Aika Washington

CPSC 2951

Requirements Analysis

User Stories:

1. As a player, I can place my token in a row to complete my turn.
2. As a player, I can view the board to determine the state of the game.
3. As a player, I can choose to play again to restart the game.
4. As a player, I can see a message that I have won by winning the game.
5. As a player, I can see a message that the game has tied by tieing the game.
6. As a player, I can see an error message prompting me to choose again if I choose a full or nonexistent column to place my token.
7. As a player, I can choose between a fast or efficient implementation.
8. As players, we can choose our characters.
9. As a player, I can win diagonally.
10. As a player, I can win horizontally.
11. As a player, I can win vertically.
12. As a player, I can make another move if no one else wins or there is no tie.

Nonfunctional requirements:

1. Connect X should be written in Java.
2. Connect X should be able to run on Unix.
3. Connect X first player character always goes first.
4. 0, 0 is always the bottom right of the board.
5. The bord can only be the max size x max size.

Deployment instructions:

The makefile should be placed at the same directory level as the cpsc2150 folder.

To compile, open the terminal at the level of the makefile and type make and press enter.

To run, type make run and press enter.

To run tests on GameBoard, type testGB and enter.

To run tests on GameBoardMem, type testGBMem and enter.

To clean the class files, type make clean and press enter.

GameScreen

- playerX: char [1]
- playerO: char [1]
- playerTurn: boolean [1]
- gameInPlay: boolean [1]
- lastCol: int [1]

+ getColumn(int): void
+ checkWin(void): void
+ checkForTie(void): void
+ placeMarker(void): void
+ playAgain(void): boolean
+ main(String[ ] args): void
+ printBoard(void): void

GameBoard

- connectBoard char[*][*]
- height: int[1]
- width: : int[1]
- win_num: int[1]

+ GameBoard(int, int, int) : GameBoard
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ toString(): String
+ getNumRows(); int
+ getNumColumns() : int
+ getNumToWin(): int

GameBoardMem

- connectBoard char[*][*]
- height: int[1]
- width: : int[1]
- win_num: int[1]
- gameMap: Map<Character, List<BoardPosition>>

+ GameBoard(int, int, int) : GameBoard
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ toString(): String
+ getNumRows(); int
+ getNumColumns() : int
+ getNumToWin(): int

AbsGameBoard

+toString: String

GameScreen

- playerX: char [1]
- playerO: char [1]
- playerTurn: boolean [1]
- gameInPlay: boolean [1]
- lastCol: int [1]

+ getColumn(int): void
+ checkWin(void): void
+ checkForTie(void): void
+ placeMarker(void): void
+ playAgain(void): boolean
+ main(String[ ] args): void
+ printBoard(void): void

iGameBoard

+ MAX_SIZE: int [1]
+ MIN_SIZE: int[1]
+ MAX_WIN_NUM: int[1]
+ MAX_PLAYER: : int[1]
+ MIN_PLAYER: int[1]

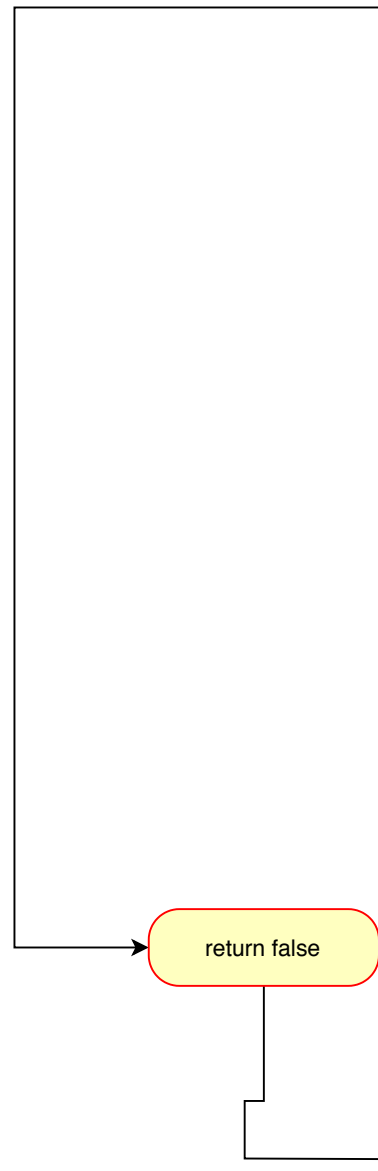
+ checkIfFree(int): boolean
+ checkForWin(int): boolean
+ placeToken(char, int): void
+ checkHorizWin(BoardPosition, char): bc
+ checkVertWin(BoardPostition, char): bo
+ checkDiagWin(BoardPostition, char): bc
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boc
+ toString(): String
+ checkTie(): boolean
+ getNumRows(); int
+ getNumColumns() : int
+ getNumToWin(): int

Use

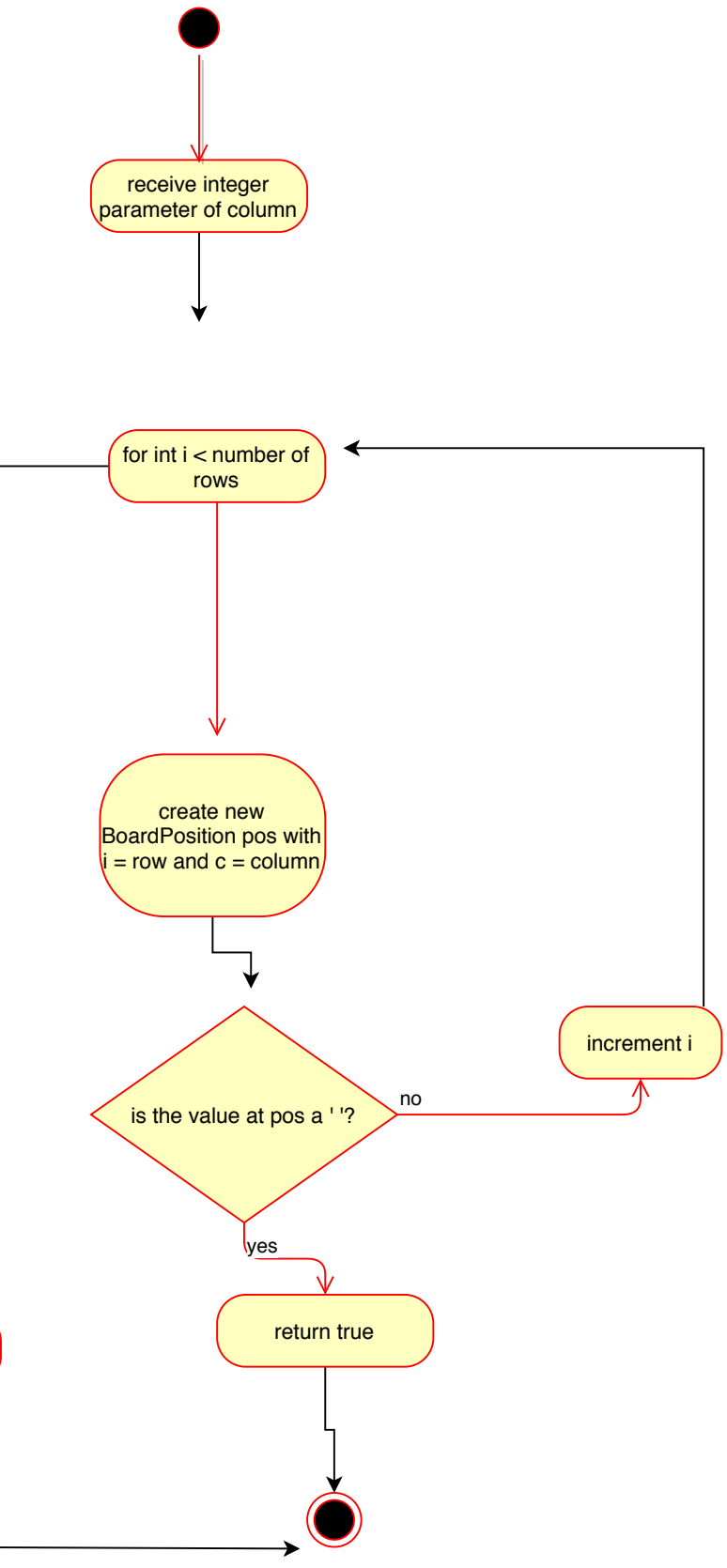
BoardPosition

- column: int [1]
- row: int [1]

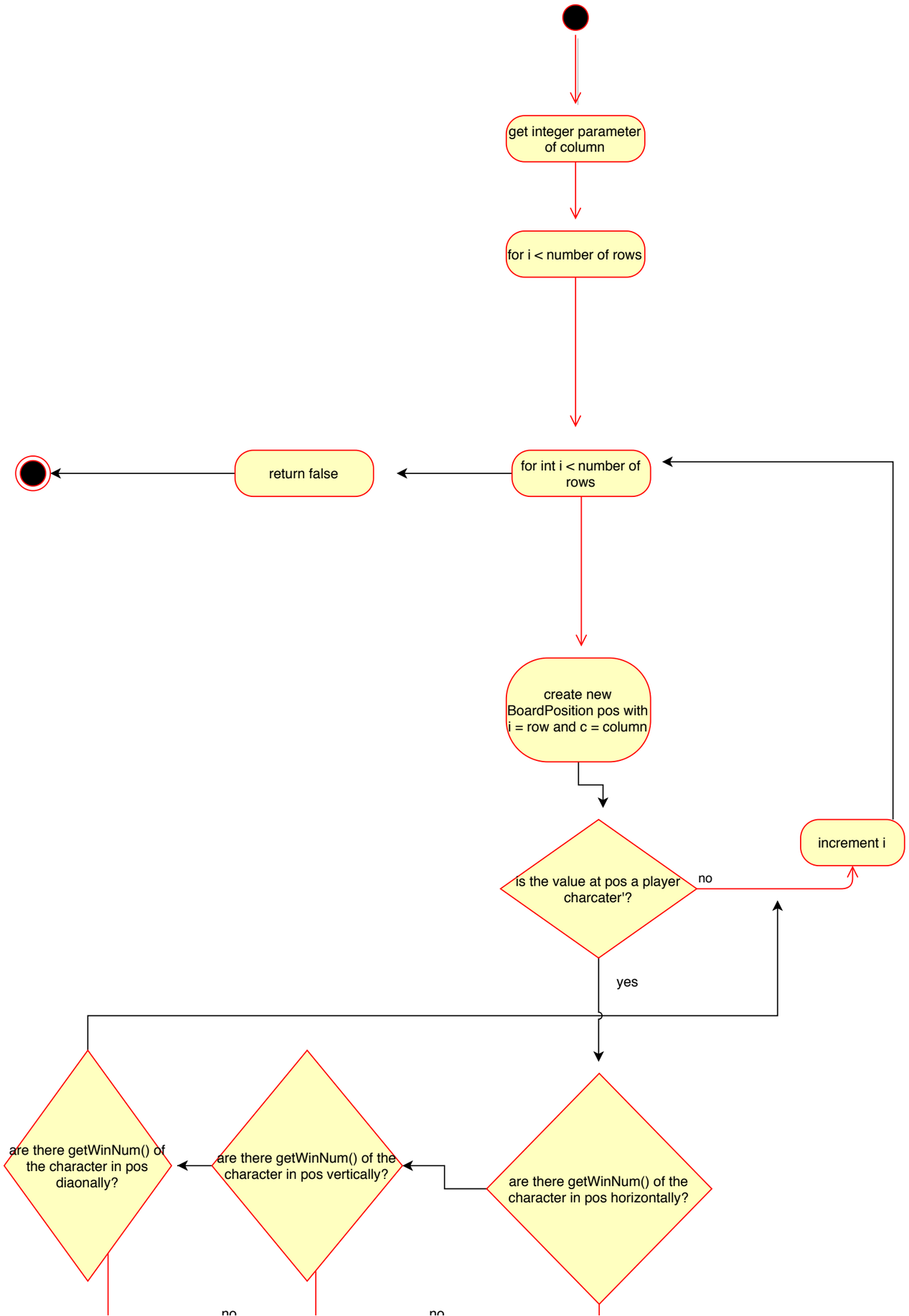
+ BoardPosition(int, int)
+ getRow(void): int
+ getColumn(void): int
+ equals(BoardPosition): boolean
+ toString(void): String

