Programación
Proyecto 3er Trimestre: Desarrollo de un juego

Fecha de vencimiento: Miercoles, 05 Junio, 2019

Desarrollo de Aplicaciones Web (DAW1)

Carlos Cánovas Servera
Eugenio Navarro Mas

Índice

Carlos Cánovas Servera
Eugenio Navarro Mas

1. Objetivos

El objetivo de este proyecto es desarrollar un programa poniendo en práctica los conocimientos de programación orientada a objetos adquiridos durante este curso.

Al finalizar el proyecto se deberá entregar la documentación sobre el trabajo realizado y se realizará una presentación y demostración del programa desarrollado.

2. Planificación

Nuestro plan de actuación ante este proyecto fue, en primer lugar, hacer una lluvia de ideas y escoger la que más se ajustara a nuestras posibilidades.

Una de las ideas ha sido el desarrollo de un ajedrez, pero sería un proyecto demasiado grande como para poder completarlo en el plazo de tiempo que tenemos, por eso la idea final tomada para este proyecto, es hacer un ajedrez un poco peculiar. Nuestro juego se basará en la distribución y algunas de las reglas del ajedrez (como los movimientos permitidos a las piezas), pero implementaremos reglas diferentes para las condiciones de victoria y derrota.

También queremos aprovechar este proyecto para investigar sobre el desarrollo e integración de interficies gráficas.

Los pasos que seguiremos para desarrollar este proyecto serán los siguientes:

1. Analizar los requisitos principales de nuestro juego.
2. Diseñar y desarrollar su estructura y componentes lógicos (lógica del juego).
3. Diseñar, desarrollar e integrar una interfáz gráfica de usuario (GUI).

3. Análisis

El desarrollo de este juego se realizará usando programación orientada a objetos en Java.

Como queremos integrar en el juego una interfaz gráfica, dividiremos el proyecto en dos partes:

- **Lógica del juego:** tendremos varias clases que representarán los elementos de nuestro juego, como son el tablero de juego y las piezas que interactuarán entre ellas.
- **Interfaz gráfica:** necesitaremos investigar como implementar e integrar una interfaz gráfica al juego que desarrollemos.

Finalmente integraremos las dos partes en un único programa.

3.1. Lógica del juego

Ésta es la parte más importante del programa, es la que describirá todos los elementos básicos del juego y como interactuarán entre ellos.

Si analizamos y abstraemos los elementos que componen un juego de ajedrez podemos decir que tendremos las siguientes clases: **Pieza, Casilla y Tablero**.

3.1.1. Pieza

Representa las diferentes piezas del ajedrez (peón, caballo, alfil...) y necesitaremos sacar de ella otras clases más específicas para definir cada una de las piezas, por lo tanto la clase "Pieza" será una clase abstracta.

Cada pieza se caracterizará por:

- Tipo: peón, caballo, alfil, torre, dama o rey
- Color: blanco o negro.
- Y un método que permita validar sus movimientos válidos.

Para definir el "Color" y "Tipo" de las piezas describiremos dos **tipos enumerados**, que aparte de restringir los valores que pueden tomar las variables, nos permiten implementar métodos específicos para esos tipos y valores.

3.1.2. Casilla

Representará cada una de las casillas que podemos encontrar en un tablero de ajedrez, y se definirá por:

- Sus coordenadas en el tablero.
- Si está vacia u ocupada.
- La pieza que contenga si está ocupada.

3.1.3. Tablero

Representará el tablero de ajedrez de 64 casillas (8 filas x 8 columnas) y en el se ejecutarán los movimientos realizados por los jugadores, por tanto se definirá por:

- Un array de 8x8 casillas.
- Los jugadores.
- Y alguna manera de mover las piezas por el tablero.

3.2. Interfaz gráfica de usuario (GUI)

Necesitaremos investigar como implementar una interfaz gráfica en Java y sobre el paradigma de programación dirigida por eventos.

