibleMoves(int iniX, int iniY, Tablero t) se devuelve una lista con todas las casillas a las que puede moverse dicha pieza en función de sus movimientos en el ajedrez (caballo en "L", alfil en diagonal etc).

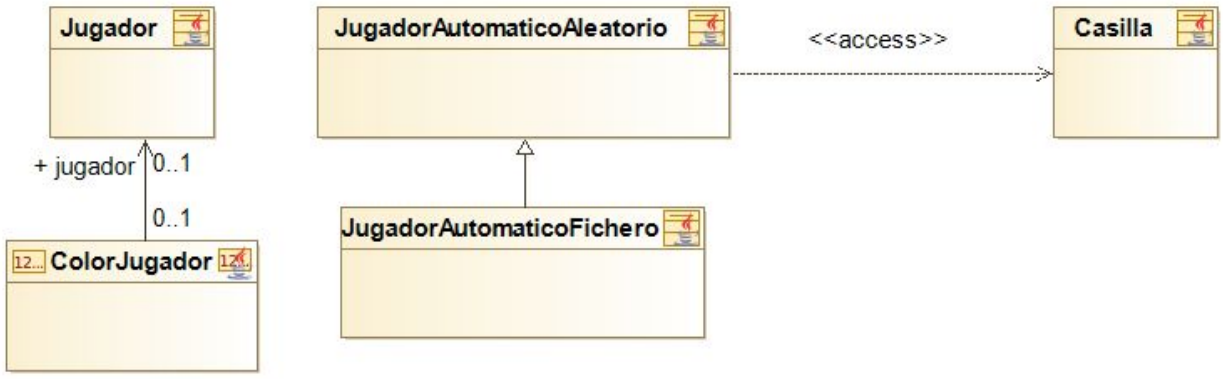
La torre, el peón y el rey tienen además el atributo booleano hasMoved que comprueba si han realizado algún movimiento a lo largo de la partida para poder realizar movimientos especiales como es el salto del peón o el enroque para la torre y el rey
El rey tiene el atributo booleano castling para en caso de haber movido dos casillas para realizar el enroque se informe de esto para completar con el movimiento de la torre y sobrescribe el método esRey() para indicar que esta pieza sufre determinadas condiciones especiales (jaque, enroque).

Todas las piezas están contenidas en las Casillas del juego. y son manipuladas por estas a través del tablero y el controlador.



Jugador

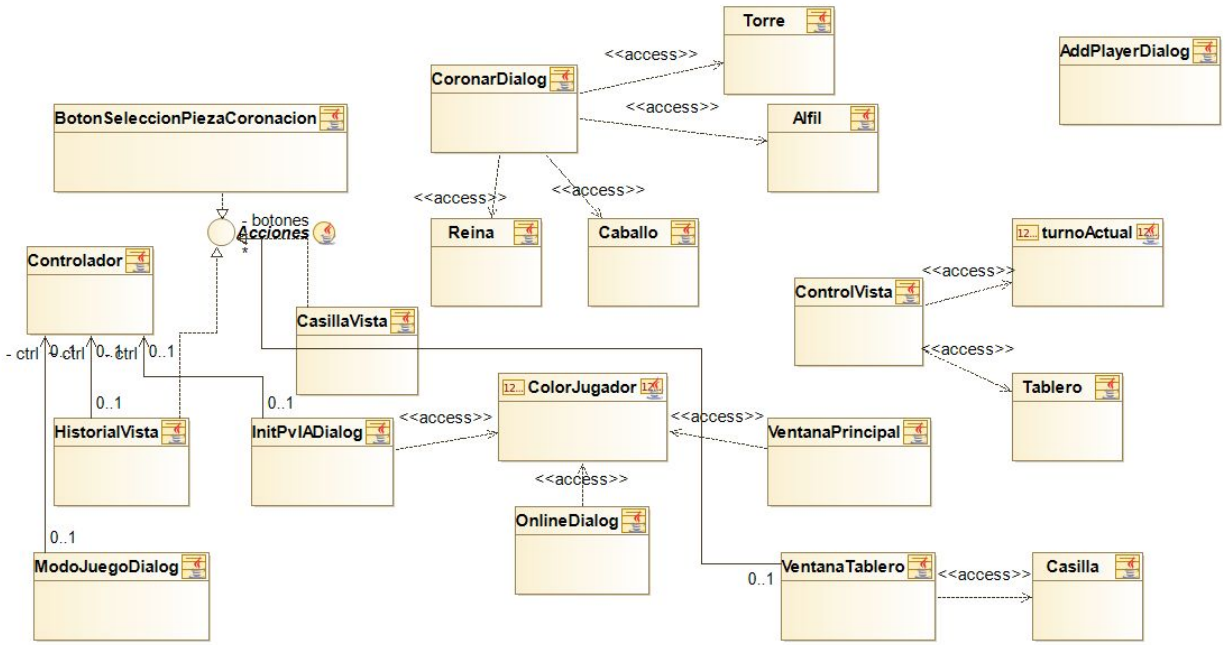
Contiene un atributo que es el nombre del jugador el cual se mostrará en la vista y un enumerado que es el color del jugador, el cual es utilizado por el Controlador para determinar si puede mover la pieza seleccionada en base a si es del mismo color o no para mostrar quién ha ganado la partida.



