Universidad ORT

Facultad de Ingeniería

Programación II

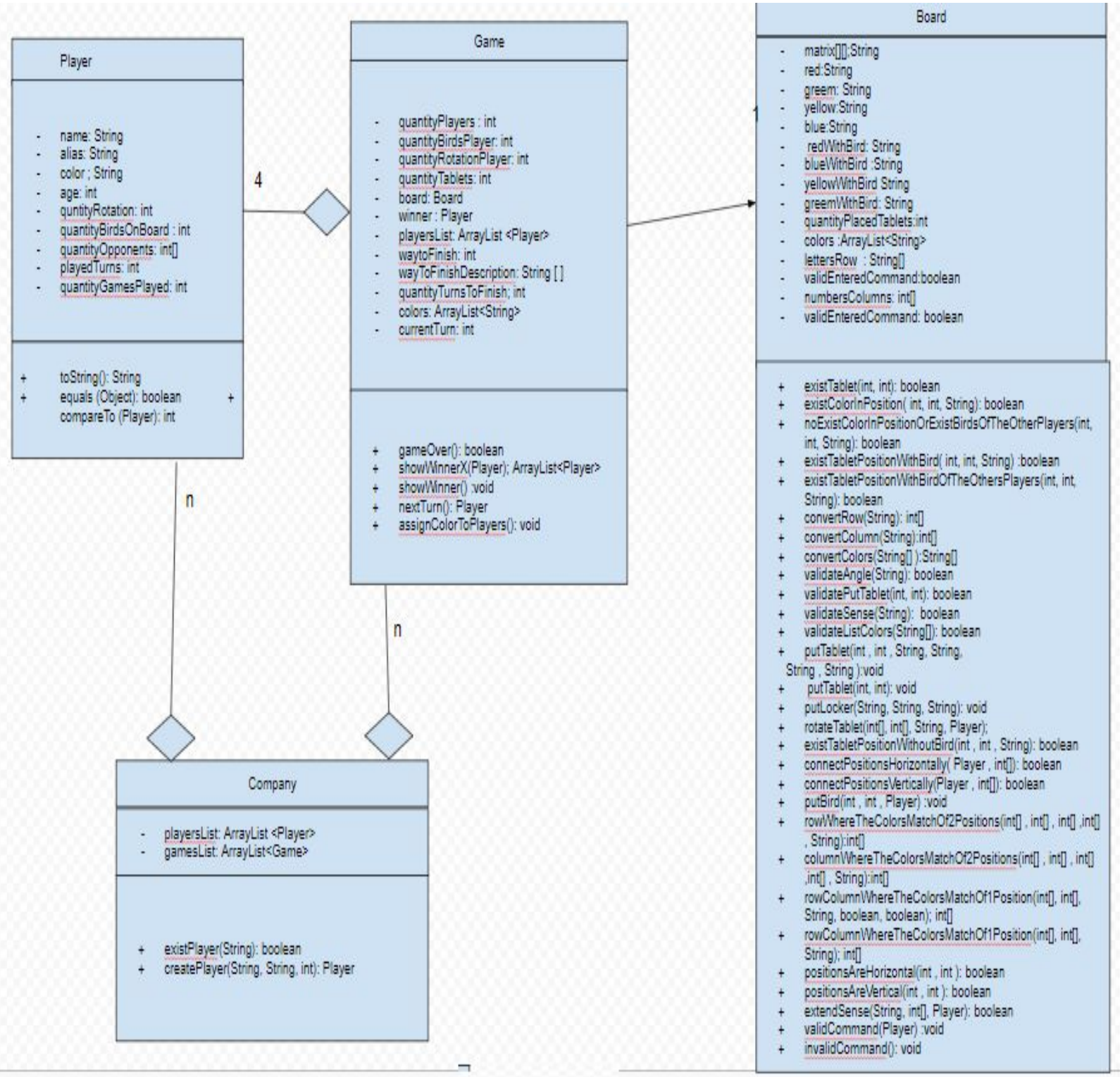
Obligatorio I - Grupo M2A

Taliah Schatz (180443)

Cinthya Calbete (229418)

Indice

UML


Player
- name: String
- alias: String
- color ; String
- age: int
- quntityRotation: int
- quantityBirdsOnBoard : int
- quantityOpponents: int[]
- playedTurns: int
- quantityGamesPlayed: int
+ toString(): String
+ equals (Object): boolean
compareTo (Player): int
4
n
Game
- quantityPlayers : int
- quantityBirdsPlayer: int
- quantityRotationPlayer: int
- quantityTablets: int
- board: Board
- winner : Player
- playersList: ArrayList <Player>
- waytoFinish: int
- wayToFinishDescription: String [ ]
- quantityTurnsToFinish; int
- colors: ArrayList<String>
- currentTurn: int
+ gameOver(): boolean
+ showWinnerX(Player); ArrayList<Player>
+ showWinner() :void
+ nextTurn(): Player
+ assignColorToPlayers(): void
n
Board
- matrix[][]:String
- red:String
- greem: String
- yellow:String
- blue:String
- redWithBird: String
- blueWithBird :String
- yellowWithBird String
- greemWithBird: String
- quantityPlacedTablets:int
- colors :ArrayList<String>
- lettersRow : String[]
- validEnteredCommand:boolean
- numbersColumns: int[]
- validEnteredCommand: boolean
+ existTablet(int, int): boolean
+ existColorInPosition(int, int, String): boolean
+ noExistColorInPositionOrExistBirdsOfTheOtherPlayers(int, int, String): boolean
+ existTabletPositionWithBird(int, int, String) :boolean
+ existTabletPositionWithBirdOfTheOthersPlayers(int, int, String): boolean
+ convertRow(String): int[]
+ convertColumn(String):int[]
+ convertColors(String[]):String[]
+ validateAngle(String): boolean
+ validatePutTablet(int, int): boolean
+ validateSense(String): boolean
+ validateListColors(String[]): boolean
+ putTablet(int , int , String, String, String , String):void
+ putTablet(int, int): void
+ putLocker(String, String, String): void
+ rotateTablet(int[], int[], String, Player);
+ existTabletPositionWithoutBird(int , int , String): boolean
+ connectPositionsHorizontally(Player , int[]): boolean
+ connectPositionsVertically(Player , int[]): boolean
+ putBird(int , int , Player) :void
+ rowWhereTheColorsMatchOf2Positions(int[] , int[] , int[] ,int[] , String):int[]
+ columnWhereTheColorsMatchOf2Positions(int[] , int[] , int[] ,int[] , String):int[]
+ rowColumnWhereTheColorsMatchOf1Position(int[], int[], String, boolean, boolean); int[]
+ rowColumnWhereTheColorsMatchOf1Position(int[], int[], String); int[]
+ positionsAreHorizontal(int , int): boolean
+ positionsAreVertical(int , int): boolean
+ extendSense(String, int[], Player): boolean
+ validCommand(Player) :void
+ invalidCommand(): void
Company
- playersList: ArrayList <Player>
- gamesList: ArrayList<Game>
+ existPlayer(String): boolean
+ createPlayer(String, String, int): Player

Testing

Caso de prueba 1:

Jugada que finaliza con un ganador
Configuración: Forma 3 de finalizar con 2 turnos por jugador, el resto por defecto.
2 jugadores

Objetivo: Probar el correcto funcionamiento de una partida, ingresando comandos válidos y finalizando con un ganador.

Descripción de dato a probar	Resultado esperado	Resultado obtenido
Jugador rojo ingresa comando: pm e4 varm	Se coloca una tableta en fila E columna 4 con colores rojo y azul arriba y verde y amarillo abajo	OK
Jugador azul ingresa comando: c e4 e5	Se colocan 3 aves en fila E desde columna 4 donde está el casillero azul hasta columna 5(2 casilleros)	OK
Jugador rojo ingresa comando: C e4 e6	Se coloca una tableta en fila E columna 7 con colores rojo y verde arriba y azul y amarillo abajo	OK
Jugador azul ingresa comando: Pm e7 ravm	Se colocan 2 aves en fila E desde columna 6 donde está el casillero azul hasta columna 7(1 casillero)	OK
	Se muestra un mensaje indicando que el jugador rojo es el ganador	OK

Caso de prueba 2:

Jugada que finaliza en un empate
Configuración: Forma 3 de finalizar con 3 turnos por jugador, el resto por defecto.
2 jugadores

Objetivo: Probar el correcto funcionamiento de una partida, ingresando comandos válidos y finalizando con un empate de aves colocadas.

