Manning Graham

Project 5

CPSC 2150

11-30-2021

Requirements Analysis

Functional Requirements:

1) Player

a) Place Token: As a player, I can decide where I wish to place my token during the game so I can play and try to win.

b) Exit Game: As a player, after a game is completed I have the option of whether or not to continue playing in a new game or exit the game.

c) Win game: As a player, after I place a piece, I am able to win 3 different ways. By having five pieces in a row horizontally, vertically, or diagonally.

d) Tie game: As a player, I can tie the game if there are no options of winning left for me or my opponent.

e) Out of bounds: As a player, if I choose a placement for a token out of bounds I am able to choose another placement to continue playing without error.

f) Playing again: As a player, after a game is completed I have the option of playing again.

g) Moving after opponent: As a player after my opponent has made their move, It is then my turn and I am able to make my move.

h) Winning Vertically: As a player, I am able to see if I have 5 of the same tokens stacked on top of one another so I can see If I have won the game

I) Winning Horizontally: As a player, I am able to see if I have 5 of the same tokens side by side of one another so I can see If I have won the game

J) Winning Diagonally: As a player, I am able to see if I have 5 of the same tokens stacked diagonally on top of one another, from either left of right, so I can see If I have won the game

e) Board Position Taken: As a player, if I choose a placement for a token that is already taken by another token I am able to choose another placement to continue playing without error.

f) Character: As a player, I can choose which character I would like to play as so I can tell where my positions are, in contrast to my opponent.

g) Number of rows: As I player I can choose the number of rows I would like to play on.

h) Number of columns: As a player, I can choose the number of columns I would like to play on.

I) Number of Markers: As a player, I can choose how many markers it takes in a row to win the game.

J) Character change: As a player, after a game has finished, I have the opportunity to change the character I am playing as if I would like.

K) Number of players: As a player I can choose how many players are going to be playing the game.

Non-Functional Requirements:

1) The program should know whether the player's column input location is valid or not.

2) The program should know whether the column input is even on the board.

3) The program should know whether the player has won after a move vertically.

4) The program should know if there are any more spaces left to be played or if it is a tie.

5) The program should know who the winner is and display that they are the winner immediately after the winning move.

6) The program should know to display a blank game board after a win if the users wish to play again.

7) The program should be able to tell what player is taking up space in a certain position.

8) The program should be able to tell what marker is in what position

9) The program should be able to display the current game board after each move.

11) The program should know 0,0 is the top left of the board.

12) The program should know that Player one goes first.

13) The program should be written in java.

14) The program should run on Unix.

15) The program should establish the Max number of players as 10 and the minimum 2.

16) The Program should establish the maximum number of rows and columns as 100 and minimum is 3.

17) The program should establish the maximum number in a row to win as 25 and minimum is 3

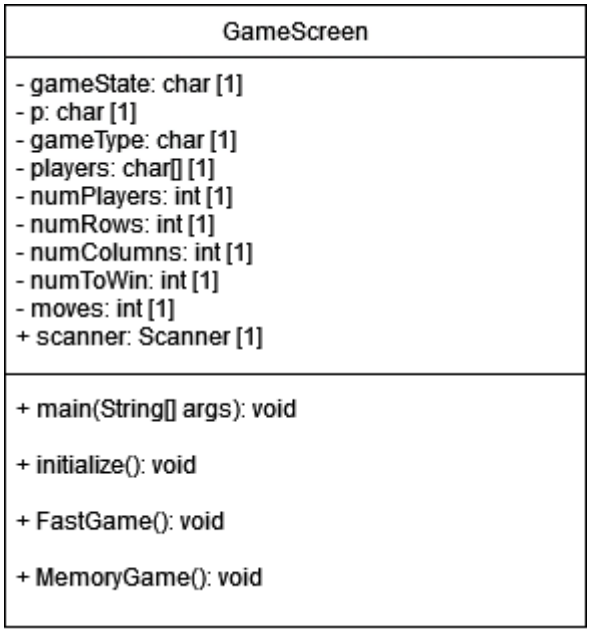
3) The program should know whether the player has won after a move Horizontally.

3) The program should know whether the player has won after a move Diagonally.

