Paige Clemons (pclemon)
Lab Section 5
Homework 4
Due: 11/18/18

Requirement Analysis

Functional Requirements

- Player should be able to see the board in order to know where they can place their tokens.
- Players should be able to choose where to place their tokens in order to effectively play the game to beat their opponent.
- Players should be able to choose the size of the board so that they can play the game on whatever size board they want.
- Players should be able to choose how many tokens in a row they need to win so that they can play the game in the way they choose.
- Players should be able to choose if they want to play again in order to have a rematch with an opponent.
- Players should be able to choose how many players they want to play with in order to play with however many people they want to.
- Players should be able to choose the token they want in order to have a unique identifier for where they placed their token.
- Players should be able to choose between a fast or memory efficient implementation of the game so that they can play using whichever implementation they would like.

Non-functional Requirements

- Usability
    - An updated game board should be printed to the screen after every move.
    - A prompt should display which players turn it is, and it should ask where the player wants to move next.
    - The player should also be asked if he would like to play again.
- Reliability
    - A prompt should also display stating that a column is full if a player tries to put a token in that column.
    - A prompt should also display if the player tries to input an invalid column number.
    - A prompt should display if the player inputs and invalid board size or invalid winning token amount.
- Performance
    - The game should keep track of two players data.
    - The data should be handled in the most efficient way possible.
- Supportability
    - The game should be playable between two players.
    - The program can run on any computer with Java installed.
- Implementation
    - The program must be coded in Java.
    - The program must be able to run on Unix.
    - Comments should be added to better understand the code.

- Javadoc comments should also be added to know the requirements of the classes and functions.
- The program should also use an interface for the methods of the class and have an interface specification.
- The program should have two implementation of the game a fast implementation and a memory efficient implementation.

Testing

Constructor

- Test 1(testConstructor_row3_column3_token3)
  - Input: row = 3, column = 3, tokens = 3
  - Expected output: an empty board of size 3x3 and rows, columns, and tokens to all equal 3
  - Reason: I chose this test case to test the smallest board and minimum number of tokens needed to win that is possible. So, this is considered the lower bound of the program making this a test of a boundary case.
- Test 2 (testConstructor_row100_column100_token25)
  - Input: row = 100, column = 100, tokens = 25
  - Expected output: an empty board of size 100x100 and rows and columns to equal 100 and tokens to equal 25
  - Reason: I chose this test case to test the largest board and maximum number of tokens needed to win that is possible. So, this is considered the upper bound of the program making this a test of a boundary case.
- Test 3 (testConstructor_row6_column7_token4)
  - Input: row = 6, column = 7, tokens = 4
  - Expected output: an empty board of size 6x7 and rows equal to 6, columns equal to 7, tokens equal to 4
  - Reason: I chose this test case to test a more routine scenario. After check that the boundary cases did work, I wanted to make sure something in the middle worked as well.

checkIfFree

- Test 1(testCheckIfFree_full_column)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

O						
X						
O						
X						
O						
X						

  - Input: I then called checkIfFree passing in column 0.
  - Expected output: I expected to get false from the call to checkIfFree, and the board to look like the one above.
  - Reason: I chose this test case because I wanted to test a case that a column was completely full. I would consider this a fairly routine case. I would also consider this a boundary case.

- Test 2 (testCheckIfFree_half_full_column)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

X						
O						
X						

    - Input: I then called checkIfFree passing in column 0.
    - Expected output: I expected to get true from the call to checkIfFree, and the board to look like the one above.
    - Reason: I chose this test case because I wanted to test a case where the column had tokens in it, but it wasn't completely full. I would consider this a more challenging case.
- Test 3 (testCheckIfFree_empty_column)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - I then left the board empty.
    - Input: I then called checkIfFree passing in column 0.
    - Expected output: I expected to get true from the call to checkIfFree, and the board that was completely empty.
    - Reason: I chose this test case because I wanted to test a case that had absolutely no tokens in the column. I would consider this both a routine and a boundary case.

<u>checkHorizWin</u>

- Test 1 (testCheckHorizWin_check_right)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

X	X	X	X			

      *The red X was the last token that was placed therefore that was passed into the function
    - Input: I then called checkHorizWin passing in row 5, column 0, and player token X
    - Expected output: I expected to get to get true from the call to checkHorizWin, and the board to look like the one above.
    - Reason: I chose this test to make sure that the function was properly checking to the right of the last placed token. I would consider this a routine case.

- Test 2 (testCheckHorizWin_check_left)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

X	X	X	X			

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkHorizWin passing in row 5, column 3, and player token X.
  - Expected output: I expected to get true from the call to checkHorizWin, and the board to look like the one above.
  - Reason: I chose this test to make sure that the function was properly checking to the left of the last placed token. I would consider this a routine case.
- Test 3 (testCheckHorizWin_check_middle)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

X	X	X	X			

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkHorizWin passing in row 5, column 1, and player token X.
  - Expected output: I expected to get true from the call to checkHorizWin, and the board to look like the one above.
  - Reason: I chose this test to make sure that the function would check both directions. I would consider this a challenging case since the function has to check in both directions.

- Test 4 (testCheckHorizWin_check_middle_more_tokens_than_needed)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

X	X	X	X	X		

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkHorizWin passing in row 5, column 2, and player token X.
  - Expected output: I expected to get true from the call to checkHorizWin, and the board to look like the one above.
  - Reason: I chose this test to make sure I would still get a win if there were more tokens in a row than the amount needed which in this case is 4. I would consider this challenging because it is easy to forget about a case like this while coding.
- Test 5 (testCheckHorizWin_no_win)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

X	O	X	X	X		

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkHorizWin passing in row 5, column 2, and player token X.
  - Expected output: I expected to get false from the call to checkHorizWIn, and the board to look like the one above.
  - Reason: I chose this test to make sure my function would not count this as a win since it still had 4 Xs, but they weren't in a row. I would consider this a challenging case since the Xs aren't in a row.

checkVertWin

- Test 1 (testCheckVertWin_four_in_a_row_from_bottom)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

X						
X						
X						
X						

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkVertWin passing in row 2, column 0, and player token X.
  - Expected output: I expected to get true from the call to checkVertWin, and the board to look like the one above.
  - Reason: I chose this test to make sure my function was working for the most simplistic case. I would consider this a routine case.
- Test 2 (testCheckVertWin_four_in_a_row_from_next_to_bottom)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

X						
X						
X						
X						
O						

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkVertWin passing in row 1, column 0, and player token X.
  - Expected output: I expected to get true from the call to checkVertWin, and the board to look like the one above.
  - Reason: I chose this test to make sure that the O at the bottom of the column would not interfere with the fact that this was still a win. I would still consider this a routine case.

- Test 3 (testCheckVertWin_close_to_win)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

X						
X						
O						
X						
X						

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkVertWin passing in row 1, column 0, and player token X.
  - Expected output: I expected to get false from the call to checkVertWin, and the board to look like the one above.
  - Reason: I chose this test to make sure that my function would not count this as a win since there is four Xs even though they weren't in a row. I would consider this a challenging case.
- Test 4 (testCheckVertWin_obvious_loss)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

O						
X						
O						
X						
O						
X						

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkVertWin passing in row 0, column 0, and player token O.
  - Expected output: I expected to get false from the call to checkVertWin, and the board to look like the one above.
  - Reason: I chose this test because there weren't any number of tokens in a row. This is a challenging case since there aren't any tokens in a row.