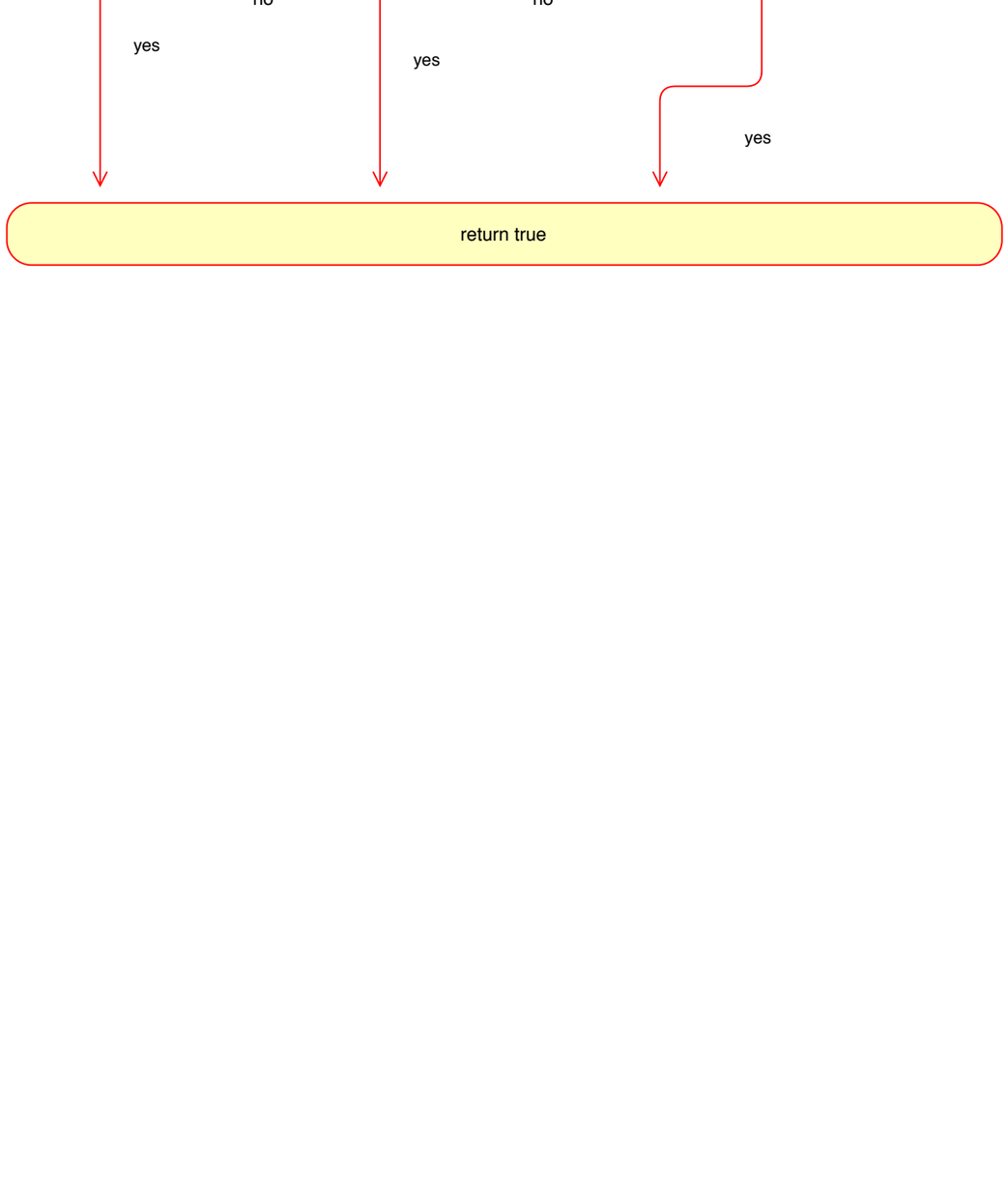

return false

public boolean checkIfFree(int c)

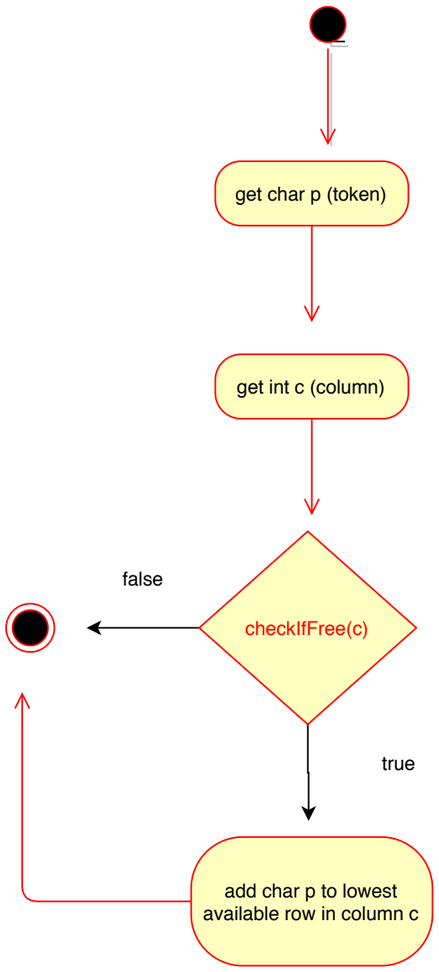


public boolean checkForWin(int c)

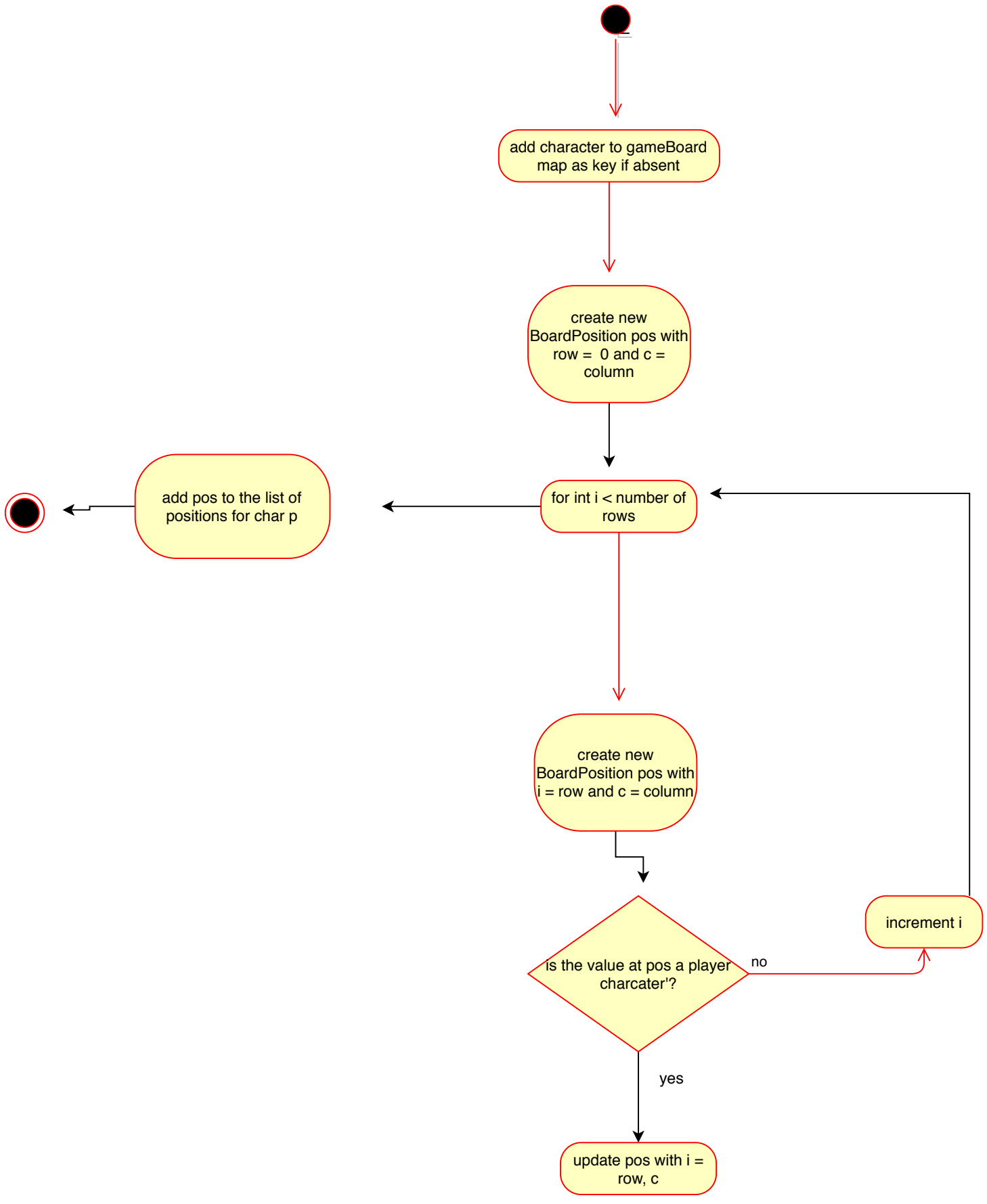



no
no
yes
yes
yes
return true

GameBoard: public void placeToken(char p, int c)