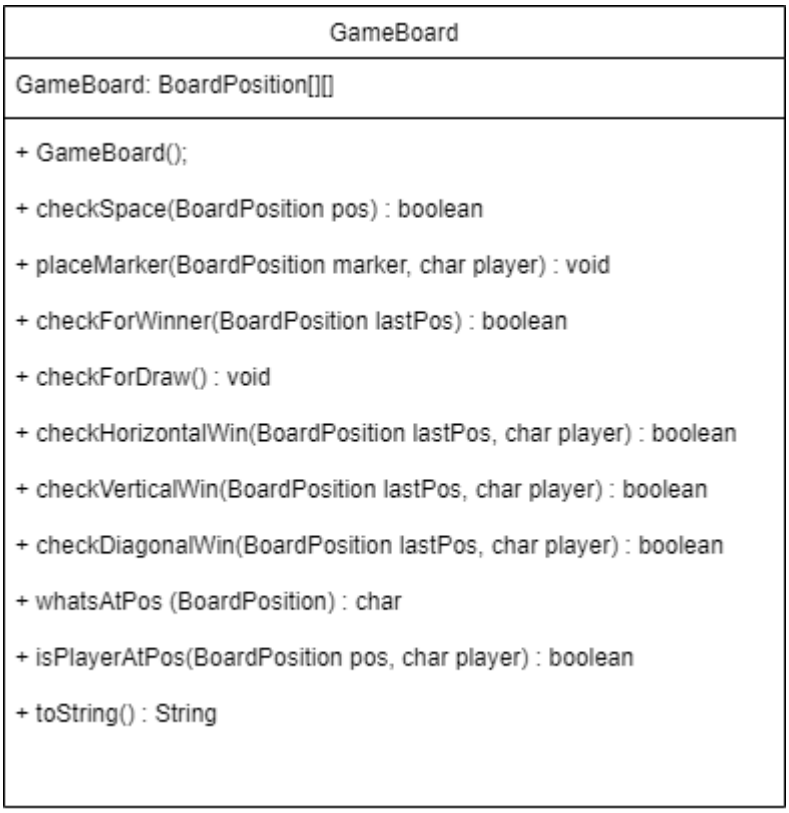
Class Diagrams

**Activity and Class Diagrams for TicTacToeController on last page

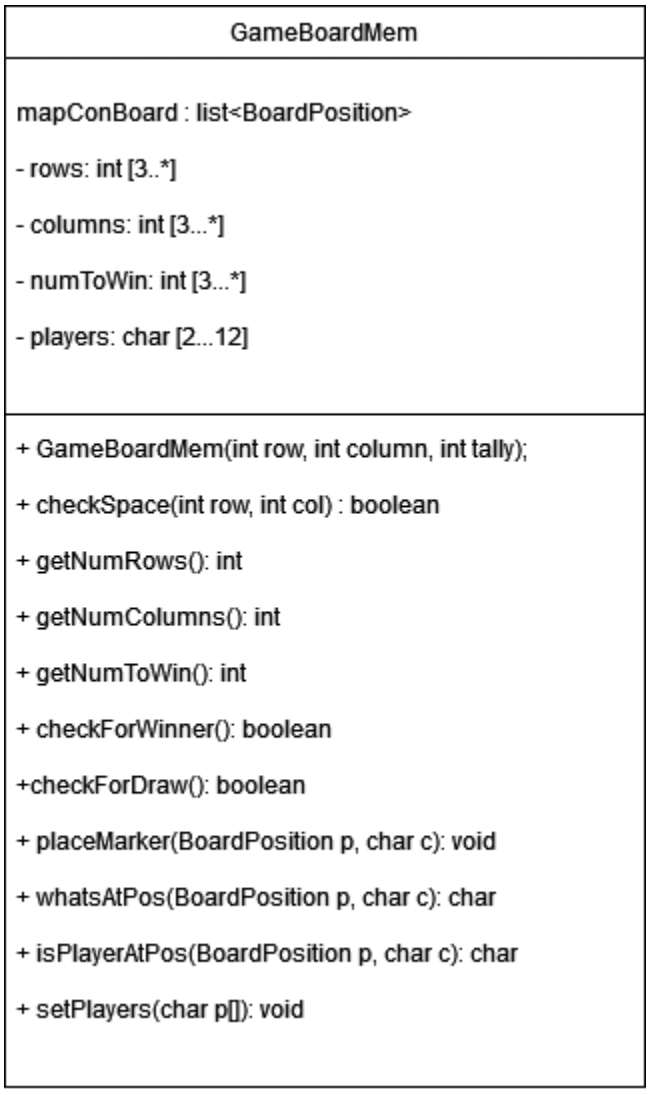
GameScreen:



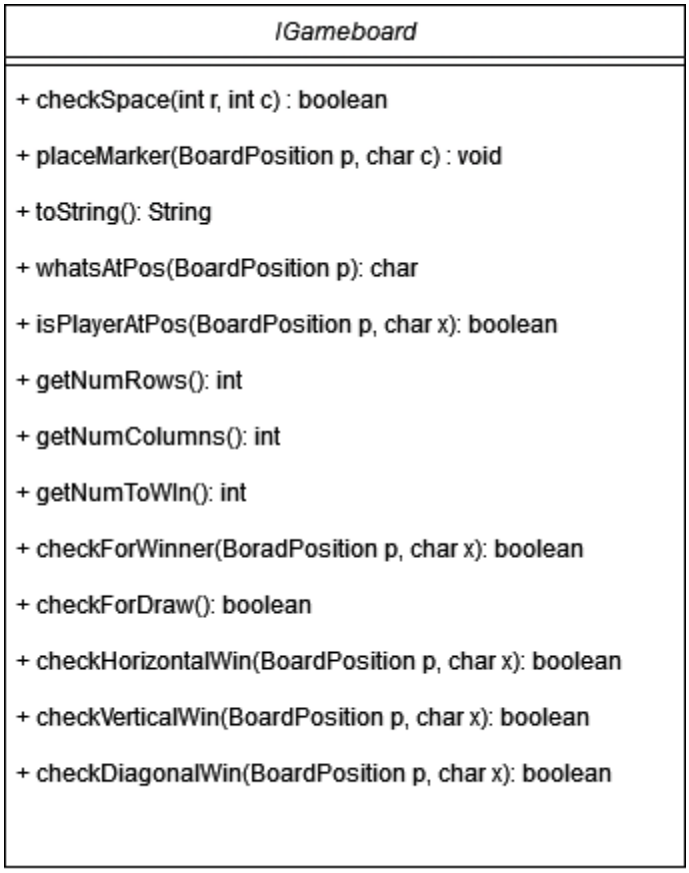
GameBoard:


GameBoard
GameBoard: BoardPosition[][]
+ GameBoard();
+ checkSpace(BoardPosition pos) : boolean
+ placeMarker(BoardPosition marker, char player) : void
+ checkForWinner(BoardPosition lastPos) : boolean
+ checkForDraw() : void
+ checkHorizontalWin(BoardPosition lastPos, char player) : boolean
+ checkVerticalWin(BoardPosition lastPos, char player) : boolean
+ checkDiagonalWin(BoardPosition lastPos, char player) : boolean
+ whatsAtPos (BoardPosition) : char
+ isPlayerAtPos(BoardPosition pos, char player) : boolean
+ toString() : String

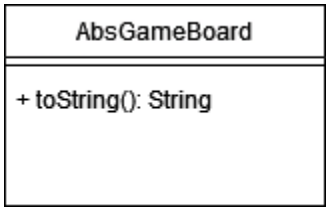
GameBoardMem:



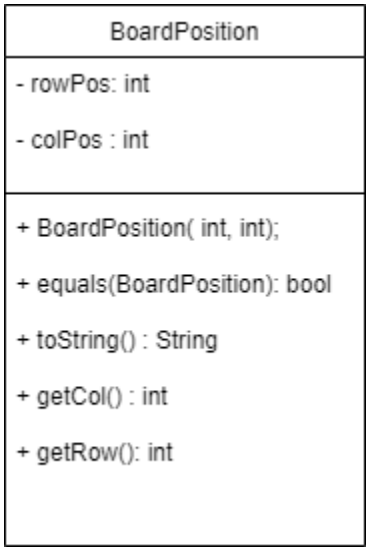
IGameBoard:


IGameboard
+ checkSpace(int r, int c) : boolean
+ placeMarker(BoardPosition p, char c) : void
+ toString(): String
+ whatsAtPos(BoardPosition p): char
+ isPlayerAtPos(BoardPosition p, char x): boolean
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ checkForWinner(BoradPosition p, char x): boolean
+ checkForDraw(): boolean
+ checkHorizontalWin(BoardPosition p, char x): boolean
+ checkVerticalWin(BoardPosition p, char x): boolean
+ checkDiagonalWin(BoardPosition p, char x): boolean

AbsGameBoard:



