Obligatorio II

Programación II

Alvaro Nicoli
220159
xmr.nkr@gmail.com

Darío Dathaguy
220839
dariodathaguy@gmail.com

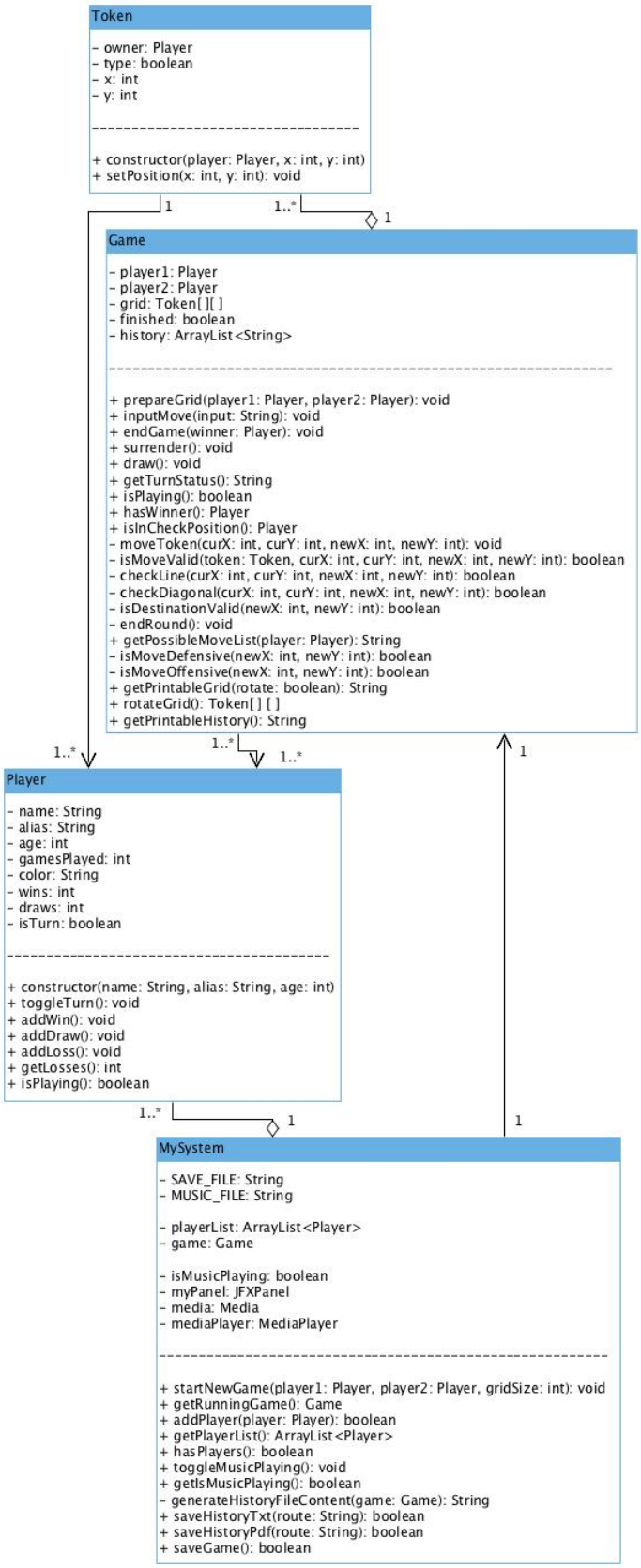
Entregado el día: 27 de noviembre, 2017

Docentes: Sebastián Pesce – Alan Tugentman

Grupo: M2B – Ingeniería en Sistemas

Índice

UML – Diagrama de Clases



Testing

Para el testing del programa utilizamos el método de caja negra que consiste en probar el programa mediante el ingreso de datos numerosos datos ya sean correctos o incorrectos, midiendo así la robustez y correctitud del programa.

Menú Principal

Escenario: Hay jugadores registrados

Menú principal		
Input	Resultado esperado	Resultado obtenido
Click en botón Registrar Jugador	Se accede a la ventana registrar jugador	OK
Click en Botón Jugar	Se accede a la ventana de preparar juego	OK
Click en botón Ranking	Se accede a la ventana de Rankings	OK
Click en botón cerrar ventana	Termina la ejecucción del programa	OK
Click en botón Minimizar	Se minimiza el programa	OK
Click en botón de Música	Se Pausa/Reanuda la música	OK
Click y arrastrar en borde de ventana	Nada, pues no se permite cambiar el tamaño	OK
Botón maximizar	No ocurre nada	OK

Escenario: No hay jugadores registrados

Menú principal		
Input	Resultado esperado	Resultado obtenido
Click en botón Registrar Jugador	Se accede a la ventana registrar jugador	OK
Click en Botón Jugar	Mensaje de error, no hay jugadores registrados	OK
Click en botón Ranking	Mensaje de error, no hay jugadores registrados	OK
Click en botón cerrar ventana	Termina la ejecucción del programa	OK
Click en botón Minimizar	Se minimiza el programa	OK
Click en botón de Música	Se Pausa/Reanuda la música	OK
Click y arrastrar en borde de ventana	Nada, pues no se permite cambiar el tamaño	OK
Botón maximizar	No ocurre nada	OK

Escenario: Hay un juego guardado

Menú principal		
Input	Resultado esperado	Resultado obtenido
Click en botón Registrar Jugador	Se accede a la ventana registrar jugador	OK
Click en Botón Jugar	Mensaje de error, no hay jugadores registrados	OK
Click en botón Ranking	Mensaje de error, no hay jugadores registrados	OK
Click en botón cerrar ventana	Termina la ejecucción del programa	OK
Click en botón Minimizar	Se minimiza el programa	OK
Click y arrastrar en borde de ventana	Nada, pues no se permite cambiar el tamaño	OK
Botón maximizar	No ocurre nada	OK