Descripción de dato a probar	Resultado esperado	Resultado obtenido
Jugador rojo ingresa comando: pm e7 rmav	Se coloca una tableta en fila E columna 7 con colores rojo y amarillo arriba y azul y verde abajo	OK
Jugador azul ingresa comando: R e7 90	Se rota la tableta quedando azul y rojo arriba y verde y amarillo abajo	OK
Jugador rojo ingresa comando: C e5 e7	Se colocan 6 aves en fila E desde columna 5 donde está el casillero rojo hasta columna 7(2 casilleros)	OK
Jugador azul ingresa comando: Pm e8 rvam	Se coloca una tableta en fila E columna 8 con colores rojo y verde arriba y azul y amarillo abajo	OK
Jugador rojo ingresa comando: E d e5	Se colocan 2 aves más, quedando un total de 8 aves desde fila e columna 5 hasta columna 8	OK
Jugador azul ingresa comando: c e6 e8	Se colocan 4 aves en fila e desde columna 6 donde esta el color azul hasta columna 8 donde esta el color azul	OK
	Se muestra un mensaje indicando que el jugador rojo es el ganador	OK

Caso de prueba 3:

Jugada donde se ingresan comandos no válidos
Configuración: Forma 2 de finalizar con 2 aves por jugador, el resto por defecto.
2 jugadores

Objetivo: Probar el correcto funcionamiento de una partida, ingresando comandos válidos e inválidos.

Descripción de dato a probar	Resultado esperado	Resultado obtenido
Jugador rojo ingresa comando (sin letra correspondiente a fila): Pm 4 vmra	Se indica que el comando no tiene formato válido y se solicita el ingreso nuevamente.	OK
Jugador rojo ingresa comando: Pm f5 rvam	Se coloca una tableta en fila F columna 5 con colores rojo y verde arriba y azul y amarillo abajo	OK
Jugador azul ingresa comando: R F5 180	Se rota la tableta quedando amarillo y azul arriba y verde y rojo abajo	OK
Jugador rojo ingresa comando (fila columna sin tableta): C F5 F6	Se indica que el comando no tiene formato válido y se solicita el ingreso nuevamente.	OK
Jugador rojo ingresa comando: Pm F6 mrva	Se coloca una tableta en fila F columna 6 con colores amarillo y rojo arriba y verde y azul abajo	OK
Jugador azul ingresa comando (no existente): J J1 J2 ravm	Se indica que el comando no tiene formato válido y se solicita el ingreso nuevamente.	OK
Jugador azul ingresa comando: C E5 F5	Se colocan aves sobre el casillero azul en e5 y el casillero azul en F5	OK
	Se muestra un mensaje indicando que el jugador azul es el ganador	OK

Innovaciones tecnológicas sudamericanas

1. Nombre: Proyecto argentino para personas no videntes.
   Sitio web:
   https://www.infobae.com/discapacidad/2017/05/08/tecnologia-inclusiva-el-innovador-proyecto-argentino-para-personas-no-videntes/
   Descripción:
   Este proyecto consta en la creación de un ajedrez tridimensional con referencias en braille al costado del tablero.
   En el tablero, los cuadros negros quedaron iguales, mientras que los blancos fueron fabricados en sub-nivel. Además, lo hicieron con un agujero en el centro para que se pueda encastrar las piezas y en los bordes del tablero se van a colocar las referencias con números, tanto como letras, todo en braille. También las piezas oscuras están editadas y se les dio una terminación puntiaguda, para que se puedan diferenciar.
   Pensando en el futuro el grupo argentino, está pensando en agregar un módulo de reproducción de audio para grabar y convertirse así en un tablero parlante.
   En el caso de nuestro juego, nos gustaría que el mismo sea al igual que el mencionado anterior, transformado en forma tridimensional, además que cada uno de los comandos se puedan leer en braille e incluso si se pudiese agregar un audio que vaya reproduciendo las instrucciones.

Nombre:HandTalk
Sitio web:https://www.handtalk.me/
Descripción:
Es un software de origen brasileño cuyo objetivo es la traducción del lenguaje de señas con la asistencia de “Hugo” un intérprete tridimensional, que convierte automáticamente el contenido a lenguaje de señas . La aplicación permite los discapacitados a leer, escribir, hablar e incluso tomar fotografías. También esta aplicación tiene una herramienta para poder llegar a cualquier sitio web en lenguaje de signos. La aplicación consta de un menú principal donde al pulsar el botón + despliega otro menú para poder escribir o dictar lo que se quiera traducir y luego hugo lo transmite, se tratan de mensajes pequeños puesto que la aplicación admite hasta 140 caracteres. De forma automática hugo vuelve a aparecer para gesticular y mostrar la frase a través de sus extremidades, por lo que es claramente expresivo.
En el caso de nuestro juego nos gustaría implementar la idea de crear un carácter similar a “Hugo” para que los comandos o cualquier tipo de expresión del lenguaje escrito pueda ser decodificado para personas sordomudas.

Listado de clases

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package obligatorio;

import static java.applet.Applet.newAudioClip;
import java.applet.AudioClip;
import java.util.ArrayList;
import java.util.Collections;
import java.awt.*;
import java.net.URL;

public class Board {

    private String matrix[][];
    private String red = (char) 27 + "[" + String.valueOf(41) + "m" + " " + "\u001B[0m";
    private String blue = (char) 27 + "[" + String.valueOf(44) + "m" + " " + "\u001B[0m";
    private String green = (char) 27 + "[" + String.valueOf(42) + "m" + " " + "\u001B[0m";
    private String yellow = (char) 27 + "[" + String.valueOf(43) + "m" + " " + "\u001B[0m";
    private String redWithBird = (char) 27 + "[" + String.valueOf(41) + "m" + "X" +
"\u001B[0m";
    private String greenWithBird = (char) 27 + "[" + String.valueOf(42) + "m" + "X" +
"\u001B[0m";
    private String yellowWithBird = (char) 27 + "[" + String.valueOf(43) + "m" + "X" +
"\u001B[0m";
    private String blueWithBird = (char) 27 + "[" + String.valueOf(44) + "m" + "X" +
"\u001B[0m";
    private int quantityPlacedTablets = 0;
    private ArrayList<String> colors = new ArrayList<>();
    private String[] lettersRow = new String[19];
    private int[] numbersColumns = new int[11];
    private boolean validEnteredCommand;

    public Board() {
        this.matrix = new String[20][20];
        putTablet(8, 8, red, blue, green, yellow);
        putTablet(8, 10, green, yellow, red, blue);
        this.colors.add((char) 27 + "[" + String.valueOf(41) + "m" + " " + "\u001B[0m");
        this.colors.add((char) 27 + "[" + String.valueOf(44) + "m" + " " + "\u001B[0m");
        this.colors.add((char) 27 + "[" + String.valueOf(42) + "m" + " " + "\u001B[0m");
        this.colors.add((char) 27 + "[" + String.valueOf(43) + "m" + " " + "\u001B[0m");
        this.lettersRow[0] = "A";
        this.lettersRow[2] = "B";
```

```
        this.lettersRow[4] = "C";
        this.lettersRow[6] = "D";
        this.lettersRow[8] = "E";
        this.lettersRow[10] = "F";
        this.lettersRow[12] = "G";
        this.lettersRow[14] = "H";
        this.lettersRow[16] = "I";
        this.lettersRow[18] = "J";
        this.numbersColumns[1] = 0;
        this.numbersColumns[2] = 2;
        this.numbersColumns[3] = 4;
        this.numbersColumns[4] = 6;
        this.numbersColumns[5] = 8;
        this.numbersColumns[6] = 10;
        this.numbersColumns[7] = 12;
        this.numbersColumns[8] = 14;
        this.numbersColumns[9] = 16;
        this.numbersColumns[10] = 18;
        this.validEnteredCommand = true;
    }

    public boolean existTablet(int row, int column) {
        return existColorInPosition(row, column, "Whatever");
    }

    public boolean existColorInPosition(int row, int column, String color) {
        return existTabletPositionWithoutBird(row, column, color)
                || existTabletPositionWithBird(row, column, color);
    }

    public boolean noExistColorInPositionOrExistBirdsOfTheOtherPlayers(int row, int
column, String color) {
        return !existTabletPositionWithoutBird(row, column, "Whatever")
                || existTabletPositionWithBirdOfTheOthersPlayers(row, column, color);
    }

    public boolean existTabletPositionWithBird(int row, int column, String colorPlayer) {
        boolean existBird = false;
        if (this.getMatrix()[row][column] != null) {
            switch (colorPlayer) {
                case "Red":
                    if (this.getMatrix()[row][column].equals(redWithBird)) {
                        existBird = true;
                    }
                    break;
                case "Blue":
                    if (this.getMatrix()[row][column].equals(blueWithBird)) {
                        existBird = true;
                    }
```

