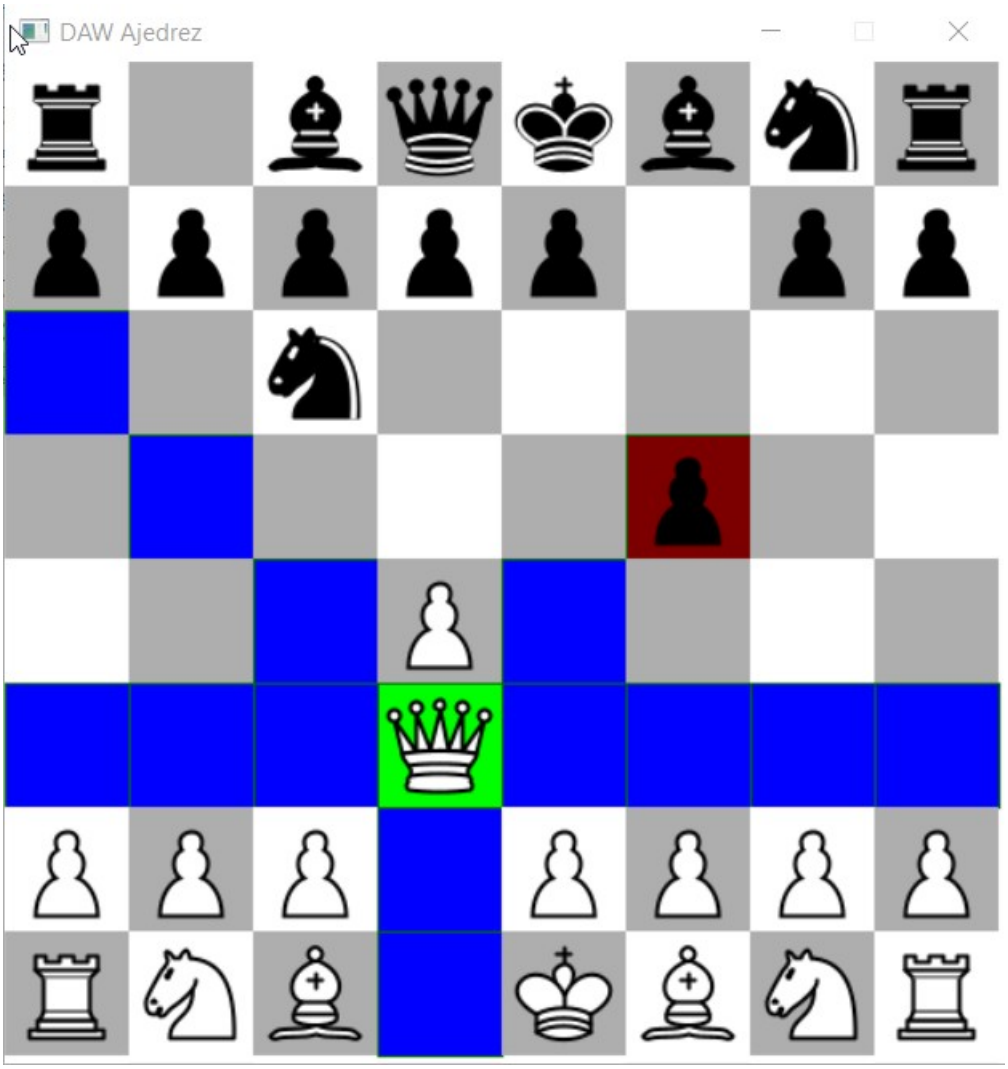
Módulo Programación.

1º DAW.

PRÁCTICA : Herencia.
UNIDAD DE TRABAJO 5.
Profesor: Pedro Antonio Santiago Santiago.

1. Introducción.

La herencia es uno de los pilares de la POO, utilizada ampliamente en todos los lenguajes OO, incluyendo conceptos como clase abstracta, interfaces o polimorfismo. En esta práctica se trabaja la herencia y los principios que esta incluye, terminando el famoso juego de ajedrez con el uso dee herencia.



2. Materiales.

Ordenador con JDK 8 instalado.

Netbeans 12.

Proyecto plantilla con Maven en GitHub. https://github.com/pass1enator/DAWArkanoid/

3. Desarrollo de la práctica.

A partir del código que se facilita crear la lógica necesaria para jugar al ajedrez. Se facilita una interfaz gráfica realizada con la librería Batik que permite gestionar SVG en Java (SG es un lenguaje XML para el dibujo vectorial).

Muchos métodos devuelven vectores de movimientos, estos se han de declarar con el tamaño máximo de movimientos posibles, y comprobar al analizarlo que no sea nulo, en el momento que un elemento sea nulo el resto del vector es nulo. En el siguiente tema se tratan las estructuras dinámicas, que serán las estructuras idóneas para este caso, utilizandose vectores por no haber tratado estas estructuras.