Registro de jugadores

Escenario: No hay jugadores registrados

Registrar jugador		
Input	Resultado esperado	Resultado
Campo Alias: Juan Campo Nombre: Juan Campo Edad: 18 Click botón OK	Se ingresa el jugador y vuelve al menú	OK
Campo Alias: juan23 Campo Nombre: Juan Campo Edad: 44 Click botón OK	Se ingresa el jugador y vuelve al menú	OK
Campo Alias: juanito Campo Nombre: Juanz123 Campo Edad: 12 Click botón OK	Se ingresa el jugador y vuelve al menú	OK
Campo Alias: pablo Campo Nombre: Pablo Campo Edad: 3 Click botón OK	Se ingresa el jugador y vuelve al menú	OK
Campo Alias: pablo Campo Nombre: Pablo Campo Edad: 100 Click botón OK	Se ingresa el jugador y vuelve al menú	OK
Campo Alias: pablo Campo Nombre: Pablo Campo Edad: diez Click botón OK	Mensaje de Error: "Debe ingresar un número entero entre 3 y 100"	OK
Campo Alias: pablo Campo Nombre: Pablo Campo Edad: 2 Click botón OK	Mensaje de Error: "Debe ingresar un número entero entre 3 y 100"	OK
Campo Alias: pablo Campo Nombre: Pablo Campo Edad: 12.1 Click botón OK	Mensaje de Error: "Debe ingresar un número entero entre 3 y 100"	OK
Campo Alias: pablo Campo Nombre: Pablo Campo Edad: 101 Click botón OK	Mensaje de Error: "Debe ingresar un número entero entre 3 y 100"	OK
Campo Alias: pablo Campo Nombre: Pablo Campo Edad: -1 Click botón OK	Mensaje de Error: "Debe ingresar un número entero entre 3 y 100"	OK

Escenario: Hay un jugador registrado con alias Juan, nombre Juan y edad 40.

Registrar jugador		
Input	Resultado esperado	Resultado
Campo Alias: Juan Campo Nombre: Juan Campo Edad: 18 Click botón OK	Mensaje de Error: "El alias ya existe"	OK
Campo Alias: JuAn Campo Nombre: Juan Campo Edad: 18 Click botón OK	Mensaje de Error: "El alias ya existe"	OK
Campo Alias: Juan Campo Nombre: JuanJuan Campo Edad: 18 Click botón OK	Mensaje de Error: "El alias ya existe"	OK
Campo Alias: juan Campo Nombre: Juan Campo Edad: 1 Click botón OK	Mensaje de Error: "El alias ya existe"	OK
Campo Alias: juan23 Campo Nombre: Juan Campo Edad: 1 Click botón OK	Se ingresa el jugador y vuelve al menú	OK

Preparar Juego

Escenario: Hay solamente un jugador registrado de alias Juan, nombre Juan y edad 40.

Preparar juego		
Input	Resultado esperado	Resultado
Jugador 1: Juan Jugador 2: Juan Tamaño del tablero: 3 Click botón OK	Mensaje de Error: "Elige 2 jugadores distintos"	OK
Jugador 1: Juan Jugador 2: Juan Tamaño del tablero: 5 Click botón OK	Mensaje de Error: "Elige 2 jugadores distintos"	OK

Escenario: Hay varios jugadores registrados

Preparar juego		
Input	Resultado esperado	Resultado
Jugador 1: Juan Jugador 2: Juan Tamaño del tablero: 3 Click botón OK	Mensaje de Error: "Elige 2 jugadores distintos"	OK
Jugador 1: Juan Jugador 2: Juan Tamaño del tablero: 5 Click botón OK	Mensaje de Error: "Elige 2 jugadores distintos"	OK
Jugador 1: Juan Jugador 2: Pepe Tamaño del tablero: 3 Click botón OK	Se abre una ventana para jugar la partida de 3x3 entre Juan y Pepe	OK
Jugador 1: Juan Jugador 2: Pepe Tamaño del tablero: 5 Click botón OK	Se abre una ventana para jugar la partida de 5x5 entre Juan y Pepe	OK
Jugador 1: Pepe Jugador 2: Agus Tamaño del tablero: 5 Click botón OK	Se abre una ventana para jugar la partida de 5x5 entre Pepe y Agus	OK

Jugar

Escenario: Los jugadores de alias Pepe, Juan y Agus juegan.

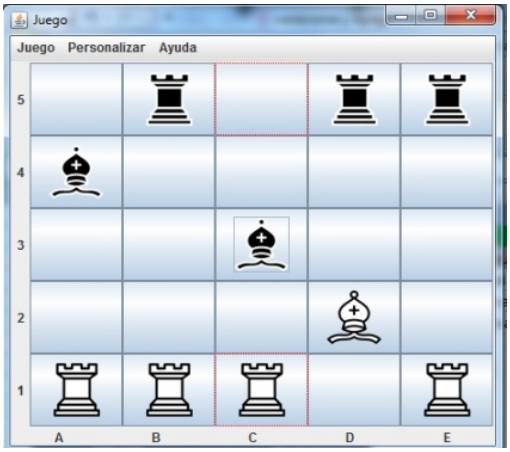
Juego 1		
Input	Resultado esperado	Resultado
Jugador 1: Pepe Jugador 2: Juan Tipo:5X5		
CTRL Y	Se abre una ventana con todas las posibles jugadas para el jugador 1 se enfatiza el ataque C5 C1	OK
Click en C5 y luego en C1	Se mueve la pieza y come a la pieza del oponente ubicada en C1 y cambia a alfil	OK
Click en B5 y luego en B2	No ocurre nada ya que el jugador no tiene ficha en B5	OK
Click en A1 y luego B2	Mensaje de error: "Movimiento invalido"	OK
Click en Ayuda y luego Posibles Movimientos	Se abre una ventana con las posibles jugadas, se enfatiza B1 C1 y D1 C1 ya que esta en jaque	OK
Click en B1 y luego C1	Se mueve la pieza y come a la pieza del oponente ubicada en su arco C1 y pasa a alfil	OK
Click en D5 y luego D2	Se mueve la pieza a desde D5 a la posicion D2 y cambia a alfil	OK
Click en A1 Y luego A2	Se mueve la pieza desde A1 a A2 y cambia a alfil	OK
Click en D2 y luego C1	Se mueve la pieza y come a la pieza del oponente ubicada en C1 y cambia a torre	OK
CTRL X	Solicita rendición y se despliega una ventana para elegir Si o No para el Oponente	OK
Click en el botón Si	Acepta la rendición y vuelve al menú	OK

