REQUIREMENT ANALYSIS

Functional Requirements

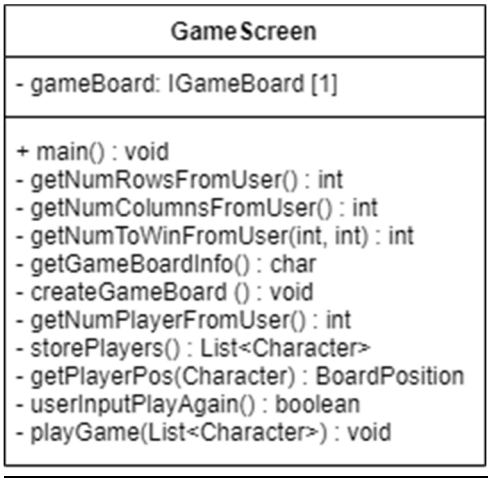
1. As a player, I can enter my position so that I can make my move on the game board
2. As a player, I can view the game board, so I know the state of the game
3. As a player, I have the option to play the game again so I can play the game again
4. As a player, I need to know if the position I entered 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

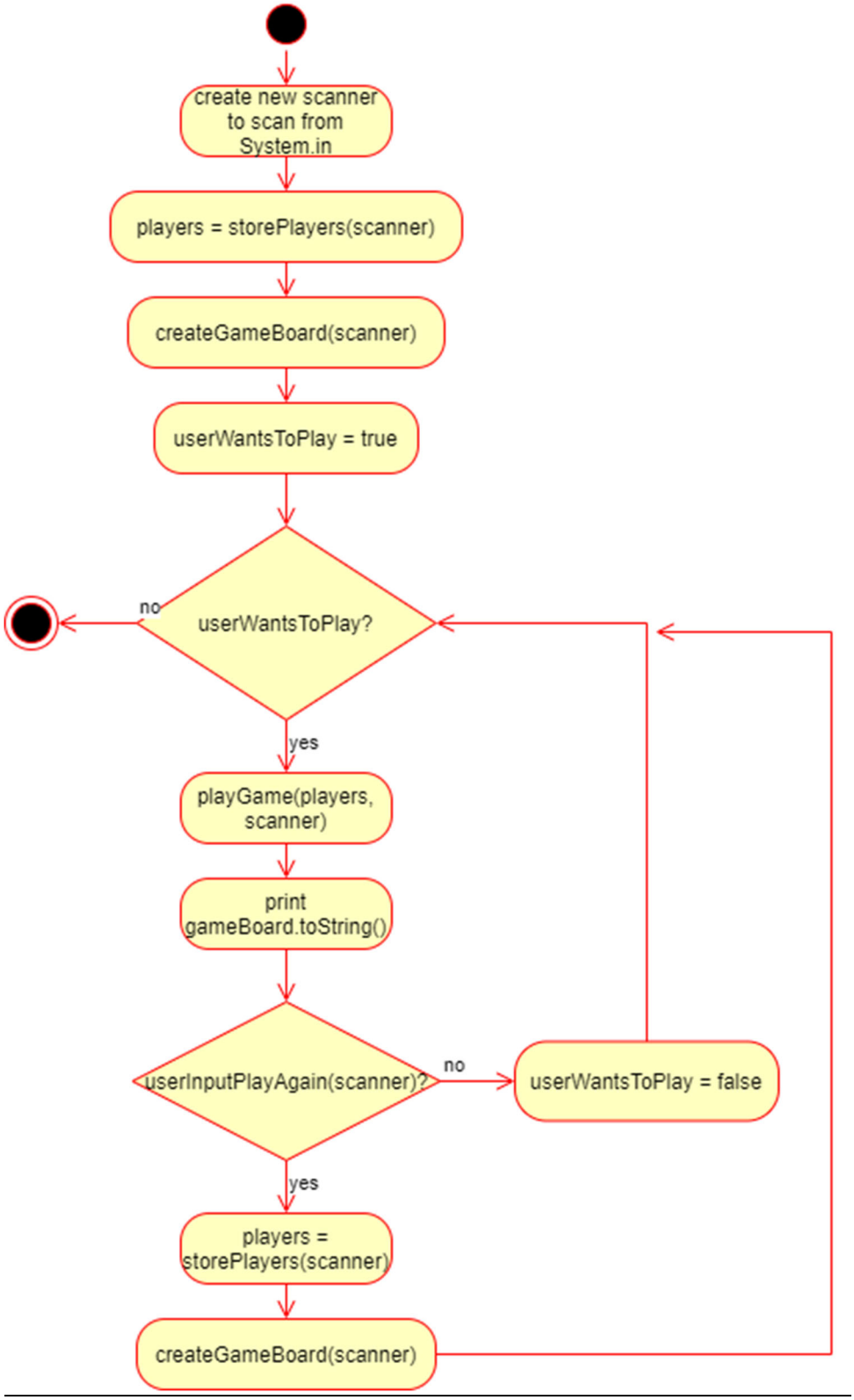
1. The code runs on Unix (school of computing computers)
2. Use Java as the coding language
3. Gameboard at least will have 3 rows and 3 columns
4. Gameboard at most will have 100 rows and 100 columns
5. The number in a row to win will be at least 3 and at most 25
6. The number in a row to win cannot 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

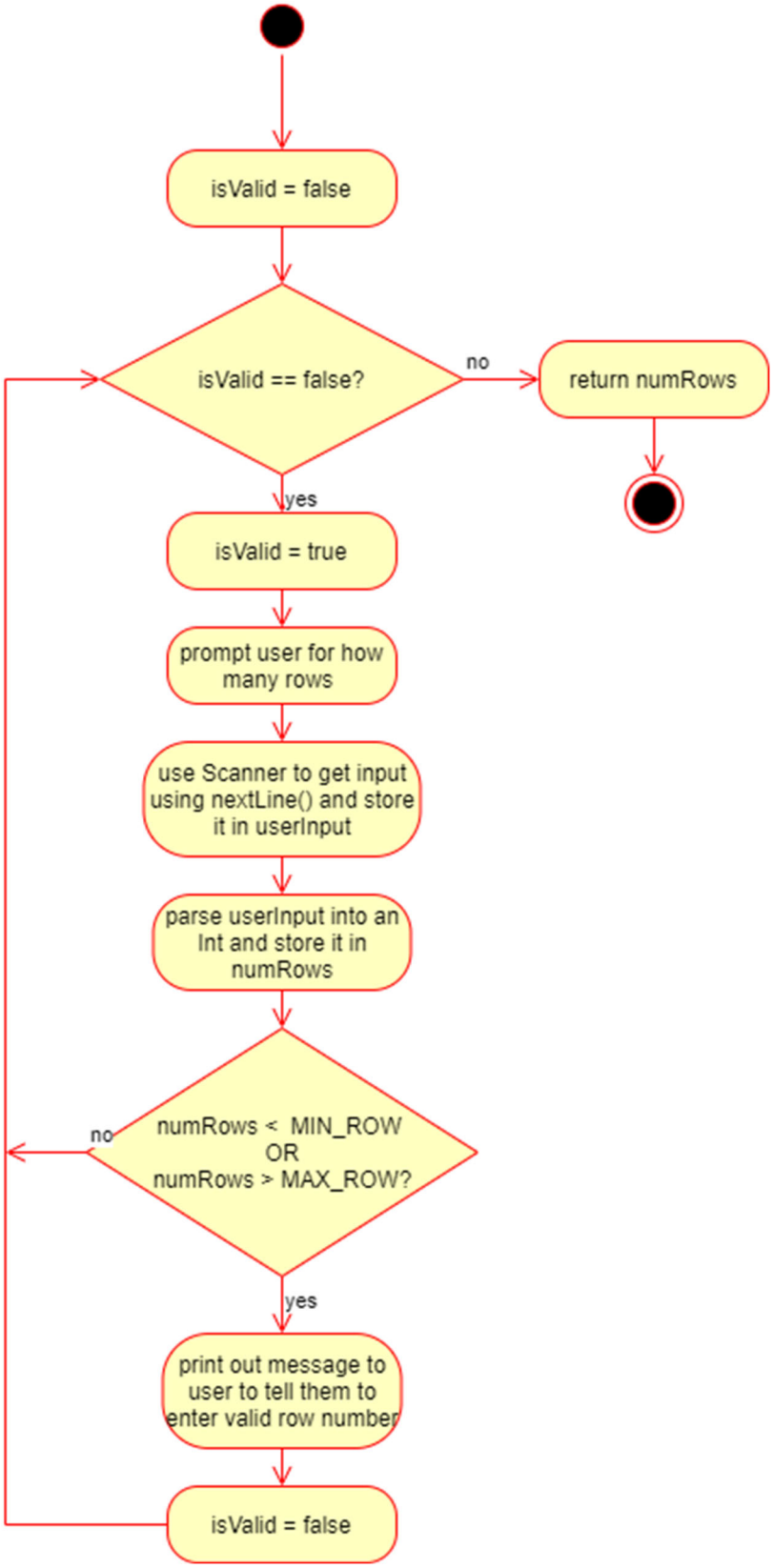
GameScreen Class



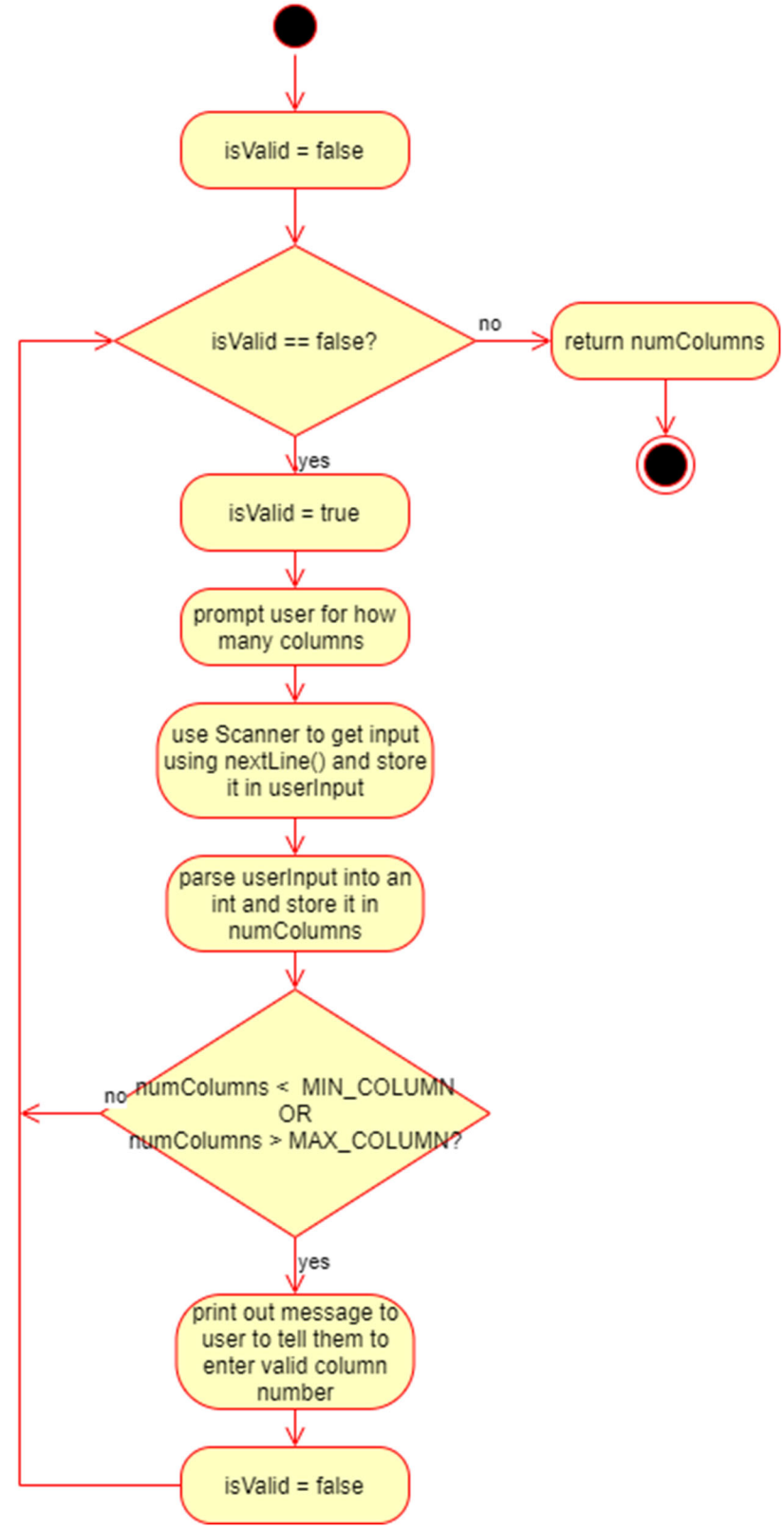
Main


create new scanner to scan from System.in
players = storePlayers(scanner)
createGameBoard(scanner)
userWantsToPlay = true
userWantsToPlay?
no
yes
playGame(players, scanner)
print gameBoard.toString()
userInputPlayAgain(scanner)?
no
userWantsToPlay = false
yes
players = storePlayers(scanner)
createGameBoard(scanner)

