CPSC 2150 Project Report

Noah Nininger

Requirements Analysis

Functional Requirements:

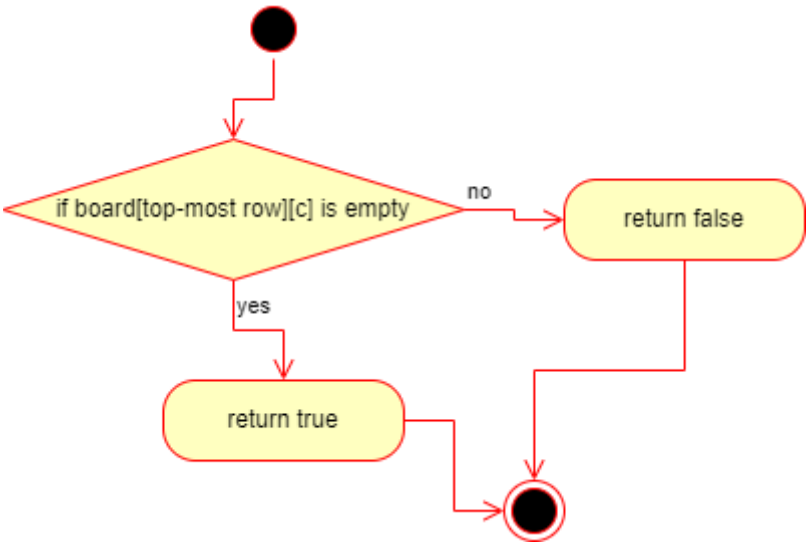
1. As a player, I can choose another column if the one selected is already full because no game pieces can fit in a full column
2. As a player, I can choose another column if the one selected is nonexistent because game pieces must be placed within valid regions
3. As a player, I can win the game because I had X amount of game pieces in a row horizontally
4. As a player, I can win the game because I had X amount game pieces in a row vertically
5. As a player, I can win the game because I had X amount game pieces in a row diagonally
6. As a player, I can have a tied game in which all spaces on the board are used up without X amount of game pieces in a row
7. As a player, I can have another turn if my opponent did not win in their last turn so that the game can have an outcome

Non-Functional Requirements

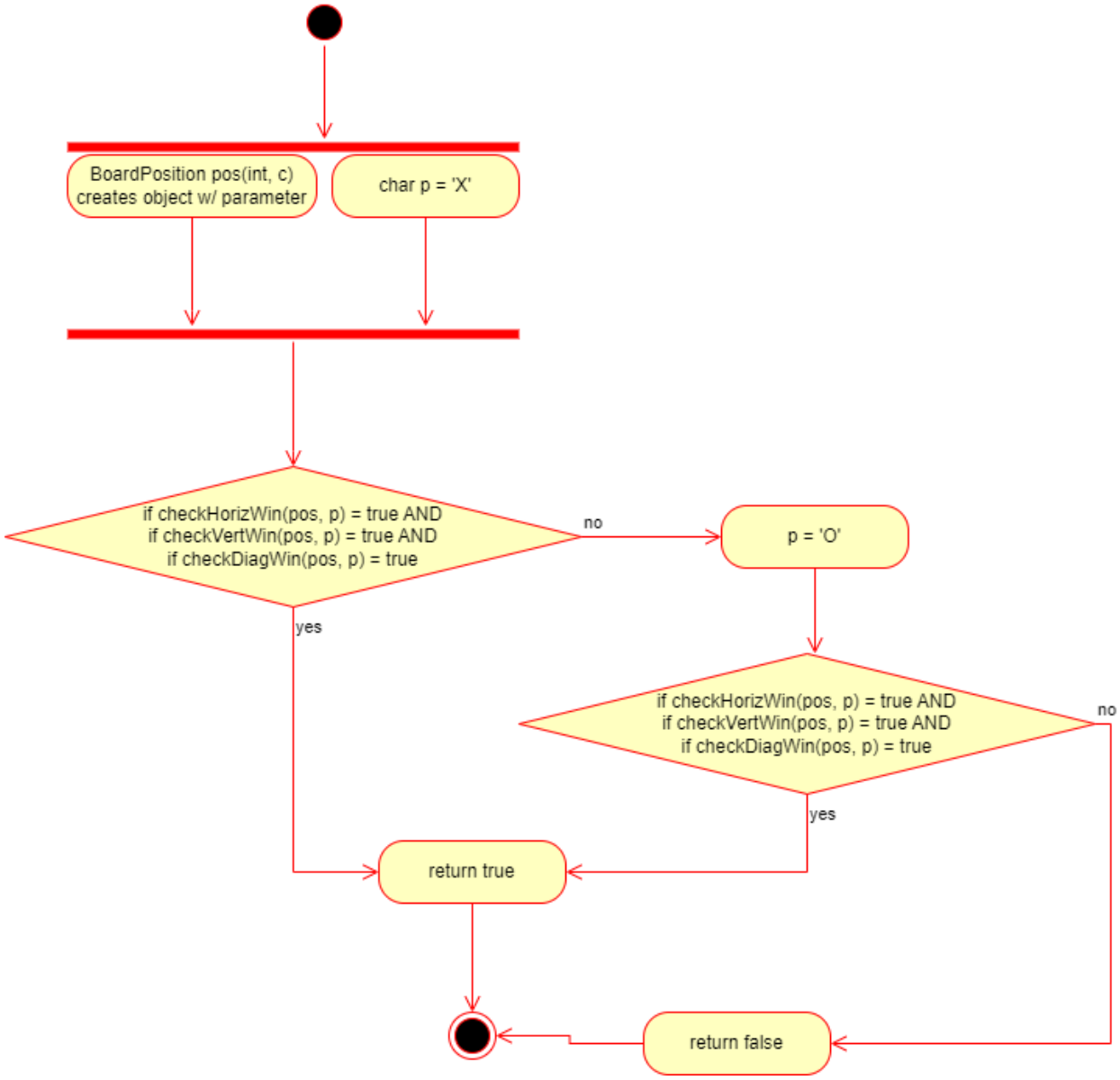
1. The game must use an MVC Architecture
2. The game must run in Java
3. The game must run on the SoC Machines
4. The game needs to validate the player's column placement
5. The game needs to count the game pieces
6. The game needs to display the board after each turn
7. Game board must be of size within the bounds of 3x3 and 20x20
8. The game must need minimum of 2 players, and maximum of 10 players
9. The number of tokens in a row to win must be within the bounds of 3 and 20
10. The number of tokens in a row to win must be less than or equal to the number of rows
11. (0,0) should be the bottom-left of the game board

IGameBoard
+ MAX_SIZE: static final int + MIN_SIZE: static final int + MAX_NUM_TO_WIN: static final int + MIN_NUM_TO_WIN: static final int + MAX_PLAYERS: static final int + MIN_PLAYERS: static final int + board: char[][]
+ checkIfFree(int): boolean + checkForWin(int): boolean + placeToken(char, int): void + checkHorizWin(BoardPosition, char): boolean + checkVertWin(BoardPosition, char): boolean + checkDiagWin(BoardPosition, char): boolean + whatsAtPos(BoardPosition): char + isPlayerAtPos(BoardPosition, char): boolean + checkTie(): boolean + getNumRows(): int + getNumColumns(): int + getNumToWin(): int