- Test 5 (testCheckVertWin_close_to_win_three_in_a_row)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

X						
X						
X						

*The red X was the last token that was placed therefore that was passed into the function

    - Input: I then called checkVertWin passing in row 3, column 0, and player token O.
    - Expected output: I expected to get false from the call to checkVertWin, and the board to look like the board above.
    - Reason: I chose this test because there were only three Xs in a row as opposed to four in a row which was needed. I consider this a routine case.

<u>checkDiagWin</u>

- Test 1 (testCheckDiagWin_four_in_a_row_top_right)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

						X
					X	O
				X	O	O
			X	O	O	O

*The red X was the last token that was placed therefore that was passed into the function

    - Input: I then called checkDiagWin passing in row 2, column 6, and player token X.
    - Expected output: I expected to get true from the call to checkDiagWin, and the board to look like the board above.
    - Reason: I chose this test to make sure my function would check going from the top right to the bottom left correctly. I would consider this case routine since this test only has to check one direction.

- Test 2 (testCheckDiagWin_four_in_a_row_top_left)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

			X			
			O	X		
			O	O	X	
			O	O	O	X

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkDiagWin passing in row 2, column 3, and player token X.
  - Expected output: I expected to get true from the call to checkDiagWin, and the board to look like the board above.
  - Reason: I chose this test to make sure my function would check going from the top left to the bottom right correctly. I would consider this case routine since this test only has to check one direction.
- Test 3 (testCheckDiagWin_four_in_a_row_bottom_right)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

			X			
			O	X		
			O	O	X	
			O	O	O	X

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkDiagWin passing in row 5, column 6, and player token X.
  - Expected output: I expected to get true from the call to checkDiagWin, and the board to look like the board above.
  - Reason: I chose this test to make sure my function would check going from the bottom right to the top left correctly. I would consider this case routine since this test only has to check one direction.

- Test 4 (testCheckDiagWin_four_in_a_row_bottom_left)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

						X
					X	O
				X	O	O
			X	O	O	O

*The red X was the last token that was placed therefore that was passed into the function

    - Input: I then called checkDiagWin passing in row 5, column 3, and player token X.
    - Expected output: I expected to get true from the call to checkDiagWin, and the board to look like the one above.
    - Reason: I chose this case to make sure my function would check going from the bottom left to the top right correctly. I would consider this case routine since this test only has to check one direction.
- Test 5 (testCheckDiagWin_four_in_a_row_middle)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

						X
					X	O
				X	O	O
			X	O	O	O

*The red X was the last token that was placed therefore that was passed into the function

    - Input: I then called checkDiagWin passing in row 4, column 4, and player token X.
    - Expected output: I expected to get true from the call to checkDiagWin, and the board to look like the one above.
    - Reason: I chose this case since the last placed token is in the middles which mean my function has to check in two directions. I would consider this case challenging since it does have to check in two directions

- Test 6 (testCheckDiagWin_middle_more_tokens_than_needed)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

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

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkDiagWin passing in row 3, column 4, and player token X.
  - Expected output: I expected to get true from the call to checkDiagWin, and the board to look like the one above.
  - Reason: I chose this test to make sure I would still get a win if there were more tokens in a row than the amount needed which in this case is 4. I would consider this challenging because it is easy to forget about a case like this while coding.
- Test 7 (testCheckDiagWin_close_to_a_win)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

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

*The red X was the last token that was placed therefore that was passed into the function

  - Input: I then called checkDiagWin passing in row 3, column 4, and player token X.
  - Expected output: I expected to get false from the call to checkDiagWin, and the board to look like the one above.
  - Reason: I chose this test to make sure my function would not count this as a win since it still had 4 Xs, but they weren't in a row. I would consider this a challenging case since the Xs aren't in a row.

- Test 8 (testCheckDiagWin_obvious_no_win)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

	X					
	O	X				
	O	O	X			
	O	X	O			

*The red X was the last token that was placed therefore that was passed into the function

    - Input: I then called checkDiagWin passing in row 3, column 2, and player token X.
    - Expected output: I expected to get false from the call to checkDiagWin, and the board to look like the one above.
    - Reason: I chose this test because I remember this test returned true on the first homework while I was still coding. This is a challenging case because the function needs to check in all direction but should not count all four of the Xs since they are not in a row.

<u>checkTie</u>

- Test 1 (testCheckTie_completely_full_board)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

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

    - Input: I then called checkTie.
    - Expected output: I expected to get true from the call to checkTie, and the board to look like the one above.
    - Reason: I chose this case to test the function on a completely full board. I would consider this a routine case as well as a boundary case.

- Test 2 (testCheckTie_one_token_shy_of_a_completely_full_board)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

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

    - Input: I then called checkTie.
    - Expected output: I expected to get false from the call to checkTie, and the board to look like the one above.
    - Reason: I chose this case to test the function on a board that was one token shy of being a tie. I would consider this a challenging case.
- Test 3 (testCheckTie_one_whole_column_empty)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

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

    - Input: I then called checkTie.
    - Expected output: I expected to get false from the call to checkTie, and the board to look like the one above.
    - Reason: I chose this test because it was something in the middle of being completely full and completely empty. I would consider this routine.
- Test 4 (testCheckTie_completely_empty_board)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - I then left the board completely empty.
    - Input: I then called checkTie.
    - Expected output: I expected to get false from the call to checkTie, and the board to be completely empty.
    - Reason: I chose this test case because I wanted to test a case that had absolutely no tokens in the board. I would consider this both a routine and a boundary case.

whatsAtPos

- Test 1 (testWhatsAtPos_bottom_left_with_token)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

X						

    - Input: I then called whatsAtPos passing in row 5 and column 0.
    - Expected output: I expected to get the char X from the call to whatsAtPos, and the board to look like the one above.
    - Reason: I chose this case as way to make sure I didn't go out the bounds of the array, list, or map when retrieving the bottom left corner of the board. I would consider this a challenging case since it is at the boundary of the data structure. I would also consider this a boundary case.
- Test 2 (testWhatsAtPos_bottom_left_empty)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - I left the board completely empty.
    - Input: I then called whatsAtPos passing in row 5 and column 0.
    - Expected output: I expected to get a blank space from the call to whatsAtPos, and the board to be completely empty.
    - Reason: I chose this case as way to make sure I didn't go out the bounds of the array, list, or map when retrieving the bottom left corner of the board. I also wanted to make sure the function would return a blank space. I would consider this a challenging case since it is at the boundary of the data structure. I would also consider this a boundary case.
- Test 3 (testWhatsAtPos_top_left_with_token)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

A						
X						
O						
X						
O						
X						

    - Input: I then called whatsAtPos passing in row 0 and column 0.
    - Expected output: I expected to get the char A from the call to whatsAtPos, and the board to look like the one above.
    - Reason: I chose this case as way to make sure I didn't go out the bounds of the array, list, or map when retrieving the top left corner of the board. I would

consider this a challenging case since it is at the boundary of the data structure. I would also consider this a boundary case.

