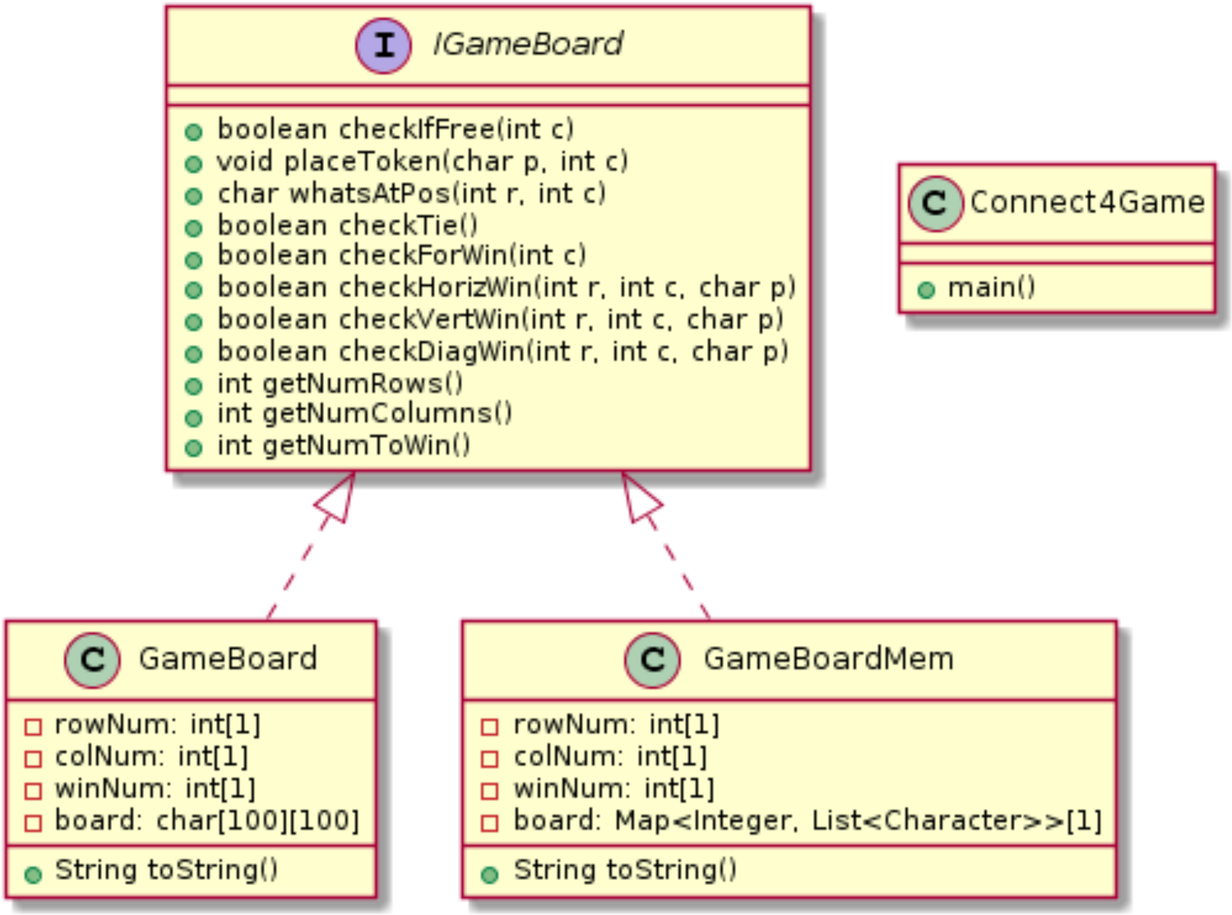
Daniel Toro
CPSC 2150
Fall 2018
Homework 4

Requirements Analysis:
Users can pick number of rows for game customization.
Users can pick number of columns for game customization.
Users can pick number to win for game customization.
Users can select number of players for inclusivity.
Users can pick any character as their token for customization.
Users can choose a fast or memory efficient implementation for adaptability.
Users can change the board size when starting a new game for convenience.
Users can pick columns to drop tokens in.
Users can see board printed for up to date info.
Users are informed when they win for up to date info.
Users are informed when they tie for up to date info.
Users are asked to play again for convenient access.
Users can play a game from the terminal for ease of access.
Two to ten users can play for competition.

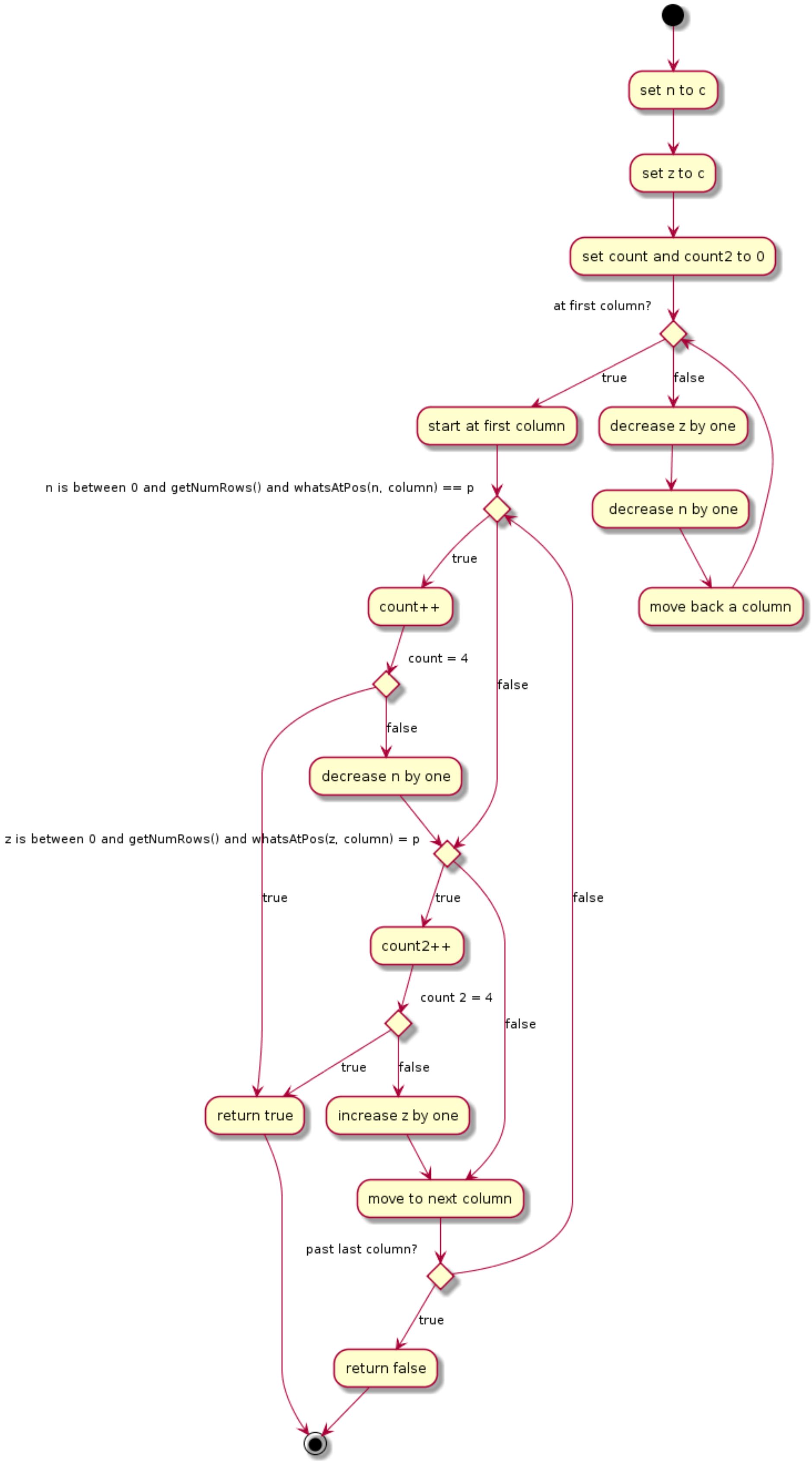
Program must be in java
Program must print to console
Program must be compiled with a makefile
Program must take user input

Design:

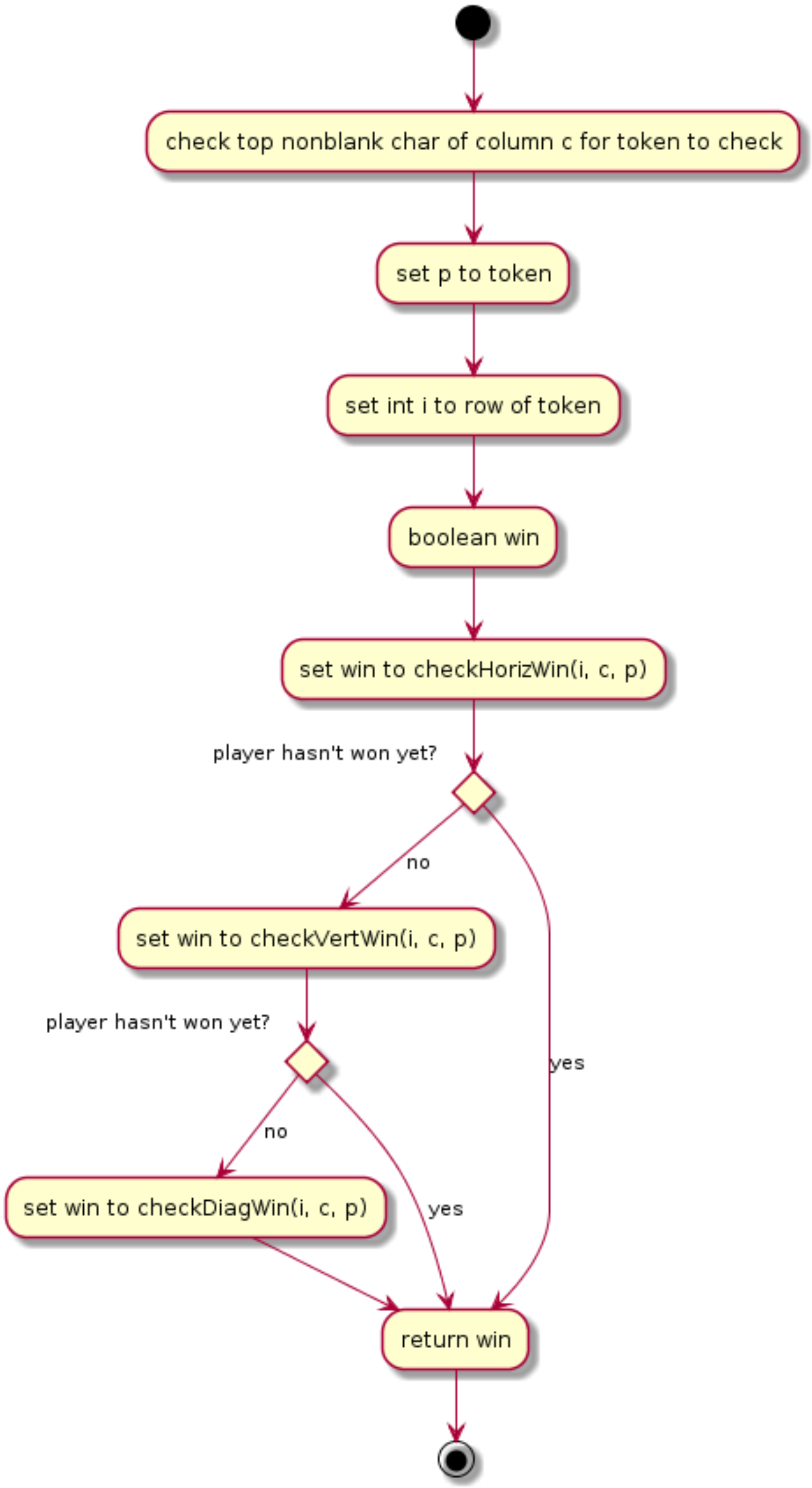
Red squares indicate private, while green circles indicate public. I stands for Interface.
C for Class.


I IGameBoard
+ boolean checkIfFree(int c)
+ void placeToken(char p, int c)
+ char whatsAtPos(int r, int c)
+ boolean checkTie()
+ boolean checkForWin(int c)
+ boolean checkHorizWin(int r, int c, char p)
+ boolean checkVertWin(int r, int c, char p)
+ boolean checkDiagWin(int r, int c, char p)
+ int getNumRows()
+ int getNumColumns()
+ int getNumToWin()
C Connect4Game
+ main()
C GameBoard
- rowNum: int[1]
- colNum: int[1]
- winNum: int[1]
- board: char[100][100]
+ String toString()
C GameBoardMem
- rowNum: int[1]
- colNum: int[1]
- winNum: int[1]
- board: Map<Integer, List<Character>>[1]
+ String toString()

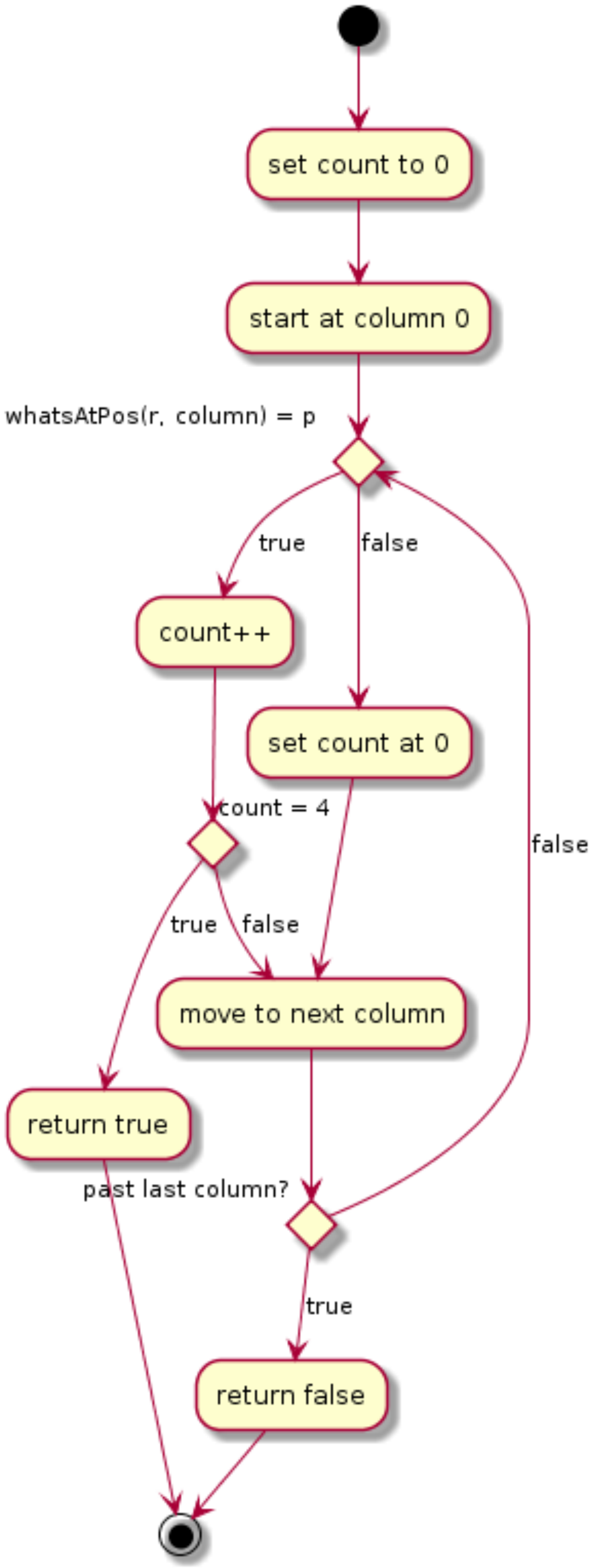
checkDiagWin(int r, int c, char p)