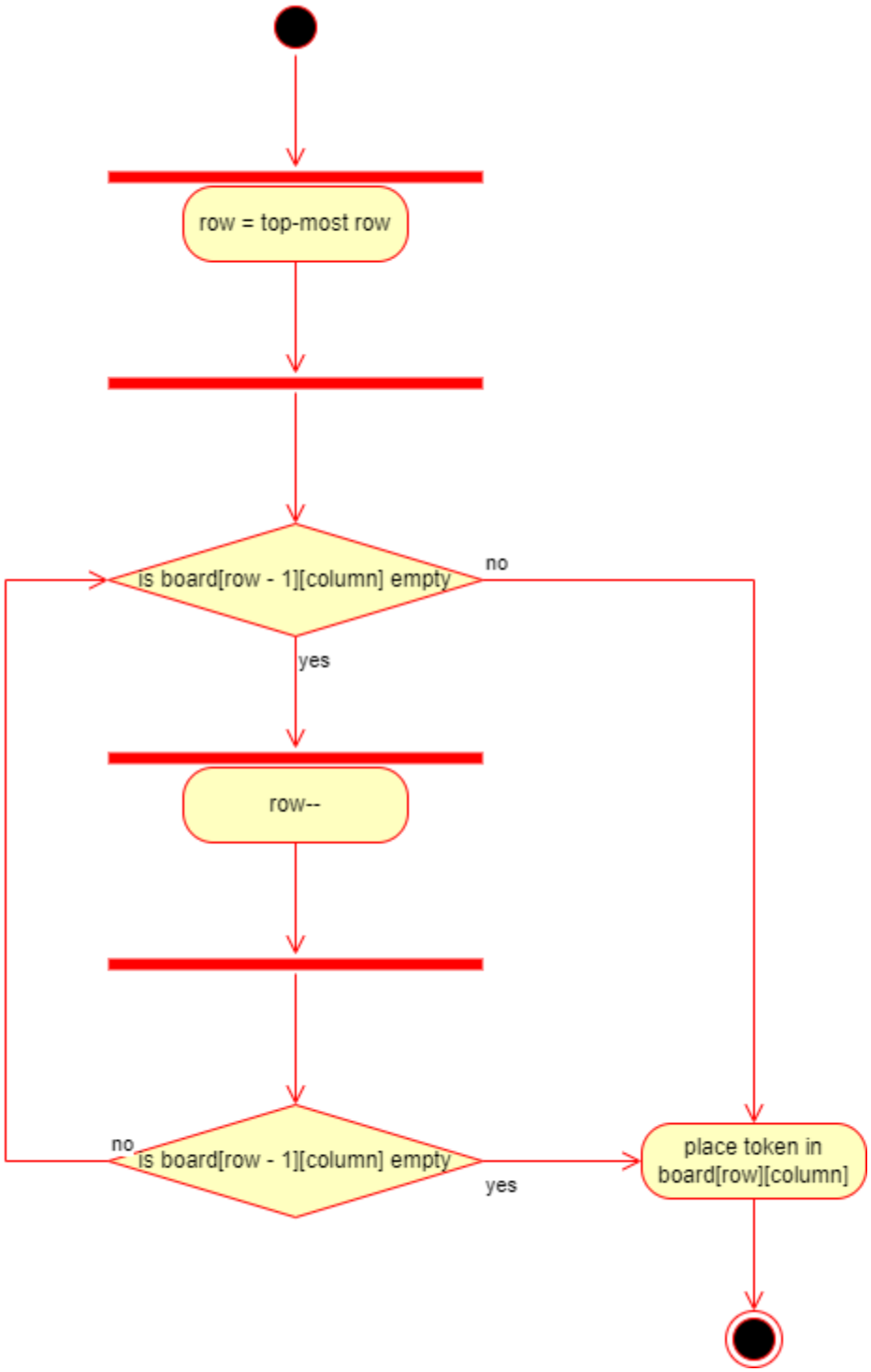
checkIfFree()



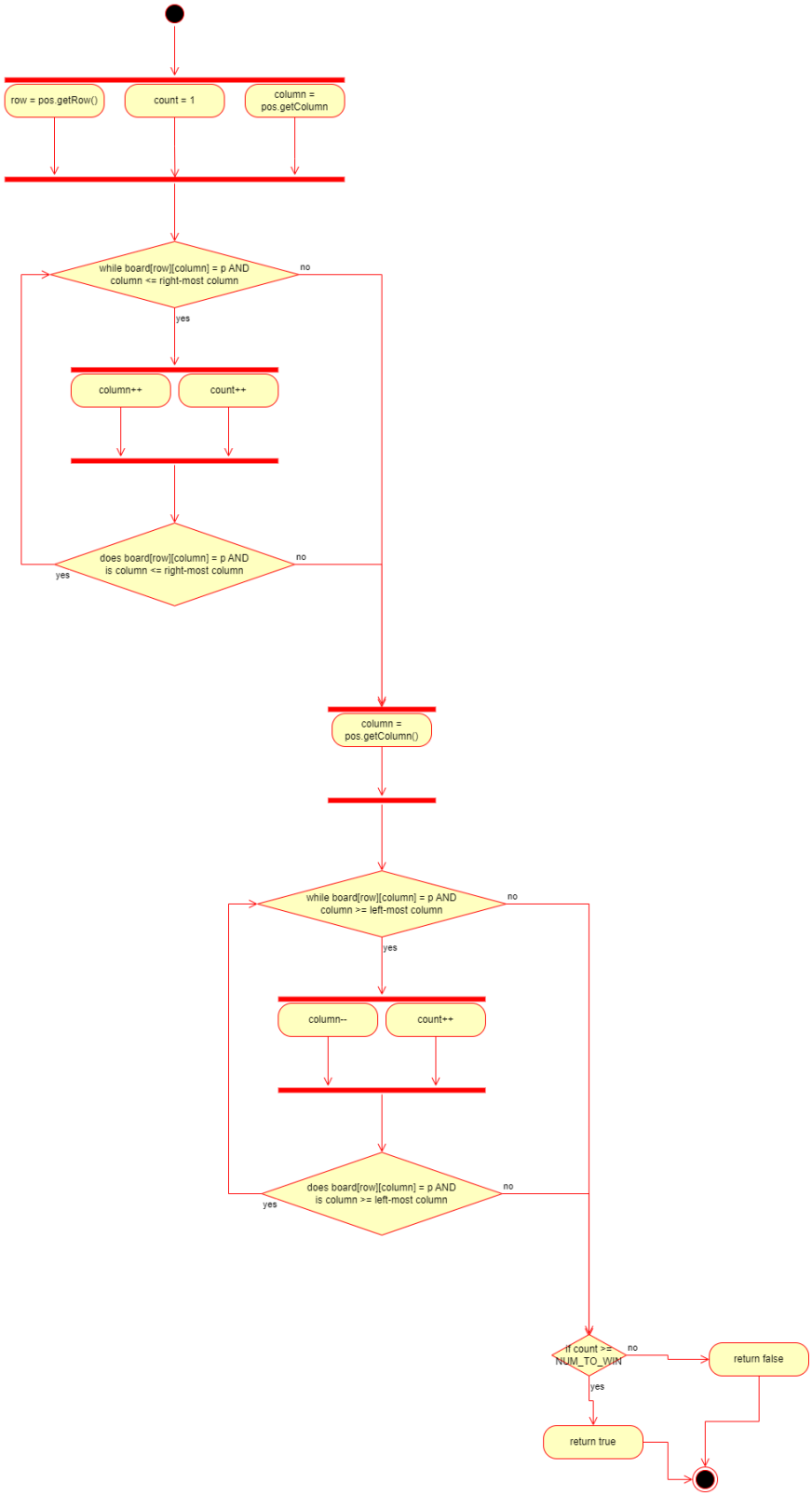
checkForWin()



