Programming II

Assignment 1

Tic-Tac-Toe

OOP Implementation

Prepared by:

Name	ID	Group
Youssef Samuel Nachaat Labib	6978	4
Youssef Amr Ismail Othman	6913	4

Classes Description

Our tic-tac-toe game is implemented by the use of OOP using 3 classes: **Board** , **Player** and **Game**. We will discuss the state and the behaviour for each class individually and how they interact with each other.

1) Board Class

The **Board** class is used to encapsulate all the attributes and methods that would be used by the board. It uses 4 constants that will specify the size and properties of the board.

The constants used are:

- *ROWS* which is set to 6 stating the row size.
- *COLUMNS* which is set to 7 stating the column size.
- *ARRAY_EXTENSION* that is set to 4 (Its use will be fully elaborated in the isWinner method).
- *NUM_BLOCKS* that states the total number of cells in the board. It is the result of the multiplication of *ROWS* and *COLUMNS*.

A character array "board" will be used as the grid. It is declared and constructed in the attributes section of the class. fullBlocks variable is used as a counter that is incremented after each successful insertion in the cell. It will state the total number of occupied cells at any time.

The class constructer is used to initialize all elements of the array board to '-' .
As the fullBlocks variable implements the encapsulation method, a setter and a getter were used to access and modify it. Since it could be set by only the previous value + 1, the setter does not have any parameters. Another method of that class is printBoard, it will print the board after each player makes his move according to the format set that will be seen in the sample runs.

insert method is used to assign the player input value to the cell specified. It has 3 parameter, one stating the row, another stating the column and the third stating the **Player** object. It uses the method getInputValue of the **Player** class to get the symbol for that **Player** object.

The methods used for checking for a winner are 2. isDraw method would check whether the inserted cells is equal to the total number of cells, if found equal and nobody won then it will announce a draw.

isWinner method checks after each cell is inserted whether this cell may result in a win or not, it has the parameter of the **Player** object that placed that cell. The getters for the row and column would be called so the checking can happen upon that cell entered.

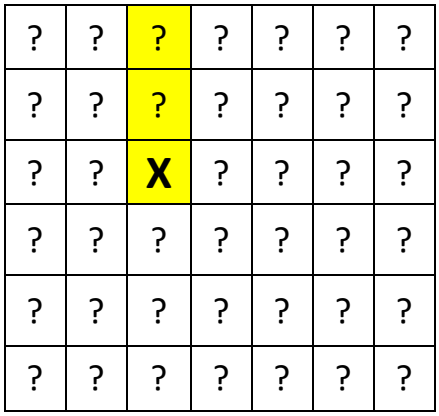
Algorithm for checking:

Upon constructing and initialising the array, we increased its size by 2 rows from above, 2 rows from below, 2 columns from the left and 2 columns from the right (Shown in red). Resulting in an increase by 4 for the wanted rows and columns dimensions "*ARRAY_EXTENSION".* The white cells are the cells that would be played in and the ones that will be displayed for the players. This step was made to facilitate checking in an easier way.

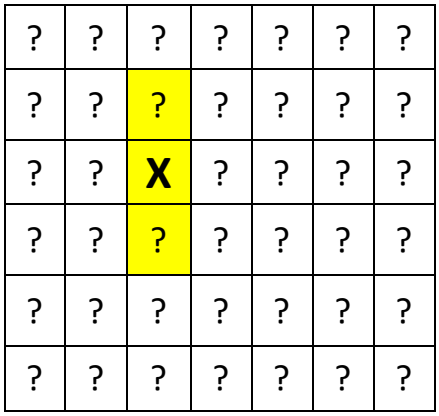
	0	1	2	3	4	5	6	7	8	9	10
0											
1			1	2	3	4	5	6	7		
2		1	?	?	?	?	?	?	?		
3		2	?	?	?	?	?	?	?		
4		3	?	?	**X**	?	?	?	?		
5		4	?	?	?	?	?	?	?		
6		5	?	?	?	?	?	?	?		
7		6	?	?	?	?	?	?	?		
8											
9											

There are 12 possibilties that may result in a win if the player placed in [3] [3] (index [4][4]) for instance. The check is emplemented in the function using that particular numbered order found in the following table, the function will be terminated upon reaching the first match .

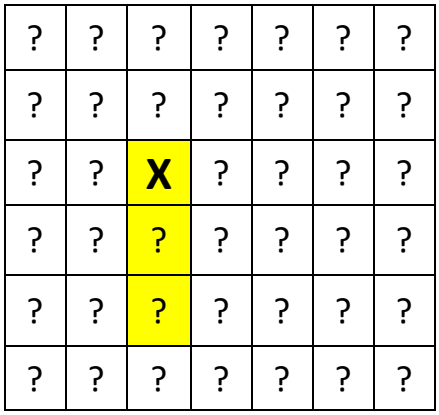
1.