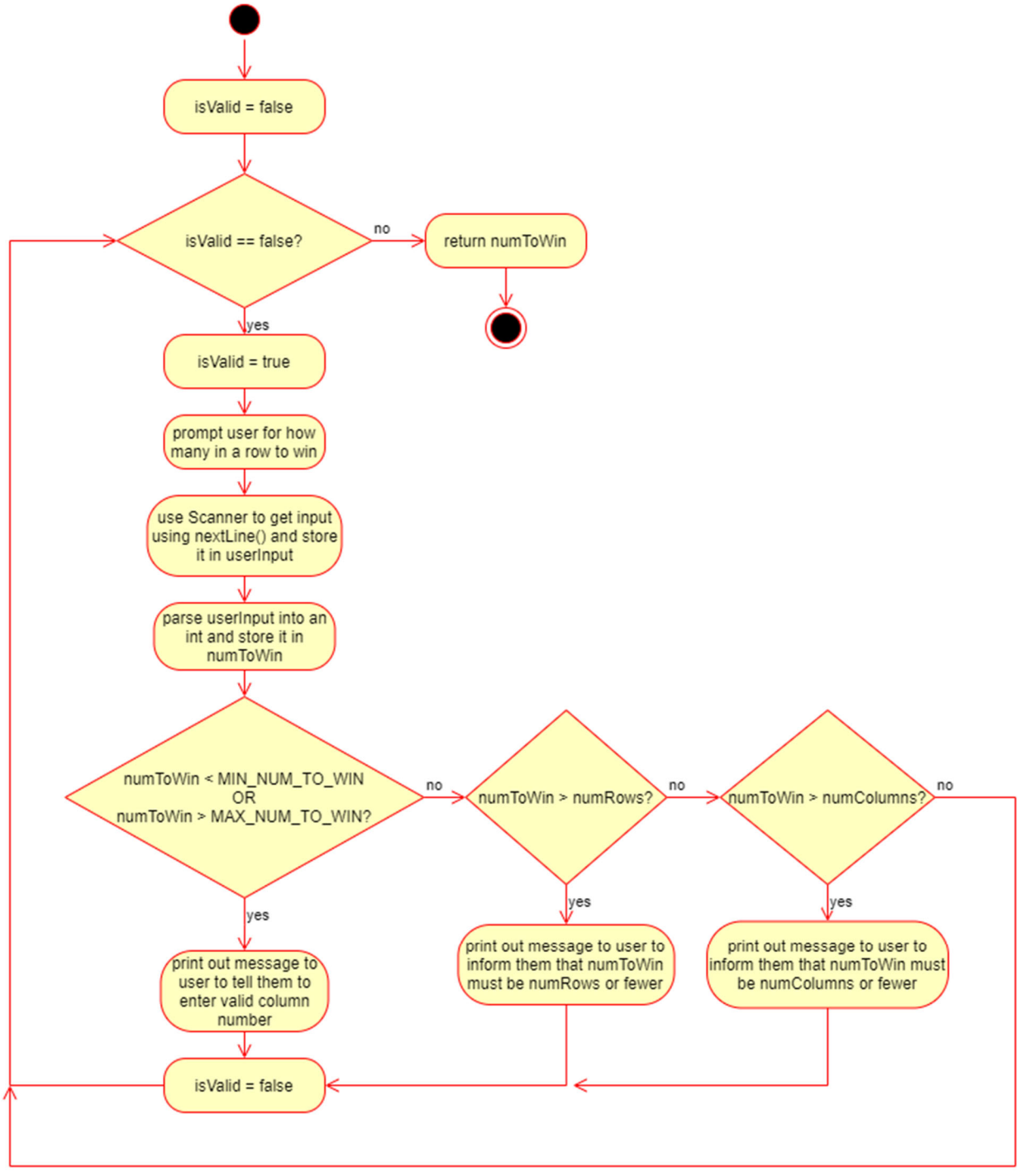
getNumRowsFromUser



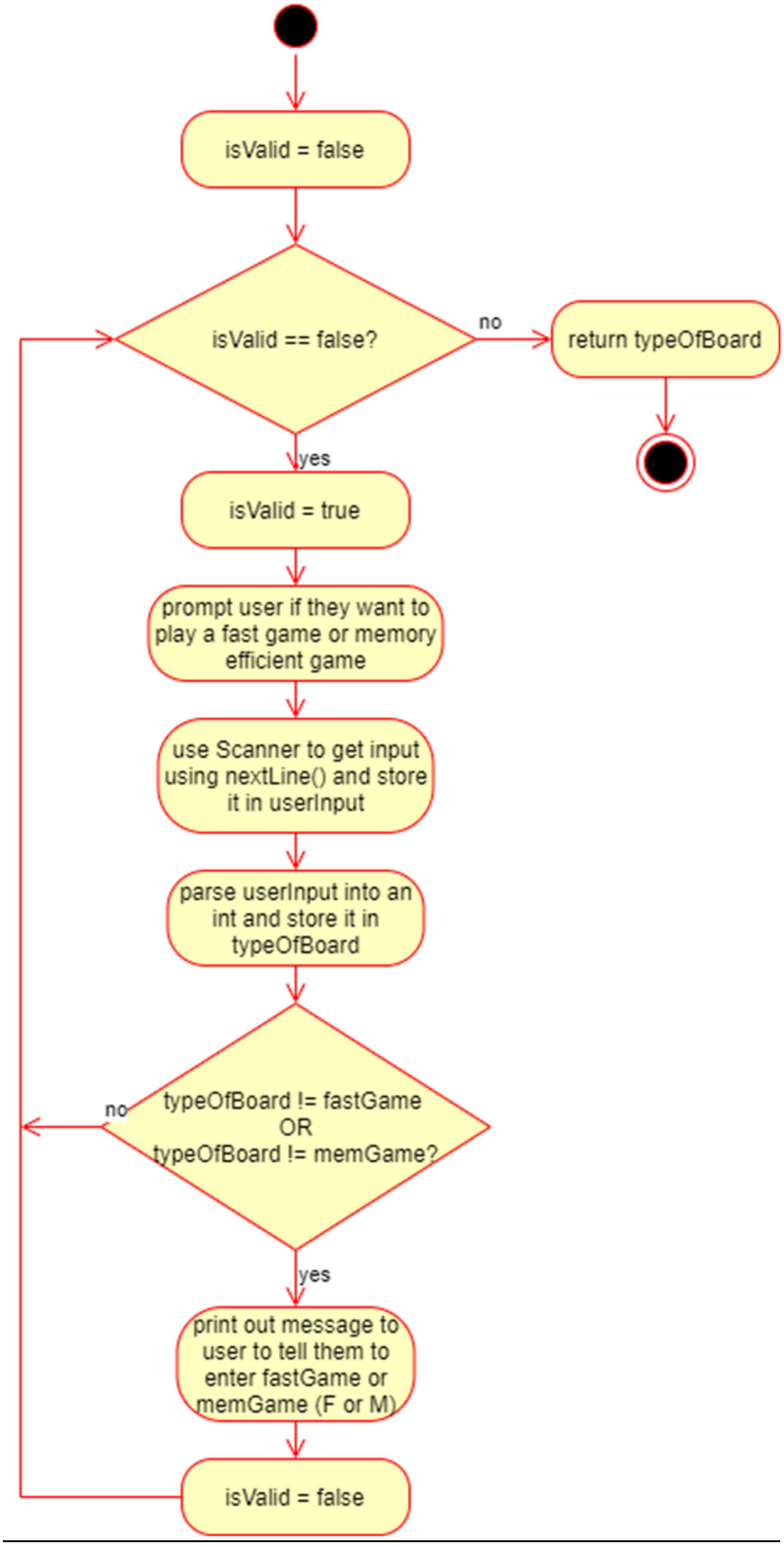
getNumColumnsFromUser



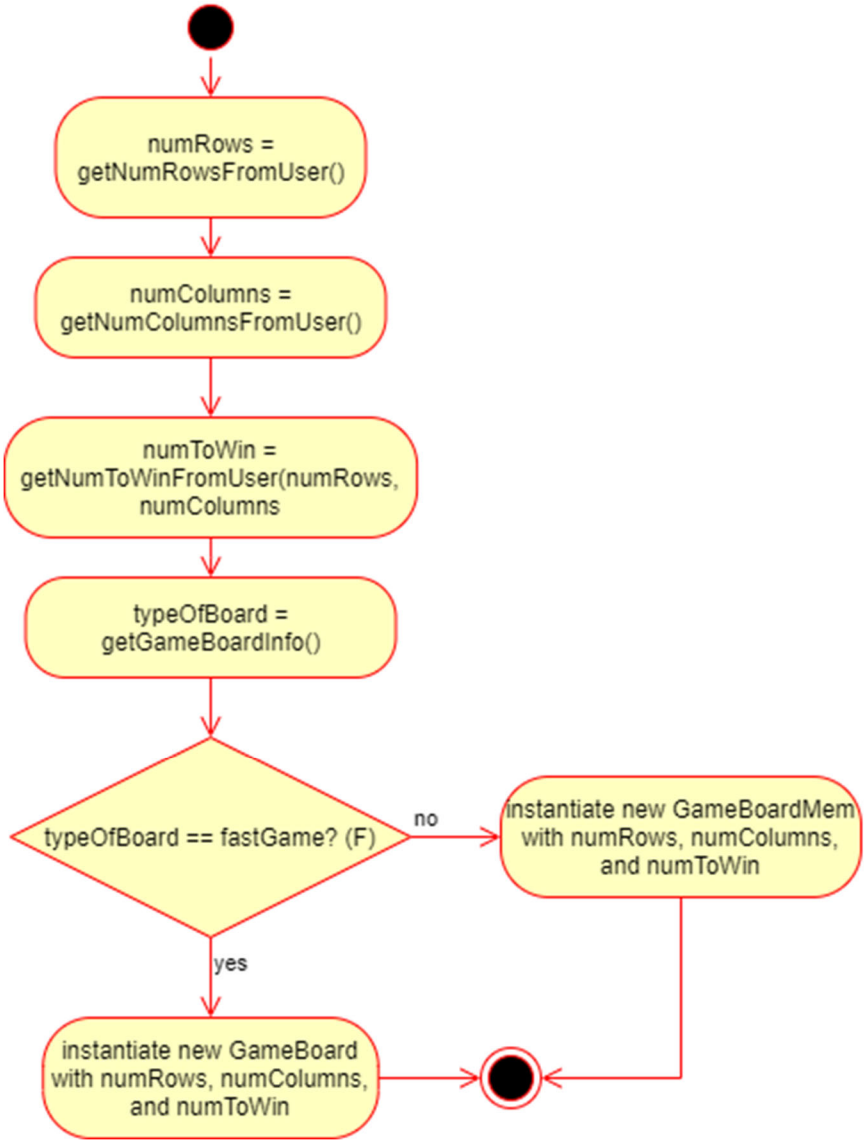
getNumToWinFromUser



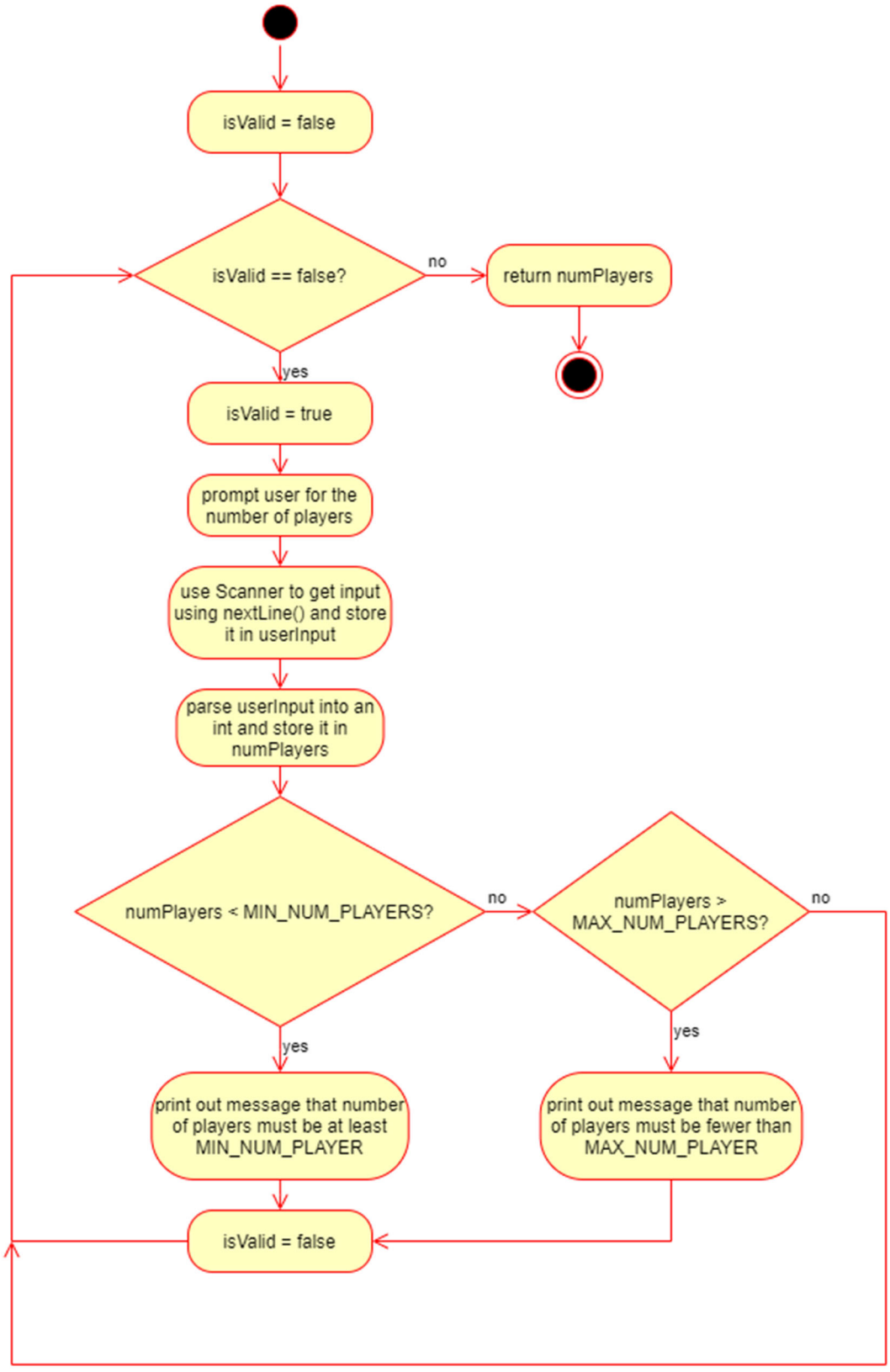
getGameBoardInfo



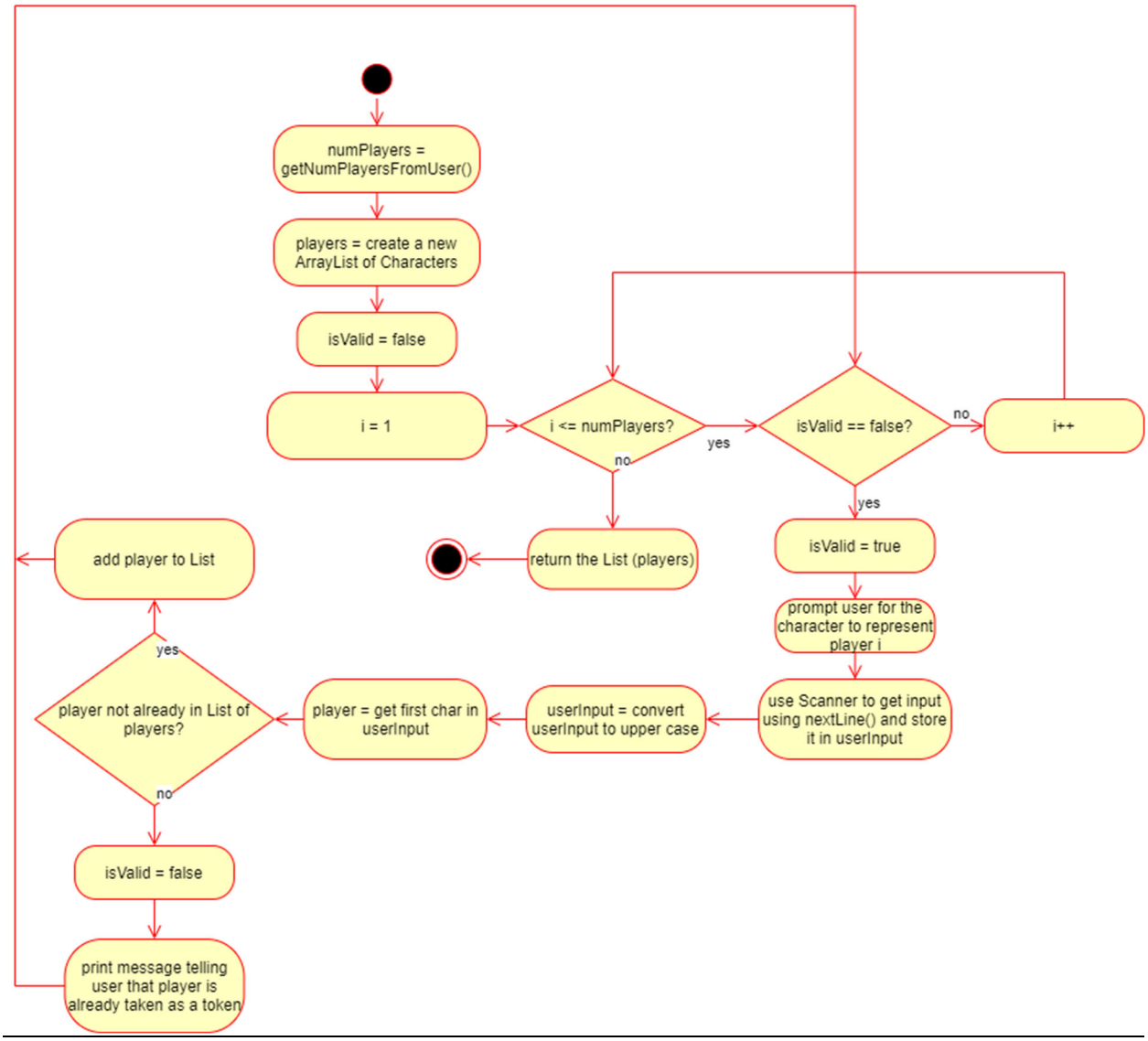
createGameBoard



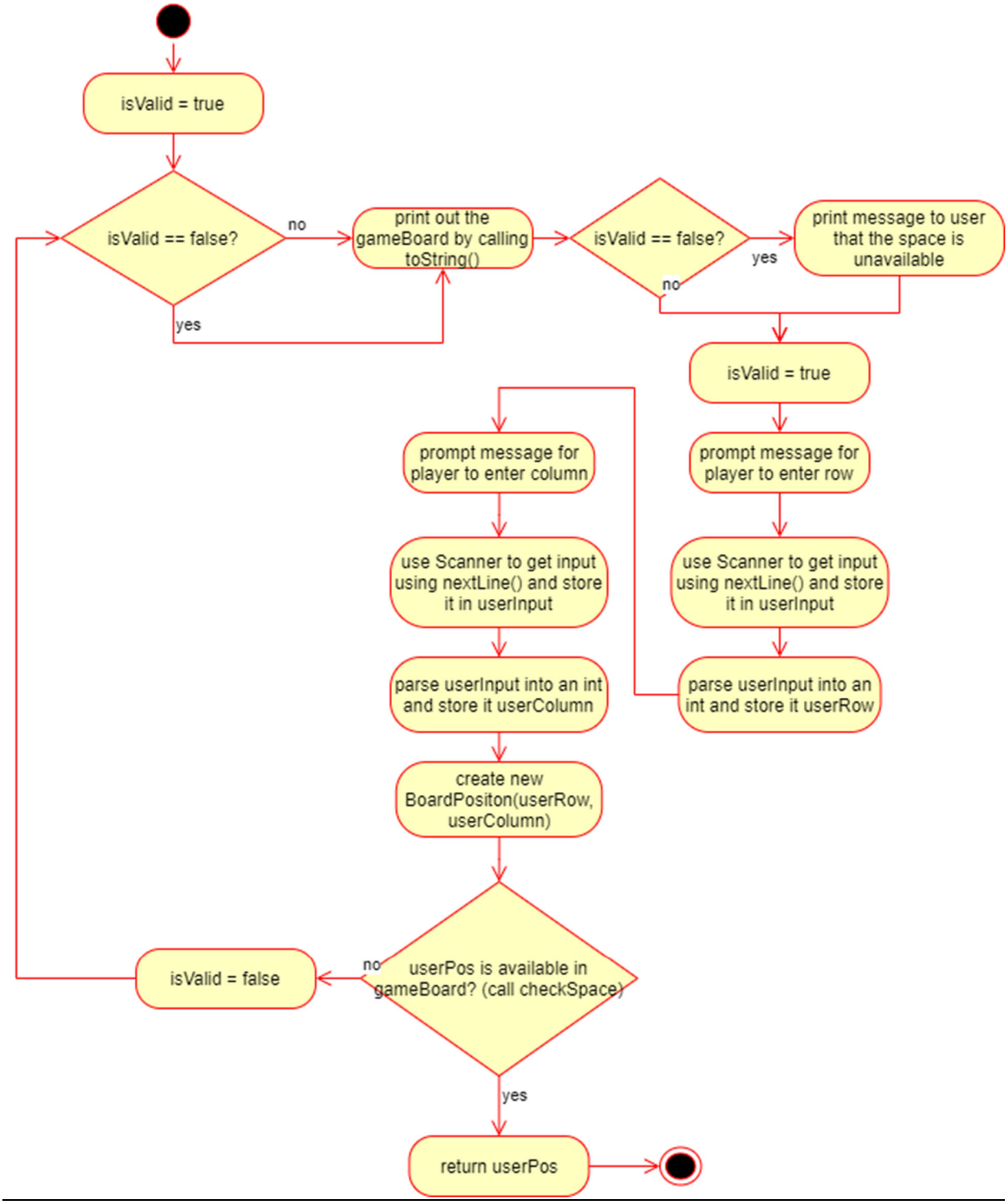
getNumPlayerfromUser



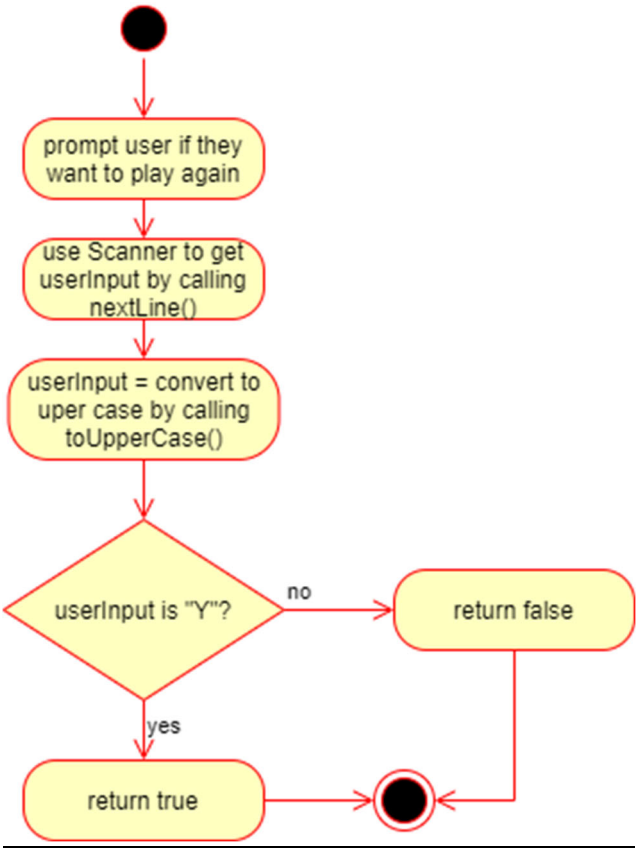