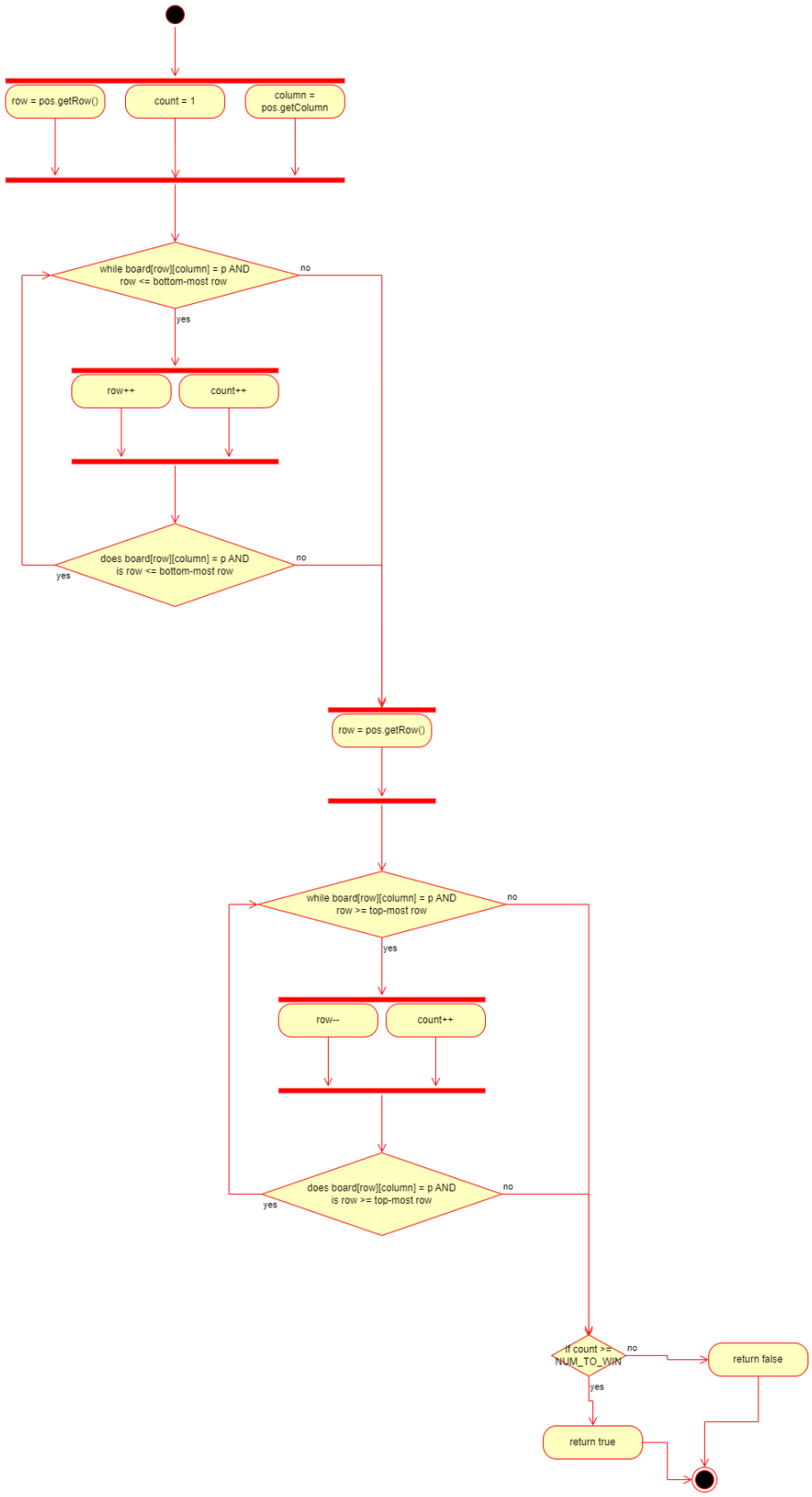
placeToken()


row = top-most row
is board[row - 1][column] empty
no
yes
row--
no
is board[row - 1][column] empty
yes
place token in board[row][column]

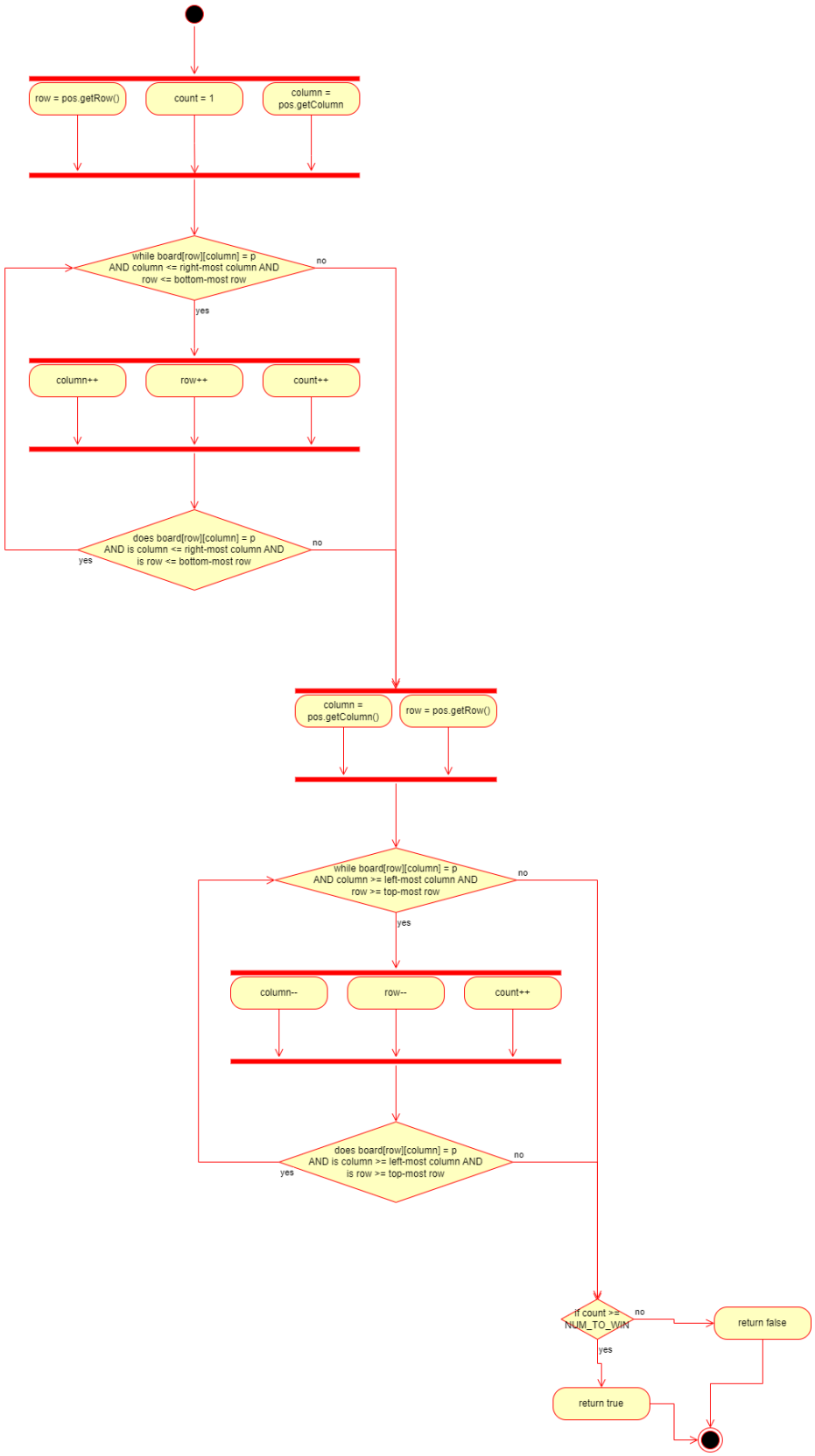
checkHorizWin()



