Connect Four

Grigori Zogka
40418304
Module: SET11102 2019-0 TR1 00

Functionality

The current application is functioning as two-player game. There are two players, “Player X” and “Player O”, that are being representing in the game board with the subsequent letters (letter “x” and letter “o”). The first move always belongs to the “Player X”. After every valid move that “Player X” makes, “Player O” takes turn.

Each player choses one of the seven available columns in order to place his piece. Each piece falls straight down, occupying the first available space within the column.

The objective of the game is for each Player to connect four of his pieces to each other vertically, horizontally or diagonally before the opponent.

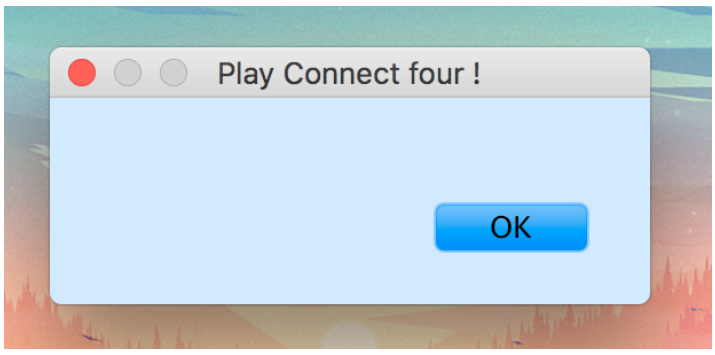
The game can have three deferent outcomes:

1. Player X wins.
2. Player O wins.
3. No Player wins as there are no more available valid moves.

The Game

Welcome Message.

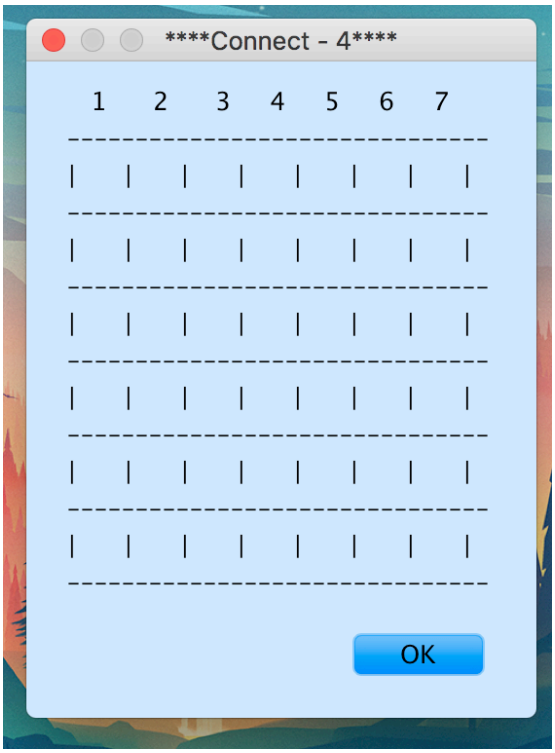
When starting the application the users receive a welcome message prompting them to start the game.



The users must select “OK” in order to proceed to actual the game.

The Game board.

After the users select “OK” they can see the empty game board. This allows the players to familiarise with the format of the game board.



The lines on the board set a seven columns and six row grid.

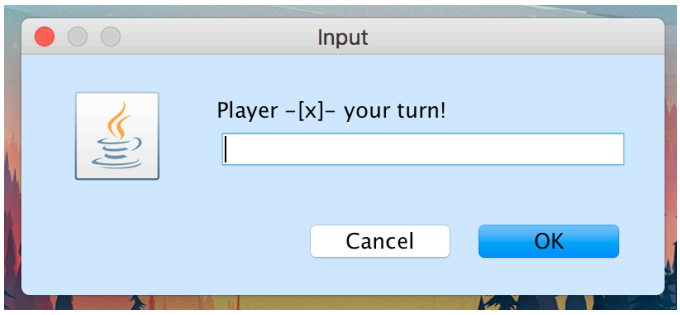
In the beginning of each column, the relevant identification number has been placed in order to allow the users to easily identify each column. Additionally the numbers represent the keys that the users need to use in order to place their pieces in the relevant columns.

The users must select “OK” in order to proceed to the next stage of the game.

