CPSC 2150 Project Report

David Hayden Copeland

Requirements Analysis

Functional Requirements:

1. As a player I receive clear instructions so that I know how to play the game
2. As a Player I can Input my game-move so that I can play the game
3. As a Player I can clearly see the board after each turn so that I can make my next decision based on what moves are available
4. As a Player I can clearly see the board after each turn so that I can see the moves that the opponent made
5. As a player, I need the game to let me know if I win or lose so that I can decide if I want to play another game
6. As a player I cannot pick a position that is already played or a position so that I do not waste my turn
7. As a player I can choose to start again by clicking another space after a win or draw so I don't need to restart the program to play again
8. As a player, I need the game to congratulate me so that there is added validation to playing well
9. As a player, I can enter the number of rows so that I can create different game scenarios to player
10. As a player, I can enter the number of columns so that I can create different game scenarios to player
11. As a player, I can enter the number of marks in a row to win so that I can create different game scenarios to player
12. As a player, I can select a row & column combination so that I can indicate which column I want to play on
13. As a player I can distinguish between my opponents input and my own, so I do not get confused who's played in which spot on the board

Non-Functional Requirements

1. The program is written in Java
2. The game must function according to the user stories
3. System must run in a reasonable amount of time – little to no input/output delay
4. Output must correctly display the current gameboard with previous moves from both sides
5. The game implements Java GUI to make the game is more enjoyable to look at
6. The game must be adaptable and able to run on other systems
7. Code must be organized and readable so that it is easy to debug, read and change things as needed
8. The game is a looped event so that the game can be restarted after it is won, lost, or drawn.

Deployment Instructions

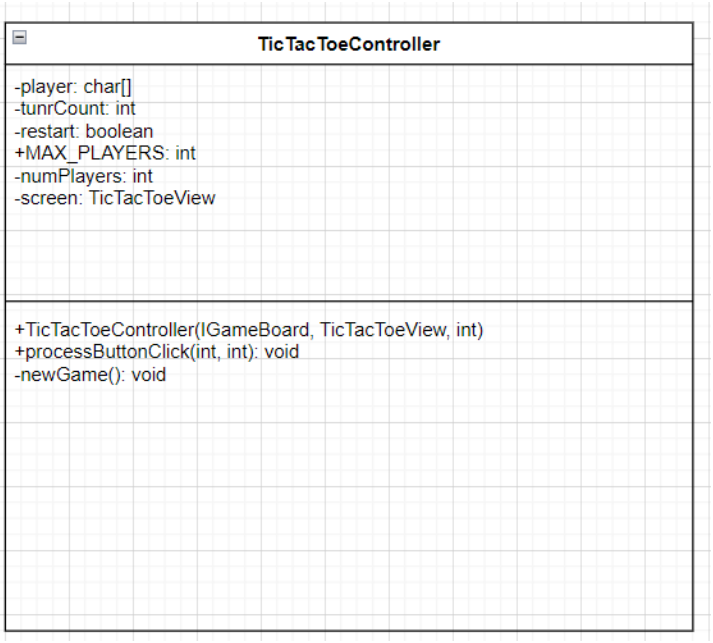
Details in Projects 2-5.

“You do not need to provide anything for the deployment of this program. We will not be able to run this program from our SoC Unix command line since it uses a GUI, so you do not need to provide a makefile. See the Submission section below for more details.”

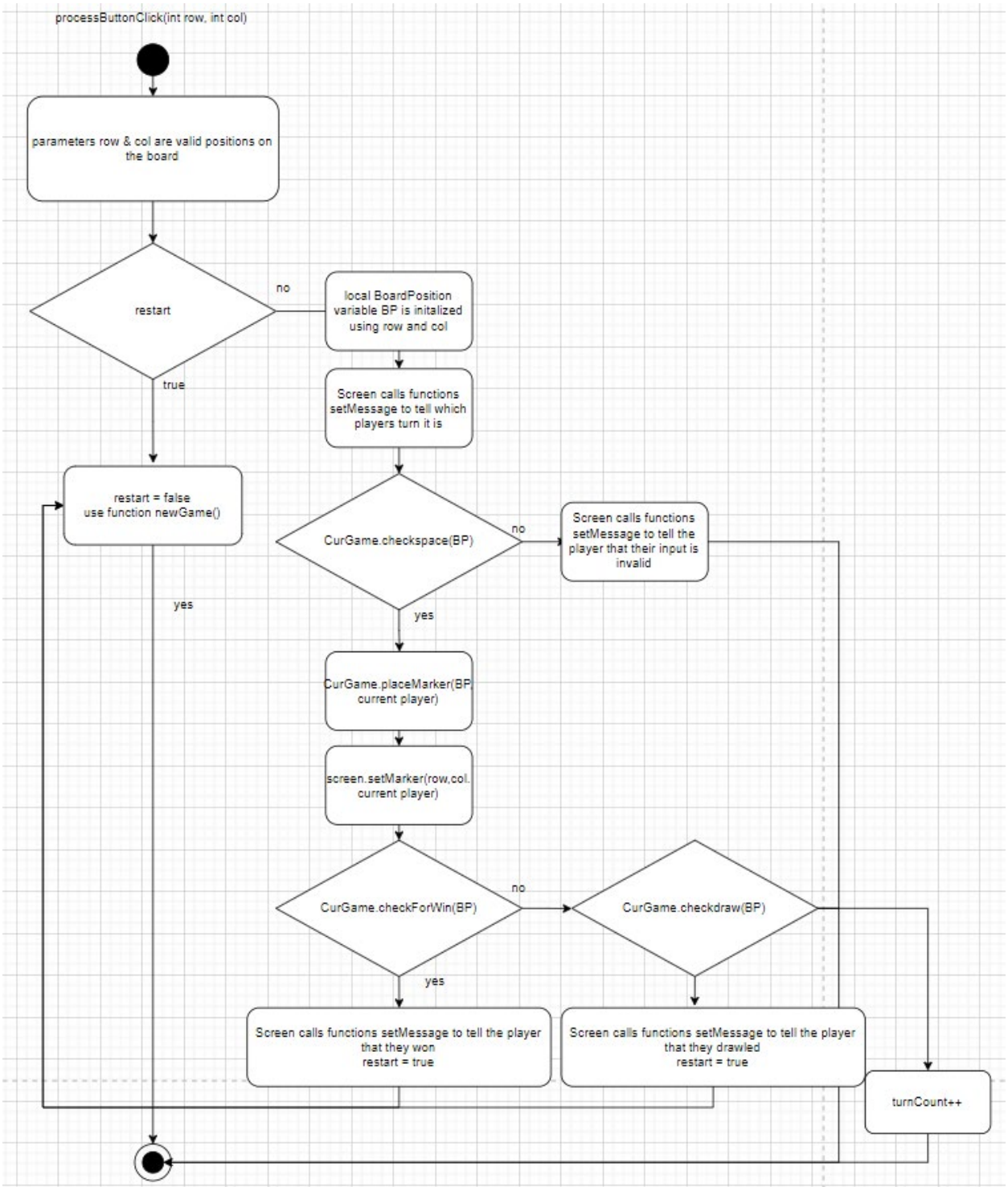
System Design

TicTacToeController:

ClassDiagram

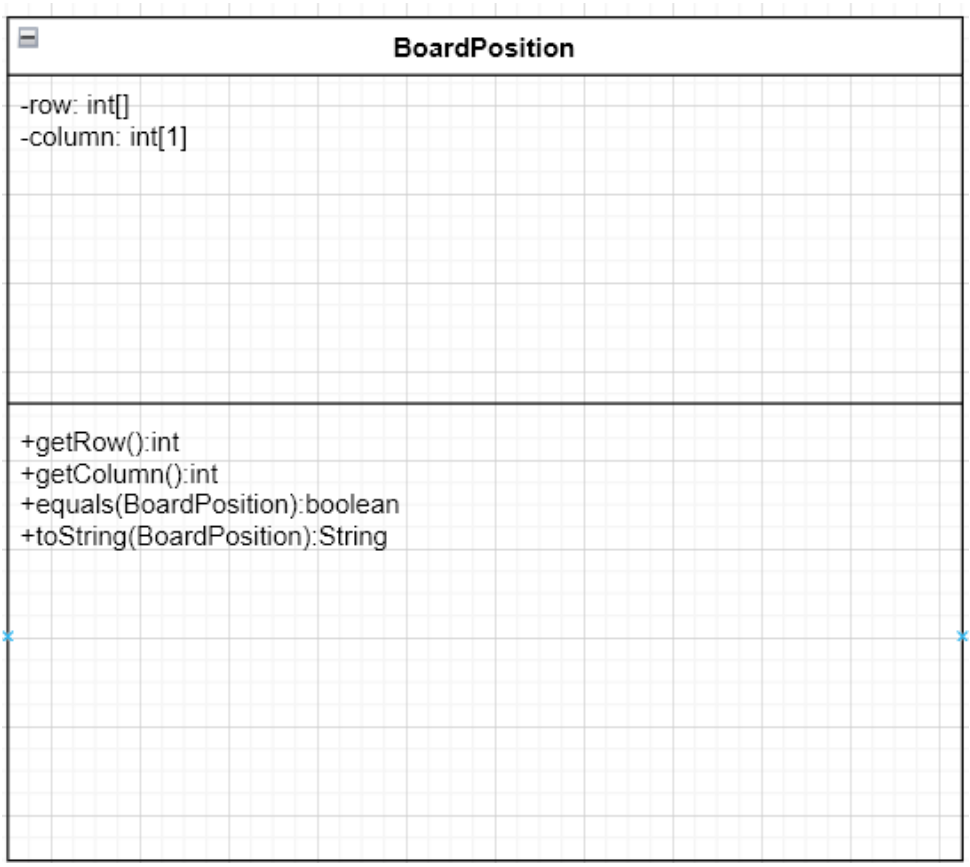


Activity diagram:



BoardPosition:

