Ashlyn Cooper
CPSC 2150
Project 5
Project Report

Requirements Analysis

Functional Requirements

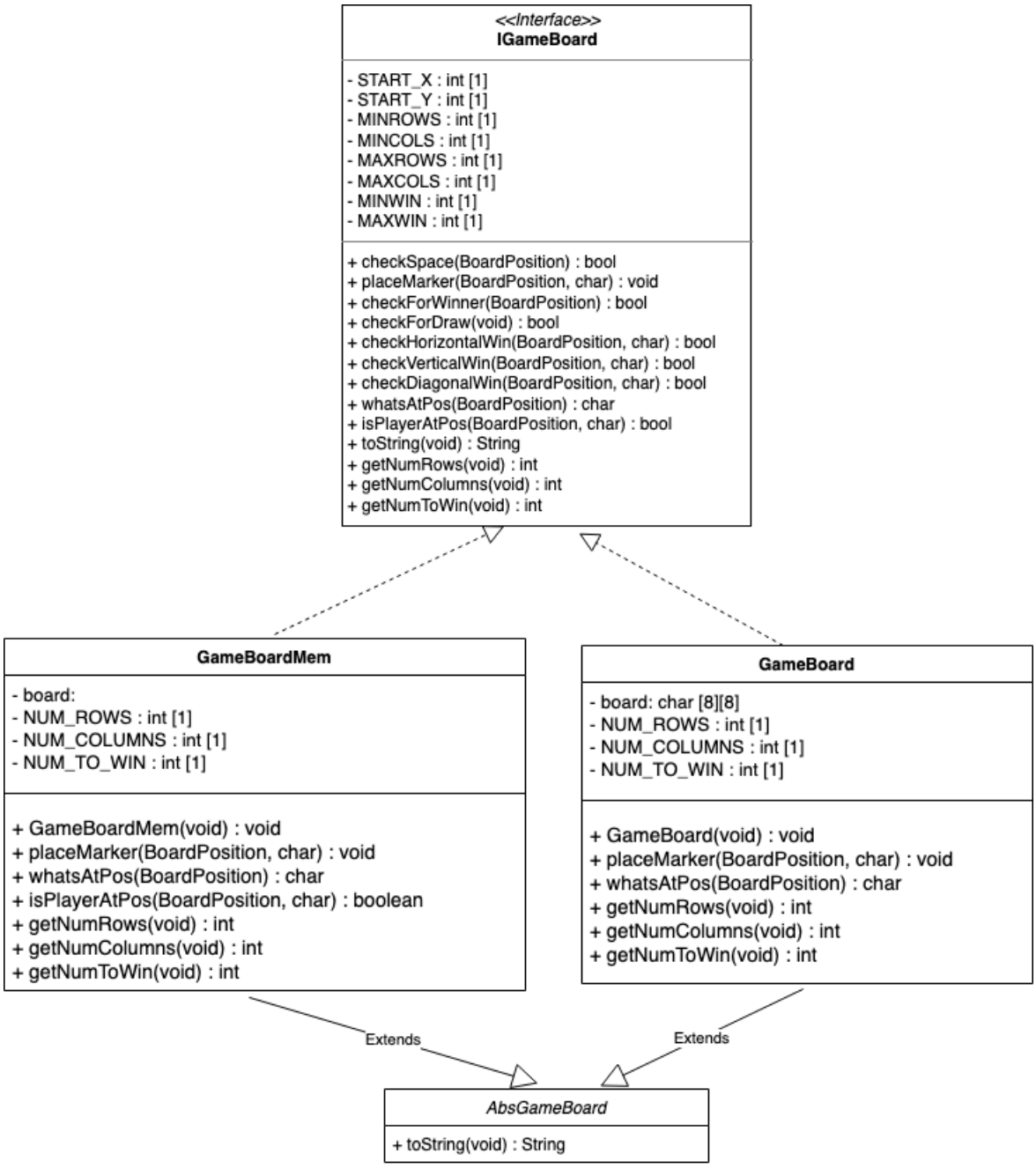
1. As a player, I can click a position so that I can play my token.
2. As a player, I can choose to play again so that I can replay tic tac toe.
3. As a player, I can have my input validated by the program so that I can place my token in a valid position.
4. As a player, I can view output prompting for an input so that I can see when I should make an input.
5. As a player, I can view an outputted message at the end of the game, so that I know how the game ended.
6. As a player, I can win horizontally, as that I can win the game.
7. As a player, I can win vertically, so that I can win the game.
8. As a player, I can win diagonally so that I can win the game.
9. As a player, I can make a move after my opponent (if they have not one), so that I can take my turn.
10. The program can end the game without either player winning, so that the game can end in a tie.
11. As a user, I can input the number of columns and number of rows, so that I can set the size of the gameboard.
12. As a user, I can input the number in a row needed to win, so that the game is played with that rule.
13. As a user, I can input the number of players, so that the game can be played with that number of players.

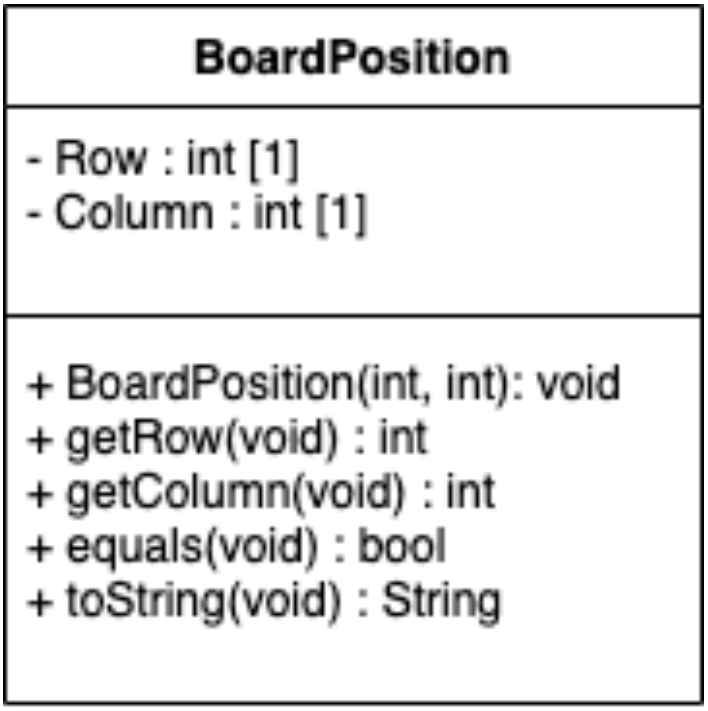
Non-functional Requirements

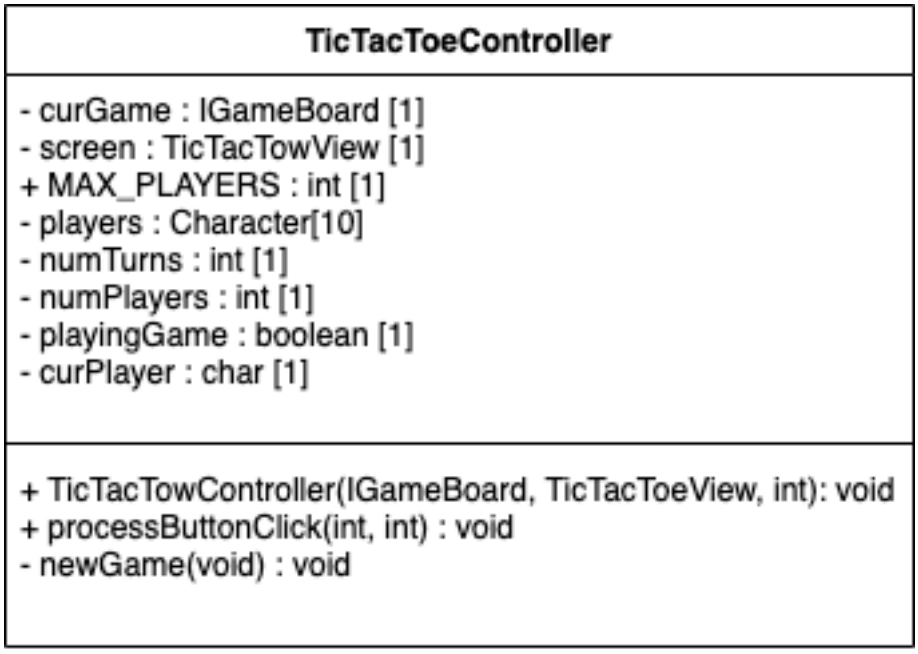
1. The system must be written in Java.
2. The program can display the game board to the screen so that the player can see the board.
3. The program can alternate between players so that the program can be played by at least 2 players and up to 10.
4. Player 1 will always go first.
5. The program can be repeated so that the program has the ability to let the players play again.
6. The game board is of size NUM_ROWS x NUM_COLUMNS as indicated by user input.
7. Coordinate (0,0) represents the top left corner of the game board.