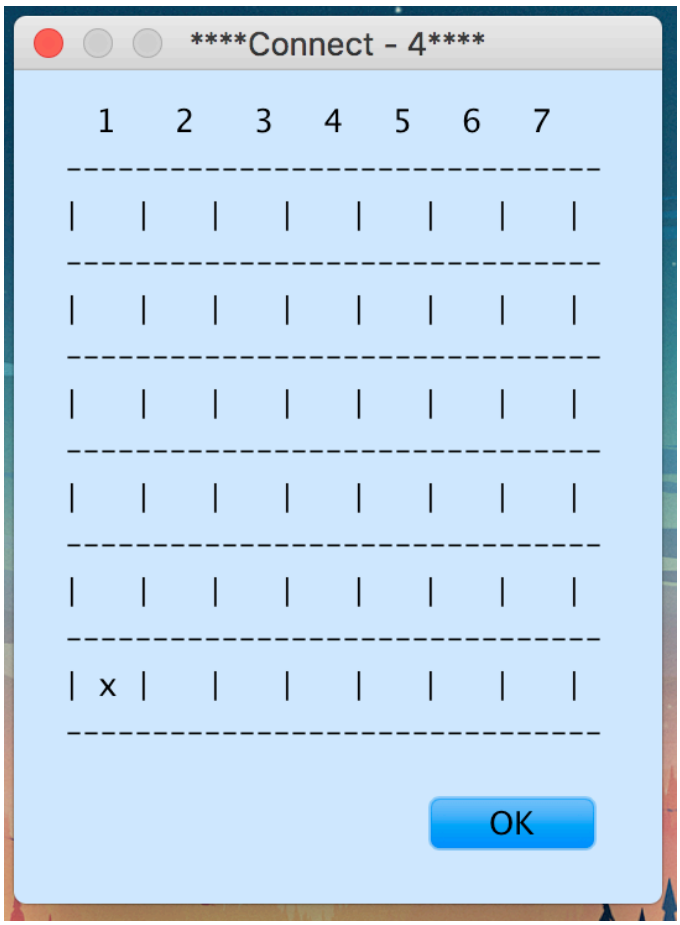
Users place their pieces I the Grid

In the next stage the users are being asked to make a move and place a piece in one of the available columns (columns one to seven).

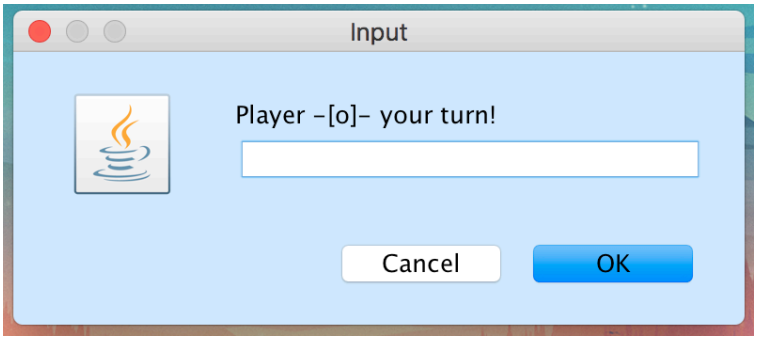
Played “X” plays first and providing that he makes a valid move Player “O” follows.
T
he player must type the column that he wishes to place his piece to occupy and select “OK in order to confirm his choice.



After a player has successfully confirmed the his choice then his piece in being placed in the board - the game board appears with the piece placed in the requested column.



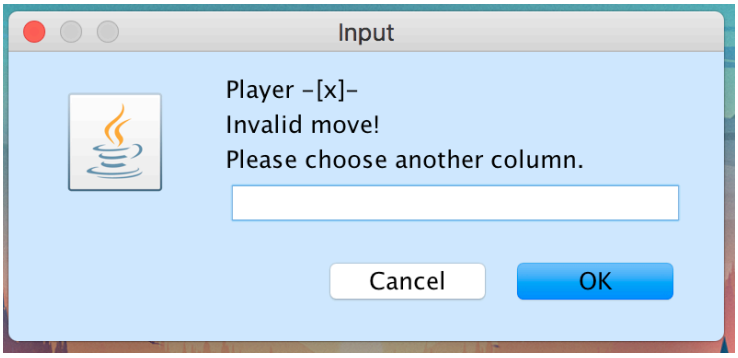
The users must select “OK” in order to proceed to the next stage where Player “O” is being asked to place his piece.



Valid Moves

The users must place their pieces within the boundaries of the Grid. This means that the can only use keys 1 to 7 in order to place their piece and they cannot place a piece in a column if that column if full.

If any of the above is true then the receive the below message prompting them to use another column.