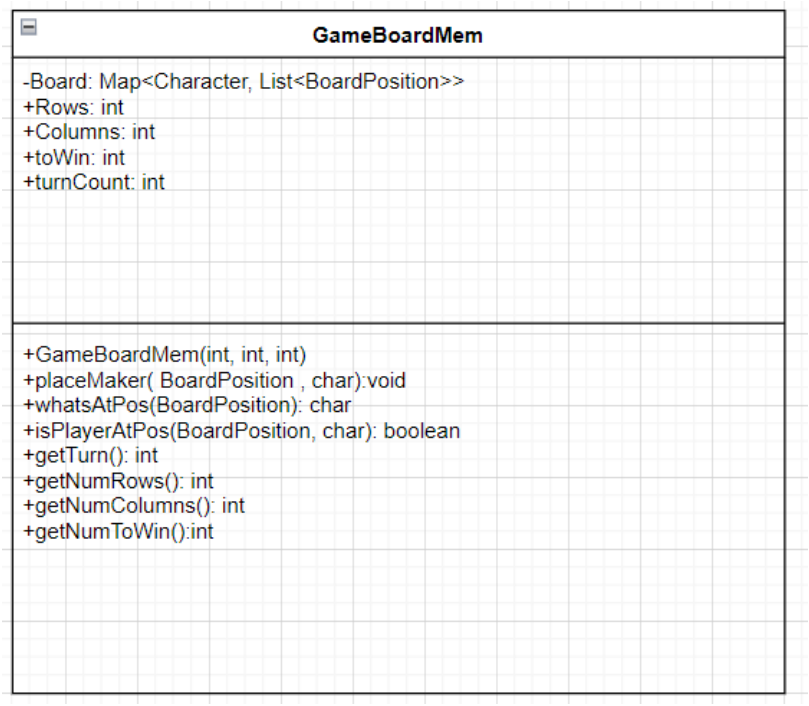
Class diagram



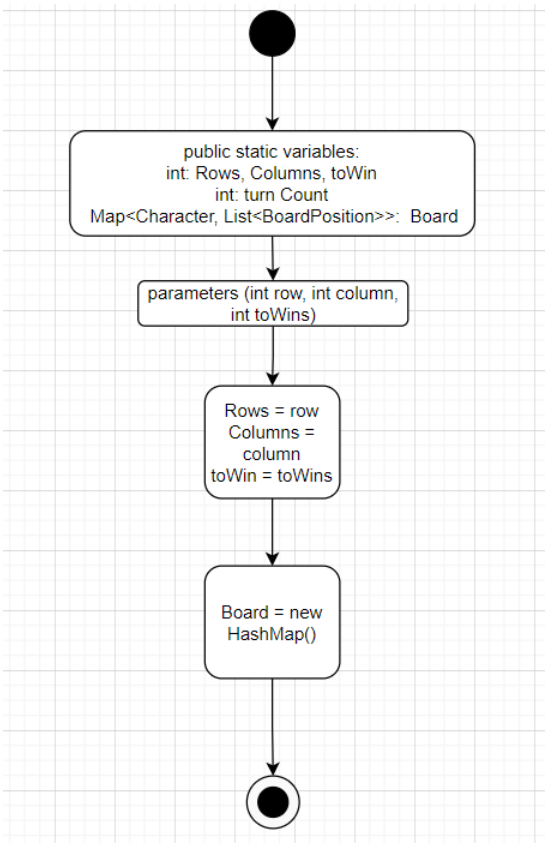
Activity diagrams

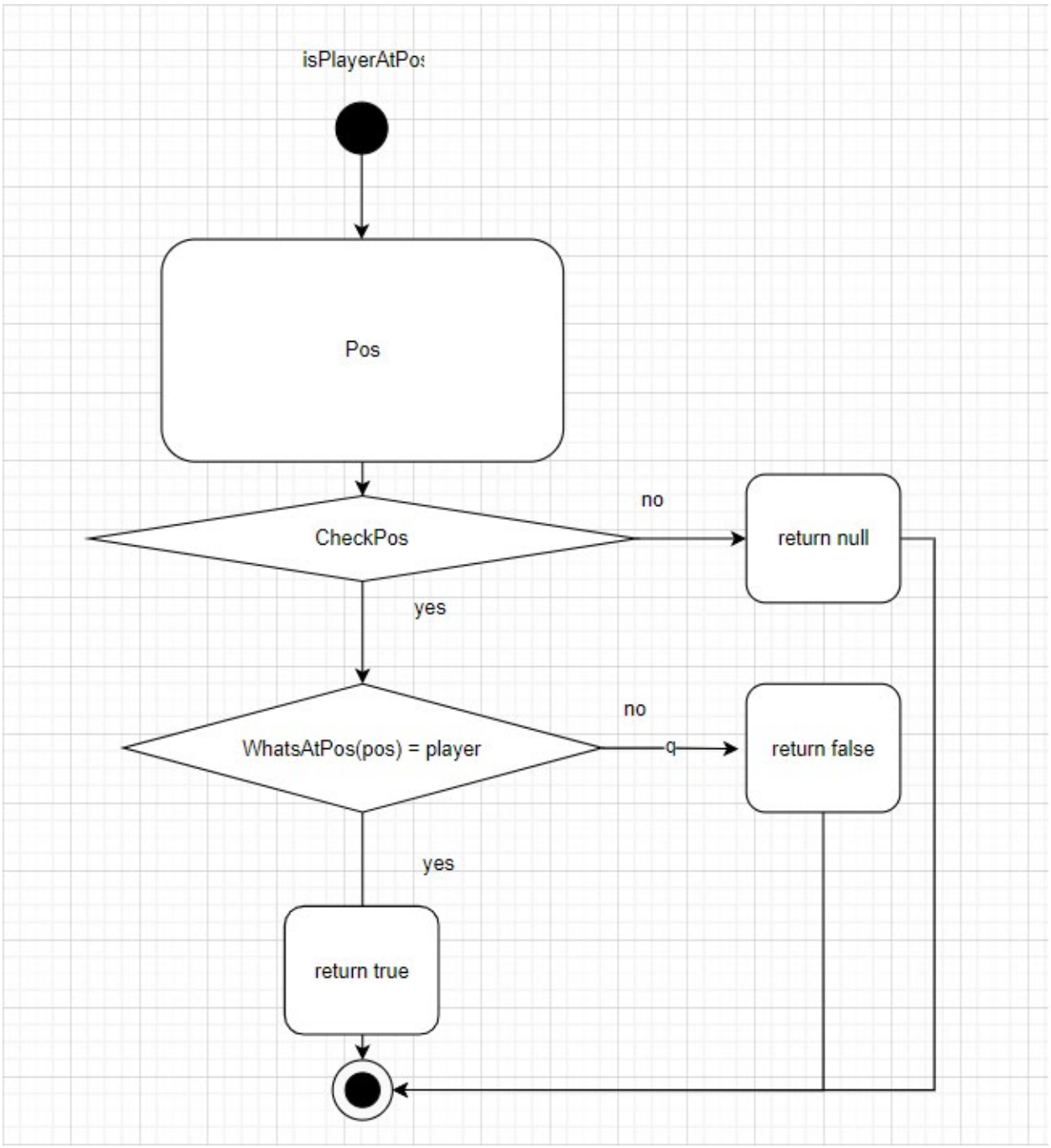
GameboardMem:

Class diagram:



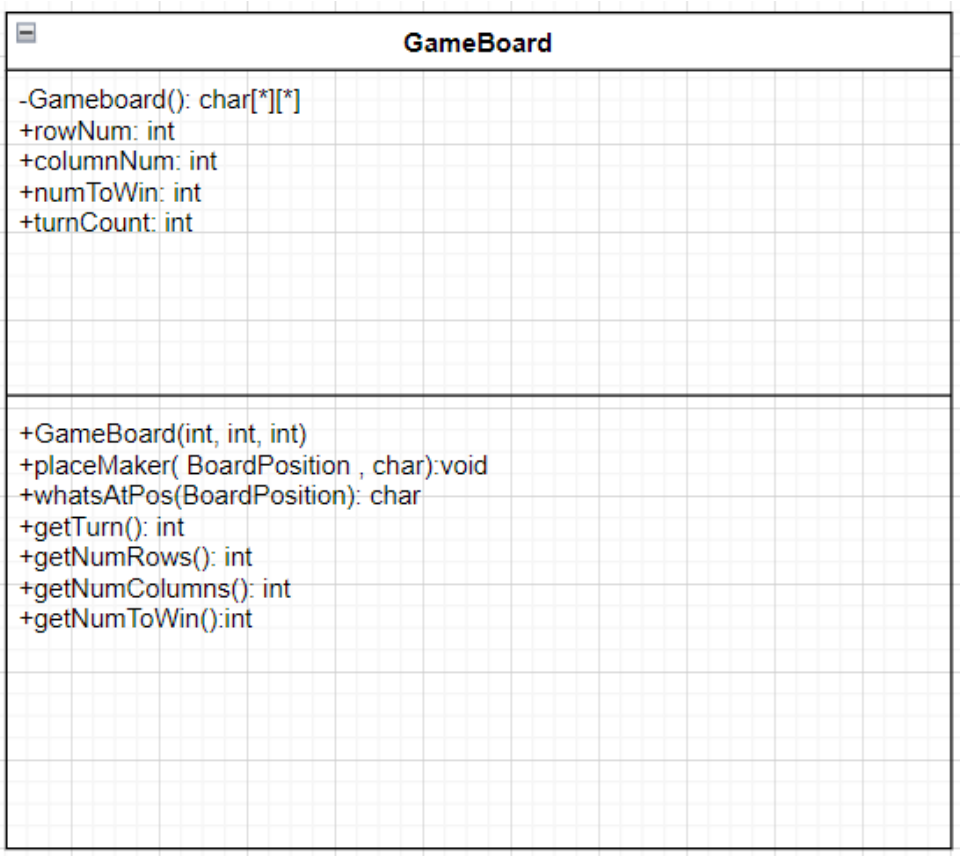
Activity Diagram:


public static variables:
int: Rows, Columns, toWin
int: turn Count
Map<Character, List<BoardPosition>>: Board
parameters (int row, int column, int toWins)
Rows = row
Columns = column
toWin = toWins
Board = new HashMap()

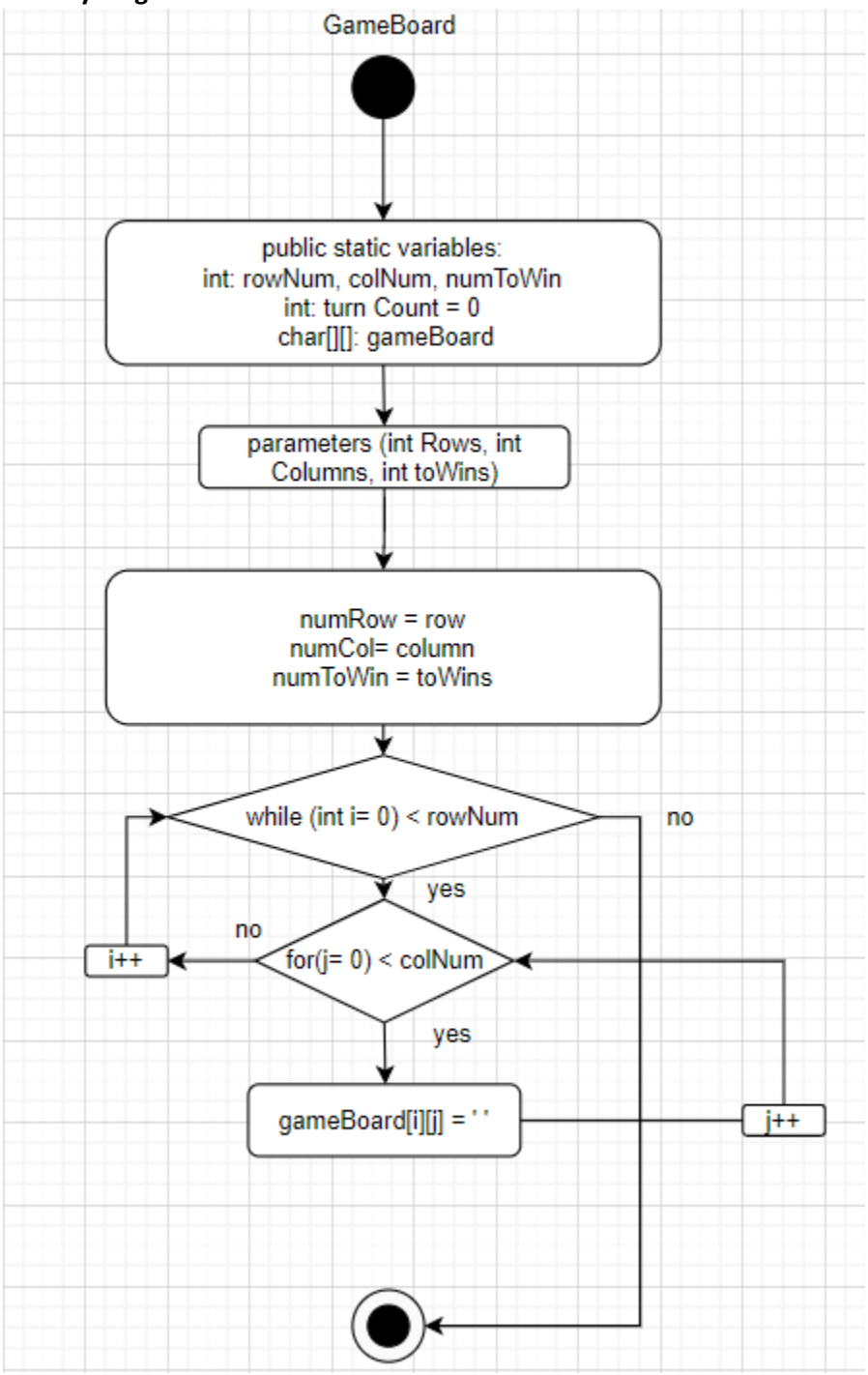

isPlayerAtPos
Pos
CheckPos
no
return null
yes
WhatsAtPos(pos) = player
no
return false
yes
return true

