

REQUIREMENT ANALYSIS

Functional Requirements

1. As a player, I can enter my position so that I can make my move on the game board
2. As a player, I can view the game board, so I know the state of the game
3. As a player, I have the option to play the game again so I can play the game again
4. As a player, I need to know if the position I entered is available so that I can play that position or enter in a new one
5. As a player, I need to know if there's a winner or draw so I can end the game or choose to play again
6. As a player, I get to view who won the game, so I know who won the game
7. As a player, I get to view who's turn it is, so I don't go when it is not my turn
8. As a player, I am informed to first enter the row and then the column, so I don't mistakenly enter in the wrong position
9. As a player, I can pick again if I enter in an invalid (out of bounds) position so that I can play a valid position
10. As a player, I can win horizontally (five in a row horizontally) so I can win the game
11. As a player, I can win vertically so I can win the game
12. As a player, I can win diagonally so I can win the game
13. As a player, I play a turn and rotate turns with the other player, so the game is fair
14. As a player, I can pick the number of players that will play the game so the amount of people that will be playing the game can play the game
15. As a player, I am informed if I have picked the number of players that is too big or too small, so I can pick again
16. As a player, I can pick the character that represents me in the game so I will know who I am
17. As a player, I am informed if I have picked a character that is already taken, so I can pick again
18. As a player, I can pick the number of rows for the game board so I can create my desired game board
19. As a player, I am informed If I have picked the number of rows that is too big or too small, so I can pick again
20. As a player, I can pick the number of columns for the game board so I can create my desired game board
21. As a player, I am informed if I have picked the number of columns that is too big or too small, so I can pick again
22. As a player, I can pick the number of tokens in a row to win so I know the number of tokens in a row to win
23. As a player, I am informed if the number of tokens in a row I have entered is too big or too small, so I can pick again
24. As a player, I can choose to play a fast game or a memory efficient game so I will know if the game is taking up too much memory or not
25. As a player, If I choose to play the game again, I am allowed to pick the number of rows, number of columns, number of tokens in a row to win, and type of game board again so I can design my game board however I want when I choose to play again

Non-Functional Requirements

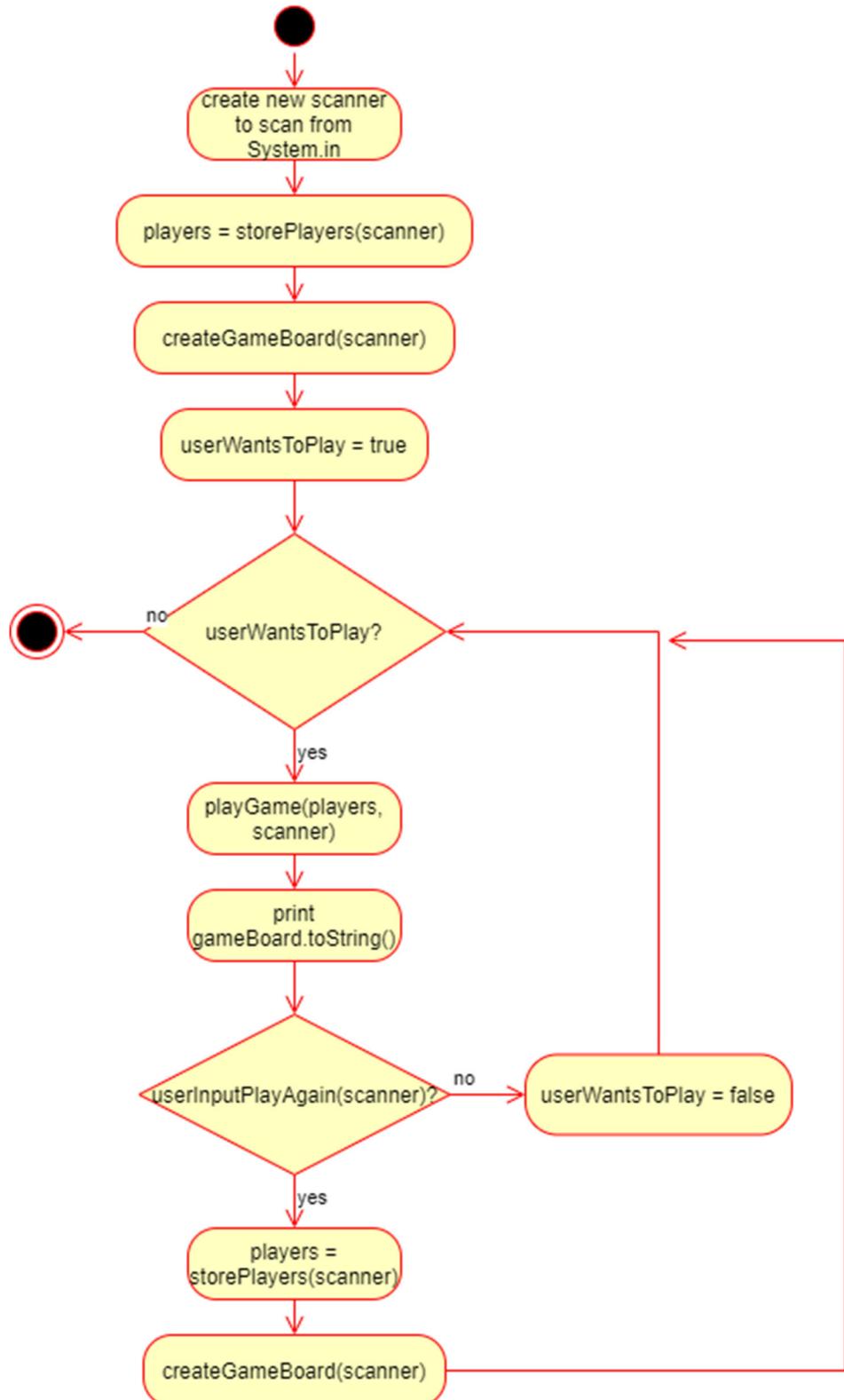
1. The code runs on Unix (school of computing computers)
2. Use Java as the coding language
3. Gameboard at least will have 3 rows and 3 columns
4. Gameboard at most will have 100 rows and 100 columns
5. The number in a row to win will be at least 3 and at most 25
6. The number in a row to win cannot be greater than the number of rows and the number of columns of the game board
7. The number of players is at least 2 and at most 10
8. Players are capitalized characters
9. Player 1 always goes first
10. 0,0 is the top left of the board
11. Code is efficient
12. Cannot have players with same character

DESIGN

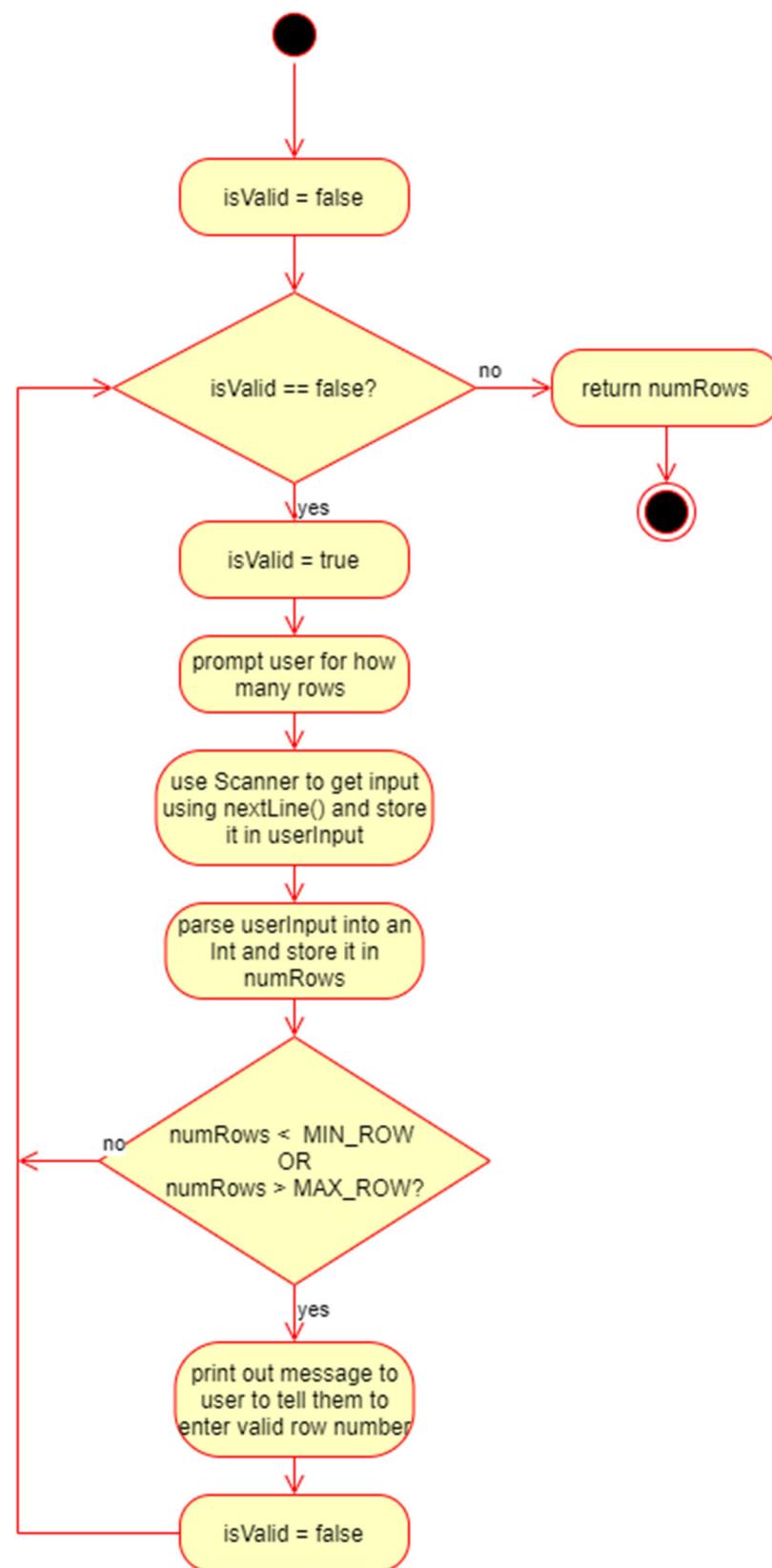
GameScreen Class

Game Screen
- gameBoard: IGameBoard [1]
+ main() : void - getNumRowsFromUser() : int - getNumColumnsFromUser() : int - getNumToWinFromUser(int, int) : int - getGameBoardInfo() : char - createGameBoard () : void - getNumPlayerFromUser() : int - storePlayers() : List<Character> - getPlayerPos(Character) : BoardPosition - userInputPlayAgain() : boolean - playGame(List<Character>) : void