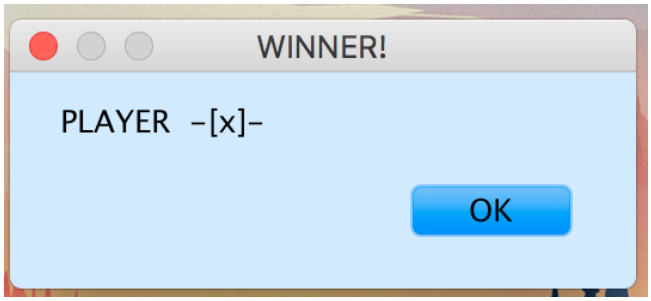
The user must choose another valid column and select “OK” in order to proceed.

Winner

A player wins places when he manages to place four pieces next o each other:

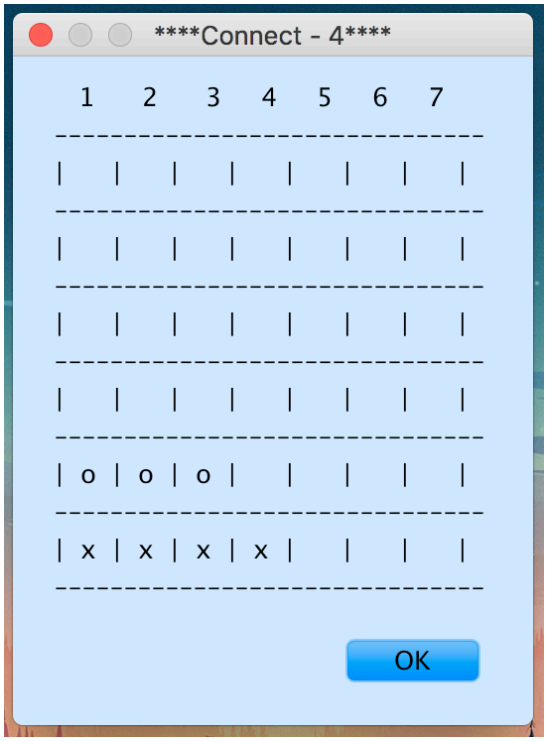
- vertically,
- horizontally
- diagonally (left to right or right to left)

When the above conditions have been met the game declares the winner.



The user must select ”OK” in order to confirm and proceed to the next part.

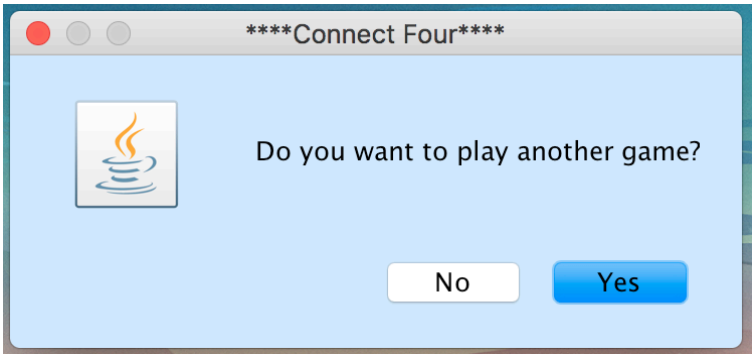
Then the board and with all the placed pieces appears once more issue that the users can see in more detail the game progress.



The users must once again select “OK” in order to proceed to the next screen.

One more Game

After the users select ok they receive the following message that asks them if they want to play one more game. If they select “YES” then the game starts again from the beginning. If they select “NO” the application is being terminated.



How the grid is being represented

The code structure consist of two basic components. The first component is called “Game_grid” that functions as the class that the object grid is created The second component is class “PlayC4game” where object “Game_grid” is being called and functions as the main class of the program.

Under the class “**Game_grid**”:

The required variables are being declared:

```
//declare variables
final static int ARRAY_HEIGHT = 6;
final static int ARRAY_WIDTH = 7;
public String Player = "";
int countMoves = 1;

                                        //rows   6      //columns 7
static String [] [] Grid = new String[ARRAY_HEIGHT] [ARRAY_WIDTH];
```

The variables ARRAY_HEIGHT and ARRAY_WIDTH represent respectively the hight and the width of the grid.

Finally the two demential array “Grid” is being declared:

static String [] [] Grid = new String[ARRAY_HEIGHT] [ARRAY_WIDTH];

Constructor

Under the constructor the Wellcome message to the users is being initiated through a JOptionPane window.