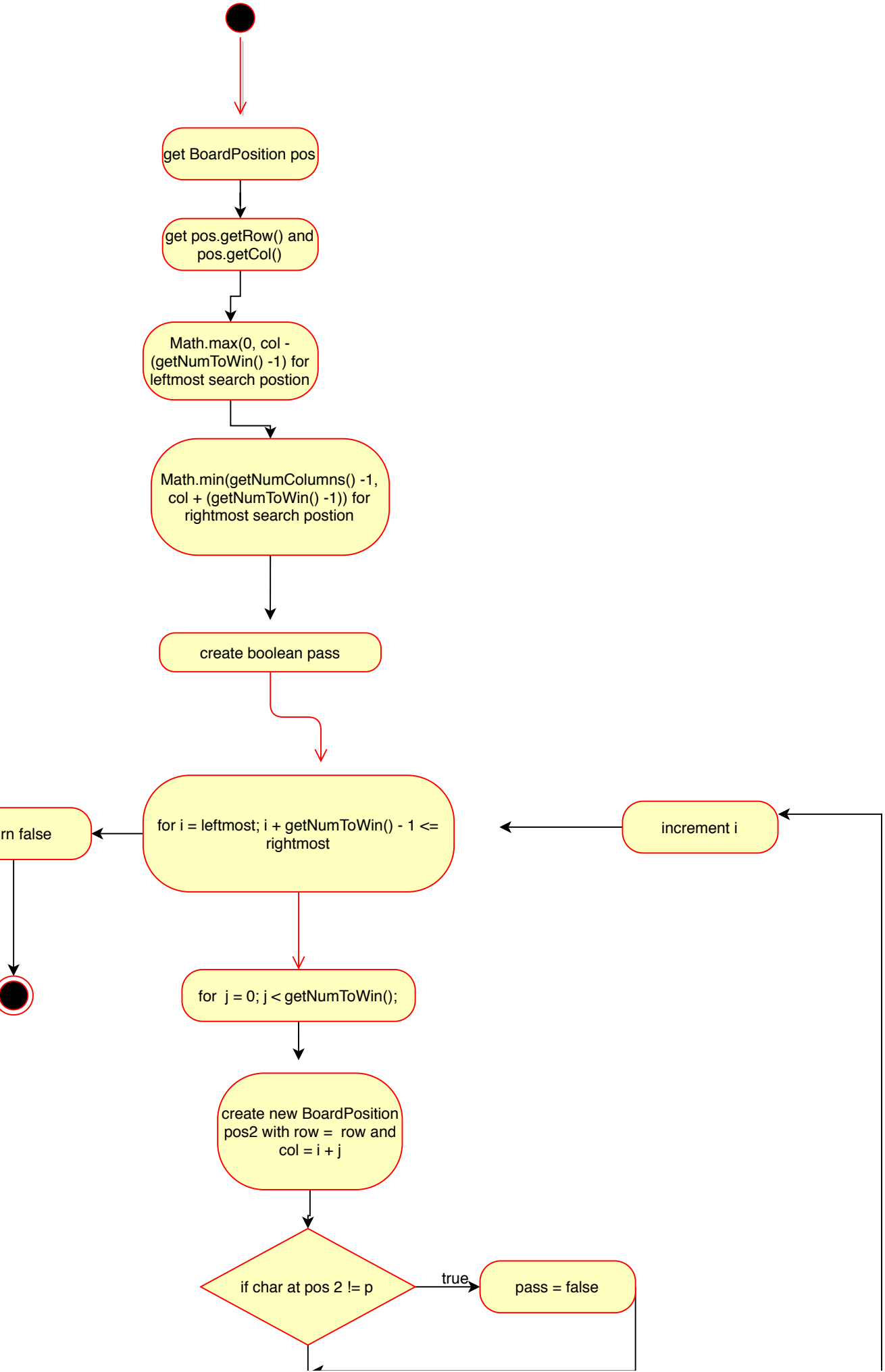


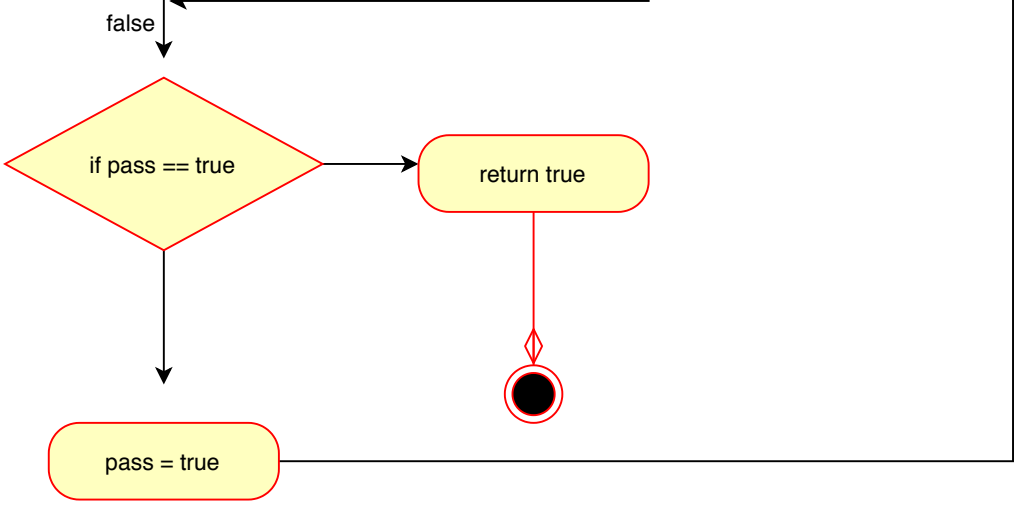
GameBoardMem: public void placeToken(char p, int c)

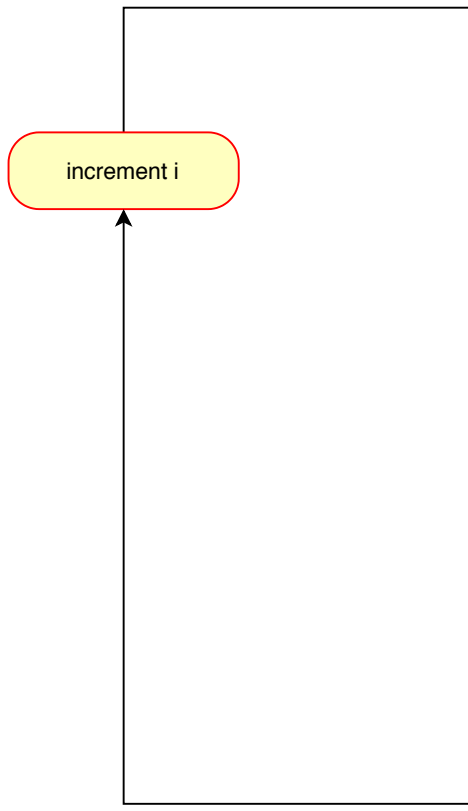


retu

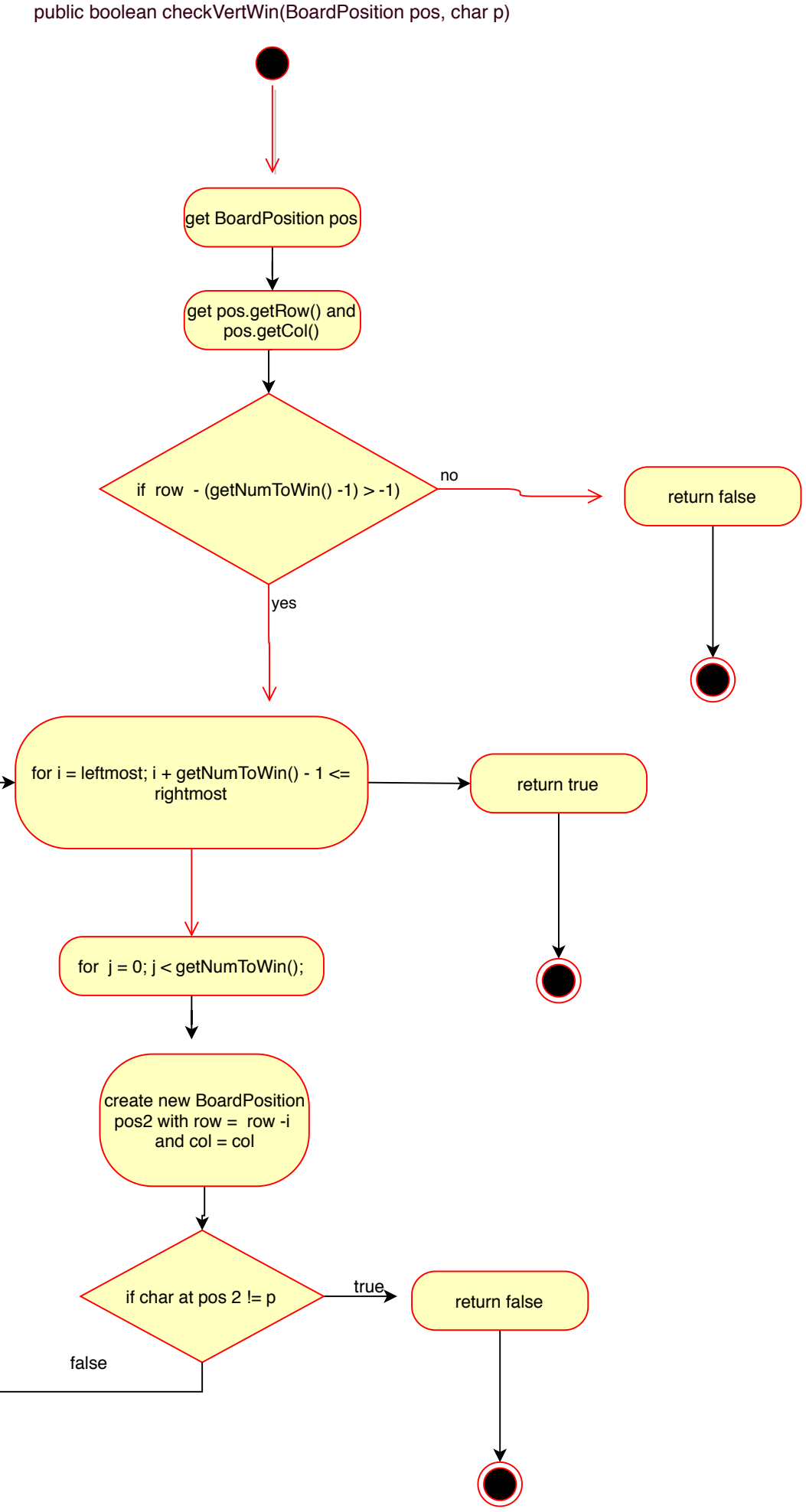
public boolean checkHorizWin(BoardPosition pos, char p)