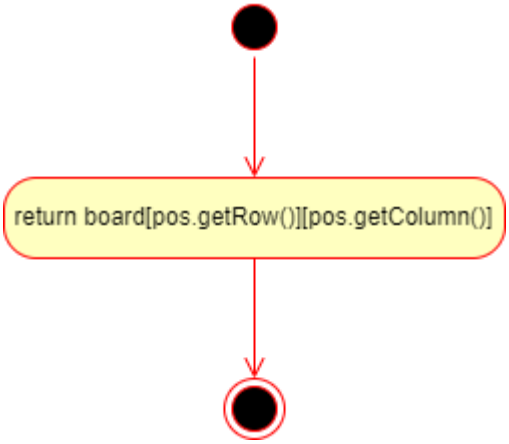
checkVertWin()



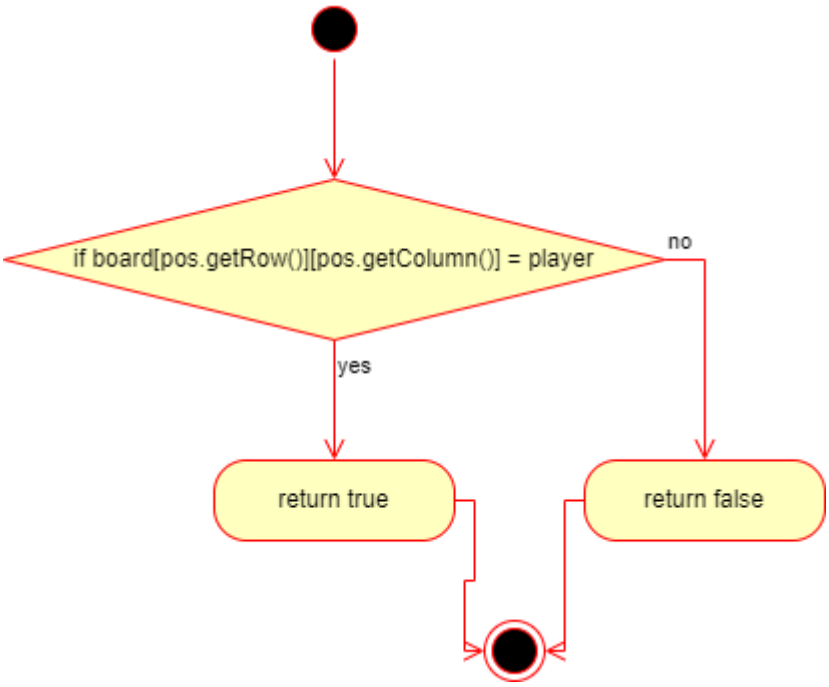
checkDiagWin()



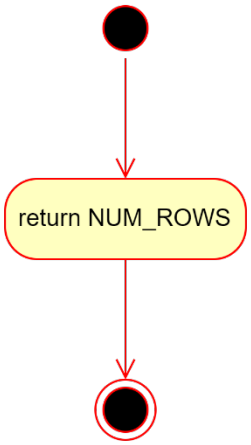
whatsAtPos()



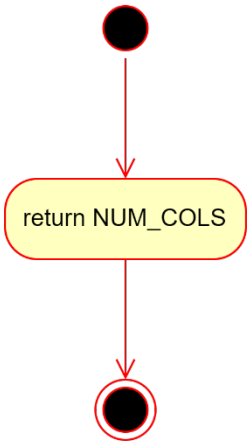
isPlayerAtPos()



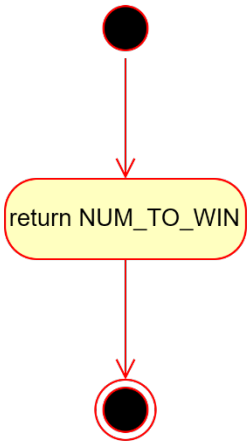
getNumRows()

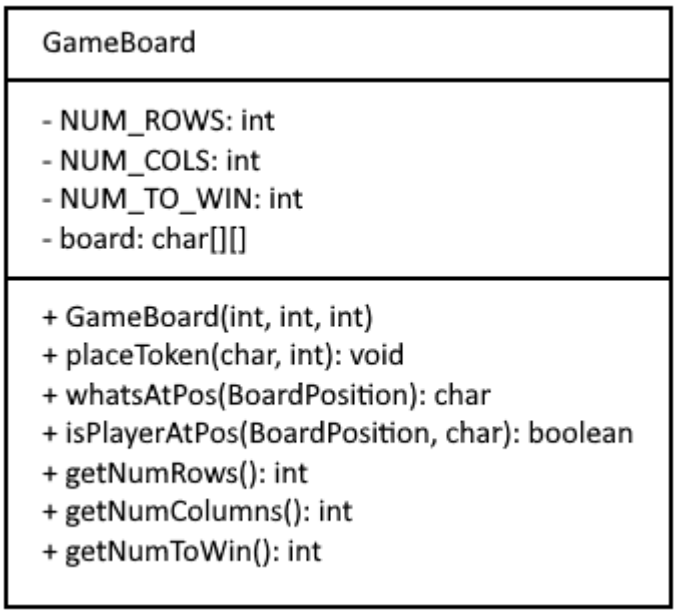


getNumColumns()