Also the Grid is been initialised with a nested for loop that sets all the grid positions to “ “ (empty spaces).

```
//constructor
public Game_grid() {

    //print welcome message
    JOptionPane.showMessageDialog(null,null, toString(),JOptionPane.PLAIN_MESSAGE);

      //initialise grid by giving value " " to all the positions
    for (int outerLoop = 0; outerLoop <ARRAY_HEIGHT; outerLoop++){
        for(int innerLoop = 0; innerLoop < ARRAY_WIDTH; innerLoop++) {
            this.Grid[outerLoop] [innerLoop] = "  ";

        }//end for
    }//end for
}//end constructor

//String to String - generates the output of the welcome message
public String toString() {

    String output = "Play Connect four !" + "\n";

    return output;

}//end toString
```

Displaying the Grid

The grid is being displayed in the console window and a JOptionPane window. Initially I used the console window in order to display my results and then I added the JOptionPane window.

The code for the console display of the Grid could have easily been removed but I included the code as it provides an easy way to check potential changes (without using the JOptionPane windows) and it was also the first step of my process to understand how the Grid could have been displayed.

```
//this method generates the required grid and displays it in a JOptioPane window
public static void printGameGrid() {

    String strBoard = " ";

    System.out.print("  1  •  2  •  3  •  4  •  5  •  6  •  7  " + "\n");

    strBoard = ("   1     2     3     4     5     6     7  " + "\n" + "———————————————————————————" + "
        for(int outerLoop  = 0; outerLoop < ARRAY_HEIGHT; outerLoop++) {
                for(int innerLoop = 0; innerLoop < ARRAY_WIDTH; innerLoop++) {
                    if(innerLoop  == 0 ) {

                        System.out.print("| ");
                        strBoard =  strBoard + "|  ";
                    }

                        System.out.print(Grid[outerLoop] [innerLoop] +" |  " );
                        strBoard = strBoard  + Grid[outerLoop] [innerLoop] +"  |  " ;

            }//end inner loop

                strBoard = strBoard +  "\n" +  "———————————————————————————" + "\n";
            System.out.println("");

        }//end outer for loop

        System.out.println("=========================================");

        JOptionPane.showMessageDialog(null,strBoard, "****Connect - 4****",JOptionPane.PLAIN_MESSAGE);

}//end printGameGrid
```

Updates Available

Initially the variable strBoard is being declared as String that in this case functions a the output for the JOptionPane windows.

Firstly I set the “strBoard” as “ (" 1 2 3 4 5 6 7 " + “\n" +"————————————————————————————" + “\n”);" so that this can be printed ton the top part of the Grid.

Then I initiates a nested for loop that loops through the columns and the row printing the lines we need to print the grid.

Inside the second fro loop there is an if statement the prints the far right character- line of the Grid (strBoard = strBoard + "| “;) only for the first column.

Then for each position in the grid the loop prints:

```
strBoard = strBoard  + Grid[outerLoop] [innerLoop] +"  |  " ;.
```

After the above has been printed for each time the inner loop runs we print the below (as part of the outer loop):

```
strBoard = strBoard +  "\n" +  "————————————————————————————" + “\n";
```

After the nested loops the output is being projected in a JOptionPane window.

JOptionPane.showMessageDialog(null, strBoard, "****Connect - 4****",JOptionPane.PLAIN_MESSAGE);

The Grid is being called in “PlayC4" Class and its colour is being changed.

Game_grid PLAY = new Game_grid();

```
mport javax.swing.JOptionPane;
mport javax.swing.UIManager;
mport javax.swing.plaf.ColorUIResource;

ublic class PlayC4game {

    public static void main(String[] args) {

        //change the background colour of the JoptionPane window
        UIManager uim =new UIManager();
        uim.put("OptionPane.background",new ColorUIResource(255, 255, 235));
        uim.put("Panel.background",new ColorUIResource(255, 255, 235));


        //declare variable Win
        boolean Win;
        int anotherGo;
        int counter = 1;


        //generate a do while loop that asks the user to enter the data as many times as he wishes
        do {

            Game_grid PLAY = new Game_grid();
```

Displaying the piecies in the grid.

Every time a player makes a valid move his piece is being displayed in the Grid in the required position.