```
                break;
            case "Yellow":
                if (this.getMatrix()[row][column].equals(yellowWithBird)) {
                    existBird = true;
                }
                break;
            case "Green":
                if (this.getMatrix()[row][column].equals(greenWithBird)) {
                    existBird = true;
                }
                break;
            case "Whatever":
                existBird = this.getMatrix()[row][column].equals(redWithBird)
                        || this.getMatrix()[row][column].equals(blueWithBird)
                        || this.getMatrix()[row][column].equals(yellowWithBird)
                        || this.getMatrix()[row][column].equals(greenWithBird);
                break;
        }
    }
    return existBird;
  }

   public boolean existTabletPositionWithBirdOfTheOthersPlayers(int row, int column,
String color) {
    boolean existBird = false;
    switch (color) {
        case "Red":
            existBird = this.getMatrix()[row][column].equals(blueWithBird)
                    || this.getMatrix()[row][column].equals(yellowWithBird)
                    || this.getMatrix()[row][column].equals(greenWithBird);
            break;
        case "Blue":
            existBird = this.getMatrix()[row][column].equals(redWithBird)
                    || this.getMatrix()[row][column].equals(yellowWithBird)
                    || this.getMatrix()[row][column].equals(greenWithBird);
            break;
        case "Yellow":
            existBird = this.getMatrix()[row][column].equals(redWithBird)
                    || this.getMatrix()[row][column].equals(blueWithBird)
                    || this.getMatrix()[row][column].equals(greenWithBird);
            break;
        case "Green":
            existBird = this.getMatrix()[row][column].equals(redWithBird)
                    || this.getMatrix()[row][column].equals(blueWithBird)
                    || this.getMatrix()[row][column].equals(yellowWithBird);
            break;
    }
    return existBird;
  }
```

```
public int[] convertRow(String letterRow) {
    int[] rows = new int[2];
    rows[0] = -1;
    rows[1] = -1;
    letterRow = letterRow.toUpperCase();
    for (int i = 0; i <= 18; i = i + 2) {
        if (lettersRow[i].equals(letterRow)) {
            rows[0] = i;
            rows[1] = i + 1;
        }
    }
    return rows;
}

public int[] convertColumn(String column) {
    int[] columns = new int[2];
    columns[0] = -1;
    columns[1] = -1;
    int columnAux = Integer.parseInt(column);
    if (columnAux >= 1 && columnAux <= 10) {
        columns[0] = this.numbersColumns[columnAux];
        columns[1] = this.numbersColumns[columnAux] + 1;
    }
    return columns;
}

public String[] convertColors(String[] colors) {
    String[] colorsReturn = new String[colors.length];
    for (int i = 0; i < colors.length; i++) {
        switch (colors[i].toUpperCase()) {
            case "R":
                colorsReturn[i] = red;
                break;
            case "A":
                colorsReturn[i] = blue;
                break;
            case "V":
                colorsReturn[i] = green;
                break;
            case "M":
                colorsReturn[i] = yellow;
                break;
            default:
                colorsReturn[i] = "invalid";
                break;
        }

    }
```

```
        return colorsReturn;
    }

    public boolean validateAngle(String angle) {
        boolean valid = false;
        if (angle.equals("90") || angle.equals("180") || angle.equals("270")) {
            valid = true;
        }
        return valid;
    }

    public boolean validatePutTablet(int row, int column, int quantityTabletsConfigured) {
        boolean valid = true;
        if (quantityTabletsConfigured == this.quantityPlacedTablets || existTablet(row,
column)) {
            valid = false;
        } else {
            for (int i = 0; i < this.getMatrix().length; i += 2) {
                for (int j = 0; j < this.getMatrix()[0].length; j += 2) {
                    if (existTablet(i, j)) {
                        int diffRow = i - row;
                        int diffColumn = j - column;
                        if (diffRow >= 10 || diffRow <= -10 || diffColumn >= 10 || diffColumn <= -10)
{
                            valid = false;
                        }
                    }
                }
            }
        }
        return valid;
    }

    public boolean validateSense(String sense) {
        boolean valid = true;
        if (sense.equals("I") || sense.equals("D") || sense.equals("A") || sense.equals("B")) {
            valid = true;
        } else {
            valid = false;
        }
        return valid;
    }

    public Boolean validateListColors(String[] colors) {
        boolean valid = true;
        int quantityRed = 0;
        int quantityBlue = 0;
        int quantityGreen = 0;
        int quantityYellow = 0;
```

```
        if (colors.length == 4) {
            for (int i = 0; i < colors.length; i++) {

                if (colors[i].equals(red)) {
                    quantityRed++;
                }
                if (colors[i].equals(blue)) {
                    quantityBlue++;
                }
                if (colors[i].equals(green)) {
                    quantityGreen++;
                }
                if (colors[i].equals(yellow)) {
                    quantityYellow++;
                }
            }
        } else {
            valid = false;
        }

        if (quantityRed != 1 || quantityBlue != 1 || quantityGreen != 1 || quantityYellow != 1) {
            valid = false;
        }

        return valid;
    }

    public void putTablet(int firstRow, int firstColumn, String firstColor, String secondColor,
            String thirdColor, String quarterColor) {
        this.matrix[firstRow][firstColumn] = firstColor;
        this.matrix[firstRow][firstColumn + 1] = secondColor;
        this.matrix[firstRow + 1][firstColumn] = thirdColor;
        this.matrix[firstRow + 1][firstColumn + 1] = quarterColor;
        this.quantityPlacedTablets++;
    }

    public void putLocker(String row, String column, String color) {
        int rowAux = Integer.parseInt(row);
        int columnAux = Integer.parseInt(column);
        this.matrix[rowAux][columnAux] = color;
    }

    public void putTablet(int firstRow, int firstColumn) {
        Collections.shuffle(colors);
        putTablet(firstRow, firstColumn, colors.get(0), colors.get(1),
                colors.get(2), colors.get(3));
    }

    public void rotateTablet(int[] rows, int[] columns, String angle, Player player) {
```

```
        String[] locker1Aux = new String[3];
        String[] locker2Aux = new String[3];
        String[] locker3Aux = new String[3];
        String[] locker4Aux = new String[3];

        locker1Aux[0] = this.getMatrix()[rows[0]][columns[0]];
        locker1Aux[1] = Integer.toString(rows[0]);
        locker1Aux[2] = Integer.toString(columns[0]);
        locker2Aux[0] = this.getMatrix()[rows[0]][columns[1]];
        locker2Aux[1] = Integer.toString(rows[0]);
        locker2Aux[2] = Integer.toString(columns[1]);
        locker3Aux[0] = this.getMatrix()[rows[1]][columns[0]];
        locker3Aux[1] = Integer.toString(rows[1]);
        locker3Aux[2] = Integer.toString(columns[0]);
        locker4Aux[0] = this.getMatrix()[rows[1]][columns[1]];
        locker4Aux[1] = Integer.toString(rows[1]);
        locker4Aux[2] = Integer.toString(columns[1]);

        switch (angle) {
            case "90":
                putLocker(locker1Aux[1], locker1Aux[2], locker3Aux[0]);
                putLocker(locker2Aux[1], locker2Aux[2], locker1Aux[0]);
                putLocker(locker3Aux[1], locker3Aux[2], locker4Aux[0]);
                putLocker(locker4Aux[1], locker4Aux[2], locker2Aux[0]);
                break;
            case "180":
                putLocker(locker1Aux[1], locker1Aux[2], locker4Aux[0]);
                putLocker(locker2Aux[1], locker2Aux[2], locker3Aux[0]);
                putLocker(locker3Aux[1], locker3Aux[2], locker2Aux[0]);
                putLocker(locker4Aux[1], locker4Aux[2], locker1Aux[0]);
                break;
            case "270":
                putLocker(locker1Aux[1], locker1Aux[2], locker2Aux[0]);
                putLocker(locker2Aux[1], locker2Aux[2], locker4Aux[0]);
                putLocker(locker3Aux[1], locker3Aux[2], locker1Aux[0]);
                putLocker(locker4Aux[1], locker4Aux[2], locker3Aux[0]);
                break;
        }

        player.setQuantityRotation(player.getQuantityRotation() - 1);

    }

    public boolean existTabletPositionWithoutBird(int row, int column, String color) {
        boolean exist = false;
        if (this.getMatrix()[row][column] != null) {
            switch (color) {
                case "Red":
                    exist = this.getMatrix()[row][column].equals(red);
```