Design

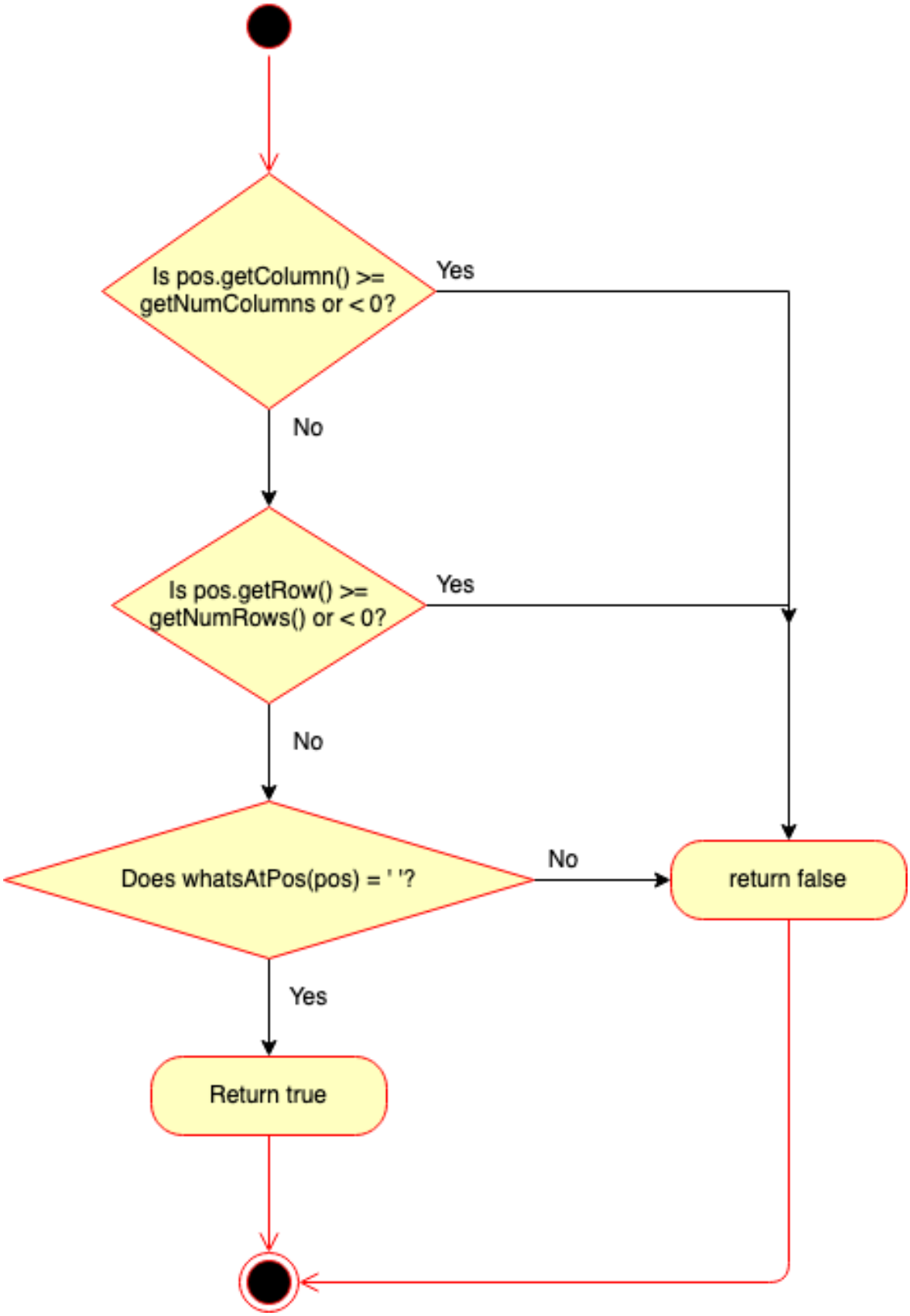
Class Diagrams



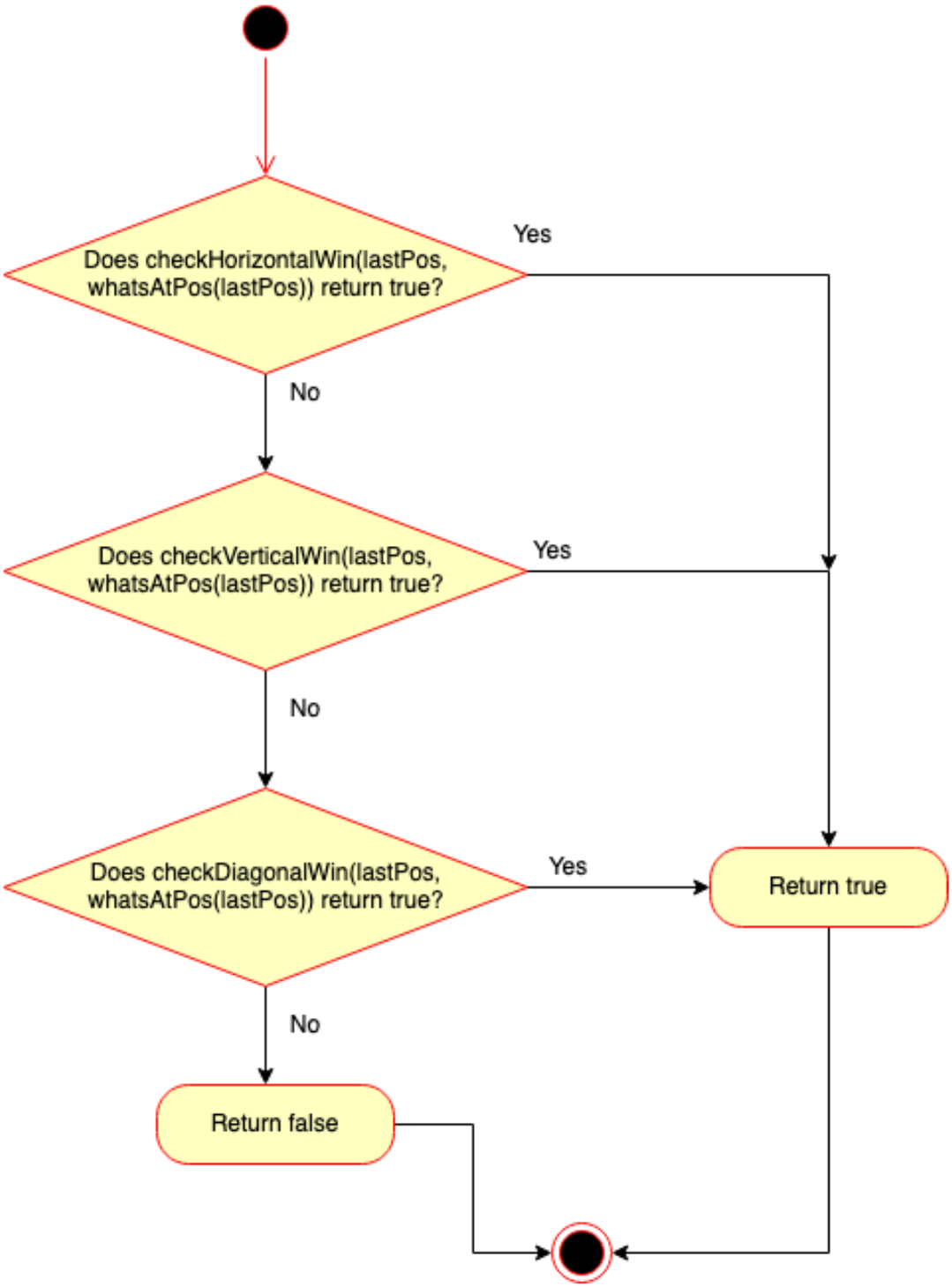

BoardPosition
- Row : int [1]
- Column : int [1]
+ BoardPosition(int, int): void
+ getRow(void) : int
+ getColumn(void) : int
+ equals(void) : bool
+ toString(void) : String


TicTacToeController
- curGame : IGameBoard [1]
- screen : TicTacTowView [1]
+ MAX_PLAYERS : int [1]
- players : Character[10]
- numTurns : int [1]
- numPlayers : int [1]
- playingGame : boolean [1]
- curPlayer : char [1]
+ TicTacTowController(IGameBoard, TicTacToeView, int): void
+ processButtonClick(int, int) : void
- newGame(void) : void

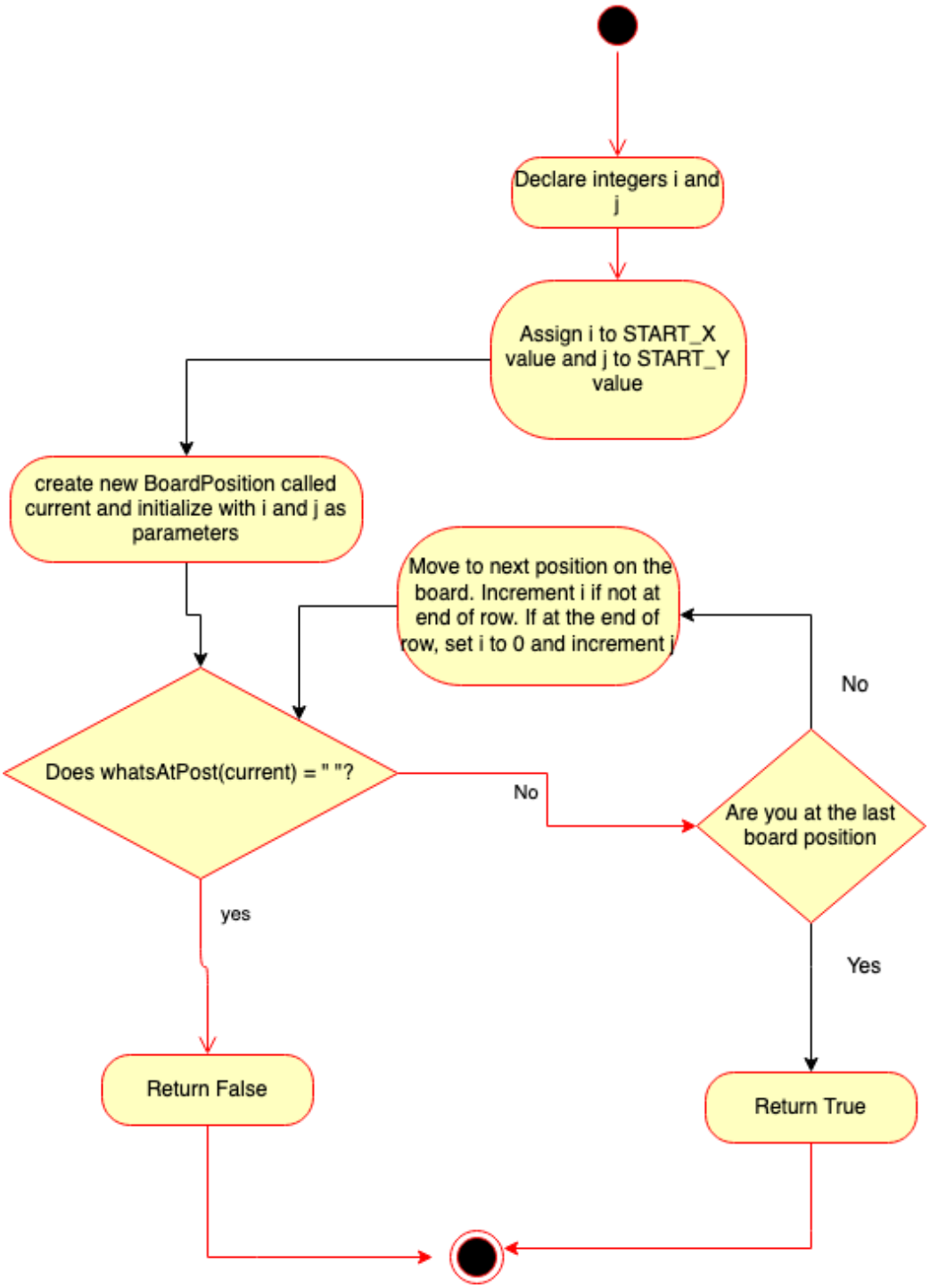
Activity Diagrams
IGameBoard (default methods)
checkSpace(BoardPosition pos)