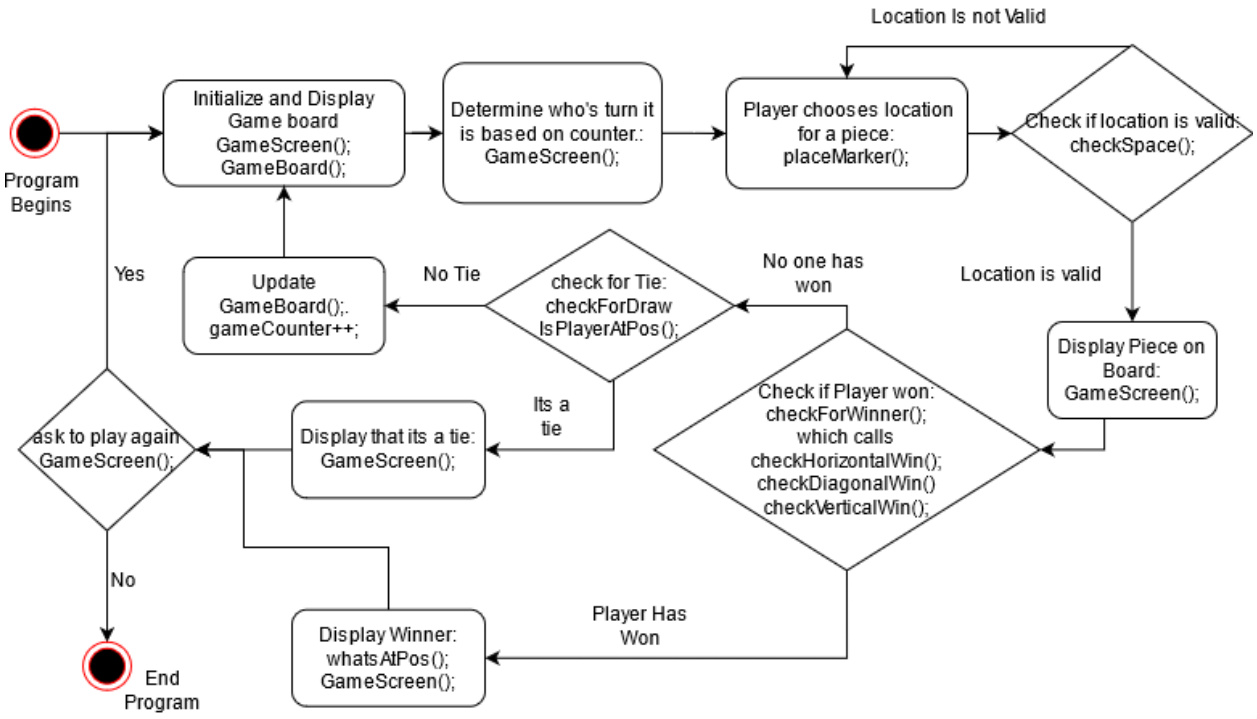
BoardPosition:



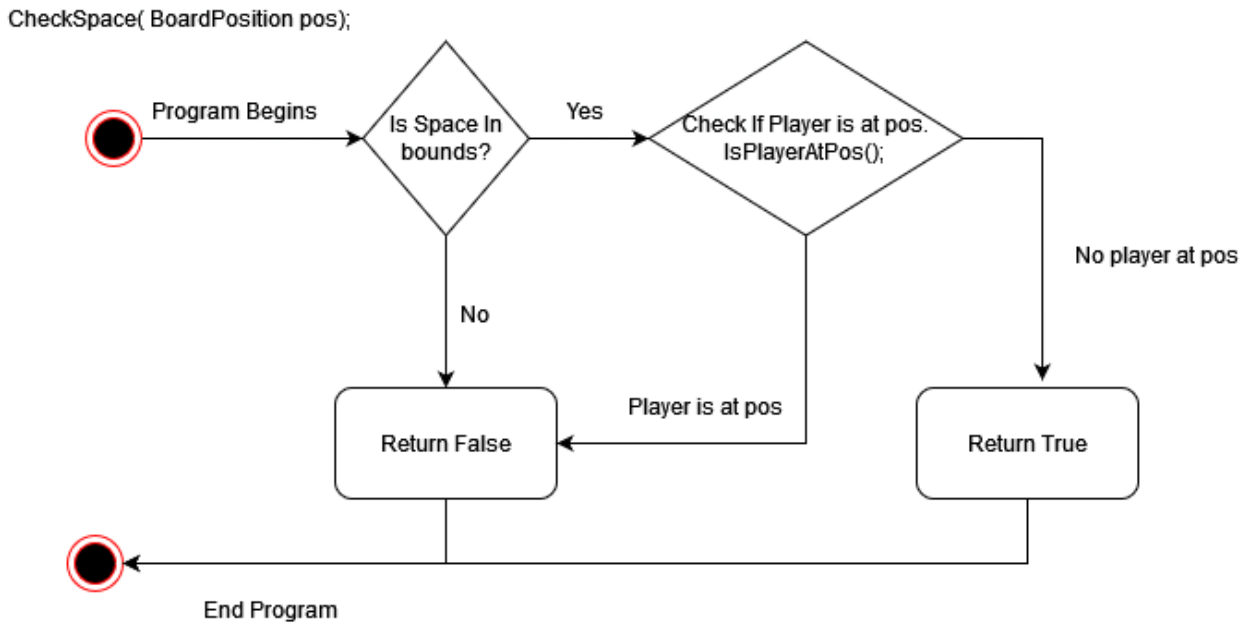
Activity Diagrams

GameScreen Activity Diagram:

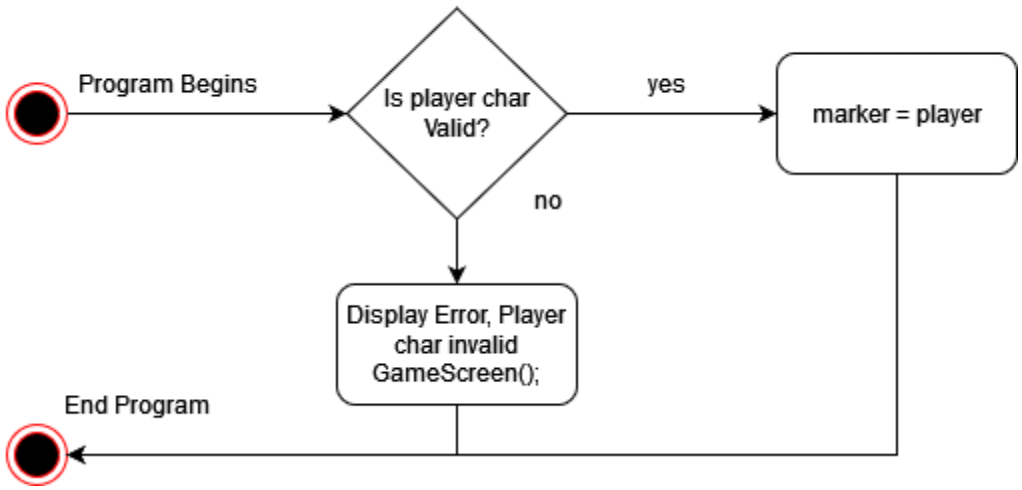
Main:



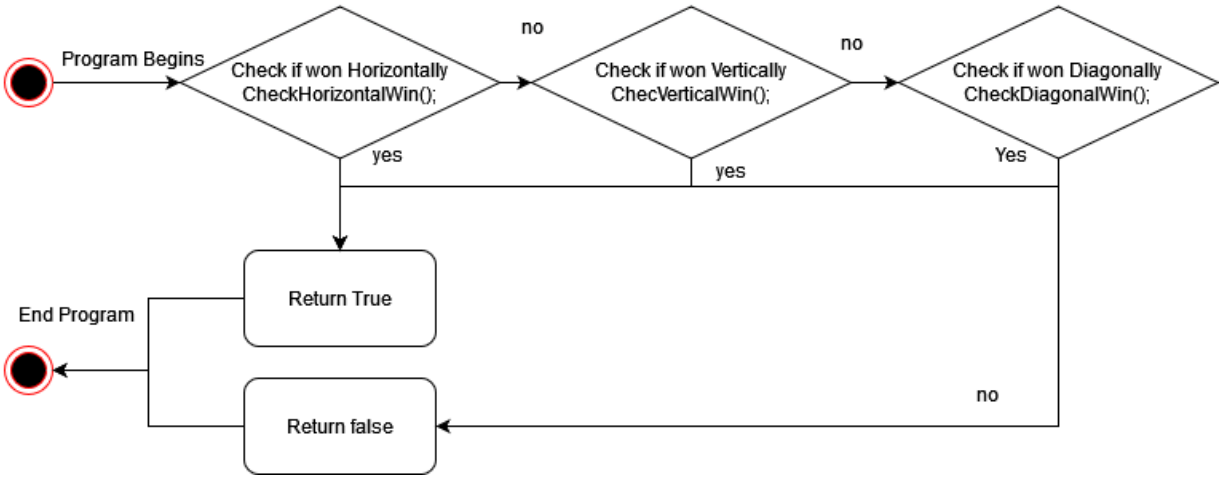
GameBoard Activity Diagrams:



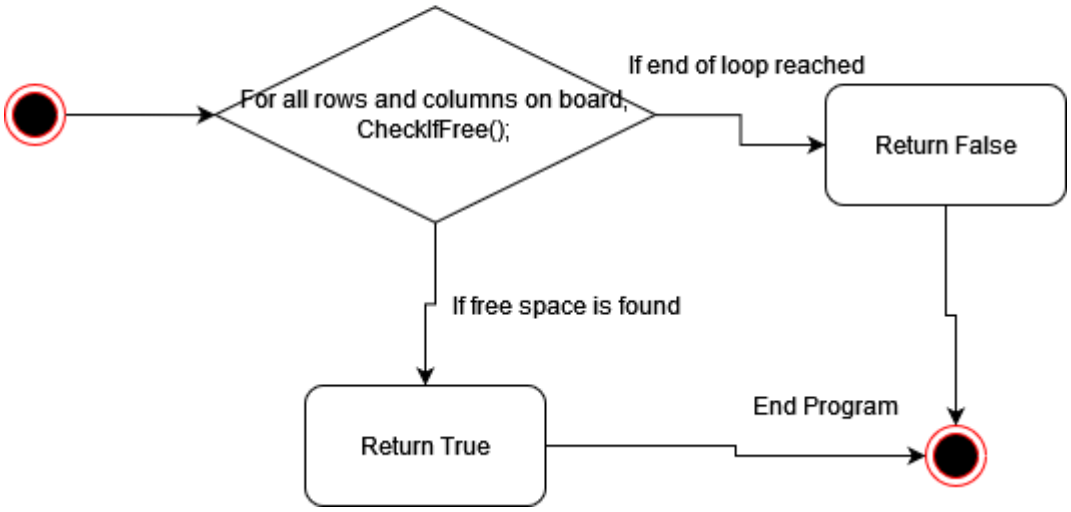
```
void placeMarker(BoardPosition marker, char player)
```



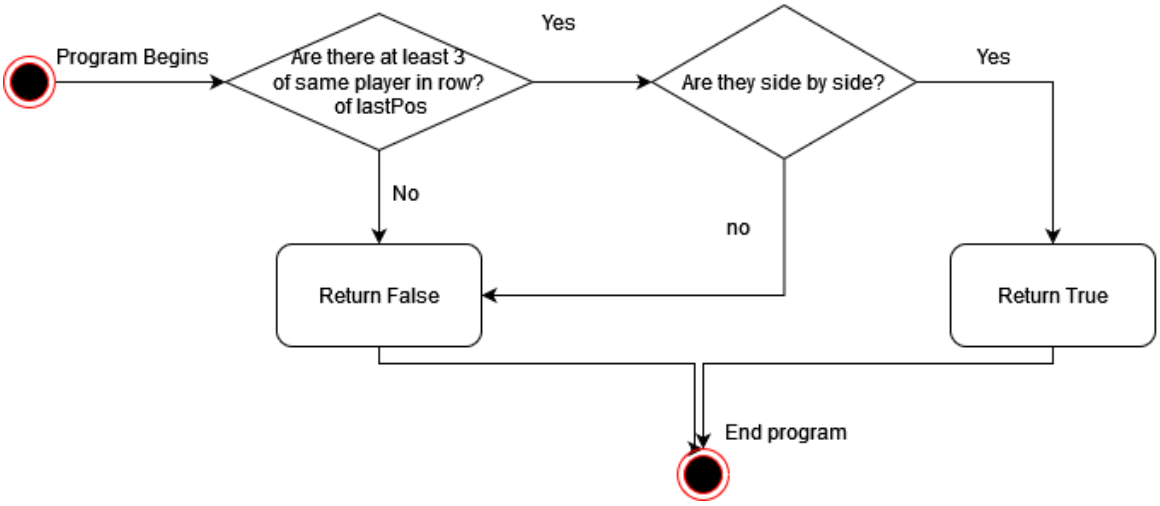

```
public boolean checkForWinner(BoardPosition lastPos)
```