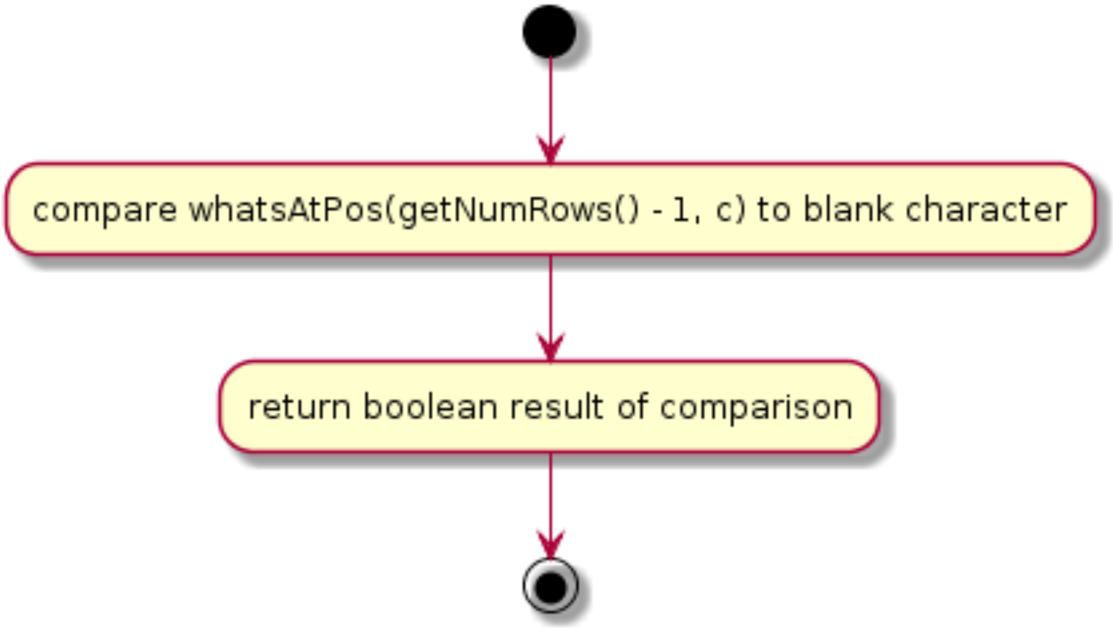
checkForWin(int c)


check top nonblank char of column c for token to check
set p to token
set int i to row of token
boolean win
set win to checkHorizWin(i, c, p)
player hasn't won yet?
no
yes
set win to checkVertWin(i, c, p)
player hasn't won yet?
no
yes
set win to checkDiagWin(i, c, p)
return win

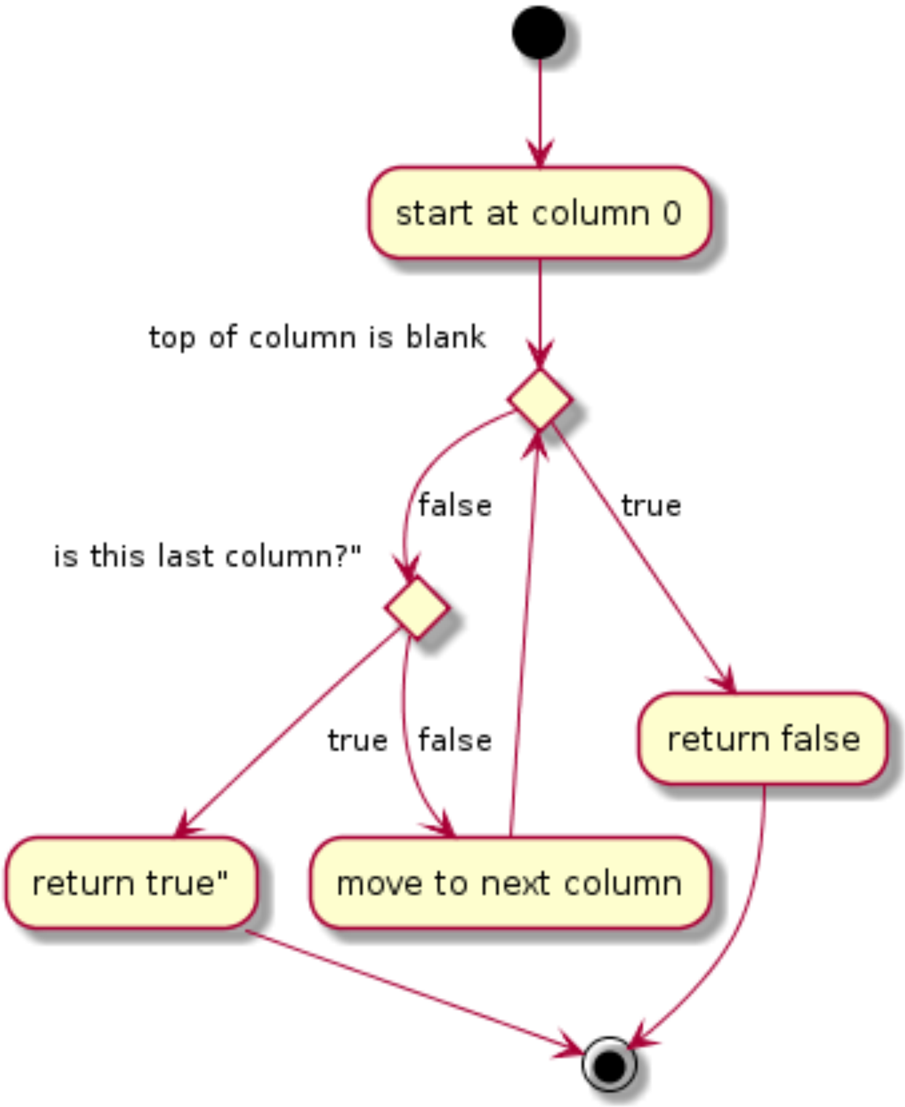
checkHorizWin(int r, int c, char p)



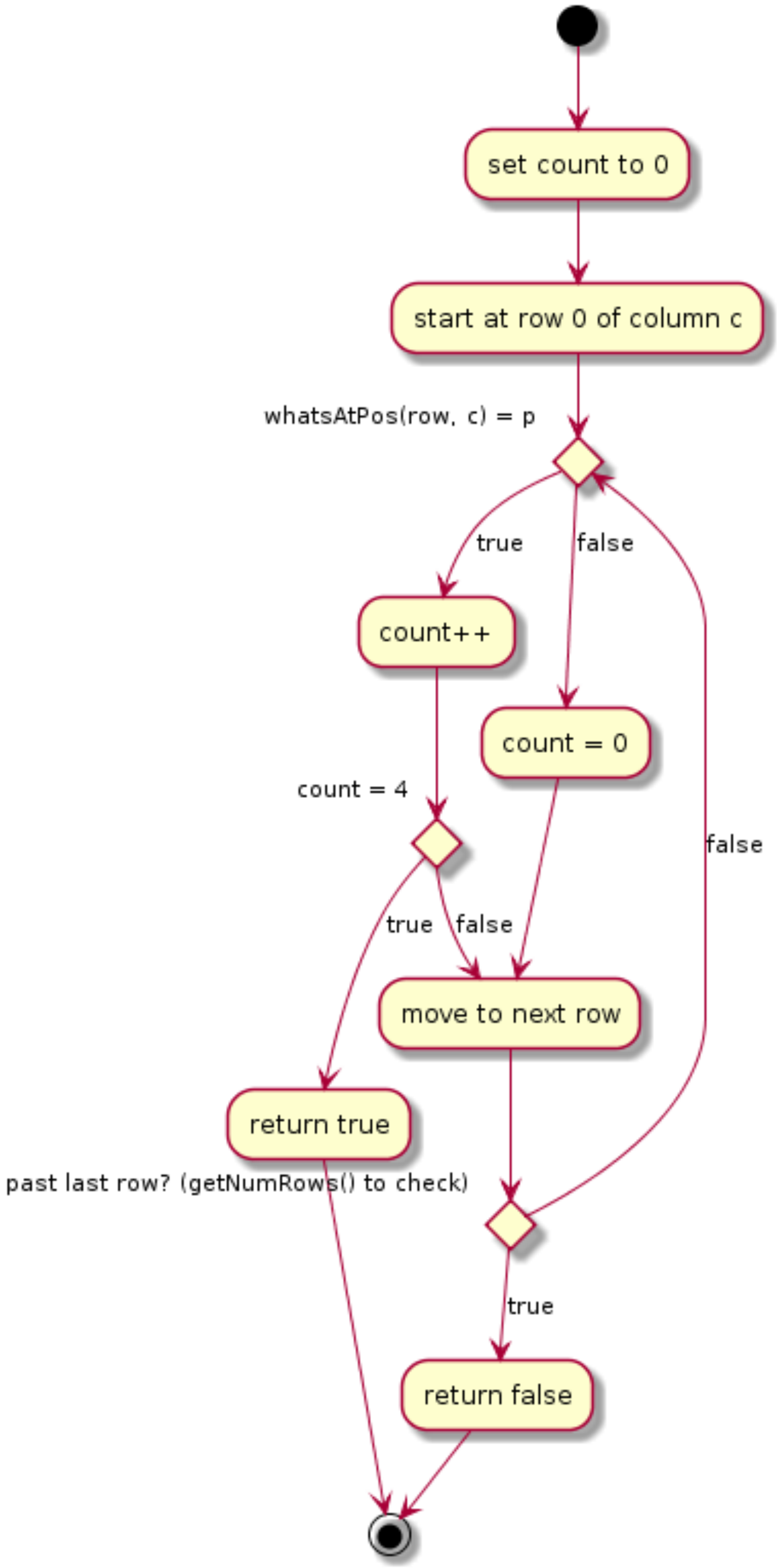
checkIfFree(intc)


compare whatsAtPos(getNumRows() - 1, c) to blank character
return boolean result of comparison