```
                break;
            case "Blue":
                exist = this.getMatrix()[row][column].equals(blue);
                break;
            case "Yellow":
                exist = this.getMatrix()[row][column].equals(yellow);
                break;
            case "Green":
                exist = this.getMatrix()[row][column].equals(green);
                break;
            case "Whatever":
                exist = this.getMatrix()[row][column].equals(red)
                        || this.getMatrix()[row][column].equals(blue)
                        || this.getMatrix()[row][column].equals(yellow)
                        || this.getMatrix()[row][column].equals(green);
                break;
        }
      }
      return exist;
   }

   public boolean connectPositionsHorizontally(Player player, int[] rowColumnMatch) {
      boolean result = true;
      int since;
      int until;
      if (rowColumnMatch[1] < rowColumnMatch[2]) {
         since = rowColumnMatch[1];
         until = rowColumnMatch[2];
      } else {
         since = rowColumnMatch[2];
         until = rowColumnMatch[1];
      }
      for (int i = since; i <= until; i++) {
         if (existTabletPositionWithBird(rowColumnMatch[0], i, player.getColor())) {
            result = true;
         } else if
(noExistColorInPositionOrExistBirdsOfTheOtherPlayers(rowColumnMatch[0], i,
player.getColor())) {
            result = false;
         }
      }

      if (result) {
         for (int column = since; column <= until; column++) {
            putBird(rowColumnMatch[0], column, player);
         }
      }
      return result;
   }
```

```
    public boolean connectPositionsVertically(Player player, int[] columnRowMatch) {
        boolean result = true;
        int since;
        int until;
        if (columnRowMatch[1] < columnRowMatch[2]) {
            since = columnRowMatch[1];
            until = columnRowMatch[2];
        } else {
            since = columnRowMatch[2];
            until = columnRowMatch[1];
        }
        for (int i = since; i <= until; i++) {
            if (existTabletPositionWithBird(i, columnRowMatch[0], player.getColor())) {
                result = true;
            } else if (noExistColorInPositionOrExistBirdsOfTheOtherPlayers(i,
columnRowMatch[0], player.getColor())) {
                result = false;
            }
        }
        if (result) {
            for (int row = since; row <= until; row++) {
                putBird(row, columnRowMatch[0], player);
            }
        }
        return result;
    }

    public void putBird(int row, int column, Player player) {
        String color = player.getColor();
        switch (color) {
            case "Red":
                this.getMatrix()[row][column] = redWithBird;
                break;
            case "Blue":
                this.getMatrix()[row][column] = blueWithBird;
                break;
            case "Yellow":
                this.getMatrix()[row][column] = yellowWithBird;
                break;
            case "Green":
                this.getMatrix()[row][column] = greenWithBird;
                break;
        }
        player.setQuantityBirdsOnBoard(player.getQuantityBirdsOnBoard() + 1);
    }

    public int[] rowWhereTheColorsMatchOf2Positions(int[] rows1, int[] columns1, int[]
rows2,
```

```
            int[] columns2, String colorPlayer) {
        int[] rowColumn1 = new int[2];
        int[] rowColumn2 = new int[2];
        rowColumn1[0] = -1;
        rowColumn1[1] = -1;
        rowColumn2[0] = -1;
        rowColumn2[1] = -1;
        int[] rowColumnsMatch = new int[3];
        rowColumnsMatch[0] = -2;
        int[] sinceUntil = new int[2];

        if (rows1[0] < rows2[1]) {
            sinceUntil[0] = rows1[0];
            sinceUntil[1] = rows2[1];
        } else {
            sinceUntil[0] = rows2[1];
            sinceUntil[1] = rows1[0];
        }

        rowColumn1 = rowColumnWhereTheColorsMatchOf1Position(sinceUntil, columns1,
colorPlayer);
        rowColumn2 = rowColumnWhereTheColorsMatchOf1Position(sinceUntil, columns2,
colorPlayer,
                columns2[0] != columns1[0], columns2[1] != columns1[1]);

        if (positionsAreHorizontal(rowColumn1[0], rowColumn2[0]) && rowColumn1[0] != -1
&& rowColumn2[0] != -1) {
            rowColumnsMatch[0] = rowColumn1[0];
        }
        rowColumnsMatch[1] = rowColumn1[1];
        rowColumnsMatch[2] = rowColumn2[1];
        return rowColumnsMatch;
    }

    public int[] columnWhereTheColorsMatchOf2Positions(int[] rows1, int[] columns1, int[]
rows2,
            int[] columns2, String colorPlayer) {
        int[] rowColumn1 = new int[2];
        int[] rowColumn2 = new int[2];
        rowColumn1[0] = -1;
        rowColumn1[1] = -1;
        rowColumn2[0] = -1;
        rowColumn2[1] = -1;
        int[] columnRowMatch = new int[3];
        columnRowMatch[0] = -2;
        int[] sinceUntil = new int[2];

        if (columns1[0] < columns2[1]) {
            sinceUntil[0] = columns1[0];
```

```
            sinceUntil[1] = columns2[1];
        } else {
            sinceUntil[0] = columns2[0];
            sinceUntil[1] = columns1[1];
        }

        rowColumn1 = rowColumnWhereTheColorsMatchOf1Position(rows1, sinceUntil,
colorPlayer);
        rowColumn2 = rowColumnWhereTheColorsMatchOf1Position(rows2, sinceUntil,
colorPlayer,
                rows2[0] != rows1[0], rows2[1] != rows1[1]);

        if (positionsAreVertical(rowColumn1[1], rowColumn2[1]) && rowColumn1[1] != -1 &&
rowColumn2[1] != -1) {
            columnRowMatch[0] = rowColumn2[1];
        }
        columnRowMatch[1] = rowColumn1[0];
        columnRowMatch[2] = rowColumn1[1];
        return columnRowMatch;
    }

    public int[] rowColumnWhereTheColorsMatchOf1Position(int[] rows1, int[] columns1,
            String colorPlayer, boolean condition1, boolean condition2) {
        int[] rowColumnMatch = new int[2];
        rowColumnMatch[0] = -1;
        rowColumnMatch[1] = -1;
        for (int i = columns1[0]; i <= columns1[1]; i++) {
            if (existColorInPosition(rows1[0], i, colorPlayer) && condition1) {
                rowColumnMatch[0] = rows1[0];
                rowColumnMatch[1] = i;
            } else if (existColorInPosition(rows1[1], i, colorPlayer) && condition2) {
                rowColumnMatch[0] = rows1[1];
                rowColumnMatch[1] = i;
            }
        }
        return rowColumnMatch;

    }

    public int[] rowColumnWhereTheColorsMatchOf1Position(int[] rows1, int[] columns1,
String colorPlayer) {
        int[] rowColumnMatch = new int[2];
        rowColumnMatch[0] = -1;
        rowColumnMatch[1] = -1;
        for (int i = columns1[0]; i <= columns1[1]; i++) {
            if (existColorInPosition(rows1[0], i, colorPlayer)) {
                rowColumnMatch[0] = rows1[0];
                rowColumnMatch[1] = i;
            } else if (existColorInPosition(rows1[1], i, colorPlayer)) {
```

```
            rowColumnMatch[0] = rows1[1];
            rowColumnMatch[1] = i;
        }
    }
    return rowColumnMatch;

}

public boolean positionsAreHorizontal(int row1, int row2) {
    return row1 == row2;
}

public boolean positionsAreVertical(int column1, int column2) {
    return column1 == column2;

}

public boolean extendSense(String sense, int[] rowColumnMatch, Player player) {
    boolean result = true;
    int since = -1;
    boolean found = false;
    int until = -1;

    switch (sense) {
        case "I":
            for (int i = 0; i < 20; i += 2) {
                if (existTablet(rowColumnMatch[0], i) && !found) {
                    since = i;
                    found = true;
                }
            }
            until = rowColumnMatch[1];
            break;
        case "D":
            for (int i = 18; i >= 0; i--) {
                if (existTablet(rowColumnMatch[0], i) && !found) {
                    until = i;
                    found = true;
                }
            }
            since = rowColumnMatch[1];
            break;
        case "A":
            for (int i = 0; i < 20; i += 2) {
                if (existTablet(i, rowColumnMatch[1])
                        && !found) {
                    since = i;
                    found = true;
                }
```

```
            }
            until = rowColumnMatch[0];
            break;
        case "B":
            for (int i = 18; i >= 0; i--) {
                if (existTablet(i, rowColumnMatch[1]) && !found) {
                    until = i;
                    found = true;
                }
            }
            since = rowColumnMatch[0];
            break;
    }
    if (since != -1 && until != -1) {

        if (sense.equals("I") || sense.equals("D")) {
            for (int i = since; i <= until; i++) {
                if (existTabletPositionWithBird(rowColumnMatch[0], i, player.getColor())) {
                    result = true;
                } else if
(noExistColorInPositionOrExistBirdsOfTheOtherPlayers(rowColumnMatch[0], i,
player.getColor())) {
                    result = false;
                }
            }

            if (result) {
                for (int i = since; i <= until; i++) {
                    putBird(rowColumnMatch[0], i, player);
                }
            }
        }

