CPSC 2150 Project Report

Mark Sheldon

Requirements Analysis

Functional Requirements:

As a player, I <what/need/can> <goal> so that <reason>

1. As a player, I can know what character I am so that I know which markers are mine
2. As a player, I can view the board so that I know where markers have been played
3. As a player, I can view the number of rows on the board so that I know what indices are valid
4. As a player, I can view the number of columns on the board so that I know what indices are valid
5. As a player, I need to know if my position choice is invalid so that I can choose a new one
6. As a player, I need to know who wins so that I know who the winner is
7. As a player, I need to know when I get enough markers in a row so that I know I won the game
8. As a player, I need to know when my opponent gets enough markers in a row so that I know I lost
9. As a player, I need to know if someone wins so that I know the game is over
10. As a player, I need to know if there are no more spaces available so I know the game is a draw
11. As a player, I need to know if the game is a draw so I know the game is over
12. As a player, I need to be able to play again once the game is over so that I can play another game
13. As a player, I need to be asked if I want to stop playing so that I can choose to not player another game
14. As a player, I can see the board after my opponent plays so I can plan my next move
15. As a player I can see the board after I play so that I can make sure I played in the right spot
16. As a player, I need to know whose turn it is so that I can know if it is my turn
17. As a player, I need to be asked how many rows the board should have so I can choose the board size
18. As a player, I need to be asked how many columns the board should have so I can choose the board size
19. As a player, I need to be asked how many markers in a row are needed to win so I can decide how long it will take to win
20. As a player, I need to be asked how many players are playing so I can have everyone who wants to play, play

Non-Functional Requirements

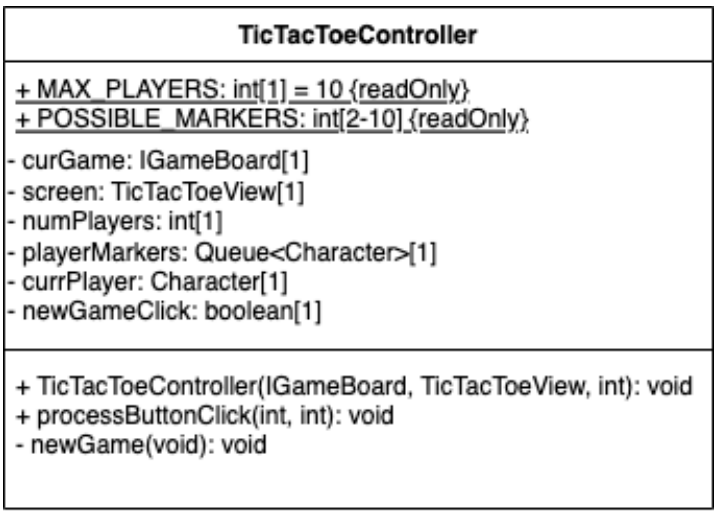
1. The game must be coded in java
2. The board can be structured as a 2D array of characters
3. The board can be structured as a map of character-BoardPosition list pairs
4. The game must have a GUI

5. The GUI must be coded in Java Swing
6. The board must have a flexible number of rows and columns
7. The board's indexing must start in the top left corner
8. The program must have a fast version and a memory efficient version
9. The program must allow for between 2 and 10 players
10. The program must check for a win after each turn
11. The program must check for a draw after each turn
12. The program must cycle until the player closes the window
13. The program must allow a user to change settings between games
14. There should only be one main method
15. The main method should be inside of the GameScreen class
16. Input for a marker's indices must be read row first, then column
17. Input must only be read through the GameScreen class
18. Output must only be sent through the GameScreen class

System Design

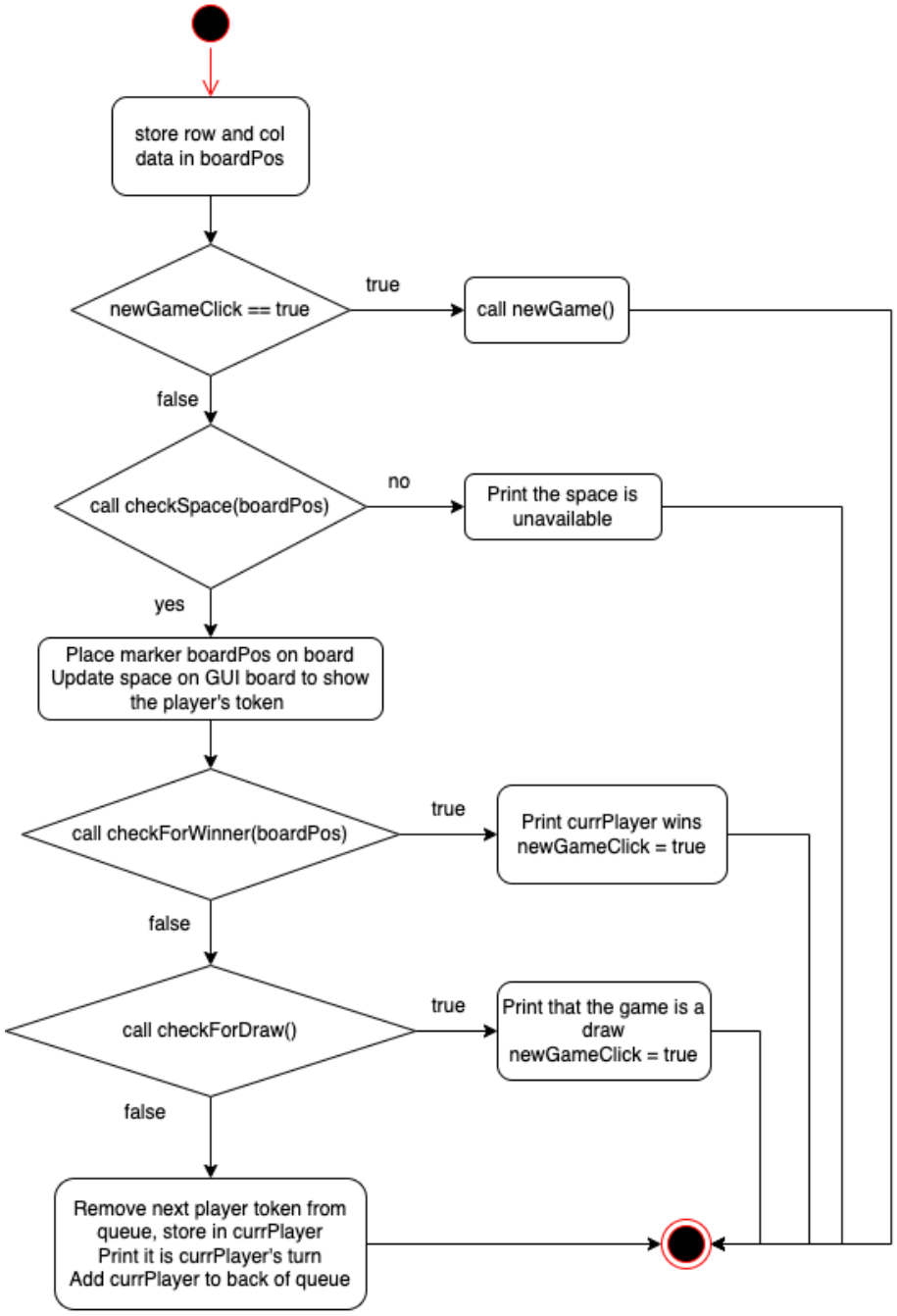
Class 1: TicTacToeController

Class Diagram:



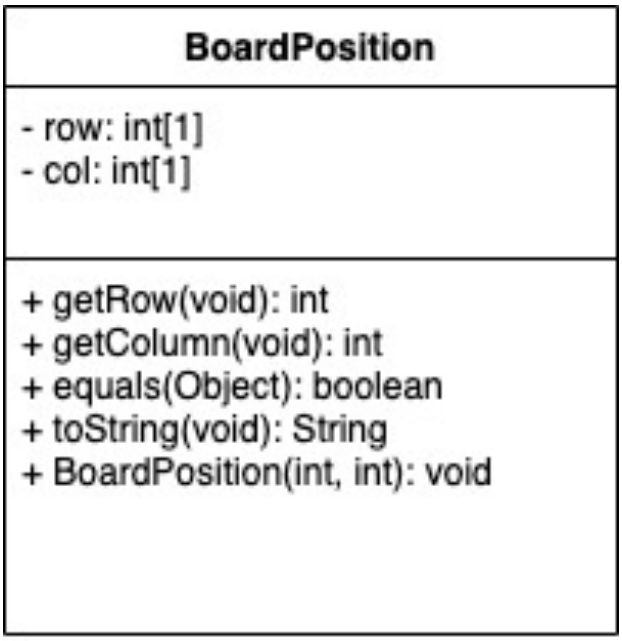
Activity Diagrams:

processButtonClick



Class 2: BoardPosition.java

Class diagram

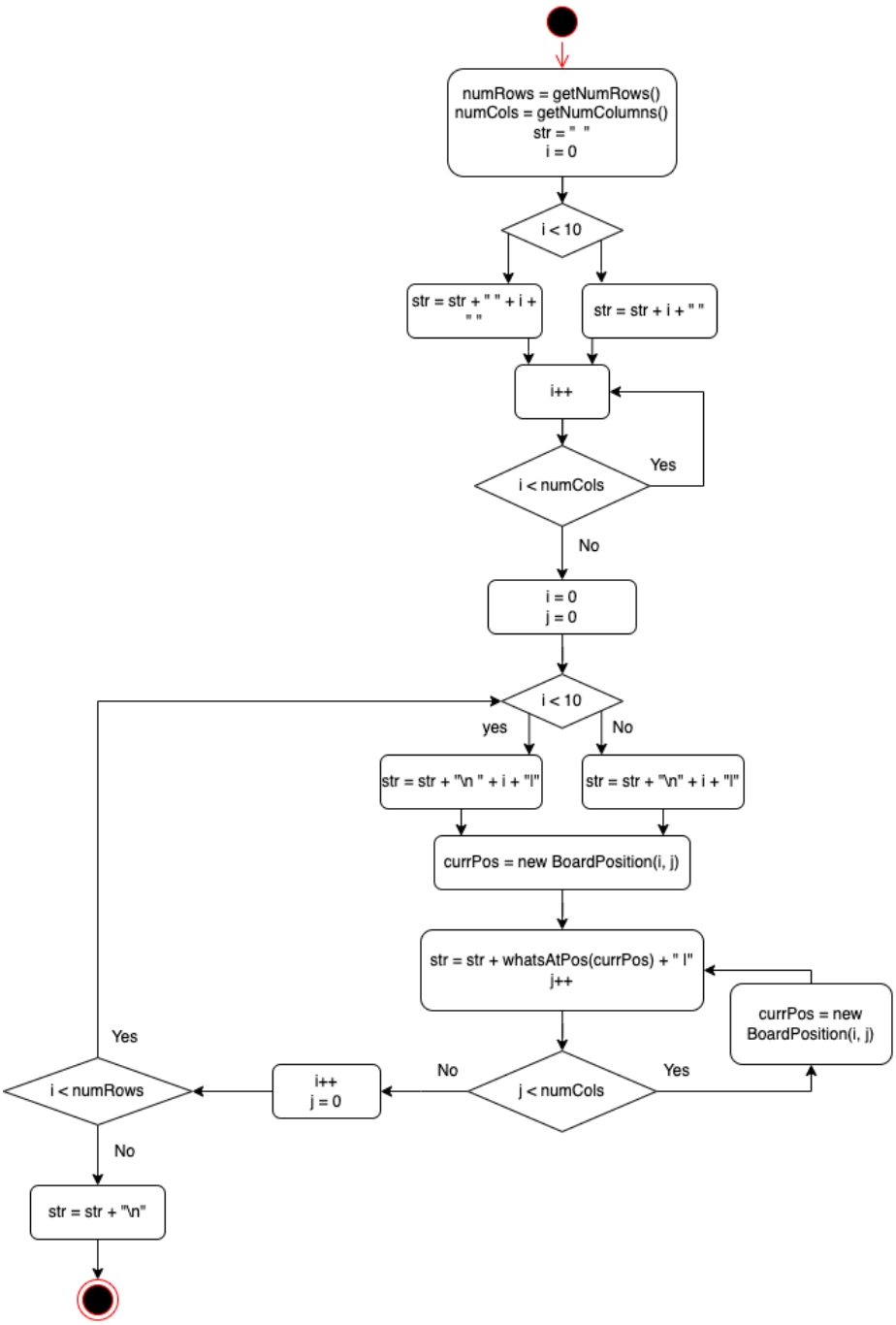


Class 3: AbsGameBoard.java

Class Diagram:

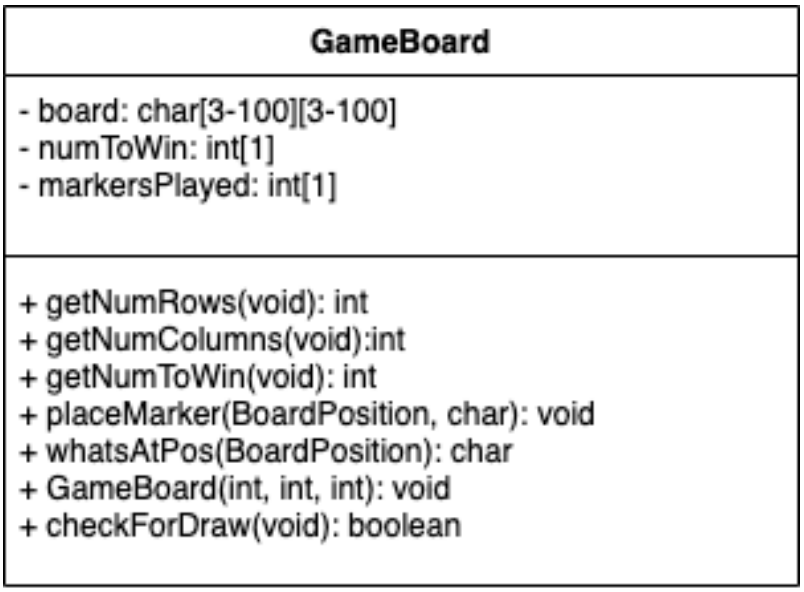
AbsGameBoard
+ toString(void): (String)

Activity Diagram:



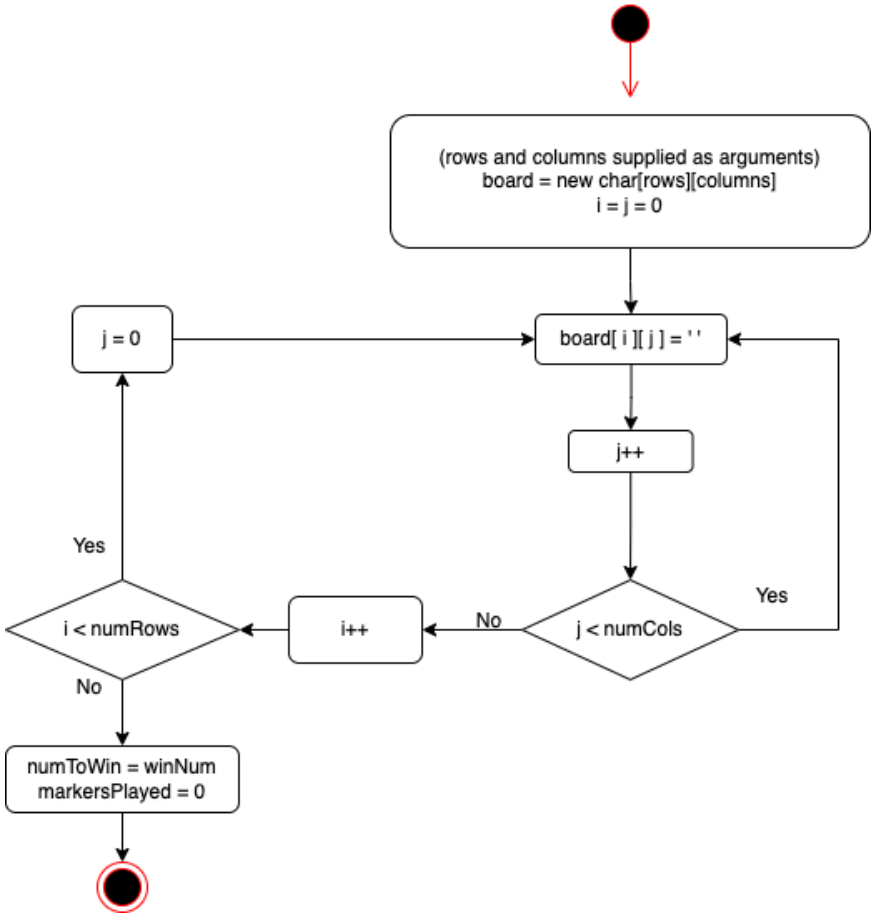
Class 4: GameBoard.java

Class diagram

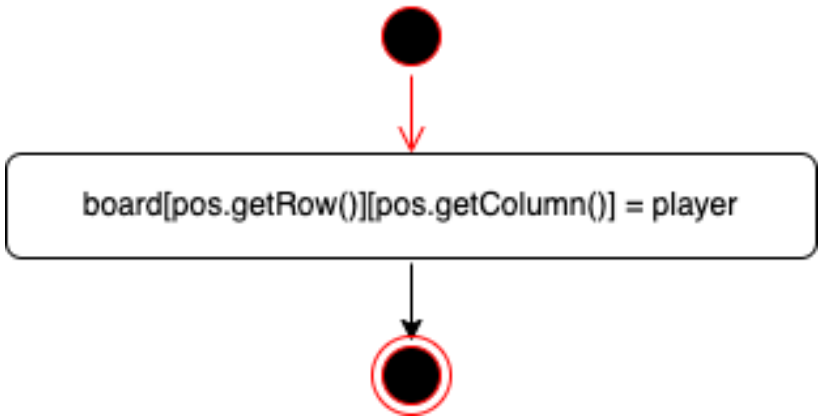


Activity diagrams:

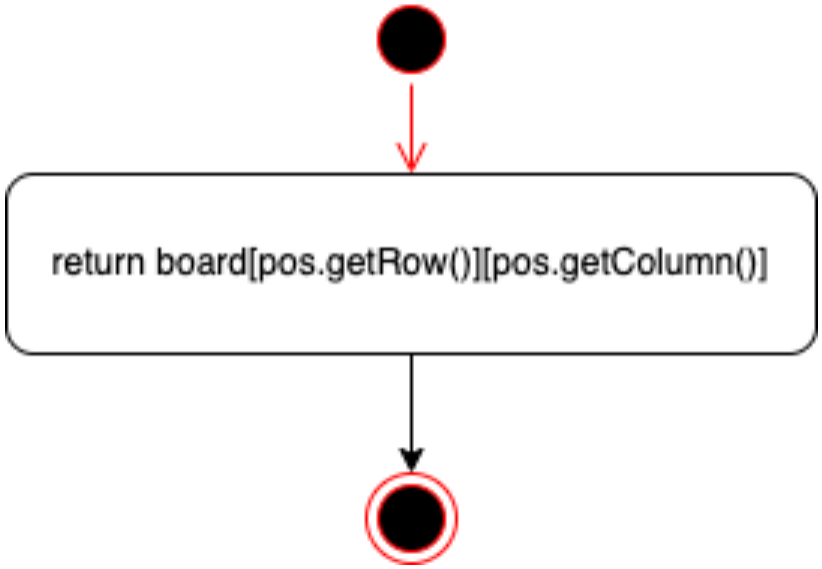
GameBoard:



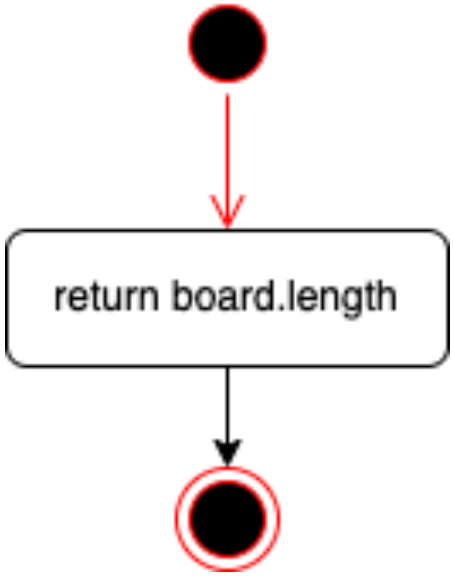
placeMarker:



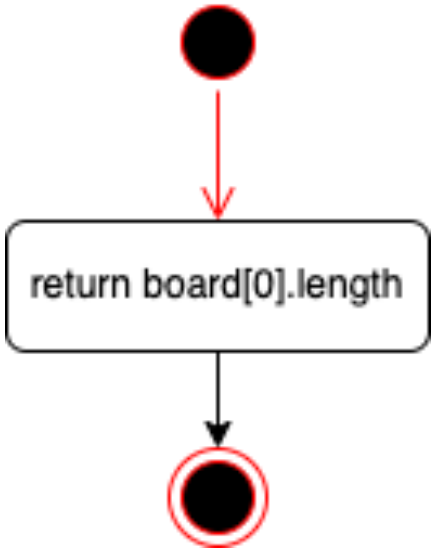
whatsAtPos:



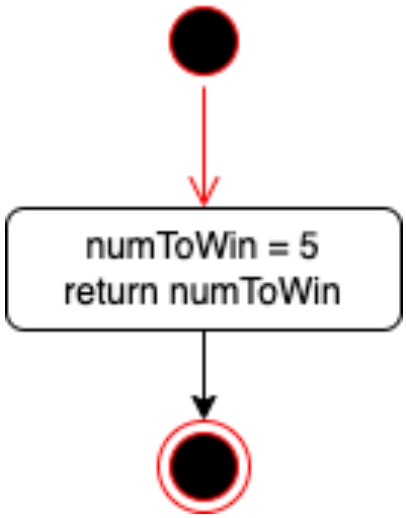
getNumRows:



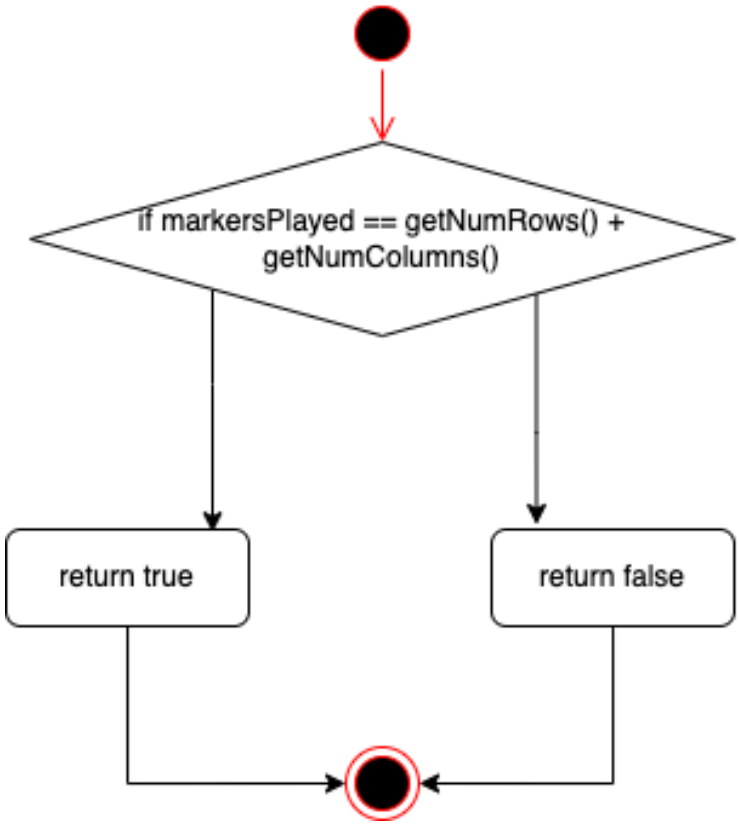
getNumColumns:



getNumToWin:

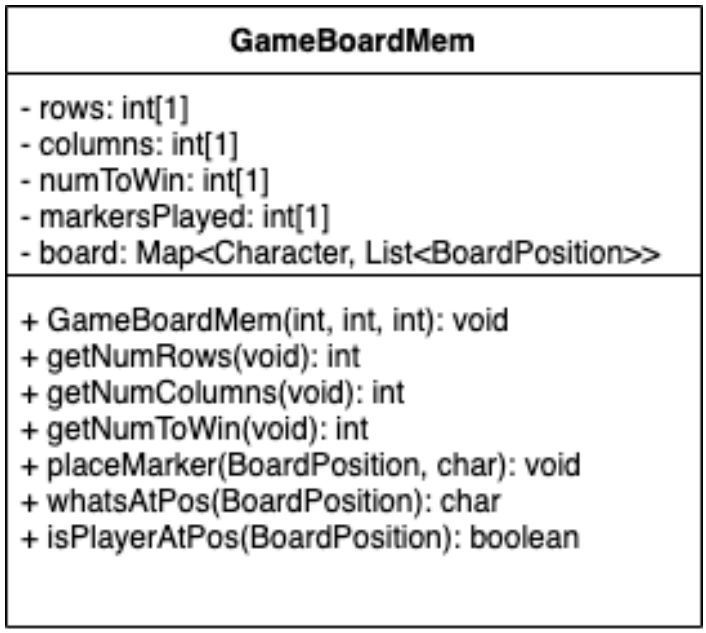


(override) **checkForDraw:**



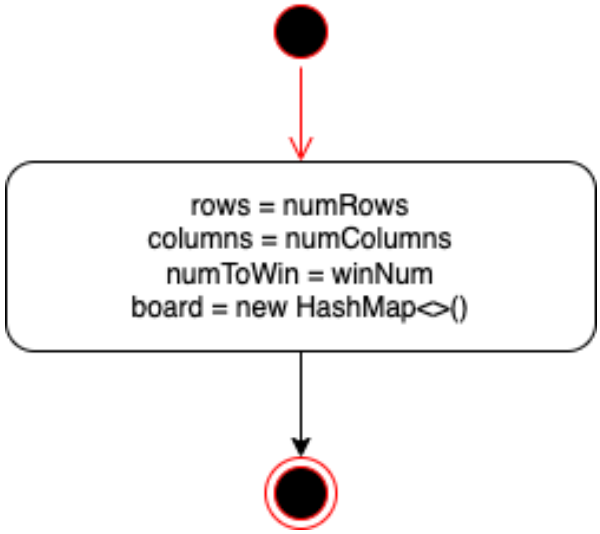
Class 5: GameBoardMem

Class Diagram

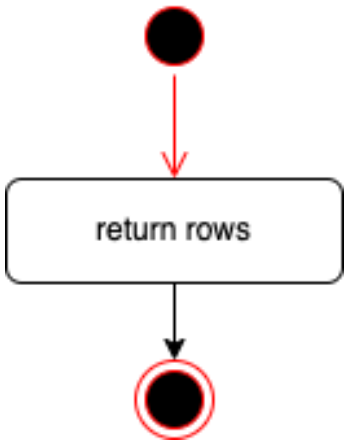


Activity Diagrams

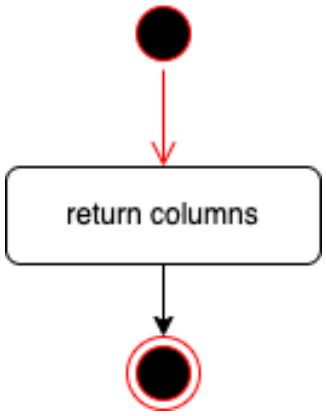
GameBoardMem:



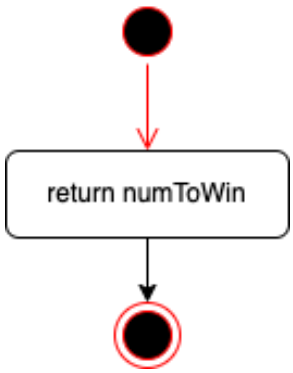
getNumRows:



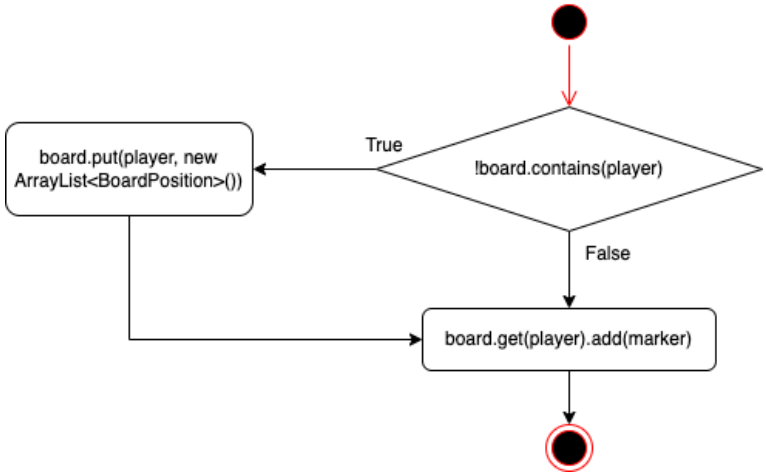
getNumColumns:



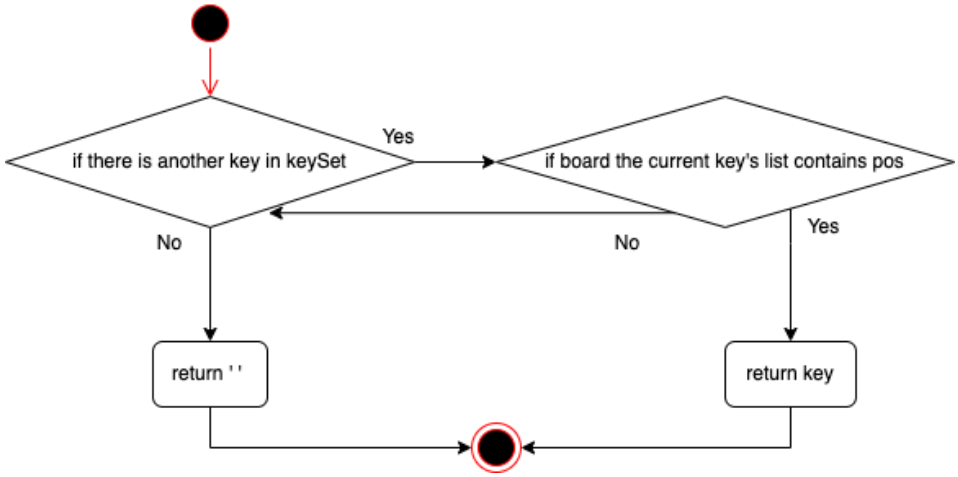
getNumToWin:



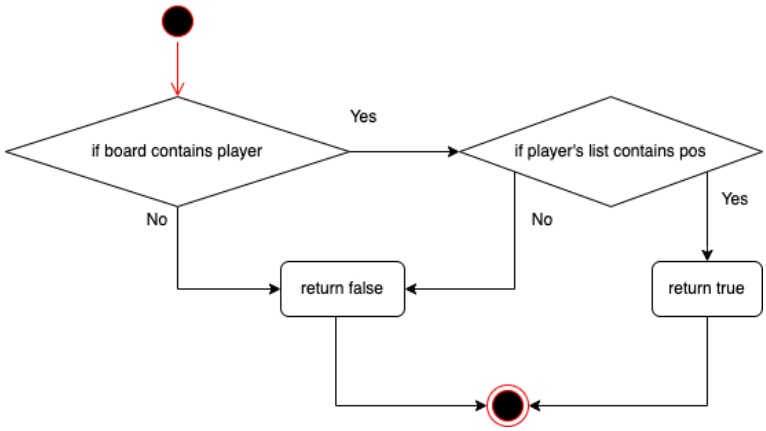
placeMarker:



whatsAtPos:



isPlayerAtPos:

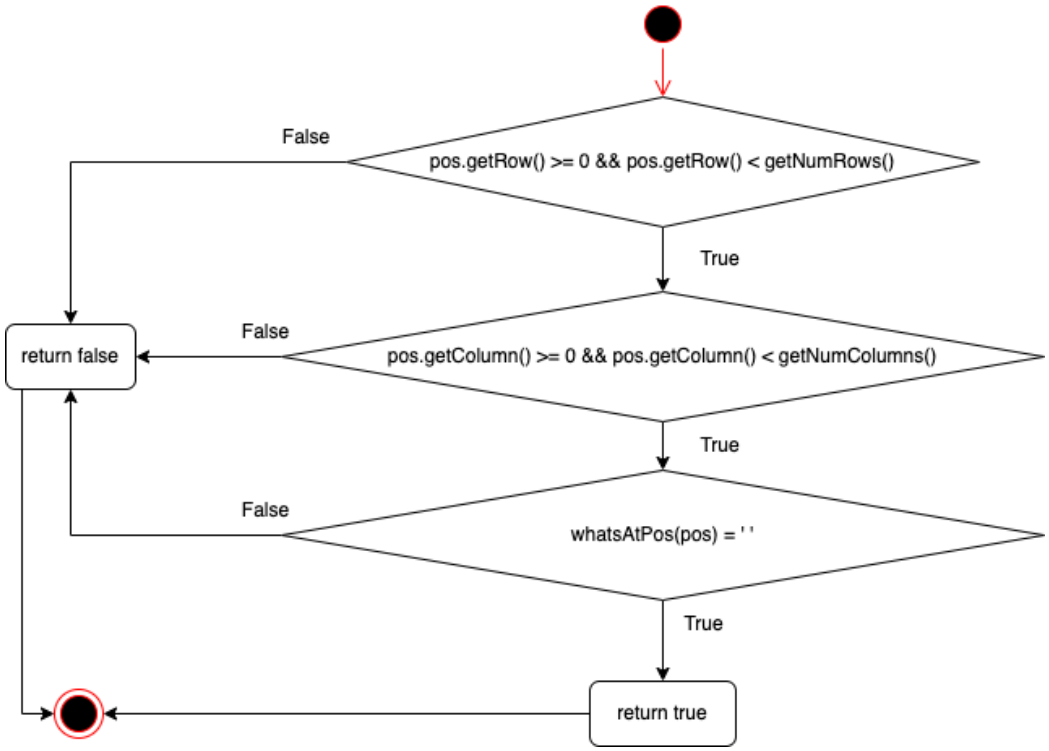


Class 6: IGameBoard

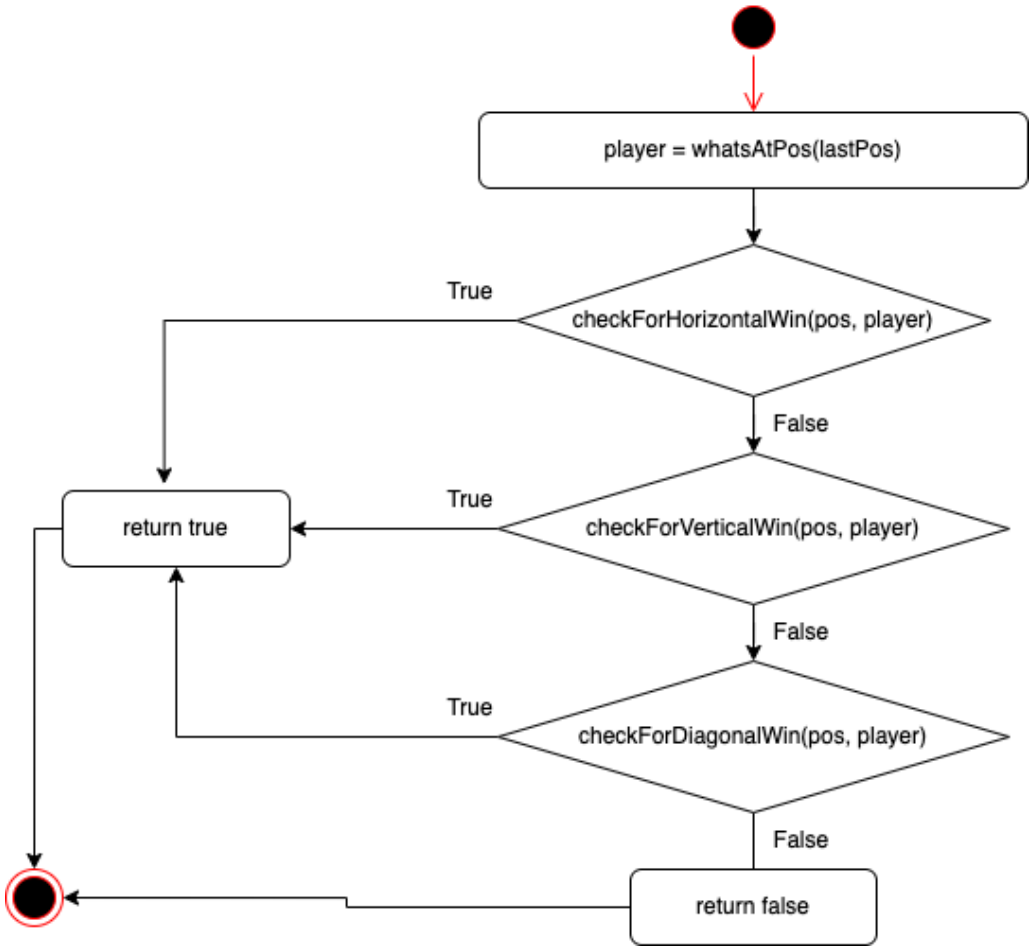
Class Diagram:

Activity Diagram:

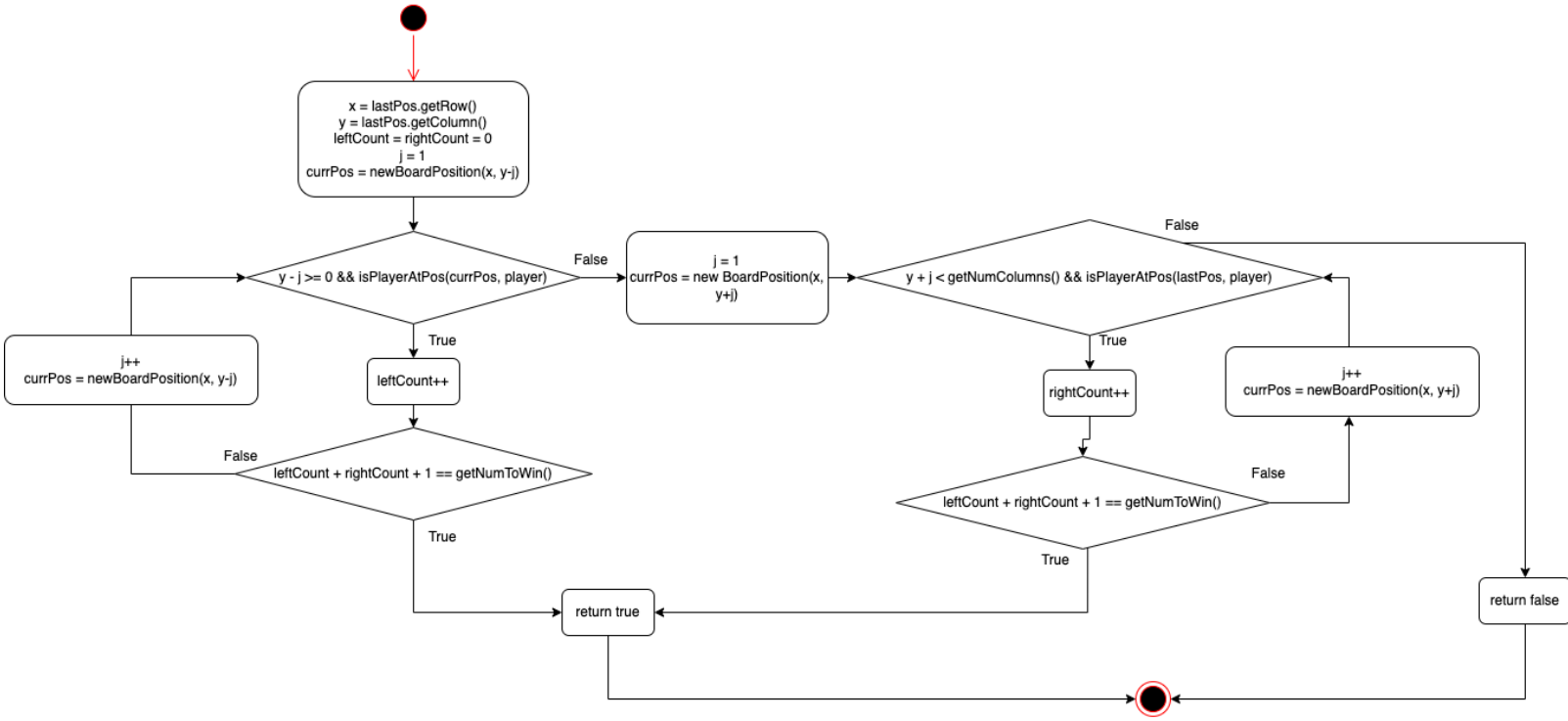
checkSpace:



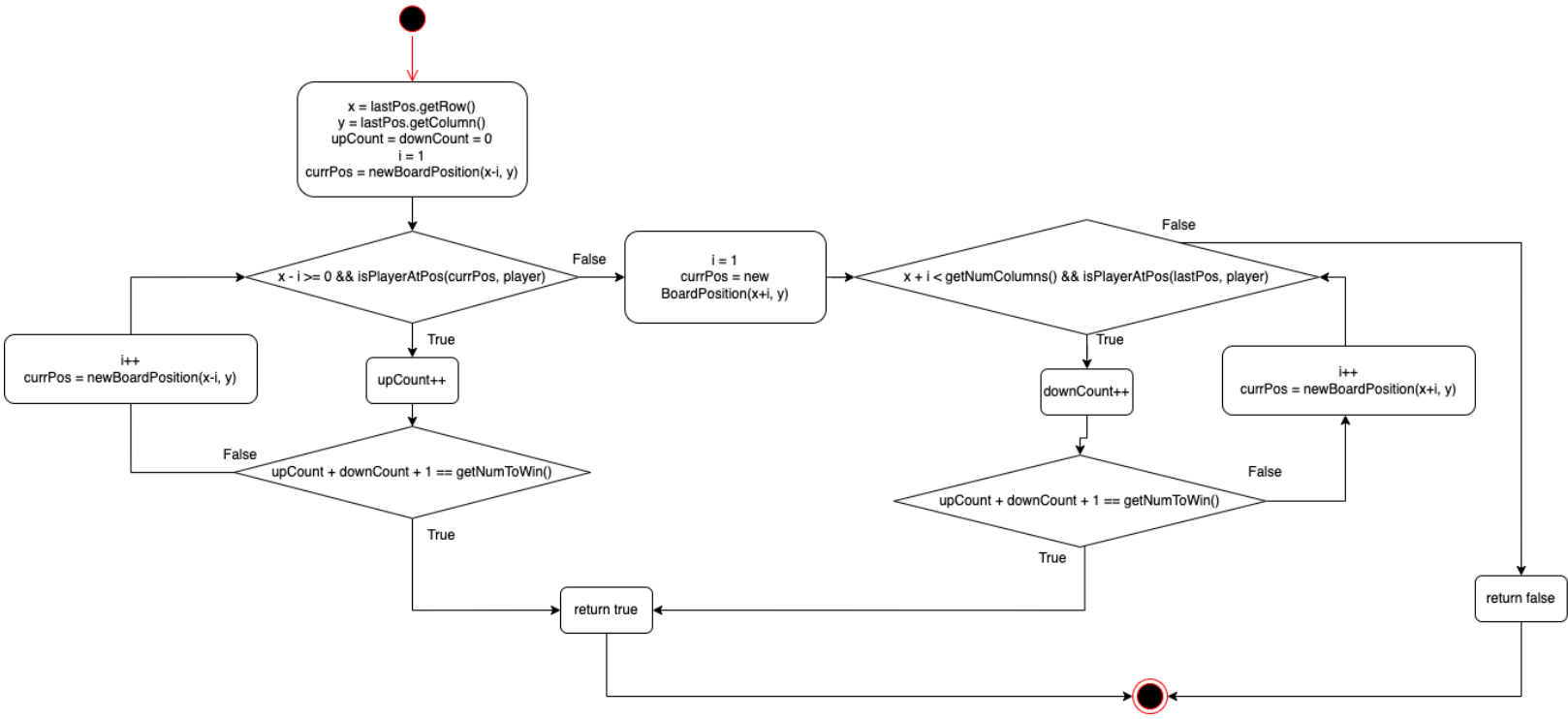
checkForWinner:



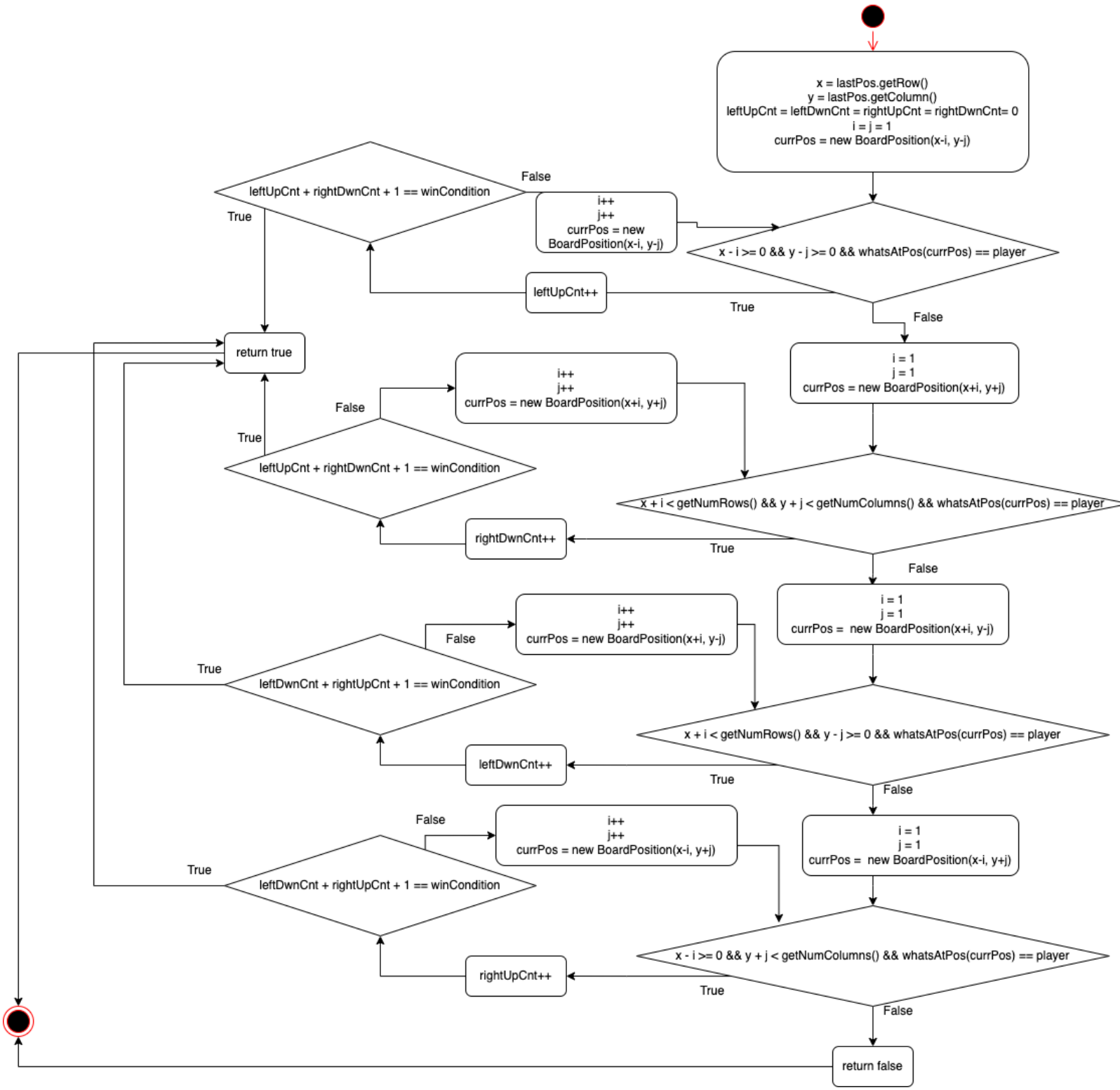
checkHorizontalWin:



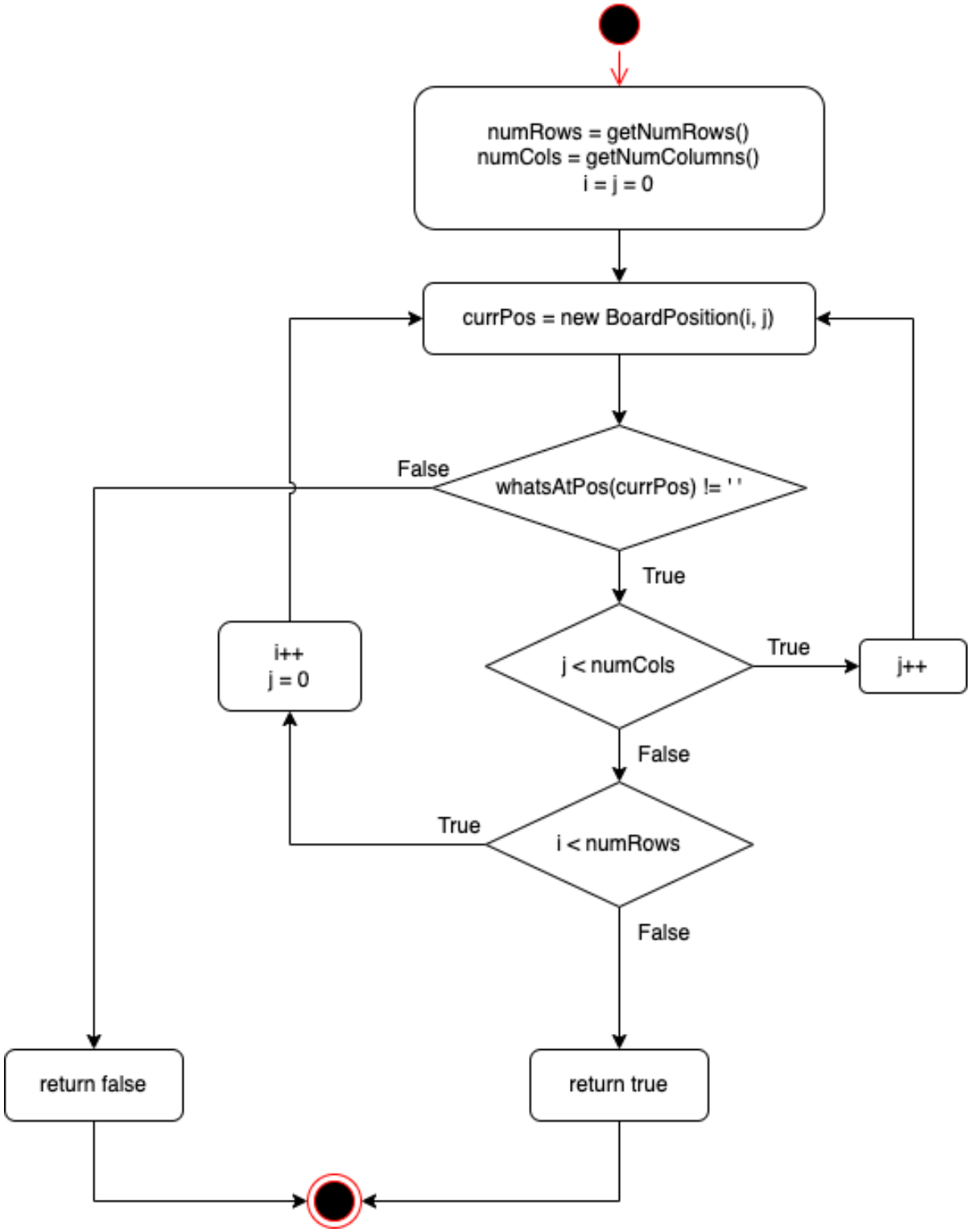
checkVerticalWin:



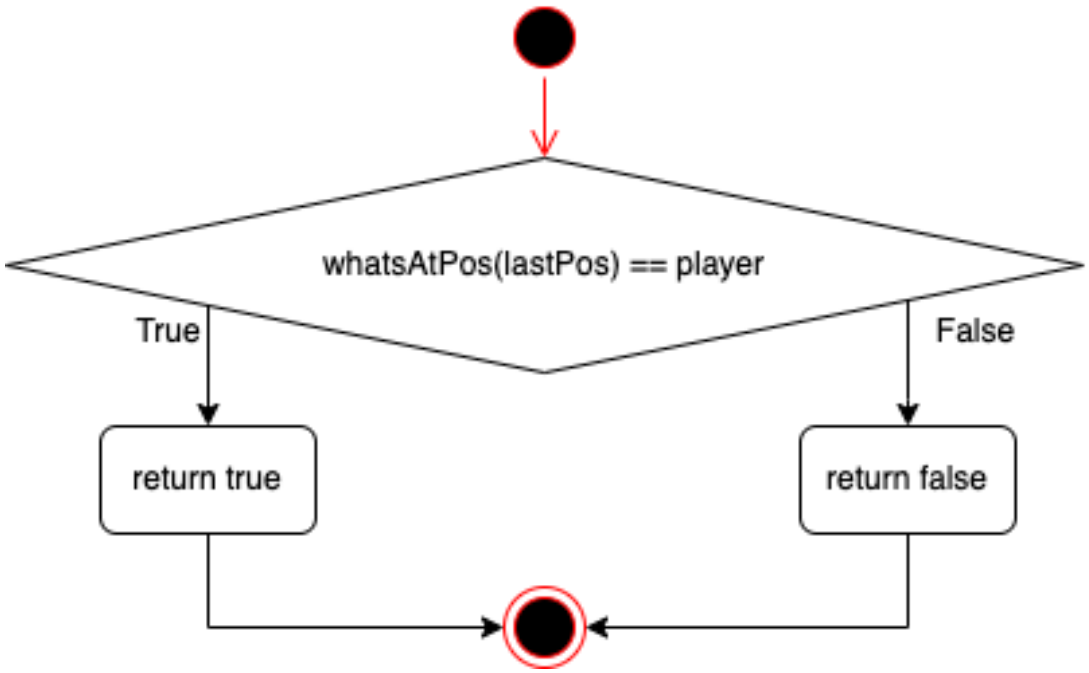
checkDiagonalWin:



checkForDraw:



isPlayerAtPos:



Test Cases

Details in Project 4.

`GameBoard(int rows, int columns, int winNum)`

<table>
<tr>
<td>Input:

State:

rows = 3
columns = 3
winNum = 3</td>
<td>Output:

State:
<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>
numToWin = 3
markersPlayed = 0;</td>
<td>Reason:
This test case is unique because it is creating a board of the minimum size for all 3 parameters

Function:
<code>testGameBoard_min_rows_cols_winNum</code></td>
</tr>
<tr>
<td>Input:

State:

rows = 100
columns = 100
winNum = 25</td>
<td>Output:

State:
(empty 100x100 table)

numToWin = 25
markersPlayed = 0;</td>
<td>Reason:
This test case is unique because it creates a board with the maximum number of rows and columns using the maximum number to win

Function:
<code>testGameBoard_max_rows_cols_winNum</code></td>
</tr>
<tr>
<td>Input:

State:

rows = 5
columns = 5
winNum = 4</td>
<td>Output:

State:
<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>
numToWin = 4
markersPlayed = 0;</td>
<td>Reason:
This test case is unique because it creates a routine board with no minimums or maximums for any parameters

Function:
<code>testGameBoard_no_max_no_min</code></td>
</tr>
</table>

`checkSpace(BoardPosition pos)`

<table>
<tr>
<td>Input:

State:
<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>

pos.getRow = 0
pos.getColumn = 0</td>
<td>Output:
checkSpace = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a routine test of an open space

Function:
<code>testCheckSpace_open_space</code></td>
</tr>
<tr>
<td>Input:

State:
<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>
<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>

pos.getRow = 0
pos.getColumn = 0</td>
<td>Output:
checkSpace = false

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a routine test of an occupied space

Function:
<code>testCheckSpace_taken_space</code></td>
</tr>
<tr>
<td>Input:

State:
<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>

pos.getRow = 0
pos.getColumn = 5</td>
<td>Output:
checkSpace = false

state of the board is unchanged</td>
<td>Reason:
This case is unique because it tests that the function returns false when a position not on the board is passed in

Function:
<code>testCheckSpace_invalid_space</code></td>
</tr>
</table>

`checkHorizontalWin(BoardPosition lastPos, char player)`

<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>O</td><td>O</td><td>O</td><td>O</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>X</td><td></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>X</td><td></td><td></td><td></td><td></td></tr>
</table>

lastPos.getRow = 0
lastPos.getColumn = 3
player = ‘O’</td>
<td>Output:
checkHorizontalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a win resulting from a play on the right side of the win

Function:
<code>testCheckHorizontalWin_play_on_end</code></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></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>X</td><td></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>X</td><td></td><td></td><td></td><td></td></tr>
</table>

lastPos.getRow = 0
lastPos.getColumn = 2
player = ‘O’</td>
<td>Output:
checkHorizontalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a win resulting from a play in the middle of the winning string (meaning the function must check both left and right of the last position played)

Function:
<code>testCheckHorizontalWin_play_in_middle</code></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>X</td><td>X</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>4</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td></tr>
</table>

lastPos.getRow = 0
lastPos.getColumn = 3
player = ‘O’</td>
<td>Output:
checkHorizontalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it tests that the function can recognize a win with a full board

Function:
<code>testCheckHorizontalWin_full</code></td>
</tr>
</table>

Input:

State: (numToWin = 4)

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

lastPos.getRow = 0
lastPos.getColumn = 0
player = 'O'

Output:
checkHorizontalWin = true

state of the board is unchanged

Reason:
This case is unique because it is a win resulting from a play in the corner of the board that results in win. (It should not try and read out of bounds when reading left)

Function:
`testCheckHorizontalWin_corner_win`

`checkVerticalWin(BoardPosition lastPos, char player)`

<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>O</td><td>X</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td></td><td>X</td><td></td><td></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>X</td><td></td><td></td><td></td></tr>
</table>

lastPos.getRow = 1
lastPos.getColumn = 1
player = ‘X’</td>
<td>Output:
checkVerticalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a win resulting from a play on the end of the winning string

Function:
<code>testCheckVerticalWin_play_on_end</code></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>O</td><td>X</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td></td><td>X</td><td></td><td></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>X</td><td></td><td></td><td></td></tr>
</table>

lastPos.getRow = 2
lastPos.getColumn = 1
player = ‘X’</td>
<td>Output:
checkVerticalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a win resulting from a play in the middle of the winning string, meaning the function must check both above and below the last played marker

Function:
<code>testCheckVerticalWin_play_in_middle</code></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>X</td><td>X</td><td>O</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>2</td><td>O</td><td>O</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>X</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 = 1
lastPos.getColumn = 3
player = ‘X’</td>
<td>Output:
checkVerticalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it tests that the function can recognize a win with a full board

Function:
<code>testCheckVerticalWin_full</code></td>
</tr>
</table>

Input:

State: (numToWin = 4)

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

lastPos.getRow = 0
lastPos.getColumn = 0
player = 'X'

Output:
checkVerticalWin = true

state of the board is unchanged

Reason:
This case is unique because it is a win where the last played token is in the corner of the board. The function must not access out of bounds when attempting to check for markers above the last played marker

Function:
`testCheckVerticalWin_corner_win`

`checkDiagonalWin(BoardPosition lastPos, char player)`

<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>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></td><td>X</td><td></td><td>O</td><td></td></tr>
<tr><td>3</td><td>X</td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td>O</td><td></td></tr>
</table>

lastPos.getRow = 0
lastPos.getColumn = 3
player = ‘X’</td>
<td>Output:
checkDiagonalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a win resulting from a play on the upper right end of the winning string

Function:
<code>testCheckDiagonalWin_upper_right_win</code></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>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></td><td>X</td><td></td><td>O</td><td></td></tr>
<tr><td>3</td><td>X</td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td>O</td><td></td></tr>
</table>

lastPos.getRow = 3
lastPos.getColumn = 0
player = ‘X’</td>
<td>Output:
checkDiagonalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a win resulting from a play on the lower left end of the winning string

Function:
<code>testCheckDiagonalWin_lower_left_win</code></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>X</td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td>X</td><td></td><td>O</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td>X</td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td>X</td><td></td></tr>
</table>

lastPos.getRow = 1</td>
<td>Output:
checkDiagonalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a win resulting from a play on the upper left end of the winning string