storePlayers



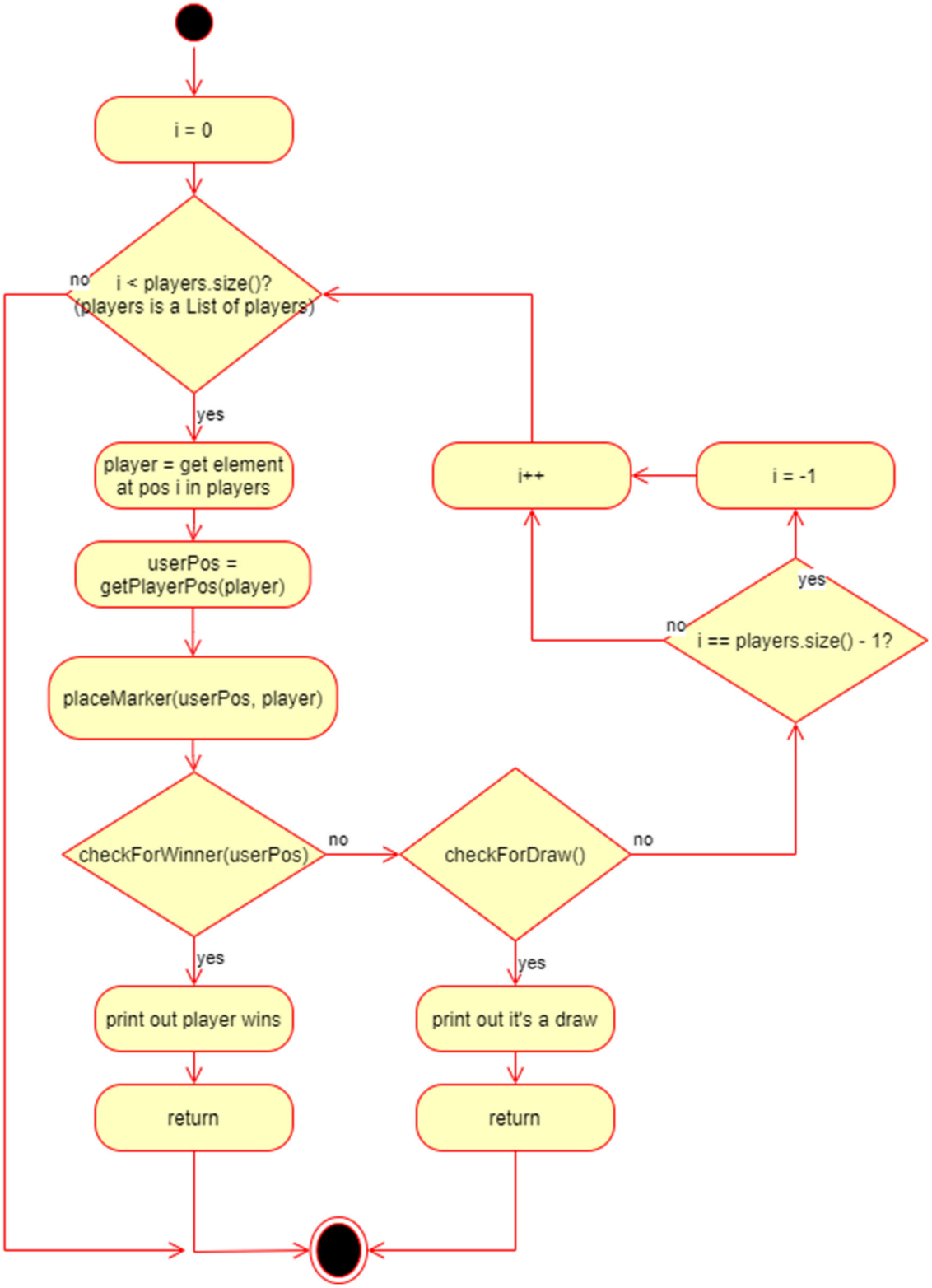
getPlayerPos



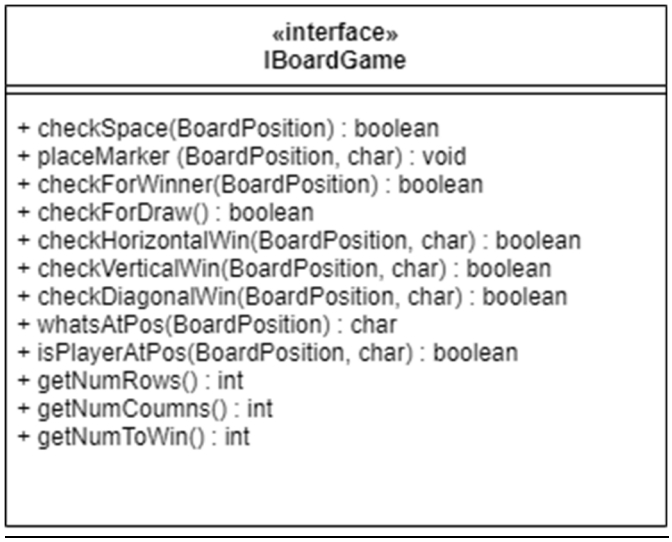
userInputPlayAgain



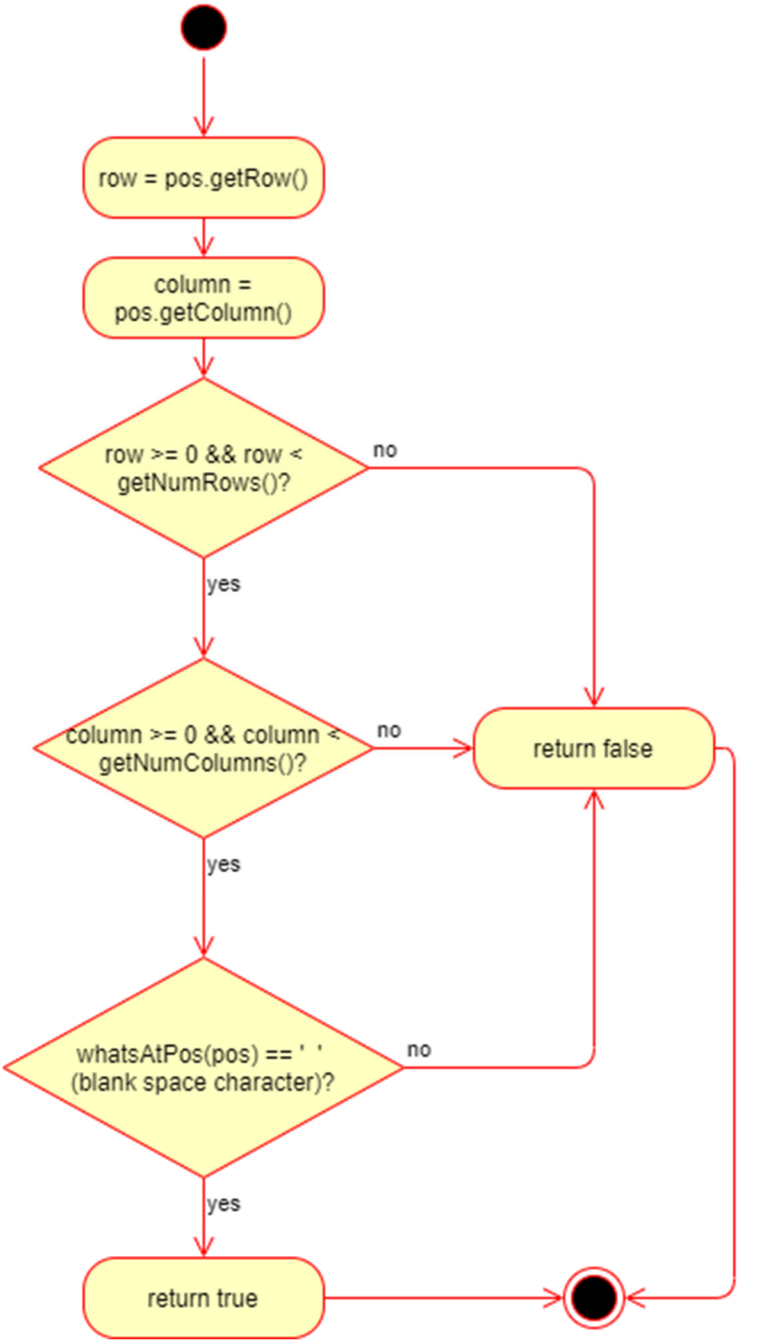
playGame


i = 0
i < players.size()?
(players is a List of players)
no
yes
player = get element at pos i in players
userPos = getPlayerPos(player)
placeMarker(userPos, player)
checkForWinner(userPos)
no
checkForDraw()
no
yes
yes
print out player wins
