Project 1

CPSC 2150

Part 1 Due: Monday, September 6th at 11:59 pm

Part 2 Due: Wednesday, September 15th at 11:59 pm

Full Assignment Due: Wednesday, September 22nd at 11:59 pm

100 pts

Introduction:

In this assignment, we will be creating an extended version of Tic-Tac-Toe. In Tic-Tac-Toe, players take turns placing X's and O's on a 3 x 3 grid, trying to get three in a row either vertically, horizontally, or diagonally. If they do, then they win. We will be extending this game to a 5 x 8 board and requiring 5 in a row either horizontally, vertically, or diagonally to win.

For the first stage of this project, we will just be designing our game. You will be creating UML diagrams for classes and methods and writing contracts for each method and class. Designing your system first will allow you to work through the complicated decisions about how your classes and methods interact without implementing them yet. This will make it easier to implement individual methods later.

To do this, you will need several classes and functions. Please note that it is essential to strictly follow these directions to design the classes and methods as described, use the same class and method names, and have identical function signatures (return type and parameter list). This will be important for future assignments as well. Projects that achieve the overall goal but do not follow these instructions will lose substantial points.

Requirements Analysis:

Projects always start with requirements, and this one is no different. Based on what you know about playing games and the information above, write the user stories that capture the game's requirements. In this project, you only have one user, "player," which makes the process easier. Put all the functional requirements in the Agile story format:

As a <user role> I <what/need/can> <goal> so that <reason>.

Make sure you include the non-functional requirements as well, but these don't need to be expressed in user story format. These requirements will drive what you implement in the classes described below, so make sure you take the time to think about all the things that a player wants to do in a game of Tic-Tac-Toe.

`GameScreen.java`

This will be the class that contains the `main()` method and controls the game's flow. It will alternate the game between players, say whose turn it is, get the location they would like, and place their marker. Suppose the player chooses a location that is not available (either because it has been claimed or because it is outside the 5 x 8 board). In that case, the program will print a message saying the space is unavailable and ask them to enter a new location. Once a placement has been made, it will check to see if either player has won. If so, it will print off a congratulatory message, and ask if they would like to play again and prompt them for a response of "Y" or "N". If they choose to play again, it

should present them with a blank board and start the game over. If a player has not won, it will check if the game has ended in a draw (no spaces left on the board). If the game has ended in a draw, it will print a message saying so and ask if the player would like to play again. If the game has not ended in a win or a draw, print the current board to the screen and prompt the next player for their next move.

When asking a player for their move, it will ask first for the `Row` coordinate (where 0,0 is the top left corner) and then ask for the `Column` coordinate. It is essential to keep this order to allow the graders to write scripts to test your program instead of running manually. You can assume the user will enter an integer value for `Row` and `Column`. You cannot assume they will enter a value within the range (they could enter 15, 21). Player X should go first in every game.

This class has the least amount of specific requirements. You may design your class and add as many private helper methods that you believe may help you. All input and output to the screen must happen in only this class.

For this stage of the assignment, you will need to include in your project report:

- A UML class diagram for `GameScreen`
- A UML activity diagram for each method in `GameScreen`

Additionally, you will submit a code file for `GameScreen` called `GameScreen.java`. This code file will just be an outline for the eventual code file. It should include the signature, Javadoc comments, and contracts for all methods, but you do not need to write the code for each method yet. Because of this, your code will not compile yet.

`BoardPosition.java`

This class will be used to keep track of an individual cell for a board. `BoardPosition` will have variables to represent the `Row` position and the `Column` position.

There will only be one constructor for the class, which will take in an `int` for the `Row` position and an `int` for the `Column` position. After the constructor has been called, there will be no other way to change any fields through any setter methods.

Other methods that will be necessary:

```
public int getRow(){
	//returns the row
}

public int getColumn(){
	//returns the column
}
```

You must also override the `equals()` method inherited from the `Object` class. Two `BoardPosition`s are equal if they have the same row and column. You should override the `toString()` method to create a string in the following format "<row>,<column>." Example "3,5". You won't call these in your code, but it is a best practice to override them and provide a meaningful implementation.