El paquete gráfico pedro.ieslaencanta.com.chess.ui posee las siguientes clases:

- BoardUI: Se encarga de dibujar el tablero, no tiene mayor importancia.
- Message: Muestra un mensaje en la pantalla durante un tiempo determinado, por ejemplo cuando se produce un mate o un jaque mate, simplemente llamar a los métodos si se quiere mostrar algo.
- Principal: Contiene el tablero gráfico y las piezas gráficas, además de un objeto Message para postrar los mensajes, su función principal es interacturar con el controlador de la lógica y representar de forma gráfica el estado interno del juego. Los metodos más destacados son:
    - seleccionarCelda(MouseEvent m): La clase posee un atributo que indica que celda se encuentra seleccionada, el método se llama al hacer click sobre una celda y no existir ninguna pieza seleccionada. La parte lógica devuelve una lista de movimientos, **estos se han de pintar en el tablero con diferentes colores dependiendo si se puede mover y si además es posible eliminar una ficha.**
    - moverCelda(MouseEvent m): En caso de hacer click y tener una figura seleccionada consulta mueve la figura a la nueva celda, tanto en la lógica como en la parte gráfica, siempre que se pueda. **En este punto es necesario al mover la ficha comprobar el estado de la partida (Jaque, Jaquemate...)**
    - ClearMoves(Move[] mvs): Limpia el coloreado de las celdas al seleccionar.

- CreatePieceBoard: Crea a partir de la lógica las figuras (Caballo, Rey….), el código se encuentra comentado y solo muestra peones negros, a medida que se creen las clases restantes se pueden ir pintando.

- PieceUI: Se encarga de cargar los ficheros, almacenándose en un vector en principal para poder crear el tablero.

En la lógica se proporciona:

Paquete pedro.ieslaencanta.com.chess.controller.

- Clase Game: contiene un atributo de la clase Board (modelo) y métodos con la lógica como:
  - Move move(int startrow, int startcol, int endrow, int endcol), mueve en caso de existir una ficha en la fila y columna indicada a la fila y columna final. No es necesario comprobar si es posible si previamente se ha llamado a canMove.
  - public Move[] Jaque(). Devuelve una lista de movimientos de cualquier ficha del tablero que producen un jaque a alguno de los reyes.
  - private boolean IsJaque(Piece rey) . Devuelve si el rey que se le pasa (es necesrio cambiar el tipo Piece por King cuando se implemente, aunque no es imprescindible) tiene un jaque.
  - public boolean JaqueMate(PieceType type). Idem que el anterior pero con JaqueMate y si es blanco o negra.
  - public boolean canMove(int startrow, int startcol, int endrow, int endcol). Si la ficha colocada en esa celda se puede mover a la final, por ejemplo un alfíl solo en diagonal.
  - public Move[] getMoves(int row, int col). Devuelve la lista de movimientos posibles de una celda en caso de tener una figura.
  - public void reset(). Inicializa el tablero.
  - public PieceType getPieceType(int row, int col). Devuelve el tipo de pieza de una celda (blanca, negra), en caso de existir.

Paquete pedro.ieslaencanta.com.chess.model, contiene todo el modelo del juego, que es controlado por el Game, las clases que cotiene son:

- Board: Tiene una matriz de 8x8 de celdas, el método más destacaso es move, para mover una ficha de una posicion a otra.
- Cell: Representa una casilla, posee una posición y una pieza (puede ser un peón, caballo).
- Move: Movimiento, tiene una posición de inicio, una posición de fin, una pieza inicial y una pieza final, se usa por ejemplo para calcular los mates o para saber el color de la celda en la parte gráfica entre otros.
- Position: coordenas en finas y columnas.

Por último dentro del paquete pedro.ieslaencanta.com.chess.model.ChessPieze se encuentran un enumerado para indicarr si una pieza es negra o blanca y la clase Piece.

- Piece. Clase **abstracta** de la que heredan el resto de las piezas, posee diferentes métodos que serán utilizado por el resto de las clases que heredan como:
  - Posee los siguientes atributos:
    - Position p, indica la celda en la que se encuentra.
    - PieceType: Si es blanca o negra.
    - Boolean alive: Si está viva.
    - Move moves[]: Movimientos posibles de la ficha en al posición, se pueden calcular cada vez o más optimo calcularlo solo cuando sea necesario y recalcularlo.
    - Char letter: Tipo de ficha K (King), Q(Queen), util para la parte gráfica.
    - String id: Identificador, para la parte gráfica, se puede usar instanceof y getPieceType también.
  - Move move(Board board, int row, int col), actualiza su posición en el tablero.
  - Move[] getHorizontalMoves(Board b, int row, int start, int end) Calcula los posibles movimientos horizontales de la figura, indicando en que columna empieza y en que columna finaliza, devolviendo la lista de movimientos correctos existentes.