print out it's a draw
return
return
i++
i = -1
yes
no
i == players.size() - 1?

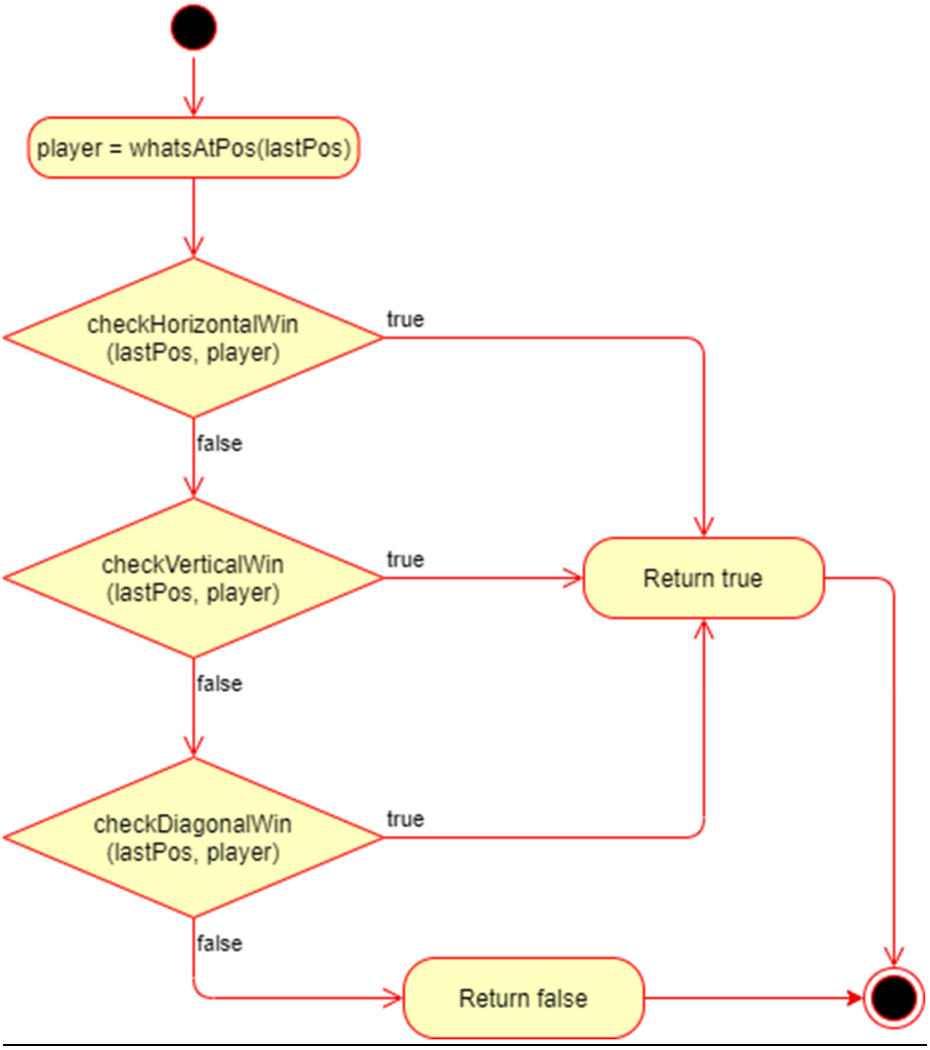
IGameBoard Interface



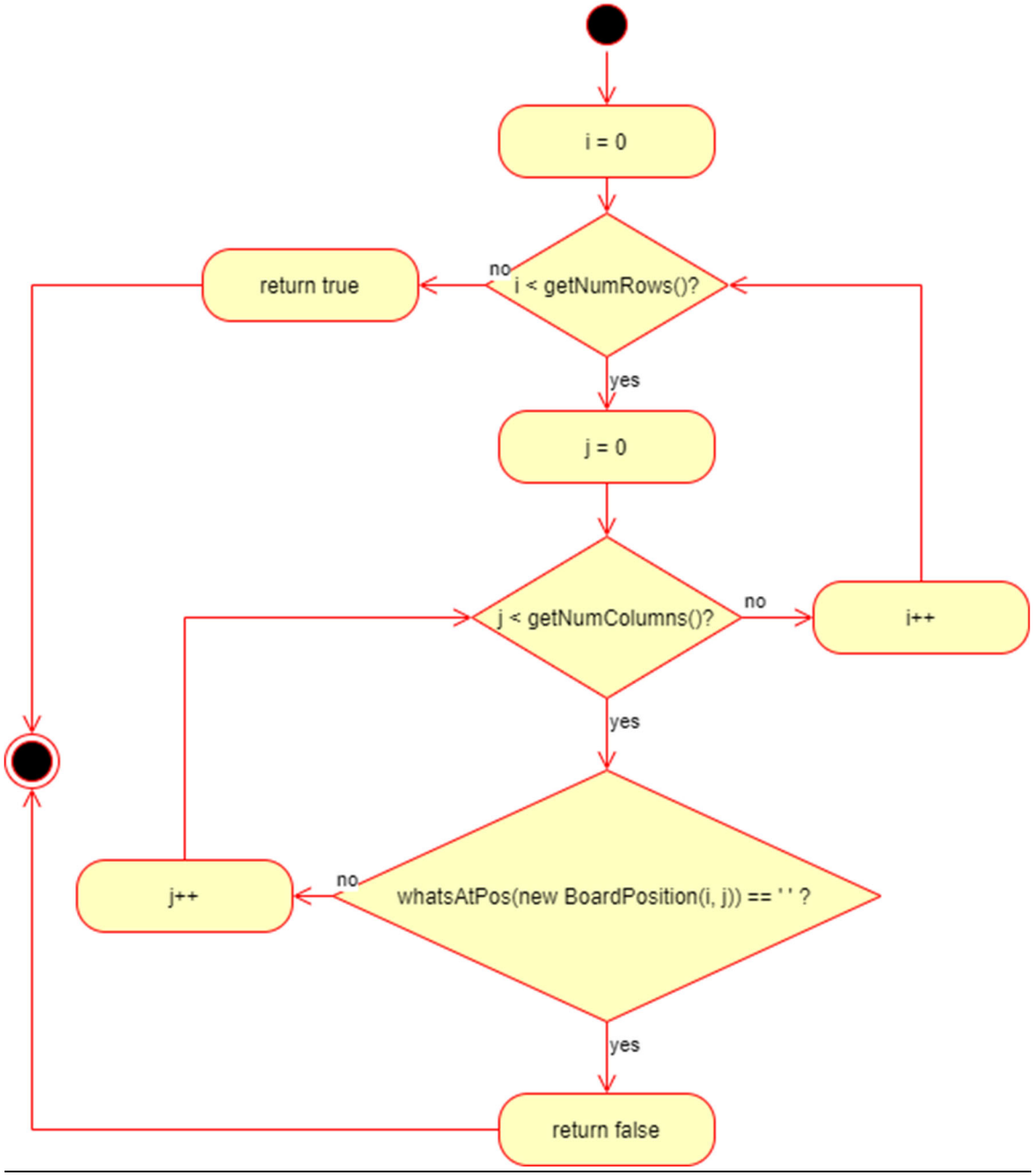
checkSpace


row = pos.getRow()
column = pos.getColumn()
row >= 0 && row < getNumRows()?
no
yes
column >= 0 && column < getNumColumns()?
no
return false
yes
whatsAtPos(pos) == ' ' (blank space character)?
no
yes
return true