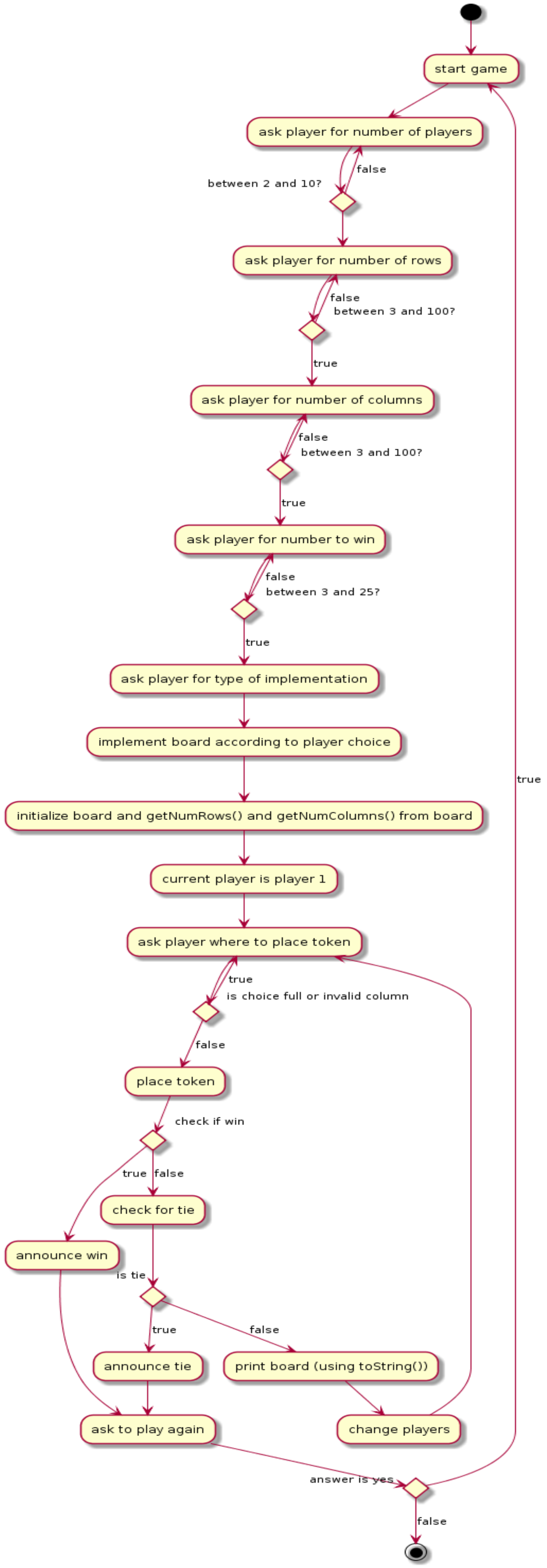
checkTie()


start at column 0
top of column is blank
false
true
is this last column?"
true
false
return false
return true"
move to next column

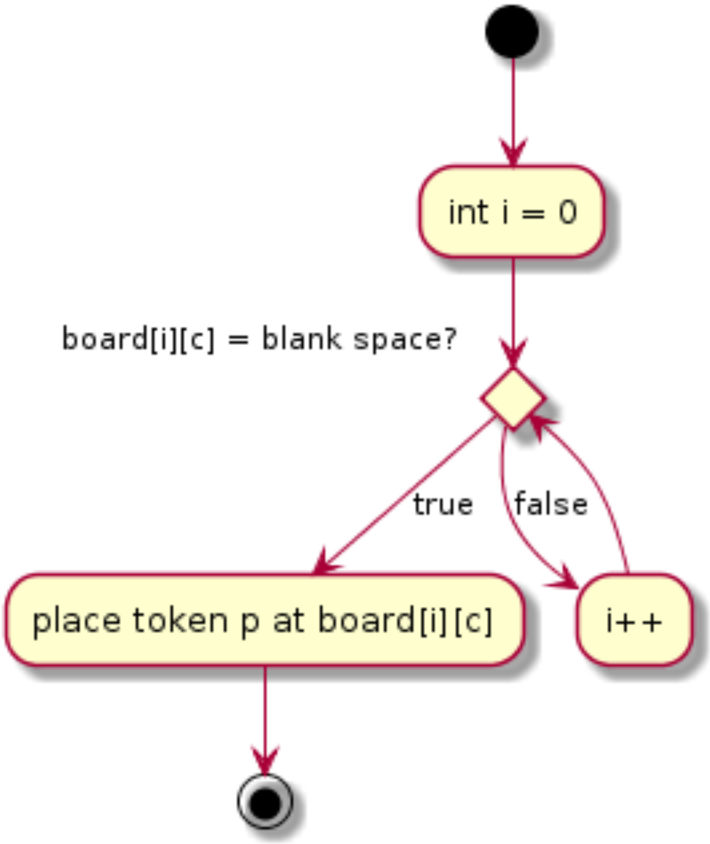
checkVertWin(int r, int c, char p)



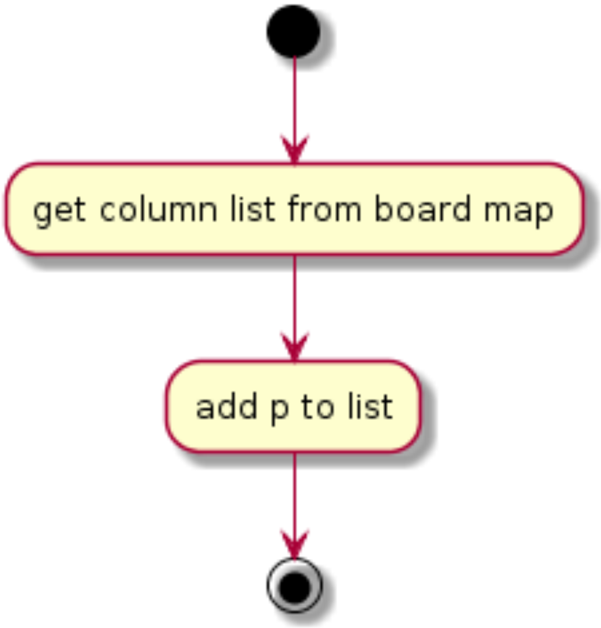
main()


start game
ask player for number of players
false
between 2 and 10?
ask player for number of rows
false
between 3 and 100?
true
ask player for number of columns
false
between 3 and 100?
true
ask player for number to win
false
between 3 and 25?
true
ask player for type of implementation
implement board according to player choice
initialize board and getNumRows() and getNumColumns() from board
current player is player 1
ask player where to place token
true
is choice full or invalid column
false
place token
check if win
true
false
check for tie
announce win
is tie
true
false
announce tie
print board (using toString())
ask to play again
change players
true
answer is yes
false