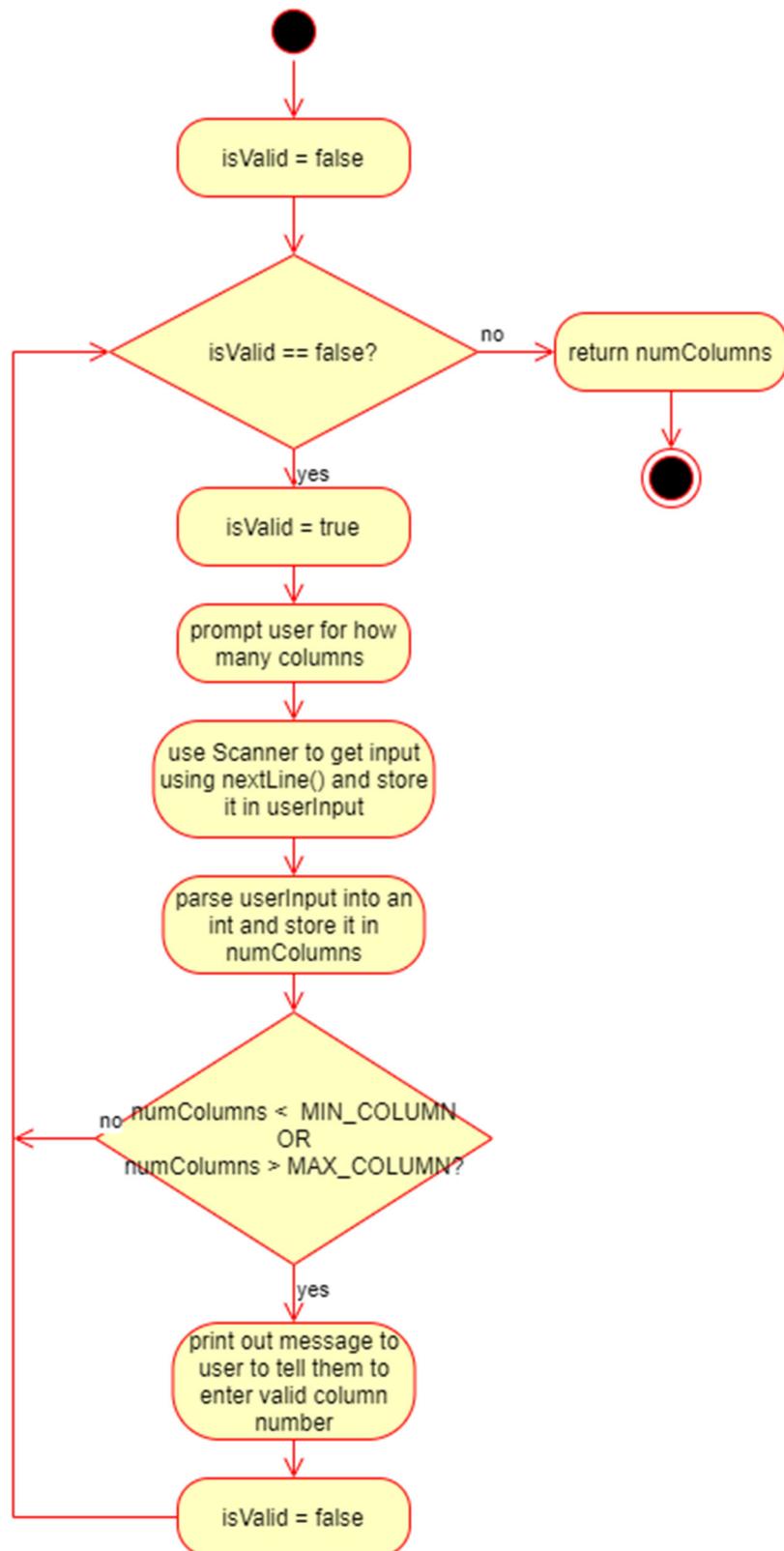
Main



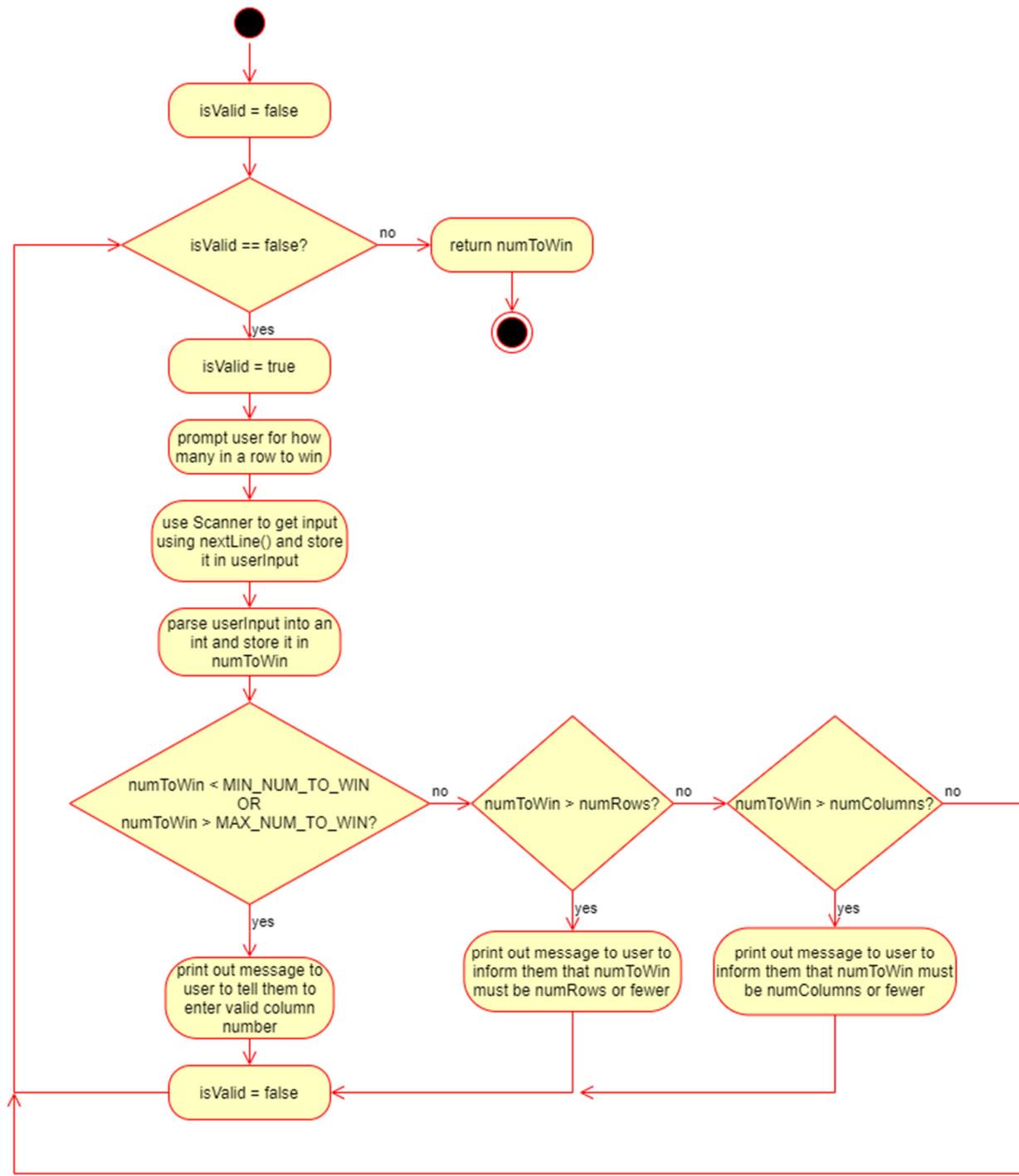
getNumRowsFromUser



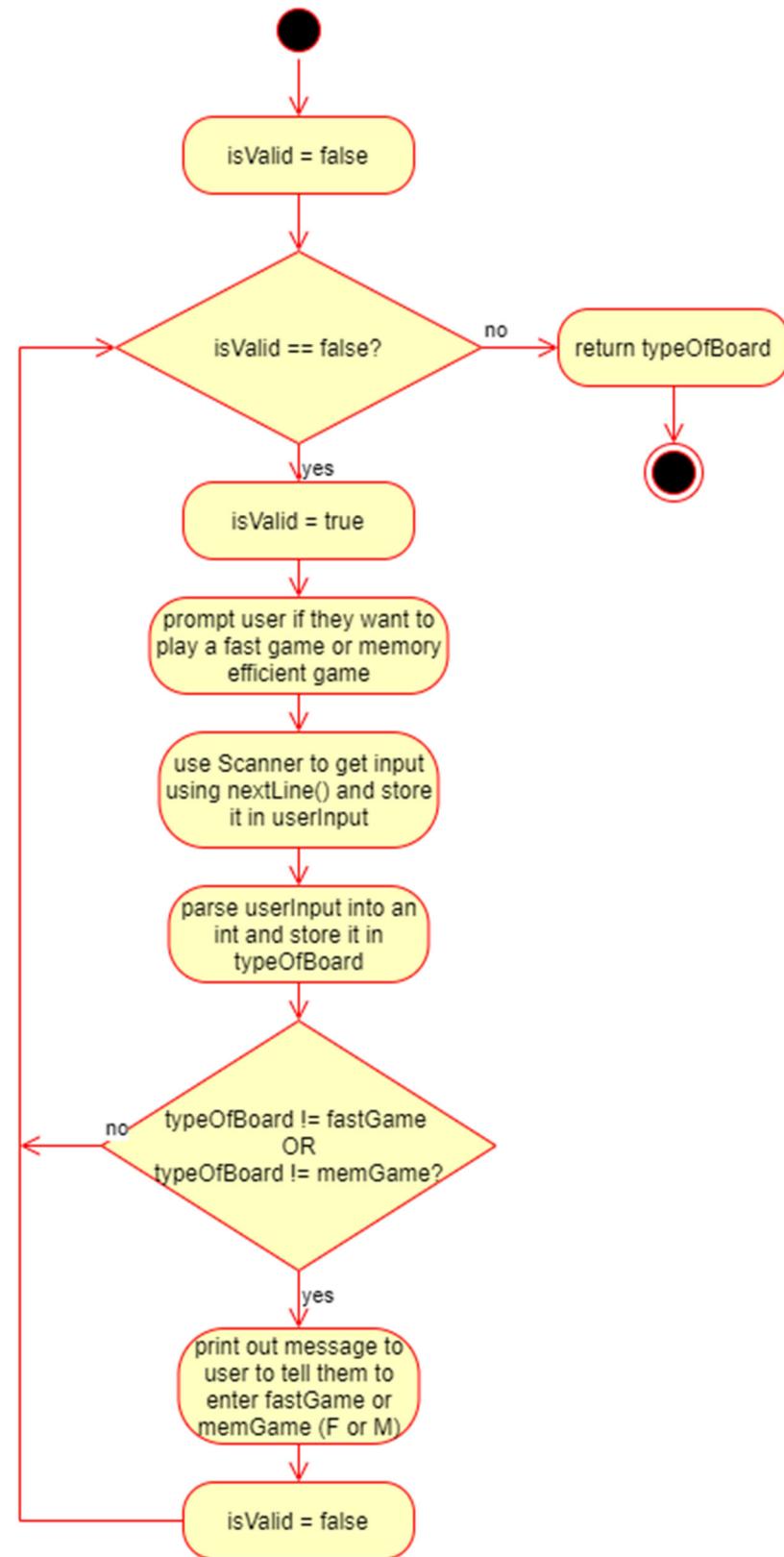
getNumColumnsFromUser



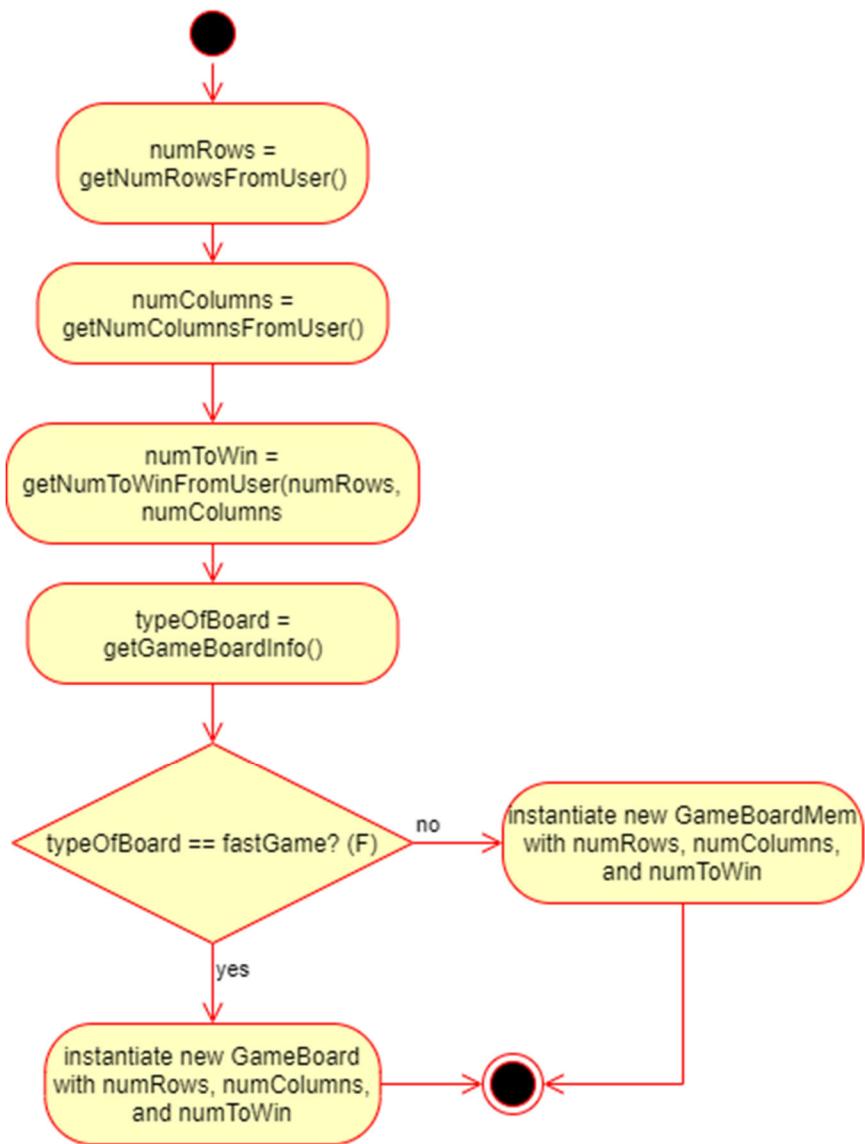
getNumToWinFromUser



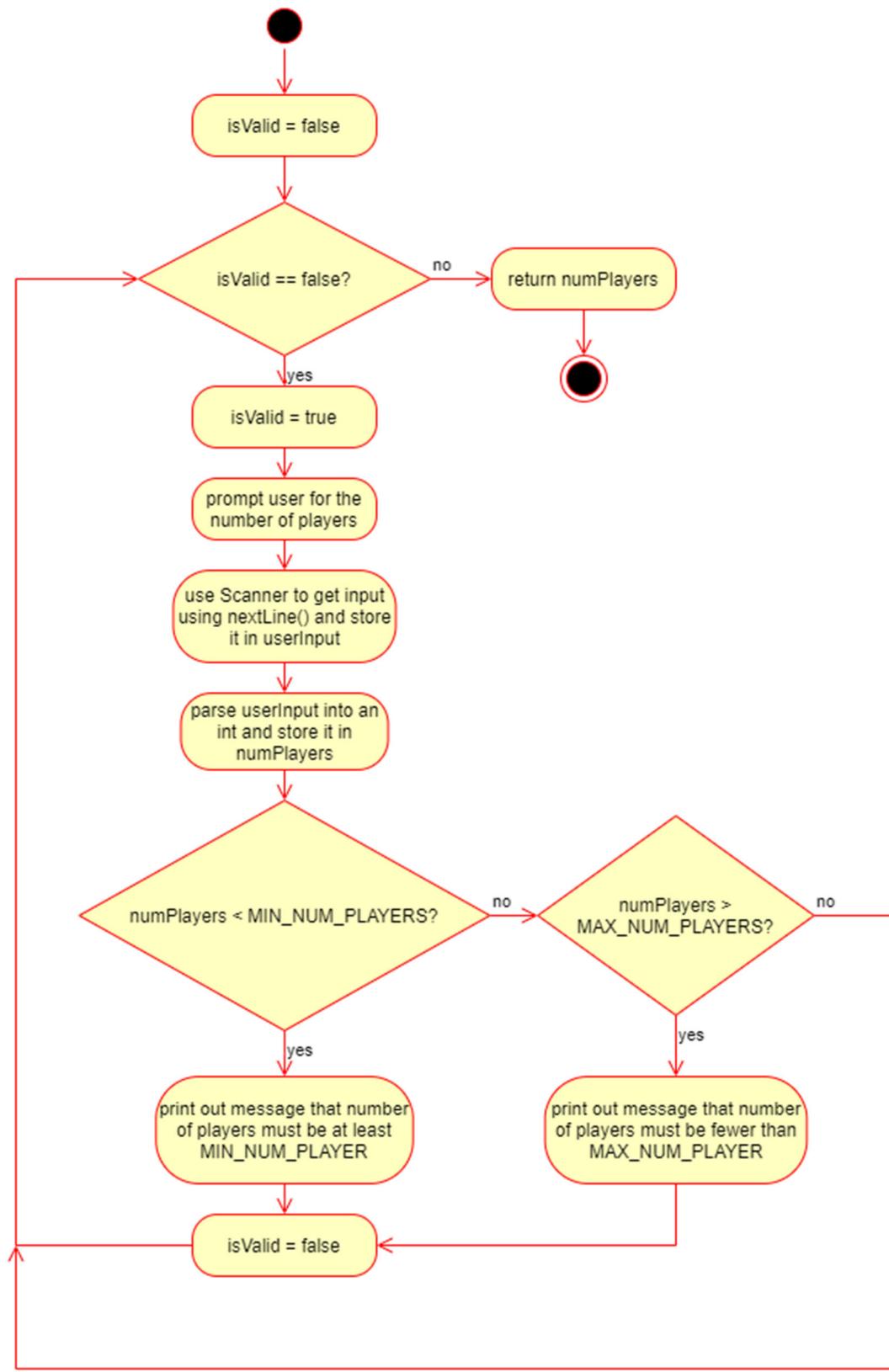
getGameBoardInfo



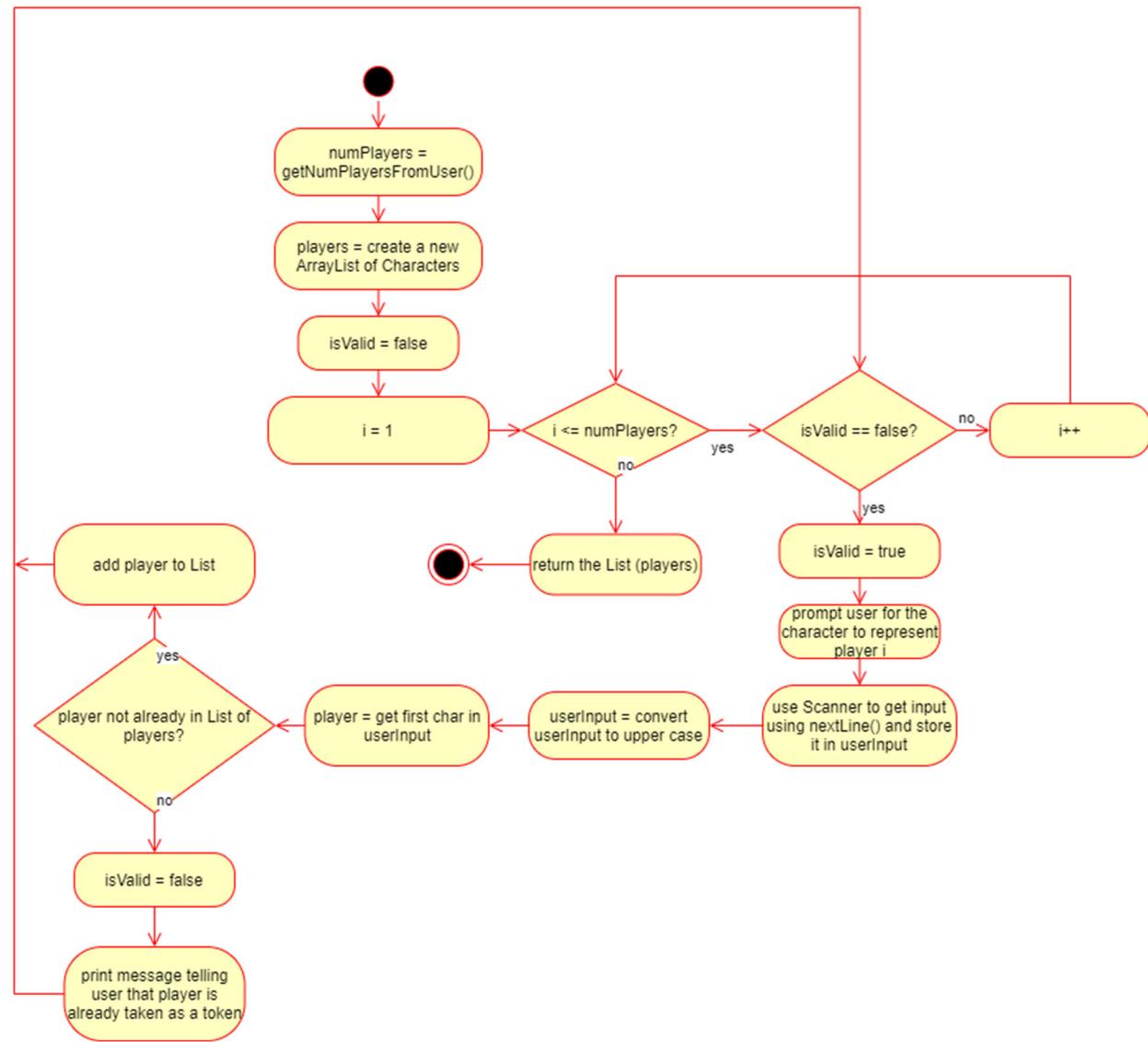
createGameBoard



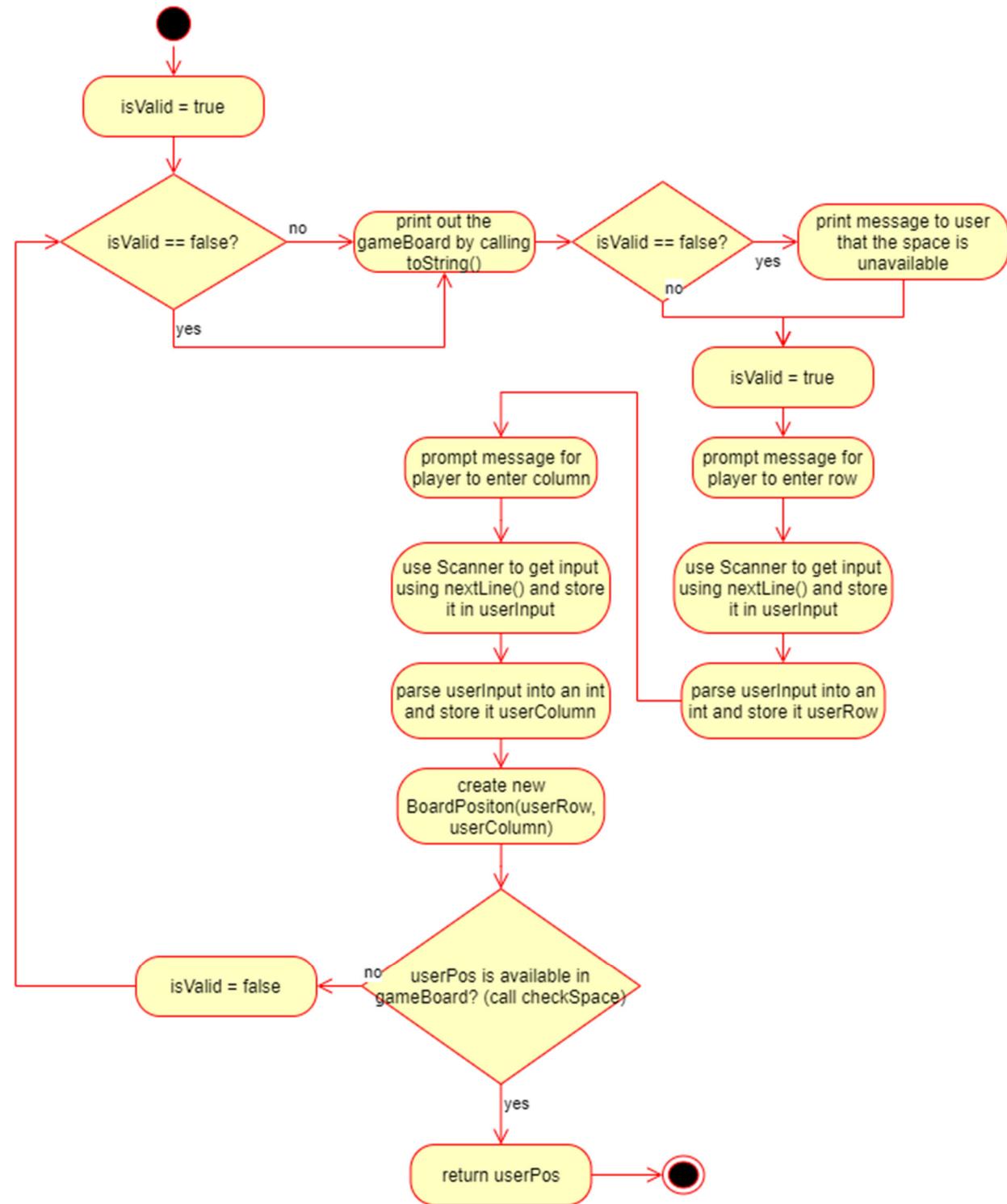
getNumPlayerfromUser



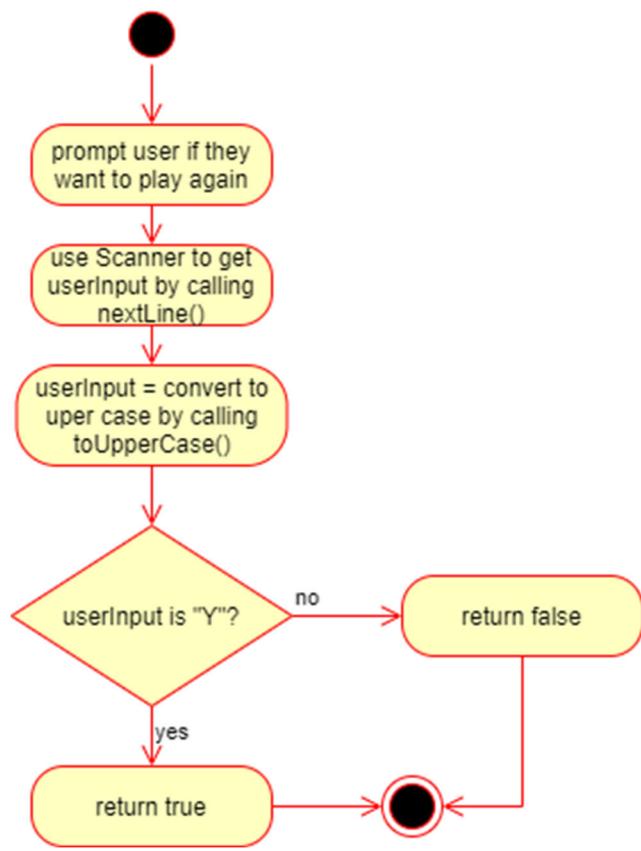
storePlayers



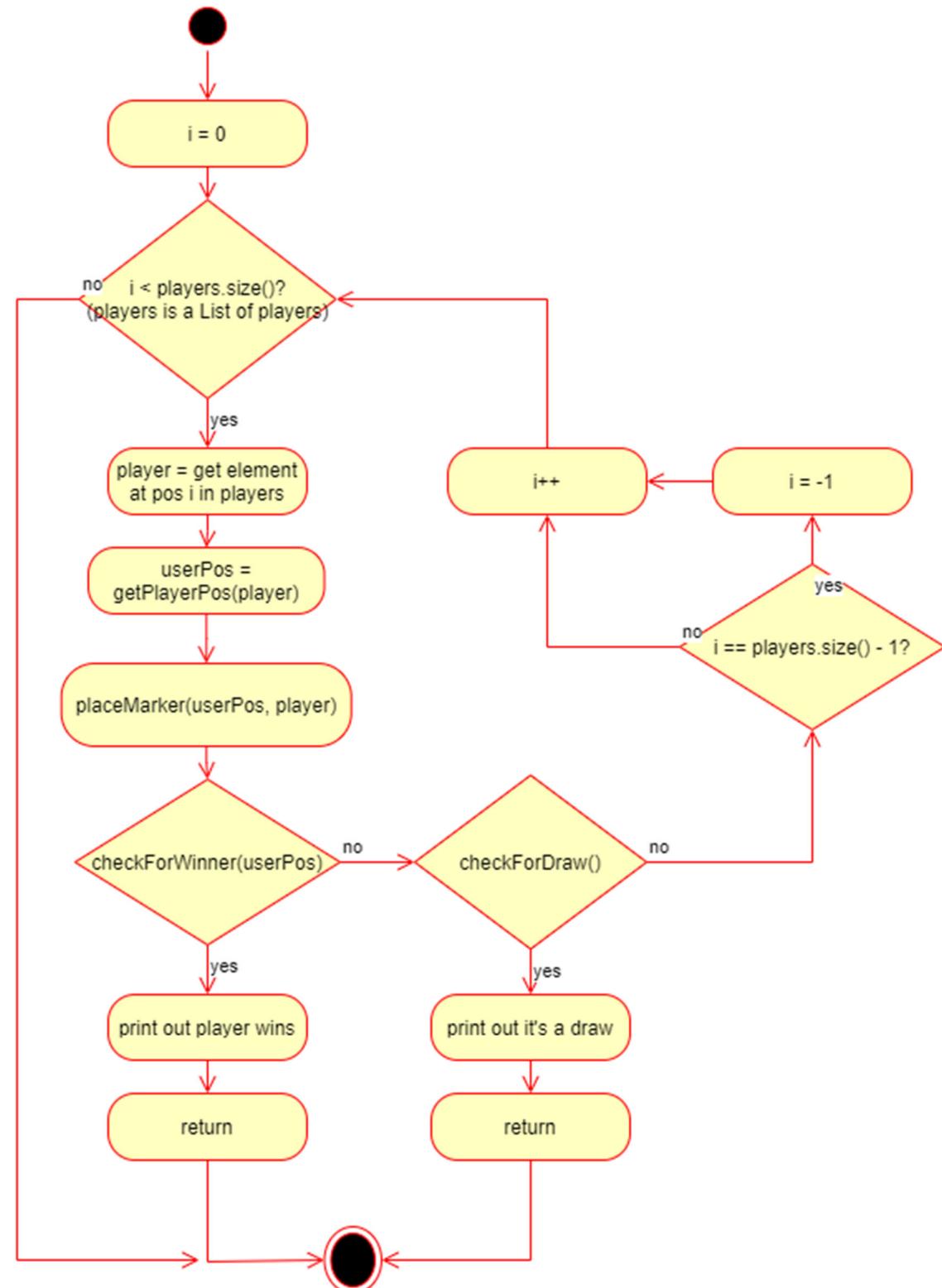
getPlayerPos



userInputPlayAgain