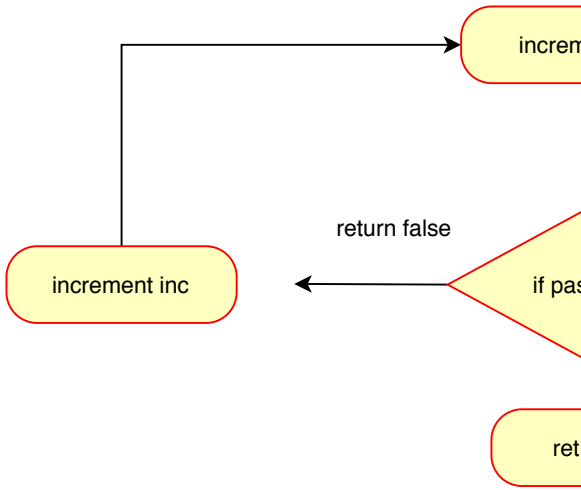

false
if pass == true
return true
pass = true

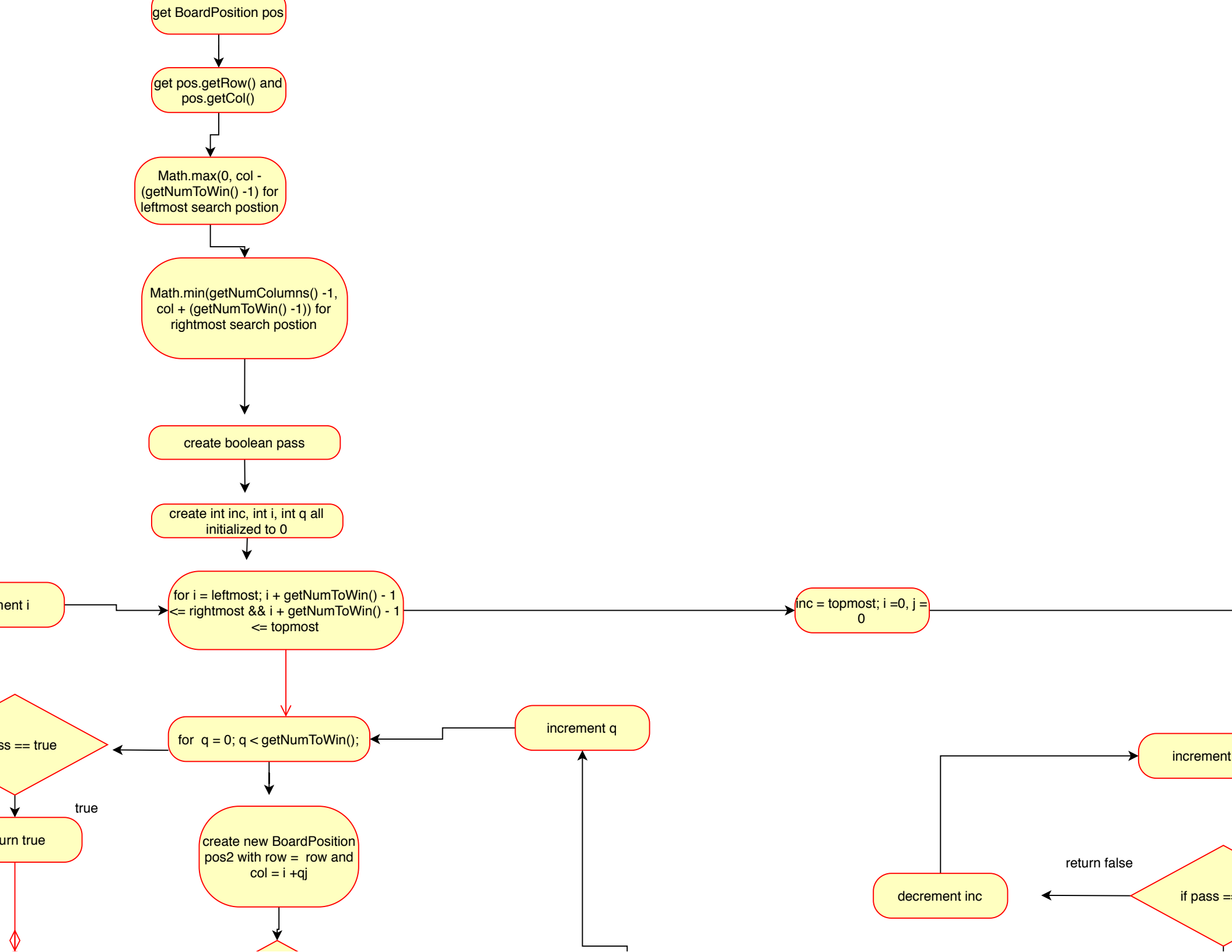

increment i

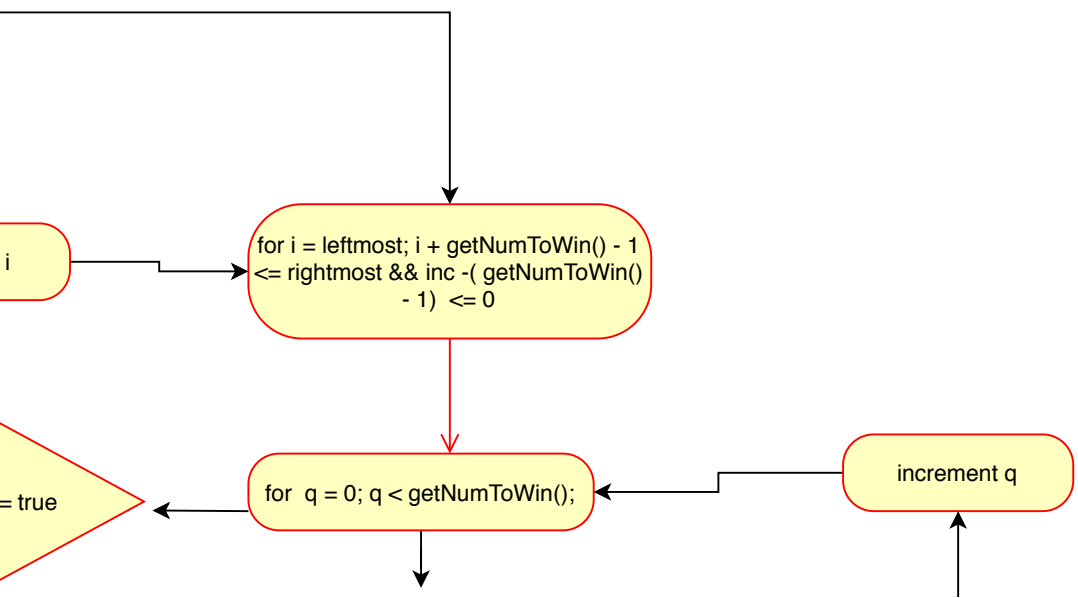
public boolean checkVertWin(BoardPosition pos, char p)

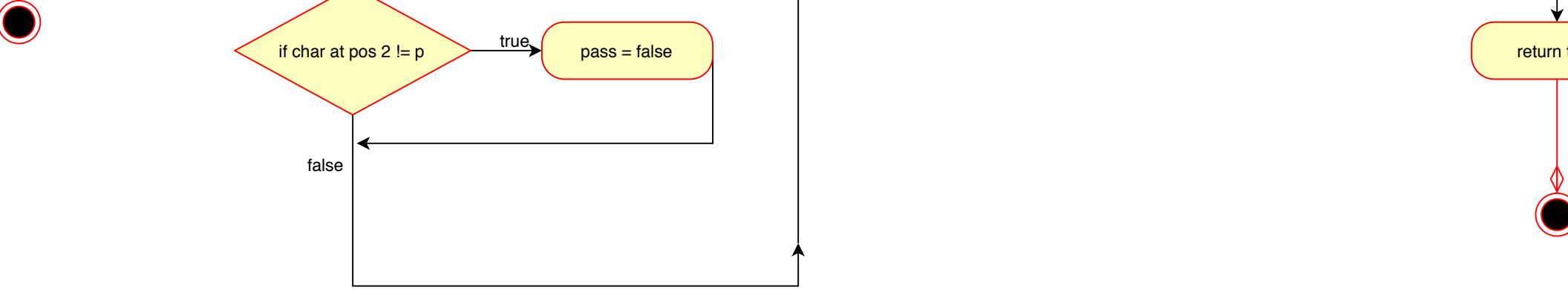


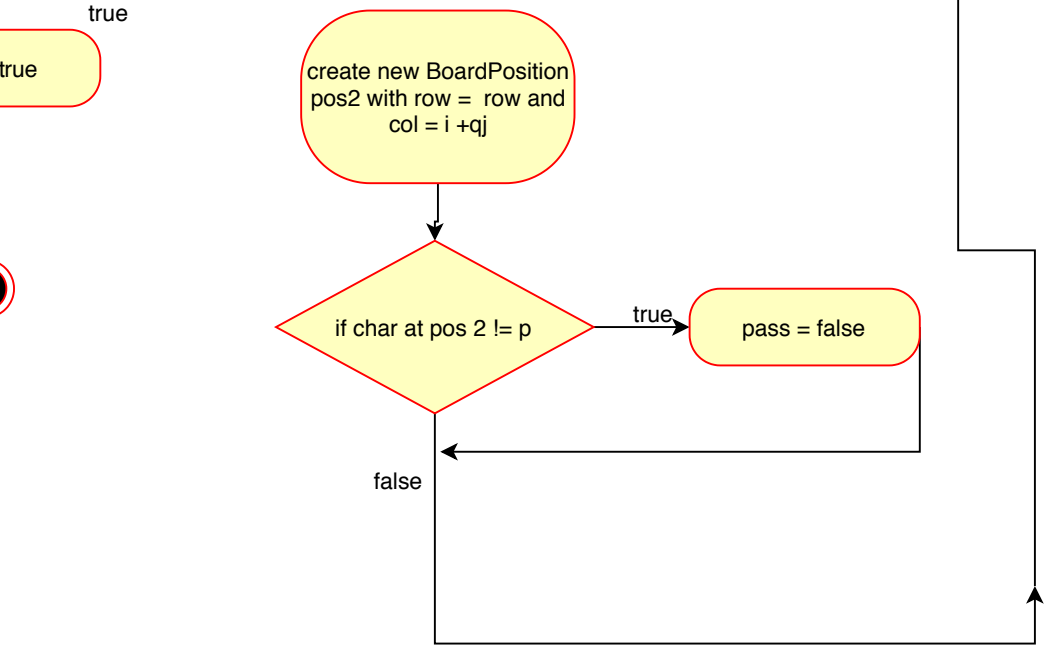
public boolean checkDiagWin(BoardPosition pos, char p)