No other methods are necessary, and no other methods should be provided.

For this stage of the assignment, you will need to include in your project report:

- A UML class diagram for `BoardPosition`

Additionally, you will submit a code file for `BoardPosition` called `BoardPosition.java`. This code file will just be an outline for the eventual code file. It should include the signature, Javadoc comments, and contracts for all methods, but you do not need to write the code for each method yet. Because of this, your code will not compile yet. You do not need to create activity diagrams for the methods in `BoardPosition`.

`GameBoard.java`

This class will contain our actual game board and will require the most work. Your game board should be a 2-dimensional 5 x 8 array of characters. Each position will have a single blank space character (') until a player plays on that position. At that point, the character will change to either X or O to represent that player. A constructor with no parameters should be included.

The following functions must be available

```
public boolean checkSpace(BoardPosition pos)
{
	//returns true if the position specified in pos is available;
false otherwise. If a space is not in bounds, then it is not available
}

public void placeMarker(BoardPosition marker, char player)
{
	//places the character in marker on the position specified by
marker and should not be called if the space is unavailable.
}

public boolean checkForWinner(BoardPosition lastPos)
{
	//this function will check to see if the lastPos placed resulted
in a winner. It so it will return true, otherwise false.
	//Passing in the last position will help limit the possible
places to check for a win condition since you can assume that any win
condition that did not include the most recent play made would have
been caught earlier.
	//You may call other methods to complete this method
}

public boolean checkForDraw()
{
	//this function will check to see if the game has resulted in a
tie. A game is tied if there are no free board positions remaining.
You do not need to check for any potential wins because we can assume
that the players were checking for win conditions as they played the
game. It will return true if the game is tied and false otherwise.
}
```

```
public boolean checkHorizontalWin(BoardPosition lastPos, char player)
{
     //checks to see if the last marker placed resulted in 5 in a row
horizontally. Returns true if it does, otherwise false
}

public boolean checkVerticalWin(BoardPosition lastPos, char player)
{
     //checks to see if the last marker placed resulted in 5 in a row
vertically. Returns true if it does, otherwise false
}

public boolean checkDiagonalWin(BoardPosition lastPos, char player)
{
     //checks to see if the last marker placed resulted in 5 in a row
diagonally. Returns true if it does, otherwise false
     //Note: there are two diagonals to check
}

public char whatsAtPos(BoardPosition pos)
{
     //returns what is in the GameBoard at position pos
     //If no marker is there, it returns a blank space char.
}

public boolean isPlayerAtPos(BoardPosition pos, char player)
{
     //returns true if the player is at pos; otherwise, it returns
false
     //Note: this method will be implemented very similarly to
whatsAtPos, but it's asking a different question. We only know they
will be similar because we know GameBoard will contain a 2D array. If
the data structure were to change in the future, these two methods
could be radically different.
}
```

Additionally, you must override the `toString()` method inherited from the `Object` class. This will return one string that shows the entire game board. You will then be able to call it in your `GameScreen.java` file to print it to the screen. It should be formatted as follows:

```
  0 1 2 3 4 5 6 7
0|  |  |  |  |X|  |  |
1|  |  |O|  |  |X|  |  |
2|  |  |  |O|  |  |  |  |
3|  |  |  |  |  |  |  |  |
4|  |  |  |  |  |  |  |  |
```

This way, it will be easy to see the layout of the board. Again, this class does not print to the screen; it just produces one string that contains the entire board that will be easy to print.

You should not add any other methods to this class.

For this stage of the assignment, you will need to include in your project report:

- A UML class diagram for `GameBoard`
- A UML activity diagram for each method in `GameBoard`

Additionally, you will submit a code file for `GameBoard` called `GameBoard.java`. This code file will just be an outline for the eventual code file. It should include the signature, Javadoc comments, and contracts for all methods and invariants for the class, but you do not need to write the code for each method yet. Because of this, your code will not compile yet.