Vistas

Las parte de vistas está compuesta por un marco principal que está dividido en tres zonas:

- La zona superior está compuesta por la clase ControlVista la cual contiene varios botones para el control de diversas opciones el juego como rendirse, pedir tablas o mostrar las reglas básicas. Algunos de estos botones llaman a ciertas funciones del Controlador que realizan cambios en el juego como rendirse.
- El panel lateral está compuesto por la clase Historial Vista que contiene zonas visuales como el historial de jugadas, a quién le toca jugar, el banquillo de las piezas comidas y un apartado que muestra los posibles errores (Como movimientos imposibles). Esta clase implementa la interfaz Acciones para ser notificada por el Tablero cuando sucedan cambios.
- La zona principal es el tablero, el cual está implementado por la clase VentanaTablero que lleva asociados los botones de las casillas implementados mediante las clases CasillaVista. Al crearse esta función debe mandarse al Tablero mediante el Controlador una matriz de clases que implementen la interfaz Acciones para que al realizar cambios el tablero notifique estos a las vistas.



Excepciones

A la hora de desarrollar este programa de ajedrez hemos empleado las excepciones que vienen incluidas en el lenguaje Java. En el caso del modo de juego online, empleamos las excepciones “IOException” y “ClassNotFoundException” para el flujo de datos entre cliente y servidor. Por otra parte, la excepción “FileNotFoundException” es utilizada en la toma de archivos de movimientos preprogramados en el modo jugador vs IA. Por último, a la hora de realizar los movimientos de las piezas empleamos la excepción “IndexOutOfBounds”.

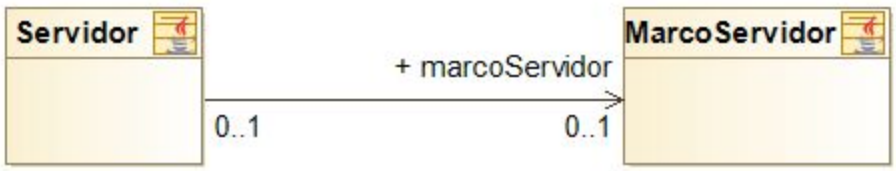
Tras debatirlo, no se ha visto necesario crear excepciones propias para este desarrollo, ya que las proporcionadas por Java cubrían las necesidades del programa.

Modos de Juego

1 vs 1 en el mismo ordenador: Dos jugadores se enfrentan en el mismo ordenador. Primero, escogen sus colores (bien ellos mismos, bien de forma aleatoria), y en cada turno, uno de los jugadores moverá una de sus piezas a alguna casilla disponible (todas ellas se marcarán al seleccionar dicha pieza). Este mismo proceso continuará hasta que uno de los jugadores logre comer al rey de su adversario (o jaque mate si se hace). Durante la partida, siempre se podrá pedir tablas, reiniciar o rendirse.

1 vs IA (aleatorio, aleatorio con jaque o de fichero): 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 , además de movimientos aleatorios, detecta cuándo realiza un jaque procede a comer al rey adversario. Por último, existe un nivel más difícil donde la IA lee los movimientos de un fichero que elige el jugador; si el movimiento no se puede hacer, se leerá el siguiente, y continuará hasta encontrar en el fichero uno posible. En caso de no poder realizar ninguno de los movimientos programados para ese turno, realizará uno aleatorio de nivel intermedio. Este modo de juego transcurre igual que el 1vs1, es decir, el juego acaba cuando uno de los jugadores logre comer al rey de su adversario o el jugador decida rendirse o reiniciar la partida. Durante la partida, el jugador siempre podrá pedir tablas, reiniciar o rendirse.