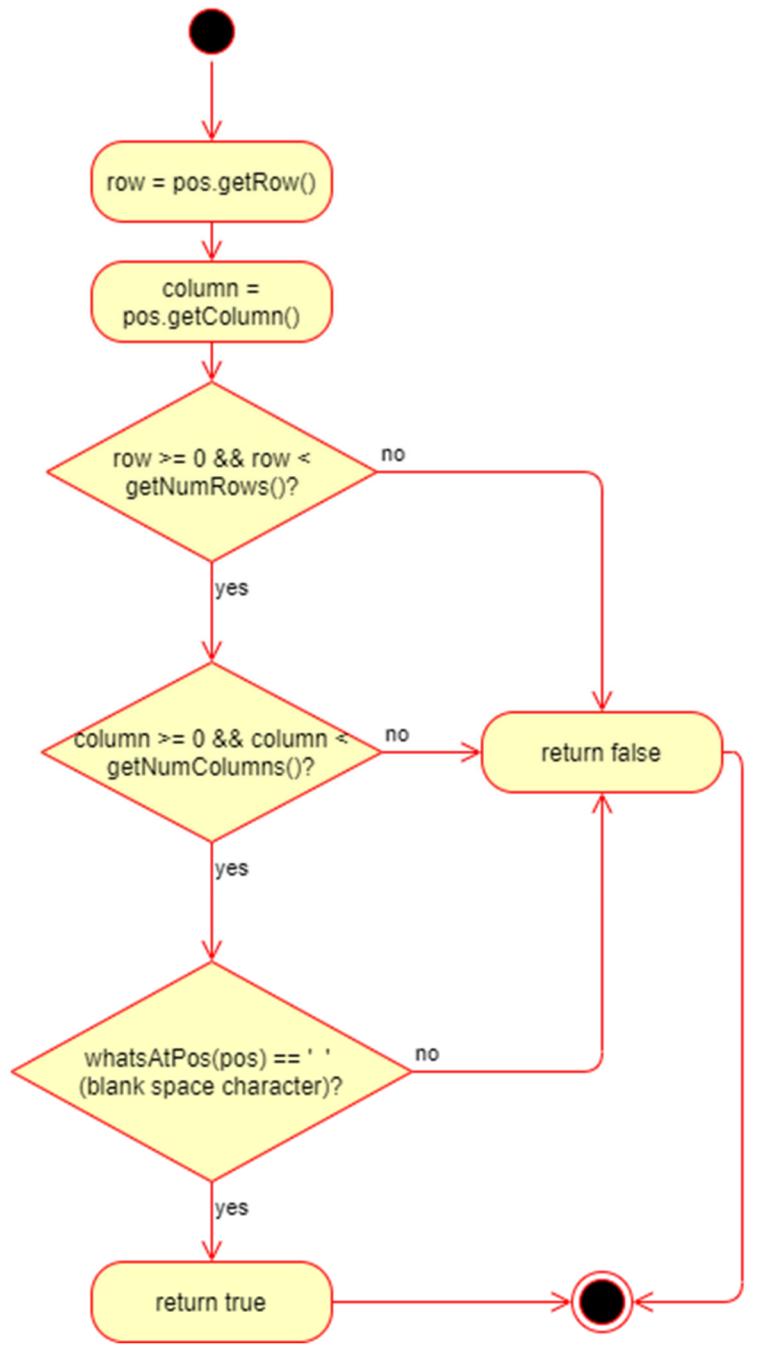
playGame



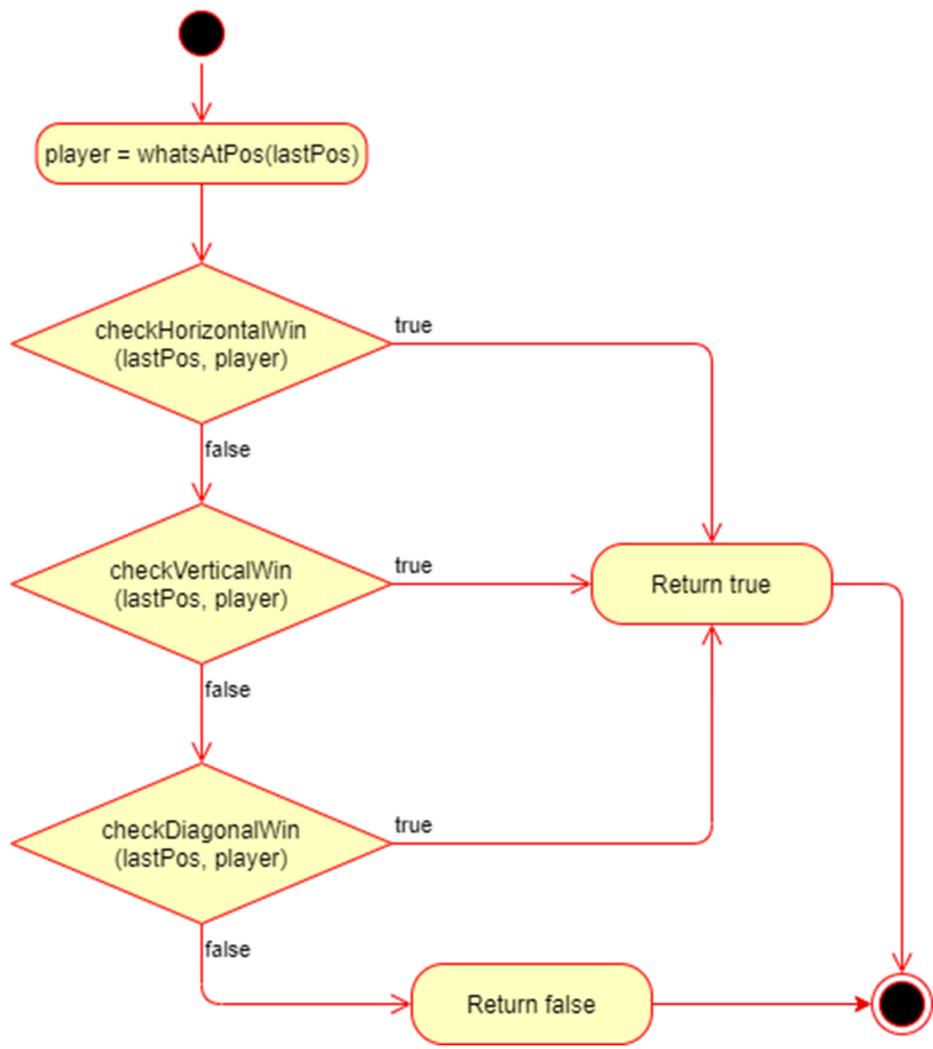
IGameBoard Interface

«interface»	IBoardGame
+ checkSpace(BoardPosition) : boolean + placeMarker (BoardPosition, char) : void + checkForWinner(BoardPosition) : boolean + checkForDraw() : boolean + checkHorizontalWin(BoardPosition, char) : boolean + checkVerticalWin(BoardPosition, char) : boolean + checkDiagonalWin(BoardPosition, char) : boolean + whatsAtPos(BoardPosition) : char + isPlayerAtPos(BoardPosition, char) : boolean + getNumRows() : int + getNumCounums() : int + getNumToWin() : int	