Juego 2		
Input	Resultado esperado	Resultado
Jugador 1: Pepe Jugador 2: Agus Tipo:5X5		
CTRL Y	Se abre una ventana con todas las posibles jugadas para el jugador 1 se enfatiza el ataque C5 C1	OK
Click en C5 y luego en C1	Se mueve la pieza y come a la pieza del oponente ubicada en C1 y cambia a alfil	OK
Click en Ayuda y luego Posibles Movimientos	Se abre una ventana con las posibles jugadas, se enfatiza B1 C1 y D1 C1 ya que esta en jaque	OK
Click en D1 y luego C1	Se mueve la pieza y come la pieza del oponente ubicada en el arco C1 y pasa a alfil	OK
Mayús H	Se abre una ventana con el historial de jugadas	OK
Click en A5 y luego en A2	Se mueve la pieza a desde A5 a la posicion A2 y cambia a alfil	OK
CTRL E	Jugador 2 Solicita tablas y se despliega una ventana para elegir Si o No para el Oponente	OK
Click en botón No	Jugador 1 rechaza tablas, continúa el turno del jugador 2	OK
CTRL R	Se rota el tablero	OK
Click en B1 y luego en B4	Se mueve la pieza desde B1 a B4 y cambia a alfil	OK
Click en A2 y luego en C4	Mueve la pieza a esa posicion(Puede ya que es un alfil), pasa a ser Torre	OK
Click en E1 y luego en E2	Mueve la pieza a esa posicion y cambia a alfil	OK
Click en C4 y luego en C1	Mueve la pieza al arco rival y pasa a ser Torre	OK
Click en E2 y luego en D3	Mensaje de error: "Movimiento invalido", el jugador esta en jaque y debe defender su arco	OK
Click en A1 y luego en C1	Mueve la pieza a C1 y defiende su arco, pasa a ser un alfil	OK
Click en D5 y luego en C2	Se mueve la pieza a desde D5 a la posicion D2 y cambia a alfil	OK
Click en B4 y lugo en A3	Mueve la pieza a A3 y pasa a ser una torre	OK
Click en D2 y luego C1	Ataca el arco rival. Como el jugador 2 no puede defender, Pepe gana	OK

Juego 3		
Input	Resultado esperado	Resultado
Jugador 1: Agus Jugador 2: Juan Tipo 3X3		
Click en C3 y luego en B1	Mensaje de error: "Movimiento Inválido"	OK
Click en C1 y luego en A2	No ocurre nada ya que C1 no es ficha de Agus	OK
Click en Ayuda y luego Posibles Movimientos	Se abre una ventana con todas las posibles jugadas, se enfatiza en ataque B3 B1	OK
Click en C3 y luego en C2	Se mueve la pieza desde C3 a C2 y cambia a alfil	OK
Click en ayuda y luego Historial de Movimientos	Se abre una ventana con el historial de jugadas	OK
Click en Juego y luego Rotar tablero	Se rota el tablero	OK
Click en A1 y luego en A2	Se mueve la pieza desde A1 a A2 y cambia a alfil	OK
Click en C2 y luego B1	Se mueve la pieza y come a la pieza del oponente ubicada en B1 (arco rival) y cambia a torre	OK
CTRL M	Se Pausa/Reanuda la música	OK
Click en C1 y luego B1	El jugador 2 defiende su arco y cambia a alfil	OK
Click en juego y luego Música	Se Pausa/Reanuda la música	OK
Click en B3 y luego B1	Se mueve la pieza y come a la pieza del oponente ubicada en B1 (arco rival) y cambia a alfil	OK
Click en juego y luego en Me Rindo	Solicita rendición y se despliega una ventana para elegir Si o No para el Oponente	OK
Click en No	Rechaza rendición y continúa Juan	OK
Click en A2 y luego en B1	El jugador 2 defiende su arco y cambia a torre	OK
Click en A3 y luego en A2	Se mueve la pieza y cambia a alfil	OK
Click en B1 y luego B3	El jugador 2 ataca el arco rival y cambia a alfil	OK
CTRL S	Se abre una ventana para guardar el historial, se elije Mis Documentos y se guarda como h1.txt	OK
Click en A2 y luego B3	Defiende su arco y gana el juego ya que el jugador 2 no tiene más fichas.	OK

Escenario: Se está jugando una partida y el tablero está como en la siguiente captura.

Mueven blancas.



Input	Resultado esperado	Resultado
Click en D2 luego en B4	Mensaje de error: "Movimiento invalido" (no se pueden atravesar fichas)	OK
Click en D2 luego en B1	Mensaje de error: "Movimiento invalido"	OK
Click en D2 luego en B2	Mensaje de error: "Movimiento invalido"	OK
Click en A1 luego en A3	Mueve la pieza desde A1 a A3 y cambia a alfil	OK
Click en D2 luego en A3	Mensaje de error: "Movimiento invalido"	OK
Click en D5 luego en E4	No ocurre nada ya que D5 no es una pieza blanca	OK
Click en B1 luego en B5	Mensaje de error: Movimiento invalido". No se pueden comer fichas a menos que sea en el arco rival	OK
Click en C1 luego en C1	Mensaje de error: "Movimiento invalido" (No se puede mover una ficha al mismo lugar)	OK

Escenario: Cualquiera dentro de la partida.