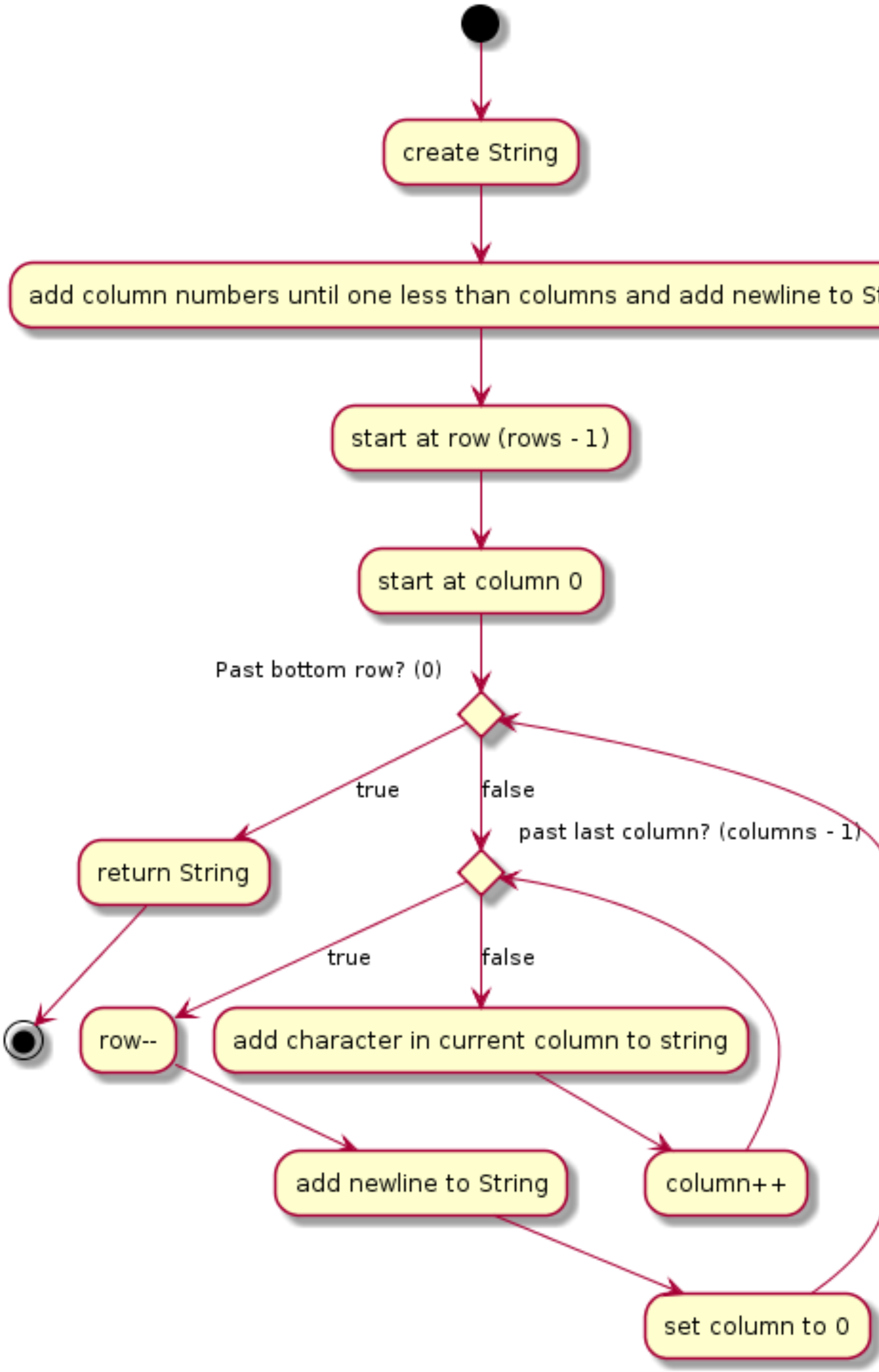
placeToken(char p, int c)



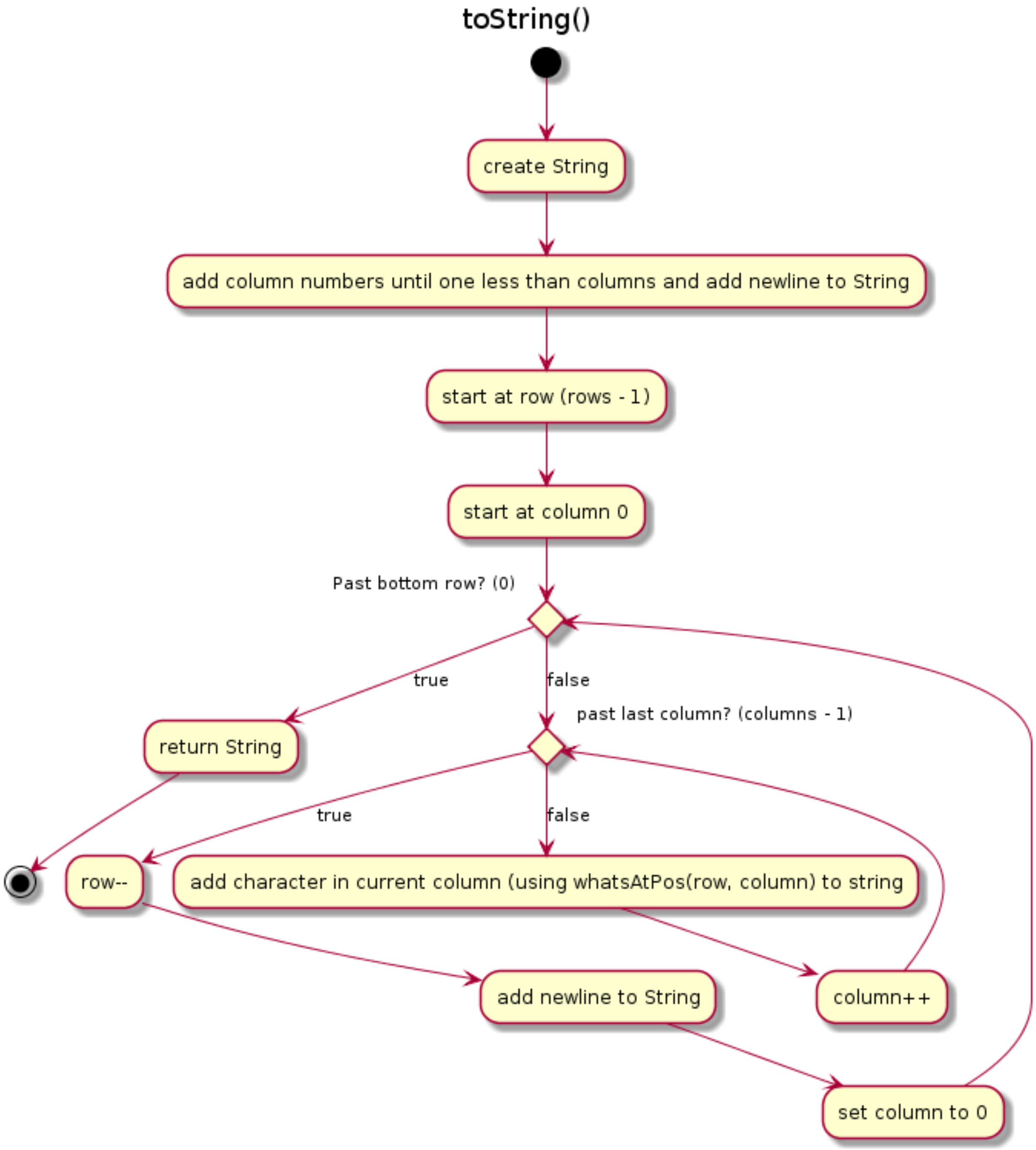
placeToken(char p, int c)



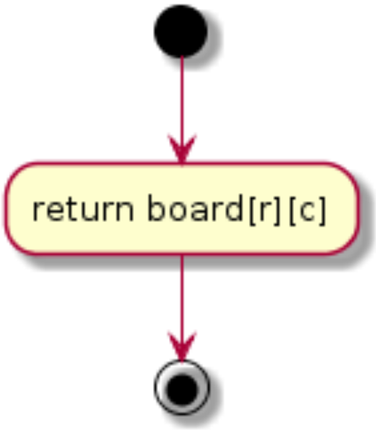
toString()