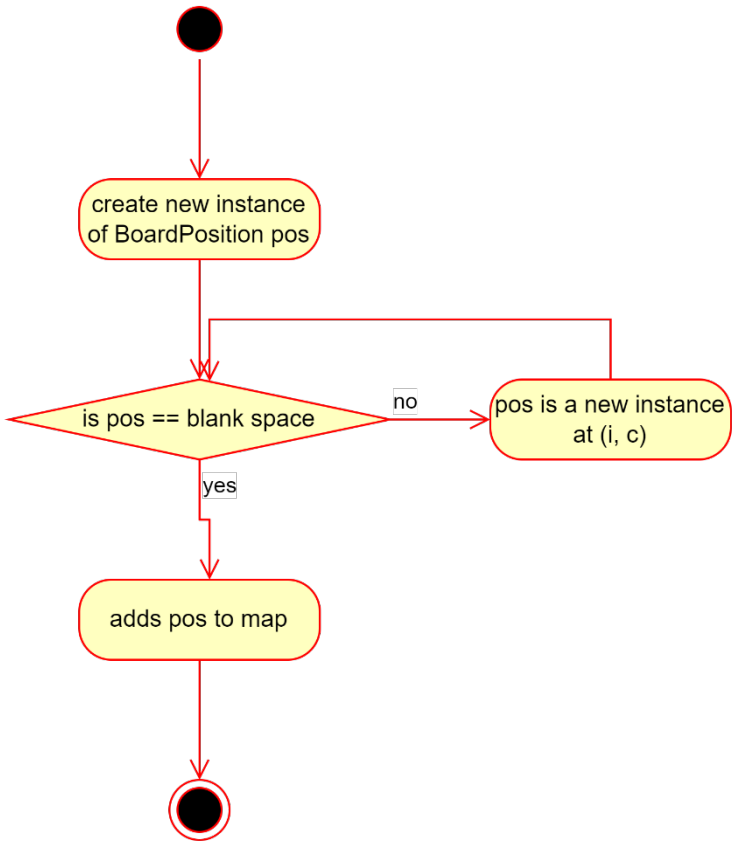
getNumToWin()



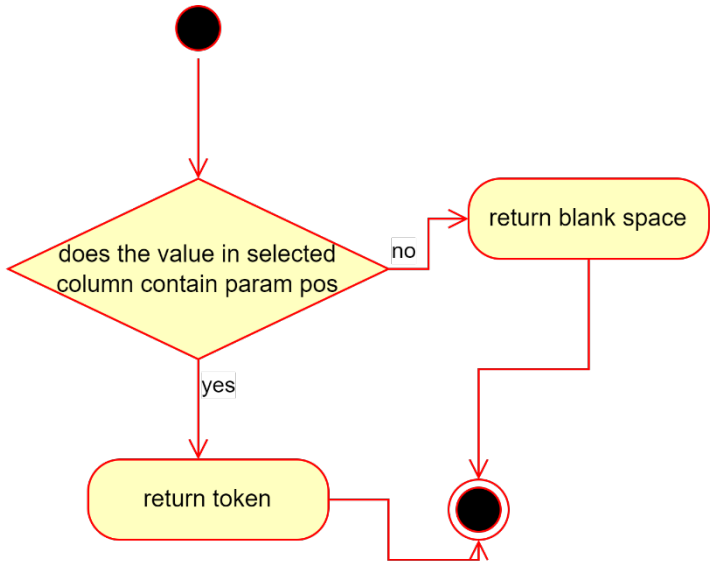

GameBoard
- NUM_ROWS: int
- NUM_COLS: int
- NUM_TO_WIN: int
- board: char[][]
+ GameBoard(int, int, int)
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int

```
GameBoardMem
-------------------------------------------------------------
- NUM_ROWS: int
- NUM_COLS: int
- NUM_TO_WIN: int
- map: Map<Character, List<BoardPosition>>
-------------------------------------------------------------
+ GameBoard(int, int, int)
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
```

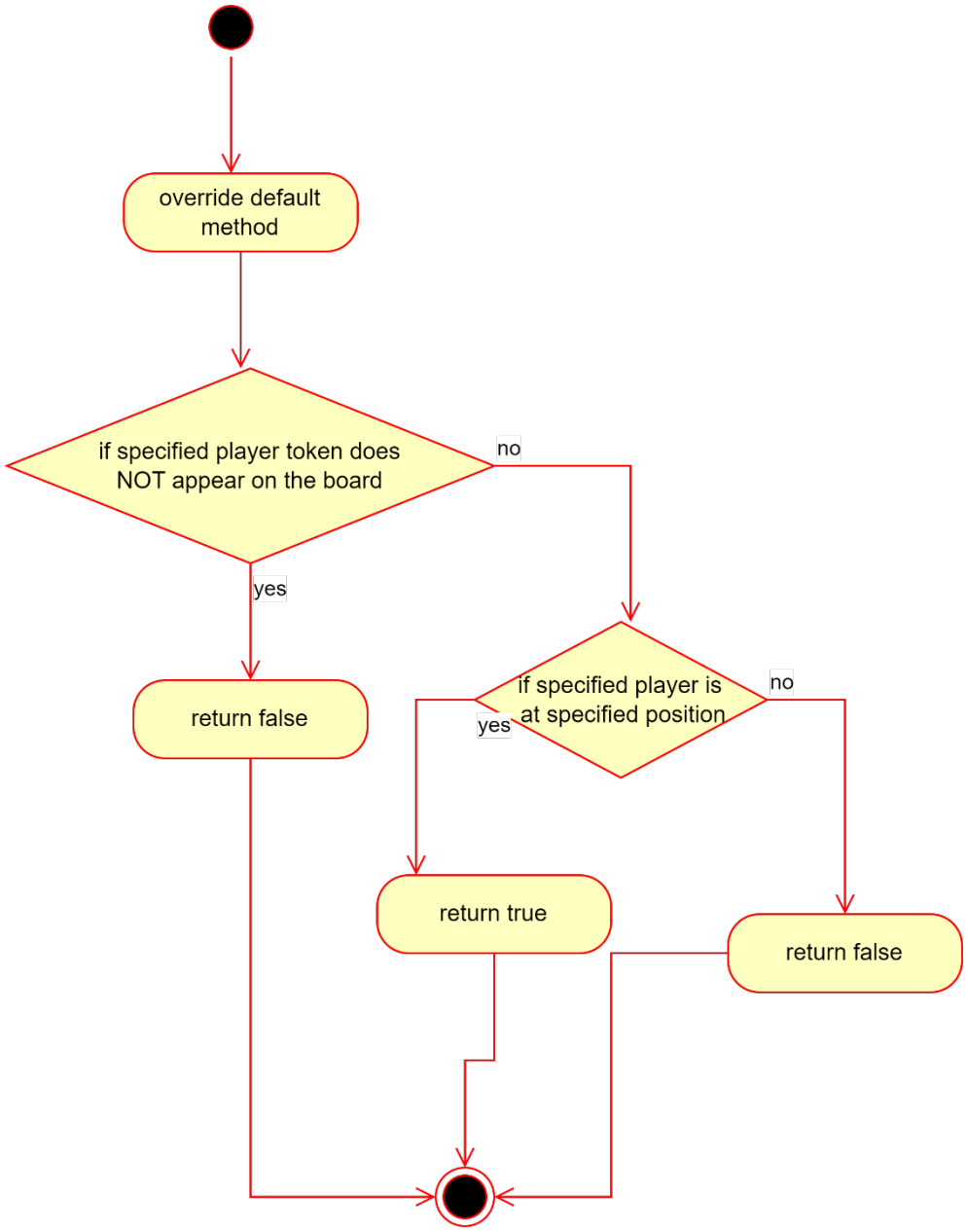
placeToken()

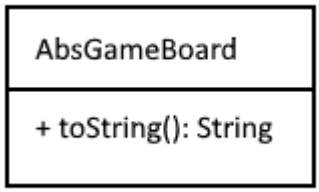

create new instance of BoardPosition pos
is pos == blank space
no
pos is a new instance at (i, c)
yes
adds pos to map

whatsAtPos()