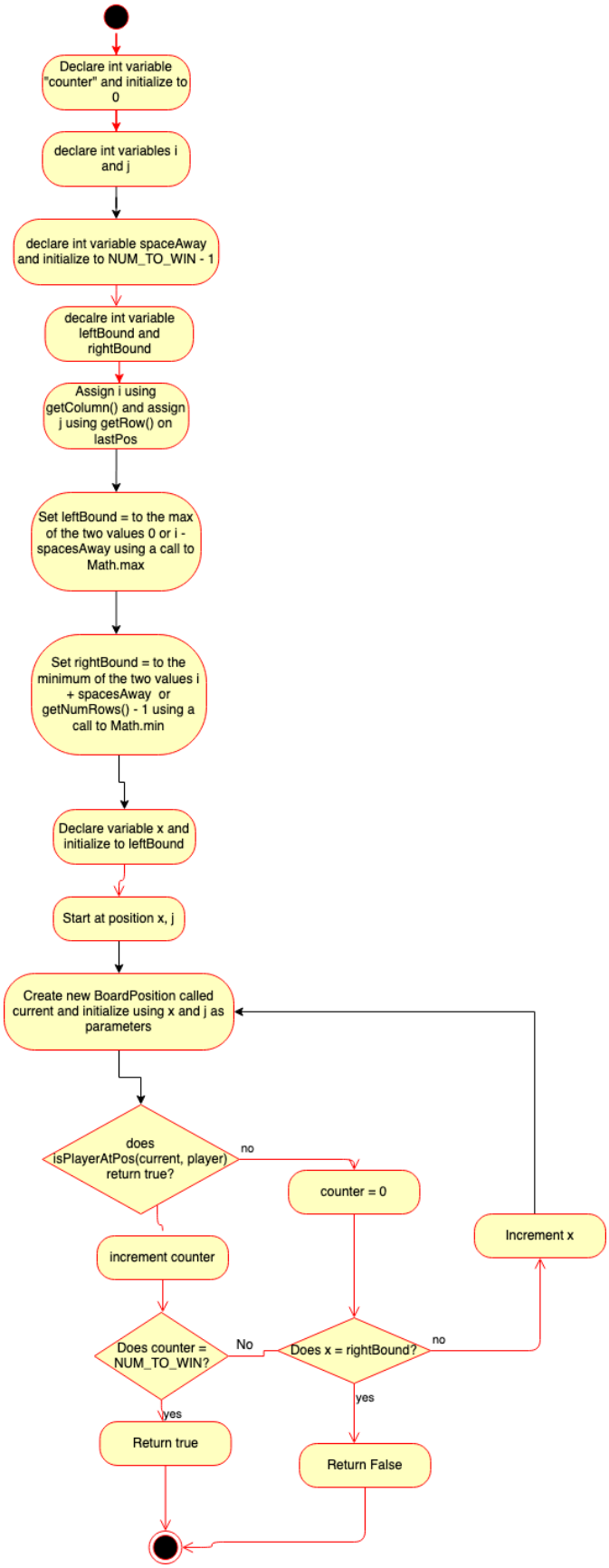
checkForWinner(BoardPosition lastPos)



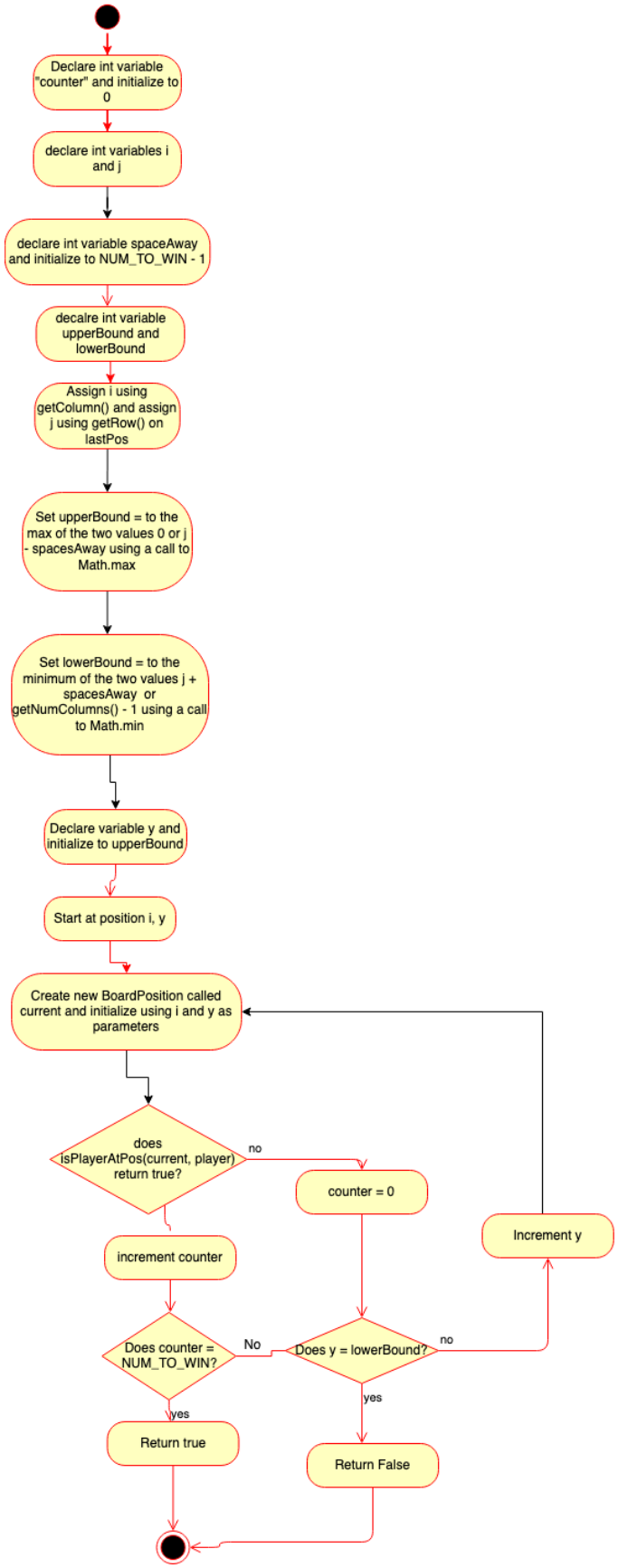
checkForDraw()



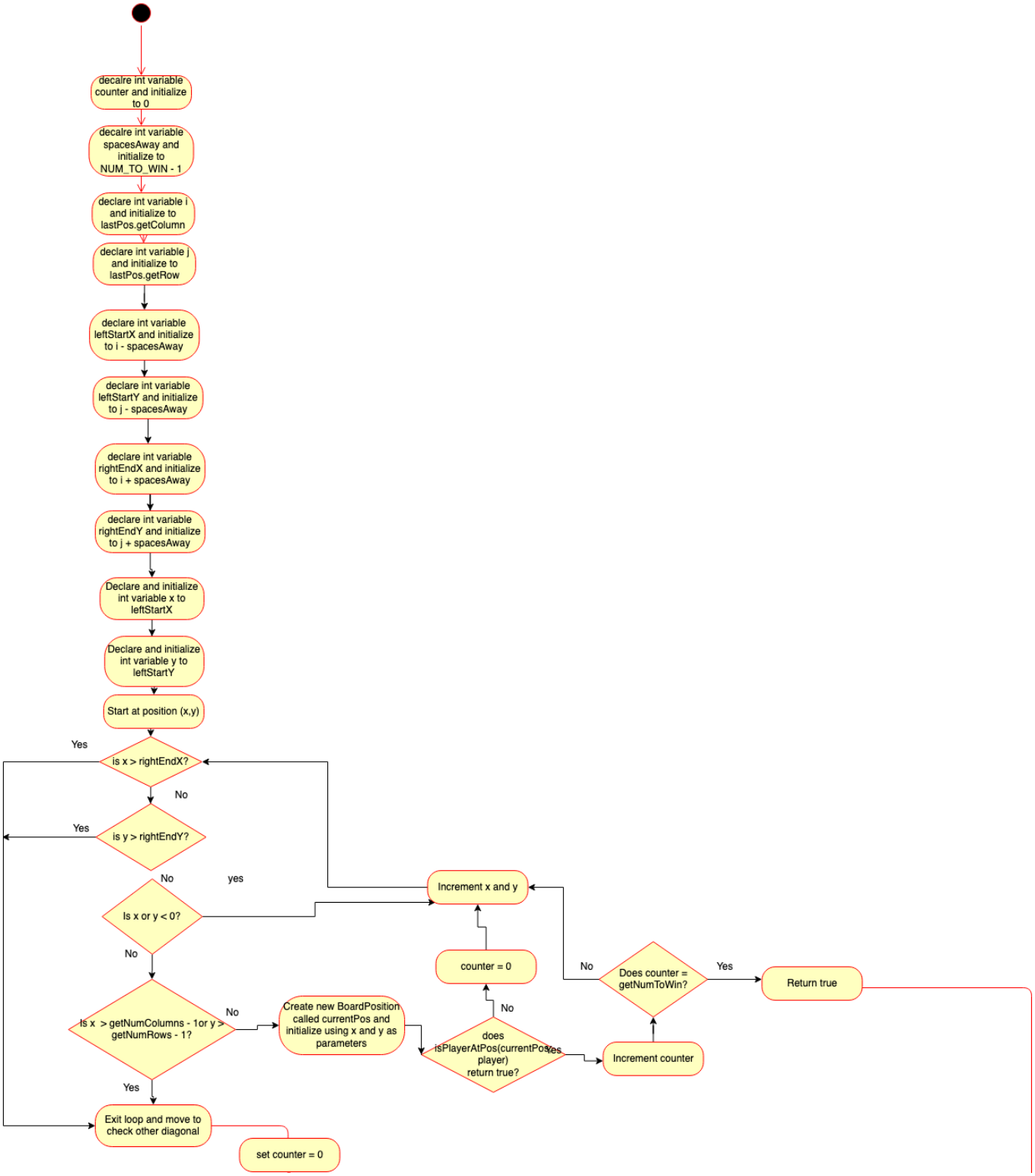
checkHorizontalWin(BoardPosition lastPos, char player)