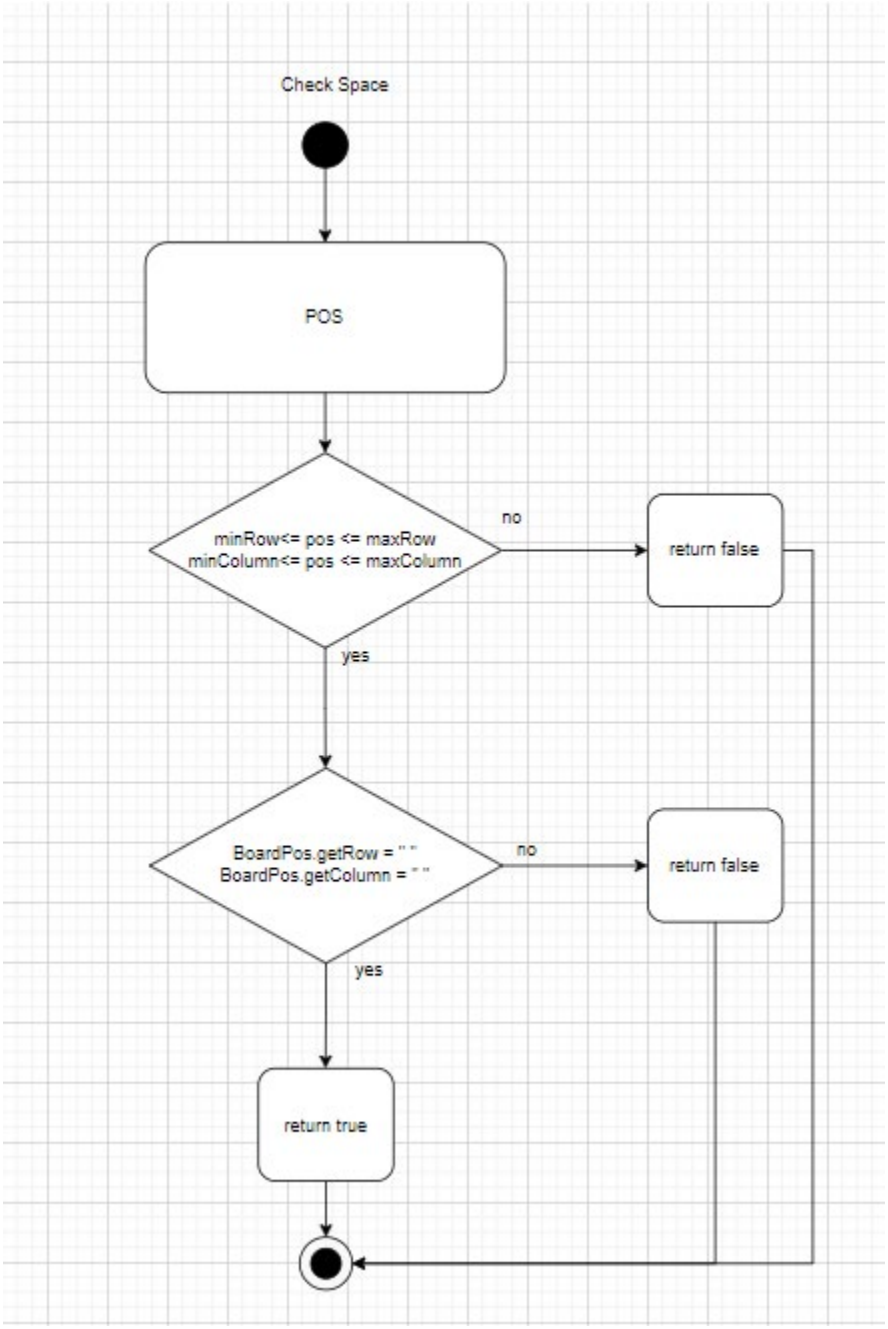
Gameboard:

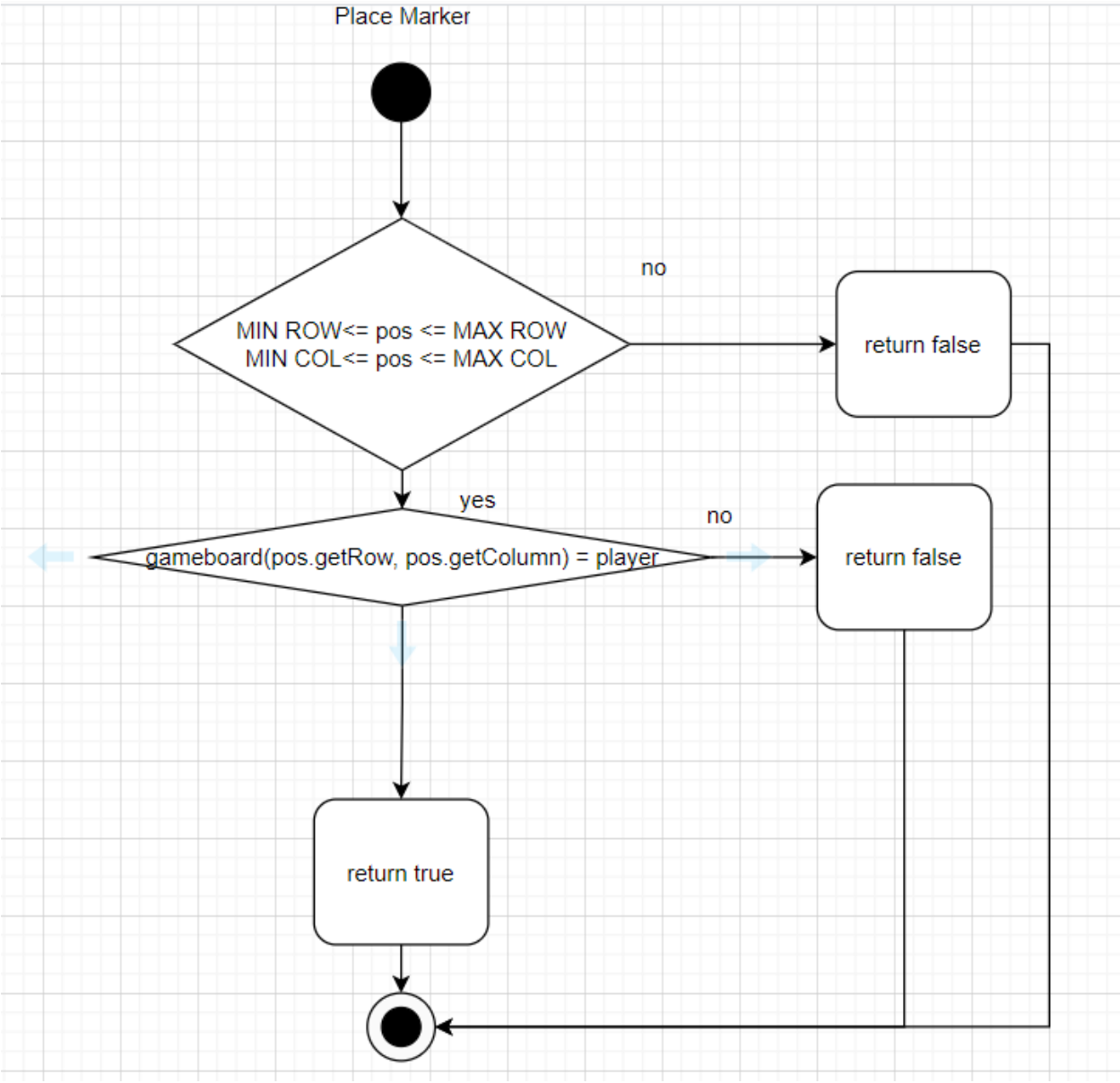
Class diagram

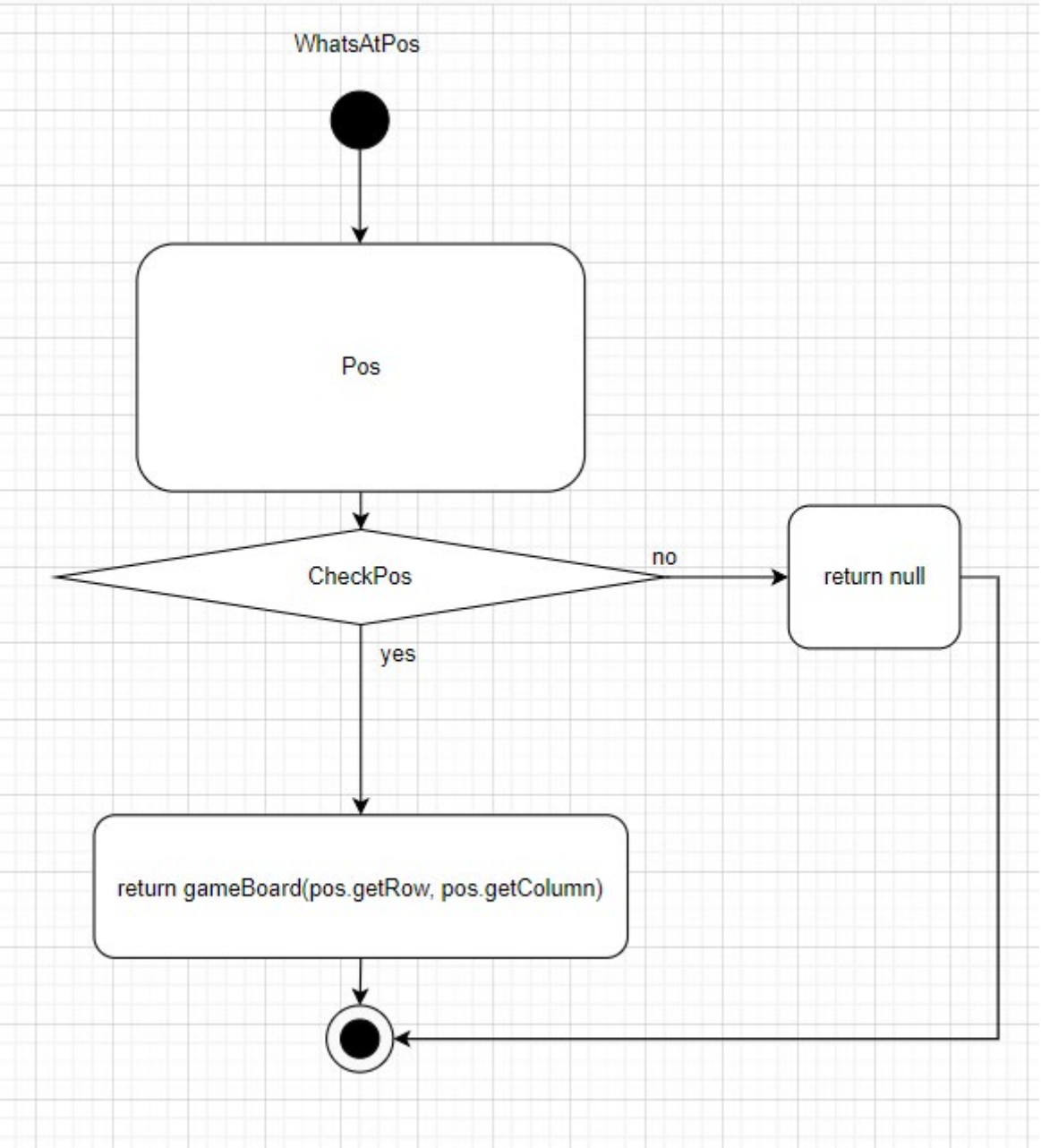


Activity diagrams



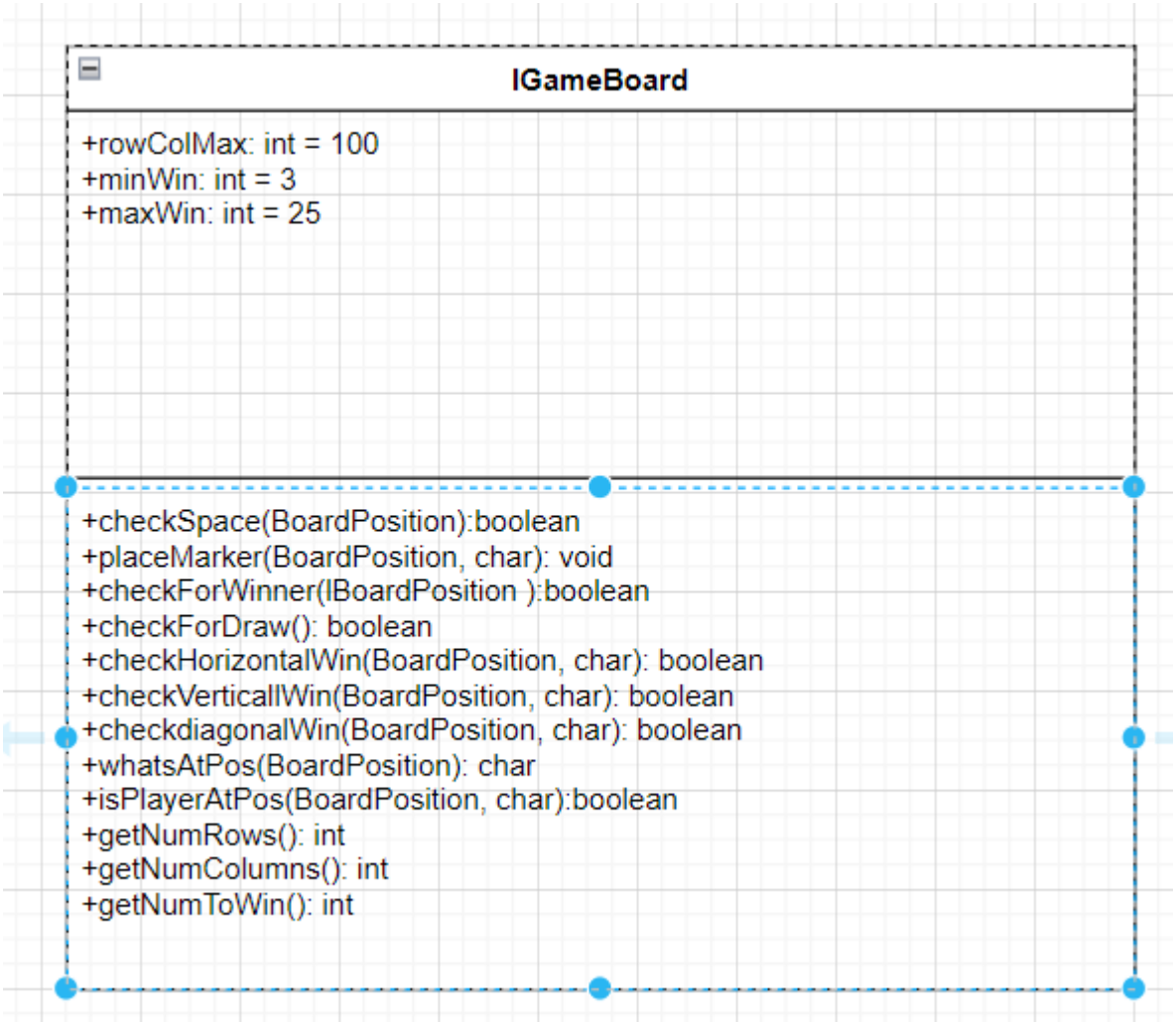

Check Space
POS
minRow<= pos <= maxRow
minColumn<= pos <= maxColumn
no
return false
yes
BoardPos.getRow = " "
BoardPos.getColumn = " "
no
return false
yes
return true


Place Marker
MIN ROW<= pos <= MAX ROW
MIN COL<= pos <= MAX COL
no
return false
yes
gameboard(pos.getRow, pos.getColumn) = player
no
return false
return true

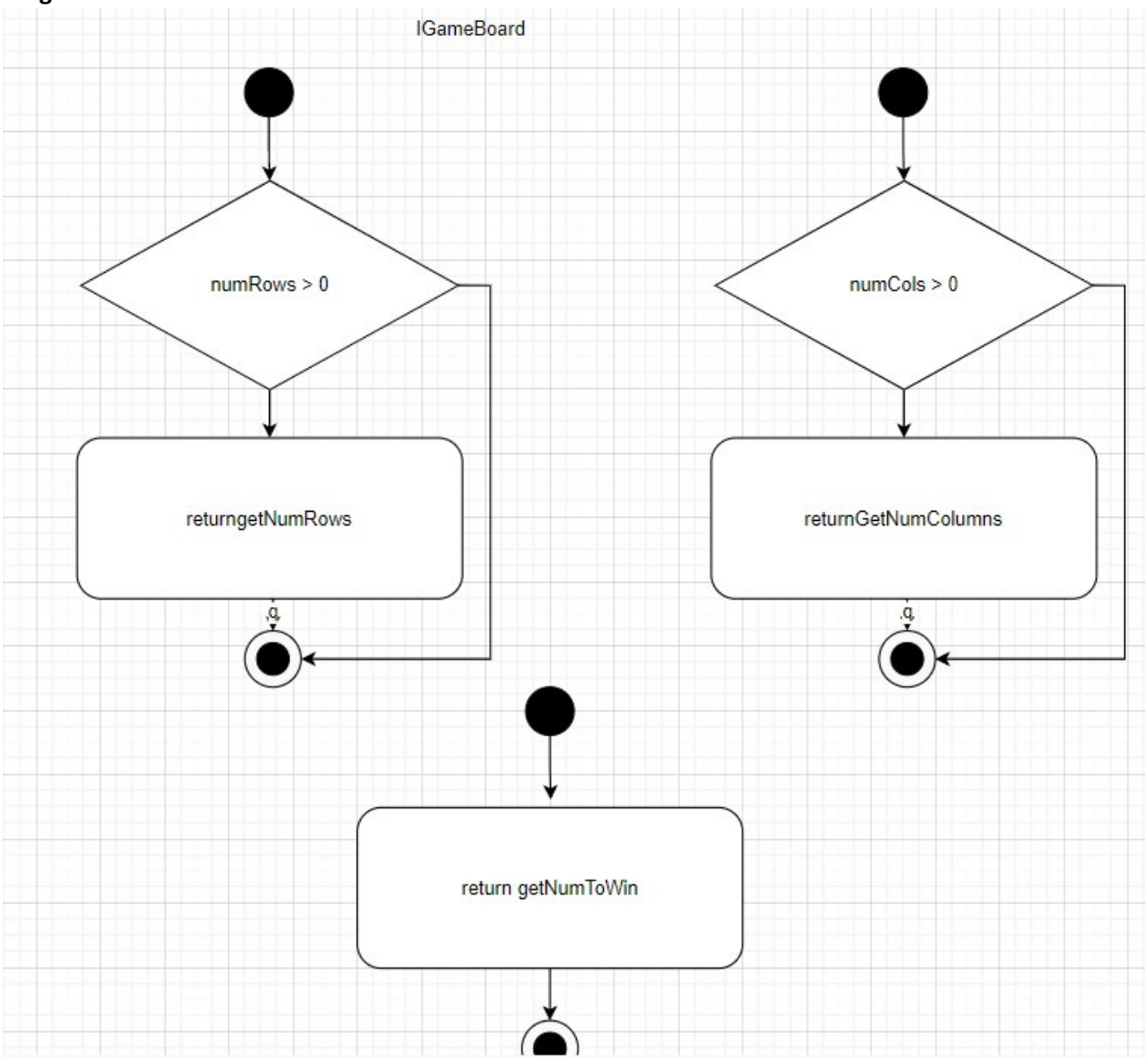

WhatsAtPos
Pos
CheckPos
no
return null
yes
return gameBoard(pos.getRow, pos.getColumn)

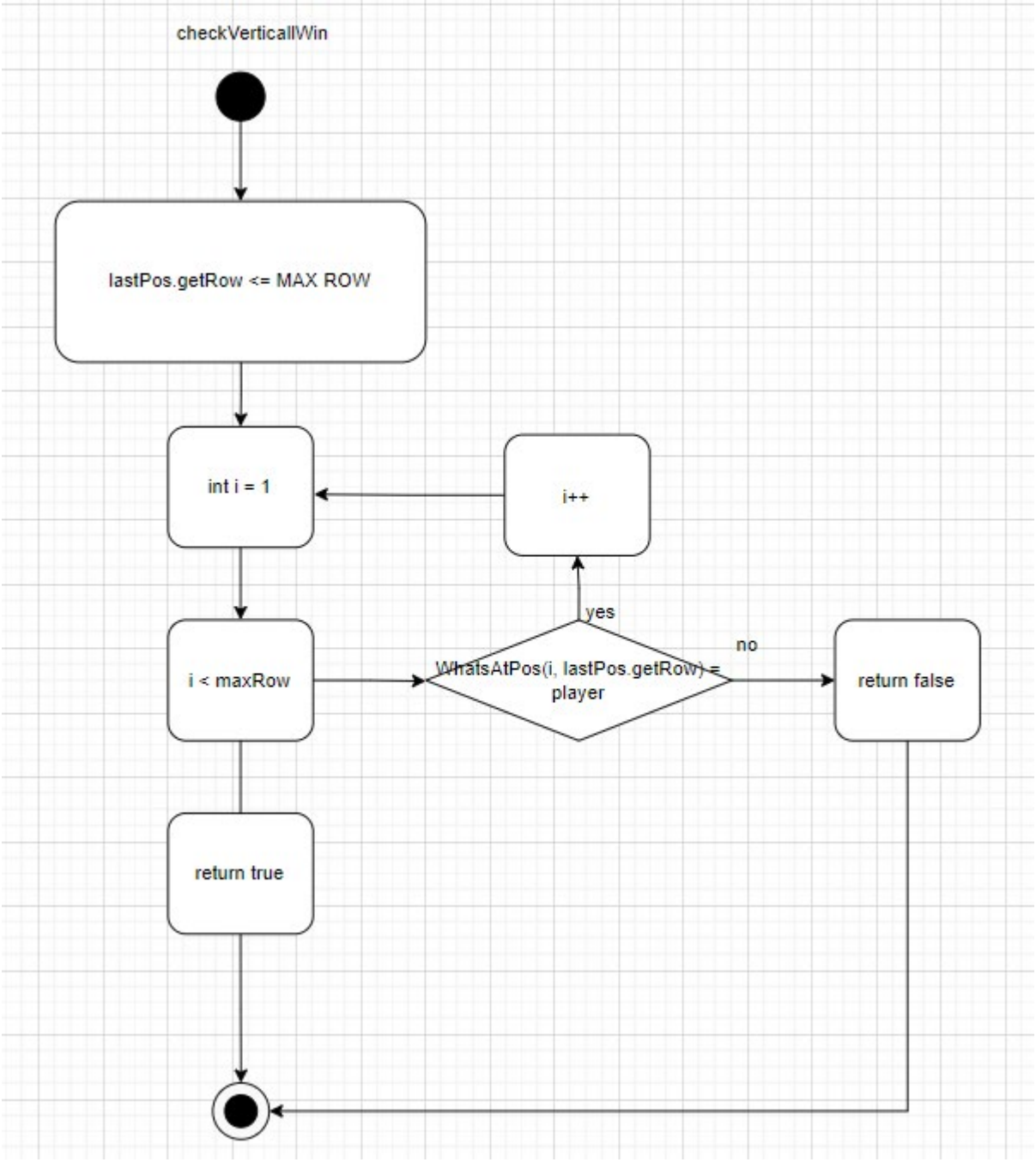
IGameboard:

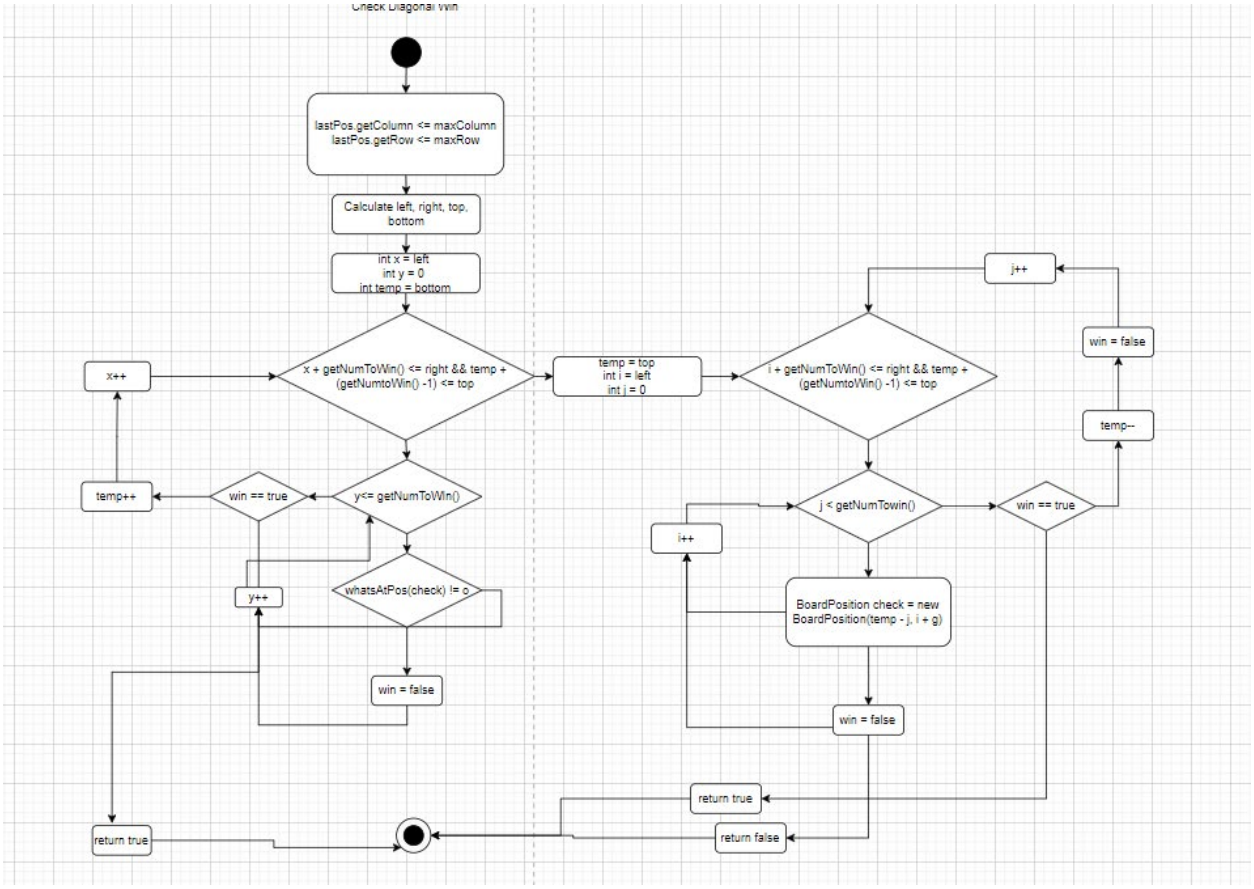
Class diagram

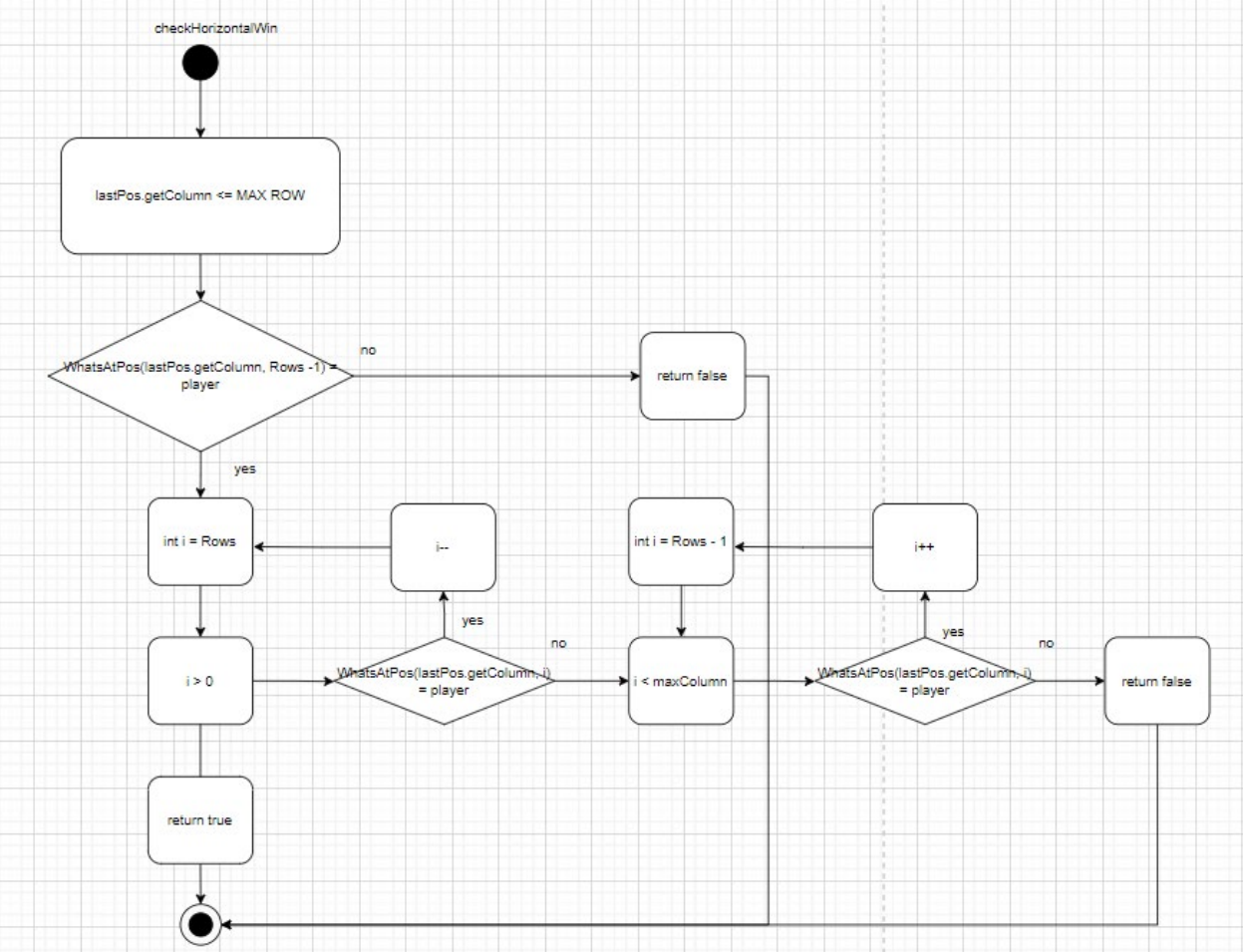


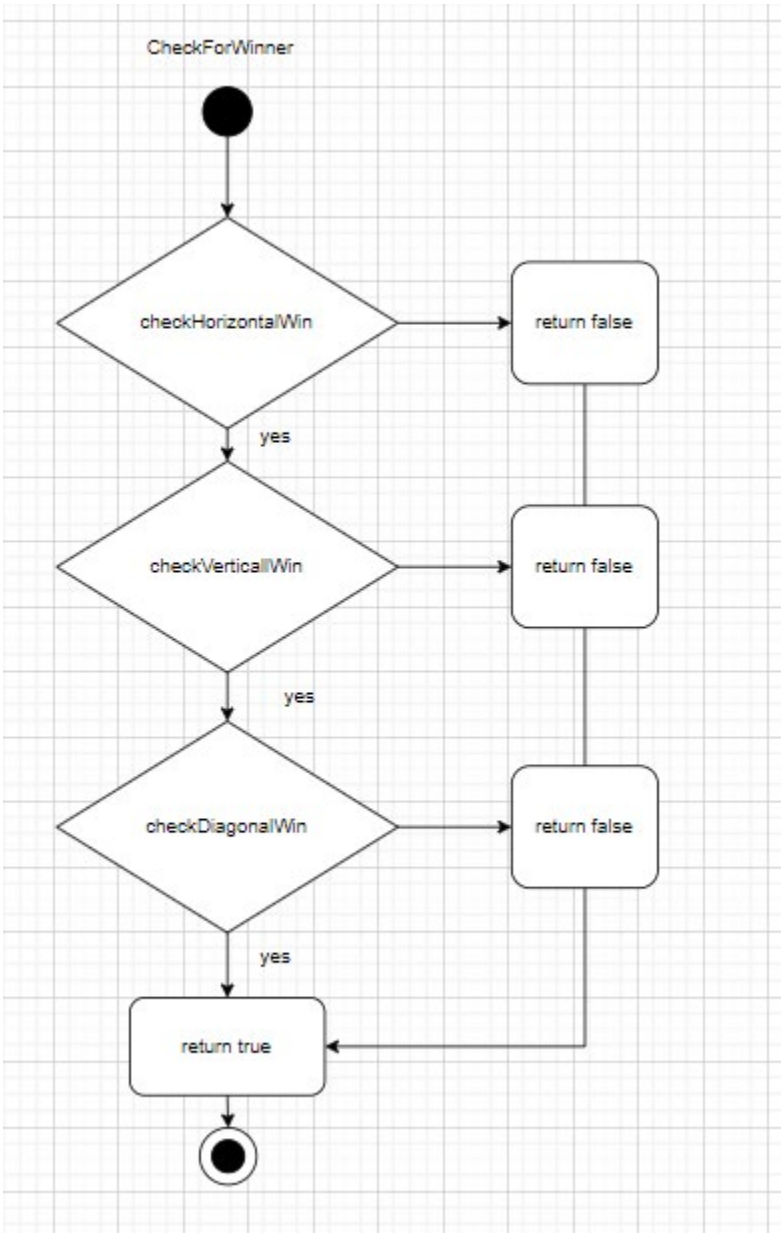
Diagrams:

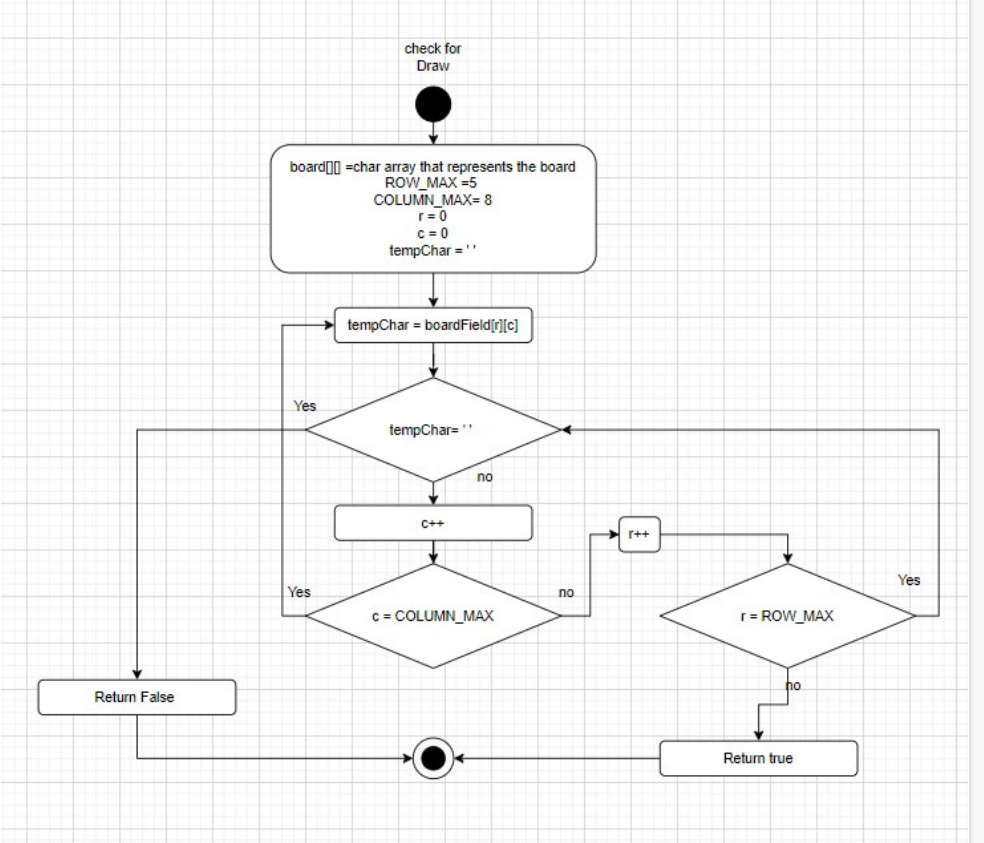

IGameBoard
numRows > 0
returngetNumRows
numCols > 0
returnGetNumColumns
return getNumToWin


checkVerticallWin
lastPos.getRow <= MAX ROW
int i = 1
i++
i < maxRow
yes
WhatsAtPos(i, lastPos.getRow) = player
no
return false
return true


Check Diagonal Win
lastPos.getColumn <= maxColumn
lastPos.getRow <= maxRow
Calculate left, right, top, bottom
int x = left
int y = 0
int temp = bottom
x++
x + getNumToWin() <= right && temp + (getNumtoWin() -1) <= top
temp = top
int i = left
int j = 0
i + getNumToWin() <= right && temp + (getNumtoWin() -1) <= top
j++
win = false
temp--
temp++
win == true
y<= getNumToWin()
j < getNumTowin()
win == true
i++
y++
whatsAtPos(check) != o
BoardPosition check = new BoardPosition(temp - j, i + g)
win = false
win = false
return true
return true
return false