3.2.1. Interfaz gráfica en Java

Queremos mostrar de forma gráfica en una ventana el tablero de ajedrez, las piezas y un menú de selección con los que los usuarios podrán interactuar.

Para mostrar de forma gráfica usaremos las librerías *javax.swing* y *java.awt* de Java para hecer marcos, paneles, botones, menús, colorear, etc.

3.2.2. Programación dirigida por eventos

La programación dirigida por eventos es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.

Necesitamos que el programa interactúe con los usuarios, de manera que si un usuario selecciona una pieza en el tablero gráfico mediante un click de ratón, el programa debe responder y actuar en consecuencia.

La librería *java.awt* también nos permitirá controlar los eventos que lancen los componentes gráficos como los paneles y botones.

4. Diseño

A continuación mostraremos el diseño de las diferentes clases que desarrollaremos para este juego:

4.1. Lógica de juego

A continuación describiremos como serán las clases que formarán la lógica del juego, para entender mejor que elementos formarán el ajedrez y como interactúan entre ellos.

4.1.1. Clase *Tablero*

Es la clase principal y estará definida por:

- **Atributos**
  - **Tablero** : será un array bidimensional de 8x8 casillas
  - **Jugador activo** : será una variable del tipo "Color" que identificará al jugador que tiene el turno.
  - **Fin** : será un booleano que nos indicará si la partida a finalizado o no.
- **Constructor** : generará un tablero y lo inicializará con las piezas y el jugador activo (en el ajedrez siempre empieza el blanco).
- **Getters y setters**
  - **getCasilla** : nos devolverá una casilla del tablero según las coordenadas pasadas.
  - **getJugadorActivo** : devolverá el jugador activo (el Color de éste).
  - **setJugadorActivo** : pondrá al otro jugador como activo (cambio de Color).
  - **setFin** : modificará el estado a partida terminada.
- **Métodos**
  - **generarTablero** : generará una tablero de 8x8 casillas.
  - **inicializarTablero** : colocará las piezas en el tablero para poder iniciar la partida.
  - **ejecutarJugada** : realizará el movimiento de las piezas por el tablero. Necesitará las casillas origen y destino del movimiento y después ejecutará el movimiento si es posible.

4.1.2. Clase *Casilla*

Definirá las casillas de las que estará compuesto el tablero.

- **Atributos**
  - **Fila** : la coordenada X en el tablero.
  - **Columna** : la coordenada Y en el tablero.
  - **Pieza** : la pieza que ocupa la casilla. Si no está ocupada deberá tener un *null*.
- **Constructor** : generará una casilla vacía (Pieza=*null*) con las coordenadas que se le pasen.
- **Getters y setters**
  - **getFila** : nos devolverá la coordenada X que tiene en el tablero.
  - **getColumna** : nos devolverá la coordenada Y que tiene en el tablero.
  - **getPieza** : devolverá la pieza que ocupa la casilla.
  - **setPieza** : asignará una pieza a la casilla.
- **Métodos**
  - **isOcupada** : devolverá un booleano que indicará si la casilla está ocupada o no.
  - **sacarPieza** : quitará la pieza de la casilla devolviendo la pieza quitada.

4.1.3. Clase *Pieza*

Será una clase abstracta que representará las piezas del ajedrez de forma genérica.

- **Atributos**
  - **Tipo** : el tipo de la pieza (peón, caballo, alfil, torre, dama o rey), será un tipo enumerado.
  - **Color** : el color de la pieza (blanco o negro), será un tipo enumerado.
- **Constructor** : generará una pieza según el tipo y color pasados.
- **Getters y setters**
  - **getTipo** : nos devolverá el tipo de la pieza.
  - **getColor** : nos devolverá el color de la pieza.
- **Métodos**
  - **movimientosValidos** : método abstracto. Devolverá una lista de casillas que contendrá las casillas donde la pieza pueda moverse en el tablero dependiendo de su posición. Cada tipo de pieza tiene una reglas propias a la hora de moverse.
  - **coordenadaValida** : método estático. Devolverá un booleano indicando si la coordenada pasada está dentro del tablero.