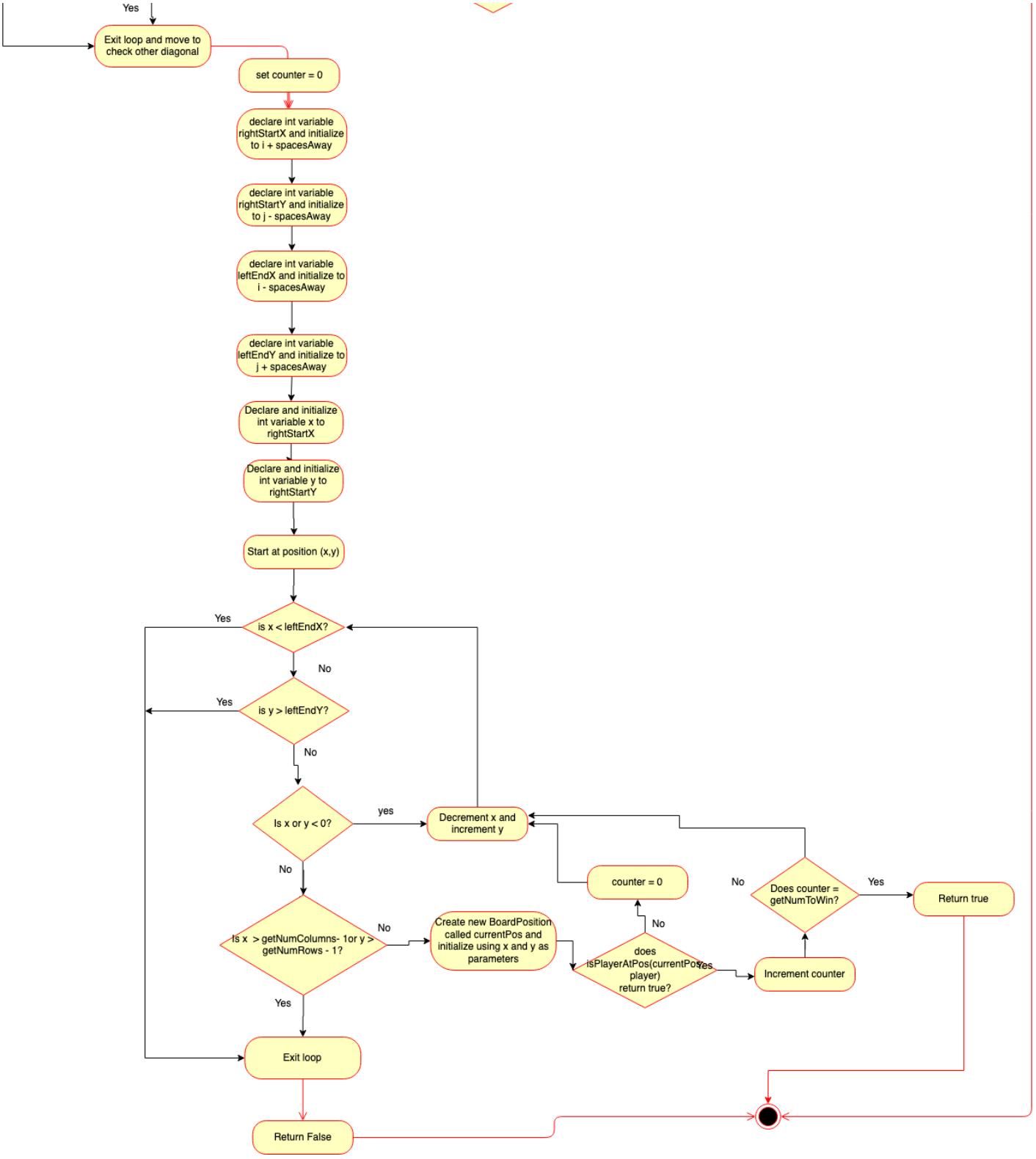


checkVerticallWin(BoardPosition lastPos, char player)

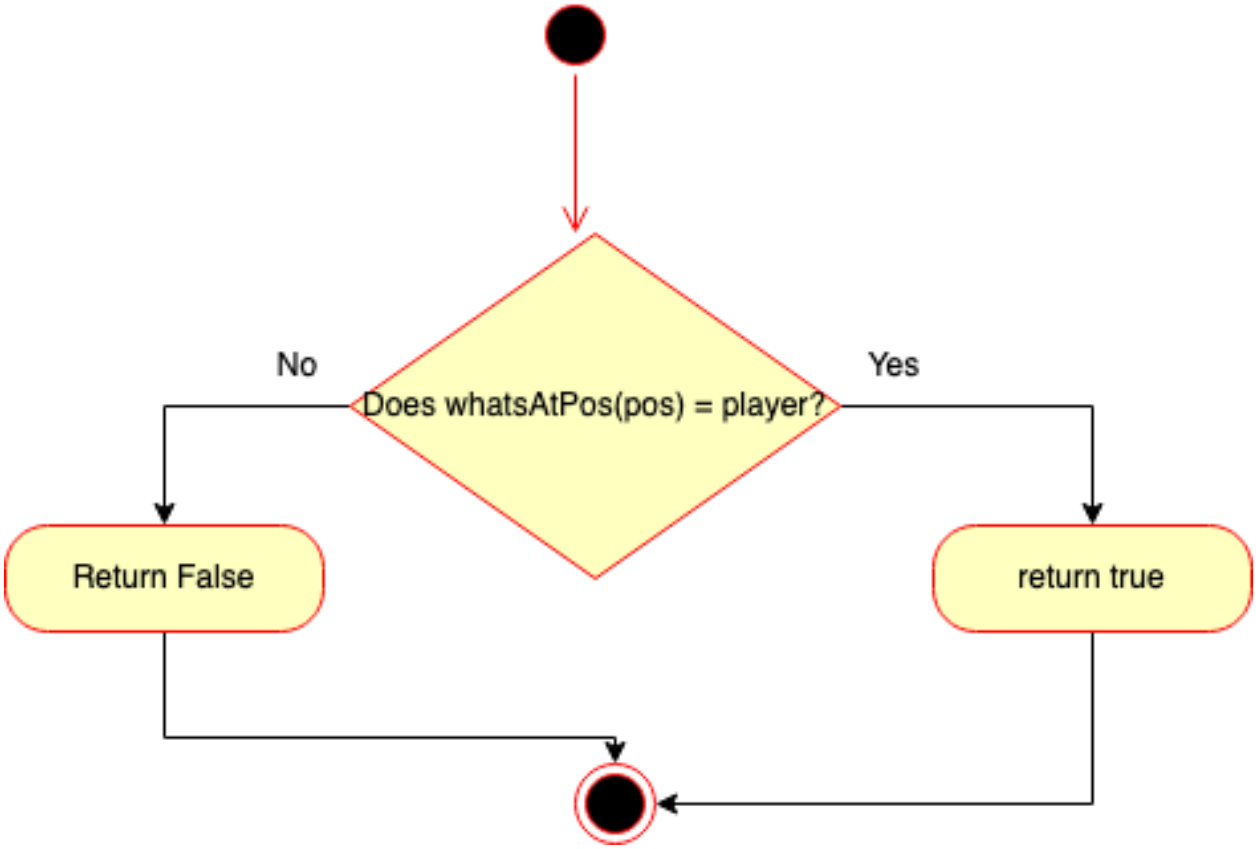


checkDiagonalWin(BoardPosition lastPos, char player)



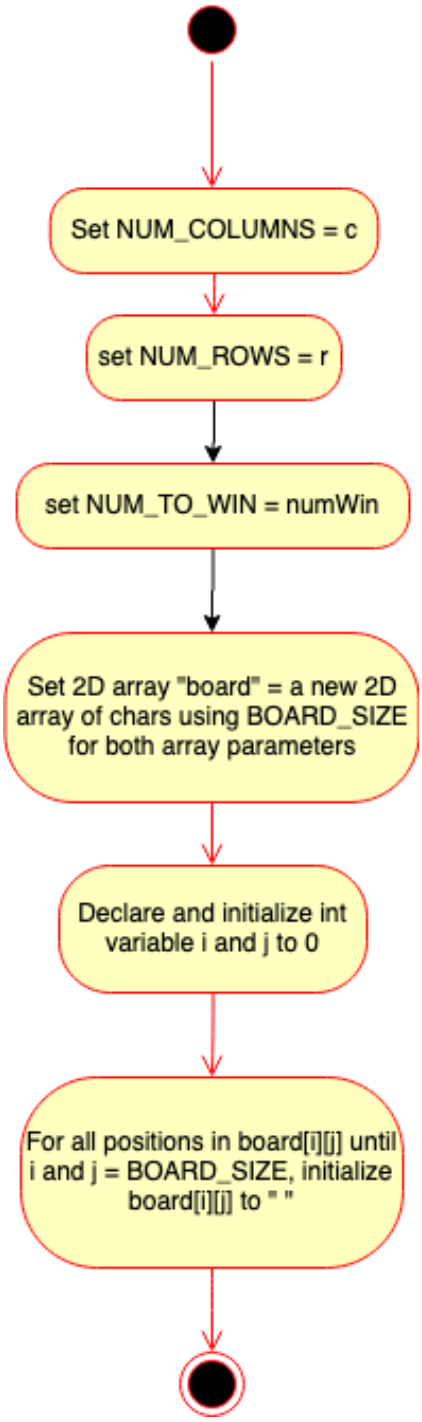

Yes
Exit loop and move to check other diagonal
set counter = 0
declare int variable rightStartX and initialize to i + spacesAway
declare int variable rightStartY and initialize to j - spacesAway
declare int variable leftEndX and initialize to i - spacesAway
declare int variable leftEndY and initialize to j + spacesAway
Declare and initialize int variable x to rightStartX
Declare and initialize int variable y to rightStartY
Start at position (x,y)
Yes
is x < leftEndX?
No
Yes
is y > leftEndY?
No
Is x or y < 0?
yes
Decrement x and increment y
No
Is x > getNumColumns- 1or y > getNumRows - 1?
No
Create new BoardPosition called currentPos and initialize using x and y as parameters
does isPlayerAtPos(currentPos, player) return true?
No
Yes
counter = 0
Increment counter
No
Does counter = getNumToWin?
Yes
Return true
Yes
Exit loop
Return False

isPlayerAtPos(BoardPosition pos, char player)