All the conditions under witch the each player places a piece in the board are being structured under the method “playApiece()”.

```
//this method creates the conditions under witch the each player drops a piece in the grid
public void playApiece() {

    //Player chooses the column
            String chosseCol = JOptionPane.showInputDialog("Player " + "-[" + this.Player + "]-" + " your turn!" );

                    //convert from string to integer
                    int intChosseCol = Integer.parseInt(chosseCol);


            //creating an if statement that places the peace in the first available empty square (ARRAY_HEIGHT - 1 so that rows start from 1 and not 0)
             for (int row = ARRAY_HEIGHT -1; row >= 0; row--) {


                    //validate the player input to input pieces between the columns 1-7
                    while(intChosseCol > ARRAY_WIDTH || intChosseCol <= 0 || Grid[0][intChosseCol-1] != "  "){

                        //ask from the player to input a valid number
                        chosseCol = JOptionPane.showInputDialog("Player " + "-[" + this.Player + "]-" + "\n" + "Invalid move!" + "\n" + "Please choose another column.");

                        //convert from string to integer
                        intChosseCol = Integer.parseInt(chosseCol);

                    }//end while


                    //Using intChosseCol-1 in order to be easer for the user to Place the disc
                if (Grid[row][intChosseCol-1] == "  " ) {

                     Grid[row][intChosseCol-1]= this.Player;

                     return;

                }//end if

             }// end for


    }//end playApeace
```

In this method the user is being prompted to select the column that he wishes to place the piece in.

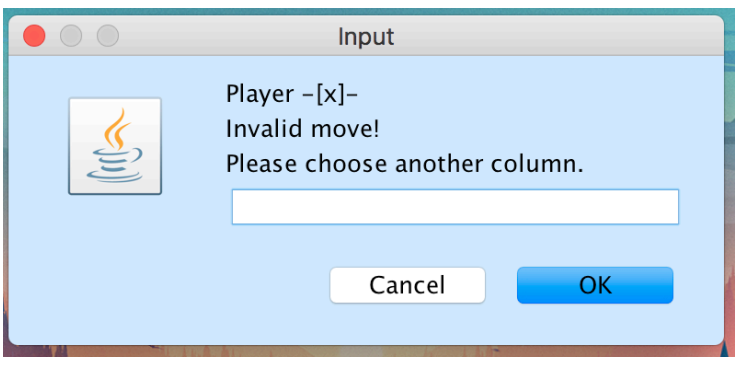
In the next step I created a for loop that is being repeated for each column and every time a piece has been successfully placed then it refuses the the column number by 1. This allows the pieces to seat on the top of each-other.

```
for (int row = ARRAY_HEIGHT -1; row >= 0; row--) {
```

If the player selects a column that is smaller than the first column of the board and larger that the number of the columns, the user receives a message informing the player that his move is not valid and that he must select another column. The message will be repeated until the user makes a valid move.

The same message will be displayed if the player places a piece in a full column.

```
“while(intChosseCol > ARRAY_WIDTH || intChosseCol <= 0 || Grid[0][intChosseCol-1] != "
“)”
```



If the movement is valid the the piece is being placed in the first available space in the board.

```
//Using intChosseCol-1 in order to be easer for the user to Place the piece
if (Grid[row][intChosseCol-1] == "  " ) {

                        Grid[row][intChosseCol-1]= this.Player;

                    return;
```

Winning conditions

In order to check for a winner we need to go through the whole board and check if any of he below conditions are met.

A player must place four consequent pieces:

- Vertically
- Horizontally
- Diagonally (left to right)
- Diagonally (right to left)

All the above conditions are being checked in the method " checkForWin()".
This method returns a Boolean variable, if the returned variable "Win" is "false" then that means than the game ended because there is a winner.

Checking horizontally.

In order to check for winning combinations horizontally I created two nested for loops that go through the board and check if there are four pieces of the same player in each row. If the condition has been met then the process stops and the winner is being declared as the

```
//check horizontally for PLAYER
for(int row = 0; row < ARRAY_HEIGHT; row++) {

    for(int column = 0; column < ARRAY_WIDTH - 3; column++) {

        if(this.Grid[row][column] != "  " &&
                this.Grid[row][column] == this.Player &&
                this.Grid[row][column] == this.Grid[row][column+1] &&
                this.Grid[row][column] == this.Grid[row][column+2] &&
                this.Grid[row][column] == this.Grid[row][column+3]) {


            JOptionPane.showMessageDialog(null,"PLAYER  " + "-[" + this.Player + "]-" , "WINNER!" ,JOptionPane.PLAIN_MESSAGE);
            Win = false;
            break;

        }//end if
    }//end for column
}//end for row
```

Because we check the the initial position for each sequence of pieces and then the three subsequent cells we do not need to include the last for columns in our loop as there cannot be any possible winning combinations starting form these cells.