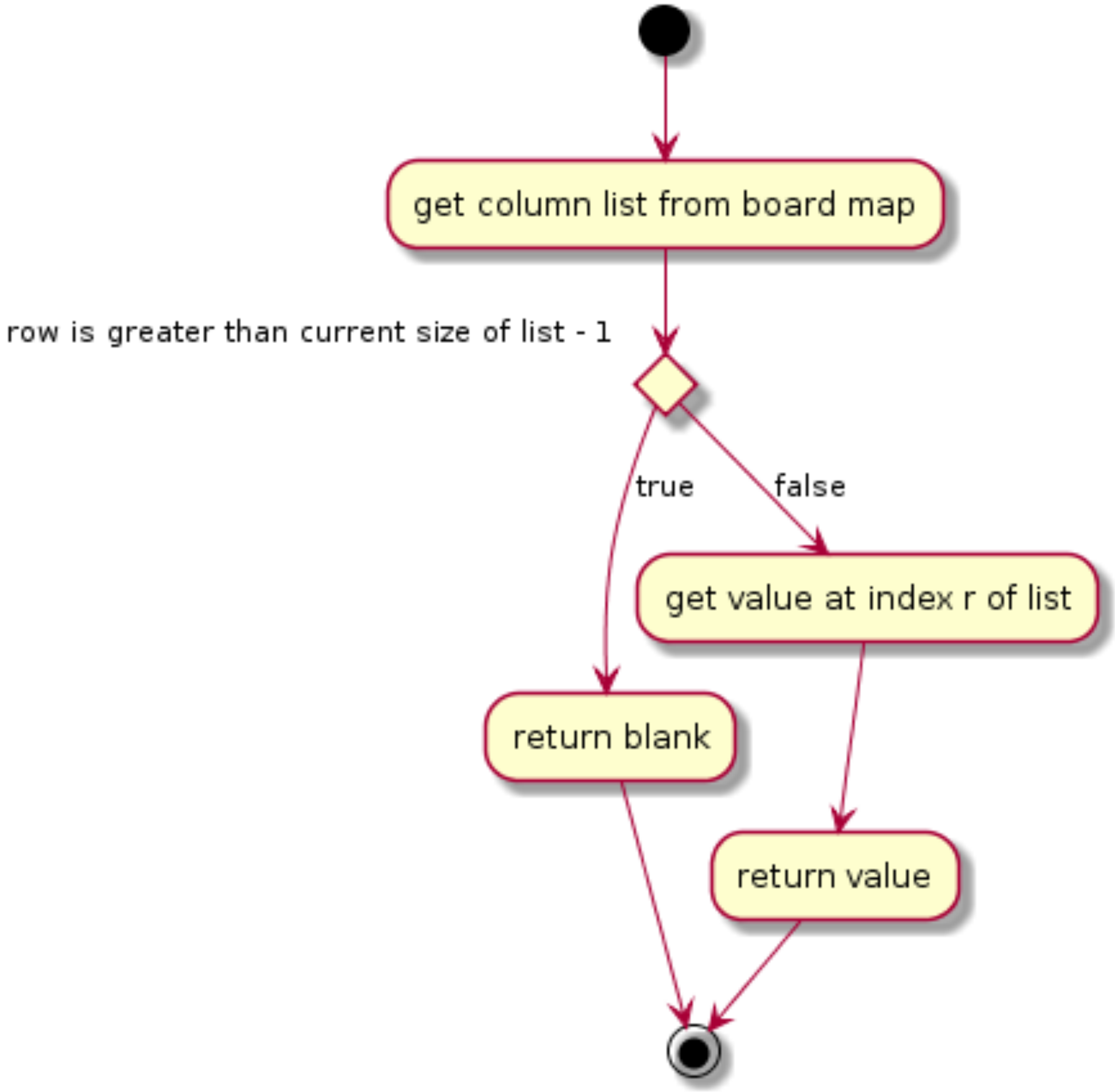
create String
add column numbers until one less than columns and add newline to String
start at row (rows - 1)
start at column 0
Past bottom row? (0)
true
false
return String
past last column? (columns - 1)
true
false
row--
add character in current column to string
add newline to String
column++
set column to 0

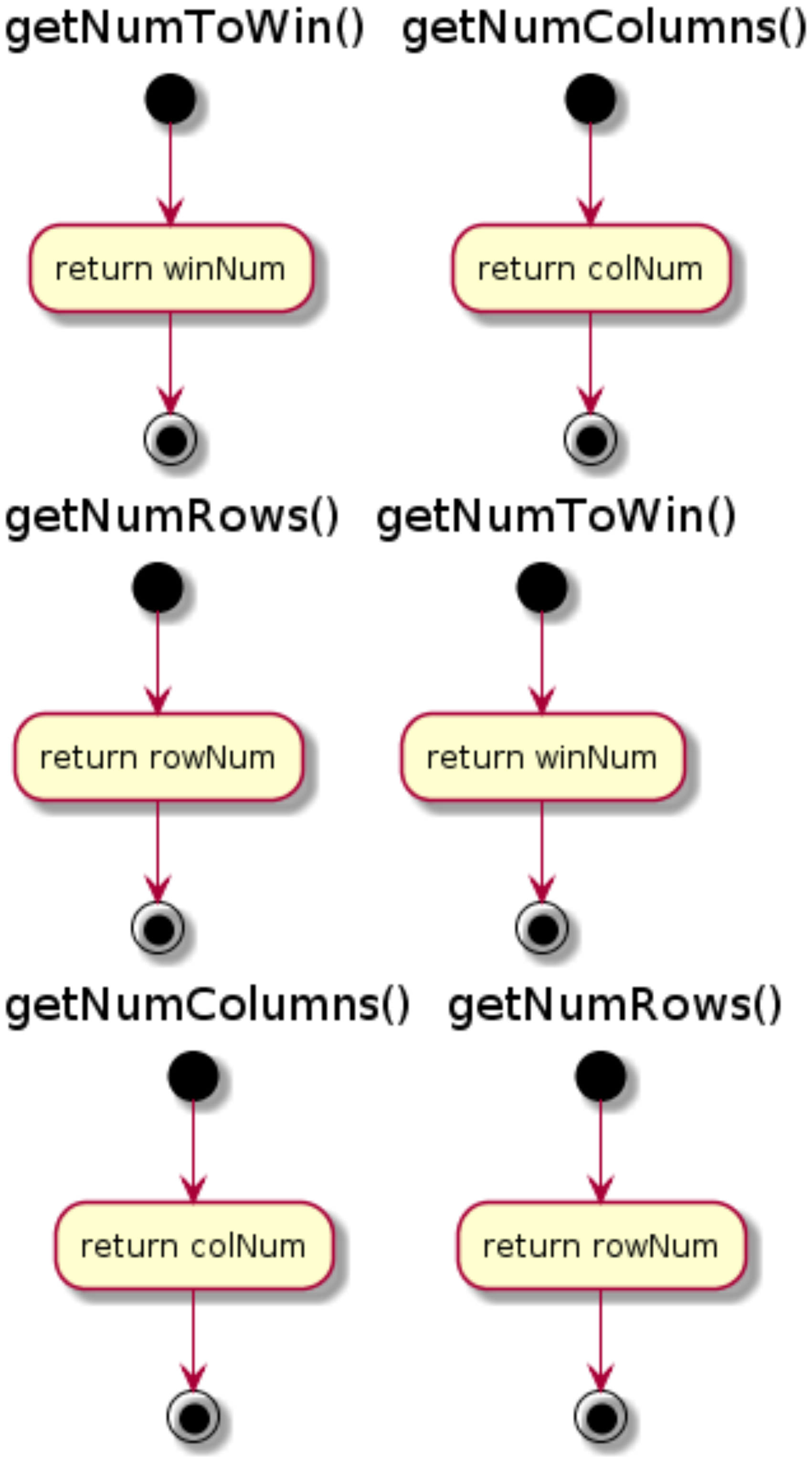

toString()
create String
add column numbers until one less than columns and add newline to String
start at row (rows - 1)
start at column 0
Past bottom row? (0)
true
false
return String
past last column? (columns - 1)
true
false
row--
add character in current column (using whatsAtPos(row, column) to string
add newline to String
column++
set column to 0

whatsAtPos(int r, int c)



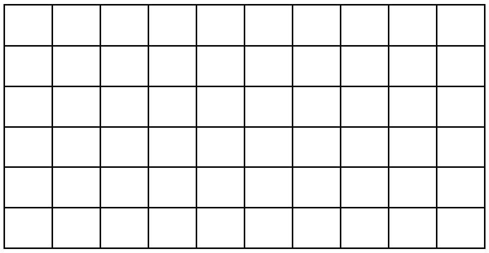
whatsAtPos(int r, int c)


get column list from board map
row is greater than current size of list - 1
true
false
get value at index r of list
return blank
return value


getNumToWin()
return winNum
getNumColumns()
return colNum
getNumRows()
return rowNum
getNumToWin()
return winNum
getNumColumns()
return colNum
getNumRows()
return rowNum

Testing:
The following cases were tested

Constructor

Input:
Columns = 10
Rows = 4
Win = 4
	Output:
	columns = 10
	rows = 4
	number to win = 4
	State:
	
		This test case simply tests a typical use of the constructor to produce a board of 10 by 4 with the number to win set to 4.
		Function:
		testconstructor_4_10_4

Constructor

Input:
Columns = 100
Rows = 100
Win = 25
	Output:
	columns = 100
	rows = 100
	number to win = 25
	State:
	A blank board of 100 by 100 (too large to draw here).
		This test case tests an edge case with the maximum number of columns, rows, and number to win specified in the contracts.
		Function:
		testconstructor_100_100_25

Constructor

<table>
<tr>
<td>Input:
Columns = 3
Rows = 3
Win = 3</td>
<td>Output:
columns = 3
rows = 3
number to win = 3
State:
<table>
<tr><td> </td><td> </td><td> </td></tr>
<tr><td> </td><td> </td><td> </td></tr>
<tr><td> </td><td> </td><td> </td></tr>
</table>
</td>
<td>This case tests an edge case with the minimum number of columns, rows, and number to win specified in the contracts.

