CPSC 2150

Project 5

Project Report

Charlie Hartsell

Part 1, Requirements:

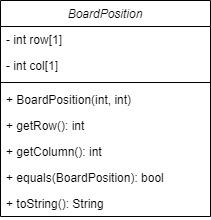
**User Stories**

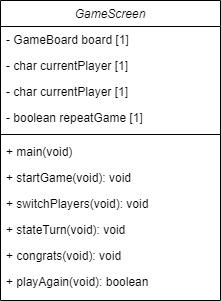
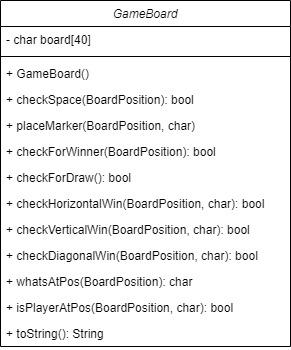
* As a player I need to choose a space so that I can claim it with my marker.
* As a player I need to know what board spaces are available to be claimed.
* As a player I need to know which spaces I have claimed.
* As a player I need to know which spaces my opponent has claimed.
* As a player I need to be informed when I have won or lost.
* As a player I need to be informed when my opponent has won or lost.
* As a player I need to be informed when a tie is reached.
* As a player I need to have the option to restart the game once it has ended.

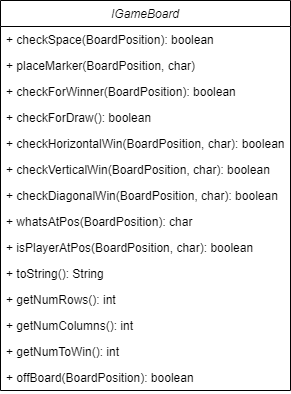
**Non-Functional Requirements**

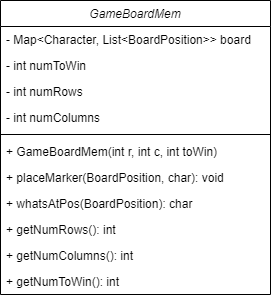
* The program should be made in Java.
* The program should not use magic numbers.
* The program should have UML diagrams for each pre-specified class.
* All user input must be validated before being processed.
* The program should present information to the player in a way that is easy to understand.

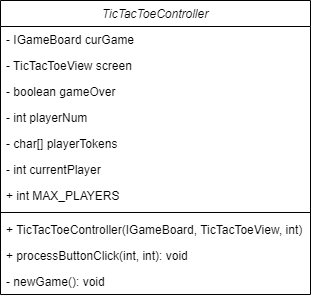
Part 2, Design:

**UML Diagrams**



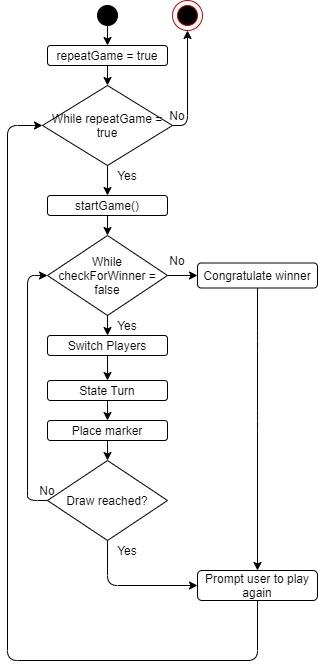




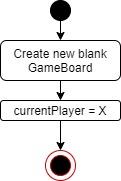


**Activity Diagrams**

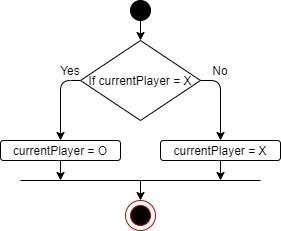
GameScreen: main



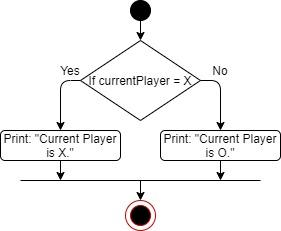
GameScreen: startGame

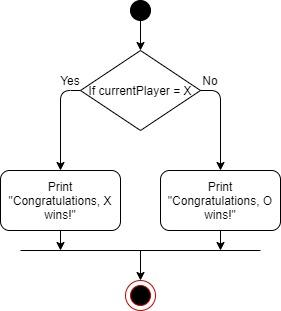


GameScreen: switchPlayers

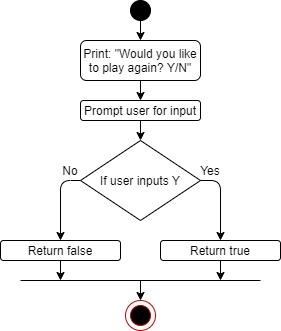


GameScreen: stateTurn

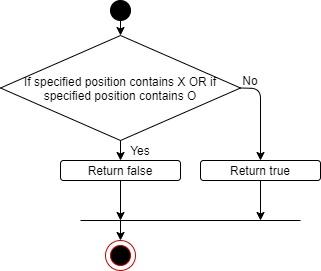


GameScreen: congrats  


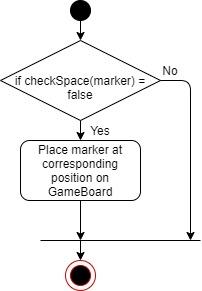
GameScreen: playAgain



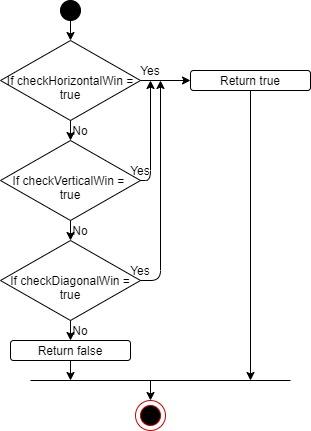
GameBoard: checkSpace



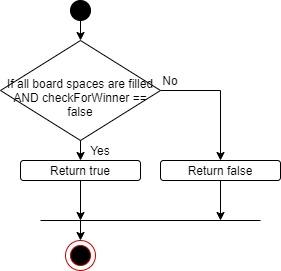
GameBoard: placeMarker



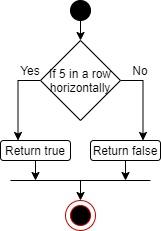
GameBoard: checkForWinner



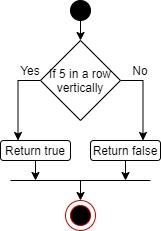
GameBoard: checkForDraw



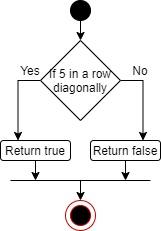
GameBoard: checkHorizontalWin



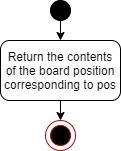
GameBoard: checkVerticalWin



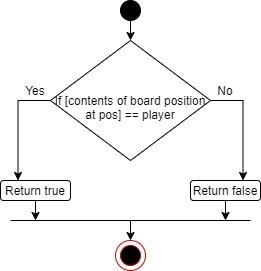
GameBoard: checkDiagonalWin



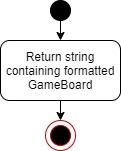
GameBoard: whatsAtPos



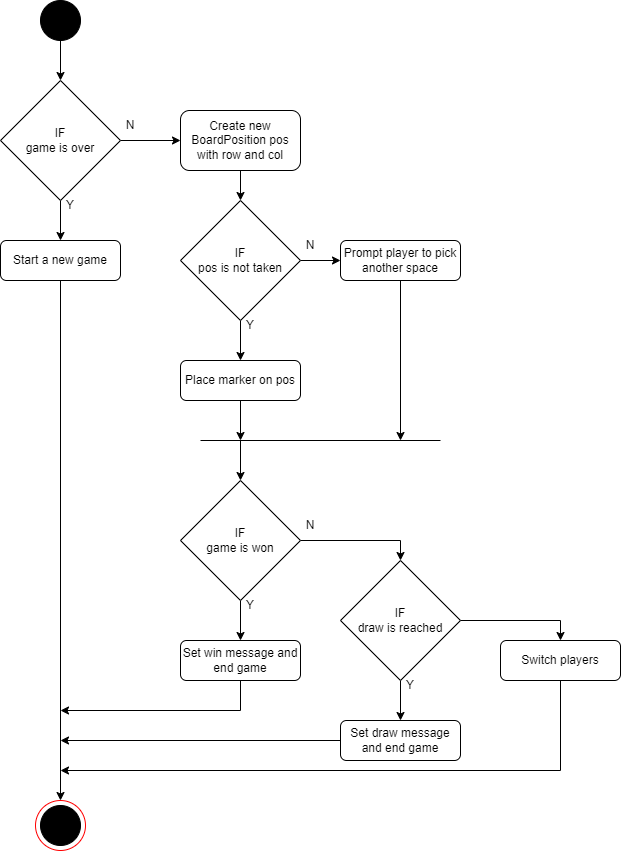
GameBoard: isPlayerAtPos



GameBoard: toString



TicTacToeController: processButtonClick



Part 3, Testing:

GameBoard(int r, int c, int toWin)

| **Input**:  r: 3  c: 3  toWin: 3 | **Output**:  State: 3x3 board with 3 in a row needed to win. | **Reason**:  Standard constructor call.  **Function name**:  testConstructor\_3x3 |
| --- | --- | --- |

| **Input**:  r: 100  c: 100  toWin: 3 | **Output**:  State: 100x100 board with 3 in a row needed to win. | **Reason**:  Large number used for rows and columns.  **Function name**:  testConstructor\_100x100 |
| --- | --- | --- |

| **Input**:  r: 3  c: 3  toWin: 100 | **Output**:  State: 3x3 board with 100 in a row needed to win. | **Reason**:  Large number used for toWin.  **Function name**:  testConstructor\_largeWinNum |