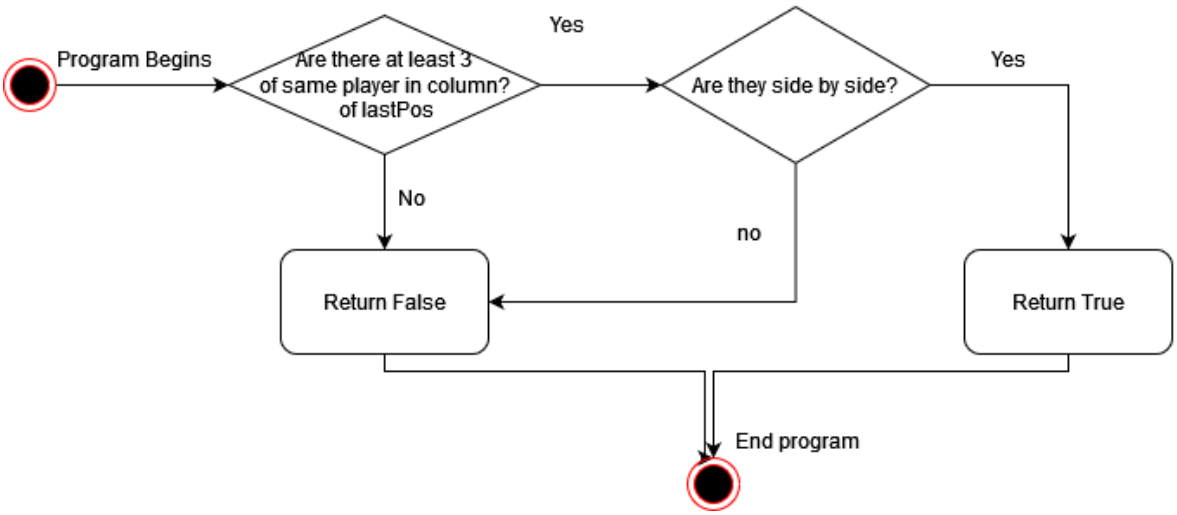
```
public boolean checkForDraw()
```



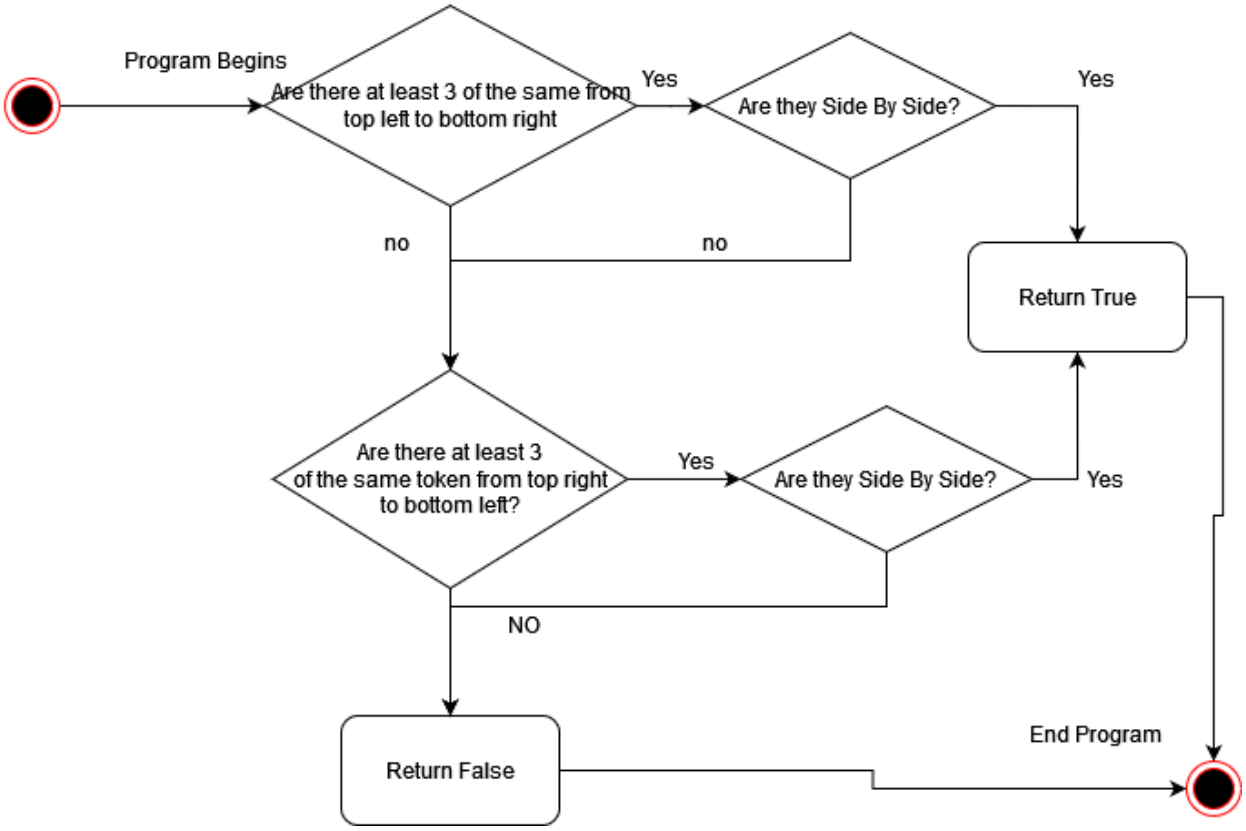

```
publicboolean checkHorizontalWin(BoardPosition lastPos, char player)
```



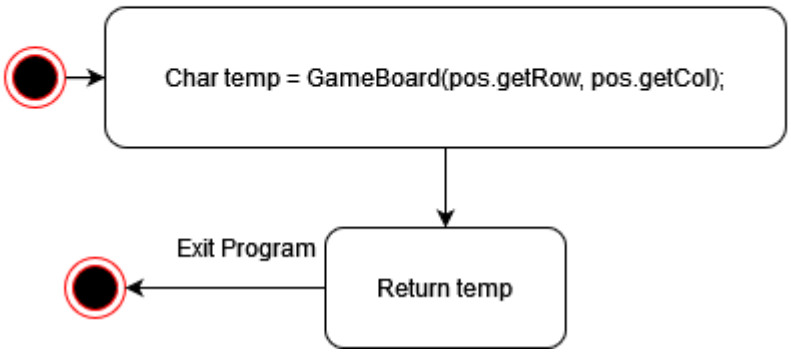
```
publicboolean checkVerticalWin(BoardPosition lastPos, char player)
```



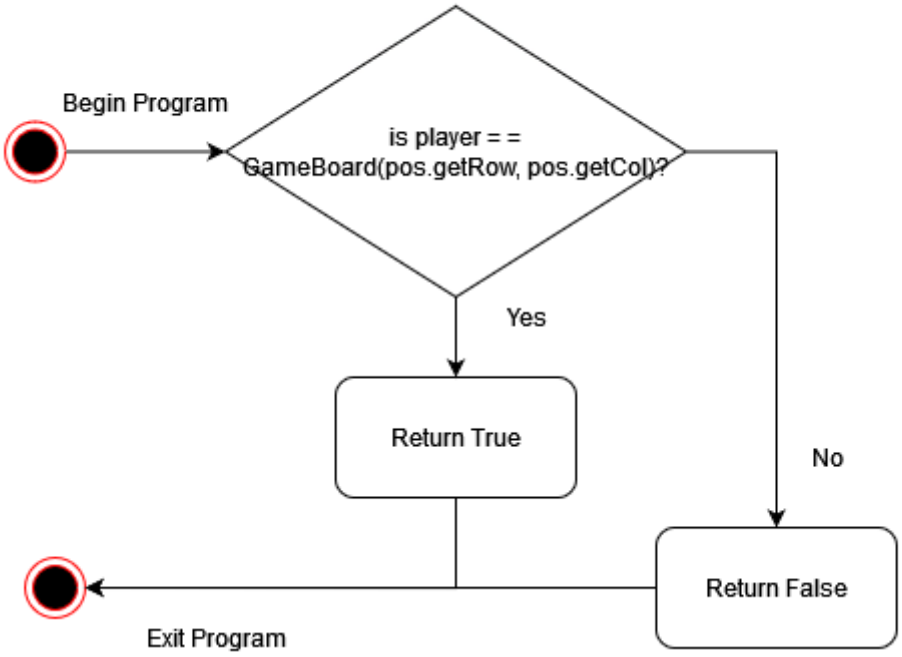
```
publicboolean checkDiagonalWin(BoardPosition lastPos, char player)
```



```
public char whatsAtPos(BoardPosition pos)
```




```
boolean isPlayerAtPos(BoardPosition pos, char player)
```



Testing:

- Constructor Tests:

Test 1:

Function Name: *test_constructor_5x5_()*

Input:

State:

IGameBoard gb = Board(5, 5, 3);

Output:

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

Reason: Conducted so that we know the board can be created if the width and the height are the same.