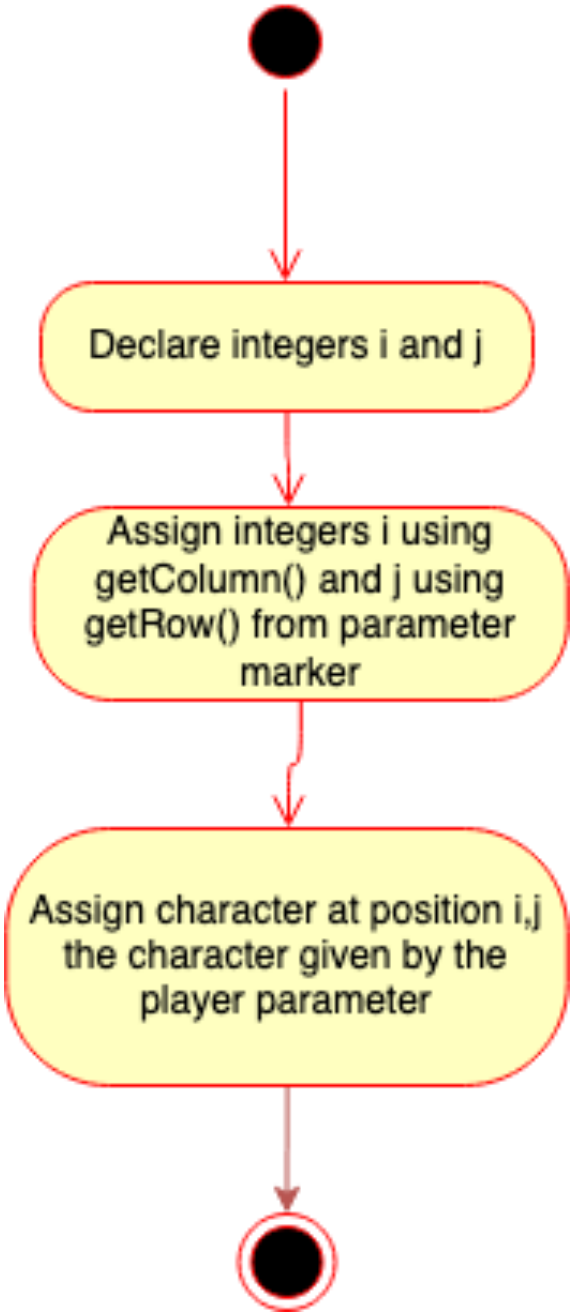


GameBoard – Primary Implementations

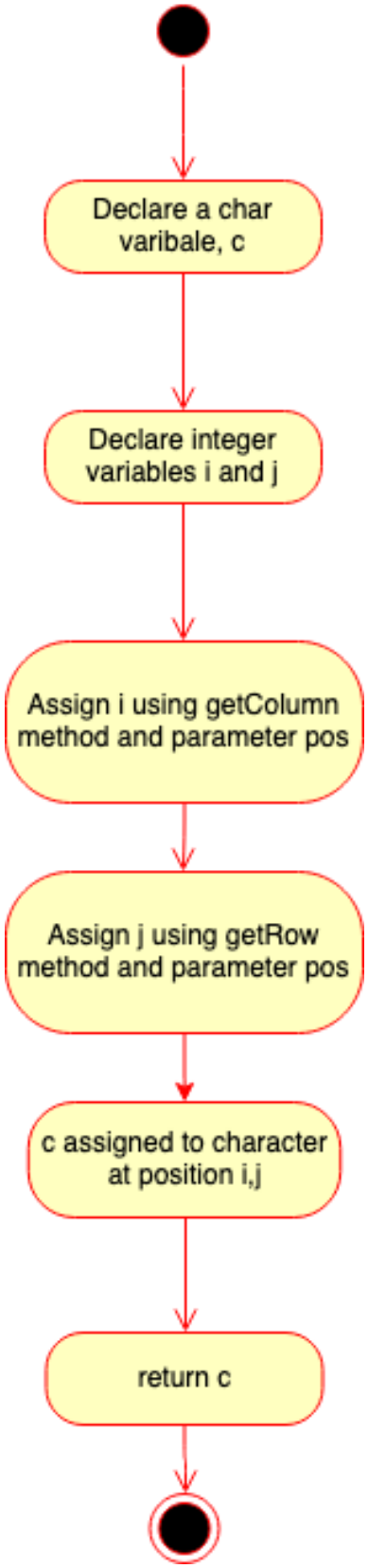
GameBoard(int r, int c, int numWin) (constructor)



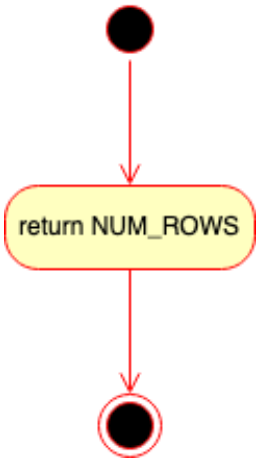
placeMarker(BoardPosition marker, char player)



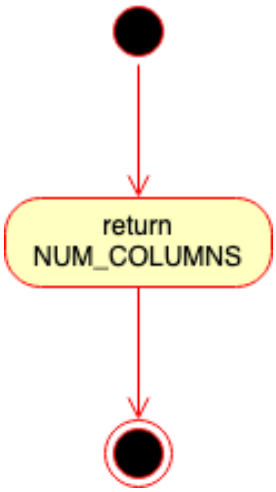
whatsAtPos(BoardPosition pos)



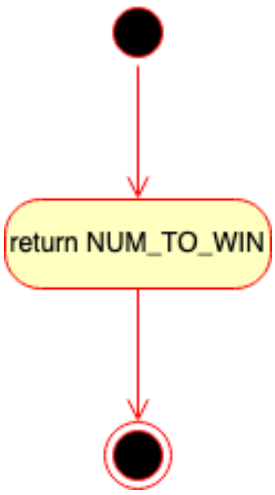
getNumRows()



getNumColumns()

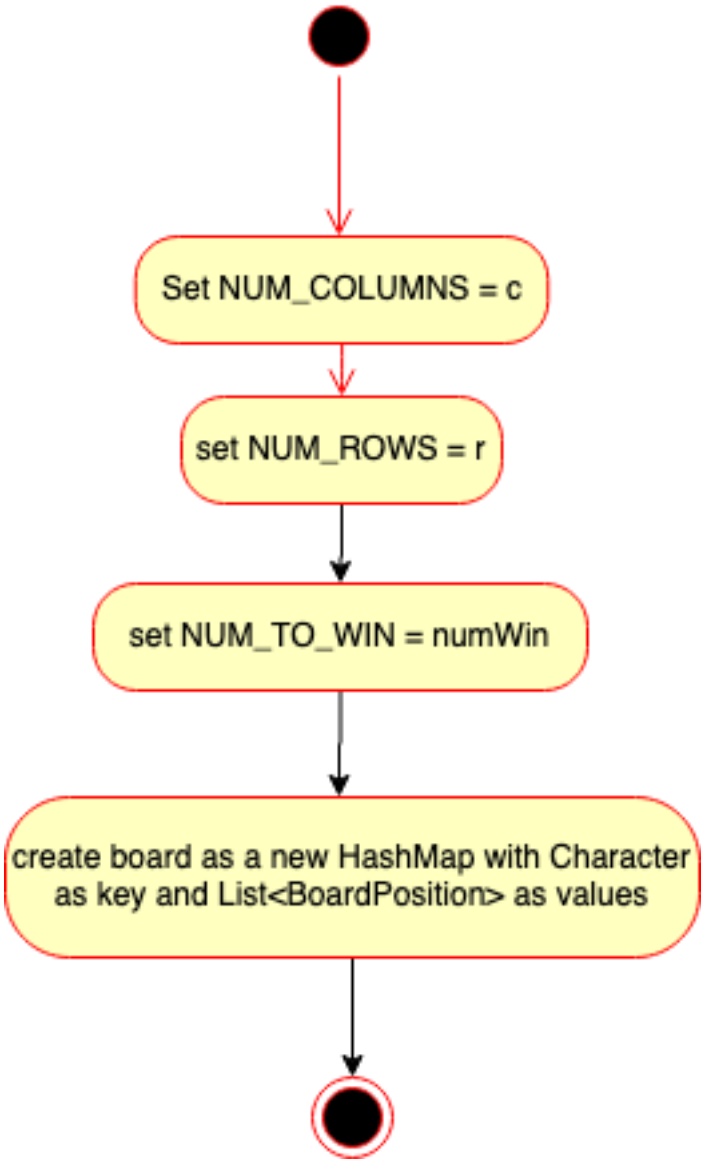


getNumToWin()