| --- | --- | --- |

boolean checkSpace(BoardPosition pos)

| **Input**:  State: 3x3 board is empty.  Position (0,0). | **Output**:  True. | **Reason**:  The space in question is empty.  **Function name**:  testCheckSpace\_empty |
| --- | --- | --- |

| **Input**:  State: 3x3 board filled with ‘X’.  Position (0,0). | **Output**:  False. | **Reason**:  The space checked contains an X.  **Function name**:  testCheckSpace\_full |
| --- | --- | --- |

| **Input**:  State: 3x3 board filled with ‘X’.  Position: (4,4). | **Output**:  False. | **Reason**:  The space checked is out of bounds.  **Function name**:  testCheckSpace\_outOfBounds() |
| --- | --- | --- |

boolean checkHorizontalWin(BoardPosition lastPos, char player)

| **Input**:  State: 3x3 board with top row filled with ‘X’.  Position: (0,0) | **Output**:  True. | **Reason**:  The position is on the far left of the winning row.  **Function name**:  testCheckHorizontalWin\_right\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 board with the top row filled with ‘X’.  Position: (0,2) | **Output**:  True. | **Reason**:  The position is on the far right of the winning row.  **Function name**:  testCheckHorizontalWin\_left\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 board with top row filled with ‘X’.  Position: (0,1) | **Output**:  True. | **Reason**:  The position is in the middle of the winning row.  **Function name**:  testCheckHorizontalWin\_middle\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 empty board.  Position: (0,0) | **Output**:  False. | **Reason**:  A win has not been reached.  **Function name**:  testCheckHorizontalWin\_noWin |
| --- | --- | --- |

boolean checkVerticalWin(BoardPosition lastPos, char player)