increm
return false
increment inc
if pas
ret

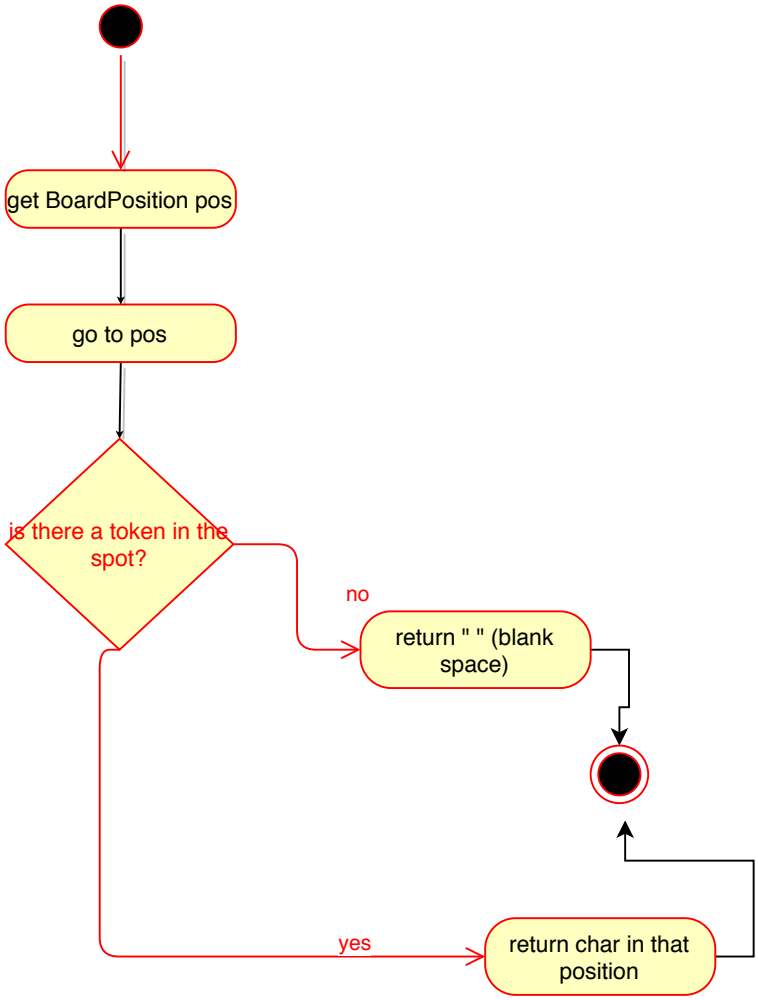

get BoardPosition pos
get pos.getRow() and pos.getCol()
Math.max(0, col - (getNumToWin() -1) for leftmost search postion
Math.min(getNumColumns() -1, col + (getNumToWin() -1)) for rightmost search postion
create boolean pass
create int inc, int i, int q all initialized to 0
for i = leftmost; i + getNumToWin() - 1 <= rightmost && i + getNumToWin() - 1 <= topmost
inc = topmost; i =0, j = 0
for q = 0; q < getNumToWin();
increment q
create new BoardPosition pos2 with row = row and col = i +qj
true
increment
decrement inc
return false


i
for i = leftmost; i + getNumToWin() - 1 <= rightmost && inc -(getNumToWin() - 1) <= 0
= true
for q = 0; q < getNumToWin();
increment q

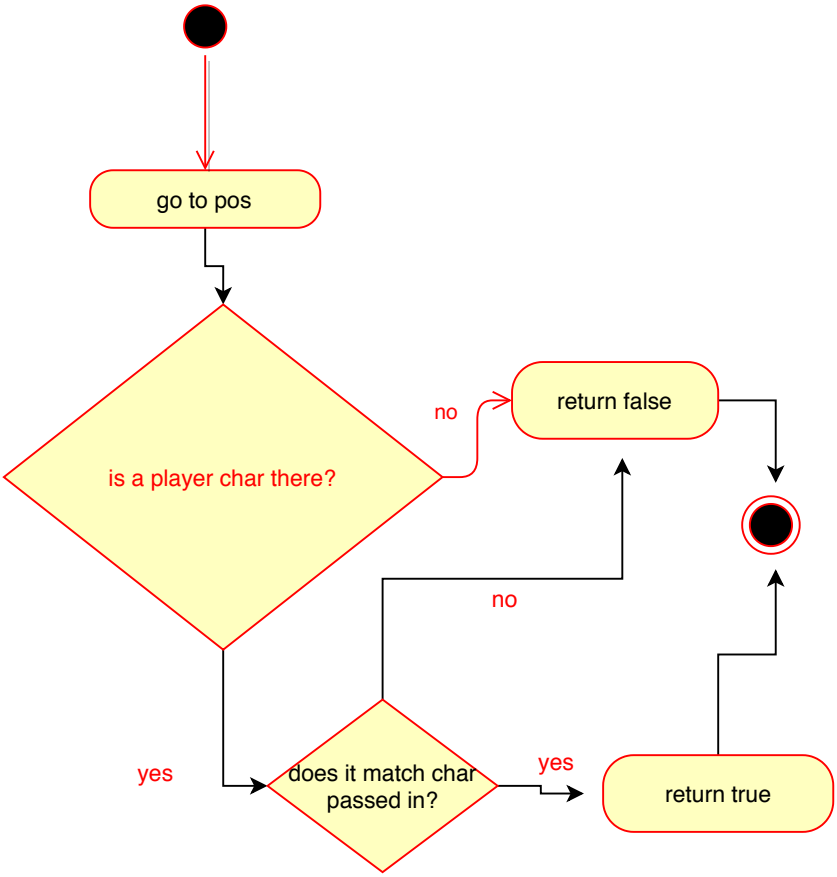

if char at pos 2 != p
true
pass = false
false
return

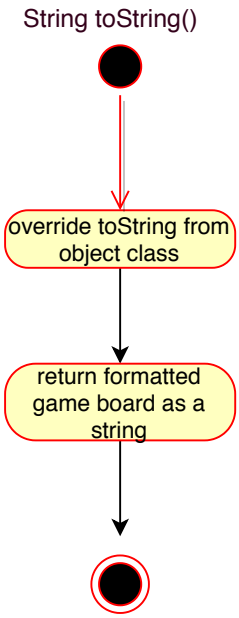

true
true
create new BoardPosition pos2 with row = row and col = i +qj
if char at pos 2 != p
true
pass = false
false

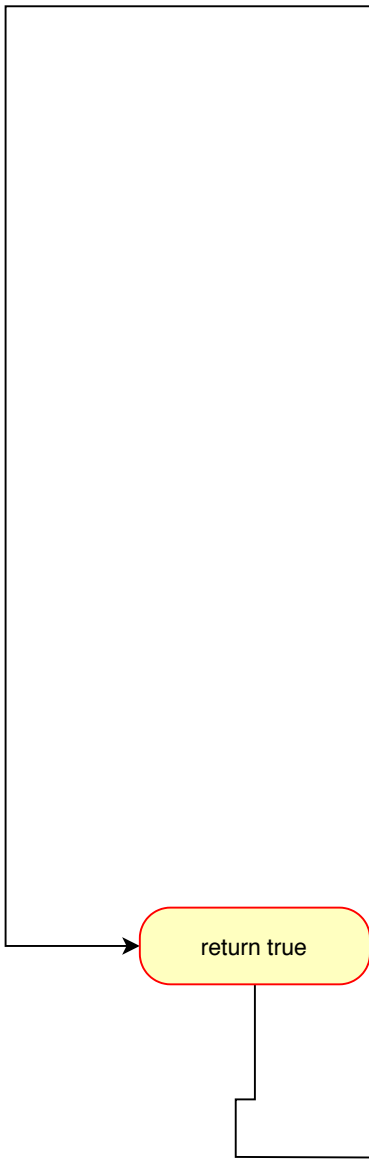
public char whatsAtPos(BoardPosition pos)