Test 2:

Function Name: *test_constructor_5x10_()*

Input:

State:

IGameBoard gb = Board(10 , 5, 3);

Output:

```
   | 0 | 1 | 2 | 3 | 4 | 5 |
0 |   |   |   |   |   |   |
1 |   |   |   |   |   |   |
2 |   |   |   |   |   |   |
3 |   |   |   |   |   |   |
4 |   |   |   |   |   |   |
5 |   |   |   |   |   |   |
6 |   |   |   |   |   |   |
7 |   |   |   |   |   |   |
8 |   |   |   |   |   |   |
9 |   |   |   |   |   |   |
```

Reason: Conducted so we know the board can be created if there are more rows than there are columns

Test 3:

Function Name: *test_constructor_10x5_()*

Input:

State: Gameboard not initialized yet

IGameBoard gb = Board(5, 10, 3);

Output:

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |   |
4 |   |   |   |   |   |   |   |   |   |   |
```

Reason: Conducted so we know the board can be created if there are more columns than there are rows

- checkSpace Tests:

Test 1:

Function Name: *test_checkSpace_empty_space_()*

Input:

State:

```
   | 0 | 1 | 2 |
0 |   |   |   |
1 |   |   |   |
2 |   |   |   |
```

Output: checkSpace(1, 1) = True

Reason: Conducted so that we know checkSpace knows when a position is empty

---

Test 2:

Function Name: *test_checkSpace_taken_space_()*

Input:

State:

```
   | 0 | 1 | 2 |
0 |   |   | X |
1 |   |   | O |
2 |   |   |   |
```

Output: checkSpace(1, 2) = False

Reason: Conducted so that we know checkSpace returns false when a position is taken

---

Test 3:

Function Name: *test_checkSpace_space_out_of_bounds_()*

Input:

State:

```
   | 0 | 1 | 2 |
0 |   |   | X |
1 |   |   | O |
2 |   |   |   |
```

Output: checkSpace(101, 101) = False

Reason: Conducted so that we know checkSpace returns false when a position is out of bounds

- checkHorizontalWin tests:

Test 1:

Function Name: *test_Horizontal_Win_middle_()*

Input:

State: (Number to win = 3)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   | X | X | X |   |   |   |
3 |   |   |   | O | O |   |   |   |   |   |
4 |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 2

pos.getColumn = 5

Output: checkHorizontalWin(pos) = True

Reason: Conducted so that we know a horizontal win will be detected if the last piece placed is in the middle of the pattern.

Test 2:

Function Name: *test_Horizontal_Win_left_()*

Input:

State: (Number to win = 3)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   | X | X | X |   |   |   |
3 |   |   |   | O | O |   |   |   |   |   |
4 |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 2

pos.getColumn = 4

Output: checkHorizontalWin(pos) = True

Reason: Conducted so that we know a horizontal win will be detected if the last piece placed is on the left of the pattern.

Test 3:

Function Name: *test_Horizontal_Win_right_()*

Input:

State: (Number to win = 3)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   | X | X | X |   |   |   |
3 |   |   |   | O | O |   |   |   |   |   |
4 |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 2.

pos.getColumn = 6

Output: checkHorizontalWin(pos) = True

Reason: Conducted so that we know a horizontal win will be detected if the last piece placed was on the right of the pattern.

Test 4:

Function Name: *test_Horizontal_middle_empty_()*

Input:

State: (Number to win = 5)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   | X | X |   | X | X |   |
3 |   |   |   | O | O | O |   |   |   |   |
4 |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 2

pos.getColumn = 8

p = 'X'

Output: checkHorizontalWin(pos) = False

Reason: Conducted so that we know no win will be detected if there is a gap in the middle of the consecutive pieces.

- checkVerticalWin Tests:

Test 1:

Function Name: *test_Vertical_Win_middle_()*

Input:

State: (Number to Win = 3)

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

pos.getRow = 3

pos.getColumn = 4

p = 'X'

Output: checkVerticalWin(pos) = True

Reason: Conducted so that we know a vertical win will be detected if the last piece placed was in the middle of the pattern.

Test 2:

Function Name*: test_Vertical_Win_top_()*

Input:

State: (Number to Win = 3)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2					X					
3					X					
4					X					
5				O	O					
6										
7										
8										
9										

pos.getRow = 2

pos.getColumn = 4

p = 'X'

Output: checkVerticalWin = True

Reason: Conducted so that we know a vertical win will be detected if the last piece placed was at the top of the pattern.

Test 3:

Function Name: *test_Vertical_Win_bottom_()*

Input:

State: (Number to Win = 3)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   | X |   |   |   |   |   |
3 |   |   |   |   | X |   |   |   |   |   |
4 |   |   |   |   | X |   |   |   |   |   |
5 |   |   |   | O | O |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 4

pos.getColumn = 4

p = ‘X’

Output: checkVerticalWin = True

Reason: Conducted so that we know a vertical win will be detected if the last piece placed was at the bottom of the pattern.

Test 4:

Function Name*: test_Vertical_middle_empty_()*

Input:

State: (Number to Win = 5)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   | O |   |   |   |   |   |   |
2 |   |   |   |   | X |   |   |   |   |   |
3 |   |   |   |   | X |   |   |   |   |   |
4 |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   | X | O |   |   | O |   |
6 |   |   |   |   | X |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 5

pos.getColumn = 4

p = ‘X’

Output: checkVerticalWin(pos) = False

Reason: Conducted so that we know a vertical win will not be detected if there is a gap in the middle of the pattern.

- checkDiagonalWin Tests:

Test 1:

Function Name: *test_Diagonal_Win_bottom_left_()*

Input:

State: (Number to Win = 3)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   | O |   |   |   |   |   |   |
2 |   |   |   |   |   |   | X |   |   |   |
3 |   |   |   |   |   | X |   |   |   |   |
4 |   |   |   |   | X |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   | O |   |
6 |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 4

pos.getColumn = 4

p = 'X'

Output: checkDiagonalWin(pos) = True

Reason: Conducted so that we know a diagonal win will be detected if the last piece placed was in the bottom left of the pattern.

Test 2:

Function Name: *test_Diagonal_Win_top_right_()*

Input:

State: (Number to Win = 3)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   | O |   |   |   |   |   |   |
2 |   |   |   |   |   |   | X |   |   |   |
3 |   |   |   |   |   | X |   |   |   |   |
4 |   |   |   |   | X |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   | O |   |
6 |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 2

pos.getColumn = 6

p = ‘X’

Output: checkDiagonalWin(pos) = True

Reason: Conducted so that we know a diagonal win will be detected if the last piece placed was in the top right of the pattern.

Test 3:

Function Name: *test_Diagonal_Win_bottom_right_()*

Input:

State: (Number to Win = 3)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   | O |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |   |
4 |   |   |   |   | X |   |   |   |   |   |
5 |   |   |   |   |   | X |   |   | O |   |
6 |   |   |   |   |   |   | X |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 6

pos.getColumn = 6

p = 'X'

Output: checkDiagonalWin(pos) = True

Reason: Conducted so that we know a diagonal win will be detected if the last piece placed was in the bottom right of the pattern.

Test 4:

Function Name: *test_Diagonal_Win_top_left_()*

Input:

State: (Number to Win = 3)