- Test 4 (testWhatsAtPos_top_left_no_token)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

X						
O						
X						
O						
X						

  - Input: I then called whatsAtPos passing in row 0 and column 0.
  - Expected output: I expected to get a blank space from the call to whatsAtPos, and the board to be completely empty.
  - Reason: I chose this case as way to make sure I didn't go out the bounds of the array, list, or map when retrieving the top left corner of the board. I also wanted to make sure the function would return a blank space. I would consider this a challenging case since it is at the boundary of the data structure. I would also consider this a boundary case.
- Test 5 (testWhatsAtPos_bottom_right_with_token)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

						X

  - Input: I then called whatsAtPos passing in row 5 and column 6.
  - Expected output: I expected to get the char X from the call to whatsAtPos, and the board to look like the one above.
  - Reason: I chose this case as way to make sure I didn't go out the bounds of the array, list, or map when retrieving the bottom right corner of the board. I would consider this a challenging case since it is at the boundary of the data structure. I would also consider this a boundary case.
- Test 6 (testWhatsAtPos_bottom_right_no_token)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - I left the board completely empty.
  - Input: I then called whatsAtPos passing in row 5 and column 6.
  - Expected output: I expected to get a blank space from the call to whatsAtPos, and the board to be completely empty.
  - Reason: I chose this case as way to make sure I didn't go out the bounds of the array, list, or map when retrieving the bottom right corner of the board. I also

wanted to make sure the function would return a blank space. I would consider this a challenging case since it is at the boundary of the data structure. I would also consider this a boundary case.

- Test 7 (testWhatsAtPos_top_right_with_token)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

						A
						X
						O
						X
						O
						X

  - Input: I then called whatsAtPos passing in row 0 and column 6.
  - Expected output: I expected to get the char A from the call to whatsAtPos, and the board to look like the one above.
  - Reason: I chose this case as way to make sure I didn't go out the bounds of the array, list, or map when retrieving the top right corner of the board. I would consider this a challenging case since it is at the boundary of the data structure. I would also consider this a boundary case.
- Test 8 (testWhatsAtPos_top_right_no_token)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

						X
						O
						X
						O
						X

  - Input: I then called whatsAtPos passing in row 0 and column 6.
  - Expected output: I expected to get a blank space from the call to whatsAtPos, and the board to be completely empty.
  - Reason: I chose this case as way to make sure I didn't go out the bounds of the array, list, or map when retrieving the top right corner of the board. I also wanted to make sure the function would return a blank space. I would consider this a challenging case since it is at the boundary of the data structure. I would also consider this a boundary case.

placeToken

- Test 1 (testPlaceToken_one_token_on_empty_board)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

			X			

Since this is testing the placeToken function there was no other input

  - Expected output: I expected the board to look like the one above.
  - Reason: I chose this test to make sure that just a single call to placeToken was working. I would consider this a routine case.
- Test 2 (testPlaceToken_filling_half_of_a_column)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

			X			
			X			
			X			

Since this is testing the placeToken function there was no other input

  - Expected output: I expected the board to look like the one above.
  - Reason: I chose this test to make sure that multiple calls to placeToken in a single column was working. I would consider this a more challenging case.
- Test 3 (testPlaceToken_completely_filling_a_column)
  - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
  - This is the board that I created:

			X			
			X			
			O			
			X			
			X			
			X			

Since this is testing the placeToken function there was no other input

  - Expected output: I expected the board to look like the one above.
  - Reason: I chose this test to make sure that filling an entire column with tokens would work. I would consider this a more challenging case.

- Test 4 (testPlaceToken_placing_tokens_in_different_columns)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

			X			
X			X	O		O

Since this is testing the placeToken function there was no other input

    - Expected output: I expected the board to look like the one above.
    - Reason: I chose this test to make sure that filling that placing tokens in random spots on the board was working. I would consider this a more challenging case.

- Test 5 (testPlaceToken_completely_filling_a_row)
    - I created a board with the following attributes: row = 6, column = 7, and tokens = 4
    - This is the board that I created:

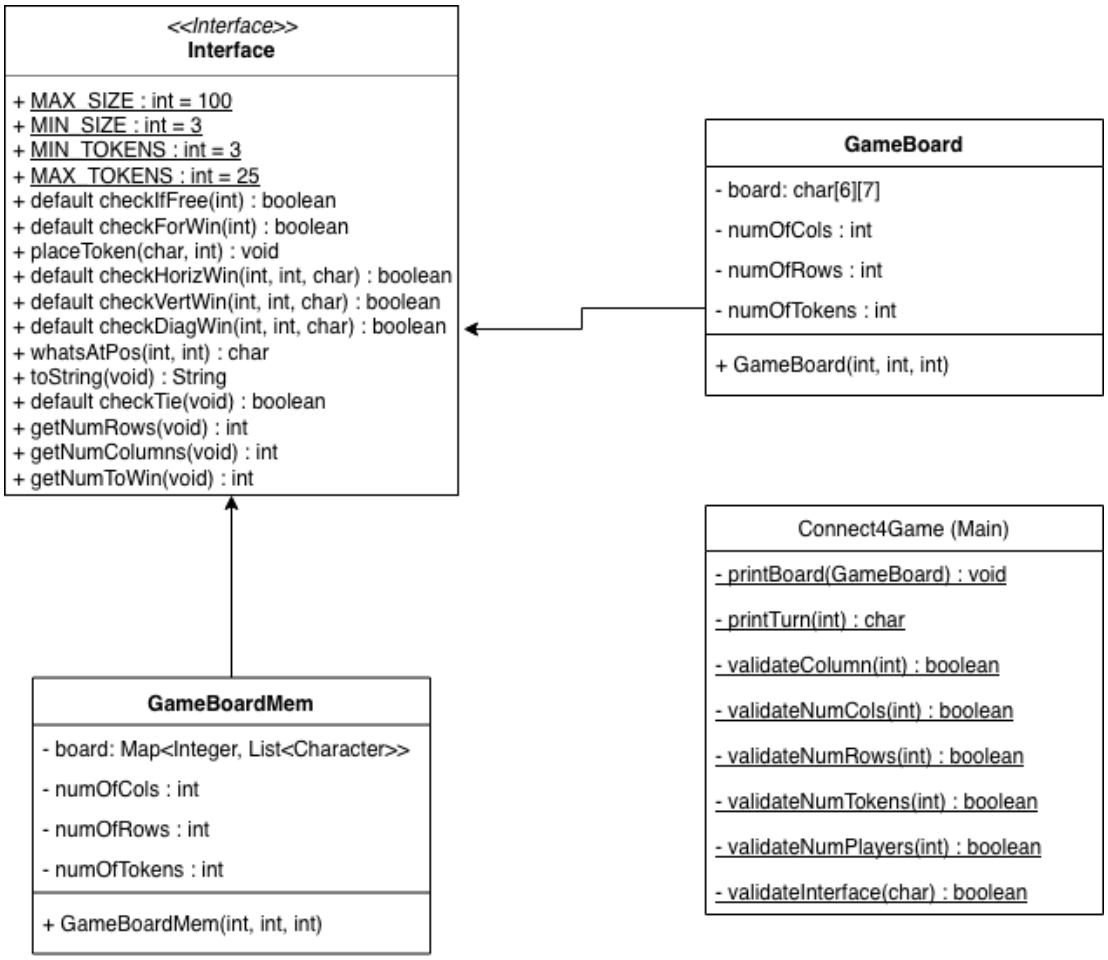
X	O	X	O	X	O	X

Since this is testing the placeToken function there was no other input

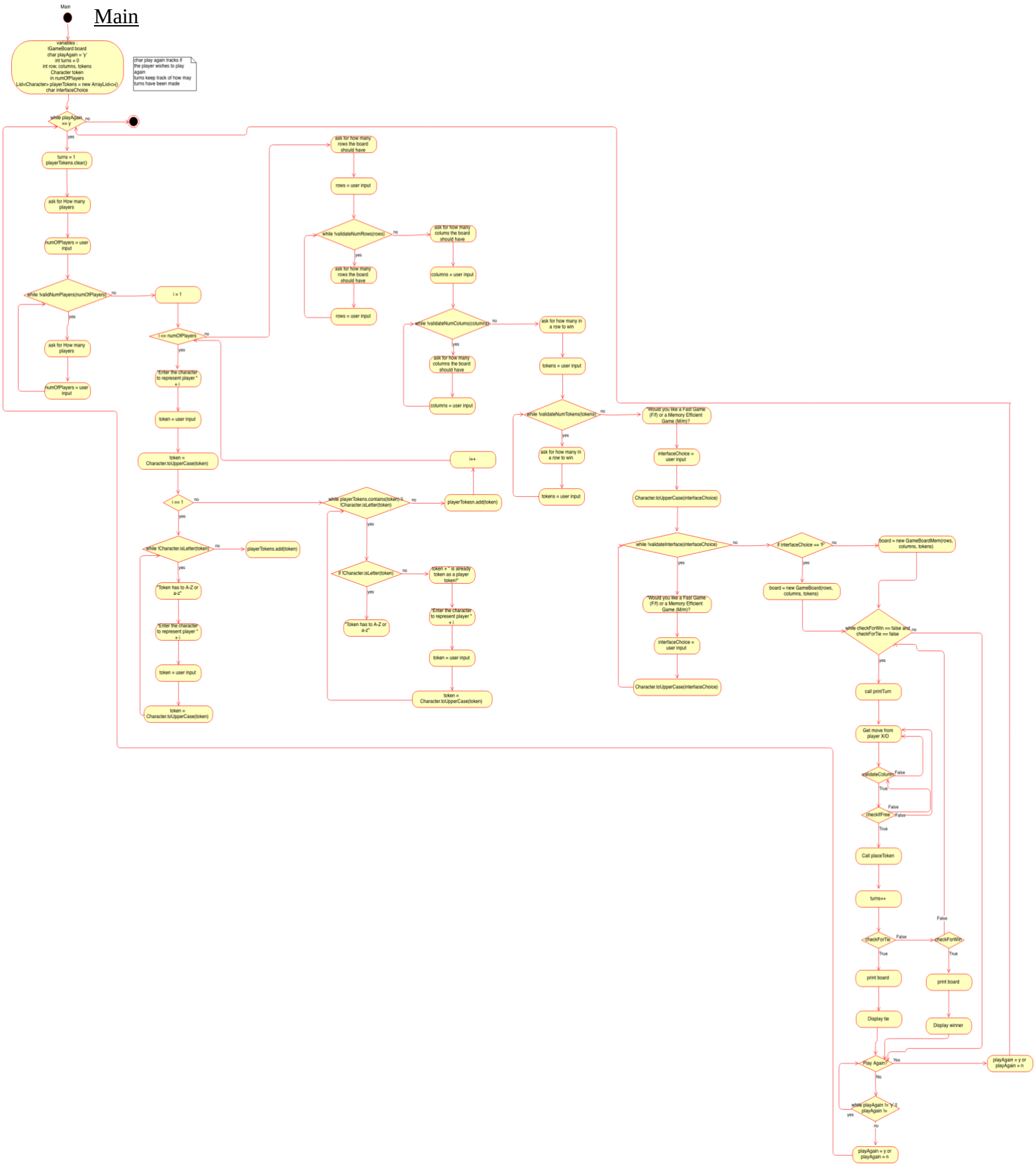
    - Expected output: I expected the board to look like the one above.
    - Reason: I chose this test as a way of testing that placeToken worked when multiple calls were made across the entirety of the board. I would consider this a more challenging case.