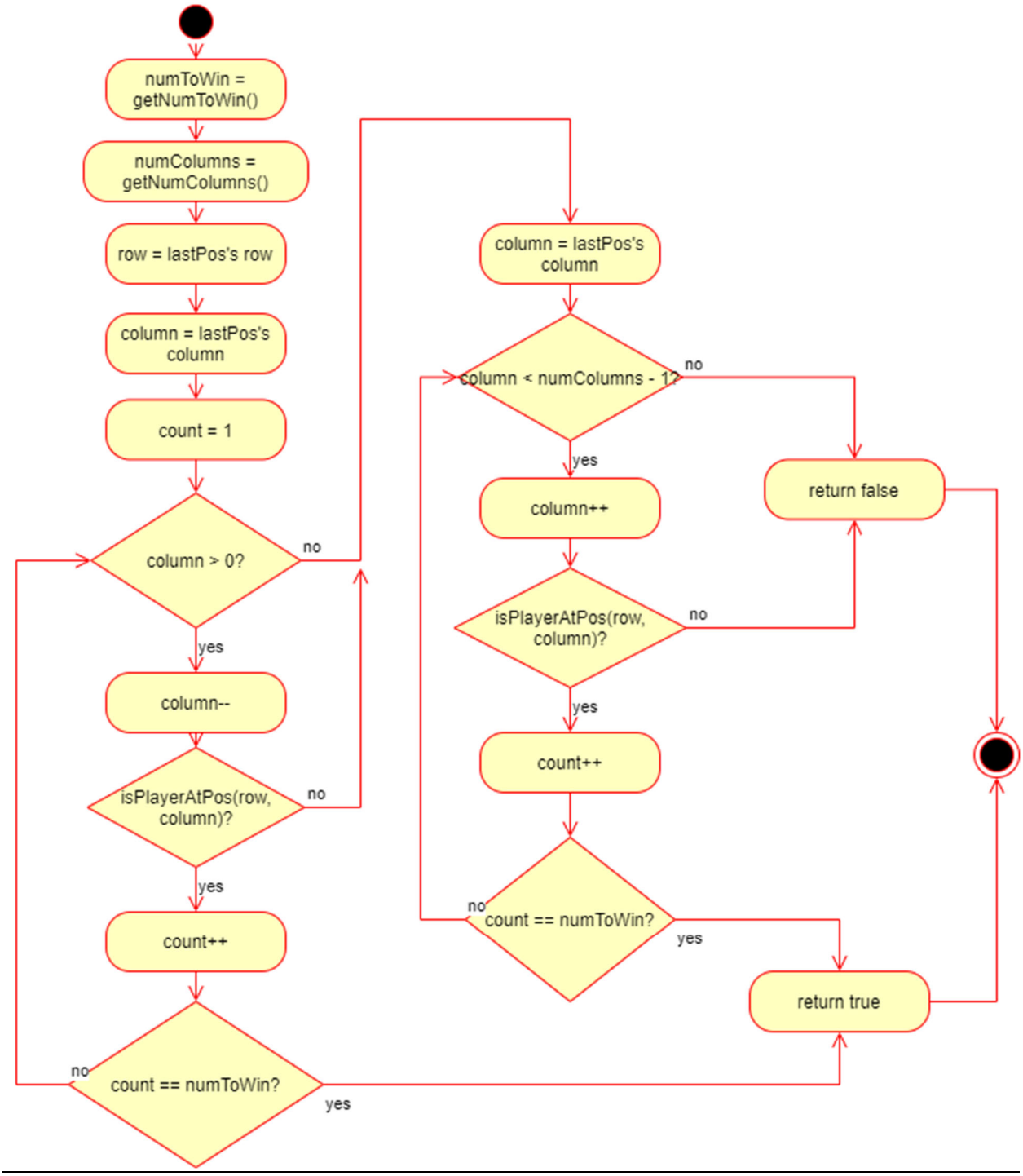
checkForWinner



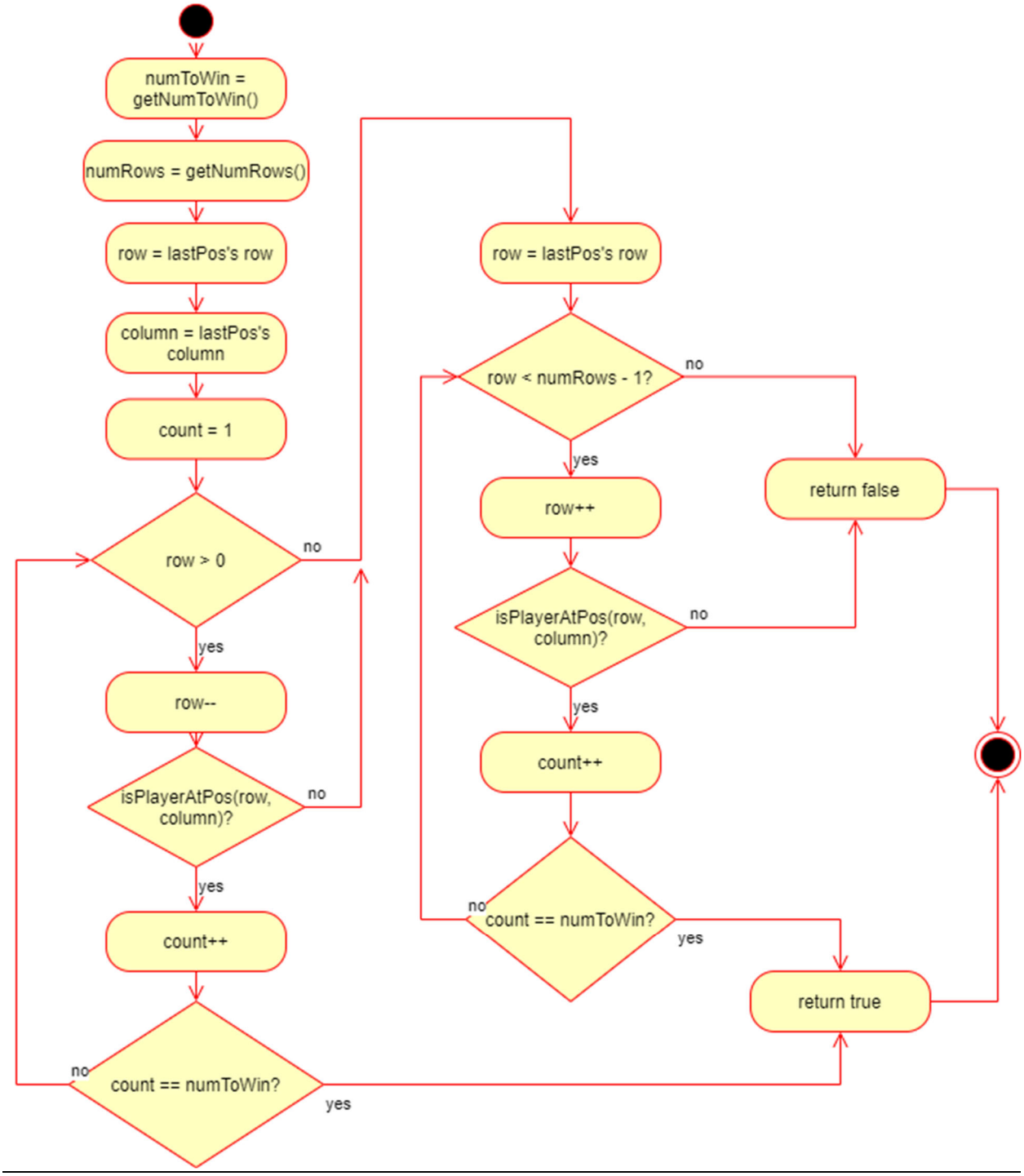
checkForDraw



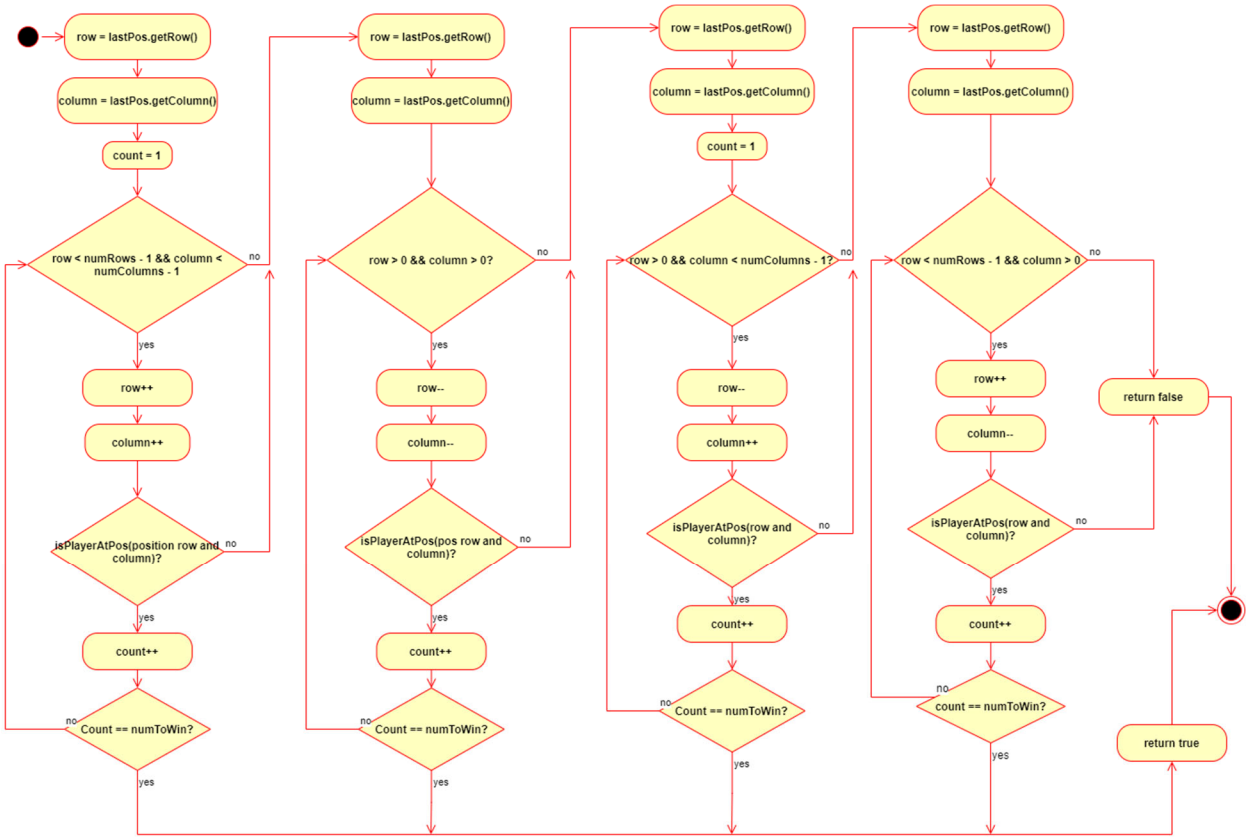
checkHorizontalWin



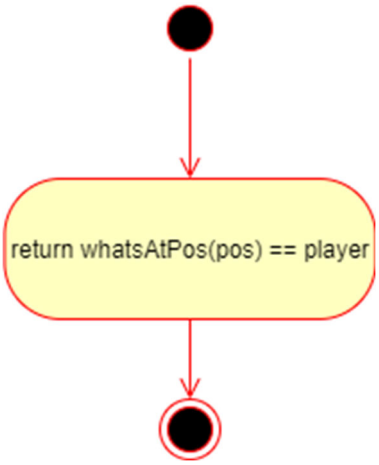
checkVerticalWin



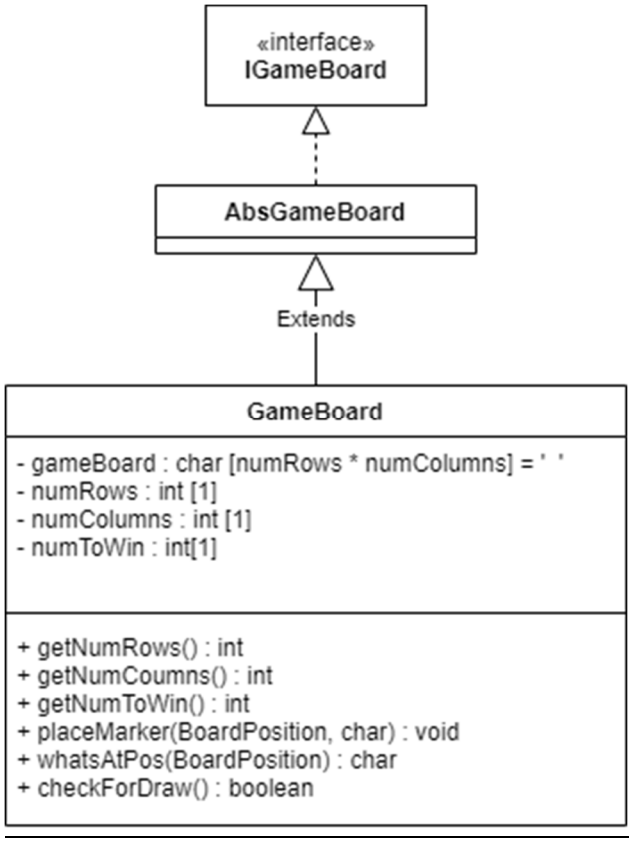
checkDiagonalWin



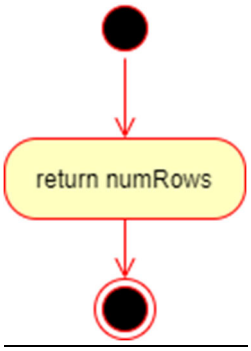
isPlayerAtPos



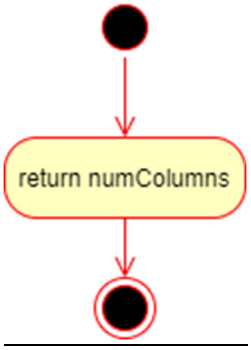
GameBoard Class



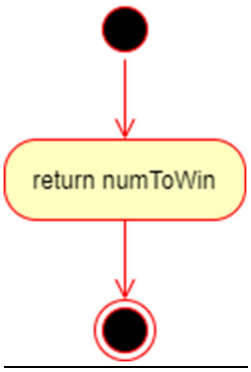
getNumRows