```
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
0 |   |   |   |   |   |   |   |   |   |   |
1 |   |   |   | O |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |   |
4 |   |   |   |   | X |   |   |   |   |   |
5 |   |   |   |   |   | X |   |   | O |   |
6 |   |   |   |   |   |   | X |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |
```

pos.getRow = 4

pos.getColumn = 4

p = 'X'

Output: checkDiagonalWin(pos) = True

Reason: Conducted so that we know a diagonal win will be detected if the last piece placed was in the top left of the pattern.

Test 5:

Function Name: *test_Diagonal_2nd_to_last_from_right_()*

Input:

State: (Number to Win = 3)

0| 1| 2| 3| 4| 5| 6| 7| 8| 9|

0| | | | | | | | |X |

1| | | | | | | |X | |

2| | | | | | |X | | |

3| | | | | |X | | | |

4| | | | |X | | | | |

5| | | | | | | | | |

6| | | | | | | | | |

7| | | | | | | | | |

8| | | | | | | | | |

9| | | | | | | | | |

pos.getRow = 1

pos.getColumn = 8

p = 'X'

Output: checkDiagonalWin(pos) = True

Reason: conducted so that we know a diagonal win will be detected if the last piece placed was in the corner second to last position from the right, ensuring that the checkDiagonalWin method will check both from the right and then from the left.

Test 6:

Function Name: *test_Diagonal_second_to_last_from_left_()*

Input:

State: (Number to Win = 3)

```
  0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
0|  |  |  |  |  |  |  |  |  |
1|  |  |  |  |  |  |  |  |  |
2|  |  |  |  |  |  |  |  |  |
3|  |  |  |  |  |  |  |  |  |
4|  |  |  |  |  |  |  |  |  |
5|  |  |  |X |  |  |  |  |  |
6|  |  |  |  |X |  |  |  |  |
7|  |  |  |  |  |X |  |  |  |
8|  |  |  |  |  |  |X |  |  |
9|  |  |  |  |  |  |  |X |  |
```

pos.getRow = 6

pos.getColumn = 5

p = ‘X’

Output: checkDiagonalWin(pos) = True

Reason: conducted so that we know a diagonal win will be detected if the last piece placed was in the second to last from the left of the sequence, ensuring the checkDiagonalWin method will check for a win both from the left and then from the right .

Test 7:

Function Name: *test_Diagona_middle_()*

Input:

State: (Number to Win = 3)

  0| 1| 2| 3| 4| 5| 6| 7| 8| 9|

0| | | | | | | | | | |

1| | | | | | | | | | |

2| | | | | | | | | | |

3| | | | | | | | | | |

4| | | | | | | | | | |

5| | | | | |X | | | | |

6| | | | | | |X | | | |

7| | | | | | | |X | | |

8| | | | | | | | |X | |

9| | | | | | | | | |X |

pos.getRow = 7

pos.getColumn = 7

p = 'X'

Output: checkDiagonalWin(pos) = True

Reason: conducted so that we know a diagonal win will be detected if the last piece placed was in the middle of the sequence, ensuring the checkDiagonal method will check both from the right and then from the left.

- checkForDraw Tests:

Test 1:

Function Name: *test_Tie_full_board()*

Input:

State: (Number to Win = 5)

```
  0| 1| 2| 3| 4|
0|X |O |X |O |X |
1|O |X |O |X |O |
2|X |O |O |O |X |
3|O |X |O |X |O |
4|X |O |X |O |X |
```

Output: checkForDraw() = True.

Reason: Conducted so that we know a draw will be detected if the board is full.

Test 2:

Function Name: *test_Tie_full_board()*

Input:

State: (Number to Win = 3)

```
  0| 1| 2|
0|X |O |O |
1|X |X |  |
2|  |  |  |
```

Output: checkForDraw() = False.

Reason: Conducted so that we know a draw will not be detected if the board is only half full.

Test 3:

Function Name: *test_Tie_empty_board()*

Input:

State: (Number to Win = 3)

  0| 1| 2|

0| | | |

1| | | |

2| | | |

Output: checkForDraw() = False.

Reason: Conducted so that we know a draw will not be detected if the board is empty.

Test 4:

Function Name: *test_Tie_one_left()*

Input:

State: (Number to Win = 5)

  0| 1| 2| 3| 4|

0| |O |X |O |X |

1|O |X |O |X |O |

2|X |O |O |O |X |

3|O |X |O |X |O |

4|X |O |X |O |X |

Output: checkForDraw = False.

Reason: Conducted so that we know a draw will not be detected if there is only one space left on the board.

- whatsAtPos Tests:

Test 1:

Function Name *test_whatsAtPos_empty_slot()*

Input:

State:

  0| 1| 2|

0| | | |

1| | | |

2| | | |

pos.getRow = 0.

pos.getCol = 0.

Output: whatsAtPos(pos) = ‘ ‘.

Reason: Conducted to see if the character can be recognized based on position, in this case an empty slot, so it should be a space.

Test 2:

Function Name *test_whatsAtPos_top_right_corner()*

Input:

State:

  0| 1| 2|

0| | | X |

1| | | |

2| | | |

pos.getRow = 0.

pos.getCol = 2.

Output: whatsAtPos(pos) = ‘X‘.

Reason: Conducted to see if the character can be recognized based on position, in this case the top right corner, so it should be an X.

Test 3:

Function Name: *test_whatsAtPos_top_left_corner()*

Input:

State:

  0| 1| 2|

0| X | | |

1| | | |

2| | | |

pos.getRow = 0.

pos.getCol = 0.

Output: whatsAtPos(pos) = 'X'.

Reason: Conducted to see if the character can be recognized based on position, in this case the top left corner, so it should be an X.

Test 4:

Function Name *test_whatsAtPos_bottom_left_corner()*

Input:

State:

  0| 1| 2|

0| | | |

1| | | |

2|X | | |

pos.getRow = 2.

pos.getCol = 0.

Output: whatsAtPos(pos) = 'X'.

Reason: Conducted to see if the character can be recognized based on position, in this case the bottom left corner, so it should be an X.

Test 5:

Function Name *test_whatsAtPos_bottom_right_corner()*

Input:

State:

  0| 1| 2|

0| | | |

1| | | |

2| | | X |

pos.getRow = 2.

pos.getCol = 2.

Output: whatsAtPos(pos) = 'X'.

Reason: Conducted to see if the character can be recognized based on position, in this case the bottom right corner, so it should be an X.

- <u>isPlayerAtPos Tests:</u>

Test 1:

Function Name *test_isPlayerAtPos_empty_board()*

Input:

State:

  0| 1| 2|

0| | | |

1| | | |

2| | | |

pos.getRow = 1.

pos.getCol = 1.

Output: isPlayerAtPos(pos, 'X') = False.

Reason: Conducted to see if a certain character can be recognized based on position, in this case an empty board so the method should return that the character is not there.

Test 2:

Function Name *test_isPlayerAtPos_top_right_corner()*

Input:

State:

  0| 1| 2|

0| | | X |

1| | | O |

2| | | |

pos.getRow = 0.

pos.getCol = 2.

Output: isPlayerAtPos(pos, 'X') = True.

Reason: Conducted to see if a certain character can be recognized based on position, in this case a character at the top right of the board so the method should return that the character 'X' is there.

Test 3:

Function Name *test_isPlayerAtPos_top_left_corner()*

Input:

State:

  0| 1| 2|

0| X | | |

1| | | O |

2| | | |

pos.getRow = 0.

pos.getCol = 0.

Output: isPlayerAtPos(pos, 'X') = True.

Reason: Conducted to see if a certain character can be recognized based on position, in this case a character at the top left of the board so the method should return that the character 'X' is there.

Test 4:

Function Name *test_isPlayerAtPos_bottom_left_corner()*

Input:

State:

  0| 1| 2|

0| | | |

1| | | O |

2| X | | |

pos.getRow = 2.

pos.getCol = 0.

Output: isPlayerAtPos(pos, 'X') = True.

Reason: Conducted to see if a certain character can be recognized based on position, in this case a character at the bottom left of the board so the method should return that the character 'X' is there.

Test 5:

Function Name *test_isPlayerAtPos_bottom_right_corner()*

Input:

State:

  0| 1| 2|

0| | | |

1| | | O |

2| | | X |

pos.getRow = 2.

pos.getCol = 2.

Output: isPlayerAtPos(pos, 'X') = True.

Reason: Conducted to see if a certain character can be recognized based on position, in this case a character at the bottom right of the board so the method should return that the character 'X' is there.

- placeMarker() Tests:

Test 1:

Function Name test_placeMarker_2players_on_board()

Input:

State:

```
  0| 1| 2| 3| 4|
0|  |  |  |  |  |
1|  |  |  |  |  |
2|  |  |  |  |  |
3|  |  |  |  |  |
4|  |  |  |  |  |
```

Pos1.getRow = 2

Pos1.getColumn = 2

Pos2.getRow = 2

Pos2.getColumn = 4

placeMarker(pos1, 'X')

placeMarker(pos2, 'O')

Output:

```
  0| 1| 2| 3| 4|
0|  |  |  |  |  |
1|  |  |  |  |  |
2|  |  |X |  |O |
3|  |  |  |  |  |
4|  |  |  |  |  |
```

Reason: conducted to see if placeMarker places the respective tokens of the two players in the correct separate locations

Test 2:

Function Name test_placeMarker_bottom_right_corner()

Input:

State:

```
  0| 1| 2| 3| 4|
0|  |  |  |  |  |
1|  |  |  |  |  |
2|  |  |  |  |  |
3|  |  |  |  |  |
4|  |  |  |  |  |
```

pos.getRow = 4

pos.getColumn = 4

placeMarker(pos, 'X')

Output:

```
  0| 1| 2| 3| 4|
0|  |  |  |  |  |
1|  |  |  |  |  |
2|  |  |  |  |  |
3|  |  |  |  |  |
4|  |  |  |  |X |
```

Reason: conducted to see if placeMarker places the respective token in the correct location, in this case the bottom right corner of the board.

Test 3:

Function Name test_placeMarker_top_right_corner()

Input:

State:

```
  0| 1| 2| 3| 4|
0|  |  |  |  |  |
1|  |  |  |  |  |
2|  |  |  |  |  |
3|  |  |  |  |  |
4|  |  |  |  |  |
```

pos.getRow = 0

pos.getColumn = 4

placeMarker(pos, 'X')

Output:

```
  0| 1| 2| 3| 4|
0|  |  |  |X |  |
1|  |  |  |  |  |
2|  |  |  |  |  |
3|  |  |  |  |  |
4|  |  |  |  |  |
```

Reason: conducted to see if placeMarker places the respective token in the correct location, in this case the top right corner of the board.

Test 4:

Function Name test_placeMarker_top_left_corner()

Input:

State:

```
  0| 1| 2| 3| 4|
0|  |  |  |  |  |
1|  |  |  |  |  |
2|  |  |  |  |  |
3|  |  |  |  |  |
4|  |  |  |  |  |
```

pos.getRow = 0

pos.getColumn = 0

placeMarker(pos, 'X')

Output:

```
  0| 1| 2| 3| 4|
0|X |  |  |  |  |
1|  |  |  |  |  |
2|  |  |  |  |  |
3|  |  |  |  |  |
4|  |  |  |  |  |
```

Reason: conducted to see if placeMarker places the respective token in the correct location, in this case the top left corner of the board.

Test 5:

Function Name test_placeMarker_bottom_left_corner()

Input:

State:

  0| 1| 2| 3| 4|

0| | | | | |

1| | | | | |

2| | | | | |

3| | | | | |

4| | | | | |

pos.getRow = 4

pos.getColumn = 0

placeMarker(pos, 'X')

Output:

  0| 1| 2| 3| 4|

0| | | | | |

1| | | | | |

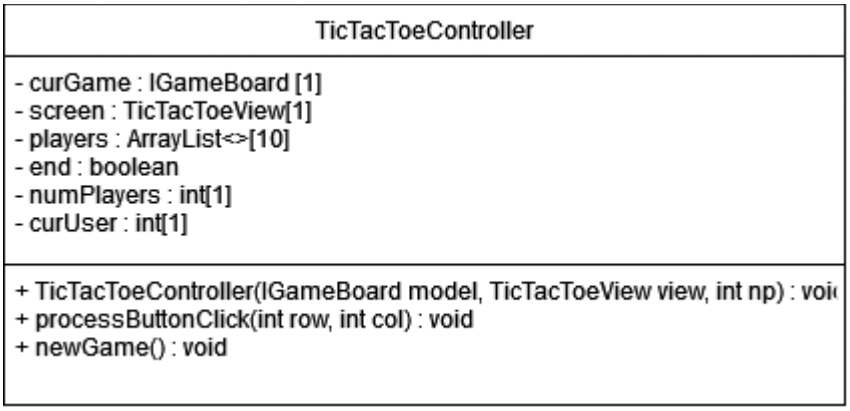
2| | | | | |

3| | | | | |

4| X | | | | |

Reason: conducted to see if placeMarker places the respective token in the correct location, in this case the bottom left corner of the board.

TicTacToeController Class Diagram



TicTacToeController Activity Diagram

```
public void processButtonClick(int row,int col)
```