4.1.4. Subclases de *Pieza*

Tendremos tantas subclases de "Pieza" como tipos de pieza hay en el ajedrez. Por lo tanto tendremos 6 subclases: **Peón, Caballo, Alfil, Torre, Dama y Rey**.

Todas ellas tendrán el método de **movimientosValidos** que devuelve una lista de casillas que indican donde pueden moverse dependiendo de su posición en el tablero.

Algunas de las piezas siguen unos mismos patrones de movimiento, pero con direcciones o posiciones diferentes. Podemos definir 3 tipos de movimientos:

- **Movimiento lineal** : el Alfil, la Torre y la Dama tienen un tipo de movimiento lineal en varias direcciones.
- **Movimiento fijo** : el Caballo y el Rey se mueven solo a posiciones fijas.
- **Movimiento de peón** : el Peón se caracteriza en que solo puede moverse en un sentido y además su tipo de movimiento depende de varios factores.
  - Si es su primer movimiento puede desplazarse 2 casillas hacia delante.
  - Si se mueve sin atacar, solo podrá hacerlo en la misma columna.
  - Si se mueve atacando, solo podrá hacerlo en diagonal.

  Por esta razón el peón tendrá una variable que indicará si ha realizado su primer movimiento o no.

Para definir los movimientos que hace cada tipo de pieza, declararemos en cada subclase una variable estática que será una lista de coordenadas que indicarán las posiciones o direcciones de los desplazamientos que pueden hacer respecto a su posición actual.

4.1.5. Tipo enumerado *Tipo*

Crearemos este tipo para identificar los tipos de piezas que hay: **Peón, Caballo, Alfil, Torre, Dama y Rey**.

También implementaremos 2 métodos abstractos para cada valor que puede tomar:

- **esRey** : devolverá un booleano indicando si el tipo es el del Rey o no.
- **esPeon** : devolverá un booleano indicando si el tipo es el del Peón o no.

Esto nos servirá para que el código sea más legible.

4.1.6. Tipo enumerado *Color*

Este tipo nos identificará los colores de las piezas y los jugadores.