- Move[] getVertical(Board b, int col, int start, int end). Idem que el anterior, pero vertical.
- Move[] getDiagonal(Board b, int dirx, int diry). Idem anteriores pero en diagonal, calculando hasta el final.
- Move[] getDiagonal(Board b, int dirx, int diry, int lenght), se le indica la longitud de la diagonal, por ejemplo el rey es longitud 1.
- boolean canmove(Board board, int row, int col). Si puede mover la ficha a una posición dada (el movimiento tiene que ser correcto, no puede haber una ficha de su color, el camino ha de estar libre…)
- public abstract Move[] getMoves(Board board) Calcula los movimientos posibles de cada pieza, se ha de implementar en cada una de las piezas

El principal trabajo de la práctica, es la implementación de los métodos para completar la clase abstracta Piece y heredar de esta implementando los métodos abstractos y pudiendo sobreescribir los que sean necesarios. También se ha de completar la clase Game haciendo uso de las diferentes clases abstractas y completando los métodos necesarios.

3.1. Código de ejemplo.

Por ejemplo se crean la clase Pawn (Peón) que hereda de Piece:

```
/**
 *
 * @author Pedro
 */
public class Pawn extends Piece {
    private boolean can_move_two;
    public Pawn() {
        super();
        this.can_move_two = true;
        this.letter = 'P';
    }
    public Pawn(int row, int col, PieceType type, boolean alive, String id) {
        super(row, col);
        this.can_move_two = true;
        this.type = type;
        this.alive = alive;
        this.letter = 'P';
        this.setId(id);
    }
    @Override
    public Move move(Board board, int row, int col) {
        this.can_move_two = false;

        return super.move(board, row, col);

    }
    @Override
    public Move[] getMoves(Board board
    ) {
       //PENDIENTE DE IMPLEMENTAR
        return moves;

    }
}
```

Ahora se crea en el Board del modelo los peones, por ejemplo peones blancosy negros en la fila 0 y 7 :

```
for (int i = 0; i < this.cells[1].length ; i++) {
            this.cells[1][i].setPiece(new Pawn(1, i, PieceType.Black,
true,String.valueOf(i)));
            this.cells[6][i].setPiece(new Pawn(6, i, PieceType.White,
true,String.valueOf(i)));
        }
```

Las llamadas a la mecánica del juego se desencadenan al hacer click en las celdas, llamando a los métodos:

void seleccionarCelda(MouseEvent m). En pseudocódigo será:

- Si la fila y columna seleccionada tiene una ficha y es el turno correcto:
  - Obtener la ficha y sleeeccionarla,
  - Obtener los movimientos válidos.
  - Pintar los movimientos válidos en la parte gráfica.

void moverCelda(MouseEvent m). En pseudocódigo:

- Si se puede mover a la celda destino:
  - Limpiar las celdas gráficas(colores).
  - Eliminar figura si mata a alguna.
  - Si jaque
  - Si jaque mate
    - mensaje (“Jaque mate”)
  - sino
    - Mensaje (“Mate”);
  - Cambiar turno.