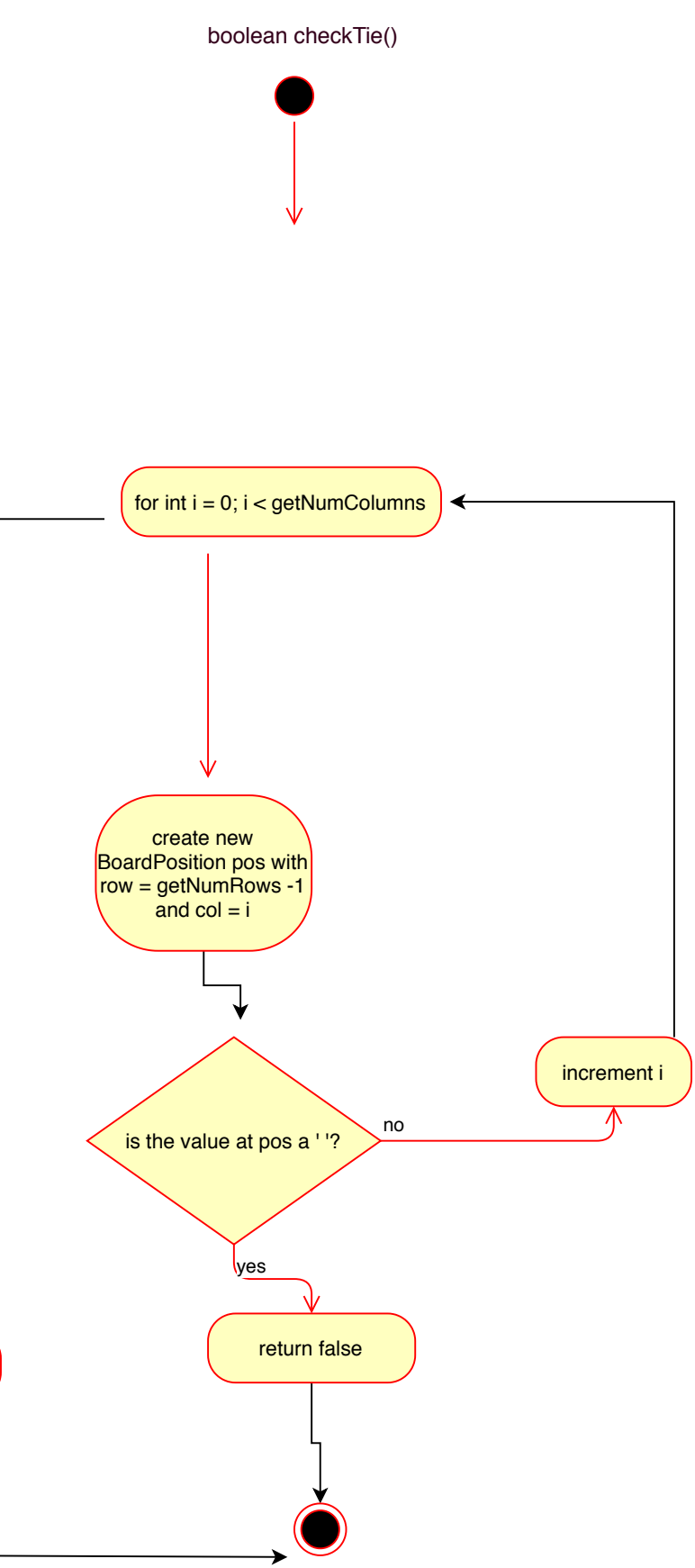


public bool isPlayerAtPos(BoardPosition pos, char player)



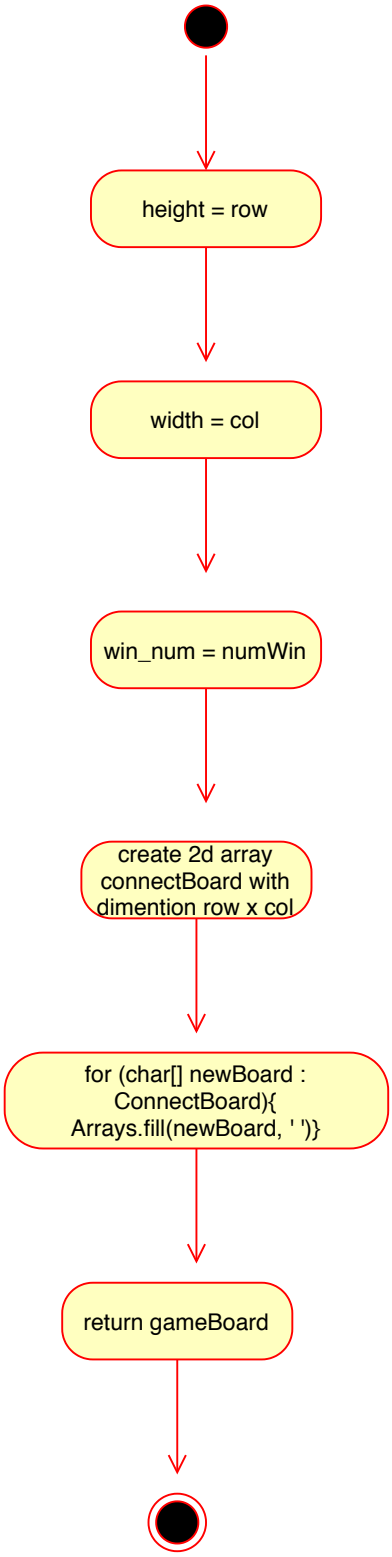

String toString()
override toString from object class
return formatted game board as a string


return true

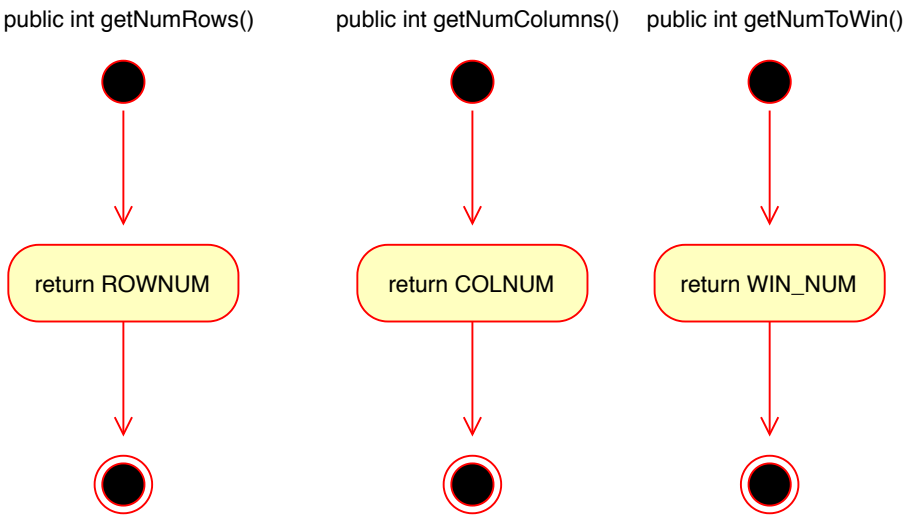

boolean checkTie()
for int i = 0; i < getNumColumns
create new BoardPosition pos with row = getNumRows -1 and col = i
is the value at pos a ' '?
no
increment i
yes
return false

Text

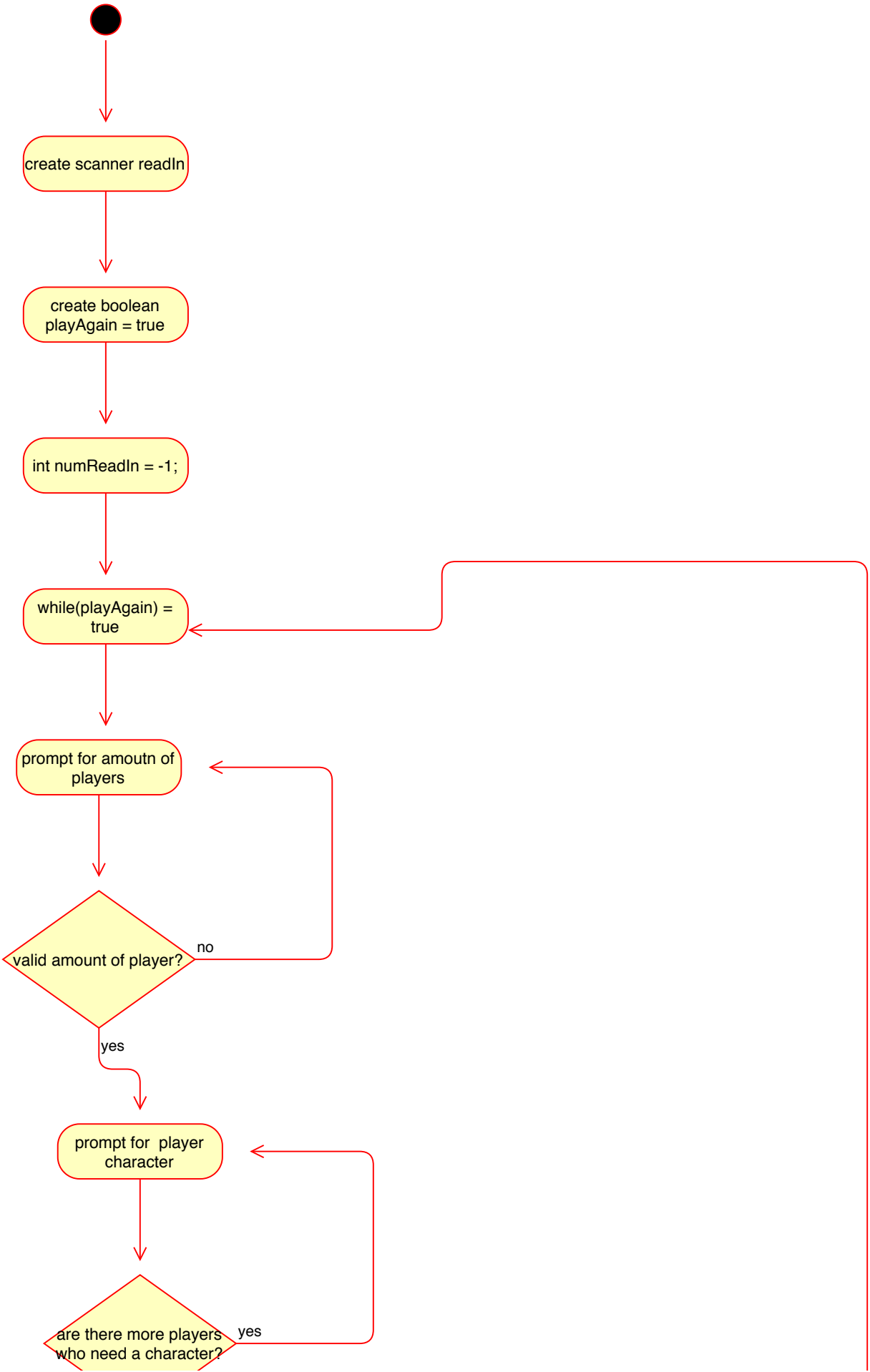
GameBoard gameBoard(int row, col, int numWin)