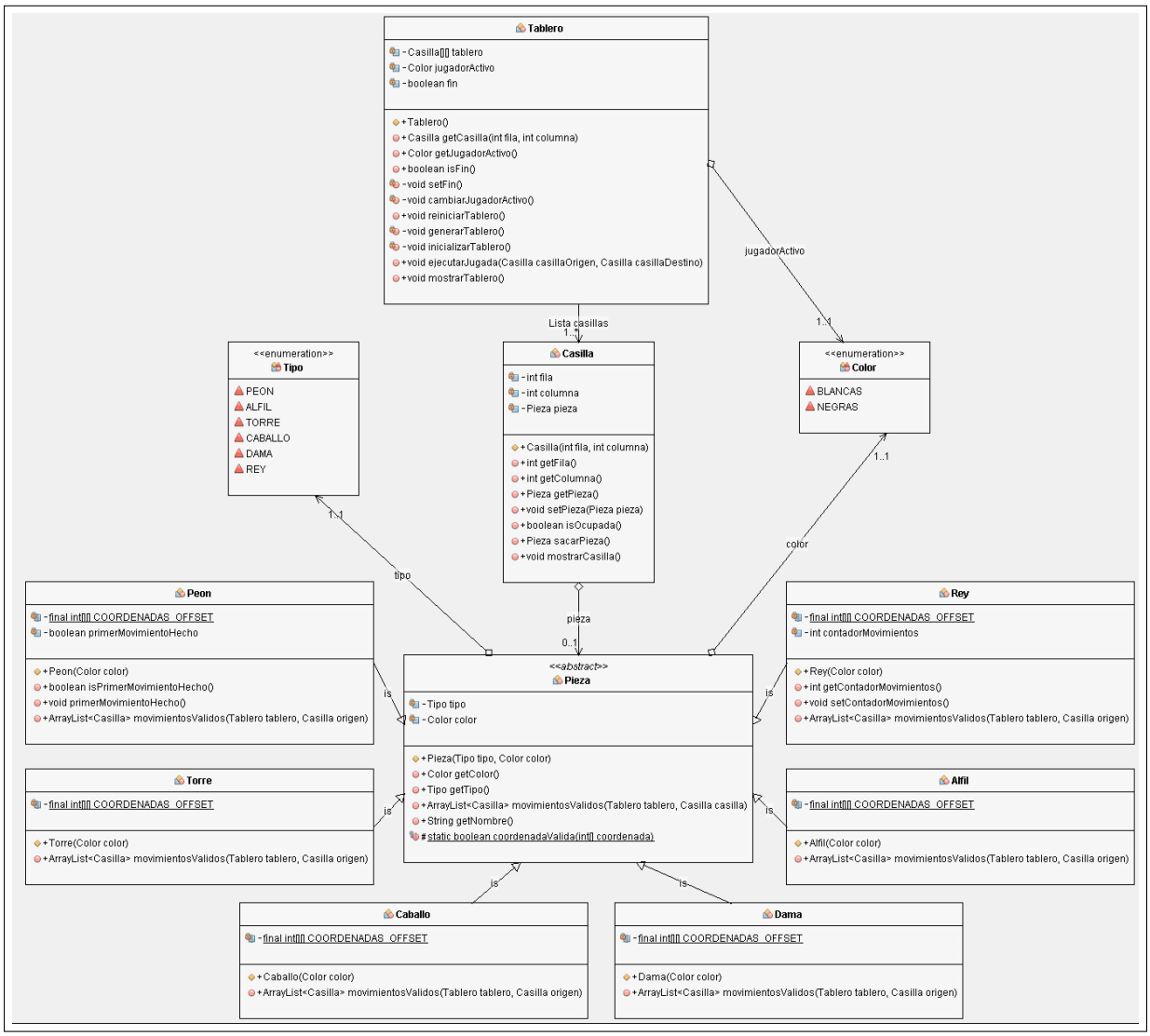
En el ajedrez tenemos 2 colores: **Blanco y Negro**.

También implementaremos 2 métodos abstractos para cada valor que puede tomar:

- **getDireccion** : devolverá 1 o -1 dependiendo del color. Lo usaremos para los peones que solo pueden ir en una dirección pra calcular sus movimientos válidos.
- **cambiarJugador** : devolverá el color contrario. Lo usaremos para cambiar el jugador activo.

4.1.7. Diagrama de clases

A continuación se puede ver un diagrama de clases de la lógica del juego:



4.2. Interfaz gráfica

4.2.1. Diagrama de clases

A continuación describiremos como serán las clases que formarán la interfaz gráfica del juego:

4.2.2. Clase *Ventana*

Esta clase representa la interfaz completa del juego, que contiene una barra de menú, y el tablero de juego.

- **Atributos**
  - **Ventana** : la ventana *JFrame* de la interfaz.
  - **Menú** : la barra de menú *JMenuBar* que tendrá varias pestañas con opciones.
  - **PanelTablero** : el panel que representará el tablero de juego.
  - **Tablero** : la lógica de juego. El tablero visto en los apartados anteriores.

    Además contará con varias variables estáticas que representarán las opciones que puede seleccionar el usuario como el mostrar los movimientos que pueden hacer las piezas y si quiere que el tablero gire.
    También tendrá otras 2 variables estáticas que representarán las casillas seleccionadas por el jugador activo: **casillaOrigen** y **casillaDestino**.
- **Constructor** : creará la ventana con el menú y el tablero de juego. Además inicializará una partida.
- **Métodos** : contará con varios métodos que permitirán dibujar la ventana y su barra de menú.