Function:
testconstructor_3_3_3</td>
</tr>
</table>

checkIfFree(int c)

<table>
<tr>
<td>Input:
C = 4
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>Output:
checkIfFree = true

state is unchanged</td>
<td>This case tests an edge case with the column to check being the last column and the board being empty.

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

checkIfFree(int c)

<table>
<tr>
<td>Input:
C = 0
State:
<table>
<tr><td> </td><td> </td><td> </td><td> </td><td> </td></tr>
<tr><td>X</td><td> </td><td> </td><td> </td><td> </td></tr>
<tr><td>X</td><td> </td><td> </td><td> </td><td> </td></tr>
<tr><td>X</td><td> </td><td> </td><td> </td><td> </td></tr>
<tr><td>x</td><td>x</td><td>x</td><td>x</td><td>X</td></tr>
</table>
</td>
<td>Output:
checkIfFree = true

state is unchanged</td>
<td>This case tests to make sure checkiffree is testing for full columns and not rows as row 0 is full but <em>column</em> 0 is not.

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

checkIfFree(int c)

<table>
<tr>
<td>Input:
C = 4
State:
<table>
<tr><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td>x</td></tr>
</table>
</td>
<td>Output:
checkIfFree = false

state is unchanged</td>
<td>This case tests an edge case with the column to check being the last column which is full.

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

checkHorizWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 0
c = 2
number to win = 3
p = ‘x’
State:
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td>Output:
checkHorizWin = true

state is unchanged</td>
<td>This case tests a case where the number in a row is more than the number to win. Making sure it tests for at least as many and not an exact count.

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

checkHorizWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 4
c = 5
number to win = 3
p = ‘o’
State:
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>o</td><td>o</td><td>o</td><td>o</td><td>o</td><td></td><td></td></tr>
<tr><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>X</td><td>x</td><td>X</td><td>x</td></tr>
<tr><td>X</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>X</td><td>X</td><td>x</td><td>X</td></tr>
<tr><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>X</td></tr>
</table>
</td>
<td>Output:
checkHorizWin = true

state is unchanged</td>
<td>This case tests a case where winning streak is in the middle of the board to make sure it can test more than the bottom row.

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

checkHorizWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 4
c = 0
number to win = 3
p = ‘o’
State:
<table>
<tr><td>o</td><td>o</td><td>o</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>X</td></tr>
<tr><td>X</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>X</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>X</td></tr>
<tr><td>X</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
</table>
</td>
<td>Output:
checkHorizWin = true

state is unchanged</td>
<td>This case tests an edge case where the winning streak is in the top left of the board.

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

checkHorizWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 0
c = 2
number to win = 5
p = ‘x’
State:

| | | | | | | | | | |
x	x	x	x	x						</td>
<td>Output:
checkHorizWin = true

state is unchanged</td>
<td>This case tests a case where the last token was placed in the middle of the winning streak.

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

checkHorizWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 0
c = 0
number to win = 5
p = ‘x’
State:

|x | | | | | | | | | |
x									
x									
x									
x									|</td>
<td>Output:
checkHorizWin = false

state is unchanged</td>
<td>This case tests a case where the player has won vertically. Checkhorizwin should return false if it is properly testing a horizontal win and not a vertical win.

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

checkVertWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 0
c = 0
number to win = 5
p = ‘x’
State:

| | | | | | | | | | |
x	x	x	x	x						</td>
<td>Output:
checkVertWin = false

state is unchanged</td>
<td>This case tests a case where the player has won horizontally. CheckVertwin should return false if it is properly testing a vertical win and not a horizontal win.

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

checkVertWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 4
c = 4
number to win = 5
p = ‘x’
State:

| | | | |x | | | | | |
				x					
				x					
				x					
			|x					|</td>
<td>Output:
checkVertWin = true

state is unchanged</td>
<td>This case tests a typical case where the player has won in a middle column of the board.

Function:
testcheckvertwin_column4</td>
</tr>
</table>

checkVertWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 5
c = 4
number to win = 5
p = ‘x’
State:

| | | | | | | | | | |
				o					
				x					
				x					
				x					
				x					
			|x					|</td>
<td>Output:
checkVertWin = true

state is unchanged</td>
<td>This case tests a case where the top token on the column is not the one being checked by checkVertWin.

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

checkVertWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 5
c = 4
number to win = 5
p = ‘x’
State:

| | | | | | | | | | |
				x					
				x					
				o					
				x					
				x					
			|x					|</td>
<td>Output:
checkVertWin = false

state is unchanged</td>
<td>
This case tests a case where there are enough tokens for a win in the column checked but they are not consecutive so checkVertWin should return false.

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

checkVertWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 9
c = 9
number to win = 5
p = ‘o’
State:

| | | | | | | | | |o |
									o
									o
									o
									o
									x
									x
									x
									x
								|x |</td>
<td>Output:
checkVertWin = true

state is unchanged</td>
<td>
This case tests an edge case where the row and column checked are the last ones.

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

checkDiagWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 4
c = 4
number to win = 5
p = ‘x’
State:

| | | | | | | | | | |
				x					
			x	x					
		x	x	x					
	x	x	x	x					
x	x	x	x	x						</td>
<td>Output:
checkDiagWin = true

state is unchanged</td>
<td>
This case tests a typical case where the user has won diagonally up left to right.
Function:
testcheckdiagwin_typicalcaseup</td>
</tr>
</table>

checkDiagWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 4
c = 0
number to win = 5
p = ‘x’
State:

| | | | | | | | | | |
x									
x	x								
x	x	x							
x	x	x	x						
x	x	x	x	x					
x	x	x	x	x	x					</td>
<td>Output:
checkDiagWin = true

state is unchanged</td>
<td>
This case tests a typical case where the user has won diagonally down left to right.
Function:
testcheckdiagwin_typicalcasedown</td>
</tr>
</table>

checkDiagWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 2
c = 3
number to win = 6
p = ‘x’
State:

| | | | | | | | | | |
x									
x	x								
x	x	x							
x	x	x	x						
x	x	x	x	x					
x	x	x	x	x	x					</td>
<td>Output:
checkDiagWin = true

state is unchanged</td>
<td>This case tests a typical case where the last token needed to win was placed in the middle of the diagonal streak.
Function:
testcheckdiagwin_lastinmiddle</td>
</tr>
</table>

checkDiagWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 9
c = 0
number to win = 5
p = ‘x’
State:

|x | | | | | | | | | |
x	x								
x	x	x							
x	x	x	x						
o	x	x	x	x					
o	o	x	x	x	x				
o	o	o	x	x	x				
o	o	o	o	x	x				
o	o	o	o	o	x				
o	o	o	o	o	o					</td>
<td>Output:
checkDiagWin = true

state is unchanged</td>
<td>This case testsan edge case where the check starts at the top left corner of the board. This would place 3 legs of the x pattern to check out of bounds.
Function:
testcheckdiagwin_startatcorner</td>
</tr>
</table>

checkDiagWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 2
c = 2
number to win = 6
p = ‘x’
State:

| | | | | | | | | | |
					x				
				x	x				
			x	x	x				
		x	x	x	x				
	x	x	x	x	x				
x	x	x	x	x	x					</td>
<td>Output:
checkDiagWin = true

state is unchanged</td>
<td>This case tests a typical case where the last token needed to win was placed in the middle of the diagonal streak. This time it tests the other diagonal
Function:
testcheckdiagwin_lastinmiddlereverse</td>
</tr>
</table>

checkDiagWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 4
c = 4
number to win = 5
p = ‘x’
State:

| | | | | | | | | |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	x	x
				x	x	x	x	x	x
				x	x	x	x	x	x
				x	x	x	x	x	x	</td>
<td>Output:
checkDiagWin = true

state is unchanged</td>
<td>This case tests an edge case where the check starts at the last row and last column. This would place 3 legs of the x pattern to check out of bounds.
Function:
testcheckdiagwin_rightcorner</td>
</tr>
</table>

checkDiagWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 4
c = 0
number to win = 5
p = ‘o’
State:

| | | | | | | | | | |
x									
x	x								
x	x	x							
x	x	x	x						
x	x	x	x	x					
x	x	x	x	x	x					</td>
<td>Output:
checkDiagWin = false

state is unchanged</td>
<td>
This case tests an edge case where the winning conditions for another token are present but it is not the token asked for.
Function:
testcheckdiagwin_wrongtoken</td>
</tr>
</table>

checkDiagWin (int r, int c, char p)