Input	Resultado esperado	Resultado
Click personalizar, luego Cambiar fondo Se elige un color y se da a aceptar	Se abre una ventana donde se elige el nuevo fondo del tablero Se cambia el color del tablero	OK OK
Click personalizar, luego Cambiar fondo Se elige un color y se clickea cancelar	Se abre una ventana donde se elige el nuevo fondo del tablero El color del tablero no cambia	OK OK
Click personalizar, luego Cambiar fondo Se elige un color y se clickea reestablecer	Se abre una ventana donde se elige el nuevo fondo del tablero El color elegido vuelve a ser el actual del tablero	OK OK
Click en personalizar, luego en Cambiar Torre negra Se selecciona el archivo.	Se abre una ventana donde se selecciona un archivo para la torre negra Las torres negras cambian a la imagen	OK OK
Click en personalizar, luego en Cambiar Torre blanca Se selecciona el archivo.	Se abre una ventana donde se selecciona un archivo para la torre blanca Las torres blancas cambian a la imagen	OK OK
Click en personalizar, luego en Cambiar Alfil negro Se selecciona el archivo.	Se abre una ventana donde se selecciona un archivo para el alfil negro Los alfiles negros cambian a la imagen	OK OK
Click en personalizar, luego en Cambiar Alfil blanco Se selecciona el archivo.	Se abre una ventana donde se selecciona un archivo para el alfil blanco Los alfiles blancos cambian a la imagen	OK OK
Click en Ayuda luego Guardar historial, CTRL S Nombre del archivo termina en .txt y click en guardar Nombre del archivo termina en .pdf y click en guardar El nombre del archivo no es un .txt o pdf Click en botón cancelar	Se abre una ventana donde se selecciona la ubicación y el nombre del archivo a guardar Se guarda el archivo en la ubicación mostrando los jugadores, el historial y la fecha como txt Se guarda el archivo en la ubicación mostrando los jugadores, el historial y la fecha como pdf Se muestra un mensaje de error al guardar el archivo Se vuelve al juego	OK OK OK OK OK

Ranking

Escenario: Hay jugadores registrados pero ninguno ha jugado aún.

Input	Resultado esperado	Resultado
En el menú se clickea el botón Ranking	Se muestran todos los jugadores, todos sus datos en la tabla con 0.	OK

Escenario: Se jugaron tres partidas entre Pepe, Juan y Agus.

Input	Resultado esperado	Resultado
En el menú se clickea el botón Ranking	Se despliega el ranking. pepe23 lidera el ranking con 2 victorias y 2 partidas, los 2 restantes cuentan con una partida ganada y una perdida cada uno	OK

Escenario: Se jugaron seis partidas, todos contra todos los cuatro jugadores.

Input	Resultado esperado	Resultado
En el menú se clickea el botón Ranking	Se despliega el ranking, todos tienen los mismos datos, 3 partidos jugados, 3 empates, 0 victorias y 0 derrotas	OK