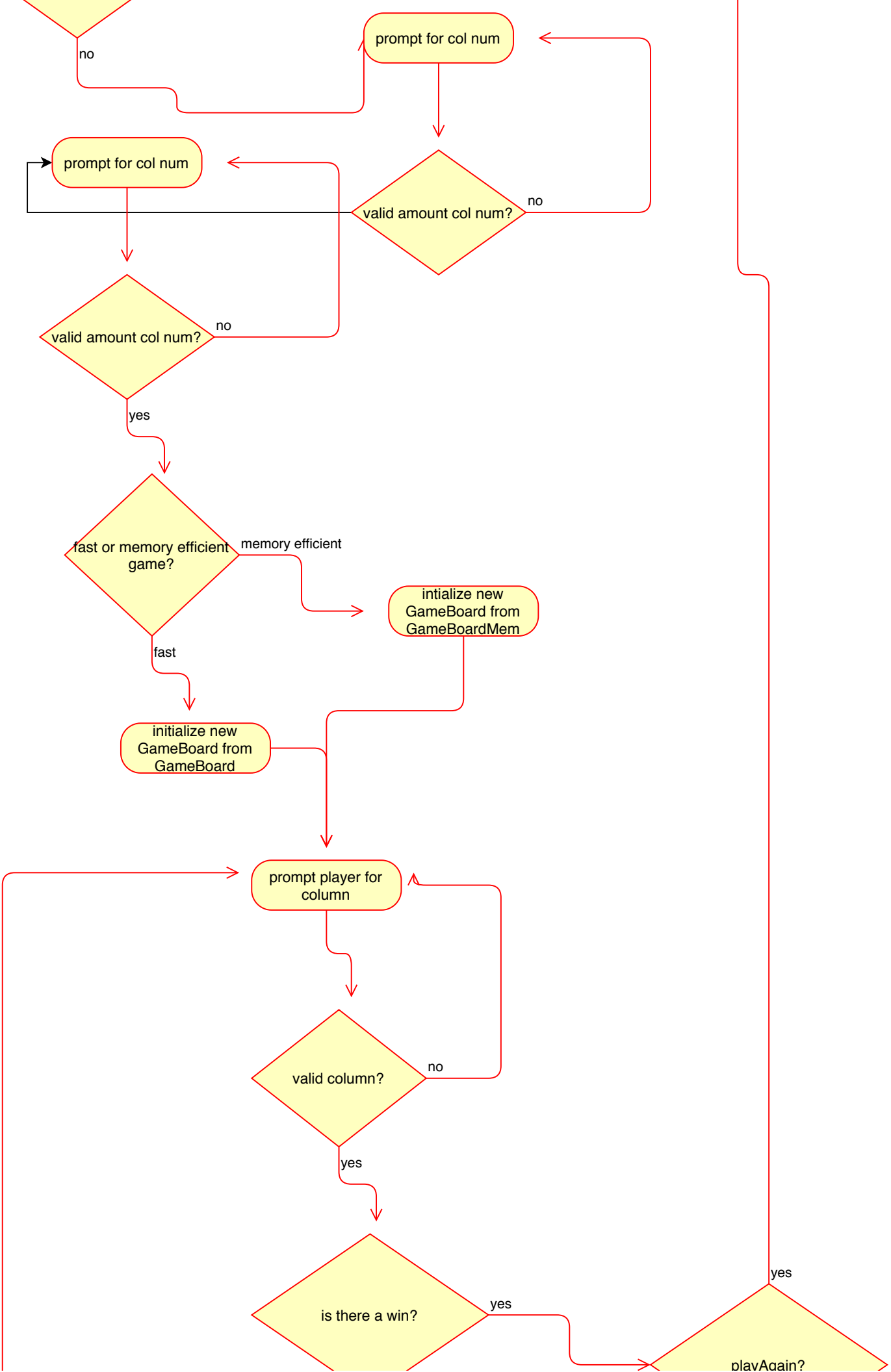


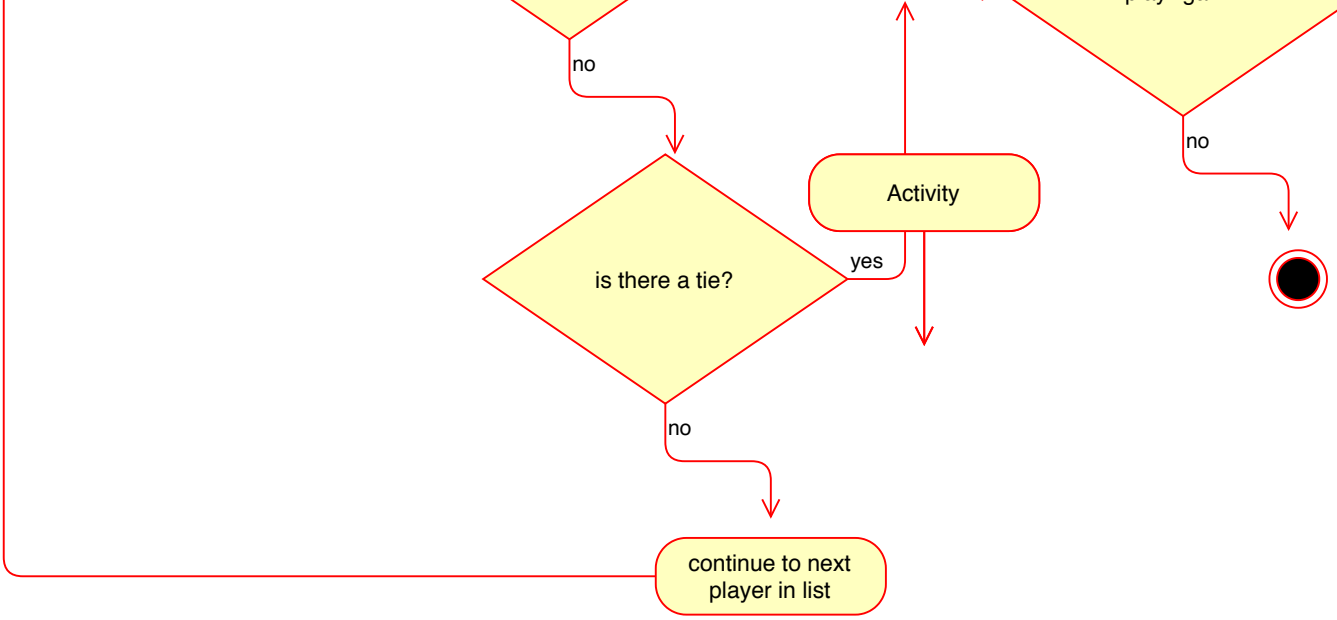

public void main(String[ ] args)
new GameBoard
connectXBoard
initialized
while gameInPlay ==
true
playerTurn is equal
to?
false (player O)
true (player X)
placeMarker()
getColumn(int)
checkIfFree(int)
returns?
false
true
checkIfWin(int)
returns?
false
playerTurn = true
true
checkForTie()
returns?
false
true
playAgain()
returns?
true
false
placeMarker()
getColumn(int)
checkIfFree(int)
returns?
false
true
checkIfWin(int)
returns?
false
playerTurn = false
true
playAgain()
returns?
true
false


public int getNumRows()
public int getNumColumns()
public int getNumToWin()
return ROWNUM
return COLNUM
return WIN_NUM

public void main(String[ ] args)




no
prompt for col num
prompt for col num
valid amount col num?
no
valid amount col num?
no
yes
fast or memory efficient game?
memory efficient
intialize new GameBoard from GameBoardMem
fast
initialize new GameBoard from GameBoard
prompt player for column
valid column?
no
yes
is there a win?
yes
yes
playAgain?


no
Activity
yes
is there a tie?
no
no
continue to next
player in list

Aika Washington
CPSC 2150
HW 3 – test cases

GameBoard IGameBoard() - constructor

<table>
<tr>
<td>Input
State (number to win = 3)

row = 5
col = 5</td>
<td>Output:
State:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Reason:

This test case is distinct because it creates a GameBoard and fills it with empty spots.

Function:

test_constructor</td>
</tr>
</table>

GameBoard IGameBoard() - constructor

<table>
<tr>
<td>Input
State (number to win = 3)

row = 3
col = 3</td>
<td>Output:
State:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Reason:

This test case is distinct because it creates a GameBoard of a different size.

Function:

Constructor_min_size</td>
</tr>
</table>

GameBoard IGameBoard() - constructor

<table>
<tr>
<td>Input
State (number to win = 25)

row = 100
col = 100</td>
<td>Output:
State:
Should be 100 x 100 board
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>
Max table in word is 63 x 63 so I can’t create 100 x 100 in this doc.</td>
<td>Reason:

This test case is distinct because it creates a GameBoard of a maximum size.

Function:

Constructor_max_size</td>
</tr>
</table>

boolean checkIfFree(int col)

Input	Output:	Reason: This test case is distinct because it checks if checkIfFree works on a full column (it should not)
State (number to win = 3) checkIfFree(0) X | | | | | X | | | | | X | | | | | X | | | | | X | | | | | col = 0	State: State of the board is unchanged	Function: checkIfFree_column_full

boolean checkIfFree(int col)

Input	Output:	Reason: This test case is distinct because it checks if checkIfFree works on an empty board or row
State (number to win = 3) checkIfFree(0) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | col = 0	State: State of the board is unchanged	Function: checkIfFree_column_empty

boolean checkIfFree(int col)

Input	Output:	Reason: This test case is distinct because it checks if checkIfFree works on a partially full row.
State (number to win = 3) checkIfFree(0) | | | | | | X | | | | | X | | | | | X | | | | | X | | | | | col = 0	State: State of the board is unchanged	Function: checkIfFree_column_not_full

boolean checkHorizWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if checkHorizWin works at the beginning of a horizontal win in row 0
State (number to win = 3) X X X X pos = 0,0 p = 'X'	State: State of the board is unchanged	Function: checkHorizWin_row0_beginning

boolean checkHorizWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if checkHorizWin works at the beginning of a full row
State (number to win = 3) X X X X X pos = 0,0 p = 'X'	State: State of the board is unchanged	Function: checkHorizWin_beginning_of_row

boolean checkHorizWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if checkHorizWin works at the beginning of a full row that doesn't not have a win with the given token
State (number to win = 3) X X O O O pos = 0,0 p = 'X'	State: State of the board is unchanged	Function: checkHorizWin_row0_false_beginning

boolean checkHorizWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if checkHorizWin works at the end of a row that does not have a win with the given token and is empty at that pos
State (number to win = 3) pos = 0,4 p = 'X'	State: State of the board is unchanged	Function: checkHorizWin_row0_false_end