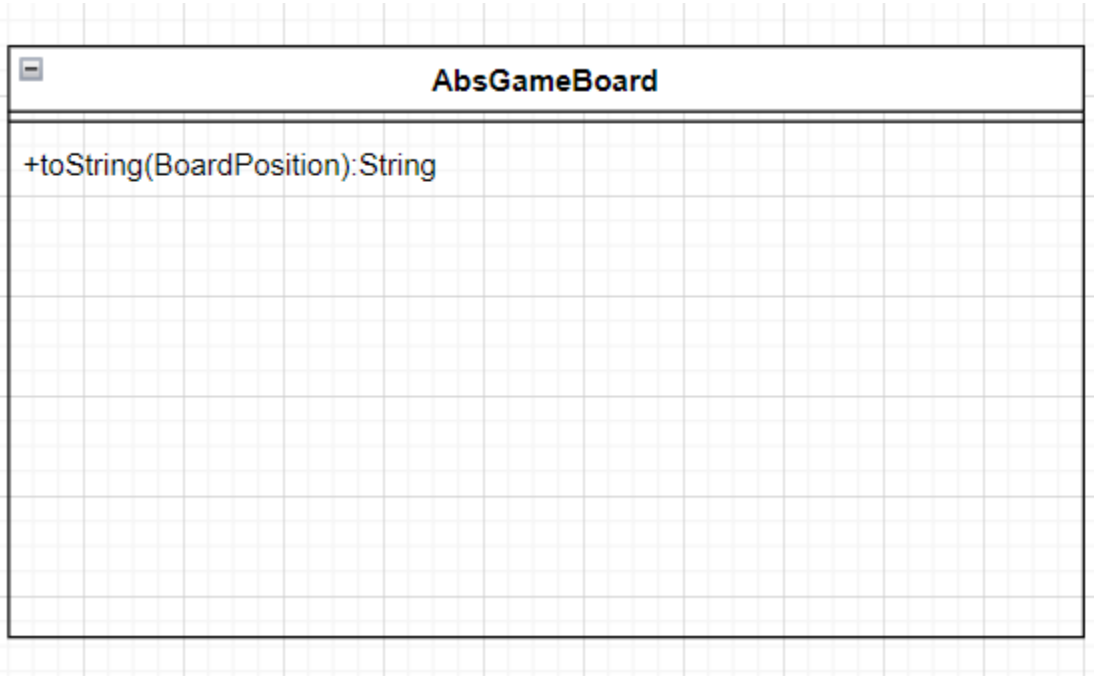

checkHorizontalWin
lastPos.getColumn <= MAX ROW
WhatsAtPos(lastPos.getColumn, Rows -1) = player
no
return false
yes
int i = Rows
i--
int i = Rows - 1
i++
i > 0
yes
WhatsAtPos(lastPos.getColumn, i) = player
no
i < maxColumn
yes
WhatsAtPos(lastPos.getColumn, i) = player
no
return false
return true


CheckForWinner
checkHorizontalWin
return false
yes
checkVerticallWin
return false
yes
checkDiagonalWin
return false
yes
return true

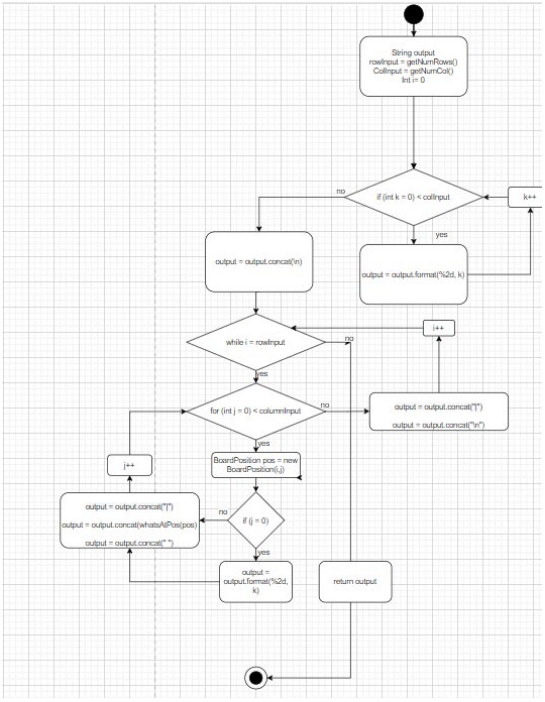

check for
Draw
board[][] =char array that represents the board
ROW_MAX =5
COLUMN_MAX= 8
r = 0
c = 0
tempChar = ' '
tempChar = boardField[r][c]
Yes
tempChar= ' '
no
c++
r++
Yes
c = COLUMN_MAX
no
Yes
r = ROW_MAX
no
Return False
Return true

AbsGameBoard:

Class diagram



Activity Diagram:



Test Cases
Details in Project 4.
TestGameBoardMem

Test the following methods:

1. public GameBoard(int Rows,int Columns, int toWins)

Input:

Rows = 3

Columns = 3

toWins= 3

Output:

	0	1	2
0			
1			
2			

Reason:

This test case is unique and distinct because it creates a board that is 3X3 which is the minimum required

Function Name:

testConstMin

Input:

Rows = 1

Columns = 102

toWins= 26

Output:

	0	1	2	3	4
0					
1					
2					
3					
4					

Reason:

This test case is unique and distinct because it checks to see if the loop will re prompt you if you input a value that is out of bounds

Function Name:

testConstDifRowCol

Input:

Rows = 11

Columns = 11

toWins= 11

Output:

	0	1	2	3	4	5	6	7	8	9	10
0											
1											
2											

Reason:

This test case is unique and distinct because it checks to see if the formatting is still correct even the size of the board include multi digit numbers

Function Name:

3												testConstMultiDig
4												
5												
6												
7												
8												
9												
10												

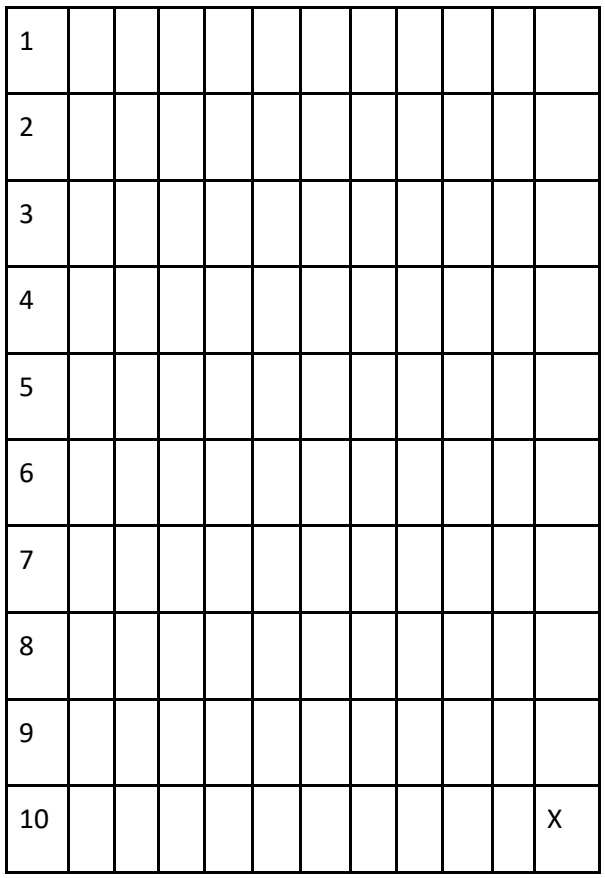
2. public boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td>Input:
State: (number to win = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>pos.getRow = 1
pos.getCol = 1</td>
<td>Output:
checkSpace = true

state of the board is unchanged</td>
<td>Reason:
This check space Test is unique because it checks a space for player one on the smallest board
Function Name:
testCheckSpaceIsThere</td>
</tr>
<tr>
<td>Input:
State: (number to win = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>pos.getRow = 0
pos.getCol = 0</td>
<td>Output:
checkSpace = true

state of the board is unchanged</td>
<td>Reason:
This check space Test is unique because it checks a space with a character other than player one
Function Name:
testCheckSpaceDifferentPlayerType</td>
</tr>
<tr>
<td>Input:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Output:
checkSpace = true

state of the board is unchanged</td>
<td>Reason:
This check space Test is unique because it checks a space on a board that is a bigger with double digits.
Function Name:</td>
</tr>
</table>

 pos.getRow = 10 pos.getCol = 10		testCheckSpaceDifferentSizeBoard

3. public boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input:	**Output:**	**Reason:**
State: (numtoWin = 3)	checkHorizontalWin = true state of the board is unchanged	This test case is unique and distinct because it checks for a win when there is a valid horizontal streak of 3 on the small board.

<table>
<tr>
<td>
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td>O</td></tr>
</table>
lastPos.getRow = 0
lastPos.getCol = 0
player = ‘X’
</td>
<td></td>
<td>Function Name:
testCheckHorizontalWin_smallBoard</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td>O</td><td>O</td><td>O</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow = 3
lastPos.getCol = 2
player = ‘X’
</td>
<td>Output: checkHorizontalWin = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a horizontal win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides

Function Name:
testCheckHorizontalWin_bigBoard_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 3)</td>
<td>Output: checkHorizontalWin = false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board when there is no valid winning streaks</td>
</tr>
</table>

<table>
<tr>
<td>
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>O</td></tr>
</table>
lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’
</td>
<td></td>
<td>Function Name:
testCheckHorizontalWin_
notWin</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>O</td><td>O</td><td>O</td><td>O</td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td>X</td><td>X</td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow = 0
lastPos.getCol = 0
player = ‘O’
</td>
<td>Output: checkHorizontalWin = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks that a different player than player 1 has won the game

Function Name:
testCheckHorizontalWin_
different_player_win</td>
</tr>
</table>

4. public boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>X</td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: checkVerticalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a win when there is a valid vertical streak of 3 on the small board.

Function Name:
testCheckVerticalWin_
win_smallBoard_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td>X</td></tr><tr><td>2</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>3</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 3
lastPos.getCol = 4
player = ‘X’</td>
<td>Output:
checkVerticalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a vertical win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides

Function Name:
testCheckVerticalWin_bigBoard_win</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>X</td><td></td></tr><tr><td>2</td><td></td><td>O</td><td>O</td></tr></table>lastPos.getRow = 0
lastPos.getCol = 0
player = ‘X’</td>
<td>Output: checkVerticalWin= false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board for a vertical win streak when there are no valid winning streaks on the board

Function Name:
testCheckVerticalWin_
no_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>1</td><td></td><td></td><td>O</td><td>X</td><td>X</td></tr><tr><td>2</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>3</td><td></td><td></td><td>O</td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 2
lastPos.getCol = 2
player = ‘O’</td>
<td>Output: checkVerticalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because because it checks that a different player than player 1 has won the game in a vertical streak

Function Name:
testCheckVerticalWin_
different_player_win</td>
</tr>
</table>

5. public boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td>X</td></tr></table>lastPos.getRow = 2
lastPos.getCol = 2
player = ‘X’</td>
<td>Output: checkDiagonalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a win when there is a valid diagonal streak of 3 on the small board going from top left corner to bottom right corner.

Function Name:
testCheckDiagonalWin_
win_smallBoard_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td>X</td></tr><tr><td>1</td><td></td><td></td><td></td><td>X</td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td>X</td><td>O</td><td></td></tr><tr><td>3</td><td></td><td>X</td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 3
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: checkDiagonalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a diagonal win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides going from top right corner to bottom left corner.