Código fuente

Game.java

```
public class Game implements Serializable {
    private Player player1;
    private Player player2;
    private Token[][] grid;
    private boolean finished;
    private ArrayList<String> history; // Store previous moves

    public Game(Player player1, Player player2, int size) {
        this.player1 = player1;
        this.player1.setColor(Utils.ANSI_RED);
        this.player1.toggleTurn();
        this.player2 = player2;
        this.player2.setColor(Utils.ANSI_BLUE);
        this.grid = new Token[size][size];
        this.prepareGrid(player1, player2);

        this.history = new ArrayList<>();
    }

    /**
     * Start getters
     */

    public int getGridSize() {
        return this.grid.length;
    }

    public Player getPlayer1() {
        return this.player1;
    }

    public Player getPlayer2() {
        return this.player2;
    }

    public Token[][] getGrid() {
        return this.grid;
    }

    /**
     * End getters
     */

    /**
     * Start game logic
     */

    /**
     * Populate the grid with the tokens
     * @param player1
     * @param player2
     */
    public void prepareGrid(Player player1, Player player2) {
        for (int i = 0; i < this.grid.length; i++) {
            this.grid[0][i] = new Token(player1, 0, i);
            this.grid[this.grid.length - 1][i] = new Token(player2, this.grid.length - 1, i);
        }
    }

    /**
     * Takes the String coordinates of the tokens to move and makes the corresponding move if valid
     * @param input - The user inputted move
     * @throws Exception - The move is invalid
     */
    public void inputMove(String input) throws Exception {
```

```
        input = input.toUpperCase();
        String historyInput = input;

        input = input.replace(" ", "");
        int curY = Utils.charToInt(input.charAt(0));
        int curX = this.grid.length - Integer.parseInt(String.valueOf(input.charAt(1)));
        int newY = Utils.charToInt(input.charAt(2));
        int newX = this.grid.length - Integer.parseInt(String.valueOf(input.charAt(3)));

        historyInput = this.grid[curX][curY].getOwner().getColor() + this.grid[curX]
[curY].getOwner().getAlias() + Utils.ANSI_RESET + " -> " + historyInput;

        // Checking if the owner of the token is the one playing has to be done here
        // Since checking it in isMoveValid() will lead to the valid play list being
        // Incomplete for the other player when checking for a winner
        if (this.grid[curX][curY].getOwner().isPlaying()) {
            moveToken(curX, curY, newX, newY);
            this.history.add(historyInput); // This line will only be reached if the move is valid,
else an exception will have been thrown
        }
    }

    /**
     * Changes the game status to finished and ends the ongoing turn
     * @param winner - The player who should be given the game
     */
    public void endGame(Player winner) {
        // End the turn of whoever is playing
        if (this.player1.isPlaying()) {
            this.player1.toggleTurn();
        } else {
            this.player2.toggleTurn();
        }

        if (winner != null) {
            winner.addWin(); // Add the victory to whoever won the game

            // Also log the loss to the other player
            if (this.player1.equals(winner)) {
                this.player2.addLoss();
            } else {
                this.player1.addLoss();
            }
        } else {
            // If the winner is null it means the game is a draw
            this.player1.addDraw();
            this.player2.addDraw();
        }

        this.finished = true;
    }

    /**
     * Will end the game with a victory of the player whose turn it is not
     */
    public void surrender() {
        if (this.player1.isPlaying()) {
            endGame(this.player2);
        } else {
            endGame(this.player1);
        }
    }

    /**
     * Will end the game as a draw
     */
    public void draw() {
        endGame(null);
    }

    /**
     * Generates a String with information on whose turn it is
```

```
     * @return - Human readable turn information
     */
    public String getTurnStatus() {
        if (this.player1.isPlaying()) {
            return "Es turno de " + this.player1.getColor() + this.player1.getAlias() +
Utils.ANSI_RESET;
        } else {
            return "Es turno de " + this.player2.getColor() + this.player2.getAlias() +
Utils.ANSI_RESET;
        }
    }

    /**
     * Returns whether the game is ongoing
     * @return - If the game is still on
     */
    public boolean isPlaying() {
        return !this.finished;
    }

    /**
     * Determines if the game has been won by someone
     * @return - The winner
     */
    public Player hasWinner() {
        // Check if a player is out of moves, in which case they are as good as dead
        if (getPossibleMoveList(this.player1).length() == 0) {
            return this.player2;
        }

        if (getPossibleMoveList(this.player2).length() == 0) {
            return this.player1;
        }

        return null;
    }

    /**
     * Check if someone is in a check position
     * @return - The player in check position
     */
    public Player isInCheckPosition() {
        Player retVal = null;

        if (grid.length == 5) {
            try {
                if (grid[4][2].getOwner().equals(this.player1)) {
                    retVal = this.player1;
                }
            } catch (Exception e) {

            }

            try {
                if (grid[0][2].getOwner().equals(this.player2)) {
                    retVal = this.player2;
                }
            } catch (Exception e) {

            }
        } else {
            try {
                if (grid[2][1].getOwner().equals(this.player1)) {
                    retVal = this.player1;
                }
            } catch (Exception e) {

            }

            try {
                if (grid[0][1].getOwner().equals(this.player2)) {
                    retVal = this.player2;
```

```
                }
            } catch (Exception e) {

            }
        }

        return retVal;
    }

    /**
     * Checks if a move is valid and commits it to the grid
     * @param curX - Current X axis position of the token to move
     * @param curY - Current Y axis position of the token to move
     * @param newX - New X axis position of the token to move
     * @param newY - New Y axis position of the token to move
     * @throws Exception - If the move is invalid or the any position passed is invalid
     */
    private void moveToken(int curX, int curY, int newX, int newY) throws Exception {
        Token aux = this.grid[curX][curY];

        if (isMoveValid(aux, curX, curY, newX, newY)) {
            aux.setPosition(newX, newY);

            this.grid[curX][curY] = null;
            this.grid[newX][newY] = aux;

            endRound();
        } else {
            throw new Exception("Invalid Move Exception");
        }
    }

    /**
     * Validates a move
     * @param token - The token to move
     * @param curX - Current X axis position of the token to move
     * @param curY - Current Y axis position of the token to move
     * @param newX - New X axis position of the token to move
     * @param newY - New Y axis position of the token to move
     * @return - If the move is valid
     */
    private boolean isMoveValid(Token token, int curX, int curY, int newX, int newY) {
        boolean retVal;
        boolean notMoving = curX == newX && curY == newY;
        boolean isMoveDiagonal = curX != newX && curY != newY && Math.abs(newX-curX) ==
Math.abs(newY-curY);
        boolean isMoveHorizontal = curX == newX;
        boolean isMoveVertical = curY == newY;
        boolean hasToDefend = !token.getOwner().equals(isInCheckPosition()) && isInCheckPosition() !
= null;
        boolean isMoveDefensive = isMoveDefensive(newX, newY);

        if (Utils.removeColorFromString(token.toString()).equals("T")) {
            // Make sure move is either vertical or horizontal and that the token didnt go over any
other token
            retVal = ((isMoveHorizontal || isMoveVertical) && checkLine(curX, curY, newX, newY));
        } else {
            // Make sure move is diagonal and that the token didnt go over any other token
            retVal = (isMoveDiagonal && checkDiagonal(curX, curY, newX, newY));
        }

        // If the move is valid and the token that is being moved belongs to the player that is
playing return true
        return (!notMoving && (hasToDefend == isMoveDefensive) && retVal && isDestinationValid(newX,
newY));
    }

    /**
     * Verifies a token does not go over others as it moves though a horizontal or vertical line
     * @param curX - Current X axis position of the token to move
     * @param curY - Current Y axis position of the token to move
     * @param newX - New X axis position of the token to move
```

```
     * @param newY - New Y axis position of the token to move
     * @return - The line to go over is clean
     */
    private boolean checkLine(int curX, int curY, int newX, int newY) {
        boolean lineIsEmpty = true;
        boolean horizontal = curX == newX; // Used to determine whether to check a horizontal line

        int curPos = curX == newX ? curY : curX;
        int newPos = curX == newX ? newY : newX;

        if (curPos > newPos) {
            int aux = newPos;
            newPos = curPos;
            curPos = aux;
        }

        for (int i = curPos + 1; i < newPos && lineIsEmpty; i++) {
            if (horizontal) {
                lineIsEmpty = lineIsEmpty ? this.grid[curX][i] == null : lineIsEmpty;
            } else {
                lineIsEmpty = lineIsEmpty ? this.grid[i][curY] == null : lineIsEmpty;
            }
        }

        return lineIsEmpty;
    }

    /**