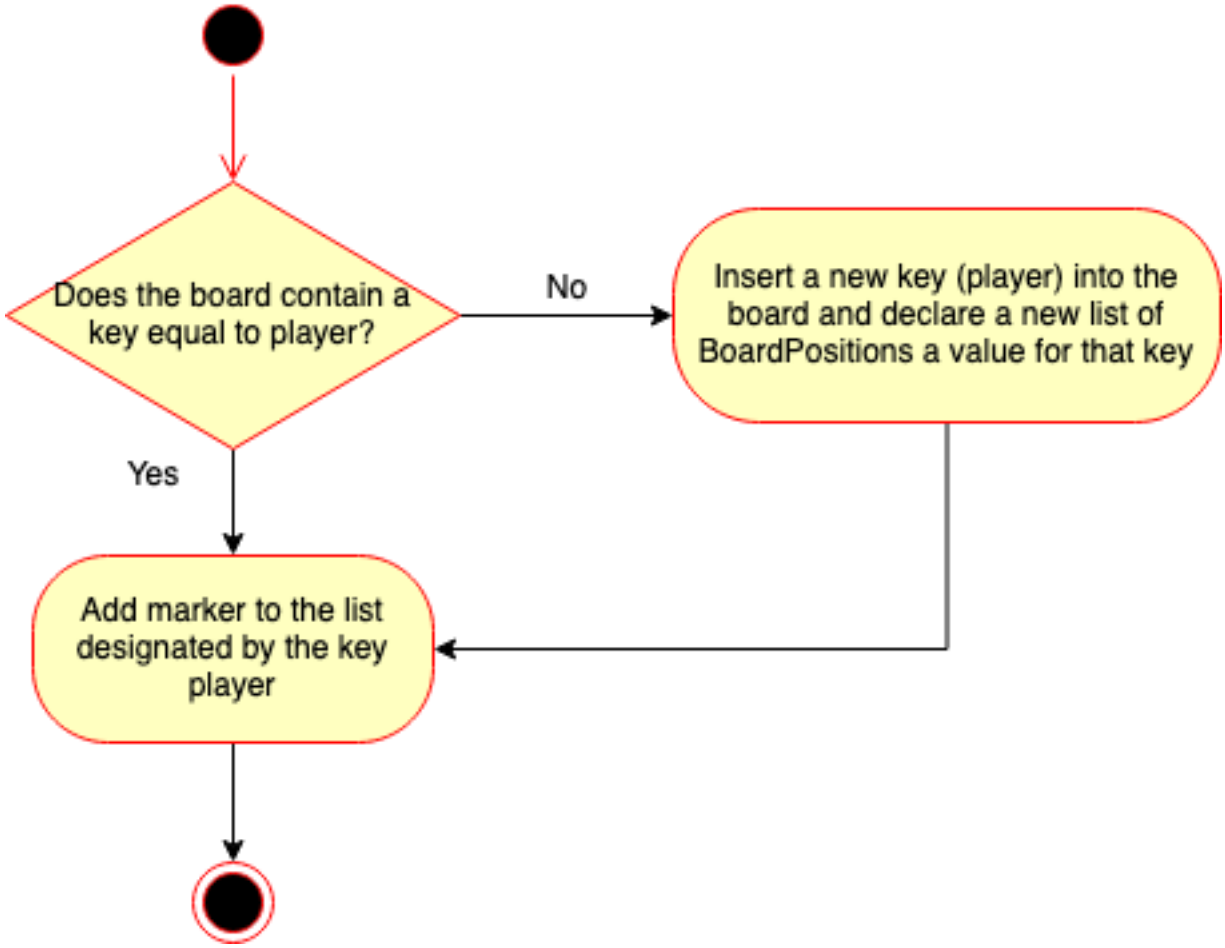


GameBoardMem – Primary Implementations

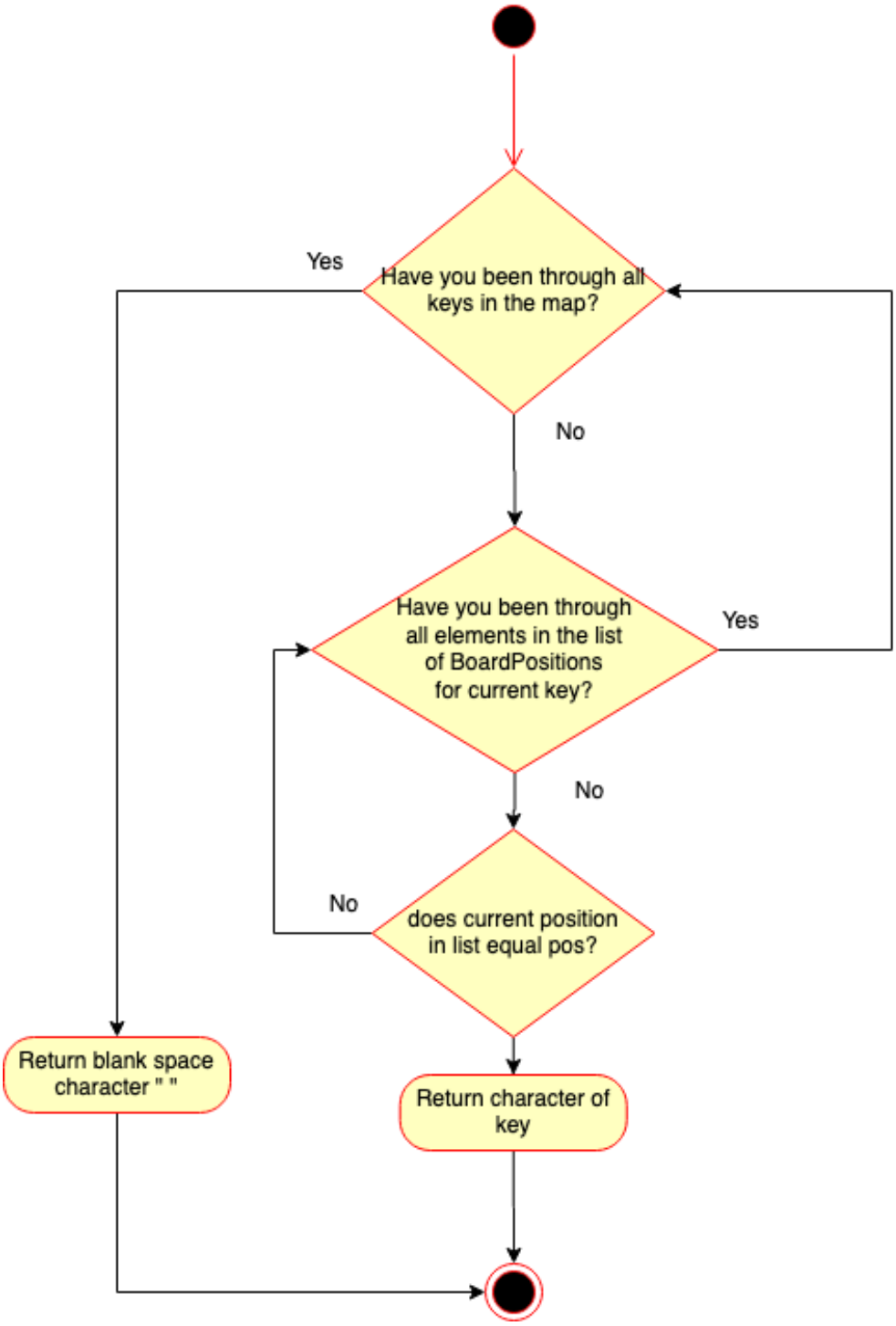
GameBoardMem(int r, int c, int numWin) (constructor)



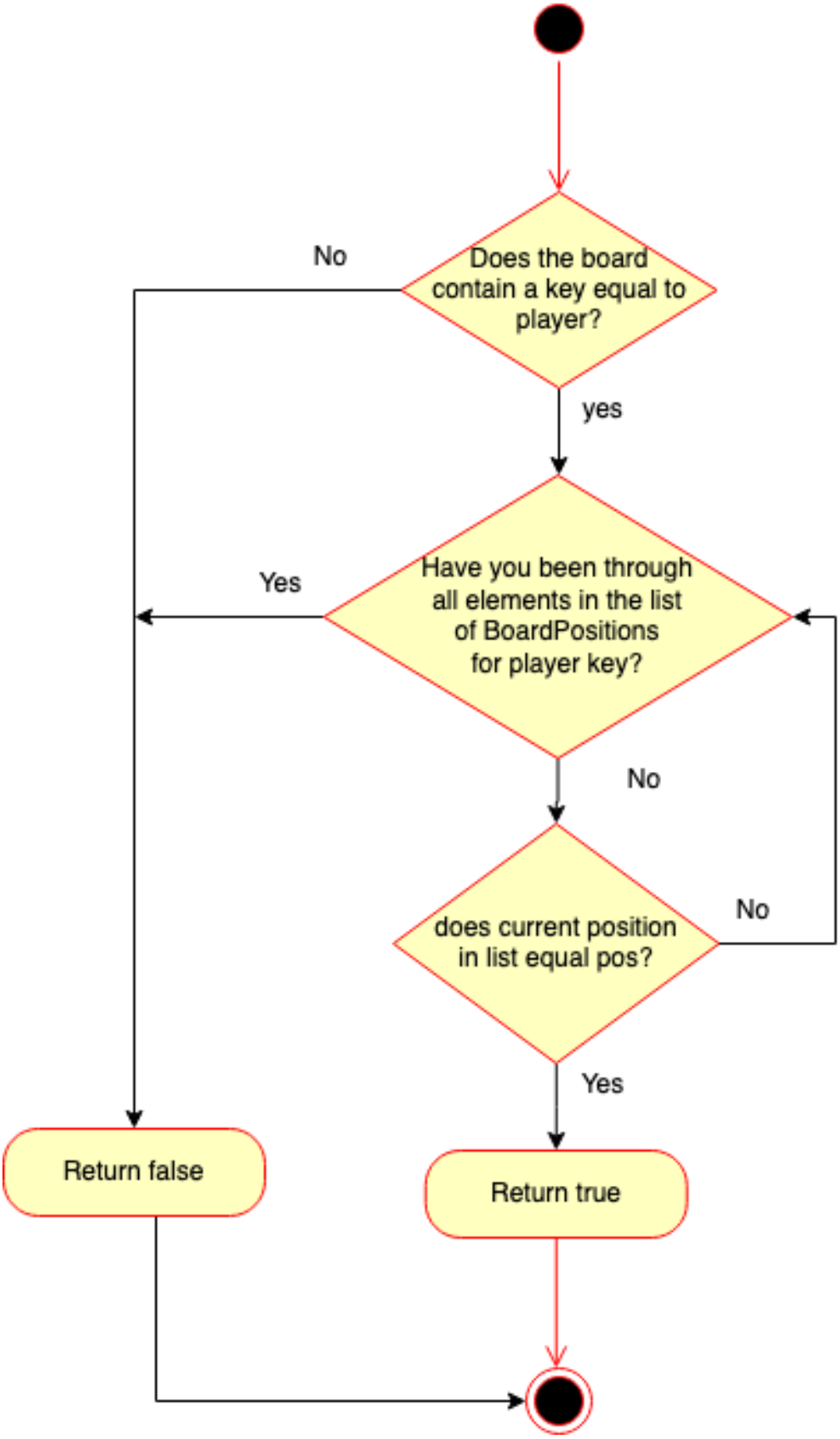
placeMarker(BoardPosition marker, char player)



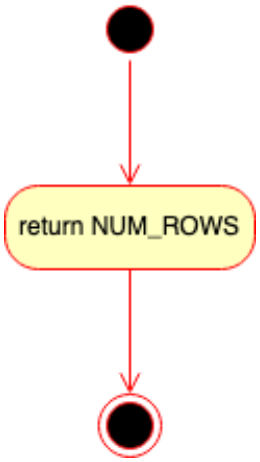
whatsAtPos(BoardPosition pos)



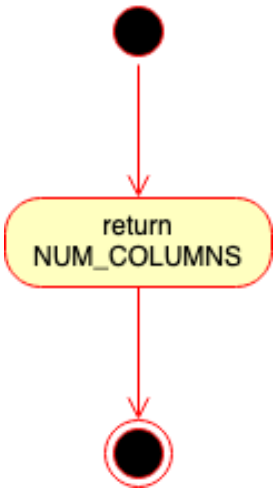
Override of isPlayerAtPos(BoardPosition pos, char player)



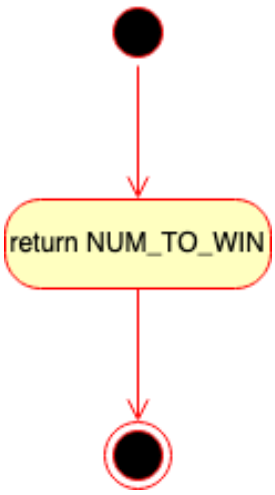
getNumRows()



getNumColumns()