<table>
<tr>
<td>Input:
r = 4
c = 7

number to win = 5
p = ‘x’
State:

| | | | | | | | | | |
							x		
						x	x		
					x	x	x		
				x	x	x	x		
			x	x	x	x	x			</td>
<td>Output:
checkDiagWin = true

state is unchanged</td>
<td>
This case tests a typical case where winning conditions are present in the middle of the board.
Function:
testcheckdiagwin_typicalmiddleboard</td>
</tr>
</table>

checkTie ()

<table>
<tr><td>Input:
number to win = 5
State:
|o |p |o |p |o |p |o |p |o |p |
x	f	x	f	x	f	x	f	x	f
o	p	o	p	o	p	o	p	o	p
x	f	x	f	x	f	x	f	x	f
o	p	o	p	o	p	o	p	o	p
x	f	x	f	x	f	x	f	x	f
o	p	o	p	o	p	o	p	o	p
x	f	x	f	x	f	x	f	x	f
o	p	o	p	o	p	o	p	o	p
x	f	x	f	x	f	x	f	x	f	</td><td>Output: checkTie = true state is unchanged</td><td>This case tests a typical tie where every column is full. Function: testchecktie_typical</td></tr>
</table>

checkTie ()

<table>
<tr><td>Input:
number to win = 5
State:
|z |t |z |t |z |t |z |t |z |t |
o	b	o	b	o	b	o	b	o	b
z	t	z	t	z	t	z	t	z	t
o	b	o	b	o	b	o	b	o	b
z	t	z	t	z	t	z	t	z	t
o	b	o	b	o	b	o	b	o	b
z	t	z	t	z	t	z	t	z	t
o	b	o	b	o	b	o	b	o	b
z	t	z	t	z	t	z	t	z	t
o	b	o	b	o	b	o	b	o	b	</td><td>Output: checkTie = true state is unchanged</td><td>This case tests another typical tie where every column is full. Function: testchecktie_typical2</td></tr>
</table>

checkTie ()

<table>
<tr>
<td>Input:
number to win = 5
State:
| | | | | | | | | | |
									|</td>
<td>Output:
checkTie = false

state is unchanged</td>
<td>This tests an edge case where the board is empty.
Function:
testchecktie_empty</td>
</tr>
</table>

checkTie ()

<table>
<tr>
<td>Input:
number to win = 5
State:
|o | | | | | | | | | |
x									
o									
x									
o									
x									
o									
x									
o									
x									|</td>
<td>Output:
checkTie = false

state is unchanged</td>
<td>This tests a case where one column is filled without winning.
Function:
testchecktie_firstcol</td>
</tr>
</table>

whatsAtPos (int r, int c)

<table>
<tr>
<td>Input:
r = 2
c = 3
State:
| | | | | | | | | | |
									|</td>
<td>Output:
whatsAtPos = ‘ ‘

state is unchanged</td>
<td>This tests a case getting a character from an empty space in the board, since every space on the board should be initialized to a blank space.
Function:
testwhatsatpos_empty</td>
</tr>
</table>

placeToken (char p, int c)

<table>
<tr>
<td>Input:
p = ‘x’
c = 2
State:
| | | | | | | | | | |
									|</td>
<td>Output: State: |									
	|x							|</td>
<td>This tests a typical case placing a token onto an empty board.
Function:
testplacetoken_empty</td>
</tr>
</table>

placeToken (char p, int c)

<table>
<tr>
<td>Input:
p = ‘x’
c = 0
State:
| | | | | | | | | | |
									|</td>
<td>Output: State: |									
x									|</td>
<td>This tests an edge case placing a token in the first column. 0 is the minimum value for c according to the contracts.
Function: testplacetoken_first</td>
</tr>
</table>

placeToken (char p, int c)

<table>
<tr>
<td>Input:
p = ‘x’
c = 9
State:
| | | | | | | | | | |
									|</td>
<td>Output: State: |									
								|x |</td>
<td>This tests an edge case placing a token in the first column. Col - 1 is the maximum value for c according to the contracts.
Function: testplacetoken_last</td>
</tr>
</table>

placeToken (char p, int c)

<table>
<tr>
<td>Input:
p = ‘o’
c = 0
State:
| | | | | | | | | | |
x									|</td>
<td>Output: State: |									
o									
x									|</td>
<td>This tests a case where there is already a token in the column provided. This tests to make sure placeToken stacks the tokens.
Function:
testplacetoken_stack</td>
</tr>
</table>

placeToken (char p, int c)

<table>
<tr>
<td>Input:
p = ‘x’
c = 4
State:
| | | | | | | | | | |
				x					
				x					
				x					
				x					
				x					
				x					
				x					
				x					
			|x					|</td>
<td>Output: State: |			|x					
				x					
				x					
				x					
				x					
				x					
				x					
				x					
				x					
			|x					|</td>
<td>This tests a case where the column is filled by the function, making sure it can fill all the way to the top of a column.
Function:
testplacetoken_fillcol</td>
</tr>
</table>

whatsAtPos (int r, int c)

<table>
<tr><td>Input:
r = 0
c = 0
State:
| | | | | | | | | | |
x									|</td><td>Output: whatsAtPos = ‘x‘ state is unchanged</td><td>This tests an edge case getting the token in the bottom left corner, making c = 0 and r = 0, which are the minimums. Function: testwhatsatpos_bottomleftcorner</td></tr>
</table>

whatsAtPos (int r, int c)

<table>
<tr><td>Input:
r = 0
c = 9
State:
| | | | | | | | | | |
								|x |</td><td>Output: whatsAtPos = ‘x‘ state is unchanged</td><td>This tests an edge case getting the token in the bottom right corner, making c =9 and r = 0, which is the max column. Function: testwhatsatpos_bottomrightcorner</td></tr>
</table>

whatsAtPos (int r, int c)

<table>
<tr>
<td>Input:
r = 9
c = 9
State:
| | | | | | | | | |x |
									o
									o
									o
									o
									o
									o
									o
									o
								|o |</td>
<td>Output:
whatsAtPos = ‘x‘

state is unchanged</td>
<td>This tests an edge case getting the token in the top right corner, making c =9 and r = 9, which is the max column and max row.
Function:
testwhatsatpos_toprightcorner</td>
</tr>
</table>

whatsAtPos (int r, int c)

<table>
<tr>
<td>Input:
r = 9
c = 0
State:
|x | | | | | | | | | |
o									
o									
o									
o									
o									
o									
o									
o									
o									|</td>
<td>Output:
whatsAtPos = ‘x‘

state is unchanged</td>
<td>This tests an edge case getting the token in the top left corner.
Function:
testwhatsatpos_topleftcorner</td>
</tr>
</table>

whatsAtPos (int r, int c)

<table>
<tr>
<td>Input:
r = 3
c = 0
State:
|x | | | | | | | | | |
o									
x									
o									
x									
o									
x									
o									
x									
o									|</td>
<td>Output:
whatsAtPos = ‘x‘

state is
unchanged</td>
<td>This tests a case where the position to be gotten is in the middle of a stack and not at the top.
Function:
testwhatsatpos_nottop</td>
</tr>
</table>

whatsAtPos (int r, int c)

<table>
<tr>
<td>Input:
r = 0
c = 2
State:
|x | | | | | | | | | |
o									
x									
o									
x									
o									
x									
o									
x									
o	|x							|</td>
<td>Output:
whatsAtPos = ‘x‘

state is
unchanged</td>
<td>This tests a case where if the function mixed up the row and column, it would return the wrong token. This is testing to make sure the function is not mixing up the coordinates.
Function:
testwhatsatpos_checkorder</td>
</tr>
</table>

Deployment

Unzip DanielToroHW4.zip
In the generated directory, run make to compile
To run: make run
Run make clean to remove .class files
To test: make test && make runtest