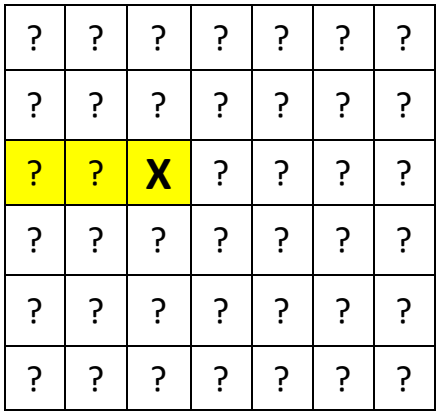
2.



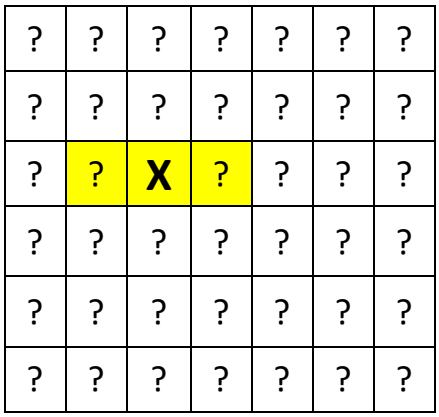
3.



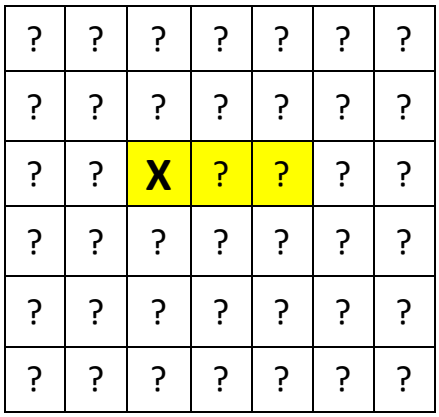
4.



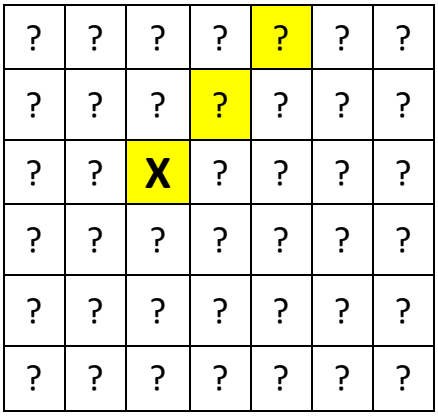
5.



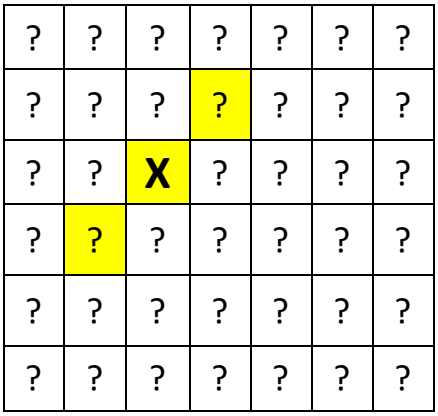
6.



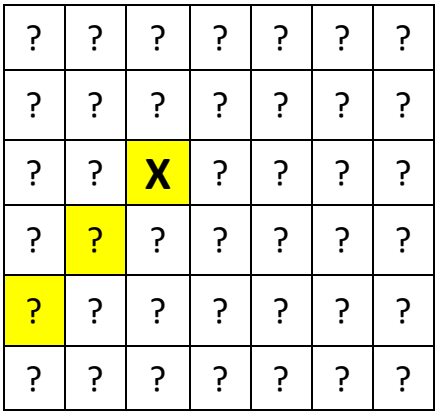
7.



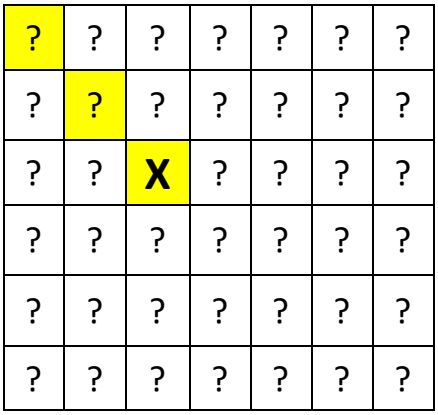
8.



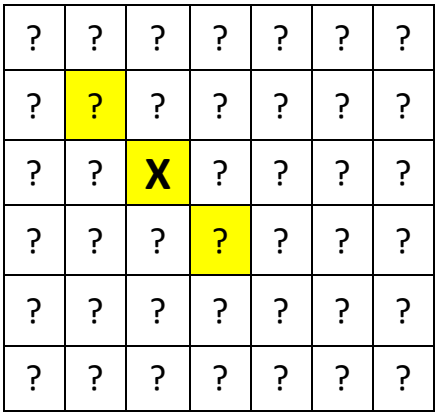
9.