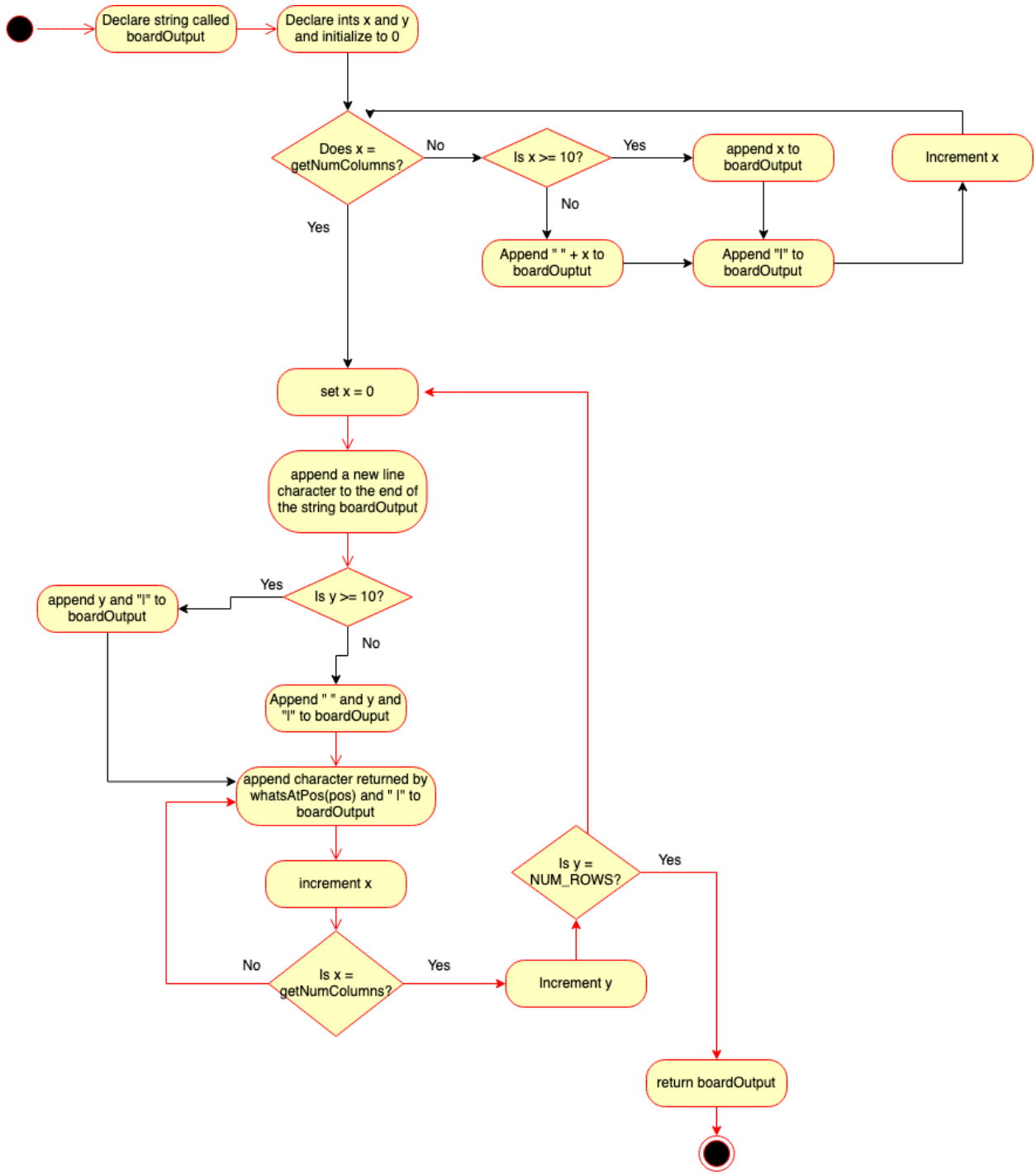


getNumToWin()



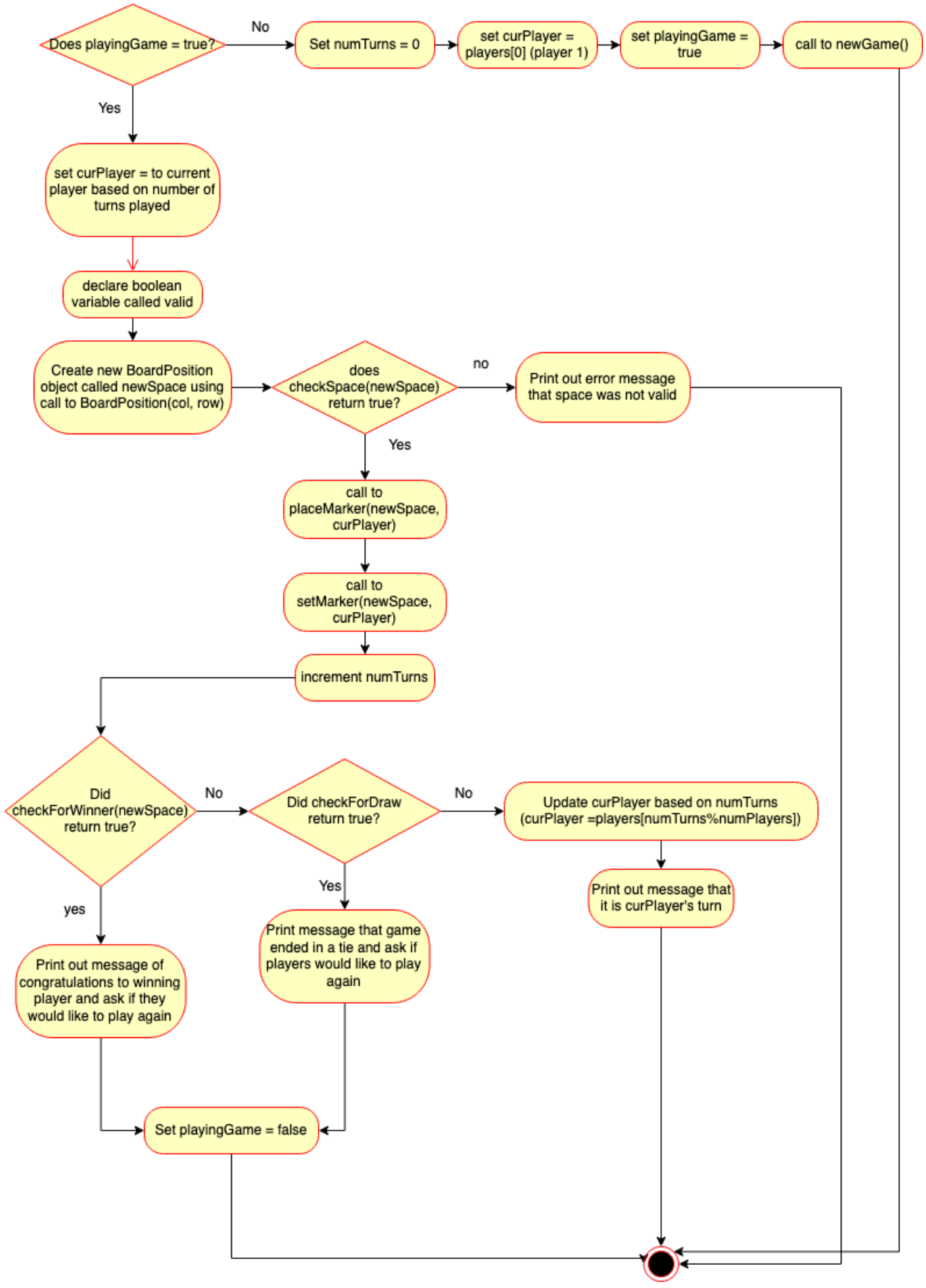
AbsGameBoard

toString()



TicTacToeController

processButtonClick(int row, int col)



Testing

Below are the test cases for each method.

`Constructor`

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr><td>r = 3
c = 3
numWin = 3</td><td>State: NUM_TO_WIN= 3
NUM_ROWS = 3
NUM_COLUMNS = 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></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td><td>This test case is distinct because it tests the constructor’s boundary case. 3 is the minimum number of rows, columns, and spaces in a row to win needed.

Function Name:
<code>test_Constructor_Min_Size</code></td></tr>
<tr><td>r = 100
c = 100
numWin = 25</td><td>State: NUM_TO_WIN= 25
NUM_ROWS = 100
NUM_COLUMNS = 100

Makes and empty board with sides that go from 0-99.

<table><tr><td></td><td>0</td><td>1</td><td>…</td><td>99</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td></tr><tr><td>…</td><td></td><td></td><td></td><td></td></tr><tr><td>99</td><td></td><td></td><td></td><td></td></tr></table></td><td>This test case is distinct because it tests the other boundary case for the constructor. The max number of rows and columns the board can have is 100 and the max needed in a row to win can be 25.

Function Name:
<code>test_Constructor_Max_Size</code></td></tr>
<tr><td>r = 6
c = 6
numWin = 4</td><td>State: NUM_TO_WIN= 4
NUM_ROWS = 6
NUM_COLUMNS = 6

<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>0</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></tr><tr><td>2</td><td></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><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>5</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>This test case is distinct because it tests a routine scenario for the constructor. 6 is between the minimum and maximum for the number of rows and columns. 4 is in between the minimum and maximum for the number needed to win.
Function Name:
<code>test_Constructor_Routine</code></td></tr>
</table>

`boolean checkSpace(BoardPosition pos)`

Input	Output	Reason
State: (num to win = 3) | | 0 | 1 | 2 | 3 | | 0 | | | | | | 1 | | | | | | 2 | | | | | | 3 | | | | | Pos.getRow = 0 Pos.getColumn = 0	checkSpace = true State of board is unchanged	This test case is distinct because it checks if a space is available at a boundary position on the gameboard (0,0). The test case is also distinct as it is testing a space without a character. **Function Name:** `test_checkSpace_at_upperLeftCorner`
State: (num to win = 3) | | 0 | 1 | 2 | 3 | | 0 | | | | | | 1 | | | | | | 2 | | | | | | 3 | | | | | Pos.getRow = 4 Pos.getColumn = 4	checkSpace = false State of the board is unchanged	This test case is distinct because it is testing a scenario in which the player is checking a space that is off of the board (out of bounds), so checkSpace is expected to return false, as it is an invalid position. **Function Name:** `test_checkSpace_at_outOfBounds`
State: (num to win = 3) | | 0 | 1 | 2 | 3 | | 0 | | | | | | 1 | | x | | | | 2 | | | | | | 3 | | | | | Pos.getRow = 1 Pos.getColumn = 1	checkSpace = false State of the board is unchanged	This test case is distinct because it is testing a scenario in which the player is checking a space that is already taken by another player token, so the space is unavailable. **Function Name:** `test_checkSpace_Unavailable`

`boolean checkHorizontalWin(BoardPosition lastPos, char player)`

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>State(number to win = 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>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>
player = ‘x’
lastPos.getRow = 2
lastPos.getColumn = 2</td>
<td>checkHorizontalWin = true

state of the board is unchanged</td>
<td>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.

Function Name:
<code>test_Horizontal_MiddleWin</code></td>
</tr>
<tr>
<td>State(number to win = 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>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>
player = ‘x’
lastPos.getRow = 2
lastPos.getColumn = 0</td>
<td>checkHorizontalWin = true

state of the board is unchanged</td>
<td>This test case is unique and distinct because the last x was placed on the left boundary of the board, so the function needs to start counting at the 0 column of the board and make sure it does not start counting from outside of the board.

Function Name:
<code>test_Horizontal_left_Bound</code></td>
</tr>
<tr>
<td>State(number to win = 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>x</td><td>x</td><td>x</td><td>x</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>
player = ‘x’
lastPos.getRow = 2
lastPos.getColumn = 4</td>
<td>checkHorizontalWin = true

state of the board is unchanged</td>
<td>This test case is unique and distinct because the last x was placed on the right boundary of the board, so the function needs to stop counting at the 5th column of the board and make sure it does not go past the edge of the board.