     * Verifies a token does not go over others as it moves through a diagonal line
     * @param curX - Current X axis position of the token to move
     * @param curY - Current Y axis position of the token to move
     * @param newX - New X axis position of the token to move
     * @param newY - New Y axis position of the token to move
     * @return - The line to go over is clean
     */
    private boolean checkDiagonal(int curX, int curY, int newX, int newY) {
        boolean diagonalIsEmpty = true;
        Token origin = this.grid[curX][curY];

        while (curX != newX && curY != newY) {
            if (!origin.equals(this.grid[curX][curY])) {
                diagonalIsEmpty = diagonalIsEmpty ? this.grid[curX][curY] == null : diagonalIsEmpty;
            }

            if (curX > newX) {
                curX--;
            } else {
                curX++;
            }

            if (curY > newY) {
                curY--;
            } else {
                curY++;
            }
        }

        return diagonalIsEmpty;
    }

    /**
     * Will determine whether the destination of a token is valid
     * Any destination is valid if there is no token there
     * A token can overtake another if such token does not belong to the same player and it is in goal
     * @param newX - Destination X pos
     * @param newY - Destination Y pos
     * @return - Move has valid destination?
     */
    private boolean isDestinationValid(int newX, int newY) {
        boolean retVal = true;

        if (this.grid[newX][newY] != null) {
```

```
            // If the owner is playing then the move is not valid
            // You cannot remove your own tokens
            if (this.grid[newX][newY].getOwner().isPlaying()) {
                retVal = false;
            } else {
                // To remove the other player's tokens, those have to be in some goal
                retVal = !(newX != 0 || newY != (this.grid.length - 1) / 2) || !(newX !=
this.grid.length - 1 || newY != (this.grid.length - 1) / 2);
            }
        }

        return retVal;
    }

    /**
     * Finishes a player's turn and starts the other's
     */
    private void endRound() {
        this.player1.toggleTurn();
        this.player2.toggleTurn();
    }

    /**
     * End game logic
     */

    /**
     * Will list all possible moves for a player
     * @param player - the player to check for possible moves
     * @return - Moves
     */
    public String getPossibleMoveList(Player player) {
        String retVal = "";

        ArrayList<String> regularMoves = new ArrayList<>();
        ArrayList<String> defensiveMoves = new ArrayList<>();
        ArrayList<String> offensiveMoves = new ArrayList<>();

        for (int i = 0; i < this.grid.length; i++) {
            for (int j = 0; j < this.grid[i].length; j++) {
                if (this.grid[i][j] != null) {
                    if (this.grid[i][j].getOwner().equals(player)) {
                        for (int m = 0; m < this.grid.length; m++) {
                            for (int n = 0; n < this.grid[m].length; n++) {
                                if (this.isMoveValid(this.grid[i][j], i, j, m, n)) {
                                    if (isMoveDefensive(m, n)) {
                                        defensiveMoves.add(Utils.makeMoveString(this, i, j, m, n));
                                    } else if (isMoveOffensive(m, n)) {
                                        offensiveMoves.add(Utils.makeMoveString(this, i, j, m, n));
                                    } else {
                                        regularMoves.add(Utils.makeMoveString(this, i, j, m, n));
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        if (!defensiveMoves.isEmpty()) {
            retVal += "Movimientos de defensa\n";
            for (int i = 0; i < defensiveMoves.size(); i++) {
                retVal += defensiveMoves.get(i);
            }
        }
        if (!offensiveMoves.isEmpty()) {
            retVal += "Movimientos de ataque\n";
            for (int i = 0; i < offensiveMoves.size(); i++) {
                retVal += offensiveMoves.get(i);
            }
        }
```

```
        if (!regularMoves.isEmpty()) {
            retVal += "Movimientos normales\n";
            for (int i = 0; i < regularMoves.size(); i++) {
                retVal += regularMoves.get(i);
            }
        }

        return retVal;
    }

    /**
     * By means of the destination of a move determines whether it is defensive
     * @param newX - X pos
     * @param newY - Y pos
     * @return - Is the destination the player's goal?
     */
    private boolean isMoveDefensive(int newX, int newY) {
        boolean retVal;

        if (this.grid.length == 5) {
            if (this.player1.isPlaying()) {
                retVal = newX == 0 && newY == 2;
            } else {
                retVal = newX == 4 && newY == 2;
            }
        } else {
            if (this.player1.isPlaying()) {
                retVal = newX == 0 && newY == 1;
            } else {
                retVal = newX == 2 && newY == 1;
            }
        }

        return retVal;
    }

    /**
     * By means of the destination of a move determines whether it is offensive
     * If its destination is the opponent's goal it is true
     * @param newX - X pos
     * @param newY - Y pos
     * @return - Is the destination the opponent's goal?
     */
    private boolean isMoveOffensive(int newX, int newY) {
        boolean retVal;

        if (this.grid.length == 5) {
            if (this.player1.isPlaying()) {
                retVal = newX == 4 && newY == 2;
            } else {
                retVal = newX == 0 && newY == 2;
            }
        } else {
            if (this.player1.isPlaying()) {
                retVal = newX == 2 && newY == 1;
            } else {
                retVal = newX == 0 && newY == 1;
            }
        }

        return retVal;
    }

    /**
     * Generates a String object to present the grid to the user by means of the terminal
     * @param rotate - If the grid should be rotated
     * @return - The beautiful grid
     */
    public String getPrintableGrid(boolean rotate) {
        String retVal = ""; // Value to be returned
        Token [][] grid = rotate ? rotateGrid() : this.grid;
```

```
        boolean addingElements = false; // Whether the row being added to retVal contains Tokens or
only divider characters

        for (int i = 0; i < grid.length; i++) {
            // Determines whether the row being added contains part of a goal
            // If i == 4 the behavior will always be the same, if i == 0 or 1 the behavior is the same
in case of the divider
            boolean hasGoal = i == 0 || (i == 1 && !addingElements) || i == grid.length - 1;

            for (int j = 0; j < grid[i].length; j++) {
                // Determines whether the horizontal position corresponds to that of the goal
                boolean isGoal = hasGoal && j >= (grid[i].length / 2) && j < ((grid[i].length / 2) +
2);

                if (addingElements) {
                    if (j == 0) {
                        if (rotate) {
                            retVal += i + 1 + " ";
                        } else {
                            retVal += grid.length - i + " ";
                        }
                    }

                    retVal += (isGoal ? (Utils.ANSI_GREEN + "*" + Utils.ANSI_RESET) : "|") +
(grid[i][j] != null ? (grid[i][j] + Utils.ANSI_RESET) : " ");
                } else {
                    if (j == 0) {
                        retVal += "   ";
                    }

                    retVal += isGoal ? ((j == (grid[i].length / 2) + 1) ? (Utils.ANSI_GREEN + "*" +
Utils.ANSI_RESET + "-") : (Utils.ANSI_GREEN + "**" + Utils.ANSI_RESET)) : "+-";
                }
            }

            if (addingElements) {
                // Finishes a line with Tokens
                retVal += "|\n";

                // Subtracts one to the vertical index in order to add elements correctly
                if (i == grid.length - 1) {
                    i--;
                }
            } else {
                // Finishes a divider line
                retVal += "+\n";

                // Remove escape characters from retVal and store the clean String in memory
                // Doing this in order for java not to get confused when calculating the length of the
String
                // Since it counts the color escape chars as regular chars
                String retValWithoutColor = Utils.removeColorFromString(retVal);

                // Subtracts one to the vertical index in order to add the last divider line
                if ((this.grid.length == 5 && retValWithoutColor.length() < (11 * 11 + 22)) ||
(this.grid.length == 3 && retValWithoutColor.length() < (7 * 7 + 14))) {
                    i--;
                }
            }

            // Change addingElements to its opposite value
            addingElements = !addingElements;
        }

        if (rotate) {
            retVal += grid.length == 5 ? "   E D C B A\n" : "   C B A\n";
        } else {
            retVal += grid.length == 5 ? "   A B C D E\n" : "   A B C\n";
        }

        return retVal;
    }
```

```
    public Token[][] rotateGrid() {
        Token[][] retVal = new Token[this.grid.length][this.grid.length];

        for (int i = 0; i < this.grid.length; i++) {
            for (int j = 0; j < this.grid[i].length; j++) {
                retVal[this.grid.length - i - 1][this.grid[i].length - j - 1] = this.grid[i][j];
            }
        }

        return retVal;
    }

    public String getPrintableHistory() {
        String retVal = "";

        for (int i = history.size() - 1; i >= 0; i--) {
            retVal += history.get(i) + "\n";
        }

        return retVal;
    }
}
```