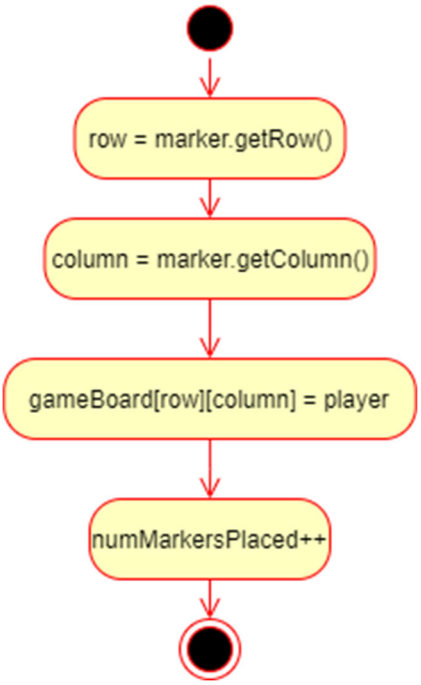
getNumColumns



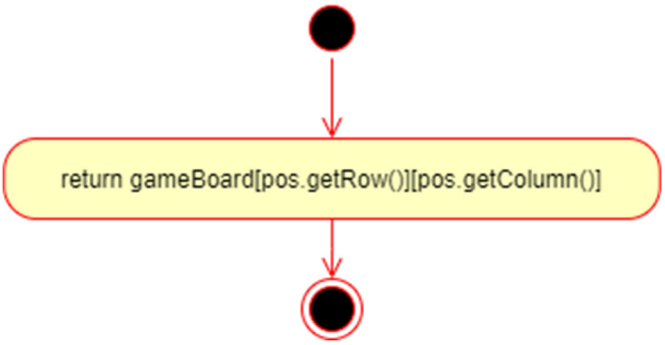
getNumToWin



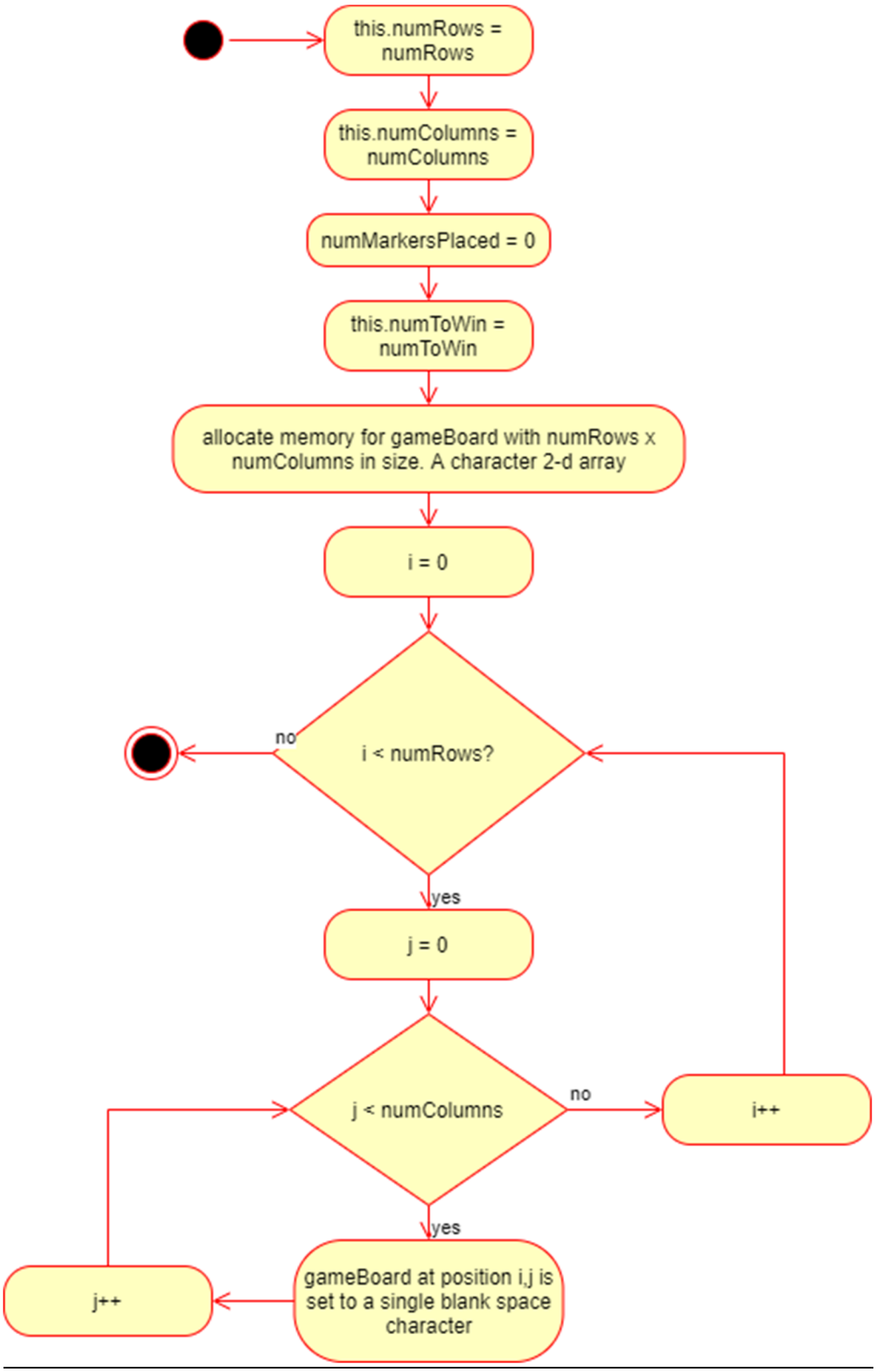
placeMarker



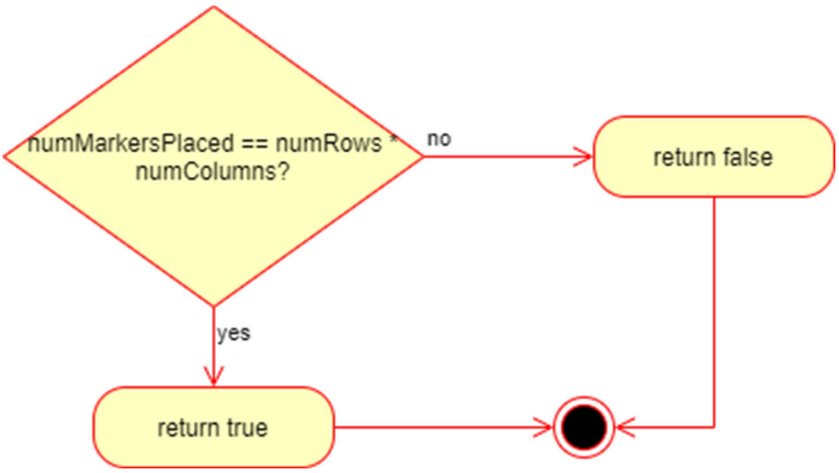
whatsAtPos



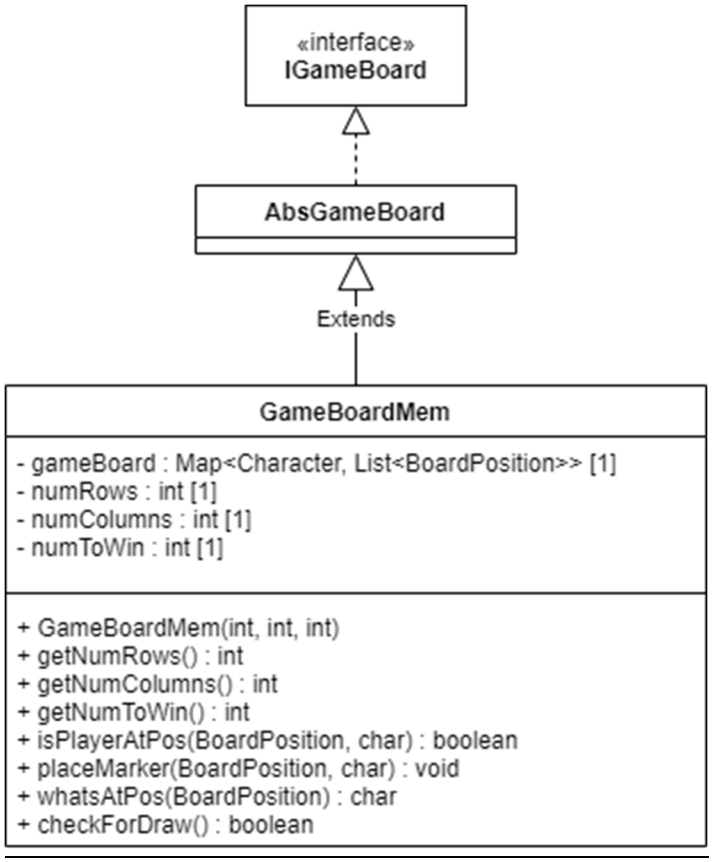
GameBoard (constructor)



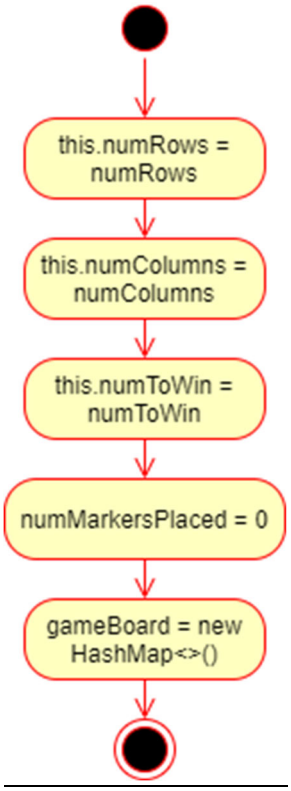
checkForDraw



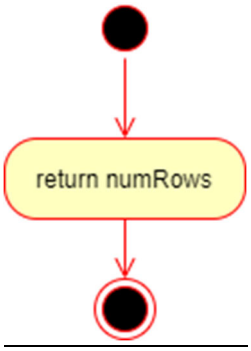
GameBoardMem Class



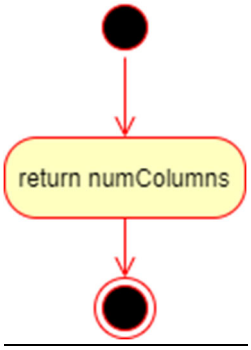
GameBoardMem (constructor)