Function Name:
<code>test_Horizontal_right_Bound</code></td>
</tr>
<tr>
<td>State(number to win = 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>x</td><td></td><td>x</td><td>x</td></tr>
<tr><td>2</td><td></td><td>o</td><td>o</td><td>o</td><td></td></tr>
<tr><td>3</td><td></td><td>o</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>checkHorizontalWin = false

state of the board is unchanged</td>
<td>This test case is unique and distinct because the last x was placed in a position that does not result in there being enough x’s in a row to satisfy the condition of 4 in a row needed to win, so we expect the function to return false.</td>
</tr>
</table>

player = ‘x’ lastPos.getRow = 1 lastPos.getColumn = 0		**Function Name:** `test_Horizontal_No_Win`

`boolean checkVerticalWin(BoardPosition lastPos, char player)`

Input	**Output**	**Reason**
State(number to win = 4) player = ‘x’ lastPos.getRow = 1 lastPos.getColumn = 2	checkVerticalWin = true state of the board is unchanged	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 both above and below the final x placed. **Function Name:** `test_Vertical_Middle`
State(number to win = 4) player = ‘x’ lastPos.getRow = 0 lastPos.getColumn = 2	checkVerticalWin = true state of the board is unchanged	This test case is unique and distinct because the last x was placed on upper boundary of the board, so the function needs to make sure it does not go into a negative index (past row 0) when checking for a win. **Function Name:** `test_Vertical_Upper`
State(number to win = 4) player = ‘x’ lastPos.getRow = 4 lastPos.getColumn = 2	checkVerticalWin = true state of the board is unchanged	This test case is unique and distinct because the last x was placed on the lower boundary of the board, so the function needs to make sure it does not go to an index out of bounds (past the last row) on the gameboard. **Function Name:** `test_Vertical_Lower`
State(number to win = 4)	checkVerticalWin = false	This test case is unique and distinct because the last x

	0	1	2	3	4
0			x		
1	o		x		
2	x		x		
3	o		x	o	
4	o				

	0	1	2	3	4
0			x		
1	o		x		
2	x		x		
3	o		x	o	
4	o				

	0	1	2	3	4
0					
1	o		x		
2	x		x		
3	o		x	o	
4	o		x		

	0	1	2	3	4

0			x		
1	o		x		
2	x				
3	o		x	o	
4	o		x		

player = ‘x’
lastPos.getRow = 3
lastPos.getColumn = 2

state of the board is unchanged

was placed in a position that does not result in there being enough x’s in a row to satisfy the condition of 4 in a row needed to win, so we expect the function to return false.

Function Name:
`test_Vertical_No_Win`

`boolean checkDiagonalWin(BoardPosition lastPos, char player)`

Input	**Output**	**Reason**
State(number to win = 3) player = ‘x’ lastPos.getRow = 0 lastPos.getColumn = 0	checkDiagonalWin = true state of the board is unchanged	This test case is unique and distinct because the last x was placed at the corner (0,0) which is a boundary case where it is at the minimum possible location. It is also different because the diagonal is at a left to right angle. **Function Name:** `test_Diagonal_Upper_Left`
State(number to win = 3) player = ‘x’ lastPos.getRow = 0 lastPos.getColumn = 4	checkDiagonalWin = true state of the board is unchanged	This test case is unique and distinct because the last x was placed at the upper right-hand corner which is another boundary case. It is also different because the diagonal is at a right to left angle. **Function Name:** `test_Diagonal_Upper_Right`
State(number to win = 3)	checkDiagonalWin = true state of the board is unchanged	This test case is unique and distinct because the last x was placed at another boundary case (the lower righ- hand corner). Also, the diagonal is oriented left to right going into the corner

	0	1	2	3	4
0	x				
1	o	x			
2	o		x		
3					
4					

	0	1	2	3	4
0					x
1	o			x	
2	o		x		
3					
4					

	0	1	2	3	4
0					
1	o				
2	o		x		
3				x	
4					x

<table>
<tr>
<td>player = ‘x’
lastPos.getRow = 4
lastPos.getColumn = 4</td>
<td></td>
<td>Function Name:
<code>test_Diagonal_Lower_Right</code></td>
</tr>
<tr>
<td>State(number to win = 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>o</td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td>o</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>player = ‘x’
lastPos.getRow = 4
lastPos.getColumn = 0</td>
<td>checkDiagonalWin = true

state of the board is unchanged</td>
<td>This test case is unique and distinct because the last x was placed in another boundary case (lower left-hand corner). Also the diagonal is oriented going right to left into the corner.

Function Name:
<code>test_Diagonal_Lower_Left</code></td>
</tr>
<tr>
<td>State(number to win = 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>o</td><td>x</td><td></td><td></td><td></td></tr><tr><td>2</td><td>o</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>player = ‘x’
lastPos.getRow = 2
lastPos.getColumn = 2</td>
<td>checkDiagonalWin = true

state of the board is unchanged</td>
<td>This test case is unique and distinct because the last x was placed in the middle of a win and represents a routine win where the win is bounded without hitting an edge of the board.

Function Name:
<code>test_Diagonal_Routine</code></td>
</tr>
<tr>
<td>State(number to win = 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>x</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>o</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>player = ‘x’
lastPos.getRow = 2
lastPos.getColumn = 4</td>
<td>checkDiagonalWin = true

state of the board is unchanged</td>
<td>This test case is unique and distinct because the last x was placed at an edge of the board. Also, each of the ends of the diagonal is at an edge or boundary of the board, so the function cannot go out of bounds at both ends.

Function Name:
<code>test_Diagonal_Edges</code></td>
</tr>
<tr>
<td>State(number to win = 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>o</td><td>x</td><td></td><td>x</td><td></td></tr><tr><td>2</td><td>o</td><td></td><td>x</td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>checkDiagonalWin = false

state of the board is unchanged</td>
<td>This test case is unique and distinct because the last x was placed in a position that did not result in a win in the diagonal.

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

4 player = 'x' lastPos.getRow = 1 lastPos.getColumn = 3		`test_Diagonal_No_Win`

boolean checkForDraw()

Input	Output	Reason
State: (number to win = 3) 0 1 2 0 x o x 1 x x o 2 o x o	checkForDraw = true state of the board does not change	This is a distinct test case because it represents a scenario that ended in a tie. All spaces are filled in no wins are present. **Function Name:** `test_Draw_True`
State: (number to win = 3) 0 1 2 0 x 1 2	checkForDraw = false state of the board does not change	This is a distinct test case because it represents the board after the first move has been played. Only one token is on the board, so a tie is not possible. Also, the token was placed at the boundary (0,0). **Function Name:** `test_Draw_FirstSpace`
State: (number to win = 3) 0 1 2 0 x o x 1 x x o 2 o x	checkForDraw = false state of the board does not change	This is a distinct test case because it represents a case where the last token played has not quite filled up the board (only one position is left). **Function Name:** `test_Draw_Almost_Full`
State: (number to win = 3) 0 1 2 0 x x	checkForDraw = false	This is a distinct test case because it represents a routine scenario in which

1	x	o	
2			o

state of the board does not change

some of the board has been filled but not all of it.

Function Name:
`test_Draw_Routine`

`char whatsAtPos(BoardPosition pos)`

Input	**Output**	**Reason**
State: (number to win = 3) | | 0 | 1 | 2 | | 0 | x | | | | 1 | | | | | 2 | | | | pos.getRow = 0 pos.getColumn = 0	whatsAtPos = 'x' state of the board does not change	This is a distinct test case because it looks at the position at the board that is located at the most minimum boundary, the upper left-hand corner (0,0). **Function Name:** `test_whatsAtPos_upperLeft`
State: (number to win = 3) | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | | | x | pos.getRow = 2 pos.getColumn = 2	whatsAtPos = 'x' state of the board does not change	This is a distinct test case because it looks at the position at the board that is located at the most maximum boundary, the lower right-hand corner (2,2) **Function Name:** `test_whatsAtPos_lowerRight`
State: (number to win = 3) | | 0 | 1 | 2 | | 0 | | | | | 1 | | o | | | 2 | | | | pos.getRow = 1 pos.getColumn = 1	whatsAtPos = 'o' state of the board does not change	This is a distinct test case because it looks at a board position that represents a routine scenario. The board position is in the middle of the board and not at any boundary. **Function Name:** `test_whatsAtPos_Middle`
State: (number to win = 3) | | 0 | 1 | 2 |	whatsAtPos = ' '	This is a distinct test case because it looks at a position on the board that is not currently

0			
1		o	
2			

pos.getRow = 2
p pos.getColumn = 2

state of the board does not change

occupied by a player token, so we should expect the function to return a blank space character.

Function Name:
`test_whatsAtPos_EmptySpace`

State: (number to win = 3)

	0	1	2
0			
1			x
2			

pos.getRow = 1
pos.getColumn = 2

whatsAtPos = 'x'

state of the board does not change

This is a distinct test case because it looks at a position on the board that is at a boundary for the columns, but not at a boundary for the rows. It is at the right most column but is not near the edge of the board with regards to row location.

Function Name:
`test_whatsAtPos_Edge`

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input	**Output**	**Reason**
State: (number to win = 3) | | 0 | 1 | 2 | | 0 | x | | | | 1 | | | | | 2 | | | | player = 'x' pos.getRow = 0 pos.getColumn = 0	isPlayeratPos = true state of the board does not change	This is a distinct test case because it looks at the position at the board that is located at the most minimum boundary, the upper left-hand corner (0,0). **Function Name:** `test_isPlayerAt_at_upperLeftCorner`
State: (number to win = 3) | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | | | x | player = 'x'	isPlayeratPos = true state of the board does not change	This is a distinct test case because it looks at the position at the board that is located at the most maximum boundary, the lower right-hand corner (2,2) **Function Name:** `test_isPlayerAt_at_LowerRight`

<table>
<tr>
<td>pos.getRow = 2
pos.getColumn = 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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>
player = ‘x’
pos.getRow = 1
pos.getColumn = 1</td>
<td>isPlayeratPos = true

state of the board does not change</td>
<td>This is a distinct test case because it looks at a board position that represents a routine scenario. The board position is in the middle of the board and not at any boundary and another player is on the board too.

Function Name:
<code>test_isPlayerAt_at_Middle</code></td>
</tr>
<tr>
<td>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></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>
player = ‘x’
pos.getRow = 1
pos.getColumn = 1</td>
<td>isPlayeratPos = false

state of the board does not change</td>
<td>This is a distinct test case because it looks at a position on the board that is not currently occupied by the player token, so the player is not at that position, so we expect the function to return false.

Function Name:
<code>test_isPlayerAt_No_Players</code></td>
</tr>
<tr>
<td>State: (number to win = 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>o</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>
player = ‘x’
pos.getRow = 1
pos.getColumn = 1</td>
<td>isPlayeratPos = false

state of the board does not change</td>
<td>This is a distinct test case because it looks at a case where there is a player in the space, but it is not the player being searched for, as compared to just an empty space.

Function Name:
<code>test_isPlayerAt_Wrong_Player</code></td>
</tr>
</table>

Deployment

Run GUI Configuration of program through IntelliJ.