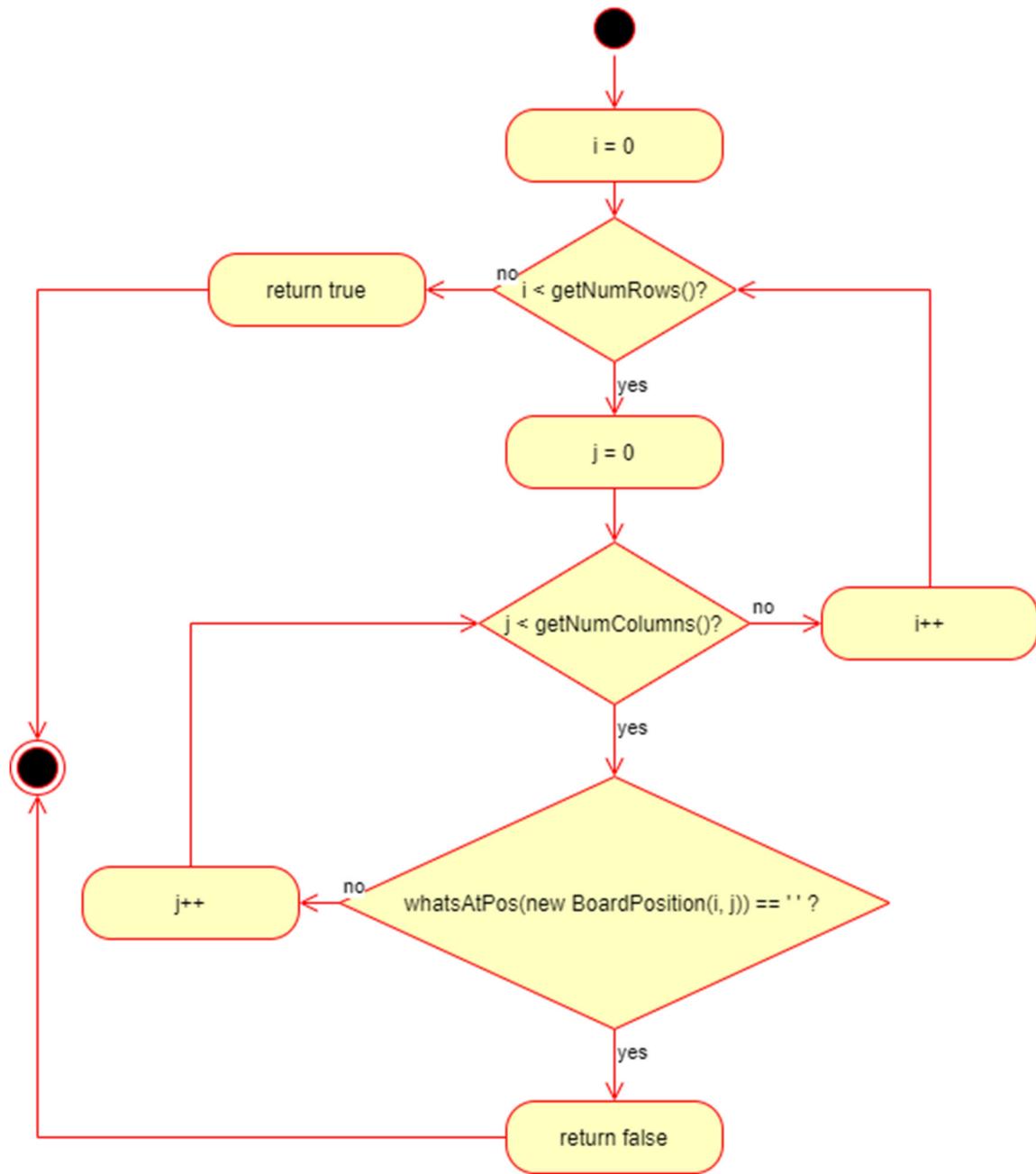
checkSpace



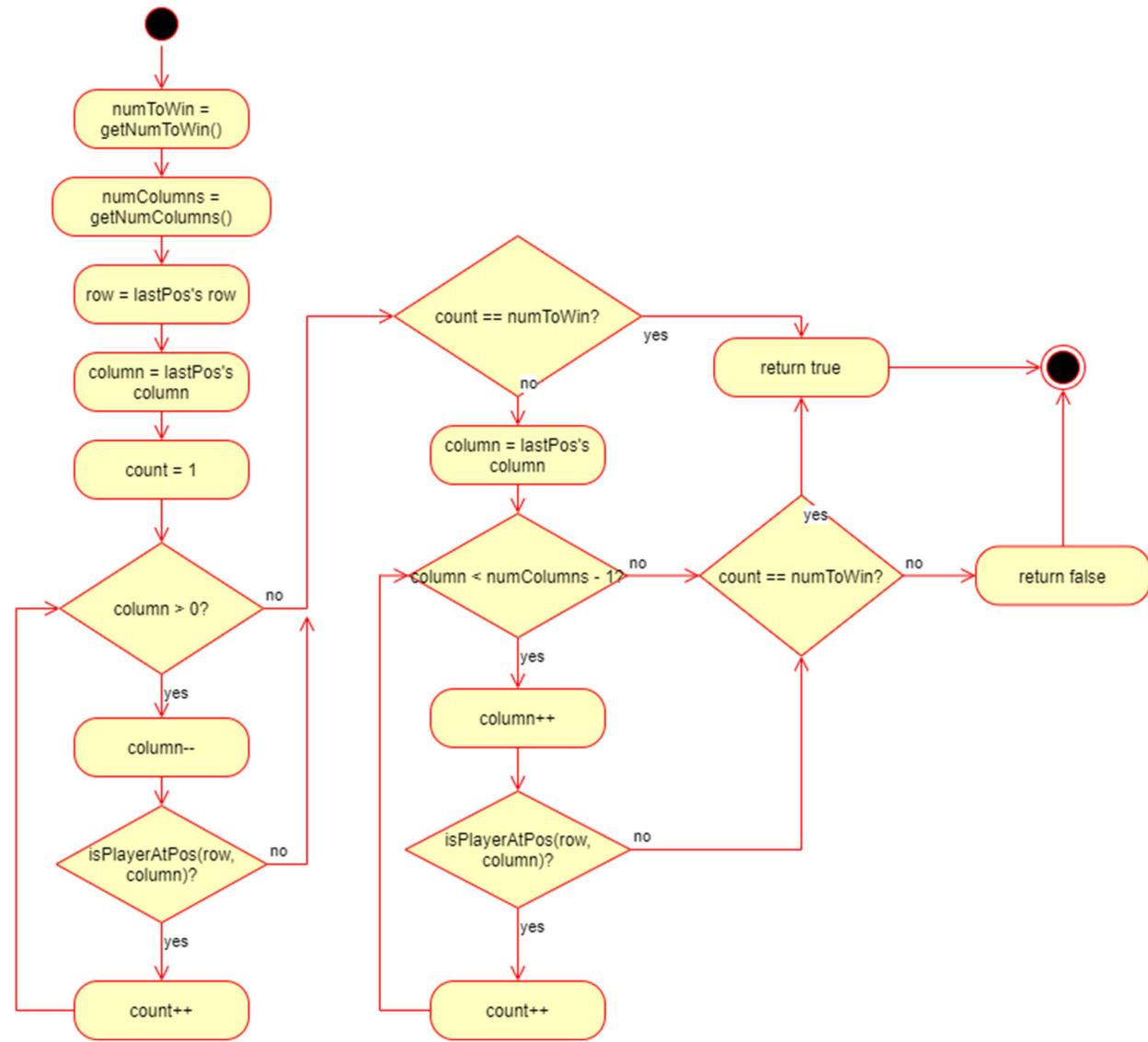
checkForWinner



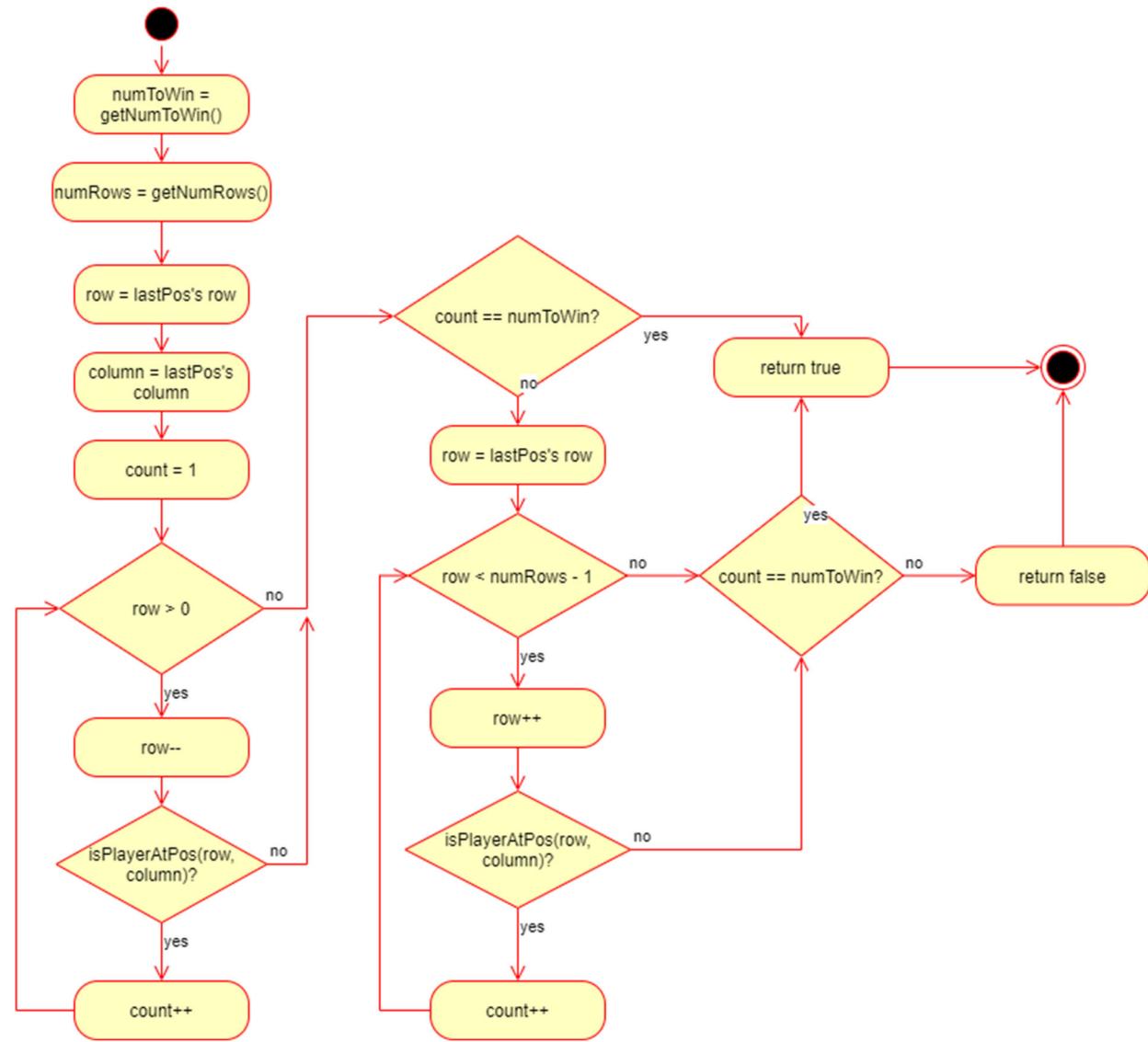
checkForDraw



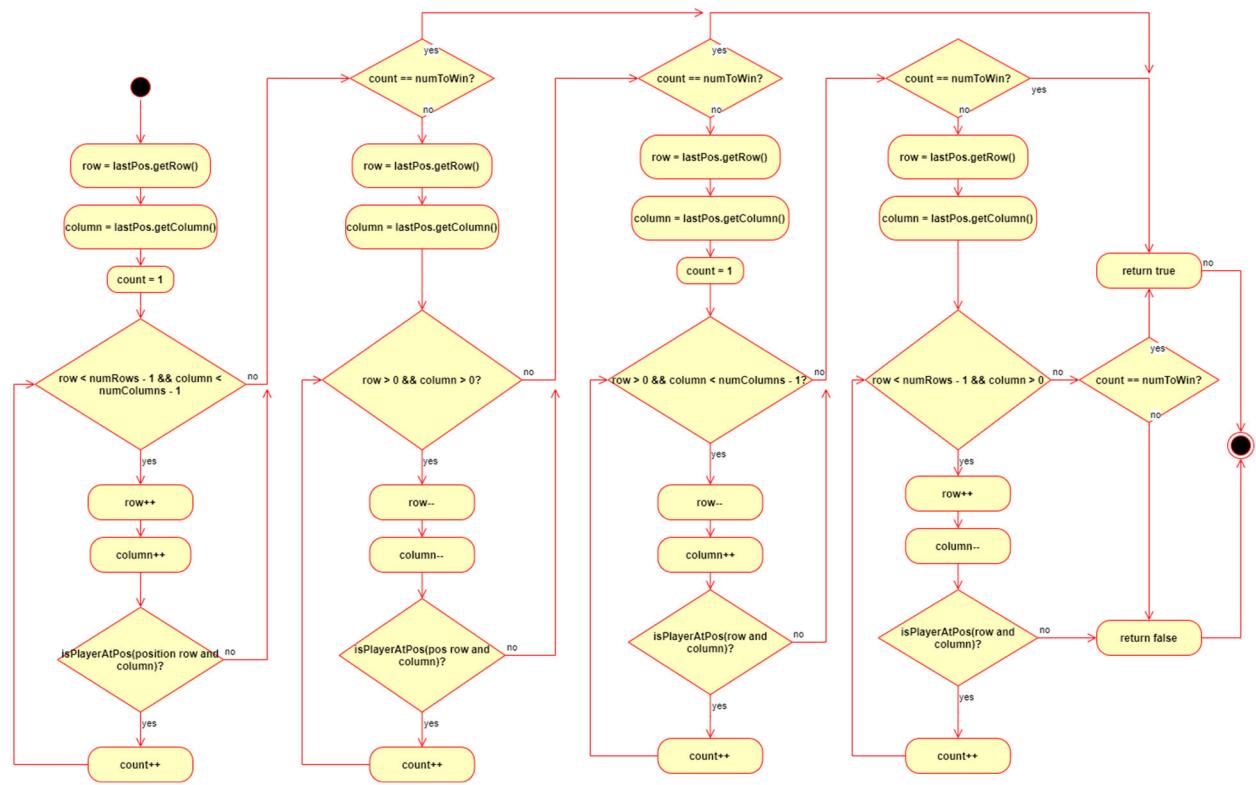
checkHorizontalWin



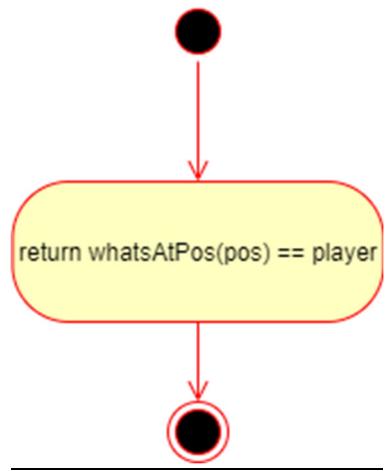
checkVerticalWin



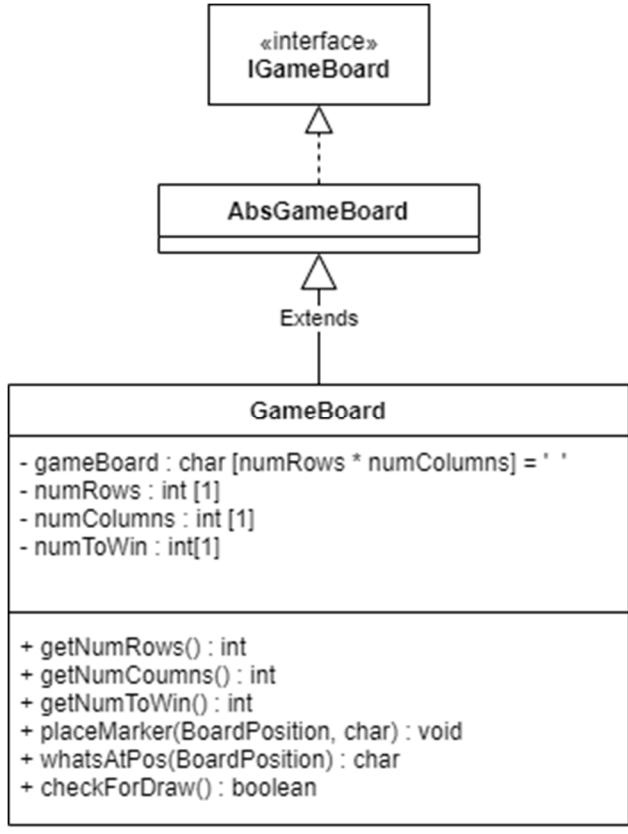
checkDiagonalWin



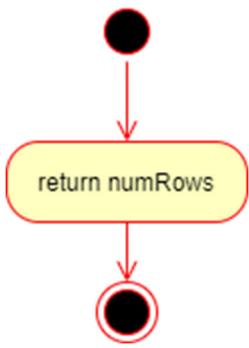
isPlayerAtPos