Diagrams

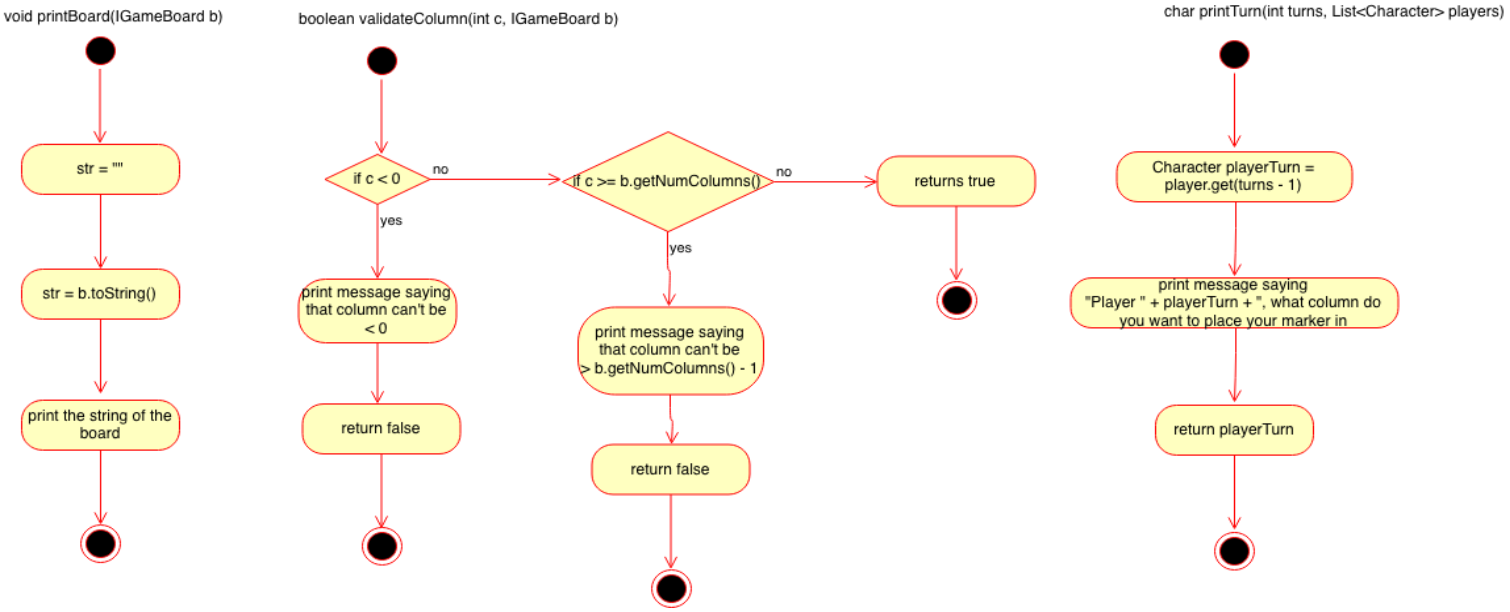
Class Diagrams

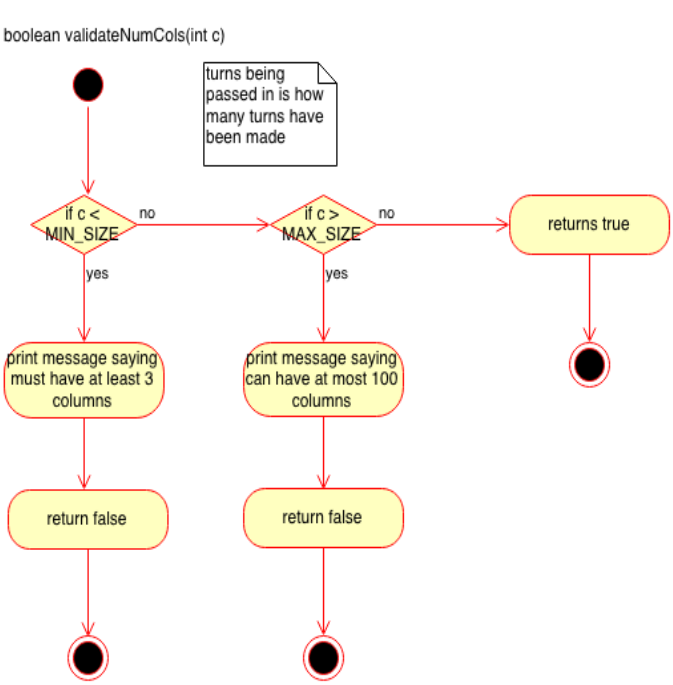


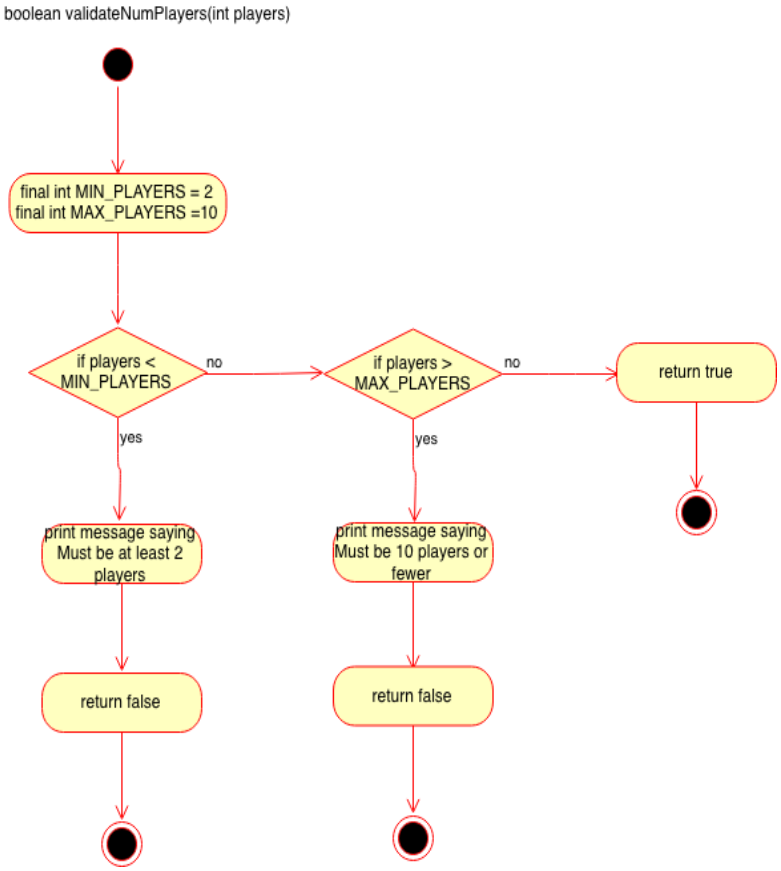
Main


Main
variables :
IGameBoard board
char playAgain = 'y'
int turns = 0
int row, columns, tokens
Character token
in numOfPlayers
List<Character> playerTokens = new ArrayList<>()
char interfaceChoice
char play again tracks if the player wishes to play again
turns keep track of how may turns have been made
while playAgain == y
no
yes
turns = 1
playerTokens.clear()
ask for How many players
numOfPlayers = user input
while !validNumPlayers(numOfPlayers)
ask for How many players
numOfPlayers = user input
i = 1
i <= numOfPlayers
"Enter the character to represent player " + i
token = user input
token = Character.toUpperCase(token)
i == 1
while !Character.isLetter(token)
"Token has to A-Z or a-z"
"Enter the character to represent player " + i
token = user input
token = Character.toUpperCase(token)
playerTokens.add(token)
ask for how many rows the board should have
rows = user input
while !validateNumRows(rows)
ask for how many rows the board should have
rows = user input
ask for how many colums the board should have
columns = user input
while !validateNumColums(columns)
ask for how many columns the board should have
columns = user input
while playerTokens.contains(token)||!Character.isLetter(token)
playerTokesn.add(token)
i++
if !Character.isLetter(token)
"Token has to A-Z or a-z"
token + " is already token as a player token!"
"Enter the character to represent player " + i
token = user input
token = Character.toUpperCase(token)
ask for how many in a row to win
tokens = user input
while !validateNumTokens(tokens)
ask for how many in a row to win
tokens = user input
"Would you like a Fast Game (F/f) or a Memory Efficient Game (M/m)?
interfaceChoice = user input
Character.toUpperCase(interfaceChoice)
while !validateInterface(interfaceChoice)
"Would you like a Fast Game (F/f) or a Memory Efficient Game (M/m)?
interfaceChoice = user input
Character.toUpperCase(interfaceChoice)
if interfaceChoice == 'F'
board = new GameBoard(rows, columns, tokens)
board = new GameBoardMem(rows, columns, tokens)
while checkForWin == false and checkForTie == false
call printTurn
Get move from player X/O
validateColumn
checkIfFree
Call placeToken
turns++
checkForTie
checkForWin
print board
Display tie
print board
Display winner
Play Again?
playAgain = y or playAgain = n
while playAgain != 'y'||playAgain != 'n'
playAgain = y or playAgain = n

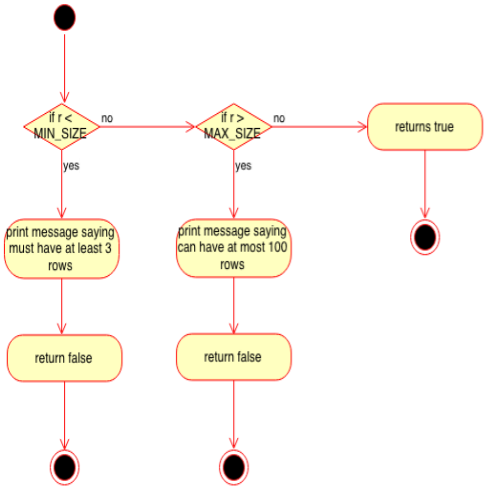
Private Helper Functions in Main



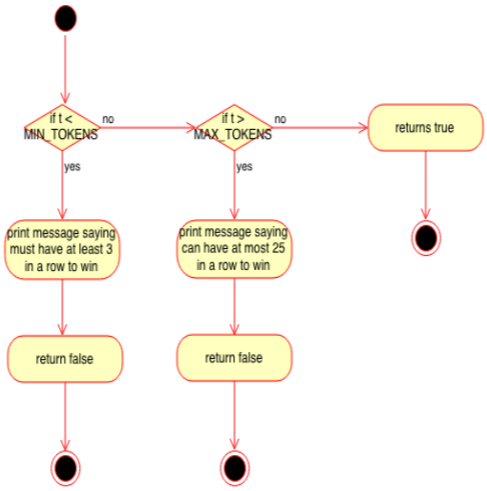




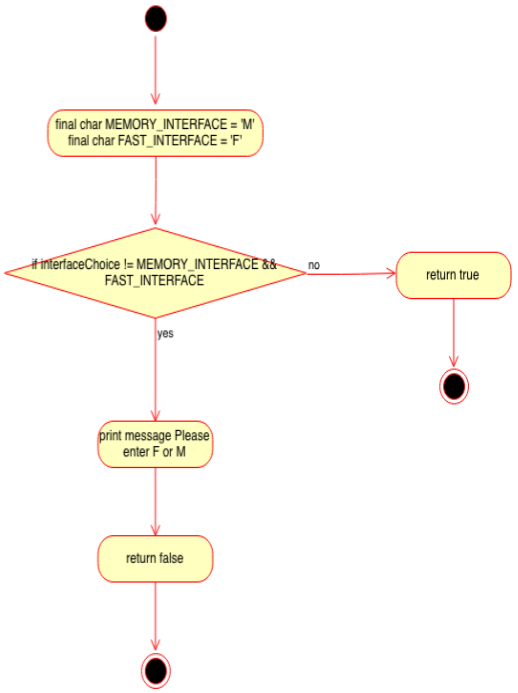
boolean validateNumRows(int r)