MySystem.java

```
public class MySystem implements Serializable {
    private static final transient String SAVE_FILE = "/tmp/inversions-save-data.txt";
    private static final transient String MUSIC_FILE = "/res/Local Forecast - Elevator.mp3";

    private ArrayList<Player> playerList;
    private Game game;

    private transient boolean isMusicPlaying;
    private transient JFXPanel myPanel;
    private transient Media media;
    private transient MediaPlayer mediaPlayer;

    /**
     * The constructor will attempt to load a saved instance before creating a new one
     */
    public MySystem() {
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(SAVE_FILE))) {
            MySystem sys = (MySystem) in.readObject();
            this.playerList = sys.playerList;
            this.game = sys.game;
        } catch (Exception e) {
            this.playerList = new ArrayList<>();
        }

        try {
            this.myPanel = new JFXPanel();
            this.media = new Media(getClass().getResource(MUSIC_FILE).toURI().toString());
            this.mediaPlayer = new MediaPlayer(media);
            this.mediaPlayer.setAutoPlay(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void startNewGame(Player player1, Player player2, int gridSize) {
        this.game = new Game(player1, player2, gridSize);
    }

    /**
     * Returns the current game instance
     * @return - The game if there is an ongoing one - else null
     */
```

```
    public Game getRunningGame() {
        if (this.game != null) {
            if (!this.game.isPlaying()) {
                this.game = null;
            }
        }

        return this.game;
    }

    public boolean addPlayer(Player player) {
        if (this.playerList.contains(player)) {
            return false;
        }

        this.playerList.add(player);
        return true;
    }

    public ArrayList<Player> getPlayerList(){
        Collections.sort(this.playerList);
        return playerList;
    }

    public boolean hasPlayers(){
        return !this.getPlayerList().isEmpty();
    }

    public void toggleMusicPlaying() {
        if (this.mediaPlayer == null) {
            return;
        }

        this.isMusicPlaying = !this.isMusicPlaying;

        if (this.isMusicPlaying) {
            this.mediaPlayer.play();
        } else {
            this.mediaPlayer.pause();
        }
    }

    public boolean getIsMusicPlaying() {
        return this.isMusicPlaying;
    }

    private String generateHistoryFileContent(Game game) {
        String retVal = "Inversiones - Historial\n";

        Date now = new Date();
        DateFormat df = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        retVal += df.format(now) + "\n\n";

        retVal += "Jugadores:\n\n";
        retVal += game.getPlayer1().toString() + "\n\n" + game.getPlayer2().toString() + "\n\n";

        String[] history = getRunningGame().getPrintableHistory().split("\n");
        for (String item : history) {
            retVal += Utils.removeColorFromString(item) + "\n";
        }

        return retVal;
    }

    public boolean saveHistoryTxt(String route) {
        boolean retVal = false;
        FileOutputStream fos = null;

        try {
            File fout = new File(route);
            fos = new FileOutputStream(fout);
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(fos));
```

```
            bw.write(generateHistoryFileContent(getRunningGame()));

            bw.close();
            retVal = true;
        } catch (Exception ex) {
            // Logger.getLogger(MySystem.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            try {
                fos.close();
            } catch (Exception ex) {
                // Logger.getLogger(MySystem.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

        return retVal;
    }

    public boolean saveHistoryPdf(String route) {
        boolean retVal = false;

        try {
            Document document = new Document();
            // step 2
            PdfWriter.getInstance(document, new FileOutputStream(route));
            // step 3
            document.open();
            // step 4
            document.add(new Paragraph(generateHistoryFileContent(getRunningGame())));

            // step 5
            document.close();
            retVal = true;
        } catch (Exception ex) {
            // Logger.getLogger(MySystem.class.getName()).log(Level.SEVERE, null, ex);
        }

        return retVal;
    }

    /**
     * Saves the instance of MySystem to a txt file
     * @return - Whether the MySystem instance could successfully be saved
     */
    public boolean saveGame() {
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(SAVE_FILE))) {
            out.writeObject(this);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
}
```