    }

    if (sense.equals("A") || sense.equals("B")) {
        for (int i = since; i <= until; i++) {
            if (existTabletPositionWithBird(i, rowColumnMatch[1], player.getColor())) {
                result = true;
            } else if (noExistColorInPositionOrExistBirdsOfTheOtherPlayers(i,
rowColumnMatch[1], player.getColor())) {
                result = false;
            }
        }
        if (result) {
            for (int i = since; i <= until; i++) {
                putBird(i, rowColumnMatch[1], player);
            }
        }
```

```
        }

        return result;
    }

    public void validCommand(Player player) {
        this.setValidEnteredCommand(true);
        player.setPlayedTurns(player.getPlayedTurns() + 1);
    }

    public void invalidCommand() {
        this.setValidEnteredCommand(false);
        URL url = getClass().getClassLoader().getResource("resources/Error.wav");

        AudioClip clip = newAudioClip(url);
        clip.play();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
        }
    }

    public String[][] getMatrix() {
        return matrix;
    }

    public void setMatrix(String[][] matrix) {
        this.matrix = matrix;
    }

    public int getQuantityPlacedTablets() {
        return quantityPlacedTablets;
    }

    public void setQuantityPlacedTablets(int quantityPlacedTablets) {
        this.quantityPlacedTablets = quantityPlacedTablets;
    }

    public boolean isValidEnteredCommand() {
        return validEnteredCommand;
    }

    public void setValidEnteredCommand(boolean validEnteredCommand) {
        this.validEnteredCommand = validEnteredCommand;
    }

}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package obligatorio;

import java.util.ArrayList;
import java.util.Collections;

public class Game {

    private int quantityPlayers;
    private int quantityBirdsPlayer;
    private int quantityRotationPlayer;
    private int quantityTablets;
    private Board board;
    private Player winner;
    private ArrayList<Player> playersList;
    private int wayToFinish;
    private String[] wayToFinishDescription;
    private int quantityTurnsToFinish;
    private ArrayList<String> colors = new ArrayList<>();
    private int currentTurn;

    public Game() {
        this.quantityPlayers = 2;
        this.quantityBirdsPlayer = 45;
        this.quantityRotationPlayer = 5;
        this.quantityTablets = 25;
        this.playersList = new ArrayList<>();
        this.board = new Board();
        this.wayToFinish = 3;
        this.wayToFinishDescription = new String[4];
        this.wayToFinishDescription[1] = "When the last tablet is placed on the 5*5 board.";
        this.wayToFinishDescription[2] = "When all the birds of a player are placed.";
        this.wayToFinishDescription[3] = "When a player's turn ends.";
        this.quantityTurnsToFinish = 10;
        this.colors.add("Red");
        this.colors.add("Blue");
        this.colors.add("Yellow");
        this.colors.add("Green");
        this.currentTurn = -1;
    }

    public boolean gameOver() {
```

```
        boolean over = false;

        if (this.getWayToFinish() == 1) {
            if (getBoard().getQuantityPlacedTablets() == getQuantityTablets()) {
                over = true;
            }
        }
        if (this.getWayToFinish() == 2) {
            for (int i = 0; i < getPlayersList().size(); i++) {
                if (getPlayersList().get(i).getQuantityBirdsOnBoard() ==
getQuantityBirdsPlayer()) {
                    over = true;
                }
            }
        }
        if (this.getWayToFinish() == 3) {
                if (getPlayersList().get(getPlayersList().size()-1).getPlayedTurns() ==
getQuantityTurnsToFinish()) {
                    over = true;
            }
        }
        return over;
    }

    /**
     *
     * @param player: Player that select X for to way finish
     * @return
     */
    public ArrayList<Player> showWinnerX(Player player) {
        ArrayList<Player> winnersPlayers = new ArrayList<>();
        for (int i = 0; i < getPlayersList().size(); i++) {
            if (!getPlayersList().get(i).equals(player)) {
                winnersPlayers.add(getPlayersList().get(i));
            }
        }
        return winnersPlayers;
    }

    public ArrayList<Player> showWinner() {
        ArrayList<Player> winnersPlayers = new ArrayList<>();
        int maxBirds = 0;
        for (int i = 0; i < getPlayersList().size(); i++) {
            if (getPlayersList().get(i).getQuantityBirdsOnBoard() > maxBirds) {
                maxBirds = getPlayersList().get(i).getQuantityBirdsOnBoard();
            }
        }
         for (int i = 0; i < getPlayersList().size(); i++) {
             if(getPlayersList().get(i).getQuantityBirdsOnBoard() == maxBirds){
```

```
            winnersPlayers.add(getPlayersList().get(i));
        }
        }
        return winnersPlayers;

    }

    public Player nextTurn() {
        currentTurn++;
        if (currentTurn > getPlayersList().size() - 1) {
            currentTurn = 0;
        }
        return getPlayersList().get(currentTurn);
    }

    public void assignColorToPlayers() {
        Collections.shuffle(this.getPlayersList());
        int colorPosition = 0;
        for (int i = 0; i < this.getPlayersList().size(); i++) {
            this.getPlayersList().get(i).setColor(colors.get(colorPosition));
            colorPosition++;

        }
    }

    public int getQuantityPlayers() {
        return quantityPlayers;
    }

    public void setQuantityPlayers(int quantityPlayers) {
        this.quantityPlayers = quantityPlayers;
    }

    public int getQuantityBirdsPlayer() {
        return quantityBirdsPlayer;
    }

    public void setQuantityBirdsPlayer(int quantityBirdsPlayer) {
        this.quantityBirdsPlayer = quantityBirdsPlayer;
    }

    public int getQuantityRotationPlayer() {
        return quantityRotationPlayer;
    }

    public void setQuantityRotationPlayer(int quantityRotationPlayer) {
        this.quantityRotationPlayer = quantityRotationPlayer;
    }
```

```
public int getQuantityTablets() {
    return quantityTablets;
}

public void setQuantityTablets(int quantityTablets) {
    this.quantityTablets = quantityTablets;
}

public Board getBoard() {
    return board;
}

public void setBoard(Board board) {
    this.board = board;
}

public Player getWinner() {
    return winner;
}

public void setWinner(Player winner) {
    this.winner = winner;
}

public ArrayList<Player> getPlayersList() {
    return playersList;
}

public void setPlayersList(ArrayList<Player> playersList) {
    this.playersList = playersList;
}

public int getWayToFinish() {
    return wayToFinish;
}

public void setWayToFinish(int wayToFinish) {
    this.wayToFinish = wayToFinish;
}

public int getQuantityTurnsToFinish() {
    return quantityTurnsToFinish;
}

public void setQuantityTurnsToFinish(int quantityTurnsToFinish) {
    this.quantityTurnsToFinish = quantityTurnsToFinish;
}

public String[] getWayToFinishDescription() {
```

```
        return wayToFinishDescription;
    }

    public void setWayToFinishDescription(String[] wayToFinishDescription) {
        this.wayToFinishDescription = wayToFinishDescription;
    }

    @Override
    public String toString() {
        return "Game " + "quantityPlayers: " + quantityPlayers + ", quantityBirdsPlayer: " +
quantityBirdsPlayer + ", quantityRotationPlayer: " + quantityRotationPlayer + ",
quantityTablets: " + quantityTablets + ", board: " + board + ", winner: " + winner + ",
playersList: " + playersList + ", wayToFinish: " + wayToFinish + ", maxTurnsToFinish: " +
quantityTurnsToFinish;
    }

}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package obligatorio;

import java.util.ArrayList;

public class Company {

    ArrayList<Player> playersList;
    ArrayList<Game> gamesList;

    public Company() {
        this.playersList = new ArrayList<>();
        this.gamesList = new ArrayList<>();
    }

    public boolean existPlayer(String alias){
        boolean exist = false;
        if(this.getPlayersList().size()!=0){
     for(int i=0; i< this.getPlayersList().size(); i++){
         exist = this.getPlayersList().get(i).getAlias().equals(alias);
     } }
     return exist;
    }
```

```
    public Player createPlayer(String name, String alias, int age){
        Player player = new Player(name, alias, age);
        this.getPlayersList().add(player);
        return player;
    }

    public ArrayList<Player> getPlayersList() {
        return playersList;
    }

    public void setPlayersList(ArrayList<Player> playersList) {
        this.playersList = playersList;
    }

    public ArrayList<Game> getGamesList() {
        return gamesList;
    }

    public void setGamesList(ArrayList<Game> gamesList) {
        this.gamesList = gamesList;
    }