| **Input**:  State: 3x3 board with left column filled with ‘X’.  Position: (2,0) | **Output**:  True. | **Reason**:  The position is at the bottom of the winning column.  **Function name**:  testCheckVerticalWin\_up\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 board with left column filled with ‘X’.  Position: (0,0) | **Output**:  True. | **Reason**:  The position is at the top of the winning column.  **Function name**:  testCheckVerticalWin\_down\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 board with left column filled with ‘X’.  Position: (1,0) | **Output**:  True. | **Reason**:  The position is in the middle of the winning column.  **Function name**:  testCheckVerticalWin\_middle\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 empty board.  Position: (0,0) | **Output**:  False. | **Reason**:  A win has not been reached.  **Function name**:  testCheckVerticalWin\_noWin |
| --- | --- | --- |

boolean checkDiagonalWin(BoardPosition lastPos, char player)

| **Input**:  State: 3x3 board filled with ‘X’.  Position: (2,0) | **Output**:  True. | **Reason**:  The position is at the bottom left of the winning diagonal.  **Function name**:  testCheckDiagonalWin\_upRight\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 board filled with ‘X’.  Position: (0,2) | **Output**:  True. | **Reason**:  The position is at the top right of the winning diagonal.  **Function name**:  testCheckDiagonalWin\_downLeft\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 board filled with ‘X’.  Position: (2,2) | **Output**:  True. | **Reason**:  The position is at the bottom right of the winning diagonal.  **Function name**:  testCheckDiagonalWin\_upLeft\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 board filled with ‘X’.  Position: (0,0) | **Output**:  True. | **Reason**:  The position is at the top left of the winning diagonal.  **Function name**:  testCheckDiagonalWin\_downRight\_win |
| --- | --- | --- |

| **Input**:  State: 3x3 board filled with ‘X’.  Position: (1,1) | **Output**:  True. | **Reason**: The position is in the middle of the winning diagonal, which has a positive slope.  **Function name**:  testCheckDiagonalWin\_posSlope\_middleWin |
| --- | --- | --- |

| **Input**:  State: 3x3 board filled with ‘X’.  Position: (1,1) | **Output**:  True. | **Reason**:  The position is in the middle of the winning diagonal, which has a negative slope.  **Function name**:  testCheckDiagonalWin\_negSlope\_middleWin |
| --- | --- | --- |

| **Input**:  State: 3x3 empty board.  Position: (1,1) | **Output**:  False. | **Reason**:  A win has not been reached.  **Function name**:  testCheckDiagonalWin\_noWin |
| --- | --- | --- |

boolean checkForDraw()

| **Input**:  State: 3x3 board filled with ‘X’. | **Output**:  True. | **Reason**:  The board is entirely full of a single letter.  **Function name**:  testCheckForDraw\_drawReached\_singleLetter |
| --- | --- | --- |

| **Input**:  State: 3x3 board filled with alternating ‘X’ and ‘O’. | **Output**:  True. | **Reason**:  The board is full of mixed letters.  **Function name**:  testCheckForDraw\_drawReached\_mixedLetters |
| --- | --- | --- |

| **Input**:  State: 3x3 board with the top row filled with ‘X’. | **Output**:  False. | **Reason**:  A draw has not been reached but a win has.  **Function name**:  testCheckForDraw\_noDraw\_winReached |
| --- | --- | --- |

| **Input**:  State: 3x3 empty board. | **Output**:  False. | **Reason**:  The board is empty, so both a draw and a win have not been reached.  **Function name**:  testCheckForDraw\_noDraw\_emptyBoard |
| --- | --- | --- |

char whatsAtPos(BoardPosition pos)

| **Input**:  State: 3x3 board with ‘X’ at (0,0)  Position: (0,0) | **Output**:  X | **Reason**:  The given position contains ‘X’.  **Function name**:  testWhatsAtPos\_X |
| --- | --- | --- |

| **Input**:  State: 3x3 board with ‘O’ at (0,0)  Position: (0,0) | **Output**:  O | **Reason**:  The given position contains ‘O’, which is different from ‘X’.  **Function name**:  testWhatsAtPos\_O |
| --- | --- | --- |