Checking vertically.

The same logic with above allies to the vertical winning combinations only in this case we do not need to run our loop through the last 3 row again there is no possible whining combination starting from these.

```
//check vertically for PLAYER
for(int column = 0; column < ARRAY_WIDTH; column++) {

    for(int row = 0; row < ARRAY_HEIGHT-3;row++) {

        if(this.Grid[row][column] != "  " &&
                this.Grid[row][column] == this.Player &&
                this.Grid[row][column] == this.Grid[row+1][column] &&
                this.Grid[row][column] == this.Grid[row+2][column] &&
                this.Grid[row][column] == this.Grid[row+3][column]) {

            JOptionPane.showMessageDialog(null,"PLAYER  " + "-[" + this.Player + "]-" , "WINNER!" ,JOptionPane.PLAIN_MESSAGE);
            Win = false;
            break;

        }//end if
    }//end for column
}//end for row
```

Checkin diagonally (left to right).

In order to check for left to right diagonal winning combinations I created two nested for loops that check the board for possible winning combinations.

These for check for player pieces and then check if diagonally below that piece there is the same piece. This process checks this pattern in the next three subsequent positions.

```
if(this.Grid[row][column] != "  " &&
                                        this.Grid[row][column] == this.Player &&
                                        this.Grid[row][column] == this.Grid[row+1][column+1] &&
                                        this.Grid[row][column] == this.Grid[row+2][column+2] &&
                                        this.Grid[row][column] == this.Grid[row+3][column+3])
```

No winning combinations can start from the last three rows and the last three columns and this is the reason that these have been excluded from the loop.

```
//check left to right diagonally for PLAYER
for (int row = 0; row < ARRAY_HEIGHT - 3 ; row++ ) {

    for(int column = 0; column < ARRAY_WIDTH- 3 ; column++ ) {

        if(this.Grid[row][column] != "  " &&
                this.Grid[row][column] == this.Player &&
                this.Grid[row][column] == this.Grid[row+1][column+1] &&
                this.Grid[row][column] == this.Grid[row+2][column+2] &&
                this.Grid[row][column] == this.Grid[row+3][column+3]) {

            JOptionPane.showMessageDialog(null,"PLAYER  " + "-[" + this.Player + "]-" , "WINNER!" ,JOptionPane.PLAIN_MESSAGE);
            Win = false;
            break;

        }//end if
    }//end for column
}//end for row
```

Checkin diagonally (right to left)

The same logic as above is being repeated in order to check for right to left winning combinations. The main difference in this case is that the loops check if the same piece can be found diagonally above a piece. This again is repeated for next three subsequent positions.

```
if(this.Grid[row][column] != "  " &&
                                                this.Grid[row][column] == this.Player &&
                                                this.Grid[row][column] == this.Grid[row+1][column-1] &&
                                                this.Grid[row][column] == this.Grid[row+2][column-2] &&
                                                this.Grid[row][column] == this.Grid[row+3][column-3]
```

In this case no stating combinations can start from the first three columns and the last three rows.

```
//check right to left diagonally for PLAYER1
for (int row = 0; row < ARRAY_HEIGHT - 3 ; row++ ) {

    for(int column = 3; column < ARRAY_WIDTH  ; column++ ) {

        if(this.Grid[row][column] != "  " &&
                this.Grid[row][column] == this.Player &&
                this.Grid[row][column] == this.Grid[row+1][column-1] &&
                this.Grid[row][column] == this.Grid[row+2][column-2] &&
                this.Grid[row][column] == this.Grid[row+3][column-3]) {

            JOptionPane.showMessageDialog(null,"PLAYER  " + "-[" + this.Player + "]-" , "WINNER!" ,JOptionPane.PLAIN_MESSAGE);
            Win = false;
            break;

        }//end if
    }//end for column
}//end for row
```

No winner

If there is no winner and all the grid positions are full then the game stops and a “Game Over” message appears.

I implemented the above through method “countMoves()”. This method sets a counter that is being incremented by one every time both players make move.

```
// this method is being used to ensure that ehen the board is full the game will end
public boolean countMoves(int counter) {

    boolean Win = true;

    if(counter*2 > 42) {

     JOptionPane.showMessageDialog(null,"There are no more available moves", "GAME OVER!" ,JOptionPane.PLAIN_MESSAGE);
        Win = false;

     }// end if
    return Win;

}//end countMoves
```

If the players make more that 42 moves (maximum amount of available moves) then the game is terminated with no winner.

Class diagram