Contracts, Comments, and formatting

For every method, you will need to create Javadoc comments and create contracts for each method. Contracts should be formally stated when possible. We have not covered much formal notation, but if a parameter named `row` needs to be greater than 0, then the precondition should say "row > 0" not "[row must be greater than 0]." The invariant for the classes should also be specified in Javadoc comments.

Everything that has been mentioned as a "best practice" in our lectures or a video should be followed, or you risk losing points on your assignment.

Project Report.

Along with your "code," you must submit a well-formatted report with the following parts:

Requirements Analysis

Include all your functional and non-functional requirements in the Project Report.

Design

Create a UML class diagram for each class in the program. Also, create UML Activity diagrams as specified in the above directions. All diagrams should be well formatted and easy to read. I recommend Diagrams.net (formally Draw.io) as a program to create the diagrams. You may not submit hand-drawn diagrams, even if they are scanned in or drawn on a table.

Your project report should be one well-formatted PDF document, not a folder of individual image files. The diagrams should be sized and arranged to be easy to read.

Questions to consider while designing your program:

- Remember to consider possible changes that could be made to this program. We will be building off of this program all semester, so things will change
- Remember to consider what information is publicly available and what is hidden. All data fields should be private, and you should not add any other public methods other than the ones specified.
- Remember to consider what each class knows and what it cares about. Follow separation of concerns, and encapsulate properly.

- Our contracts make our code easier to write by defining what situations should not even be considered. Write those first before making activity diagrams.

General Notes:

- Name your package `cpsc2150.extendedTicTacToe`
- Remember our "best practices" that we've discussed in class. Use good variables, avoid magic numbers, etc.
- Start early. You have time to work on this assignment, but you will need time to work on it.
- Starting early means more opportunities to go to TA or instructor office hours for help
- The instructor and TAs have lives as well, so don't expect email responses late at night. If you need help, you will need to start early enough to ask for help.
- Start by analyzing the requirements of this program to ensure you understand what you are designing.
- Remember, this class is about more than just functioning code. Your design and quality of code are essential. Remember the best practices we are discussing in class. Your code should be easy to change and maintain. You should not be using magic numbers. You should be considering Separation of Concerns, Information Hiding, and Encapsulation when designing your code. You should also be following the idea and rules of design by contract.
- Your UML Diagrams must be made electronically. I recommend the freely available program Diagrams.net (formally Draw.io), but if you have another diagramming tool you prefer to feel free to use that. It may be faster to draw rough drafts by hand, and then when you have the logic worked out, create the final version electronically.
- While you need to validate all input from the user, you can assume that they are entering numbers or characters when appropriate. You do not need to check to see that they entered 6 instead of "six."
- You may want to call one `public` method from another `public` method. We can do this, but we have to be very careful about doing this. Remember that during the process of a `public` method, we cannot assume that the invariant is true, but at the beginning of one, we assume that it is true. So if we want to call one `public` method from inside another, we need to be sure that the invariants are true and the preconditions of the method we are calling.
- Activity Diagrams should be detailed. Every decision that is made should be shown, not just described in one step. In other words, it is not sufficient to have one step/activity in the diagram say, "Check if the input is valid and prompt again if it is not." You need to show the logic for that check. Calling another method can be a single step in the activity diagram since it will have its own diagram. The activities in the diagrams do not need to be written as code. It is fine to write “Check if X is in position i, j” instead of “if (‘X’ == board[i][j])”
- You do not need to write the actual code yet. Do not try to solve this by writing the code, then drawing diagrams to match your code. It saves time to use the activity diagrams to work through the complicated logic in the long run.
- `GameScreen` is the only class that should get input from the user or print to the screen.
- 0,0 should be the top left of the board. If you do not make 0,0 the top left of the board, it could cause issues with the TA's scripts to grade the assignment.

- Future assignments will build off of this assignment. If you do not put your best effort into this assignment, future projects will be more difficult.