}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package obligatorio;

public class Player implements Comparable<Player>, Cloneable {

    private String name;
    private String alias;
    private String color;
    private int age;
    private int quantityRotation;
    private int quantityBirdsOnBoard;
    private int[] quantityOpponents;
    private int playedTurns;
    private int quantityGamesPlayed;
```

```
public Player() {
    super();
}

public Player(String name, String alias, int age) {
    this.name = name;
    this.alias = alias;
    this.age = age;
    this.quantityOpponents = new int[4];
    this.playedTurns = 0;
    this.quantityGamesPlayed = 0;
}

public int getQuantityGamesPlayed() {
    return quantityGamesPlayed;
}

public void setQuantityGamesPlayed(int quantityGamesPlayed) {
    this.quantityGamesPlayed = quantityGamesPlayed;
}

public String getName() {
    return name;
}

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

public String getAlias() {
    return alias;
}

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

public String getColor() {
    return color;
}

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

public int getAge() {
    return age;
}
```

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

public int getQuantityRotation() {
    return quantityRotation;
}

public void setQuantityRotation(int quantityRotation) {
    this.quantityRotation = quantityRotation;
}

public int getQuantityBirdsOnBoard() {
    return quantityBirdsOnBoard;
}

public void setQuantityBirdsOnBoard(int quantityBirdsOnBoard) {
    this.quantityBirdsOnBoard = quantityBirdsOnBoard;
}

public int[] getQuantityOpponents() {
    return quantityOpponents;
}

public void setQuantityOpponents(int[] quantityOpponents) {
    this.quantityOpponents = quantityOpponents;
}

public int getPlayedTurns() {
    return playedTurns;
}

public void setPlayedTurns(int playedTurns) {
    this.playedTurns = playedTurns;
}

@Override
public boolean equals(Object o) {
    Player aux = (Player) o;
    return this.getAlias().equals(aux.getAlias());
}

public int compareTo(Player t) {
    return t.getName().compareTo(this.getName());

}

@Override
```

```
    public String toString() {
        return alias;
    }
}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package obligatorio;

import java.util.Comparator;


public class OrderGamesPlayed implements Comparator<Player>{



    @Override
    public int compare(Player o1, Player o2) {
        return o2.getQuantityGamesPlayed() - o1.getQuantityGamesPlayed();
    }

}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package obligatorio;

import java.util.ArrayList;
import java.util.Collections;
import java.util.InputMismatchException;
import java.util.Iterator;
import java.util.Scanner;

public class Data {
```

```
    static Game game = new Game();
    static char letter = 65;
    static int auxRow = 0;
    static int quantityPutColorsTabletRow1 = 0;
    static int quantityPutColorsTabletRow2 = 0;
    static Player player = new Player();

    public static void main(String[] args) {
        Company company = new Company();

        int option = 0;
        String showSpecialSettings = "";
        int optionSpecialSettings = 0;
        String command = "";
        int finish = 0;

        String menu = "\n\n        MENU\n";
        menu += "1. Define a special settings.\n";
        menu += "2. Create a new player.\n";
        menu += "3. Select a player created.\n";
        menu += "4. Play.\n";
        menu += "5. List by player.\n";
        menu += "6. End.\n";
        do {

            System.out.println(menu);
            option = readInt("Enter an option: ");

            switch (option) {
                case 1:
                    System.out.println("\nDEFINE A SPECIAL SETTINGS: \n");
                    do {
                        showSpecialSettings = "1. Quantity of players: "
                                + game.getQuantityPlayers() + " (Valid values between 2 and 4).\n";
                        showSpecialSettings += "2. Quantity of birds per player: "
                                + game.getQuantityBirdsPlayer() + " (Valid values between 5 and
45).\n";
                        showSpecialSettings += "3. Quantity of rotation per player: "
                                + game.getQuantityRotationPlayer() + " (Valid values between 0 and
5).\n";
                        showSpecialSettings += "4. Quantity of maximum tablets for the board: "
                                + game.getQuantityTablets() + " (Valid values between 5 and 25).\n";
                        showSpecialSettings += "5. Way to finish: "
                                + game.getWayToFinishDescription()[game.getWayToFinish()]
                                + "  (3 valid ways to finish).\n";
                        showSpecialSettings += "6. Return to main menu.";

                        System.out.println(showSpecialSettings);
                        optionSpecialSettings = readInt("\nEnter an option: ");
```

```
                    switch (optionSpecialSettings) {
                        case 1:
                            game.setQuantityPlayers(readAndValidateInt(2, 4,
                                    "\nEnter the quantity of players: "));
                            System.out.println("Quantity of players it was correctly modified.");
                            break;
                        case 2:
                            game.setQuantityBirdsPlayer(readAndValidateInt(5, 45,
                                    "\nEnter the quantity of birds per player: "));
                            System.out.println("Quantity of birds per player it was correctly
modified.");
                            break;
                        case 3:
                            game.setQuantityRotationPlayer(readAndValidateInt(0, 5,
                                    "\nEnter the quantity of rotation per player: "));
                            System.out.println("Quantity of rotation per player it was correctly
modified.");
                            break;
                        case 4:
                            game.setQuantityTablets(readAndValidateInt(5, 25,
                                    "\nEnter the quantity of maximum tablets for the board: "));
                            System.out.println("Quantity of maximum tablets for the board it was
correctly modified.");
                            break;
                        case 5:
                            for (int i = 1; i < game.getWayToFinishDescription().length; i++) {
                                System.out.println(i + ". " + game.getWayToFinishDescription()[i]);
                            }
                            game.setWayToFinish(readAndValidateInt(1, 3,
                                    "\nEnter a number of way to finish: "));
                            if (game.getWayToFinish() == 3) {
                                game.setQuantityTurnsToFinish(readAndValidateInt(1, 50,
                                        "\nEnter a number of turns to finish: "));
                            }
                            System.out.println("Way to finish it was correctly modified.");
                            break;
                        case 6:
                            break;
                        default:
                            System.out.println("\nMust enter an option between 1 and 6: ");
                            break;
                    }
                } while (optionSpecialSettings != 6);
                break;
            case 2:
                System.out.println("\nCREATE A NEW PLAYER\n");
                String name = readString("Enter a name: ");

                String alias = readString("Enter an alias: ");
```

```
                while (company.existPlayer(alias)) {
                    System.out.println("This alias has already been "
                            + "registered, please try another.\n");
                    alias = readString("Enter an alias: ");
                }

                int age = readAndValidateInt(5, 120, "Enter an age: ");

                player = company.createPlayer(name, alias, age);
                System.out.println("\nPlayer " + alias + " it was correctly created.");
                if (game.getPlayersList().size() <= game.getQuantityPlayers()) {
                    int select = readAndValidateInt(1, 2, "Select this player for"
                            + " this game? \n1. Yes   -   2. No: ");
                    if (select == 1) {
                        game.getPlayersList().add(player);
                    }
                }
                break;
            case 3:
                String aliasForSelect;
                System.out.println("\nSELECT A PLAYER CREATED\n");
                if (company.getPlayersList().isEmpty()) {
                    System.out.println("There are no players created\n");
                } else if (game.getPlayersList().size() < game.getQuantityPlayers()) {
                    ArrayList<String> playersForSelect = new ArrayList<>();
                    for (int i = 0; i < company.getPlayersList().size(); i++) {
                        playersForSelect.add(company.getPlayersList().get(i).getAlias());
                    }

                    Iterator<String> it = playersForSelect.iterator();
                    while (it.hasNext()) {
                        aliasForSelect = it.next();
                        Player aux = new Player();
                        aux.setAlias(aliasForSelect);
                        if (game.getPlayersList().contains(aux)) {
                            it.remove();
                        }
                    }
                    int quantityForSelect = playersForSelect.size();
                    for (int i = 0; i < quantityForSelect; i++) {
                        aliasForSelect = playersForSelect.get(i);
                        System.out.println((i + 1) + "- " + aliasForSelect);
                    }
                    int number = 0;
                    if (quantityForSelect != 0) {
                        number = readAndValidateInt(1, quantityForSelect, "0- Return to main
menu\n.Enter a number: ");
                    } else {
```

```
                    System.out.println("There are no players created who haven't been
selected.");
                }

                if (number > 0) {
                    for (int i = 0; i < company.getPlayersList().size(); i++) {
                        String aliasSelected = playersForSelect.get(number - 1);
                        if (company.getPlayersList().get(i).getAlias().equals(aliasSelected)) {
                            player = company.getPlayersList().get(i);
                        }
                    }
                    game.getPlayersList().add(player);
                }
            } else if (game.getPlayersList().size() == game.getQuantityPlayers()) {
                System.out.println(game.getQuantityPlayers() + " players "
                        + "have already been selected.\nIf you want to select more will must
use the settings special");
            } else {
                System.out.println("The quantity of players selected(" +
game.getPlayersList().size() + ")"
                        + " is more than the quantity players configured(" +