			O	
X	X	X	X	O

boolean checkVertWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if checkVertWin works from the top of a vertical win
State (number to win = 3) pos = 2,0 p = 'X'	State: State of the board is unchanged	function: checkVertWin_from_top

X				
X				
X				

boolean checkVertWin(BoardPosition pos char p)

Input	Output:	Reason: This test case is distinct because it checks if checkVertWin works with on an empty spot in an almost full column
State (number to win = 3) pos = 4,0 p = 'X'	State: State of the board is unchanged	function: checkVertWin_false_with_empty_spot_above

X				
X				
O				
O				

boolean checkVertWin(BoardPosition pos, char p)

Input State (number to win = 3) pos = 3,0 p = 'X'	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks if checkVertWin works on a given token at the top of the col if it is not a win. function: checkVertWin_false_with_spots_below

X				
X				
O				
O				

boolean checkVertWin(BoardPosition pos, char p)

Input State (number to win = 3) pos = 0,0 p = 'X'	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks if checkVertWin works on an empty board (no), otherwise it would result in null pointer exception function: checkVertWin_false_with_empty_board

boolean checkDiagWin(BoardPosition pos, char p)

Input State (number to win = 3) pos = 0,0 p = 'X'	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks if checkDiagWin works on an empty board (no), otherwise it would result in null pointer exception function: checkDiagWin_false_with_empty_board

boolean checkDiagWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if checkDiagWin works a bottom left to top right diagonal condition. function: checkDiagWin_bottom_left_to_top_right
State (number to win = 3) pos = 2,2 p = 'X'	State: State of the board is unchanged	

		X		
	X	X		
X	X	X		

boolean checkDiagWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if checkDiagWin works a bottom right to top left diagonal condition. function: checkDiagWin_bottom_right_to_top_left
State (number to win = 3) pos = 2,0 p = 'X'	State: State of the board is unchanged	

X				
X	X			
X	X	X		

boolean checkDiagWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if checkDiagWin works a bottom left to top right diagonal condition that is filled beneath the diagonal. function: checkDiagWin_bottom_left_to_top_right_filled_under
State (number to win = 3) pos = 4,4 p = 'X'	State: State of the board is unchanged	

				X
			X	X
		X	X	X
O	O	O	O	O
O	O	O	O	O

boolean checkDiagWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if
State (number to win = 3) pos = 0,0 p = 'X'	State: State of the board is unchanged	checkDiagWin works a bottom left to top right diagonal condition that does not have enough chars to fufill the win condition. function: checkDiagWin_false_ bottom_left_to_top_right_insufficient_chars

	X			
X	X	X		

boolean checkDiagWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if
State (number to win = 3) pos = 1,1 p = 'X'	State: State of the board is unchanged	checkDiagWin works a top left to bottom right diagonal condition that does not have enough chars to fufill the win condition. function: checkDiagWin_false_ top_left_to_bottom_right_insufficient_chars

	X			
X	X	X		

boolean checkDiagWin(BoardPosition pos, char p)

Input	Output:	Reason: This test case is distinct because it checks if
State (number to win = 3) pos = 2,2 p = 'X'	State: State of the board is unchanged	checkDiagWin works on a full board with no diagonals for the given token. function: checkDiagWin_false_full_tied_board

X	X	O	X	O
X	X	O	X	O
X	X	O	X	O
X	X	O	X	O
X	X	O	X	O

boolean checkTie()

Input State (number to win = 3) X X X X X X X X X X X X X X X X X X X X X X X X X	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks if To make sure checkTie works when the board is full (same character) function: checkTie_true_full_board

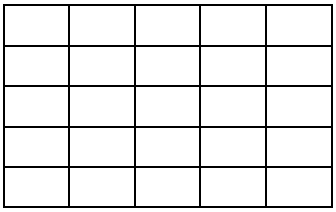
boolean checkTie()

Input State (number to win = 3) X X X X X X X X X X X X X X X	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks if To make sure checkTie works correctly when the board has some full columns function: checkTie_some_full_columns

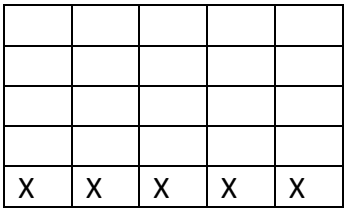
boolean checkTie()

Input State (number to win = 3) X X O X O X X O X O X X O X O X X O X O X X O X O	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks if To make sure checkTie works correctly when the board is full (with alternating characters) function: checkTie_full_alternating_board

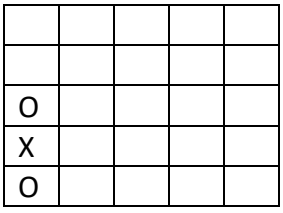
char whatsAtPos(BoardPosition pos)

Input State (number to win = 3)  pos = 0,0	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks whatsAtPos on a spot on an empty board function: WhatsAtPos_empty_space_empty_board

char whatsAtPos(BoardPosition pos)

Input State (number to win = 3)  pos = 1,0	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks whatsAtPos on a spot above the only full row on an otherwise empty board. function: WhatsAtPos_one_full_row_empty_space

char whatsAtPos(BoardPosition pos)

Input State (number to win = 3)  pos = 1,0	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks whatsAtPos on a occupied spot surrounded by other tokens. function: WhatsAtPos_spot_surrounded_by_chars

char whatsAtPos(BoardPosition pos)

Input State (number to win = 3) O X O pos = 1,0	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks whatsAtPos on a occupied spot surrounded by other tokens. function: WhatsAtPos_spot_surrounded_by_chars

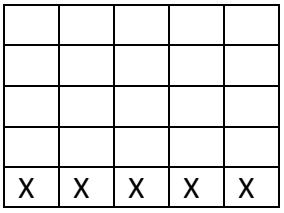
boolean isPlayerAtPos(BoardPosition pos)

Input State (number to win = 3) pos = 0,0	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks isPlayerAtPos on a spot on an empty board. function: isPlayerAtPos_false_empty_spot_empty_board

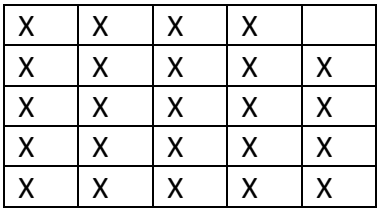
boolean isPlayerAtPos(BoardPosition pos)

Input State (number to win = 3) O pos = 0,4	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks isPlayerAtPos on one spot with a token on an otherwise an empty board. function: isPlayerAtPos_one_char_on_board

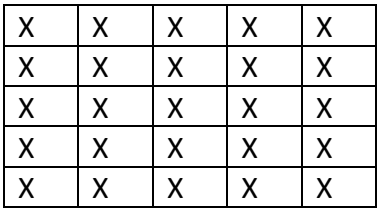
boolean isPlayerAtPos(BoardPosition pos)

Input State (number to win = 3)  pos = 0,1	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks isPlayerAtPos on a character in a filled row. function: isPlayerAtPos_one_filled_row

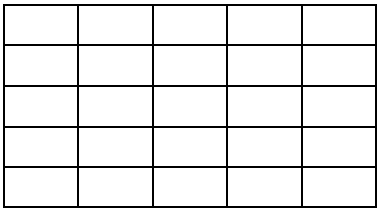
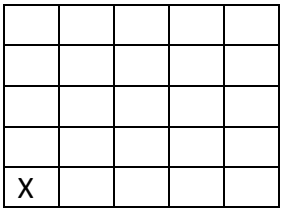
boolean isPlayerAtPos(BoardPosition pos)

Input State (number to win = 3)  pos = 4,4	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks isPlayerAtPos on an empty space function: isPlayerAtPos_false_almost_full_Board_empty_space

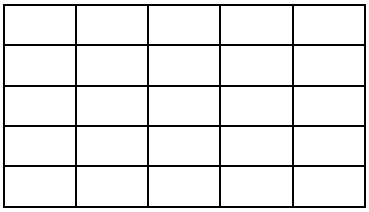
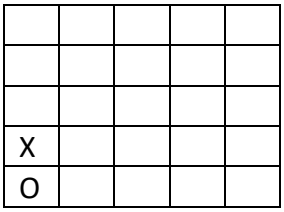
boolean isPlayerAtPos(BoardPosition pos)

Input State (number to win = 3)  pos = 4,4	Output: State: State of the board is unchanged	Reason: This test case is distinct because it checks isPlayerAtPos on a character on a full board. function: isPlayerAtPos_full_Board

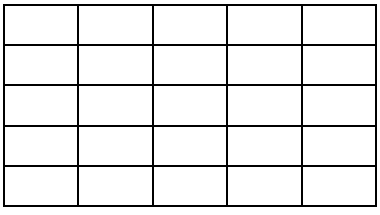
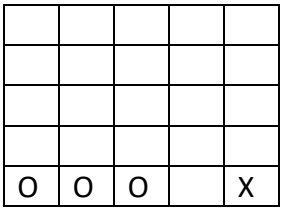
void placeToken(char p, int col)

Input State (number to win = 3)  placeToken('X',0)	Output: State: 	Reason: This test case is distinct because it checks if a token was successfully placed on an empty board. function: placeToken_on_empty_board

void placeToken(char p, int col)

Input State (number to win = 3)  placeToken ('O',0), ('X',0)	Output: State: 	Reason: This test case is distinct because it checks if a token was successfully placed in a partly filled column. function: placeToken_in_partly_filled_column

void placeToken(char p, int col)

Input State (number to win = 3)  placeToken ('O',0), ('O',1), ('O',2), ('X',4),	Output: State: 	Reason: This test case is distinct because it checks if a token was successfully placed in a partly filled row. function: placeToken_in_partly_filled_row

void placeToken(char p, int col)

Input State (number to win = 3) placeToken ('X',0), ('X',0), ('X',0), ('X',0), ('X',0), ('X',1), ('X',1), ('X',1), ('X',1), ('O',2), ('O',2), ('O',2), ('O',2), ('O',2), ('X',3), ('X',3), ('X',3), ('X',3), ('X',3), ('O',4), ('O',4), ('O',4), ('O',4),	Output: State:	Reason: This test case is distinct because it checks if tokens was successfully placed to fill a board. function: placeToken_to_fill_board

X	X	O	X	O
X	X	O	X	O
X	X	O	X	O
X	X	O	X	O
X	X	O	X	O