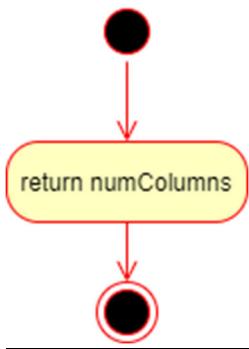
GameBoard Class



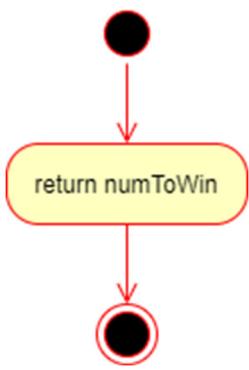
getNumRows



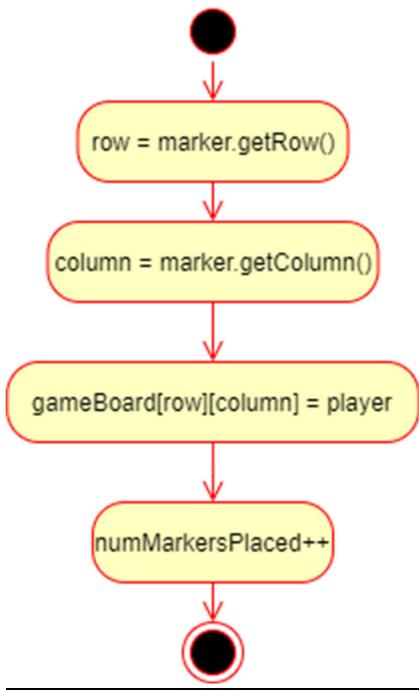
getNumColumns



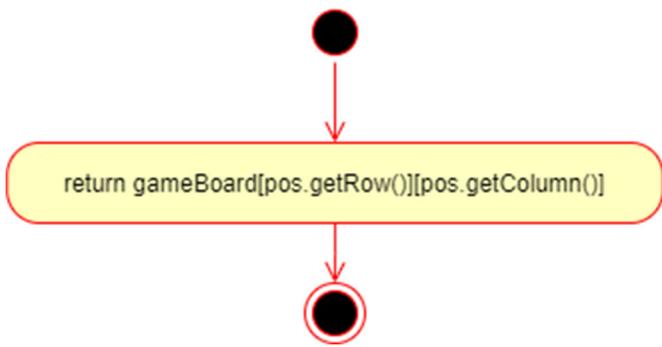
getNumToWin



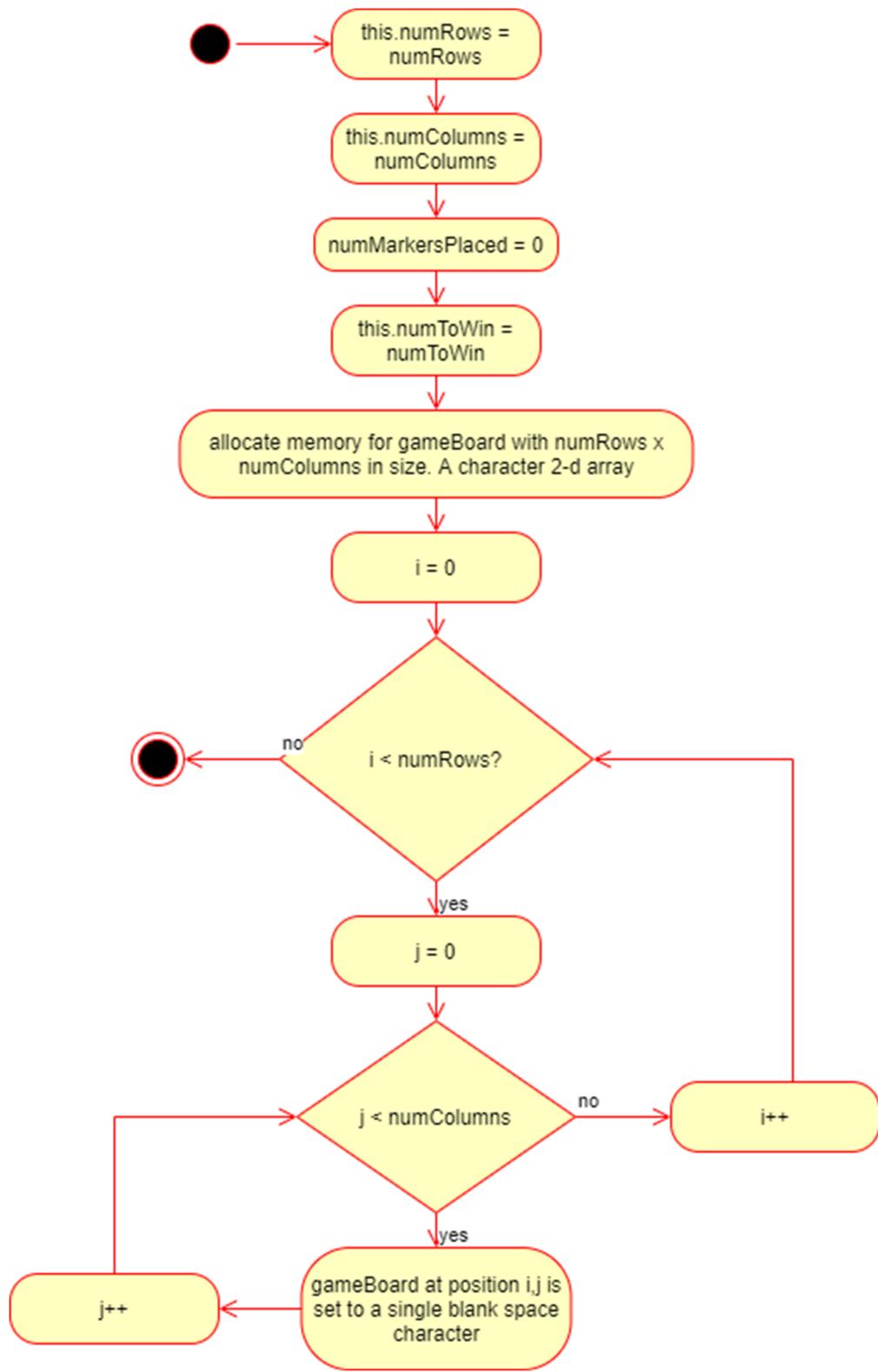
placeMarker



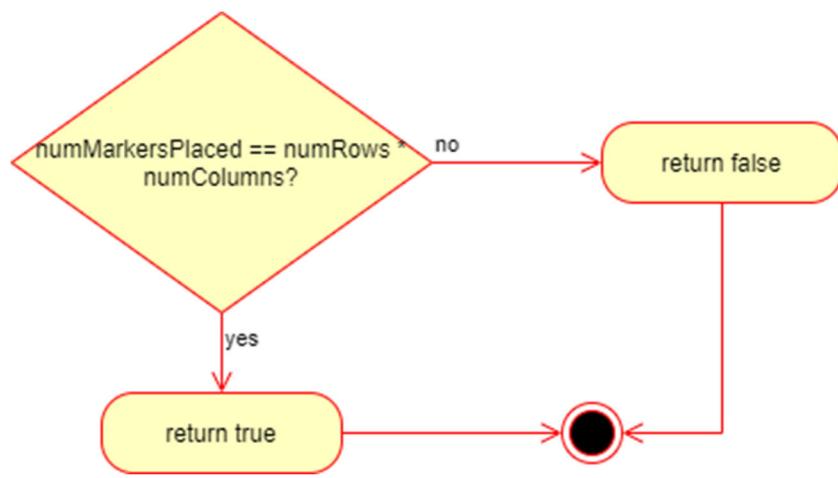
whatsAtPos



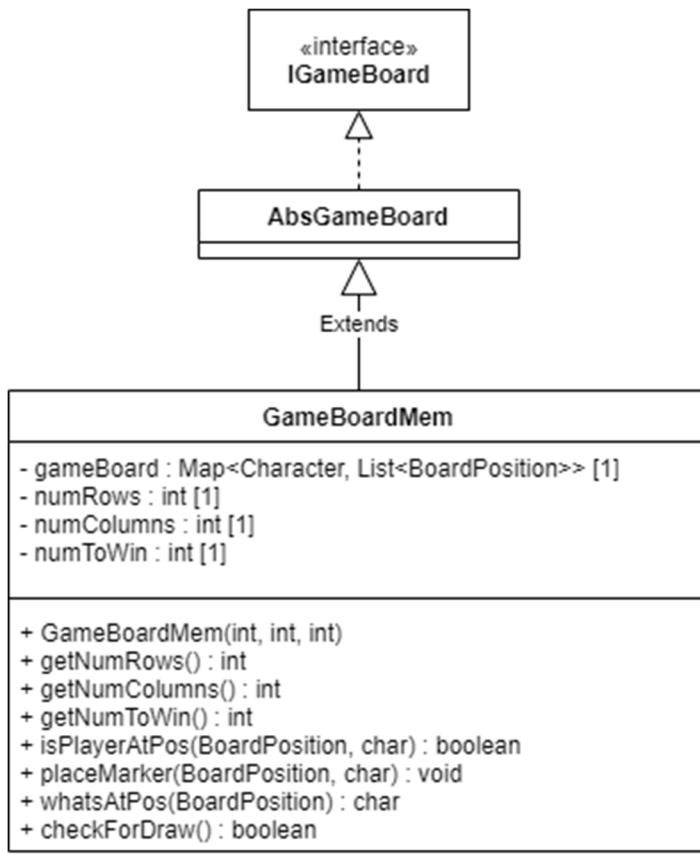
GameBoard (constructor)