4.2.3. Clase *PanelTablero*

Esta clase representa el tablero de juego en forma gráfica.

- **Atributos**
  - **Lista de casillas GUI** : una lista de casillas gráficas. Útil para redibujar el tablero y para poder girarlo.
- **Constructor** : creará el tablero gráfico añadiendo casillas gráficas en una parrilla 8x8.
- **Métodos**
  - **crearTableroGUI** : creará el tablero gráfico añadiendo casillas gráficas al panel y a la lista de casillas
  - **dibujarTableroGUI** : dibujará el tablero según la lista de casillas gráficas.

4.2.4. Clase *PanelCasilla*

Esta clase representa las casillas gráficas que formaráne l tablero.

- **Atributos**
  - **Fila** : coordenada X en el tablero. Sirve para asociar la casilla gráfica a una casilla lógica.
  - **Columna** : coordenada Y en el tablero. Sirve para asociar la casilla gráfica a una casilla lógica.
- **Constructor** : creará una casilla añadiendo:
  - **Coordenadas** : para asociar la casilla gráfica a la casilla lógica.
  - **Dibujar la casilla** : colorear la casilla y dibujar la imagen de la pieza que contenga.
  - ***MouseListener*** : para procesar los eventos que ocurran al clicar sobre las casillas.
- **Métodos** : contará con varios métodos que servirán para colorear, dibujar las piezas y marcar casillas.

4.2.5. Tipo enumerado *DireccionTablero*

Este tipo nos identificará en que sentido está el tablero. Usado para la opción de girar el tablero.

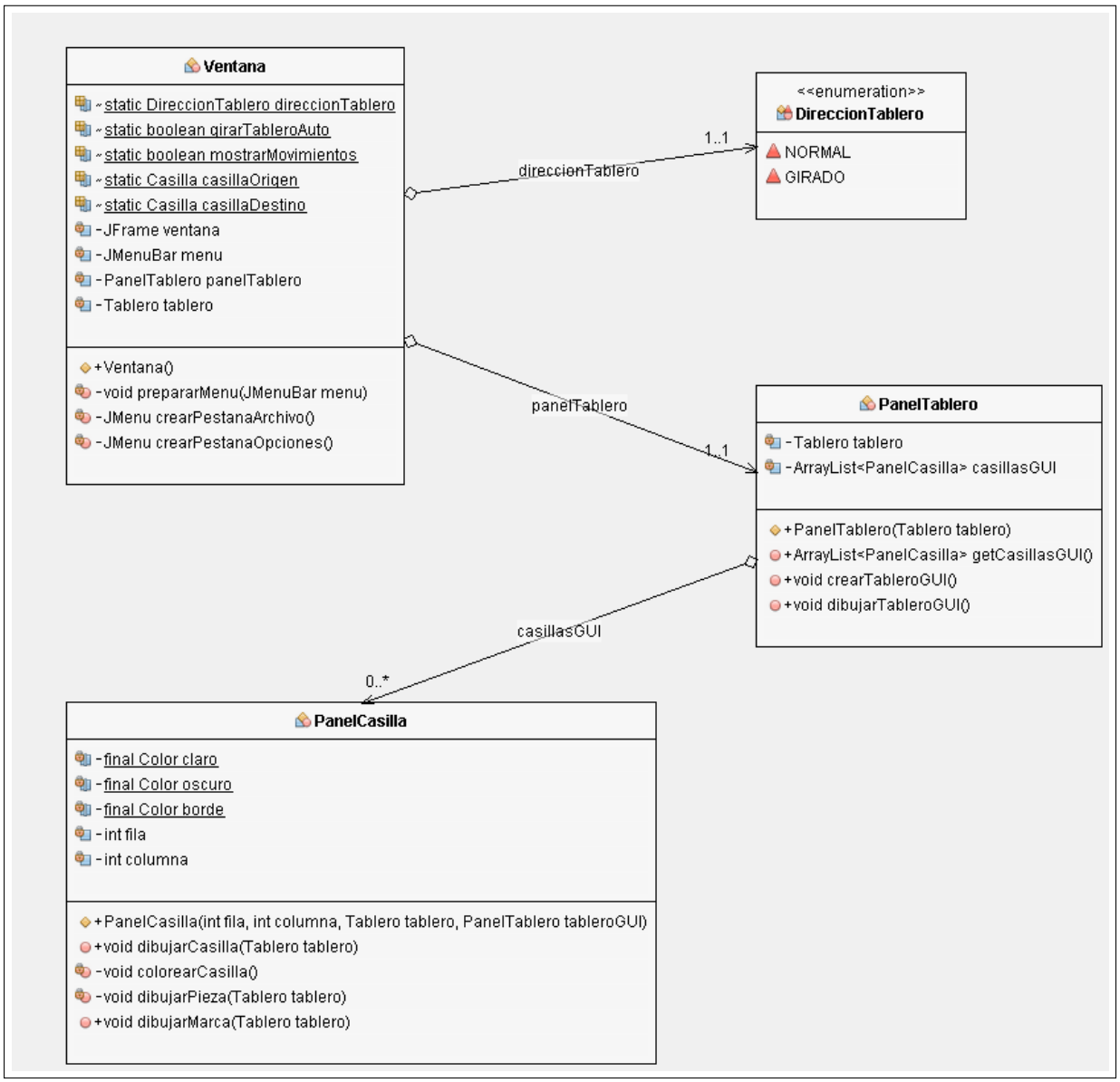
El tablero tendrá dos direcciones: **Normal y Girado**.