game.getQuantityPlayers() + ").");
            }
            break;
        case 4:
            System.out.println("\nPLAY\n");
            if (game.getPlayersList().size() == game.getQuantityPlayers()) {
                int opponents = game.getPlayersList().size() - 1;
                for (int i = 0; i < game.getPlayersList().size(); i++) {
                    player = game.getPlayersList().get(i);
                    player.getQuantityOpponents()[opponents]++;
                    player.setQuantityRotation(game.getQuantityRotationPlayer());
                    player.setQuantityGamesPlayed(player.getQuantityGamesPlayed() + 1);
                }
                game.assignColorToPlayers();
                company.getGamesList().add(game);
                String[] commandArray;

                do {
                    String[] rowColumnpos1;
                    String[] rowColumnpos2;
                    String row1;
                    String column1;
                    String row2;
                    String column2;
                    String[] colors;
                    int[] rows1;
                    int[] columns1;
                    int[] rows2;
```

```
                    int[] columns2;
                    int[] rowColumnMatch = new int[3];
                    int[] columnRowMatch = new int[3];
                    String sense;
                    String angle;

                    if (game.getBoard().isValidEnteredCommand()) {
                        player = game.nextTurn();
                        showBoard();
                    }
                    command = readString("Enter a command to play: ");
                    commandArray = command.split(" ");

                    switch (commandArray[0].toUpperCase()) {

                        case "R":
                            if (validateCommand(3, commandArray)) {
                                rowColumnpos1 = commandArray[1].split("");
                                angle = commandArray[2];
                                if (rowColumnpos1.length != 2) {
                                    game.getBoard().invalidCommand();
                                    System.out.println("Column or row not valid. Must enter a row
between A-J and column between 1-10.\n");
                                } else {
                                    row1 = rowColumnpos1[0];
                                    column1 = rowColumnpos1[1];
                                    if ((row1.equals("E") && column1.equals("5")) ||
(row1.equals("E") && column1.equals("6"))) {
                                        game.getBoard().invalidCommand();
                                        System.out.println("Can not rotate the tablets E5 and
E6.\n");
                                    } else {
                                        rows1 = game.getBoard().convertRow(row1);
                                        columns1 = game.getBoard().convertColumn(column1);
                                        if (game.getBoard().validateAngle(angle)) {
                                            if
(game.getBoard().existTabletPositionWithoutBird(rows1[0], columns1[0], "Whatever")) {
                                                if (player.getQuantityRotation() > 0) {
                                                    game.getBoard().rotateTablet(rows1, columns1,
angle, player);
                                                } else {
                                                    game.getBoard().invalidCommand();
                                                    System.out.println("The player does not have
rotation tokens available.\n");
                                                }
                                            } else {
                                                game.getBoard().invalidCommand();
                                                System.out.println("Must indicate a position where
there is a tablet without birds.\n");
```

```
                                    }
                                } else {
                                    game.getBoard().invalidCommand();
                                    System.out.println("The angle must be 90, 180 o 270.\n");
                                }
                            }
                        }
                    } else {
                        game.getBoard().invalidCommand();
                        System.out.println("Must enter a valid format of commands.\n");
                    }
                    break;
                case "C":
                    if (validateCommand(3, commandArray)) {
                        rowColumnpos1 = commandArray[1].split("");
                        rowColumnpos2 = commandArray[2].split("");
                        if (rowColumnpos1.equals(rowColumnpos2)) {
                            game.getBoard().invalidCommand();
                            System.out.println("For the command C must enter differents
positions.\n");
                        } else if (rowColumnpos1.length != 2 || rowColumnpos2.length !=
2) {
                            game.getBoard().invalidCommand();
                            System.out.println("Column or row not valid. Must enter a row
between A-J and column between 1-10.\n");
                        } else {
                            row1 = rowColumnpos1[0];
                            column1 = rowColumnpos1[1];
                            row2 = rowColumnpos2[0];
                            column2 = rowColumnpos2[1];
                            rows1 = game.getBoard().convertRow(row1);
                            columns1 = game.getBoard().convertColumn(column1);
                            rows2 = game.getBoard().convertRow(row2);
                            columns2 = game.getBoard().convertColumn(column2);
                            rowColumnMatch[0] = -1;
                            columnRowMatch[0] = -1;
                            if (rows1.length == 0 || columns1.length == 0 || rows2.length
== 0 || columns2.length == 0) {
                                game.getBoard().invalidCommand();
                                System.out.println("Column or row not valid. Must enter a
row between A-J and column between 1-10.\n");
                            } else if (game.getBoard().existTablet(rows1[0], columns1[0])
                                    && game.getBoard().existTablet(rows2[0], columns2[0])) {
                                columnRowMatch =
game.getBoard().columnWhereTheColorsMatchOf2Positions(rows1, columns1, rows2,
columns2, player.getColor());
                                rowColumnMatch =
game.getBoard().rowWhereTheColorsMatchOf2Positions(rows1, columns1, rows2,
columns2, player.getColor());
```

```
                                if (rowColumnMatch[0] != -2 || columnRowMatch[0] != -2) {
                                    if (rowColumnMatch[0] != -1 && rowColumnMatch[0] !=
-2) {
                                        if (!(game.getBoard().connectPositionsHorizontally(
                                                player, rowColumnMatch))) {
                                            game.getBoard().invalidCommand();
                                            System.out.println("Could not connect the positions,
there are no tablets on the road or there are birds on the road.\n");
                                        } else {
                                            game.getBoard().validCommand(player);
                                        }
                                    }
                                    if (columnRowMatch[0] != -1 && columnRowMatch[0] !=
-2) {
                                        if (!(game.getBoard().connectPositionsVertically(player,
columnRowMatch))) {
                                            game.getBoard().invalidCommand();
                                            System.out.println("Could not connect the positions,
there are no tablets on the road or there are birds on the road.\n");
                                        } else {
                                            game.getBoard().validCommand(player);
                                        }
                                    }
                                } else {
                                    game.getBoard().invalidCommand();
                                    System.out.println("Must enter 2 positions with player
color are aligned..\n");
                                }
                            } else {
                                game.getBoard().invalidCommand();
                                System.out.println("Must enter 2 positions with tablets.\n");
                            }
                        }
                    } else {
                        game.getBoard().invalidCommand();
                        System.out.println("Must enter a valid format of commands.\n");
                    }
                    break;

                case "P":
                    if (validateCommand(2, commandArray)) {
                        rowColumnpos1 = commandArray[1].split("");
                        if (rowColumnpos1.length == 2) {
                            rows1 = game.getBoard().convertRow(rowColumnpos1[0]);
                            columns1 =
game.getBoard().convertColumn(rowColumnpos1[1]);
                            if (rows1[0] != -1 && columns1[0] != -1) {
```

```
                                if (game.getBoard().validatePutTablet(rows1[0],
columns1[0], game.getQuantityTablets())) {
                                    game.getBoard().putTablet(rows1[0], columns1[0]);
                                    game.getBoard().validCommand(player);
                                } else {
                                    game.getBoard().invalidCommand();
                                    System.out.println("Could not put the tablet.\n");
                                }
                            } else {
                                game.getBoard().invalidCommand();
                                System.out.println("Column or row not valid. Must enter a
row between A-J and column between 1-10.\n");
                            }
                        } else {
                            game.getBoard().invalidCommand();
                            System.out.println("Column or row not valid. Must enter a row
between A-J and column between 1-10.\n");
                        }
                    } else {
                        game.getBoard().invalidCommand();
                        System.out.println("Must enter a valid format of commands.\n");
                    }
                    break;
                case "E":
                    if (validateCommand(3, commandArray)) {
                        sense = commandArray[1].toUpperCase();
                        rowColumnpos1 = commandArray[2].split("");
                        if (rowColumnpos1.length != 2) {
                            game.getBoard().invalidCommand();
                            System.out.println("Column or row not valid. Must enter a row
between A-J and column between 1-10.\n");
                        } else {
                            rows1 = game.getBoard().convertRow(rowColumnpos1[0]);
                            columns1 =
game.getBoard().convertColumn(rowColumnpos1[1]);