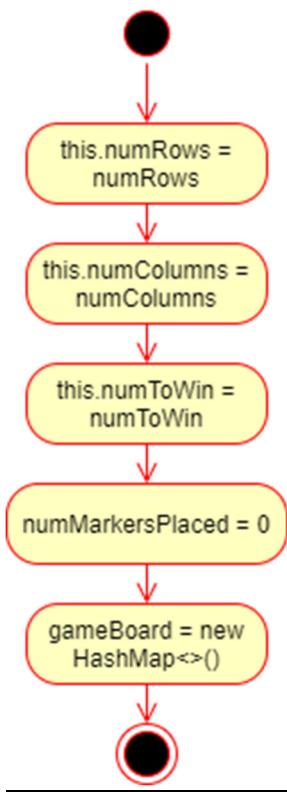
checkForDraw



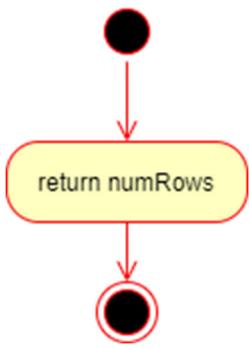
GameBoardMem Class



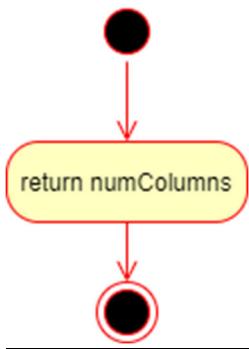
GameBoardMem (constructor)



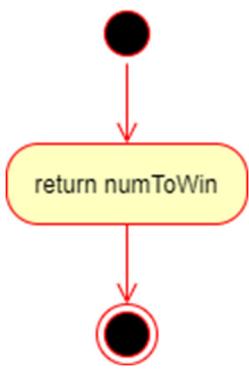
getNumRows



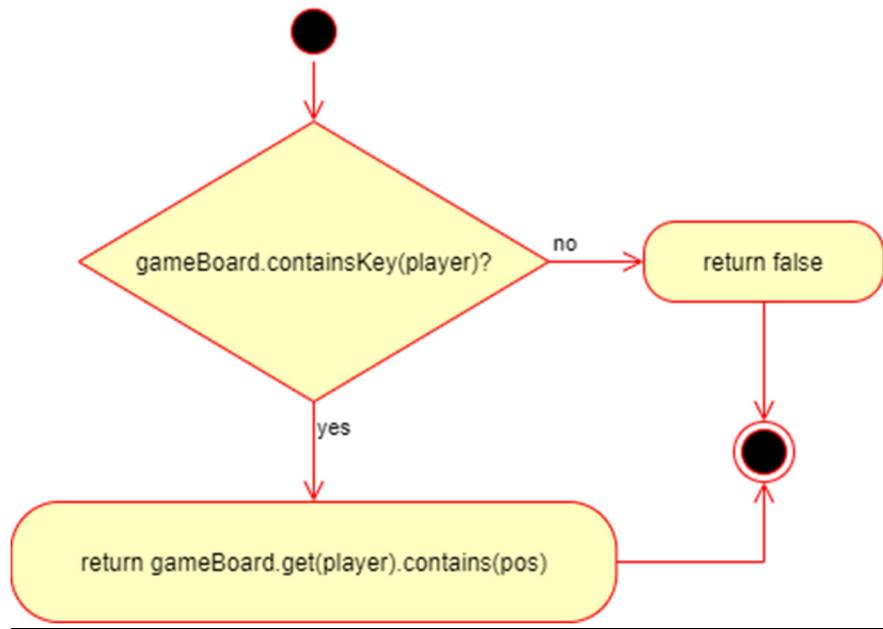
getNumColumns



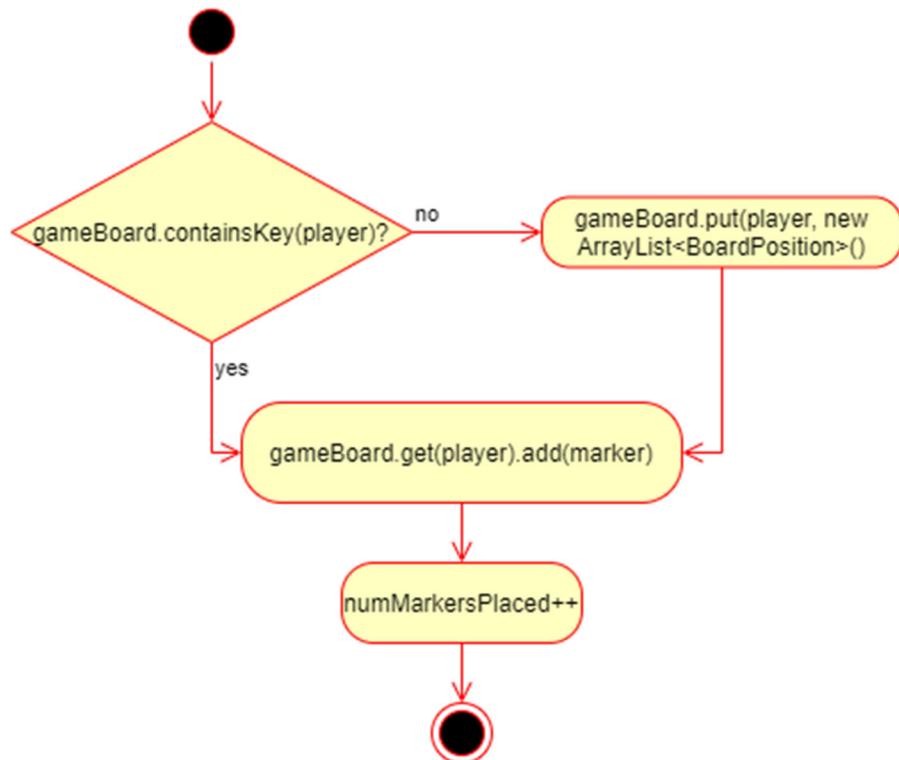
getNumToWin



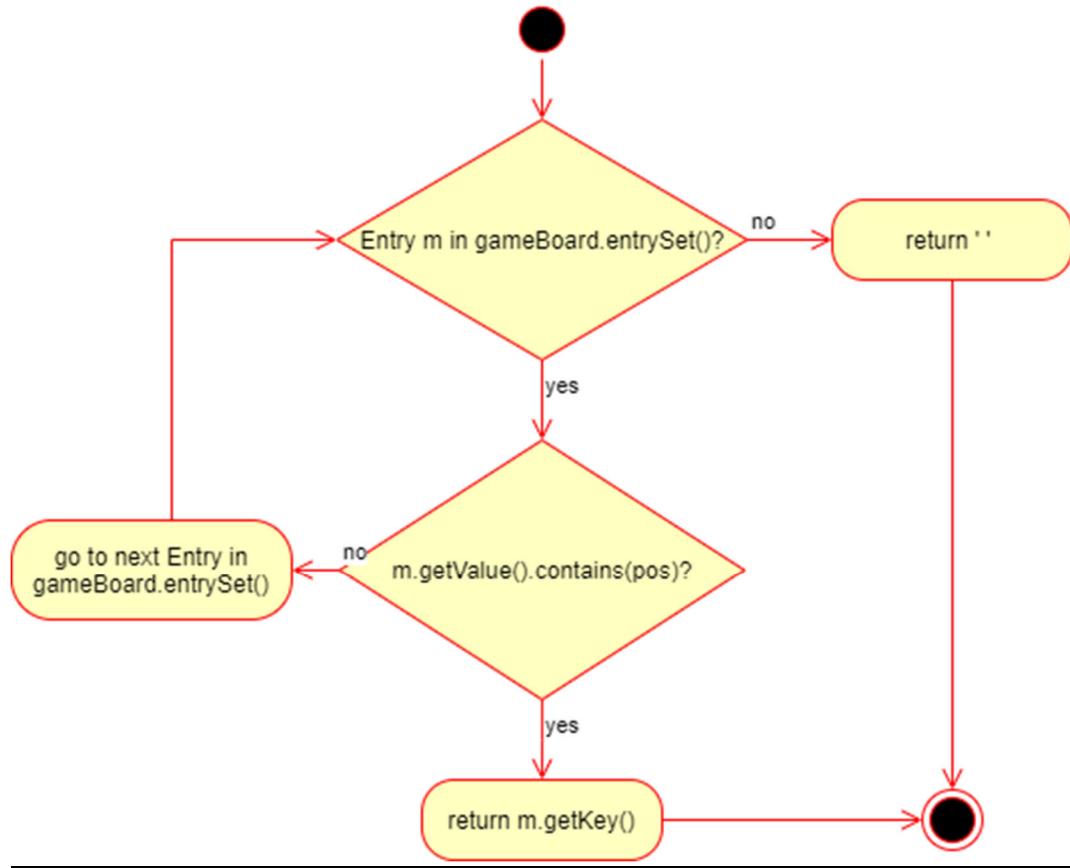
isPlayerAtPos



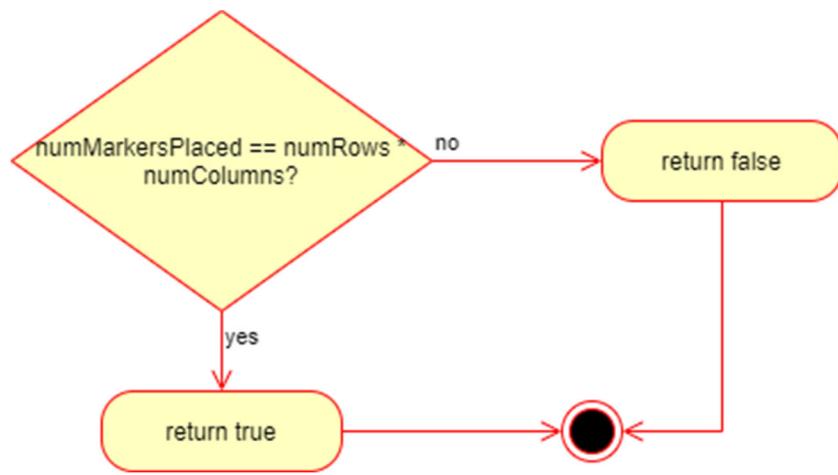
placeMarker



whatsAtPos



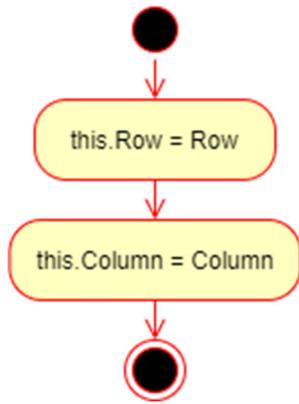
checkForDraw



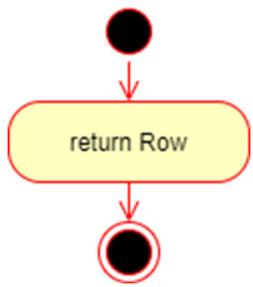
BoardPosition Class

BoardPosition
- Row : int [1] - Column : int [1]
+ BoardPosition(int, int) + getRow() : int + getColumn() : int + equals(Object) : boolean + toString() : string

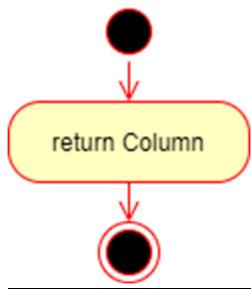
BoardPosition (constructor)