Player.java

```
public class Player implements Comparable<Player>, Serializable {
    private String name;
    private String alias;
    private int age;
    private int gamesPlayed;
    private String color;
    private int wins;
    private int draws;
    private boolean isTurn;


    public Player(String name, String alias, int age) {
        this.name = name;
```

```
        this.alias = alias;
        this.age = age;
    }

    // Setters
    public void setName(String name) {
        this.name = name;
    }

    public void setAlias(String alias) {
        this.alias = alias;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public void toggleTurn() {
        this.isTurn = !this.isTurn;
    }

    public void addWin() {
        this.gamesPlayed++;
        this.wins++;
    }

    public void addDraw() {
        this.gamesPlayed++;
        this.draws++;
    }

    public void addLoss() {
        this.gamesPlayed++;
    }

    // Getters

    public String getName() {
        return name;
    }

    public String getAlias() {
        return alias;
    }

    public int getAge() {
        return age;
    }

    public int getGamesPlayed() {
        return gamesPlayed;
    }

    public int getWins() {
        return wins;
    }

    public int getDraws() {
        return draws;
    }

    public int getLosses() {
        return this.gamesPlayed - (this.wins + this.draws);
    }

    public String getColor() {
        return color;
    }
```

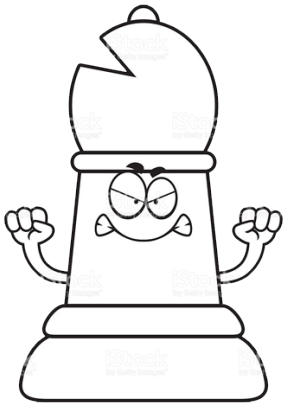
```
    public boolean isPlaying() {
        return isTurn;
    }

    @Override
    public String toString() {
        return "Alias: " + this.alias
                + "\n\tVictorias: " + this.wins
                + "\n\tEmpates: " + this.draws
                + "\n\tDerrotas: " + this.getLosses()
                + "\n\tPartidas jugadas: " + this.gamesPlayed;
    }

    @Override
    public boolean equals(Object o) {
        boolean retVal = false;

        if (o instanceof Player) {
            Player player = (Player) o;
            retVal = this.getAlias().equalsIgnoreCase(player.getAlias());
        }

        return retVal;
    }

    @Override
    public int compareTo(Player p) {
        int retVal;

        if (this.wins < p.getWins()) {
            retVal = 1;
        } else if (this.wins > p.getWins()) {
            retVal = -1;
        } else {
            if (this.draws < p.getDraws()) {
                retVal = 1;
            } else if (this.draws > p.getDraws()) {
                retVal = -1;
            } else {
                if (this.gamesPlayed < p.getGamesPlayed()) {
                    retVal = 1;
                } else if (this.gamesPlayed > p.getGamesPlayed()) {
                    retVal = -1;
                } else {
                    retVal = 0;
                }
            }
        }

        return retVal;
    }
}
```

Token.java

```
public class Token implements Serializable {
    private Player owner;
    private boolean type = false; // true = Tower - false = Bishop
    private int x;
    private int y;

    public Token(Player player, int x, int y) {
        this.setOwner(player);
        this.setPosition(x, y);
    }

    public void setOwner(Player player) {
        owner = player;
```

```
    }

    public void setPosition(int x, int y) {
        this.x = x;
        this.y = y;
        this.type = !this.type;
    }

    public Player getOwner() {
        return owner;
    }

    public boolean getType() {
        return this.type;
    }

    @Override
    public String toString() {
        return this.owner.getColor() + (type ? "T" : "A");
    }
}
```

Ficha del juego



Simple y divertido juego de mesa donde las fichas cambian sus movimientos.

Inversiones es un juego de mesa cuyo objetivo es atacar la base enemiga.

Existen dos tipos de fichas, torres y alfiles, luego de cada movimiento de una ficha, esta pasa de ser una Torre a un alfil y viceversa. Sus movimientos son los mismos que en el ajedrez.

El tablero cuenta con 2 arcos, en el medio de la primer y última fila, uno para cada jugador. Para obtener la victoria se debe atacar la base enemiga sin que el oponente sea capaz de defenderla.

Este juego cuenta con creación de usuarios, ranking y además varias opciones de personalización, entre ellas cambiar el fondo del tablero y las imágenes de las fichas.

Si te encuentras perdido o estas aprendiendo a jugar, hay opciones de ayuda e historial (que pueden guardarse en un archivo externo), además se puede solicitar empate o bien rendirse.

Inversiones ofrece una experiencia completa digna de un juego de mesa como tal.

Tipo de aplicación: Juego.

Género: Juego de mesa.

Edades: 3+