También implementaremos 2 métodos abstractos para cada valor que puede tomar:

- **girarTablero** : devolverá la dirección contraria.
- **girar** : devolverá la lista de casillas gráficas ordenadas según la dirección del tablero.

4.2.6. Diagrama de clases

A continuación se puede ver un diagrama de clases de la interfaz gráfica:



5. Conclusión

Nuestro proyecto ha resultado no ser lo que teníamos planeado, como muchos proyectos en el mundo, pero, aún así estamos muy satisfechos con el resultado. No hemos podido implementar todo lo que pretendíamos como algunas reglas adicionales e incluso reglas propias del ajedrez que hubiesen sido interensantes de implementar.

Hemos aprendido y usado cosas que no habíamos visto en detenimiento durante el curso, como por ejemplo la programación por eventos y los tipos enumerados en Java.

Si en algún momento se pudiera retomar este proyecto creemos que sería interesante completar el ajedrez como tal, hacer una versión diferente como teníamos planeado al principio e incluso integrar algortimos de intelegencia artificial para implementar el jugar contra la máquina.

Referencias

[1] Chess. Artículo en *Wikipedia* [en línea]. Fecha de consulta: 31 de Mayo de 2019.
Disponible en: https://en.wikipedia.org/wiki/Chess

[2] Design a chess game using object-oriented principles.
En *CODE REVIEW* [en línea]. Fecha de consulta: 31 de Mayo de 2019.
Disponible en:
https://codereview.stackexchange.com/questions/71790/design-a-chess-game-using-object-oriented-principles

[3] Designing an Object Oriented Chess Engine in Java.
En *E4developer Java Microservices Blog* [en línea]. Fecha de consulta: 31 de Mayo de 2019.
Disponible en:
https://www.e4developer.com/2018/08/16/designing-an-object-oriented-chess-engine-in-java/

[4] Amir Afghani (2018, Mayo 18). BlackWidow-Chess.
En *GitHub* [en línea]. Fecha de consulta: 31 de Mayo de 2019.
Disponible en: https://github.com/amir650/BlackWidow-Chess

[5] Eugenio Navarro, Carlos Cánovas (2019, Junio 05). Chess Royale.
En *GitHub* [en línea]. Fecha de consulta: 05 de Junio de 2019.
Disponible en: https://github.com/enm1986/Chess-Royale