Function Name:
testCheckDiagonalWin_bigBoard_win</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td>X</td></tr><tr><td>1</td><td></td><td>X</td><td>O</td></tr><tr><td>2</td><td></td><td></td><td>O</td></tr></table>lastPos.getRow = 0
lastPos.getCol = 2
player = ‘X’</td>
<td>Output: checkDiagonalWin= false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct it checks the board for a diagonal win streak when there are no valid winning streaks on the board

Function Name:
testCheckDiagonalWin_
no_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td>X</td><td>O</td><td></td><td></td><td></td></tr><tr><td>4</td><td>X</td><td>X</td><td>O</td><td>X</td><td></td></tr></table>lastPos.getRow = 4
lastPos.getCol = 2
player = ‘O’</td>
<td>Output: checkDiagonalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a diagonal win for a different player than player one and also on a big board with a smaller win streak

Function Name:
testCheckDiagonalWin_different_player_win</td>
</tr>
<tr>
<td>Input:</td>
<td>Output: checkDiagonalWin= true</td>
<td>Reason:</td>
</tr>
</table>

<table>
<tr>
<td>State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td>X</td><td></td></tr><tr><td>1</td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td>2</td><td>O</td><td>X</td><td></td><td></td><td></td></tr><tr><td>3</td><td>X</td><td>O</td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td>O</td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 3
player = ‘X’</td>
<td>state of the board is unchanged</td>
<td>This test case is unique and distinct because it checks for a diagonal win for a different player than player one and also on a big board with a smaller win streak
Function Name:
testCheckDiagonalWin_right_up_4_win</td>
</tr>
<tr>
<td>Input:
State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td>O</td><td></td><td></td></tr><tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 1</td>
<td>Output: checkDiagonalWin= false
state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board for a diagonal win when there is a different kind of win on the board
Function Name:
testCheckDiagonalWin_different_kind_of_win</td>
</tr>
</table>

<table>
<tr><td>player = ‘X’</td><td></td><td></td></tr>
<tr><td>Input:

State: (numtoWin = 5)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td>O</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td>X</td><td>O</td><td></td><td></td></tr><tr><td>3</td><td>X</td><td>X</td><td>X</td><td>O</td><td></td></tr><tr><td>4</td><td></td><td>X</td><td></td><td></td><td>O</td></tr></table>lastPos.getRow = 4
lastPos.getCol = 4
player = ‘X’</td><td>Output: checkDiagonalWin= true

state of the board is unchanged</td><td>Reason:
This test case is unique and distinct because it checks a win from the top left board all the way down to the bottom left. Also checks this for another player.
Function Name:
testCheckDiagonalWin_left_down_full_board_win</td></tr>
</table>

6. public boolean checkForDraw()

<table>
<tr><td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td>O</td><td>X</td></tr><tr><td>1</td><td>X</td><td>X</td><td>O</td></tr><tr><td>2</td><td>O</td><td>X</td><td>O</td></tr></table></td><td>Output: checkForDraw = true

state of the board is unchanged</td><td>Reason:
This test case is unique and distinct because it checks the board for when there is a proper draw on a small board

Function Name:
testCheckForDraw_smallBoard_draw</td></tr>
</table>

<table>
<tr>
<td>lastPos.getRow = 2
lastPos.getCol = 2
player = ‘X’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 5)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>O</td><td>X</td><td>X</td><td>O</td><td>O</td></tr><tr><td>1</td><td>X</td><td>X</td><td>O</td><td>X</td><td>O</td></tr><tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td><td>O</td></tr><tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td><td>X</td></tr><tr><td>4</td><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td></tr></table>lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: checkForDraw = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board for when there is a proper draw on a big board

Function Name:
testCheckForDraw_bigBoard</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td>O</td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output: checkForDraw = false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a draw when there is not a valid draw on the board

Function Name:
testCheckForDraw
no_draw</td>
</tr>
</table>

<table>
<tr>
<td>lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: checkForDraw = false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a draw when the board is completely empty

Function Name:
testCheckForDraw_EmptyBoard</td>
</tr>
</table>

7. whatsAtPos - Create 5 distinct test cases

<table>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Output: returns “X”
state of the board remains unchanged</td>
<td>this test case is unique because it checks to see if there's a player at 0,0

Function Name:
test_whatsAtPos_smallBoard</td>
</tr>
</table>

<table><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table> lastPos.getRow = 0 lastPos.getCol = 1 player = 'X'		
Input: State: (numtoWin = 4) <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td>X</td></tr></table>lastPos.getRow = 4 lastPos.getCol = 4 player = 'X'	**Output**: returns 'X' state of the board is unchanged	this test case is unique because it checks for a player on a bigger board outside the bounds of a board minimum Function Name: test_whatsAtPos_bigBoard
Input: State: (numtoWin = 4) <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	**Output**: returns " " state of the board is unchanged	this test case is unique because it checks an empty space test_whatsAtPos_emptySpace

<table><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0 lastPos.getCol = 1 player = ‘ ’		
Input: State: (numtoWin = 4) <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td>O</td><td></td><td></td><td></td></tr><tr><td>1</td><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0 lastPos.getCol = 1 player = ‘O’	**Output**: returns O state of the board is unchanged	this test case is unique because it checks for a board position that contains a player other than player one test_whatsAtPos_differentPlayer
Input:	**Output**: null	this test case is unique because it checks to make sure that the spots outside the bounds of the board are empty

<table>
<tr>
<td>State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’</td>
<td>state of the board is unchanged</td>
<td>test_whatsAtPos_outside_Bounds</td>
</tr>
</table>

8. public boolean isPlayerAtPos(BoardPosition pos, char player)

Input:	**Output**: isPlayerAtPos = true	**Reason:** This test case is unique and distinct

<table>
<tr>
<td>State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>X</td><td></td><td></td></tr></table>lastPos.getRow = 1
lastPos.getCol = 0
player = ‘X’</td>
<td>state of the board is unchanged</td>
<td>because it checks to see if a player at a position when it is

Function Name:
testIsPlayerAtPos_smallBoard</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>O</td><td></td><td>X</td><td></td><td></td></tr><tr><td>2</td><td></td><td>O</td><td>X</td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td>O</td><td></td></tr></table>lastPos.getRow = 4
lastPos.getCol = 3
player = ‘X’</td>
<td>Output: isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks a different position and different board size to see if a player is there

Function Name:
testIsPlayerAtPos_bigBoard</td>
</tr>
<tr>
<td>Input:</td>
<td>Output: isPlayerAtPos = false</td>
<td>Reason:
This test case is unique and distinct</td>
</tr>
</table>

<table>
<tr>
<td>State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td>X</td><td>X</td></tr><tr><td>1</td><td></td><td></td><td>X</td></tr><tr><td>2</td><td>O</td><td></td><td>O</td></tr></table>lastPos.getRow = 2
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: isPlayerAtPos = false

state of the board is unchanged</td>
<td>because it checks to see if a player is at a position when it isn’t

Function Name:
testIsPlayerAtPos_no_player</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>O</td><td></td><td></td><td></td><td>X</td></tr><tr><td>1</td><td>O</td><td></td><td>X</td><td>X</td><td>X</td></tr><tr><td>2</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 0
player = ‘X’</td>
<td>Output: isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board for a positon when the player is a player other than player 1

Function Name:
testIsPlayerAtPos_different_player</td>
</tr>
<tr>
<td>Input:</td>
<td>Output: isPlayerAtPos = false</td>
<td>This test case is unique and distinct because it checks to see if a player is at a position when a different player is</td>
</tr>
</table>

	0	1	2
0			
1			
2			

lastPos.getRow = 2

lastPos.getCol = 1

player = ‘X’

state of the board is unchanged

already there

testIsPlayerAtPos_different_player_is_in_Position

9. public void placeMarker(BoardPosition marker, char player)

Input:

State: (numtoWin = 3)

	0	1	2
0	X		
1	O		O
2	X		

marker.getRow = 2

marker.getCol = 2

player = ‘X’

	0	1	2
0	X		
1	O	X	O
2	X		

Reason:

This test case is unique and distinct because is places a marker on a small board

Function Name:

testPlaceMarker_smallBoard

Input:

State: (numtoWin = 4)

	0	1	2	3	4
0			X		

Reason:

This test case is unique and distinct because is places a marker on a bigger board

	0	1	2	3	4
0			X		
1	O		X		
2		O	X		
3			X		
4				O	

marker.getRow = 0

marker.getCol = 1

player = 'X'

1	O		X		
2		O	X		
3			X		
4				O	

Function Name:

test_placeMarker_ZeroZero

Input:

State: (numtoWin = 3)

	0	1	2
0	O		X
1			
2	X		O

marker.getRow = 4

marker.getCol = 4

player = 'X'

	0	1	2
0	O		X
1			
2	X		O

Reason:

This test case is unique and distinct because it tries to place a marker on a position that is not on the board

Function Name:

test_placeMarker_bigBoard

Input:

State: (numtoWin = 4)

State of board:

	0	1	2	3	4

Reason:

This test case is unique and distinct because it tries to place a marker of a player other than player one

	0	1	2	3	4
0					
1	X				
2					
3					
4					

marker.getRow = 1

marker.getCol = 1

player = ‘O’

0					
1	X	O			
2					
3					
4					

Function Name:

test_placeMarker_ThreePlayers

Input:

	0	1	2
0	X		
1			
2			

marker.getRow = 0

marker.getCol = 0

player = ‘O’

	0	1	2
0	X		
1			
2			

Reason:

This test case is unique and distinct because it places a marker on a position that is already occupied by another player

Function Name:

test_placeMarker_filledBoard

Test Cases

Details in Project 4.

TestGameBoard

Test the following methods:

1. public GameBoard(int Rows,int Columns, int toWins)

<table>
<tr>
<td>Input:

Rows = 3
Columns = 3
toWins= 3</td>
<td>Output:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table></td>
<td>Reason:

This test case is unique and distinct because it creates a board that is 3X3 which is the minimum required

Function Name:

testConstMin</td>
</tr>
<tr>
<td>Input:

Rows = 1
Columns = 102
toWins= 26</td>
<td>Output:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td>Reason:

This test case is unique and distinct because it checks to see if the loop will re prompt you if you input a value that is out of bounds

Function Name:

testConstDifRowCol</td>
</tr>
<tr>
<td>Input:

Rows = 11
Columns = 11
toWins= 11</td>
<td>Output:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td>Reason:

This test case is unique and distinct because it checks to see if the formatting is still correct even the size of the board include multi digit numbers

Function Name:

testConstMultiDig</td>
</tr>
</table>

3											
4											
5											
6											
7											
8											
9											
10											

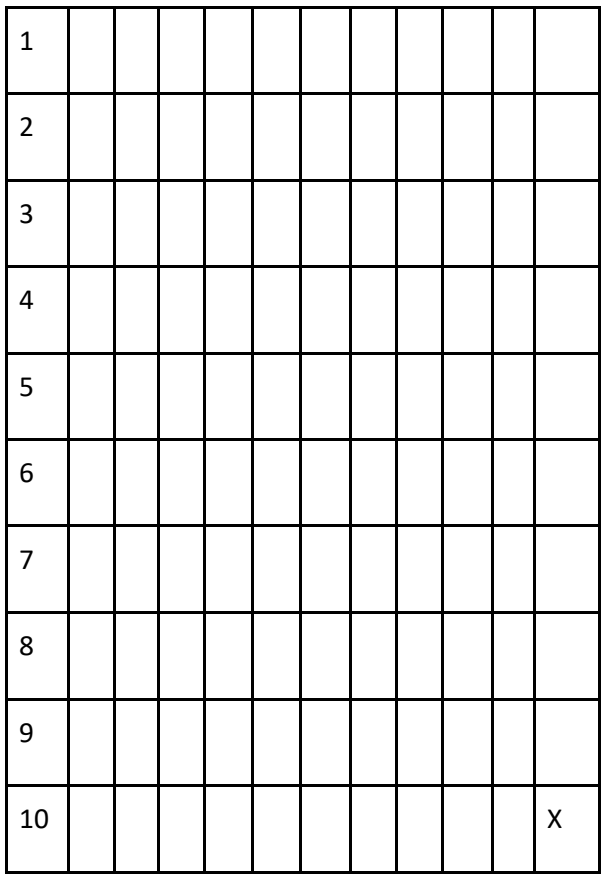
2. public boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td>Input:
State: (number to win = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>pos.getRow = 1
pos.getCol = 1</td>
<td>Output:
checkSpace = true

state of the board is unchanged</td>
<td>Reason:
This check space Test is unique because it checks a space for player one on the smallest board
Function Name:
testCheckSpaceIsThere</td>
</tr>
<tr>
<td>Input:
State: (number to win = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>pos.getRow = 0
pos.getCol = 0</td>
<td>Output:
checkSpace = true

state of the board is unchanged</td>
<td>Reason:
This check space Test is unique because it checks a space with a character other than player one
Function Name:
testCheckSpaceDifferentPlayerType</td>
</tr>
<tr>
<td>Input:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Output:
checkSpace = true

state of the board is unchanged</td>
<td>Reason:
This check space Test is unique because it checks a space on a board that is a bigger with double digits.
Function Name:</td>
</tr>
</table>

 pos.getRow = 10 pos.getCol = 10		testCheckSpaceDifferentSizeBoard

3. public boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input:	**Output:**	**Reason:**
State: (numtoWin = 3)	checkHorizontalWin = true state of the board is unchanged	This test case is unique and distinct because it checks for a win when there is a valid horizontal streak of 3 on the small board.