1 vs 1 Online: Dos jugadores se enfrentan utilizando ordenadores diferentes que se encuentren en la misma red doméstica. Estos jugadores escribirán la IP privada del oponente que a través de un socket enviará los datos de cada movimiento realizado por el jugador al servidor. Éste a su vez los reenvía al oponente donde se actualiza el movimiento. En esta versión no existirán, como tal, turnos. Debido a la falta de tiempo, en esta versión ha de contarse con la “confianza” en que tu adversario solamente toque sus piezas, puesto que es posible mover cualquiera que continúe en el tablero.



JUnit

Introducción

Para realizar pruebas a distintas historias de usuario y así comprobar si están bien ejecutadas, hemos usado la herramienta JUnit. Para empezar, hemos clasificado las historias por nivel de dificultad para agilizar el trabajo. También comprobamos que hay historias que no se pueden realizar con esta herramienta de JUnit debido a que son interfaces que muestran un mensaje o que consisten en elegir entre si o no, como por ejemplo, toolbar, vista de turno o reiniciar partida.

Después fuimos seleccionando las historias más fáciles y primero las hemos "propuesto", es decir, hemos explicado cómo lo haríamos con nuestras palabras. Por último, hemos ido implementando estas historias en el proyecto.

Pruebas realizadas

- Prueba del banquillo.
- Prueba de elección blancas o negras.
- Prueba del enroque.
- Prueba del historial.
- Prueba del IA.
- Prueba de los movimientos básicos de cada pieza.
- Prueba de la muerte del rey.
- Prueba del salto inicial de los peones.
- Prueba de la creación del tablero (incluye creación de piezas).
- Prueba del cambio de turno.
- Prueba de comer de todas las piezas.
- Prueba de casillas amenazadas.
- Prueba del jaque.
- Prueba de movimientos posibles.

Problemas/bugs

- Debido a la falta de tiempo, no ha sido posible introducir el empleo de IPs públicas en la versión online del juego; en su lugar, se emplean privadas. Además, no es posible jugar en este modo usando la misma IP privada.

- Ha resultado imposible realizar ciertas pruebas en JUnit debido a que estas dependen de ciertas interacciones con la interfaz, como es el caso de “coronar”, “mostrar reglas básicas” o “Pedir tablas”.

- Algunas pruebas de JUnit requieren de cambios en la visibilidad de algunos métodos para forzar ciertas situaciones especiales en el tablero (setPieza() debe ser public en vez de protected).

Bibliografía

JUnit

- JUnit 5 User Guide
  https://junit.org/junit5/docs/current/user-guide/

- Casos de prueba
  http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion04-apuntes.html

- Tutorial para le testeo del JUnit
  https://www.vogella.com/tutorials/JUnit/article.html

- Primeras pruebas del JUnit (Video)
  https://www.youtube.com/watch?v=1k22KuD4si0

- Problemas comunes en las pruebas del JUnit (Video)
  https://www.youtube.com/watch?v=EOkoVm3rtNQ

- Testeo con JUnit en Eclipse
  https://www.youtube.com/watch?v=v2F49zLLj-8&t=467s
  https://www.youtube.com/watch?v=I8XXfgF9GSc&t=88s

Programación

- Transparencias asignatura Tecnología de la Programación II.

- Curso Java
  https://www.youtube.com/playlist?list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk

- Estándares de java
  https://docs.oracle.com/javase/7/docs/api/

- Bases para programar en Java
  https://developer.mozilla.org/es/docs/Web/JavaScript

Documentación

- Patrones de diseño, vídeos proporcionados por el profesor de la asignatura de IS2.

- Información extra de patrones
  https://informaticapc.com/patrones-de-diseno/

- Documentación SCRUM
  https://www.scrum.org/resources/what-is-a-product-backlog
  https://proyectosagiles.org/lista-tareas-iteracion-sprint-backlog/

- Bases y normas del ajedrez
  https://thezugzwangblog.com/reglas-del-ajedrez/