Submission:

To ensure that you do not wait until the last minute to start the assignment, we are requesting three submissions for this assignment. Part 1 will be due Monday, September 6th at 11:59 pm, Part 2 will be due Wednesday, September 15th at 11:59pm, and the complete project is due Wednesday, September 22nd at 11:59 pm. There will be separate links for you to submit your files on Gradescope by the specified due date.

For part 1, you will be submitting completed class diagrams for `GameBoard`, `GameScreen` and `BoardPosition` on Gradescope. You should upload them as an image file (jpg or png) and not Diagrams.net files.

For part 2, you will be submitting complete JavaDoc + contracts for `GameBoard`, `GameScreen` and `BoardPosition` and the requirements analysis on Gradescope. Upload each Java file and fill in the requirements in their respective questions.

For the complete assignment, put your class diagrams from part 1, the requirements analysis from part 2 and the activity diagrams into one single document. Your program report should be a PDF. There is no need for you to submit your Java files again. **Late submissions will not be accepted.**

NOTE: Always check your submissions on Gradescope to ensure you uploaded the correct files and it is in the proper format that it expects. **I wouldn't wait until the last minute to submit as Gradescope could take a few minutes to display the results.** Notify your course instructor immediately of any Gradescope issues.

Disclaimer:

It is possible that these instructions may need to be updated. As students ask questions, I may realize that something is not as straightforward as I thought, and I may add detail to clarify the issue. Changes to the instructions will not change the assignment drastically and will not require reworking code. They would just be made to clarify the expectations and requirements. If any changes are made, they will be announced on Canvas.

Sample input and output:

```
  0|1|2|3|4|5|6|7|
0| | | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | | | | | | | |

Player X Please enter your ROW
12
Player X Please enter your COLUMN
5
  0|1|2|3|4|5|6|7|
0| | | | | | | | |
```

```
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | | | | | | | |

That space is unavailable, please pick again
Player X Please enter your ROW
5
Player X Please enter your COLUMN
12
  0|1|2|3|4|5|6|7|
0| | | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | | | | | | | |

That space is unavailable, please pick again
Player X Please enter your ROW
4
Player X Please enter your COLUMN
2
  0|1|2|3|4|5|6|7|
0| | | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | |X| | | | | |

Player O Please enter your ROW
0
Player O Please enter your COLUMN
0
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | |X| | | | | |

Player X Please enter your ROW
3
Player X Please enter your COLUMN
2
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | |X| | | | | |
4| | |X| | | | | |
```

```
Player O Please enter your ROW
2
Player O Please enter your COLUMN
2
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| | | | | | | | |
2| | |O| | | | | |
3| | |X| | | | | |
4| | |X| | | | | |

Player X Please enter your ROW
4
Player X Please enter your COLUMN
1
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| | | | | | | | |
2| | |O| | | | | |
3| | |X| | | | | |
4| |X|X| | | | | |

Player O Please enter your ROW
1
Player O Please enter your COLUMN
1
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | | | | | |
2| | |O| | | | | |
3| | |X| | | | | |
4| |X|X| | | | | |

Player X Please enter your ROW
2
Player X Please enter your COLUMN
2
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | | | | | |
2| | |O| | | | | |
3| | |X| | | | | |
4| |X|X| | | | | |

That space is unavailable, please pick again
Player X Please enter your ROW
2
Player X Please enter your COLUMN
3
```

```
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | | | | | |
2| | |O|X| | | | |
3| | |X| | | | | |
4| |X|X| | | | | |

Player O Please enter your ROW
3
Player O Please enter your COLUMN
3
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | | | | | |
2| | |O|X| | | | |
3| | |X|O| | | | |
4| |X|X| | | | | |

Player X Please enter your ROW
1
Player X Please enter your COLUMN
4
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | |X| | | |
2| | |O|X| | | | |
3| | |X|O| | | | |
4| |X|X| | | | | |

Player O Please enter your ROW
4
Player O Please enter your COLUMN
4
Player O wins!
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | |X| | | |
2| | |O|X| | | | |
3| | |X|O| | | | |
4| |X|X| |O| | | |

Would you like to play again? Y/N
n
```