| **Input**:  State: 3x3 board with ‘A’ at (0,0)  Position: (0,0) | **Output**:  A | **Reason**:  A is outside of the normal X or O for tic-tac-toe. |
| --- | --- | --- |

| **Input**:  State: 3x3 board with ‘B’ at (0,0).  Position: (0,0) | **Output**: B | **Reason**:  B is outside of the normal X or O for tic-tac-toe.  **Function name**:  testWhatsAtPos\_B |
| --- | --- | --- |

| **Input**:  State: 3x3 empty board.  Position: (0,0) | **Output**: ‘ ‘ | **Reason**:  The given position is empty.  **Function name**: testWhatsAtPos\_empty |
| --- | --- | --- |

boolean isPlayerAtPos(BoardPosition pos, char player)

| **Input**:  State: 3x3 board with ‘X’ at (0,0).  Position: (0,0). | **Output**:  True. | **Reason**:  Player X is at (0,0).  **Function name**:  testIsPlayerAtPos\_X\_atPos |
| --- | --- | --- |

| **Input**:  State: 3x3 board with ‘O’ at (0,0).  Position: (0,0) | **Output**:  False. | **Reason**:  Player X is not at (0,0).  **Function name**:  testIsPlayerAtPos\_X\_notAtPos |
| --- | --- | --- |

| **Input**:  State: 3x3 board with ‘O’ at (0,0). Position: (0,0) | **Output**:  True. | **Reason**: Player O is at (0,0), proving that this method works for different letters.  **Function name**:  testIsPlayerAtPos\_O\_atPos |
| --- | --- | --- |

| **Input**:  State: 3x3 board with ‘X’ at (0,0).  Position: (0,0) | **Output**:  False. | **Reason**:  Player ‘O’ is not at (0,0), proving that this method returns the correct response no matter what letter is used.  **Function name**:  testIsPlayerAtPos\_O\_notAtPos |
| --- | --- | --- |

| **Input**:  State: 3x3 empty board.  Position: (0,0) | **Output**:  False. | **Reason**:  The board is empty so the player is not present at (0,0).  **Function name**: testIsPlayerAtPos\_emptyPos() |
| --- | --- | --- |

| **Input**:  State: 3x3 empty board.  Position: (0,0) | **Output**:  State: ‘X’ at (0,0). | **Reason**:  The board was empty before a marker was placed.  **Function name**:  testPlaceMarker\_X\_emptySpot |
| --- | --- | --- |

| **Input**:  State: 3x3 empty board.  Position: (0,0) | **Output**:  State: ‘O’ at (0,0). | **Reason**:  This shows the method works no matter which player is playing.  **Function name**:  testPlaceMarker\_O\_emptySpot |
| --- | --- | --- |

| **Input**: State: 3x3 board with ‘T’ at (0,0).  Position: (0,0) | **Output**:  State: ‘X’ at (0,0). | **Reason**:  This shows that placeMarker can overwrite old data.  **Function name**:  testPlaceMarker\_X\_takenSpot |
| --- | --- | --- |

| **Input**:  State: 3x3 board with ‘T’ at (0,0).  Position: (0,0) | **Output**:  State: ‘O’ at (0,0). | **Reason**:  This shows that placeMarker can overwrite data no matter which player is playing.  **Function name**:  testPlaceMarker\_O\_takenSpot |
| --- | --- | --- |

| **Input**:  State: 3x3 board with ‘T’ at (0,0).  Position: (0,0) | **Output**:  State 3x3 empty board. | **Reason**:  This shows that placeMarker can be used to clear spots on the board.  **Function name**:  testPlaceMarker\_clearSpot |
| --- | --- | --- |

Part 4, Deployment:

To run the program:

1. Navigate to the project directory on your command line terminal.
2. Enter the command “make”.
3. Enter the command “make run”.

To test the program :

1. Navigate to the project directory on your command line terminal.
2. Enter the command “make test”.
3. Enter the command “make testGB”.
   1. Note: to test in memory efficient mode instead of fast mode, append this command with “mem”.