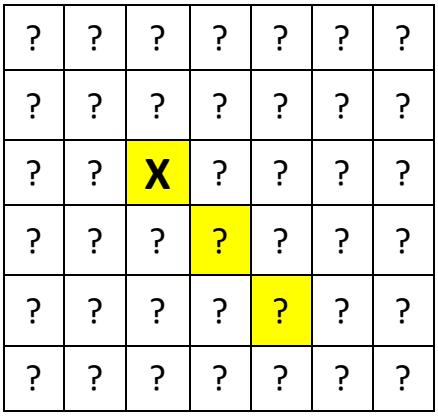
10.



11.



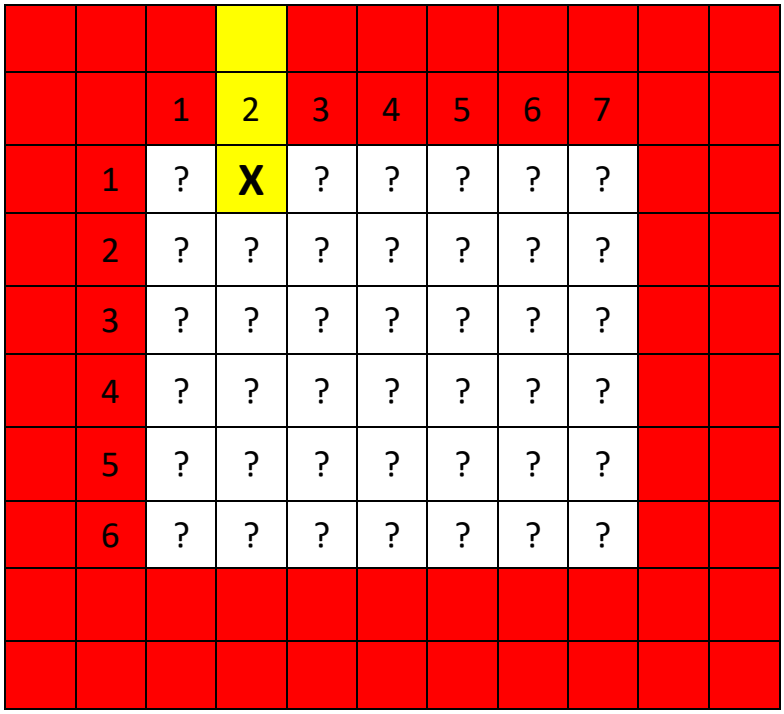
12.



Increasing the array size by 4 rows and 4 columns is done solely for the sake of the check function. By that increase, if we check for an element on the borders, no compilation errors resulting from accessing invalid index number would occur “ArrayIndexOutOfBoundsException”. It will not affect the logic for the check algorithm as well, meaning that the check would not be altered according to the place of the cell to be checked.

Consider checking an element placed in [1][2] (index [2][3]).

The first check is highlighted in yellow. Although the check is made in cells that will not be displayed, but it will not result in an error as those cells are already reserved for the array from the beginning. And the checks would continue …….



2) Player Class

The **Player** class clearly shows how the encapsulation principle was used in our code. All the attributes were set private for which the accessors and mutators methods will be used to modify or access them.

The attributes used are:

- playerNumber that stores the number for the player. It will carry the value of 1 or 2.
- playerName that will store the name for the **Player** object.
- rowInput that will store the row number for the last cell added by the **Player** object.
- columnInput that will store the column number for the last cell added by the **Player** object.
- inputValue that stores the characters 'X' or 'O' according to whether which **Player** object is used.

The constructor is used to set the inputValue of the object to 'X' for player 1 and '0' for player 2. In addition to all the setters and getters methods used, the class has play(Board board) method that prompts the user for his row and column input. The method checks whether the input values are out of range or not. If found out of range, a loop will be executed until a successful input is reached. The method would then check for the wanted cell if its vacant or preoccupied by a previous move. If found preoccupied, it would call the play method recursively, else it would use the setters of the row and column and would call the insert method in **Board** class using the getters of the row and column and this to send the current **Player** object. The fullBlocks of **Board** class would then be incremented by 1 to state that a new cell has been occupied. It's used as a counter that can have a maximum value of 42.

3) Game Class

The **Game** class is the class where the programme starts. It has the main method that creates an object of the class **Board** to be named board and 2 objects for the 2 players from the class **Player** to be named player1 and player2. The main would ask the user to enter player 1 name and player 2 name respectively using GUI (JOptionPane). A welcome message would then be displayed to the user and the board would be printed on the console. Player 1 would then be prompted to enter the row and column choice using the play method of the **Player** class. After a successful insertion, the isWinner method would be invoked to check whether player 1 won the game using that move or not, if he won, he will be congratulated using a GUI message, if not, it will check for a draw using the isDraw method. If player 1 did not win neither his move made a draw, the board will be displayed and the same sequence would be made with player 2. The function arguments would only differ according to the **Player** object used. All these statements are implemented in a while loop that will break if a win or a draw is reached.

The final board would then be displayed and a thanking message would appear for the user stating the game termination.