<table>
<tr>
<td>
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td>O</td></tr>
</table>
lastPos.getRow = 0
lastPos.getCol = 0
player = ‘X’
</td>
<td></td>
<td>Function Name:
testCheckHorizontalWin_smallBoard</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td>O</td><td>O</td><td>O</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow = 3
lastPos.getCol = 2
player = ‘X’
</td>
<td>Output: checkHorizontalWin = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a horizontal win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides

Function Name:
testCheckHorizontalWin_bigBoard_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 3)</td>
<td>Output: checkHorizontalWin = false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board when there is no valid winning streaks</td>
</tr>
</table>

<table>
<tr>
<td>
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>O</td></tr>
</table>
lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’
</td>
<td></td>
<td>Function Name:
testCheckHorizontalWin_
notWin</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>O</td><td>O</td><td>O</td><td>O</td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td>X</td><td>X</td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow = 0
lastPos.getCol = 0
player = ‘O’
</td>
<td>Output: checkHorizontalWin = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks that a different player than player 1 has won the game

Function Name:
testCheckHorizontalWin_
different_player_win</td>
</tr>
</table>

4. public boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>X</td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: checkVerticalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a win when there is a valid vertical streak of 3 on the small board.

Function Name:
testCheckVerticalWin_
win_smallBoard_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td>X</td></tr><tr><td>2</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>3</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 3
lastPos.getCol = 4
player = ‘X’</td>
<td>Output:
checkVerticalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a vertical win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides

Function Name:
testCheckVerticalWin_bigBoard_win</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>X</td><td></td></tr><tr><td>2</td><td></td><td>O</td><td>O</td></tr></table>lastPos.getRow = 0
lastPos.getCol = 0
player = ‘X’</td>
<td>Output: checkVerticalWin= false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board for a vertical win streak when there are no valid winning streaks on the board

Function Name:
testCheckVerticalWin_
no_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>1</td><td></td><td></td><td>O</td><td>X</td><td>X</td></tr><tr><td>2</td><td></td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>3</td><td></td><td></td><td>O</td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 2
lastPos.getCol = 2
player = ‘O’</td>
<td>Output: checkVerticalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because because it checks that a different player than player 1 has won the game in a vertical streak

Function Name:
testCheckVerticalWin_
different_player_win</td>
</tr>
</table>

5. public boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td>X</td></tr></table>lastPos.getRow = 2
lastPos.getCol = 2
player = ‘X’</td>
<td>Output: checkDiagonalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a win when there is a valid diagonal streak of 3 on the small board going from top left corner to bottom right corner.

Function Name:
testCheckDiagonalWin_
win_smallBoard_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td>X</td></tr><tr><td>1</td><td></td><td></td><td></td><td>X</td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td>X</td><td>O</td><td></td></tr><tr><td>3</td><td></td><td>X</td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 3
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: checkDiagonalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a diagonal win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides going from top right corner to bottom left corner.

Function Name:
testCheckDiagonalWin_bigBoard_win</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td>X</td></tr><tr><td>1</td><td></td><td>X</td><td>O</td></tr><tr><td>2</td><td></td><td></td><td>O</td></tr></table>lastPos.getRow = 0
lastPos.getCol = 2
player = ‘X’</td>
<td>Output: checkDiagonalWin= false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct it checks the board for a diagonal win streak when there are no valid winning streaks on the board

Function Name:
testCheckDiagonalWin_
no_win</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td>X</td><td>O</td><td></td><td></td><td></td></tr><tr><td>4</td><td>X</td><td>X</td><td>O</td><td>X</td><td></td></tr></table>lastPos.getRow = 4
lastPos.getCol = 2
player = ‘O’</td>
<td>Output: checkDiagonalWin= true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a diagonal win for a different player than player one and also on a big board with a smaller win streak

Function Name:
testCheckDiagonalWin_different_player_win</td>
</tr>
<tr>
<td>Input:</td>
<td>Output: checkDiagonalWin= true</td>
<td>Reason:</td>
</tr>
</table>

<table>
<tr>
<td>State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td>X</td><td></td></tr><tr><td>1</td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td>2</td><td>O</td><td>X</td><td></td><td></td><td></td></tr><tr><td>3</td><td>X</td><td>O</td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td>O</td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 3
player = ‘X’</td>
<td>state of the board is unchanged</td>
<td>This test case is unique and distinct because it checks for a diagonal win for a different player than player one and also on a big board with a smaller win streak
Function Name:
testCheckDiagonalWin_right_up_4_win</td>
</tr>
<tr>
<td>Input:
State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td>O</td><td></td><td></td></tr><tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 1</td>
<td>Output: checkDiagonalWin= false
state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board for a diagonal win when there is a different kind of win on the board
Function Name:
testCheckDiagonalWin_different_kind_of_win</td>
</tr>
</table>

<table>
<tr><td>player = ‘X’</td><td></td><td></td></tr>
<tr><td>Input:

State: (numtoWin = 5)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td>O</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td>X</td><td>O</td><td></td><td></td></tr><tr><td>3</td><td>X</td><td>X</td><td>X</td><td>O</td><td></td></tr><tr><td>4</td><td></td><td>X</td><td></td><td></td><td>O</td></tr></table>lastPos.getRow = 4
lastPos.getCol = 4
player = ‘X’</td><td>Output: checkDiagonalWin= true

state of the board is unchanged</td><td>Reason:
This test case is unique and distinct because it checks a win from the top left board all the way down to the bottom left. Also checks this for another player.
Function Name:
testCheckDiagonalWin_left_down_full_board_win</td></tr>
</table>

6. public boolean checkForDraw()

<table>
<tr><td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td>O</td><td>X</td></tr><tr><td>1</td><td>X</td><td>X</td><td>O</td></tr><tr><td>2</td><td>O</td><td>X</td><td>O</td></tr></table></td><td>Output: checkForDraw = true

state of the board is unchanged</td><td>Reason:
This test case is unique and distinct because it checks the board for when there is a proper draw on a small board

Function Name:
testCheckForDraw_smallBoard_draw</td></tr>
</table>

<table>
<tr>
<td>lastPos.getRow = 2
lastPos.getCol = 2
player = ‘X’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 5)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>O</td><td>X</td><td>X</td><td>O</td><td>O</td></tr><tr><td>1</td><td>X</td><td>X</td><td>O</td><td>X</td><td>O</td></tr><tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td><td>O</td></tr><tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td><td>X</td></tr><tr><td>4</td><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td></tr></table>lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: checkForDraw = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board for when there is a proper draw on a big board

Function Name:
testCheckForDraw_bigBoard</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td>O</td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output: checkForDraw = false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a draw when there is not a valid draw on the board

Function Name:
testCheckForDraw
no_draw</td>
</tr>
</table>

<table>
<tr>
<td>lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: checkForDraw = false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks for a draw when the board is completely empty

Function Name:
testCheckForDraw_EmptyBoard</td>
</tr>
</table>

7. whatsAtPos - Create 5 distinct test cases

<table>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td></td>
<td>this test case is unique because it checks to see if there's a player at 0,0

Function Name:
test_whatsAtPos_smallBoard</td>
</tr>
</table>

<table><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table> lastPos.getRow = 0 lastPos.getCol = 1 player = 'X'		
Input: State: (numtoWin = 4) <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table> lastPos.getRow = 0 lastPos.getCol = 1 player = 'X'		this test case is unique because it checks for a player on a bigger board outside the bounds of a board minimum Function Name: test_whatsAtPos_bigBoard
Input: State: (numtoWin = 4) <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>		this test case is unique because it checks an empty space test_whatsAtPos_emptySpace

<table><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0 lastPos.getCol = 1 player = ‘X’		
Input: State: (numtoWin = 4) <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0 lastPos.getCol = 1 player = ‘X’		this test case is unique because it checks for a board position that contains a player other than player one test_whatsAtPos_differentPlayer
Input:		this test case is unique because it checks to make sure that the spots outside the bounds of the board are empty

State: (numtoWin = 4) lastPos.getRow = 0 lastPos.getCol = 1 player = ‘X’		test_whatsAtPos_outside_Bounds

	0	1	2	3	4
0					
1					
2					
3					
4					

8. public boolean isPlayerAtPos(BoardPosition pos, char player)

Input:	**Output**: isPlayerAtPos = true	**Reason:** This test case is unique and distinct

<table>
<tr>
<td>State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>X</td><td></td><td></td></tr></table>lastPos.getRow = 1
lastPos.getCol = 0
player = ‘X’</td>
<td>state of the board is unchanged</td>
<td>because it checks to see if a player at a position when it is

Function Name:
testIsPlayerAtPos_smallBoard</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>O</td><td></td><td>X</td><td></td><td></td></tr><tr><td>2</td><td></td><td>O</td><td>X</td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td>O</td><td></td></tr></table>lastPos.getRow = 4
lastPos.getCol = 3
player = ‘X’</td>
<td>Output: isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks a different position and different board size to see if a player is there

Function Name:
testIsPlayerAtPos_bigBoard</td>
</tr>
<tr>
<td>Input:</td>
<td>Output: isPlayerAtPos = false</td>
<td>Reason:
This test case is unique and distinct</td>
</tr>
</table>

<table>
<tr>
<td>State: (numtoWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td>X</td><td>X</td></tr><tr><td>1</td><td></td><td></td><td>X</td></tr><tr><td>2</td><td>O</td><td></td><td>O</td></tr></table>lastPos.getRow = 2
lastPos.getCol = 1
player = ‘X’</td>
<td>Output: isPlayerAtPos = false

state of the board is unchanged</td>
<td>because it checks to see if a player is at a position when it isn’t

Function Name:
testIsPlayerAtPos_no_player</td>
</tr>
<tr>
<td>Input:

State: (numtoWin = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>O</td><td></td><td></td><td></td><td>X</td></tr><tr><td>1</td><td>O</td><td></td><td>X</td><td>X</td><td>X</td></tr><tr><td>2</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow = 0
lastPos.getCol = 0
player = ‘X’</td>
<td>Output: isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks the board for a positon when the player is a player other than player 1

Function Name:
testIsPlayerAtPos_different_player</td>
</tr>
<tr>
<td>Input:</td>
<td>Output: isPlayerAtPos = false</td>
<td>This test case is unique and distinct because it checks to see if a player is at a position when a different player is</td>
</tr>
</table>

	0	1	2
0			
1			
2			

lastPos.getRow = 2

lastPos.getCol = 1

player = ‘X’

state of the board is unchanged

already there

testIsPlayerAtPos_different_player_is_in_Position

9. public void placeMarker(BoardPosition marker, char player)

Input:

State: (numtoWin = 3)

	0	1	2
0	X		
1	O		O
2	X		

marker.getRow = 2

marker.getCol = 2

player = ‘X’

	0	1	2
0	X		
1	O	X	O
2	X		

Reason:

This test case is unique and distinct because is places a marker on a small board

Function Name:

testPlaceMarker_smallBoard

Input:

State: (numtoWin = 4)

	0	1	2	3	4
0			X		

Reason:

This test case is unique and distinct because is places a marker on a bigger board

	0	1	2	3	4
0			X		
1	O		X		
2		O	X		
3			X		
4				O	

marker.getRow = 0

marker.getCol = 1

player = 'X'

1	O		X		
2		O	X		
3			X		
4				O	

Function Name:

test_placeMarker_ZeroZero

Input:

State: (numtoWin = 3)

	0	1	2
0	O		X
1			
2	X		O

marker.getRow = 4

marker.getCol = 4

player = 'X'

	0	1	2
0	O		X
1			
2	X		O

Reason:

This test case is unique and distinct because it tries to place a marker on a position that is not on the board

Function Name:

test_placeMarker_bigBoard

Input:

State: (numtoWin = 4)

State of board:

	0	1	2	3	4

Reason:

This test case is unique and distinct because it tries to place a marker of a player other than player one

	0	1	2	3	4
0					
1	X				
2					
3					
4					

marker.getRow = 1

marker.getCol = 1

player = ‘O’

0					
1	X	O			
2					
3					
4					

Function Name:

test_placeMarker_ThreePlayers

Input:

	0	1	2
0	X		
1			
2			

marker.getRow = 0

marker.getCol = 0

player = ‘O’

	0	1	2
0	X		
1			
2			

Reason:

This test case is unique and distinct because it places a marker on a position that is already occupied by another player

Function Name:

test_placeMarker_filledBoard