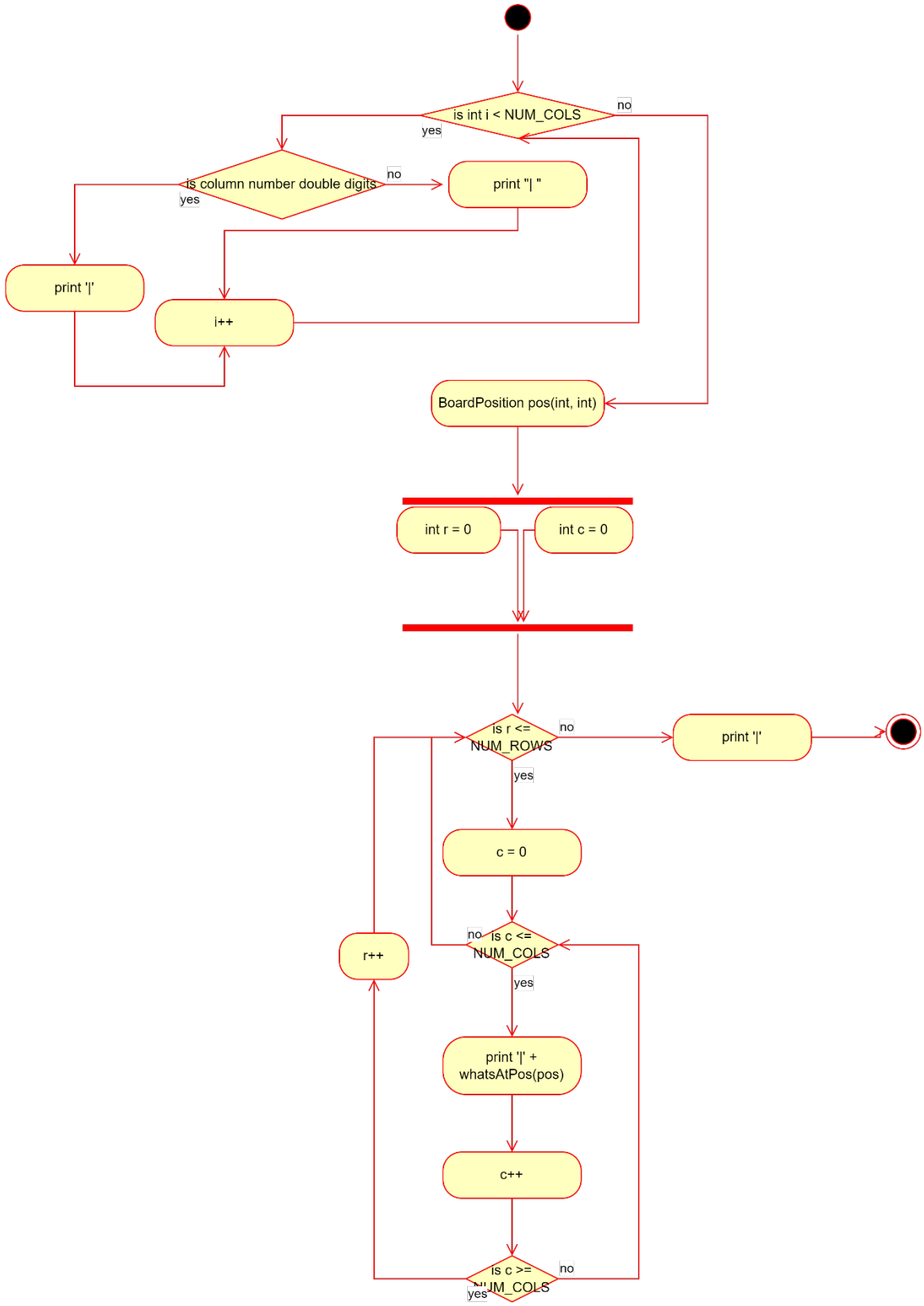

does the value in selected column contain param pos
no
return blank space
yes
return token

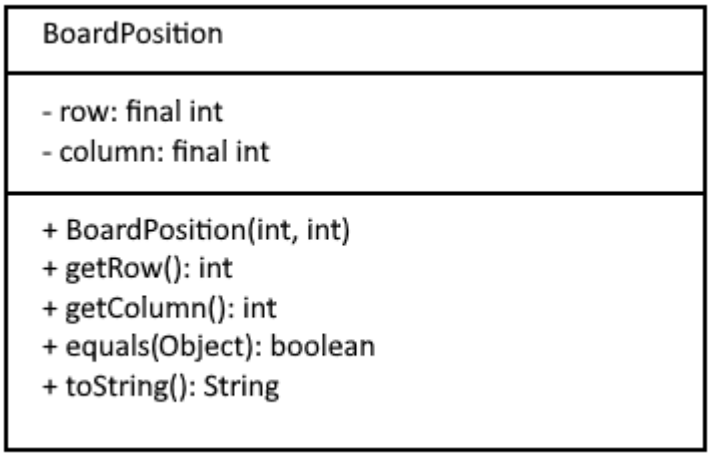
isPlayerAtPos()



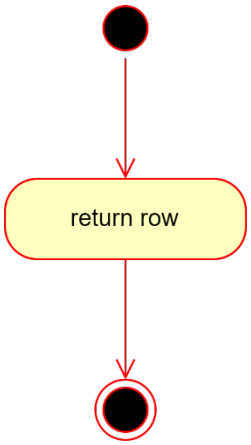

AbsGameBoard
+ toString(): String

toString()

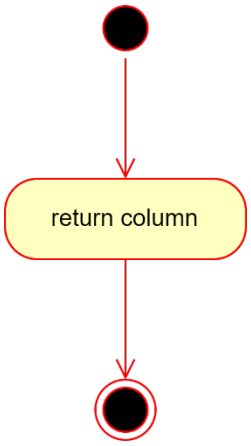

is int i < NUM_COLS
no
yes
is column number double digits
no
yes
print "| "
print '|'
i++
BoardPosition pos(int, int)
int r = 0
int c = 0
is r <= NUM_ROWS
no
yes
print '|'
c = 0
r++
no
is c <= NUM_COLS
yes
print '|' + whatsAtPos(pos)
c++
is c >= NUM_COLS
no
yes


BoardPosition
- row: final int
- column: final int
+ BoardPosition(int, int)
+ getRow(): int
+ getColumn(): int
+ equals(Object): boolean
+ toString(): String

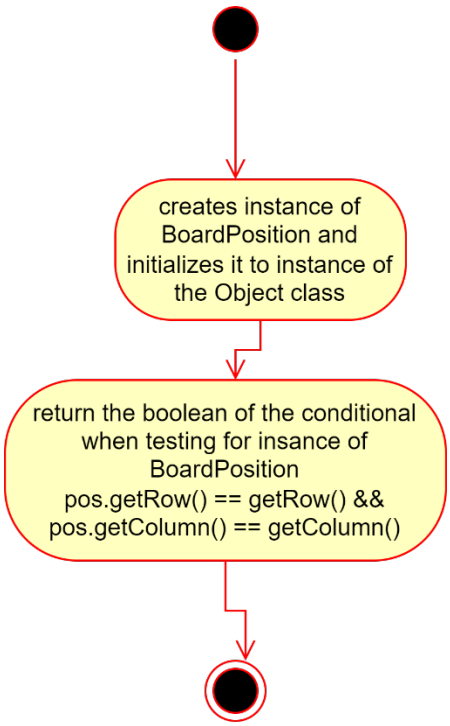
getRow()