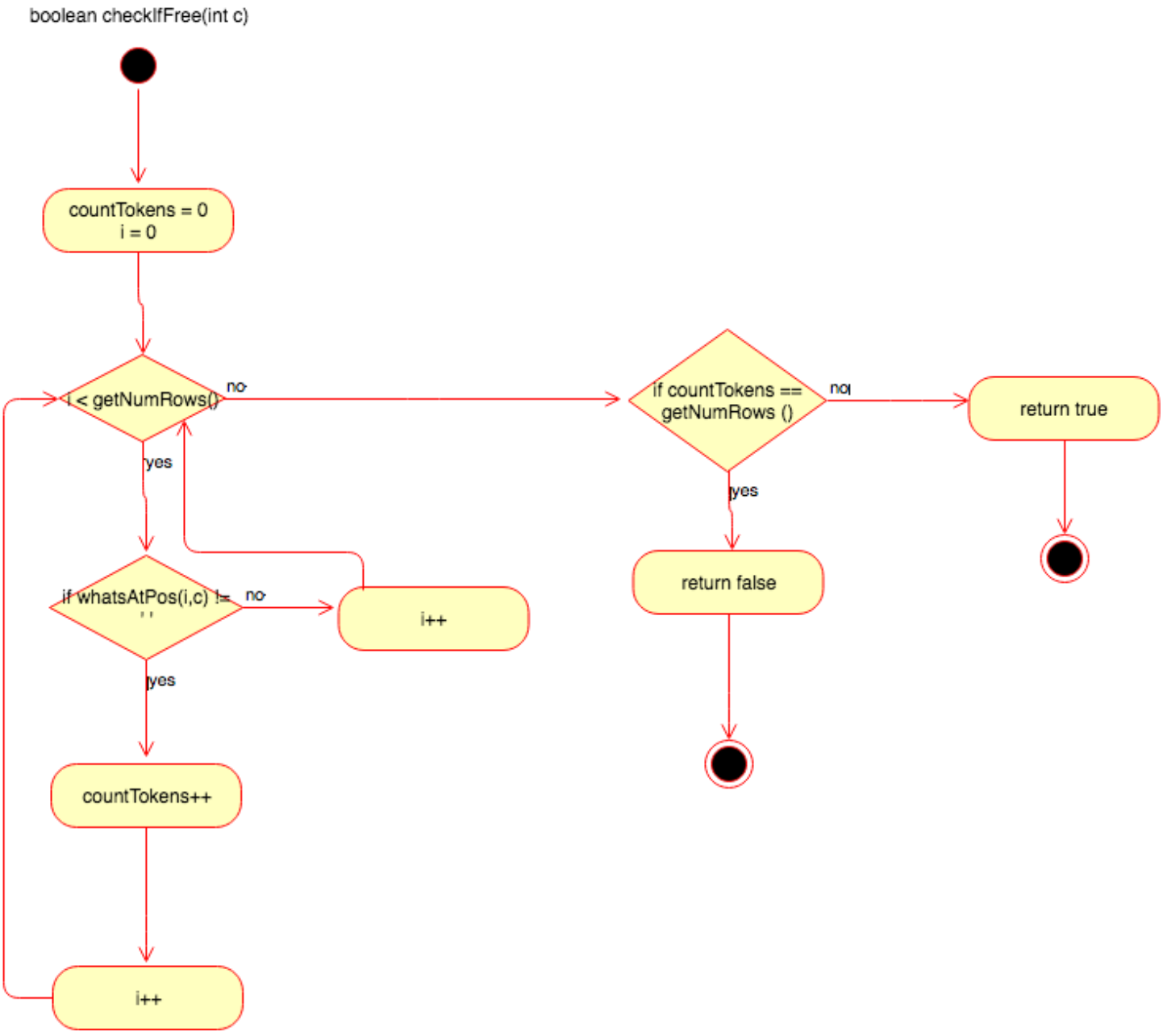
boolean validateNumTokens(int t)



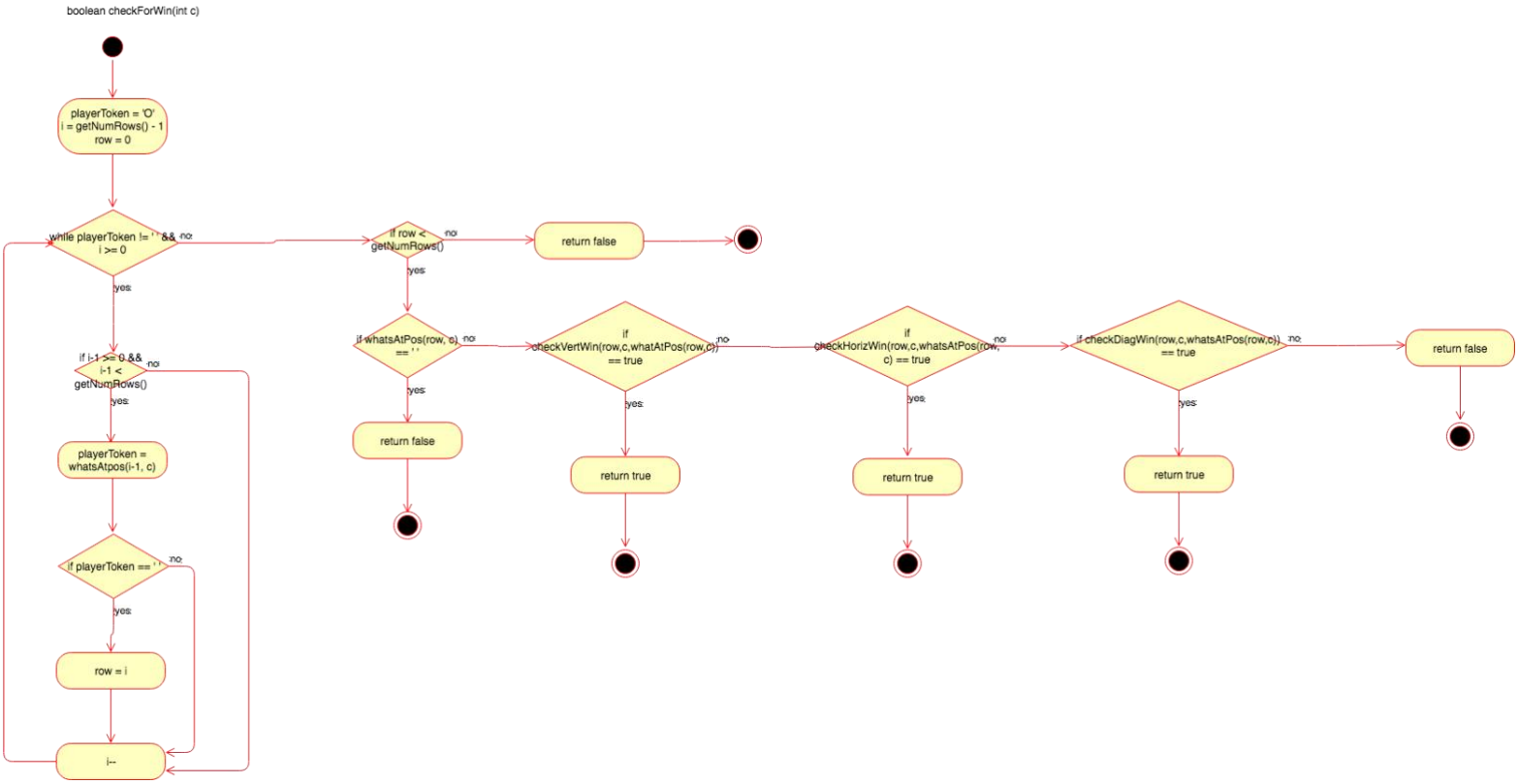
boolean validateInterface(char interfaceChoice)