                            if (game.getBoard().validateSense(sense)) {
                                if (rows1[0] != -1 && columns1[0] != -1) {
                                    rowColumnMatch =
game.getBoard().rowColumnWhereTheColorsMatchOf1Position(rows1, columns1,
player.getColor());
                                    if (rowColumnMatch[0] != -1) {
                                        if (game.getBoard().extendSense(sense,
rowColumnMatch, player)) {
                                            game.getBoard().validCommand(player);
                                        } else {
                                            game.getBoard().invalidCommand();
                                            System.out.println("Could not extend, there are no
tablets on the road or there are birds on the road.\n");
                                        }
```

```
                                    } else {
                                        game.getBoard().invalidCommand();
                                        System.out.println("Must enter a position where the
player's color is\n");
                                    }

                                } else {
                                    game.getBoard().invalidCommand();
                                    System.out.println("Column or row not valid. Must enter a
row between A-J and column between 1-10.\n");
                                }
                            } else {
                                game.getBoard().invalidCommand();
                                System.out.println("Must enter a valid format of sense.\n");
                            }
                        }
                    } else {
                        game.getBoard().invalidCommand();
                        System.out.println("Must enter a valid format of commands.\n");
                    }

                    break;
                case "PM":
                    if (validateCommand(3, commandArray)) {
                        rowColumnpos1 = commandArray[1].split("");
                        if (rowColumnpos1.length != 2) {
                            game.getBoard().invalidCommand();
                            System.out.println("Column or row not valid. Must enter a row
between A-J and column between 1-10.\n");
                        } else {
                            rows1 = game.getBoard().convertRow(rowColumnpos1[0]);
                            columns1 =
game.getBoard().convertColumn(rowColumnpos1[1]);
                            if (game.getBoard().existTablet(rows1[0], columns1[0])) {
                                game.getBoard().invalidCommand();
                                System.out.println("You must enter a position where there
are no tablets.\n");
                            } else {
                                colors = commandArray[2].split("");
                                colors = game.getBoard().convertColors(colors);
                                if (!game.getBoard().validateListColors(colors)) {
                                    game.getBoard().invalidCommand();
                                    System.out.println("Must enter a valid colors.\n");
                                } else {
                                    game.getBoard().putTablet(rows1[0], columns1[0],
                                            colors[0], colors[1], colors[2], colors[3]);
                                    game.getBoard().validCommand(player);

                                }
```

```
                                }
                            }
                        } else {
                            game.getBoard().invalidCommand();
                            System.out.println("Must enter a valid format of commands.\n");
                        }

                        break;
                    case "X":
                        finish = readAndValidateInt(1, 2, "Do you want to finish?\n1- Yes -
2-No. ");
                        break;
                    default:
                        game.getBoard().invalidCommand();
                        System.out.println("Must enter a valid command.\n");
                        break;
                    }

                } while (!game.gameOver() && finish == 0);
                for (int i = 0; i < game.getPlayersList().size(); i++) {
                    Player auxPlayer = game.getPlayersList().get(i);

auxPlayer.setQuantityGamesPlayed(auxPlayer.getQuantityGamesPlayed() + 1);
                }
                System.out.println("\nGame over.\n\nThe winner(s) are:\n");
                if (commandArray[0].toUpperCase().equals("X")) {
                    ArrayList<Player> winners;
                    winners = game.showWinnerX(player);
                    for (int i = 0; i < winners.size(); i++) {
                        System.out.println(" - " + winners.get(i));
                    }
                } else {
                    ArrayList<Player> listWinners = game.showWinner();
                    for (int i = 0; i < listWinners.size(); i++) {
                        System.out.println("- " + listWinners.get(i));
                    }
                }

            } else {
                System.out.println("The quantity players configured("
                        + game.getQuantityPlayers()
                        + ") isn't the same than the quantity players "
                        + "selected.(" + game.getPlayersList().size()
                        + ")\nSelect all the players again.");
            }
            Iterator<Player> it = game.getPlayersList().iterator();
            while (it.hasNext()) {
                player = it.next();
                it.remove();
```

```
                }
                break;

            case 5:
                System.out.println("\nLIST BY PLAYER\n");
                Collections.sort(company.getPlayersList(), new OrderGamesPlayed());
                if (company.getPlayersList().isEmpty()) {
                    System.out.println("There are no players created\n");
                } else {
                    for (int i = 0; i < company.getPlayersList().size(); i++) {
                        player = company.getPlayersList().get(i);
                        System.out.println("\n\n- " + player + ": ");
                        for (int j = 1; j < player.getQuantityOpponents().length; j++) {
                            int auxOpponents = player.getQuantityOpponents()[i];
                            System.out.print("\n  " + j + " opponents: " + auxOpponents);
                        }
                    }
                }
                break;
            case 6:
                break;
            default:
                System.out.println("Must enter an option between 1 and 6: ");
                break;
        }

    } while (option
            != 6);
}

public static void showBoard() {
    letter = 65;
    auxRow = 0;

    System.out.print("\n  1  2  3  4  5  6  7  8  9  10\n");
    for (int i = 0; i < 10; i++) {
        System.out.print("\u001B[0m");
        drawRow();
        drawColumns();
        quantityPutColorsTabletRow1 = 0;
        quantityPutColorsTabletRow2 = 0;
        auxRow++;
    }
    drawRow();

    System.out.println("\n Player turn: " + player + " - color: " + player.getColor());
    System.out.println(" Quantity birds on board: " + player.getQuantityBirdsOnBoard());
    String formatCommands = "\n Format valid of commands:\n";
```

```
        formatCommands += " R RowColumn Angle (angle is 90, 180 or 270) (to rotate
empty tablet) \n";
        formatCommands += " C RowColumn RowColumn (to connect 2 positions) \n";
        formatCommands += " P RowColumn (to put tablet) \n";
        formatCommands += " E Sense RowColumn (to extend, sense is I, D, A, B) \n";
        formatCommands += " PM (To put manually a tablet)\n";
        formatCommands += " X (to finish)\n";
        System.out.println(formatCommands);

    }

    public static boolean validateCommand(int quantityData, String[] command) {
        boolean valid = true;
        if (quantityData != command.length) {
            valid = false;
        }
        return valid;
    }

    public static void drawRow() {
        for (int i = 0; i < game.getBoard().getMatrix().length; i = i + 2) {
            if (i == 0) {
                System.out.print(" +");
            }
            System.out.print("--+");
        }
        System.out.print("\n");

    }

    public static void drawColumns() {
        quantityPutColorsTabletRow1 = 0;
        for (int j = 0; j < 20; j++) {

            if (j == 0) {
                System.out.print(letter + "|");
                letter++;
            }

            if (game.getBoard().getMatrix()[auxRow][j] != null) {
                System.out.print(game.getBoard().getMatrix()[auxRow][j]);
                quantityPutColorsTabletRow1++;
                if (quantityPutColorsTabletRow1 == 2) {
                    System.out.print("|");
                    quantityPutColorsTabletRow1 = 0;
                }
            } else if (j % 2 == 0) {
                System.out.print("  |");
            }
```

```
    }
    auxRow++;
    quantityPutColorsTabletRow2 = 0;
    System.out.print("\n");
    for (int j = 0; j < 20; j++) {

        if (j == 0) {
            System.out.print(" |");
        }

        if (game.getBoard().getMatrix()[auxRow][j] != null) {
            System.out.print(game.getBoard().getMatrix()[auxRow][j]);
            quantityPutColorsTabletRow2++;
            if (quantityPutColorsTabletRow2 == 2) {
                System.out.print("|");
                quantityPutColorsTabletRow2 = 0;
            }
        } else if (j % 2 == 0) {
            System.out.print("  |");
        }
    }
    System.out.print("\n");

}

/**
 * Method that reads a value and validates that it is integer
 *
 * @param message
 * @return value valid
 */
public static int readInt(String message) {
    Scanner in = new Scanner(System.in);
    System.out.print(message);
    int value = -1;
    while (value == -1) {
        try {
            value = in.nextInt();

        } catch (InputMismatchException e) {
            System.out.print("Value not valid.\n" + message);
            in.nextLine();
        }
    }
    return value;
}

/**
```

```
    * Method that reads a value, validates that it is integer and that it is in
    * range
    *
    * @param min
    * @param max
    * @param message
    * @return value valid
    */
   public static int readAndValidateInt(int min, int max, String message) {
       int value = readInt(message);
       while (!intIsValid(min, max, value)) {
           System.out.println("The value must be between " + min + " and " + max);
           value = readInt(message);
       }
       return value;
   }

   /**
    * Method validates that integer is in range
    *
    * @param min
    * @param max
    * @param num
    * @return value valid
    */
   public static boolean intIsValid(int min, int max, int num) {
       return num >= min && num <= max;
   }

   public static String readString(String message) {
       Scanner in = new Scanner(System.in);
       String value = "";
       System.out.print(message);
       do {
           value = in.nextLine();
       } while (value.isEmpty());

       return value;
   }
}
```