3.2. Recomendaciones.

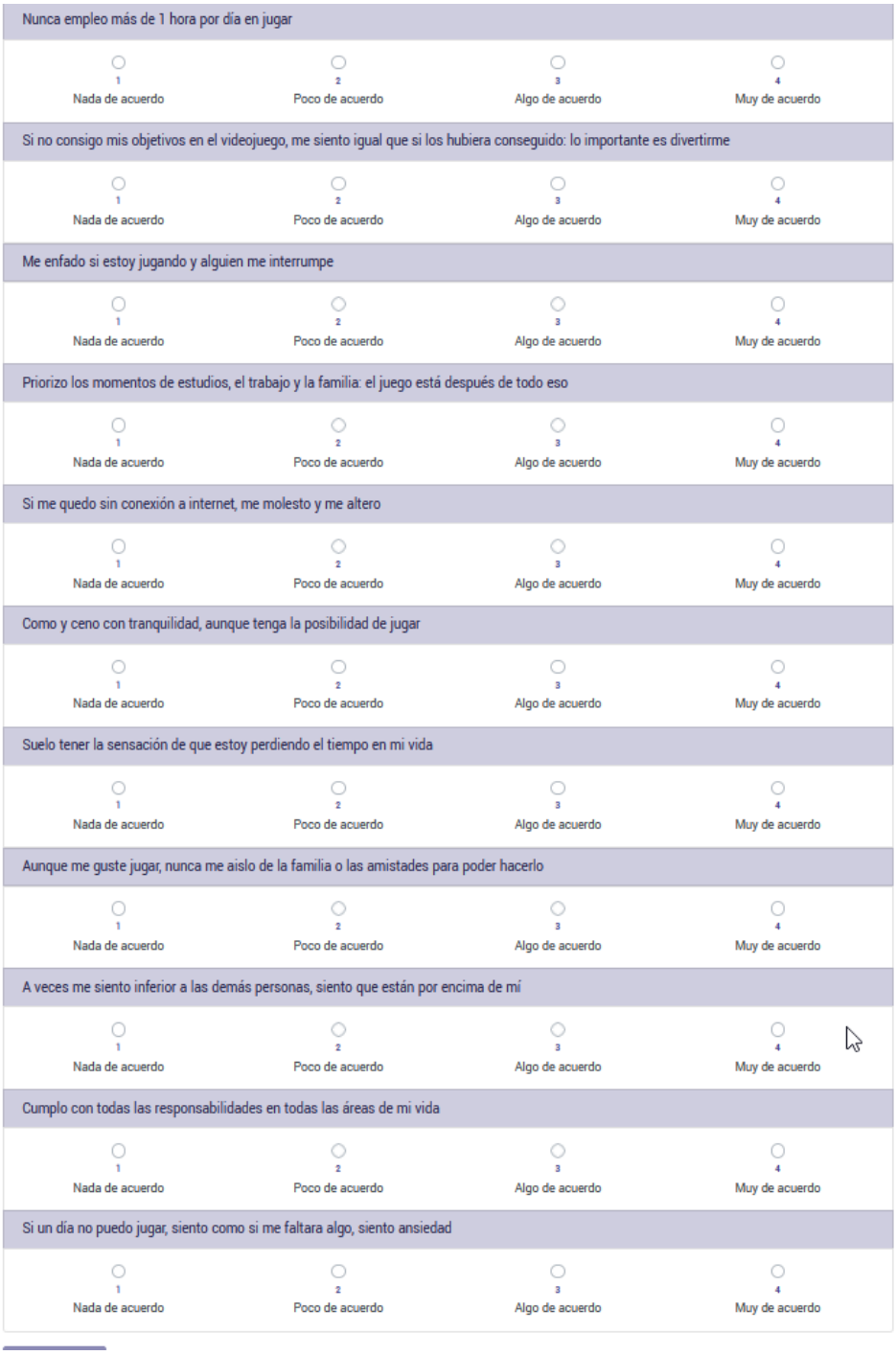
1. Empezar creando una figura, la más sencilla el peón.
2. Implementar los métodos para calcular los movimientos horizontal, vertical… en Piece.
3. Usarlo en el peón.
4. Comprobar el funcionamiento correcto en el Game y la parte gráfica.
5. Crear y probar el resto de piezas.
6. Implementar el mate y el jaque mate.

4. Temas transversales.

La adicción a los videjuegos y las redes sociales es un problema en la sociendad y en especial en la población adolesctente.

A partir del siguiente vídeo y según tus hábitos completar el siguiente cuestionario:

Vídeo:https://empantallados.com/la-adiccion-los-videojuegos-cuando-suponen-problema/



5. Entrega.

La práctica se entrega en formato ZIP con 2 ficheros (código y presentación en formato PDF, en el campus virtual ww.aules.edu.gva.es (pendiente de matricula e inicio de curso). Se fijará la fecha en AULES.

El fichero 1 será el código generado comprimido a su vez también en zip, **importante comentar el código. Cada una de las partes en un proyecto Maven.**

El fichero 2 contiene la presentación de la práctica, que tendrá al menos los siguientes apartados. .

1. Índice.
2. Introducción.
3. Justificación de clases y paquetes creados.
4. Detalles más importantes de implementación de cada clase, por ejemplo.
    - Herencia.
    - Cálculo de movimientos.
    - Algoritmo Jaque y JaqueMate.
5. Conclusiones.

6. Evaluación.

Unos días después de la entrega se realizá se realizará una presentación de 10 minutos por los alumnos en que se explicará apoyándose en la presentación presentada la realización de la práctica. Además se realizará una demostración de la funcionalidad para el resto de la clase.

Finalizada la presentación los componentes del grupo tendrán que responder a preguntas del profesor y/o resto de compañeros

- CE7a. Se han identificado los conceptos de herencia, superclase y subclase.
- CE7b. Se han utilizado modificadores para bloquear y forzar la herencia de clases y métodos.
- CE7c. Se ha reconocido la incidencia de los constructores en la herencia.

- CE7d. Se han creado clases heredadas que sobrescriban la implementación de métodos de la superclase.
- CE7e. Se han diseñado y aplicado jerarquías de clases.
- CE7f. Se han probado y depurado las jerarquías de clases.
- CE7g. Se han realizado programas que implementen y utilicen jerarquías de clases.
- CE7h. Se ha comentado y documentado el código.