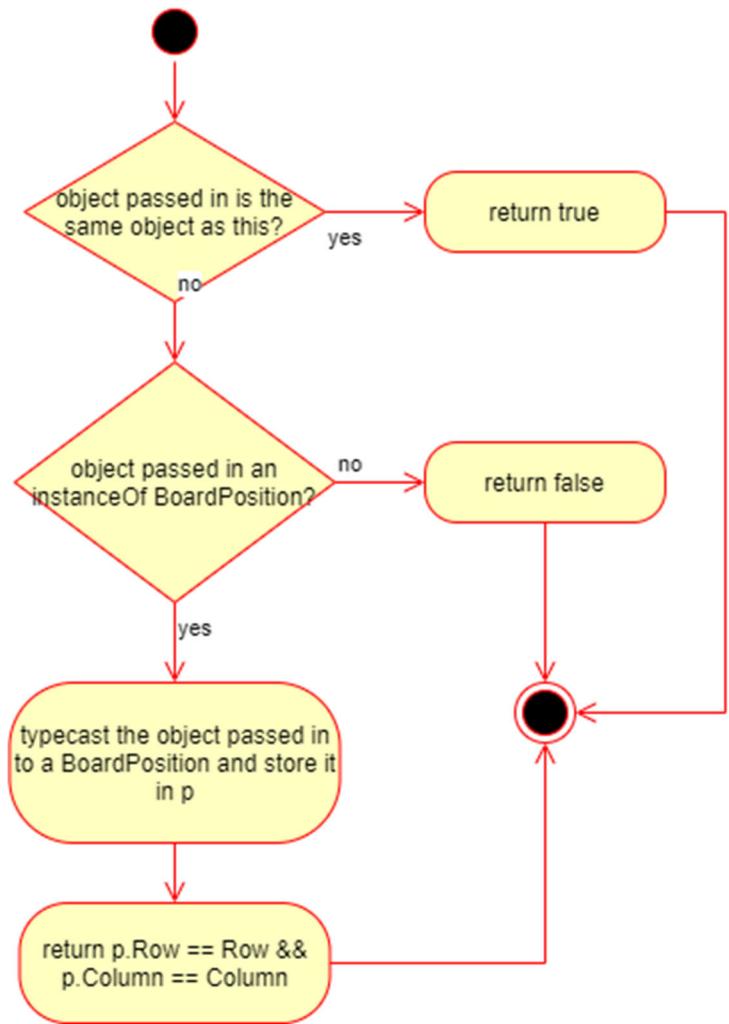
getRow



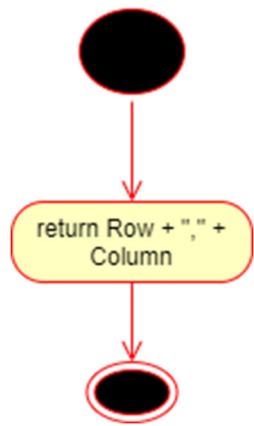
getColumn



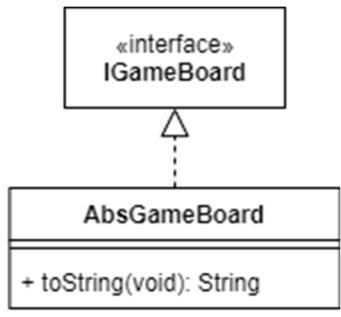
Equals



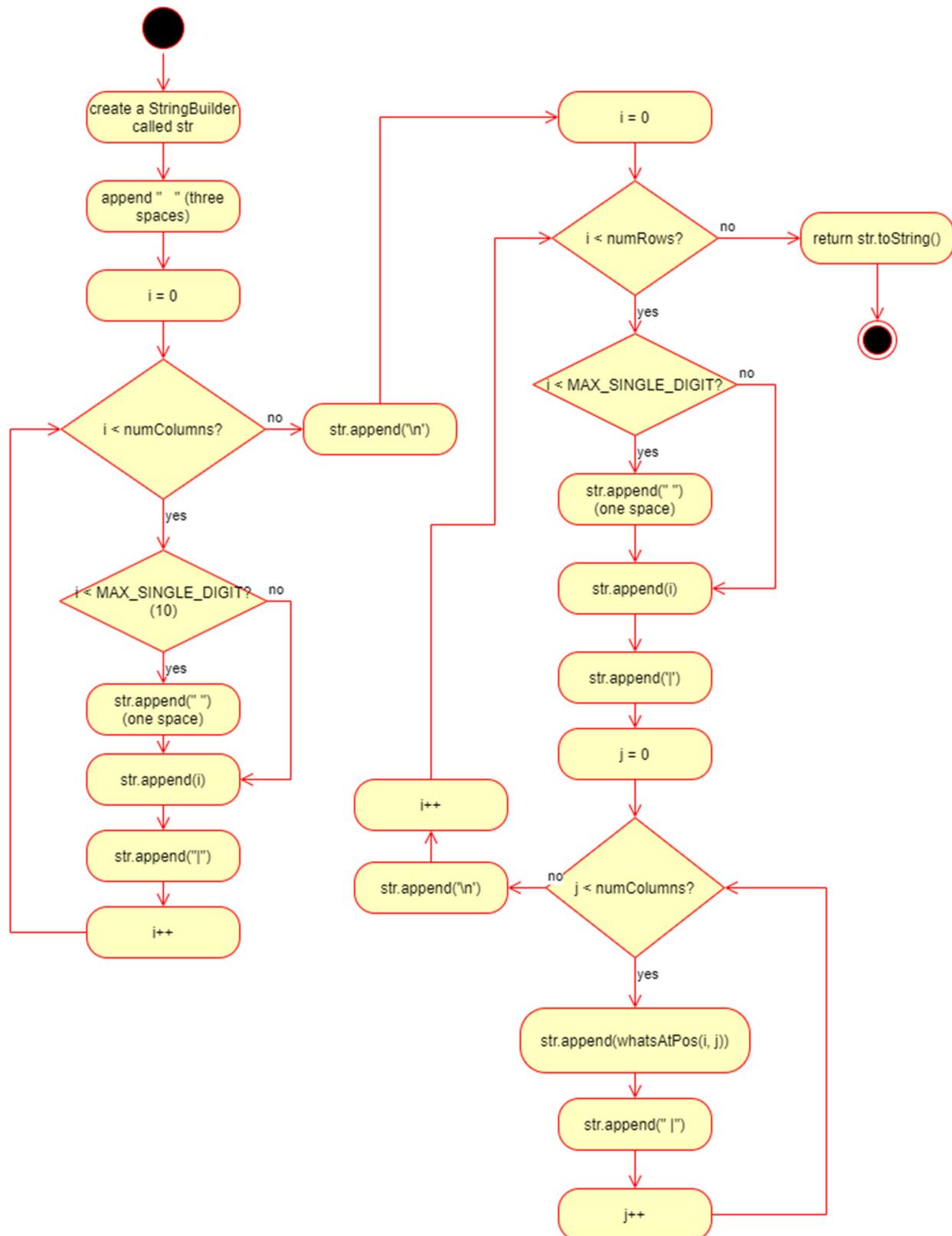
toString



AbsGameBoard Class



toString



DEPLOYMENT

Makefile is in the src directory and outside of the package cpsc2150.extendedTicTacToe

Makefile has the following targets:

- default: compiles your code. Runs with the **make** command
- run: runs your code. Runs with the **make run** command
- test: compile your test cases. Runs with **make test** command
- testGB: run test cases for GameBoard class. Runs with **make testGB** command
- testGBmem: run test cases for GameBoardMem class. Runs with **make testGBmem** command.
- clean: removes your compiled (.class) files. Runs with the **make clean** command

TESTING

constructor(int numRows, int numColumns, int numToWin)

Input: numRows = 5 numColumns = 8 numToWin = 5	Output: State: numRows = 5 numColumns = 8 numToWin = 5	Reason: This is a routine test case to see if the constructor can generate an empty 5x8 grid
Input: numRows = 3 numColumns = 3 numToWin = 3	Output: State: numRows = 3 numColumns = 3 numToWin = 3	Reason: This test case is unique and distinct because it tests to see if the constructor can generate the right grid with number of rows, number of columns, and numToWin all set to the minimum Function Name: testConstructor_boundary_min
Input: numRows = 100 numColumns = 100 numToWin = 25	Output: State: numRows = 100 numColumns = 100 numToWin = 25 game board = [100 x 100 grid of empty spaces]	Reason: This test case is unique and distinct because it tests to see if the constructor can generate the right grid with number of rows, number of columns, and numToWin all set to the maximum Function Name: testConstructor_boundary_max

boolean checkSpace(BoardPosition pos)

<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="204 454 595 601"> <tr><td></td><td>0</td><td>1</td><td>2</td></tr> <tr><td>0</td><td></td><td>X</td><td></td></tr> <tr><td>1</td><td>O</td><td></td><td></td></tr> <tr><td>2</td><td></td><td></td><td></td></tr> </table> <p>pos.getRow = 1 pos.getColumn = 1</p>		0	1	2	0		X		1	O			2				<p>Output:</p> <p>checkSpace = true state of the board is unchanged</p>	<p>Reason:</p> <p>This is a routine test case. Test on a valid (not out of bounds) and the space is not taken (blank space character).</p> <p>Function Name: testCheckSpace_ validBounds_spaceNotTaken</p>
	0	1	2															
0		X																
1	O																	
2																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="204 897 595 1045"> <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> <p>pos.getRow = 3 pos.getColumn = 3</p>		0	1	2	0				1				2				<p>Output:</p> <p>checkSpace = false state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it tests for out of bounds when the row and column are greater than the numRows and numColumns of the grid</p> <p>Function Name: testCheckSpace_outOfBounds</p>
	0	1	2															
0																		
1																		
2																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="204 1341 595 1488"> <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> <p>pos.getRow = 1 pos.getColumn = 1</p>		0	1	2	0				1		X		2				<p>Output:</p> <p>checkSpace = false state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing for valid bounds, but space is already taken</p> <p>Function Name: testCheckSpace_validBounds_ spaceTaken</p>
	0	1	2															
0																		
1		X																
2																		

boolean checkHorizontalWin(BoardPosition lastPos, char player)

<p>Input:</p> <p>State: numToWin = 4</p> <table border="1" data-bbox="208 397 592 623"> <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>X</td><td>X</td><td>X</td><td>X</td><td></td></tr> <tr><td>3</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td></tr> <tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr> </table> <p>lastPos.getRow = 2 lastPos.getColumn = 2 player = 'X'</p>		0	1	2	3	4	0						1						2	X	X	X	X		3	O	O	O	X	O	4						<p>Output:</p> <p>checkHorizontalWin = true state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because the last X was placed in the middle of the string of 4 consecutive X's as opposed to on the end, so the function needs to count X's on the right and left</p> <p>Function Name: testCheckHorizontalWin_win_lastMarkerMiddle</p>
	0	1	2	3	4																																	
0																																						
1																																						
2	X	X	X	X																																		
3	O	O	O	X	O																																	
4																																						
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 946 592 1129"> <tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr> <tr><td>0</td><td></td><td>O</td><td>O</td><td></td></tr> <tr><td>1</td><td>X</td><td>X</td><td>X</td><td></td></tr> <tr><td>2</td><td></td><td></td><td></td><td></td></tr> <tr><td>3</td><td></td><td></td><td></td><td></td></tr> </table> <p>lastPos.getRow = 1 lastPos.getColumn = 2 player = 'X'</p>		0	1	2	3	0		O	O		1	X	X	X		2					3					<p>Output:</p> <p>checkHorizontalWin = true state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because the last X was placed at the right most end, so the function needs to count X's just to the left of it</p> <p>Function Name: testCheckHorizontalWin_win_lastMarkerRightEnd</p>											
	0	1	2	3																																		
0		O	O																																			
1	X	X	X																																			
2																																						
3																																						
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 1453 592 1594"> <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> <p>lastPos.getRow = 1 lastPos.getColumn = 1 player = 'X'</p>		0	1	2	0				1		X		2				<p>Output:</p> <p>checkHorizontalWin = false state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing to see if the function will output false after not seeing any consecutive X's to the left or right</p> <p>Function Name: testCheckHorizontalWin_noWin_noLeft_noRight</p>																				
	0	1	2																																			
0																																						
1		X																																				
2																																						

<p>Input:</p> <p>State:</p> <p>numToWin = 4</p> <table border="1" data-bbox="208 340 595 566"> <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>X</td><td>X</td><td>X</td><td>O</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> <p>lastPos.getRow = 2 lastPos.getColumn = 1 player = 'X'</p>		0	1	2	3	4	0						1						2	O	X	X	X	O	3						4						<p>Ouput:</p> <p>checkHorizontalWin = false</p> <p>state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing to see if the function counts X's just to the right of it and the count would not be enough to result in a horizontal win</p> <p>Function Name: testCheckHorizontalWin_noWin_noLeft_rightNotEnough</p>
	0	1	2	3	4																																	
0																																						
1																																						
2	O	X	X	X	O																																	
3																																						
4																																						

boolean checkVerticalWin(BoardPosition lastPos, char player)

<p>Input:</p> <p>State: numToWin = 4</p> <table border="1" data-bbox="208 397 592 623"> <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>O</td><td></td></tr> <tr><td>1</td><td></td><td></td><td>X</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></td><td>X</td><td>X</td><td></td></tr> <tr><td>4</td><td></td><td></td><td></td><td>O</td><td></td></tr> </table> <p>lastPos.getRow = 2 lastPos.getColumn = 2 player = 'X'</p>		0	1	2	3	4	0			X	O		1			X	O		2			X	O		3			X	X		4				O		<p>Output: checkVerticalWin = true state of the board is unchanged</p>	<p>Reason: This test case is unique and distinct because the last X was placed in the middle of the string of 4 consecutive X's as opposed to on the end, so the function needs to count X's up and down</p> <p>Function Name: testCheckVerticalWin_win_lastMarkerMiddle</p>
	0	1	2	3	4																																	
0			X	O																																		
1			X	O																																		
2			X	O																																		
3			X	X																																		
4				O																																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 946 592 1129"> <tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr> <tr><td>0</td><td></td><td>X</td><td></td><td></td></tr> <tr><td>1</td><td></td><td>X</td><td>O</td><td></td></tr> <tr><td>2</td><td></td><td>X</td><td>O</td><td></td></tr> <tr><td>3</td><td></td><td></td><td></td><td></td></tr> </table> <p>lastPos.getRow = 2 lastPos.getColumn = 1 player = 'X'</p>		0	1	2	3	0		X			1		X	O		2		X	O		3					<p>Output: checkVerticalWin = true state of the board is unchanged</p>	<p>Reason: This test case is unique and distinct because the last X was placed at the bottom most end, so the function needs to count X's just to the top of it</p> <p>Function Name: testCheckVerticalWin_win_lastMarkerBottomEnd</p>											
	0	1	2	3																																		
0		X																																				
1		X	O																																			
2		X	O																																			
3																																						
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 1453 592 1594"> <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> <p>lastPos.getRow = 1 lastPos.getColumn = 1 player = 'X'</p>		0	1	2	0				1		X		2				<p>Output: checkVerticalWin = false state of the board is unchanged</p>	<p>Reason: This test case is unique and distinct because I am testing to see if the function will output false after not seeing any consecutive X's up or down</p> <p>Function Name: testCheckVerticalWin_noWin_noUp_noDown</p>																				
	0	1	2																																			
0																																						
1		X																																				
2																																						

Input:

State:

numToWin = 4

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

lastPos.getRow = 1
lastPos.getColumn = 2
player = 'X'

Ouput:

checkVerticalWin = false

state of the board is unchanged

Reason:

This test case is unique and distinct because I am testing to see if the function counts X's just below it and the count would not be enough to result in a vertical win

Function Name:
testCheckVerticalWin_noWin_noUp_downNotEnough

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<p>Input:</p> <p>State: numToWin = 4</p> <table border="1" data-bbox="208 397 592 618"> <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>O</td><td>O</td></tr> <tr><td>1</td><td></td><td>X</td><td></td><td></td><td>O</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></td><td></td><td>X</td><td></td></tr> <tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr> </table> <p>lastPos.getRow = 0 lastPos.getColumn = 0 player = 'X'</p>		0	1	2	3	4	0	X			O	O	1		X			O	2			X			3				X		4						<p>Output:</p> <p>checkDiagonalWin = true state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing backslash diagonal with row and column increasing (going down) where my last position placed is at the top left end so I only check in one direction (row and column increasing).</p> <p>Function Name: testCheckDiagonalWin_win_backslash_lastMarkerTopLeftEnd</p>
	0	1	2	3	4																																	
0	X			O	O																																	
1		X			O																																	
2			X																																			
3				X																																		
4																																						
<p>Input:</p> <p>State: numToWin = 4</p> <table border="1" data-bbox="208 946 592 1167"> <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>O</td><td>O</td></tr> <tr><td>1</td><td></td><td>X</td><td></td><td></td><td>O</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></td><td></td><td>X</td><td></td></tr> <tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr> </table> <p>lastPos.getRow = 3 lastPos.getColumn = 3 player = 'X'</p>		0	1	2	3	4	0	X			O	O	1		X			O	2			X			3				X		4						<p>Output:</p> <p>checkDiagonalWin = true state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing backslash diagonal. My last position placed is at the bottom right end so I will only be checking in one direction (row and column decreasing)</p> <p>Function Name: testCheckDiagonalWin_win_backslash_lastMarkerBottomLeftEnd</p>
	0	1	2	3	4																																	
0	X			O	O																																	
1		X			O																																	
2			X																																			
3				X																																		
4																																						
<p>Input:</p> <p>State: numToWin = 4</p> <table border="1" data-bbox="208 1495 592 1717"> <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>O</td><td>O</td></tr> <tr><td>1</td><td></td><td>X</td><td></td><td></td><td>O</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></td><td></td><td>X</td><td></td></tr> <tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr> </table> <p>lastPos.getRow = 1 lastPos.getColumn = 1 player = 'X'</p>		0	1	2	3	4	0	X			O	O	1		X			O	2			X			3				X		4						<p>Output:</p> <p>checkDiagonalWin = true state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing backslash diagonal where I will be testing for both directions. My last position placed is in the middle so I will have to count X's in both direction</p> <p>Function Name: testCheckDiagonalWin_win_backslash_lastMarkerMiddle</p>
	0	1	2	3	4																																	
0	X			O	O																																	
1		X			O																																	
2			X																																			
3				X																																		
4																																						