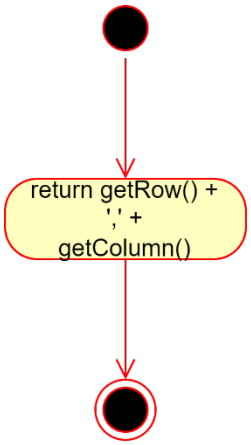
getColumn()



equals()


creates instance of BoardPosition and initializes it to instance of the Object class
return the boolean of the conditional when testing for insance of BoardPosition
pos.getRow() == getRow() &&
pos.getColumn() == getColumn()

toString()



GameBoard Constructor Test Cases

Input:

State: (New GameBoard is 4x4)

	0	1	2	3
3				
2				
1				
0				

NUM_ROWS = 4
NUM_COLS = 4

Output:

State: Unchanged

	0	1	2	3
3				
2				
1				
0				

Reason:

This test case is unique and distinct because the game board created was made with 4 rows and 4 columns

Function Name

testConstructor_4x4_board

Input:

State: (New GameBoard is 3x3)

	0	1	2
2			
1			
0			

NUM_ROWS = 3
NUM_COLS = 3

Output:

State: Unchanged

	0	1	2
2			
1			
0			

Reason:

This test case is unique and distinct because the game board created was made with 3 rows and 3 columns

Function Name

testConstructor_minSize_board

Input:

State: (New GameBoard is 100x100)

	0	1	...	99
99				
...				
1				
0				

NUM_ROWS = 100
NUM_COLS = 100

Output:

State: Unchanged

	0	1	...	99
99				
...				
1				
0				

Reason:

This test case is unique and distinct because the game board created was made with 100 rows and 100 columns

Function Name

testConstructor_maxSize_board

checkIfFree Test Cases

Input:

State: Empty column

	0	1	2	3
3				
2				
1				
0				

pos.getCol() = 0
p = 'x'

Output:

State: Token is placed

	0	1	2	3
3				
2				
1				
0	X			

Reason:
This test case is unique and distinct because the column being tested was empty

Function Name
testCheckIfFree_empty_column

Input:

State: Non-empty column

	0	1	2	3
3				
2				
1	X			
0	X			

pos.getCol() = 0
p = 'x'

Output:

State: Token is placed

	0	1	2	3
3				
2	X			
1	X			
0	X			

Reason:
This test case in unique and distinct because the column being tested was not empty

Function Name
testCheckIfFree_nonEmpty_col

Input:

State: Full column

	0	1	2	3
3	X			
2	X			
1	X			
0	X			

pos.getCol() = 0
p = 'x'

Output:

State: Unchanged

	0	1	2	3
3	X			
2	X			
1	X			
0	X			

Reason:
This test case is unique and distinct because the column being tested was full

Function Name
testCheckIfFree_full_column

checkHorizWin Test Cases

Input:

State:(number to win = 4)

X	X	X	X	
O	O	O	X	O

pos.getRow = 1
pos.getCol = 2
p = 'x'

Output:

State: Unchanged

X	X	X	X	
O	O	O	X	O

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
testCheckHorizWin_win_last_marker_middle

Input:

State: (number to win = 3)

X	X			
O	O	O		

pos.getRow = 0
pos.getCol = 2
p = 'o'

Output:

State: Unchanged

X	X			
O	O	O		

Reason:
This test case is unique and distinct because the last o was placed on the far right of the string of 3 consecutive o's, so the function needs to count to the left; also, NUM_TO_WIN is set to the minimum

Function Name
testCheckHorizWin_win_last_marker_rightMost

Input:

State: (number to win = 3)

	X	X		
O	O	O		

pos.getRow = 0
pos.getCol = 0
p = 'o'

Output:

State: Unchanged

	X	X		
O	O	O		

Reason:
This test case unique and distinct because the last o was placed on the far left of the string of 3 consecutive o's, so the function needs to count to the right

Function Name
testCheckHorizWin_win_last_marker_leftMost

Input:

State: (number to win = 4) No W

X	O	X		

pos.getRow = 0
pos.getCol = 0
pos = 'x'

Output:

State: Unchanged

X	O	X		

Reason:
This test case is unique and distinct because the last o placed did not result in a string of 4 consecutive o's

Function Name
testCheckHorizWin_no_win

checkVertWin Test Cases

Input:

State: (number to win = 4)

X				
X	O			
X	O			
X	O			

pos.getRow = 3
pos.getCol = 0
p = 'x'

Output:

State: Unchanged

X				
X	O			
X	O			
X	O			

Reason:
This test case is unique and distinct because the last x placed resulted in a string of 4 consecutive x's

Function Name
testCheckVertWin_win_4_in_a_row

Input:

State: (number to win = 3)

X				
X	O			
X	O			

pos.getRow = 2
pos.getCol = 0
p = 'x'

Output:

State: Unchanged

X				
X	O			
X	O			

Reason:
This test case is unique and distinct because the last x placed resulted in a string of 3 consecutive x's; also, NUM_TO_WIN is set to the minimum

Function Name
testCheckVertWin_win_minimum

Input:

State: (number to win = 25)

	0	1	...	99
24	X			
...	X			
1	X			
0	X	O	O	

pos.getRow = 25
pos.getCol = 0
p = 'x'

Output:

State: Unchanged

	0	1	...	99
24	X			
...	X			
1	X			
0	X	O	O	

Reason:
This test case is unique and distinct because the last x placed resulted in a string of 25 consecutive x's; also, NUM_TO_WIN is set to the maximum

Function Name
testCheckVertWin_win_maximum

Input:

State: (number to win = 4) No W

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

pos.getRow = 4
pos.getCol = 0
p = 'o'

Output:

State: Unchanged

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

Reason:
This test case is unique and distinct because the last o placed resulted in a string of 4 consecutive tokens; however, the o blocks x from having 4 in a row, resulting in no win for x

Function Name
testCheckVertWin_no_win

checkDiagWin Test Cases

Input:

State: (number to win = 3)

X				
O	X			
X	O	X		

pos.getRow = 2
pos.getCol = 0
p = 'x'

Output:

State: Unchanged

X				
O	X			
X	O	X		

Reason:
This test case is unique and distinct because the last x placed resulted in a string of 3 consecutive x's in a left diagonal, the last placed in in which was the top left-most x token

Function Name
testCheckDiagWin_win_topLeftMost_token

Input:

State: (number to win = 3)

		X		
	X	O		
X	O	X		

pos.getRow = 2
pos.getCol = 2
p = 'x'

Output:

State: Unchanged

		X		
	X	O		
X	O	X		

Reason:
This test case is unique and distinct because the last x placed resulted in a string of 3 consecutive x's in a right diagonal, the last placed in which was the top right-most x token

Function Name
testCheckDiagWin_win_topRightMost_token

Input:

State: (number to win = 3)

X				
O	X			
X	O	X		

pos.getRow = 0
pos.getCol = 2
p = 'x'

Output:

State: Unchanged

X				
O	X			
X	O	X		

Reason:
This test case is unique and distinct because the last x placed resulted in a string of 3 consecutive x's in a left diagonal, the last placed in which was the bottom right-most x token