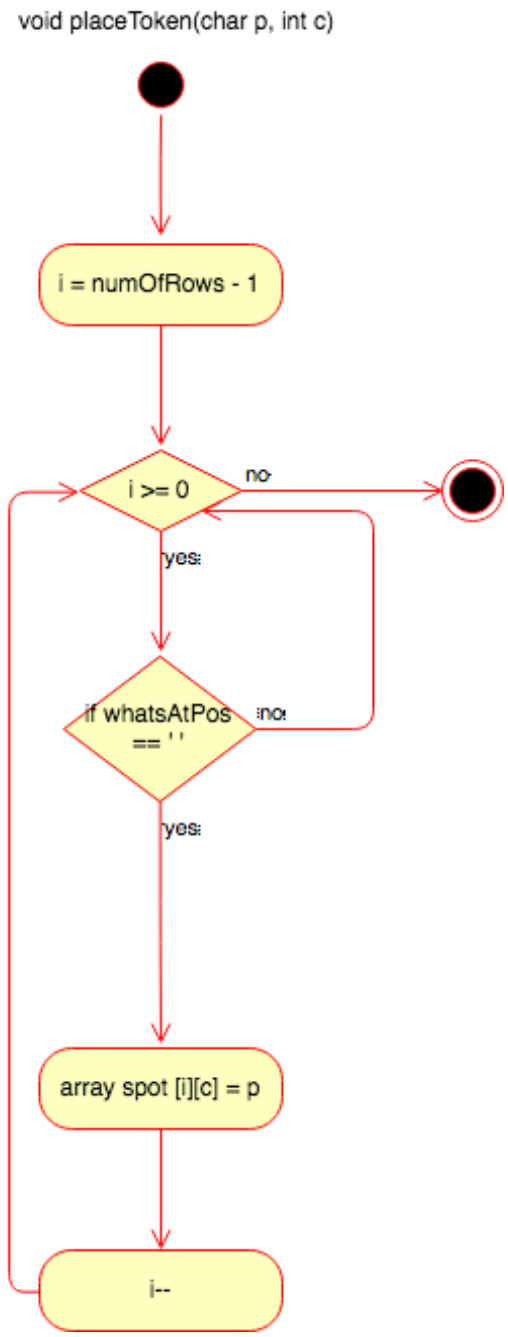
checkIfFree (default method)



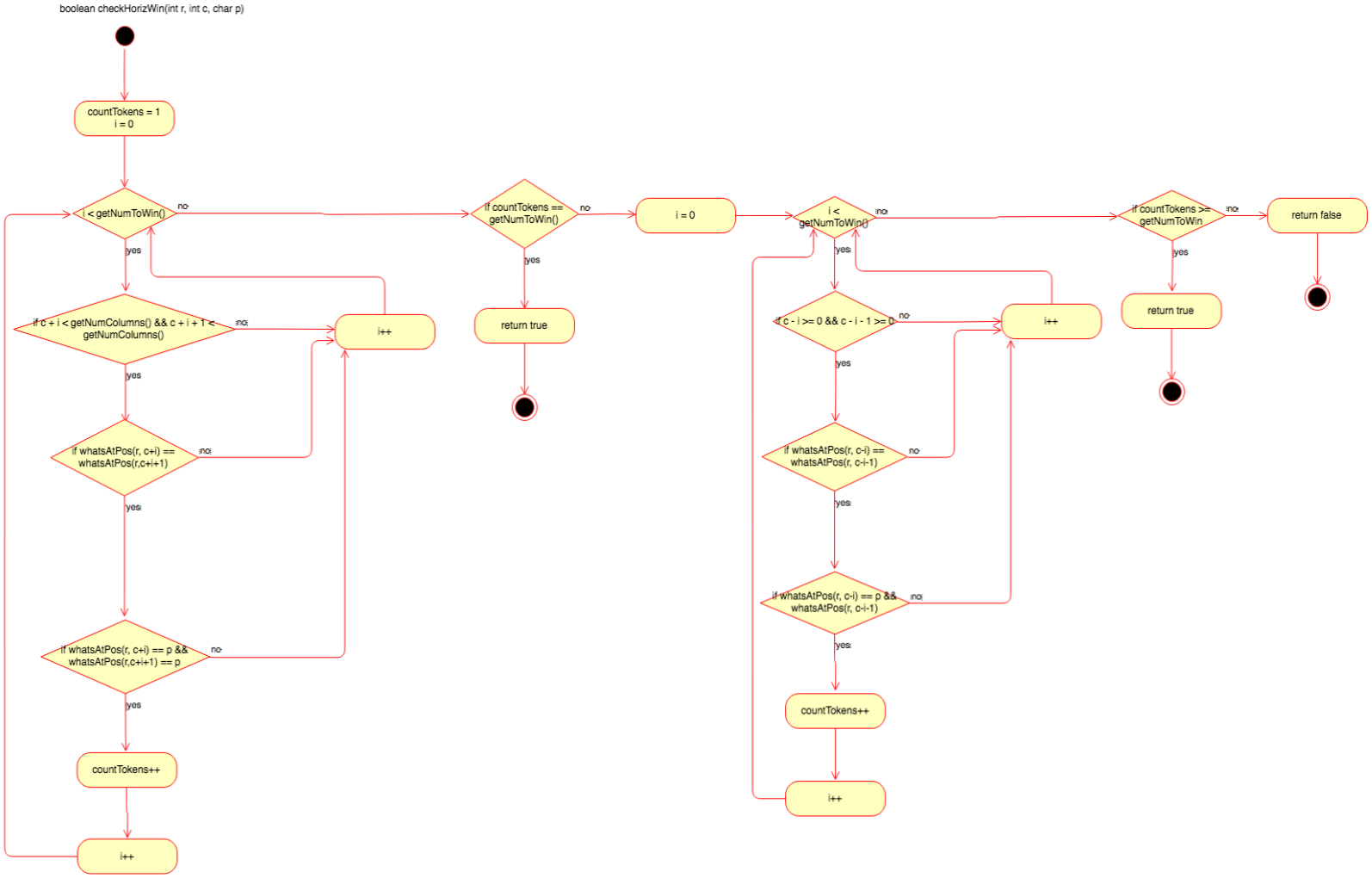
checkForWin (default method)