<p>Input:</p> <p>State: numToWin = 4</p> <table border="1" data-bbox="204 333 595 559"> <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></td><td></td><td></td></tr> <tr><td>1</td><td>O</td><td></td><td></td><td>X</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> <p>lastPos.getRow = 4 lastPos.getColumn = 0 player = 'X'</p>		0	1	2	3	4	0	O	O				1	O			X		2			X			3		X				4	X					<p>Output:</p> <p>checkDiagonalWin = true state of board is unchanged</p>	<p>Reason: This test case is unique and distinct because I am testing forward slash diagonal where my last position placed is at the bottom left end, and I am only checking in one direction where my row is decreasing and my column is increasing (going towards top-right)</p> <p>Function Name: testCheckDiagonalWin_win_forwardSlash_lastMarkerBottomLeftEnd</p>
	0	1	2	3	4																																	
0	O	O																																				
1	O			X																																		
2			X																																			
3		X																																				
4	X																																					
<p>Input:</p> <p>State: numToWin = 4</p> <table border="1" data-bbox="204 901 595 1127"> <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></td><td></td><td></td></tr> <tr><td>1</td><td>O</td><td></td><td></td><td>X</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> <p>lastPos.getRow = 1 lastPos.getColumn = 3 player = 'X'</p>		0	1	2	3	4	0	O	O				1	O			X		2			X			3		X				4	X					<p>Output:</p> <p>checkDiagonalWin = true state of board is unchanged</p>	<p>Reason: This test case is unique and distinct because I am testing forward slash diagonal where my last position placed is at the top right end, and I am only checking in one direction where my row is increasing and my column is decreasing (going towards bottom-left)</p> <p>Function Name: testCheckDiagonalWin_win_forwardSlash_lastMarkerTopRightEnd</p>
	0	1	2	3	4																																	
0	O	O																																				
1	O			X																																		
2			X																																			
3		X																																				
4	X																																					
<p>Input:</p> <p>State: numToWin = 4</p> <table border="1" data-bbox="204 1461 595 1687"> <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></td><td></td><td></td></tr> <tr><td>1</td><td>O</td><td></td><td></td><td>X</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> <p>lastPos.getRow = 2 lastPos.getColumn = 2 player = 'X'</p>		0	1	2	3	4	0	O	O				1	O			X		2			X			3		X				4	X					<p>Output:</p> <p>checkDiagonalWin = true state of board is unchanged</p>	<p>Reason: This test case is unique and distinct because I am testing forward slash diagonal where I will be testing for both directions. My last position placed is in the middle so I will have to count X's in both direction</p> <p>Function Name: testCheckDiagonalWin_win_forwardSlash_lastMarkerMiddle</p>
	0	1	2	3	4																																	
0	O	O																																				
1	O			X																																		
2			X																																			
3		X																																				
4	X																																					

<p>Input:</p> <p>State:</p> <p>numToWin = 4</p> <table border="1" data-bbox="208 340 595 566"> <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>X</td></tr> <tr><td>1</td><td></td><td>X</td><td></td><td>X</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></td><td></td><td>O</td><td>O</td></tr> <tr><td>4</td><td></td><td></td><td></td><td>O</td><td>O</td></tr> </table> <p>lastPos.getRow = 2 lastPos.getColumn = 2 player = 'X'</p>		0	1	2	3	4	0	X				X	1		X		X		2			X			3				O	O	4				O	O	<p>Output:</p> <p>checkDiagonalWin = false</p> <p>state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing backslash diagonal but it fails, and then attempting forward slash diagonal and that fails too. So I'm checking both diagonals</p> <p>Function Name: testCheckDiagonalWin_noWin_check_both_diagonals</p>
	0	1	2	3	4																																	
0	X				X																																	
1		X		X																																		
2			X																																			
3				O	O																																	
4				O	O																																	

boolean checkForDraw()

<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 397 595 544"> <tr><td></td><td>0</td><td>1</td><td>2</td></tr> <tr><td>0</td><td>X</td><td>O</td><td>O</td></tr> <tr><td>1</td><td>O</td><td>X</td><td>X</td></tr> <tr><td>2</td><td>X</td><td>X</td><td>O</td></tr> </table>		0	1	2	0	X	O	O	1	O	X	X	2	X	X	O	<p>Output:</p> <p>checkForDraw = true state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing to see if a full grid would result in a draw</p> <p>Function Name: testCheckForDraw_draw_fullGrid</p>
	0	1	2															
0	X	O	O															
1	O	X	X															
2	X	X	O															
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 745 595 893"> <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>		0	1	2	0				1		X		2				<p>Output:</p> <p>checkForDraw = false state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing for boundary cases where there is only one marker on grid</p> <p>Function Name: testCheckForDraw_noDraw_boundary_oneMarker</p>
	0	1	2															
0																		
1		X																
2																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 1115 595 1284"> <tr><td></td><td>0</td><td>1</td><td>2</td></tr> <tr><td>0</td><td>O</td><td>X</td><td>O</td></tr> <tr><td>1</td><td>O</td><td>X</td><td>X</td></tr> <tr><td>2</td><td></td><td>O</td><td>X</td></tr> </table>		0	1	2	0	O	X	O	1	O	X	X	2		O	X	<p>Output:</p> <p>checkForDraw = false state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because I am testing for the boundary case where the grid is almost full. Only one position empty.</p> <p>Function Name: testCheckForDraw_noDraw_boundary_almostFull</p>
	0	1	2															
0	O	X	O															
1	O	X	X															
2		O	X															
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 1505 595 1653"> <tr><td></td><td>0</td><td>1</td><td>2</td></tr> <tr><td>0</td><td>X</td><td>O</td><td></td></tr> <tr><td>1</td><td>O</td><td></td><td></td></tr> <tr><td>2</td><td></td><td>X</td><td>X</td></tr> </table>		0	1	2	0	X	O		1	O			2		X	X	<p>Output:</p> <p>checkForDraw = false state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it is a routine test case where it is in the middle of almost full and almost empty.</p> <p>Function Name: testCheckForDraw_noDraw_routine</p>
	0	1	2															
0	X	O																
1	O																	
2		X	X															

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

<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 397 592 544"> <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>X</td><td></td><td></td></tr> <tr><td>2</td><td></td><td></td><td></td></tr> </table> <p>pos.getRow = 1 pos getColumn = 0</p>		0	1	2	0				1	X			2				<p>Output:</p> <p>whatsAtPos = 'X'</p> <p>state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it checks boundary case where column is 0 (minimum)</p> <p>Function Name:</p> <p>testWhatsAtPos_boundary_column_min</p>
	0	1	2															
0																		
1	X																	
2																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 840 592 988"> <tr><td></td><td>0</td><td>1</td><td>2</td></tr> <tr><td>0</td><td></td><td>X</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> <p>pos.getRow = 0 pos getColumn = 1</p>		0	1	2	0		X		1				2				<p>Output:</p> <p>whatsAtPos = 'X'</p> <p>state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it checks boundary case where row is 0 (minimum)</p> <p>Function Name:</p> <p>testWhatsAtPos_boundary_row_min</p>
	0	1	2															
0		X																
1																		
2																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 1284 592 1431"> <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>X</td></tr> <tr><td>2</td><td></td><td></td><td></td></tr> </table> <p>pos.getRow = 1 pos getColumn = 2</p>		0	1	2	0				1			X	2				<p>Output:</p> <p>whatsAtPos = 'X'</p> <p>state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it checks boundary case where column is at the max</p> <p>Function Name:</p> <p>testWhatsAtPos_boundary_column_max</p>
	0	1	2															
0																		
1			X															
2																		

<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 340 589 487"> <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>X</td><td></td></tr> </table> <p>pos.getRow = 1 pos.getColumn = 2</p>		0	1	2	0				1				2		X		<p>Output:</p> <p>whatsAtPos = 'X'</p> <p>state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it checks boundary case where the row is at the max</p> <p>Function Name:WE testWhatsAtPos_boundary_row_max</p>
	0	1	2															
0																		
1																		
2		X																
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 768 589 916"> <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> <p>pos.getRow = 1 pos.getColumn = 1</p>		0	1	2	0				1		X		2				<p>Output:</p> <p>whatsAtPos = 'X'</p> <p>state of board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because this is a routine test case where it is not in any of the boundaries</p> <p>Function Name: testWhatsAtPos_routine</p>
	0	1	2															
0																		
1		X																
2																		

boolean isPlayerAtPos(BoardPosition pos, char Player)

<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 397 592 544"> <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> <p>Player = 'O' pos.getRow = 1 pos.getColumn = 1</p>		0	1	2	0				1		X		2				<p>Output:</p> <p>isPlayerAtPos = false</p> <p>state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it is routine and tests to see if the function would return false when the player is not at pos</p> <p>Function Name: testIsPlayerAtPos_false_routine</p>
	0	1	2															
0																		
1		X																
2																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 882 592 1030"> <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>X</td><td></td><td></td></tr> <tr><td>2</td><td></td><td></td><td></td></tr> </table> <p>Player = 'X' pos.getRow = 1 pos.getColumn = 0</p>		0	1	2	0				1	X			2				<p>Output:</p> <p>isPlayerAtPos = true</p> <p>state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it tests a boundary case where column is at min (0)</p> <p>Function Name: testIsPlayerAtPos_true_boundary_column_min</p>
	0	1	2															
0																		
1	X																	
2																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 1347 592 1495"> <tr><td></td><td>0</td><td>1</td><td>2</td></tr> <tr><td>0</td><td></td><td>X</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> <p>Player = 'X' pos.getRow = 0 pos.getColumn = 1</p>		0	1	2	0		X		1				2				<p>Output:</p> <p>isPlayerAtPos = true</p> <p>state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it tests a boundary case where row is at min (0)</p> <p>Function Name: testIsPlayerAtPos_true_boundary_row_min</p>
	0	1	2															
0		X																
1																		
2																		

<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 340 595 487"> <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>X</td></tr> <tr><td>2</td><td></td><td></td><td></td></tr> </table> <p>Player = 'X' pos.getRow = 1 pos.getColumn = 2</p>		0	1	2	0				1			X	2				<p>Output:</p> <p>isPlayerAtPos = true state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it tests a boundary case where column is at max</p> <p>Function Name: testIsPlayerAtPos_true_boundary_column_max</p>
	0	1	2															
0																		
1			X															
2																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 811 595 958"> <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>X</td><td></td></tr> </table> <p>Player = 'X' pos.getRow = 1 pos.getColumn = 2</p>		0	1	2	0				1				2		X		<p>Output:</p> <p>isPlayerAtPos = true state of the board is unchanged</p>	<p>Reason:</p> <p>This test case is unique and distinct because it tests a boundary case where row is at max</p> <p>Function Name: testIsPlayerAtPos_true_boundary_row_max</p>
	0	1	2															
0																		
1																		
2		X																

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

<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 397 592 544"> <tr><td></td><td>0</td><td>1</td><td>2</td></tr> <tr><td>0</td><td></td><td>X</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> <p>marker.getRow = 1 marker.getColumn = 1 player = 'A'</p>		0	1	2	0		X		1				2				<p>Output:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="625 397 1008 544"> <tr><td></td><td>0</td><td>1</td><td>2</td></tr> <tr><td>0</td><td></td><td>X</td><td></td></tr> <tr><td>1</td><td>O</td><td>A</td><td></td></tr> <tr><td>2</td><td></td><td></td><td></td></tr> </table>		0	1	2	0		X		1	O	A		2				<p>Reason: This test case is unique and distinct because I am placing a marker representing a player who has not been placed on this board before</p> <p>Function Name: testPlaceMarker_routine_new_player</p>
	0	1	2																															
0		X																																
1																																		
2																																		
	0	1	2																															
0		X																																
1	O	A																																
2																																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 872 592 1020"> <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> <p>marker.getRow = 0 marker.getColumn = 1 player = 'X'</p>		0	1	2	0				1				2				<p>Output:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="625 872 1008 1020"> <tr><td></td><td>0</td><td>1</td><td>2</td></tr> <tr><td>0</td><td></td><td>X</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>		0	1	2	0		X		1				2				<p>Reason: This test case is unique and distinct because I am placing a marker at a boundary case where row is at min</p> <p>Function Name: testPlaceMarker_boundary_row_min</p>
	0	1	2																															
0																																		
1																																		
2																																		
	0	1	2																															
0		X																																
1																																		
2																																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 1345 592 1493"> <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> <p>marker.getRow = 1 marker.getColumn = 2 player = 'X'</p>		0	1	2	0				1				2				<p>Output:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="625 1345 1008 1493"> <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>X</td></tr> <tr><td>2</td><td></td><td></td><td></td></tr> </table>		0	1	2	0				1			X	2				<p>Reason: This test case is unique and distinct because I am placing a marker at a boundary case where column is at max</p> <p>Function name: testPlaceMarker_boundary_column_max</p>
	0	1	2																															
0																																		
1																																		
2																																		
	0	1	2																															
0																																		
1			X																															
2																																		

<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 804 592 952"> <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> <p>marker.getRow = 2 marker.getColumn = 1 player = 'X'</p>		0	1	2	0				1				2				<p>Output:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="625 804 1008 952"> <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>X</td><td></td><td></td></tr> <tr><td>2</td><td></td><td></td><td></td></tr> </table>		0	1	2	0				1	X			2				<p>Reason:</p> <p>This test case is unique and distinct because I am placing a marker at a boundary case where row is at max</p> <p>Function name: testPlaceMarker_boundary_row_max</p>
	0	1	2																															
0																																		
1																																		
2																																		
	0	1	2																															
0																																		
1	X																																	
2																																		
<p>Input:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="208 804 592 952"> <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> <p>marker.getRow = 1 marker.getColumn = 0 player = 'X'</p>		0	1	2	0				1				2				<p>Output:</p> <p>State: numToWin = 3</p> <table border="1" data-bbox="625 804 1008 952"> <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>X</td><td></td><td></td></tr> <tr><td>2</td><td></td><td></td><td></td></tr> </table>		0	1	2	0				1	X			2				<p>Reason:</p> <p>This test case is unique and distinct because I am placing a marker at a boundary case where column is at min</p> <p>Function name: testPlaceMarker_boundary_column_min</p>
	0	1	2																															
0																																		
1																																		
2																																		
	0	1	2																															
0																																		
1	X																																	
2																																		