Function:
<code>testCheckDiagonalWin_upper_left_win</code></td>
</tr>
</table>

<table>
<tr>
<td>lastPos.getColumn = 0
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>X</td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td>X</td><td></td><td>O</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td>X</td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td>X</td><td></td></tr>
</table>

lastPos.getRow = 4
lastPos.getColumn = 3
player = ‘X’</td>
<td>Output:
checkDiagonalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a win resulting from a play on the lower right end of the winning string

Function:
<code>testCheckDiagonalWin_lower_right_win</code></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>X</td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td>X</td><td></td><td>O</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td>X</td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td>X</td><td></td></tr>
</table>

lastPos.getRow = 2
lastPos.getColumn = 1
player = ‘X’</td>
<td>Output:
checkDiagonalWin = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a win resulting from a play in the middle of the winning string where both the upper left and lower right diagonals must be checked

Function:
<code>testCheckDiagonalWin_upper_left_lower_right_win</code></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></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td>X</td><td>O</td><td></td></tr><tr><td>3</td><td></td><td>X</td><td></td><td>O</td><td></td></tr><tr><td>4</td><td>X</td><td></td><td></td><td>O</td><td></td></tr></table>
lastPos.getRow = 2
lastPos.getColumn = 2
player = ‘X’</td>
<td>state of the board is unchanged</td>
<td>This case is unique because it is a win resulting from a play in the middle of the winning string where both the lower left and upper right diagonals must be checked

Function:
<code>testCheckDiagonalWin_lower_left_upper_right_win</code></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>O</td><td></td></tr><tr><td>2</td><td></td><td></td><td>X</td><td>O</td><td></td></tr><tr><td>3</td><td></td><td>X</td><td></td><td>O</td><td></td></tr><tr><td>4</td><td>X</td><td></td><td></td><td></td><td></td></tr></table>
lastPos.getRow = 2
lastPos.getColumn = 2
player = ‘X’</td>
<td>Output:
checkDiagonalWin = false

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a routine check on the last played marker where there is not enough markers in a row.

Function:
<code>testCheckDiagonalWin_no_win</code></td>
</tr>
</table>

`checkForDraw()`

<table>
<tr>
<td>Input:

State:
<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>O</td><td>X</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td></td><td>X</td><td></td><td></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>X</td><td></td><td></td><td></td></tr>
</table>
markersPlayed = 7</td>
<td>Output:
checkForDraw = false

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a routine scenario where there is no draw

Function:
<code>testCheckForDraw_no_draw</code></td>
</tr>
<tr>
<td>Input:

State:
<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>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>2</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>3</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr>
<tr><td>4</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr>
</table>
markersPlayed = 25</td>
<td>Output:
checkForDraw = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a routine check that the number of markers played is equal to the spots available on the board

Function:
<code>testCheckForDraw_draw</code></td>
</tr>
<tr>
<td>Input:

State:
<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>O</td><td></td><td>O</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>2</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>3</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr>
<tr><td>4</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr>
</table>
markersPlayed = 24</td>
<td>Output:
checkForDraw = false

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a check of the board where all spaces are filled except one. The function should recognize that there is still an available space therefore there is no draw

Function:
<code>testCheckForDraw_near_draw</code></td>
</tr>
<tr>
<td>Input:
State:
<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>
markersPlayed = 0</td>
<td>Output:
checkForDraw = false

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a check for draw on an empty board. It should recognize there is not a draw

Function:
<code>testCheckForDraw_empty_board</code></td>
</tr>
</table>

`whatsAtPos(BoardPosition pos)`

<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>
<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>

pos.getRow = 1
pos.getColumn = 1</td>
<td>Output:
whatsAtPos = ‘ ‘

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a routine check of a blank space

Function:
<code>testWhatsAtPos_blank_space</code></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>X</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>

pos.getRow = 1
pos.getColumn = 1</td>
<td>Output:
whatsAtPos = ‘X‘

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a routine check of a space that has been played on

Function:
<code>testWhatsAtPos_non_edge_taken_space</code></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>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>
<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>

pos.getRow = 0
pos.getColumn = 0</td>
<td>Output:
whatsAtPos = ‘X‘

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a boundary check on the corner of the board

Function:
<code>testWhatsAtPos_taken_corner_space</code></td>
</tr>
</table>

<table>
<tr><td></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>X</td><td>X</td><td>O</td><td>O</td><td>X</td></tr>
<tr><td>1</td><td>O</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td></td><td>O</td></tr>
<tr><td>3</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>4</td><td>X</td><td>O</td><td>O</td><td>X</td><td>O</td></tr>
</table>

pos.getRow = 2
pos.getColumn = 3</td>
<td>Output:
whatsAtPos = ‘ ‘

state of the board is unchanged</td>
<td>Reason:
This case is unique because it checks that the function recognizes there is an open space left and that this spot is not taken

Function:
<code>testWhatsAtPos_almost_full_board</code></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>X</td><td>X</td><td>O</td><td>O</td><td>X</td></tr>
<tr><td>1</td><td>O</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>3</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>4</td><td>X</td><td>O</td><td>O</td><td>X</td><td>O</td></tr>
</table>

pos.getRow = 1
pos.getColumn = 3</td>
<td>Output:
whatsAtPos = ‘X‘

state of the board is unchanged</td>
<td>Reason:
This case is unique because it checks that the function can return the correct character on a full board

Function:
<code>testWhatsAtPos_full_board</code></td>
</tr>
</table>

`isPlayerAtPos(BoardPosition pos, char player)`

<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>
<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>

pos.getRow = 1
pos.getColumn = 1
player = ‘X’</td>
<td>Output:
isPlayerAtPos = false

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a routine check of a blank space

Function:
<code>testIsPlayerAtPos_blank_space</code></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>X</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>

pos.getRow = 1
pos.getColumn = 1
player = ‘X’</td>
<td>Output:
isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a routine check of a non-edge space that has been played on

Function:
<code>testIsPlayerAtPos_non_edge_taken_space</code></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>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>
<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>

pos.getRow = 0
pos.getColumn = 0
player = ‘X’</td>
<td>Output:
isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This case is unique because it is a boundary check on the corner of the board

Function:
<code>testIsPlayerAtPos_taken_corner_space</code></td>
</tr>
</table>

<table>
<tr><td></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>X</td><td>X</td><td>O</td><td>O</td><td>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>O</td><td>O</td><td>X</td><td></td><td>O</td></tr><tr><td>3</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>4</td><td>X</td><td>O</td><td>O</td><td>X</td><td>O</td></tr></table>
pos.getRow = 2
pos.getColumn = 3
player = ‘O’</td><td>Output:
isPlayerAtPos = false

state of the board is unchanged</td><td>Reason:
This case is unique because it checks that the function recognizes there is an open space left and that this spot is not taken

Function:
<code>testIsPlayerAtPos_open_almost_full_board</code></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>X</td><td>O</td><td>X</td><td>O</td></tr><tr><td>1</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>2</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr><tr><td>3</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>4</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr></table>
pos.getRow = 2
pos.getColumn = 3
player = ‘X’</td><td>Output:
isPlayerAtPos = true

state of the board is unchanged</td><td>Reason:
This case is unique because it checks that the function can return the correct character on a full board

Function:
<code>testIsPlayerAtPos_full_board</code></td></tr>
</table>

`placeMarker(BoardPosition marker, char player)`

Input:

State: (numToWin = 4)

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

pos.getRow = 2
pos.getColumn = 3
player = X

Output:

State: (numToWin = 4)

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

pos.getRow = 2
pos.getColumn = 3
player = X

Reason:
This case is unique because it places a marker on the board in the last available spot to play on (making the board full)

Function:
`testPlaceMarker_fill_board`

Input:

State: (numToWin = 4)

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

marker.getRow = 0
marker.getColumn = 0
player = X

Output:

State: (numToWin = 4)

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

Reason:
This case is unique because it places a marker of a third player that has not yet been played

Function:
`testPlaceMarker_adding_third_player`

Input:

State: (numToWin = 4)

	0	1	2	3	4
0					
1					
2					
3					
4					

marker.getRow = 0
marker.getColumn = 0
player = X

Output:

State: (numToWin = 4)

	0	1	2	3	4
0	0				
1					
2					
3					
4					

Reason:
This case is unique because it is a routine placement on a corner space

Function:
`testPlaceMarker_edge`

Input:

State: (numToWin = 4)

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

marker.getRow = 0
marker.getColumn = 1
player = O

Output:

State: (numToWin = 4)

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

Reason:
This case is unique because it places a marker in the last spot available in a column

Function:
`testPlaceMarker_fill_column`

Input:

State: (numToWin = 4)

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

marker.getRow = 1
marker.getColumn = 4
player = X

Output:

State: (numToWin = 4)

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

Reason:
This case is unique because it places a marker in the last spot available in a row

Function:
`testPlaceMarker_fill_row`