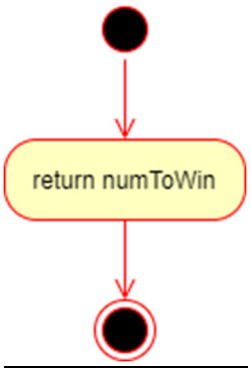
getNumRows



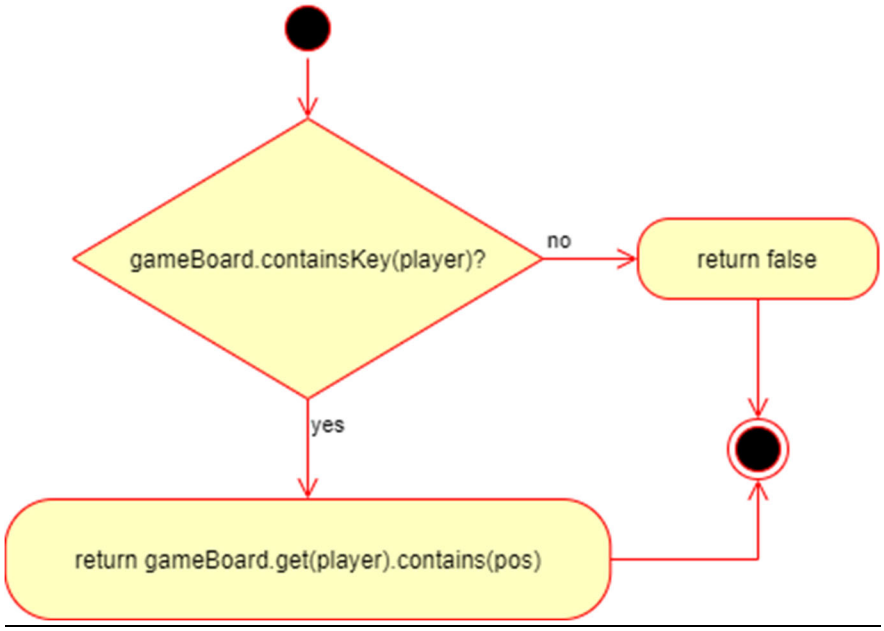
getNumColumns



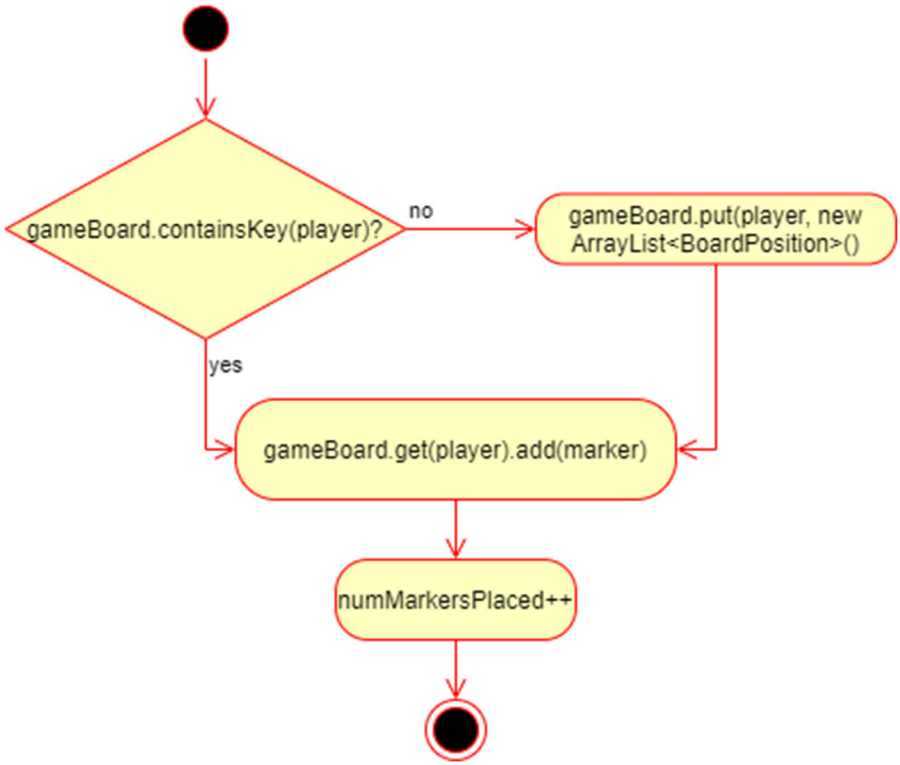
getNumToWin



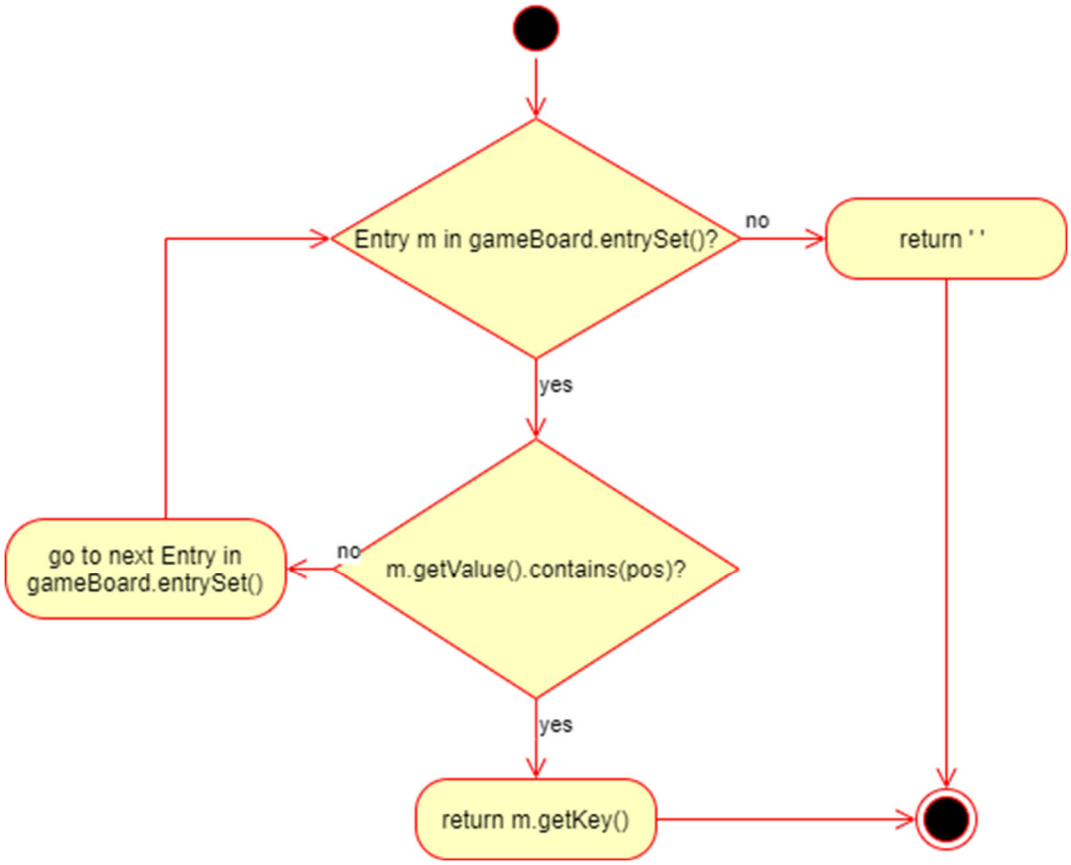
isPlayerAtPos



placeMarker

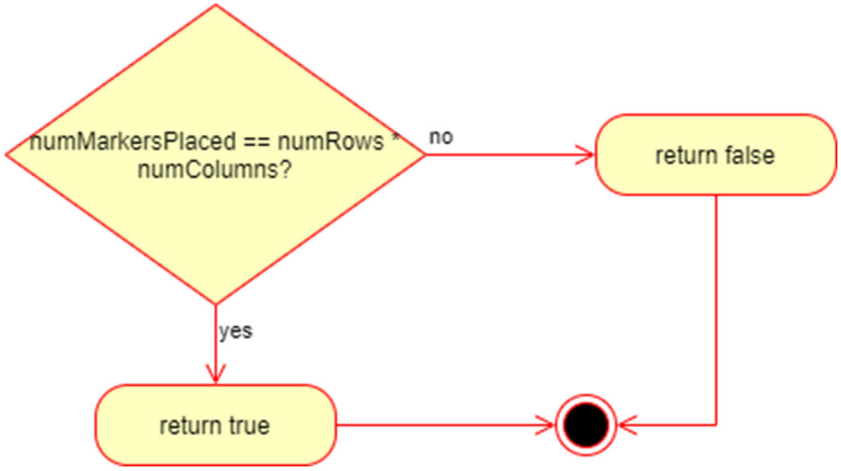


whatsAtPos



---

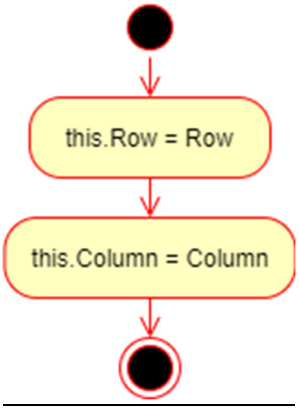
checkForDraw



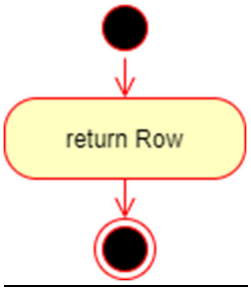
BoardPosition Class

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

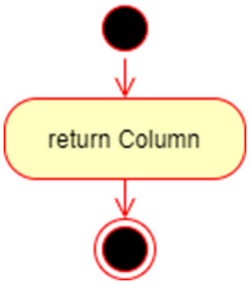
BoardPosition (constructor)