Function Name
testCheckDiagWin_win_bottomRight-Most_token

Input:

State: (number to win = 3)

		X		
	X	O		
X	O	X		

pos.getRow = 0
pos.getCol = 0
p = 'x'

Output:

State: Unchanged

		X		
	X	O		
X	O	X		

Reason:
This test case is unique and distinct because the last x placed resulted in a string of 3 consecutive x's in a right diagonal, the last placed in which was the bottom left-most x token

Function Name
testCheckDiagWin_win_bottomLeftMost_token

Input:

State:

X				
O	X			
X	O	X		

pos.getRow = 1
pos.getCol = 1
p = 'x'

Output:

State: Unchanged

X				
O	X			
X	O	X		

Reason:
This test case is unique and distinct because the last x placed resulted in a string of 3 consecutive x's in a left diagonal, the last placed in which was the middle x token

Function Name
testCheckDiagWin_win_middle_token

Input:

State: (number to win = 3)

		X		
	X	O		
X	O	X		

pos.getRow = 1
pos.getCol = 1
p = 'x'

Output:

State: Unchanged

		X		
	X	O		
X	O	X		

Reason:
This test case is unique and distinct because the last x placed resulted in a string of 3 consecutive x's in a right diagonal, the last placed in which was the middle token

Function Name
testCheckDiagWin_win_middle_token_2

Input:

State: (number to win = 3) No W

		O		
	X	X		
X	O	O		

pos.getRow = 2
pos.getCol = 2
p = 'o'

Output:

State: Unchanged

		O		
	X	X		
X	O	O		

Reason:
This test case is unique and distinct because the last o placed resulted in no win for player x; in fact, the last o placed prevented player x from getting 3-in-a-row

Function Name
testCheckDiagWin_no_win

checkTie Test Cases

Input:

State: Empty GameBoard

p = 'x'

Output:

State: Token is placed

X				

Reason:
This test case is unique and distinct because it is the first turn of the game; therefore, the board is currently empty. There is no way a tie can occur after just the first turn

Function Name

testCheckTie_empty_board

Input:

State: Non-empty Board

X	O			

p = 'x'

Output:

State: Token is placed

	X			
X	O			

Reason:
This test case is unique and distinct because it is not the first turn; however, the game is not over. Since the board is not filled, the game is not tied.

Function Name

testCheckTie_nonEmpty_board

Input:

State: Almost full

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

p = 'o'

Output:

State: Token is placed

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

Reason:
This test case is unique and distinct because the game is one turn away from tying; however, that means the game still is not a tie

Function Name

testCheckTie_final_turn

Input:

State: Final Turn

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

p = 'x'

Output:

State: Token is placed (Tie)

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

Reason:
This test case is unique and distinct because the game is finally tied. This results in the boolean value to be returned as true.

Function Name

testCheckTie_tied_game

whatsAtPos Test Cases

Input:	Output:	Reason:
State: Empty GameBoard pos.getRow = 0 pos.getCol = 0	State: Unchanged	This test case is unique and distinct because it is the first turn of the game so the board is still empty; therefore, whatever is at position pos will be " " **Function Name** testWhatsAtPos_empty_board
Input:	Output:	Reason:
State: Non-empty GameBoard X pos.getRow = 0 pos.getCol = 0	State:Unchanged X	This test case is unique and distinct because a token is already placed and the function is going to test and see what token was placed at the designated coordinates, which is a token of player x **Function Name** testWhatsAtPos_x_at_pos
Input:	Output:	Reason:
State: Non-empty GameBoard X pos.getRow = 0 pos.getCol = 0	State: Unchanged X	This test case is unique and distinct because a token is already placed on the board, and the function will see what is to the left of that token, which would be " " **Function Name** testWhatsAtPos_left_of_x
Input:	Output:	Reason:
State: NonEmpty GameBoard X pos.getRow = 1 pos.getCol = 1	State: Unchanged X	This test case is unique and distinct because a token is already placed on the board, and the function will see what is over that token, which would be " " **Function Name** testWhatsAtPos_over_x
Input:	Output:	Reason:
State: Non-empty GameBoard X pos.getRow = 0 pos.getCol = 2	State: Unchanged X	This test case is unique and distinct because a token is already placed on the board, and the function will see what is to the right of that token, which would be " " **Function Name** testWhatsAtPos_right_of_x

isPlayerAtPos Test Cases

Input:	Output:	Reason:
State: Empty GameBoard pos.getRow = 0 pos.getCol = 0 p = 'x'	State: Unchanged	This test case is unique and distinct because the game board is still empty; therefore, player x cannot be at any position on the board **Function Name** testIsPlayerAtPos_empty_x
Input:	Output:	Reason:
State: Empty GameBoard pos.getRow = 0 pos.getCol = 0 p = 'o'	State: Unchanged	This test case is unique and distinct because the game board is still empty; therefore, player o cannot be at any position on the board **Function Name** testIsPlayerAtPos_empty_o
Input:	Output:	Reason:
State: Non-empty board w/ X X pos.getRow = 0 pos.getCol = 1 p = 'x'	State: Unchanged X	This test case is unique and distinct because the board is not empty, and we are looking for player x. Since player x is at the designated position, the boolean will return as true **Function Name** testIsPlayerAtPos_x_at_pos
Input:	Output:	Reason:
State: Non-empty board w/ O O pos.getRow = 0 pos.getCol = 1 p = 'o'	State: Unchanged O	This test case is unique and distinct because the board is not empty, and we are looking for player o. Since player o is at the designated position, the boolean will return as true **Function Name** tesIsPlayerAtPos_o_at_pos
Input:	Output:	Reason:
State: Non-empty GameBoard O pos.getRow = 0 pos.getCol = 0 p = 'x'	State: Unchanged O	This test case is unique and distinct because the board is not empty, and we are looking for player x; however, x is not at the designated position so the boolean would return as false **Function Name** testIsPlayerAtPos_x_not_at_pos

placeToken Test Cases

Input:

State: Empty GameBoard

p = 'x'
c = 0

Output:

State: Token is placed

X				

Reason:
This test case is unique and distinct because the game board is still empty; therefore, player x should be perfectly fine placing a token anywhere on the board

Function Name

testPlaceToken_empty_board

Input:

State: Non-empty GameBoard

X				

p = 'o'
c = 0

Output:

State: Token is placed

O				
X				

Reason:
This test case is unique and distinct because I am placing my marker in a column that was not empty, and also was not close to be being full.

Function Name

testPlaceToken_col_not_full

Input:

State:

X				
O				
X				
O				

p = 'o'
c = 0

Output:

State:

O				
X				
O				
X				
O				

Reason:
This test case is unique and distinct because I am placing my marker ina column that was not empty, and was close to being full

Function Name

testPlaceToken_col_last_token

Input:

State: Final Turn

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

p = 'x'
c = 4

Output:

State: Token is placed (Tie)

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

Reason:
This test case is unique and distinct because I am placing the last token of the game, resulting in a tied game

Function Name

testPlaceToken_tie_the_game

Input:

State: Full Column

O				
X				
O				
X				
O				

p = 'x'
c = 0

Output:

State: Unchanged

O				
X				
O				
X				
O				

Reason:
This test case is unique and distinct because I am attempting to place my token in a column that is already full, so instead of a token being placed, input validation would occur and I would choose another column

Function Name

testPlaceToken_col_is_full