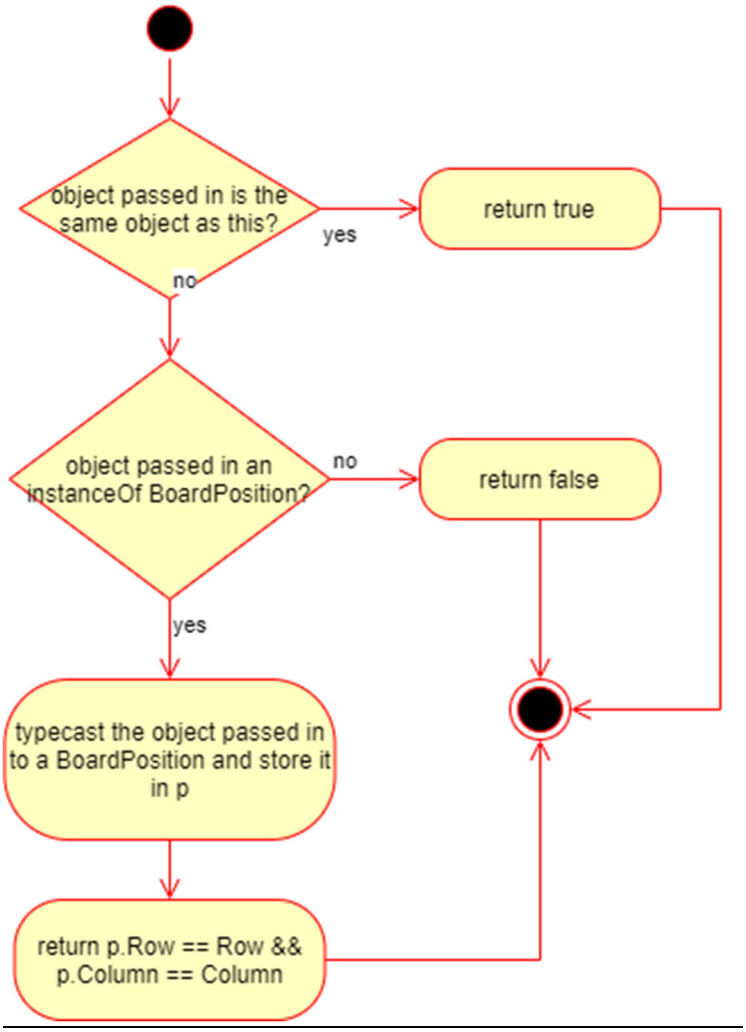
getRow



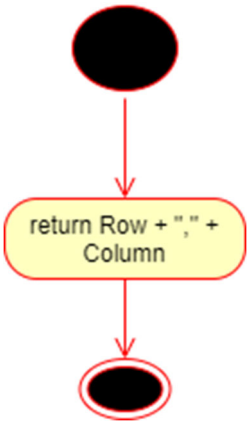
getColumn



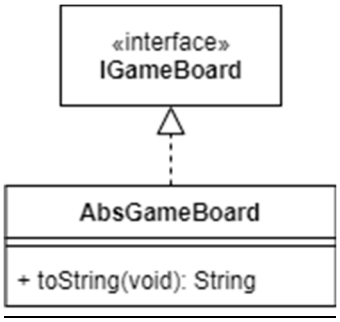
Equals



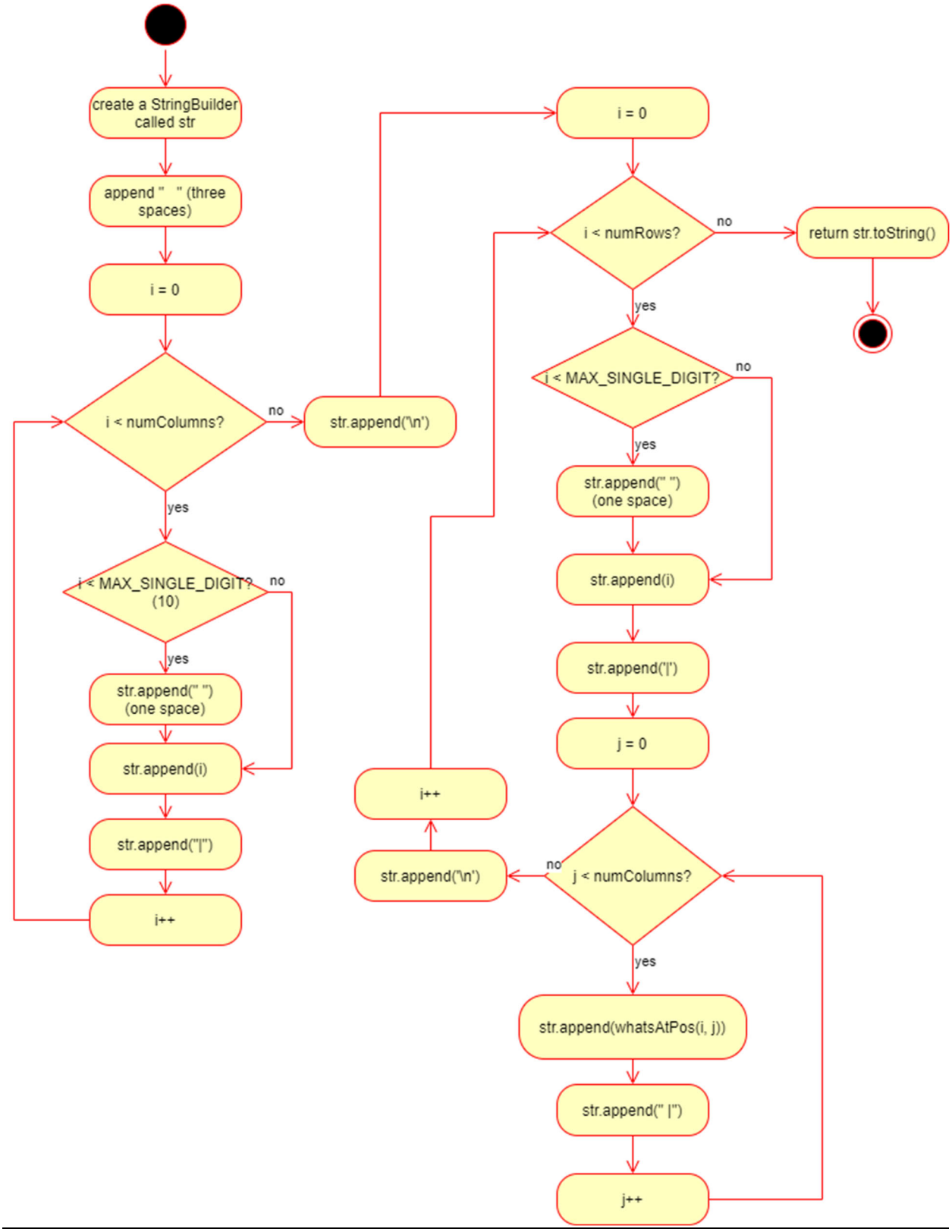
toString



AbsGameBoard Class



toString


create a StringBuilder called str
append " " (three spaces)
i = 0
i < numColumns?
no
yes
str.append('\n')
i < MAX_SINGLE_DIGIT? (10)
no
yes
str.append(" ") (one space)
str.append(i)
str.append("|")
i++
i = 0
i < numRows?
no
yes
return str.toString()
i < MAX_SINGLE_DIGIT?
no
yes
str.append(" ") (one space)
str.append(i)
str.append('|')
j = 0
j < numColumns?
no
yes
str.append('\n')
i++
str.append(whatsAtPos(i, j))
str.append(" |")
j++

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)

<table>
<tr>
<td>Input:

numRows = 5
numColumns = 8
numToWin = 5</td>
<td>Output:

State:
numRows = 5
numColumns = 8
numToWin = 5

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></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><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td>Reason:
This is a routine test case to see if the constructor can generate an empty 5x8 grid

Function Name:

testConstructor_routine</td>
</tr>
<tr>
<td>Input:

numRows = 3
numColumns = 3
numToWin = 3</td>
<td>Output:

State:
numRows = 3
numColumns = 3
numToWin = 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>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</td>
</tr>
<tr>
<td>Input:

numRows = 100
numColumns = 100
numToWin = 25</td>
<td>Output:

State:
numRows = 100
numColumns = 100
numToWin = 25

game board = [100 x 100 grid of empty spaces]

<table>
<tr><td></td><td>0</td><td>1</td><td>…</td><td>100</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>100</td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td>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</td>
</tr>
</table>

boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
pos.getRow = 1
pos.getColumn = 1</td>
<td>Output:

checkSpace = true

state of the board is unchanged</td>
<td>Reason:
This is a routine test case. Test on a valid (not out of bounds) and the space is not taken (blank space character).

Function Name:
testCheckSpace_
validBounds_spaceNotTaken</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
pos.getRow = 3
pos.getColumn = 3</td>
<td>Output:

checkSpace = false

state of the board is unchanged</td>
<td>Reason:
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

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

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>
pos.getRow = 1
pos.getColumn = 1</td>
<td>Output:

checkSpace = false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because I am testing for valid bounds, but space is already taken

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

boolean 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></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>

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

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because 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:
testCheckHorizontalWin_
win_lastMarkerMiddle</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table>
<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>

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

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X was placed at the right most end, so the function needs to count x’s just to the left of it

Function Name:
testCheckHorizontalWin_
win_lastMarkerRightEnd</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td></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>

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

checkHorizontalWin = false

state of the board is unchanged</td>
<td>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 to the left or right

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

<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>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>

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

checkHorizontalWin = false

state of the board is unchanged</td>
<td>Reason:
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

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

boolean 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>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>

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

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

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

State:
numToWin = 3
<table>
<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>

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

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X was placed at the bottom most end, so the function needs to count x’s just to the top of it

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

State:
numToWin = 3
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td></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>

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

checkVerticallWin = false

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

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

<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>0</td><td></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></td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td>O</td><td></td><td></td></tr>
</table>

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

checkVerticalWin = false

state of the board is unchanged</td>
<td>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</td>
</tr>
</table>

boolean 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>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>

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

checkDiagonalWin = true

state of board is unchanged</td>
<td>Reason:
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).

Function Name:
testCheckDiagonalWin_win_backslash_lastMarkerTopLeftEnd</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>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>

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

checkDiagonalWin = true

state of board is unchanged</td>
<td>Reason:
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)

Function Name:
testCheckDiagonalWin_win_backslash_lastMarkerBottomLeftEnd</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>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>

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

checkDiagonalWin = true

state of board is unchanged</td>
<td>Reason:
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

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

<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></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>
lastPos.getRow = 4
lastPos.getColumn = 0
player = ‘X’</td>
<td>Output:

checkDiagonalWin = true

state of board is unchanged</td>
<td>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)

Function Name:
testCheckDiagonalWin_win_ forwardSlash_ lastMarkerBottomLeftEnd</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></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>
lastPos.getRow = 1
lastPos.getColumn = 3
player = ‘X’</td>
<td>Output:

checkDiagonalWin = true

state of board is unchanged</td>
<td>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)

Function Name:
testCheckDiagonalWin_win_ forwardSlash_ lastMarkerTopRightEnd</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></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>
lastPos.getRow = 2
lastPos.getColumn = 2
player = ‘X’</td>
<td>Output:

checkDiagonalWin = true

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

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

<table>
<tr>
<td>Input:

State:
numToWin = 4
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td><td></td><td>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>

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

checkDiagonalWin = false

state of board is unchanged</td>
<td>Reason:
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

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

boolean checkForDraw()

<table>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>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></td>
<td>Output:

checkForDraw = true

state of board is unchanged</td>
<td>Reason:
This test case is unique and distinct because I am testing to see if a full grid would result in a draw

Function Name:
testCheckForDraw_draw_
fullGrid</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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></td>
<td>Output:

checkForDraw = false

state of board is unchanged</td>
<td>Reason:
This test case is unique and distinct because I am testing for boundary cases where there is only one marker on grid

Function Name:
testCheckForDraw_noDraw_
boundary_oneMarker</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>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></td>
<td>Output:

checkForDraw = false

state of board is unchanged</td>
<td>Reason:
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.

Function Name:
testCheckForDraw_noDraw_
_boundary_almostFull</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td>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></td>
<td>Output:

checkForDraw = false

state of board is unchanged</td>
<td>Reason:
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.

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

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
pos.getRow = 1
pos.getColumn = 0</td>
<td>Output:

whatsAtPos = ‘X’

state of board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks boundary case where column is 0 (minimum)

Function Name:

testWhatsAtPos_boundary_column_min</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
pos.getRow = 0
pos.getColumn = 1</td>
<td>Output:

whatsAtPos = ‘X’

state of board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks boundary case where row is 0 (minimum)

Function Name:

testWhatsAtPos_boundary_row_min</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
pos.getRow = 1
pos.getColumn = 2</td>
<td>Output:

whatsAtPos = ‘X’

state of board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks boundary case where column is at the max

Function Name:

testWhatsAtPos_boundary_column_max</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
pos.getRow = 1
pos.getColumn = 2</td>
<td>Output:

whatsAtPos = ‘X’

state of board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it checks boundary case where the row is at the max

Function Name:WE testWhatsAtPos_boundary_ row_max</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>
pos.getRow = 1
pos.getColumn = 1</td>
<td>Output:

whatsAtPos = ‘X’

state of board is unchanged</td>
<td>Reason:
This test case is unique and distinct because this is a routine test case where it is not in any of the boundaries

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

boolean isPlayerAtPos(BoardPosition pos, char Player)

<table>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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 = ‘O’
pos.getRow = 1
pos.getColumn = 1</td>
<td>Output:

isPlayerAtPos = false

state of the board is unchanged</td>
<td>Reason:
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

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

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
Player = ‘X’
pos.getRow = 1
pos.getColumn = 0</td>
<td>Output:

isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a boundary case where column is at min (o)

Function Name:
testIsPlayerAtPos_true_
boundary_column_min</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
Player = ‘X’
pos.getRow = 0
pos.getColumn = 1</td>
<td>Output:

isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a boundary case where row is at min (0)

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

<table>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
Player = ‘X’
pos.getRow = 1
pos.getColumn = 2</td>
<td>Output:

isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a boundary case where column is at max

Function Name:
testIsPlayerAtPos_true_
boundary_column_max</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
Player = ‘X’
pos.getRow = 1
pos.getColumn = 2</td>
<td>Output:

isPlayerAtPos = true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a boundary case where row is at max

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

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
marker.getRow = 1
marker.getColumn = 1
player = ‘A’</td>
<td>Output:

State:
numToWin = 3
<table><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></td>
<td>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

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

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
marker.getRow = 0
marker.getColumn = 1
player = ‘X’</td>
<td>Output:

State:
numToWin = 3
<table><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></td>
<td>Reason:
This test case is unique and distinct because I am placing a marker at a boundary case where row is at min

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

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
marker.getRow = 1
marker.getColumn = 2
player = ‘X’</td>
<td>Output:

State:
numToWin = 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>X</td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
This test case is unique and distinct because I am placing a marker at a boundary case where column is at max

Function name:
testPlaceMarker_boundary_column_max</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
marker.getRow = 2
marker.getColumn = 1
player = ‘X’</td>
<td>Output:

State:
numToWin = 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>X</td><td></td></tr></table></td>
<td>Reason:
This test case is unique and distinct because I am placing a marker at a boundary case where row is at max

Function name:
testPlaceMarker_boundary_row_max</td>
</tr>
<tr>
<td>Input:

State:
numToWin = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></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>
marker.getRow = 1
marker.getColumn = 0
player = ‘X’</td>
<td>Output:

State:
numToWin = 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>X</td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
This test case is unique and distinct because I am placing a marker at a boundary case where column is at min

Function name:
testPlaceMarker_boundary_column_min</td>
</tr>
</table>