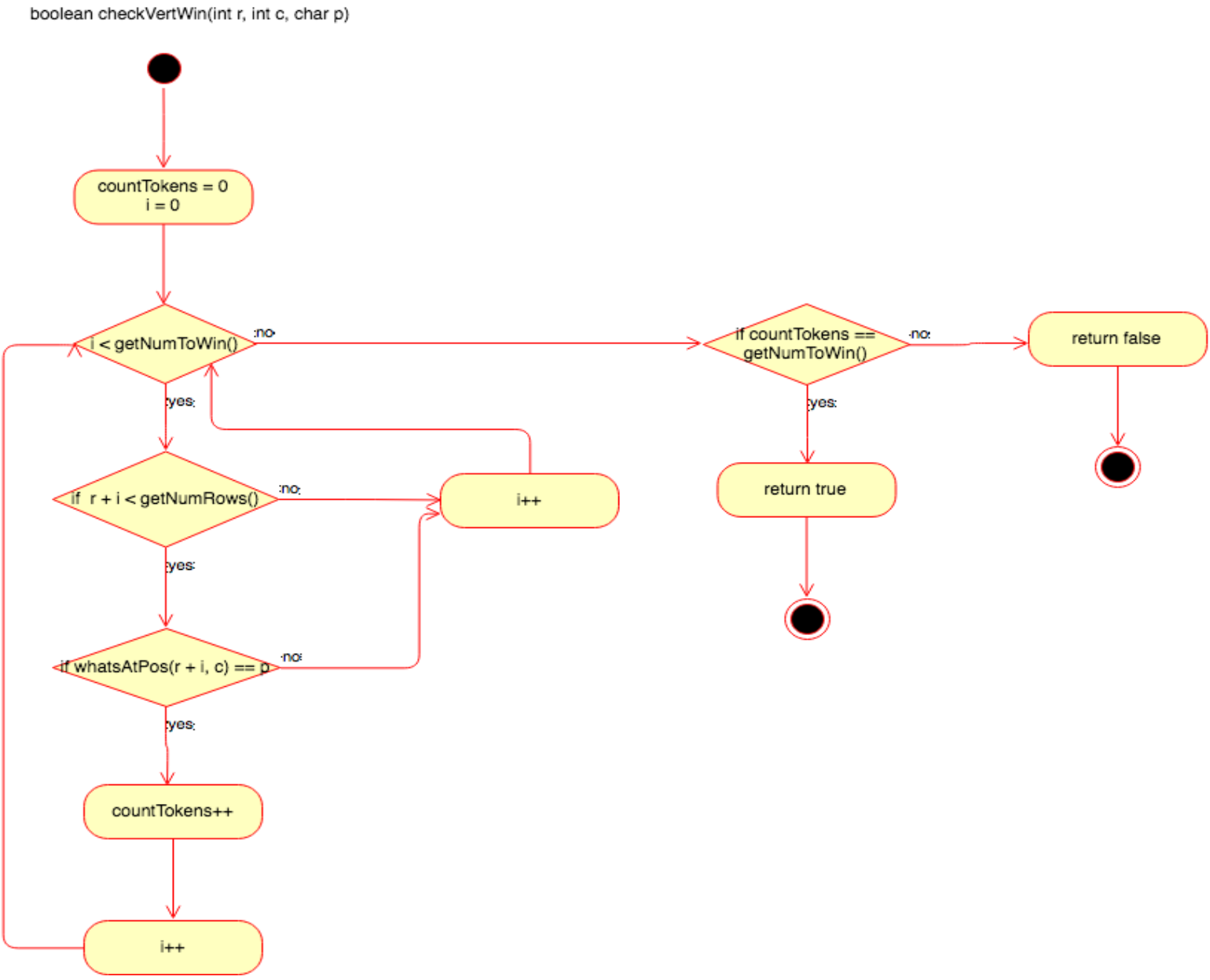
placeToken (in orginal GameBoard class)



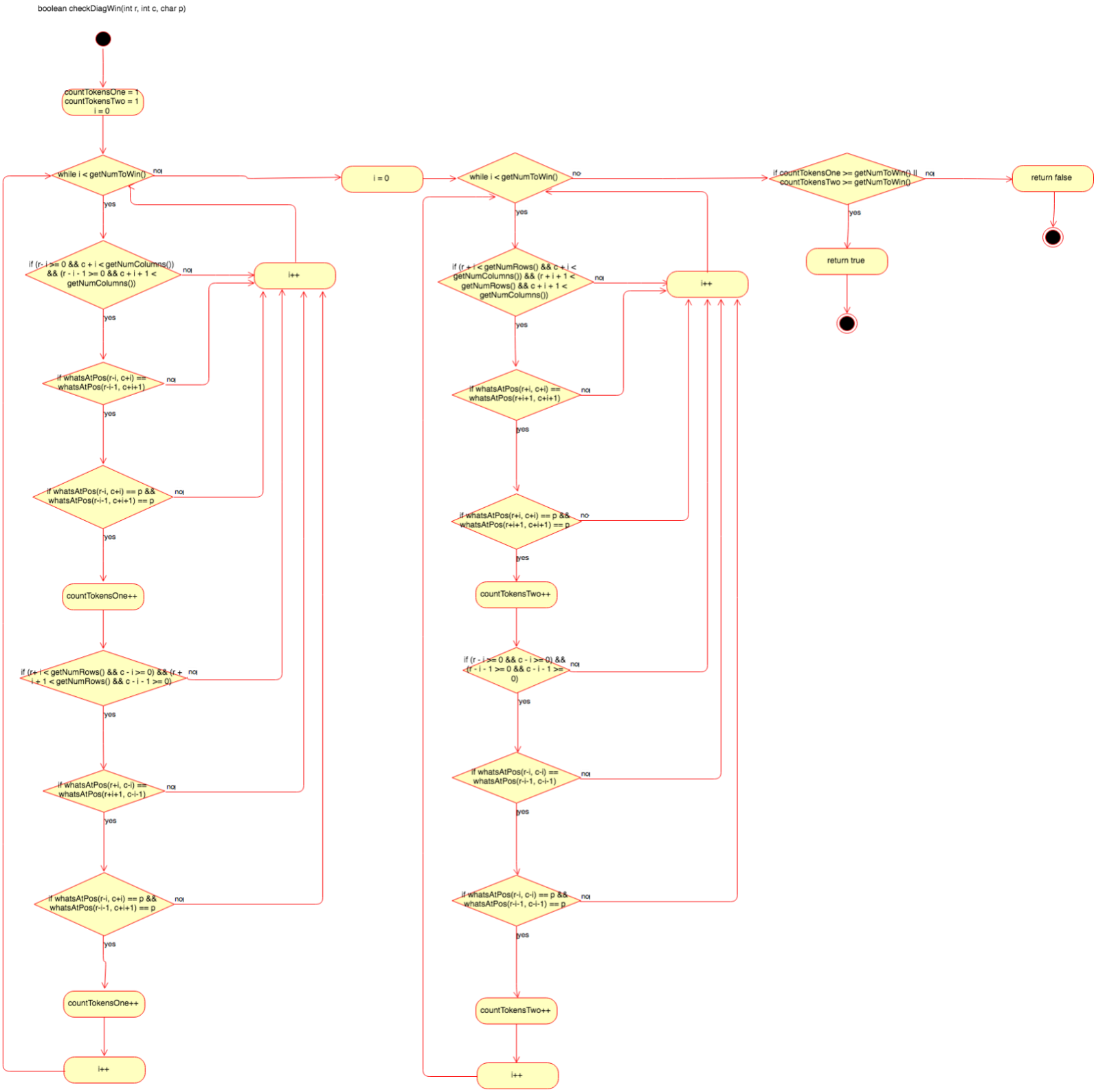
checkHorizWin (default method)



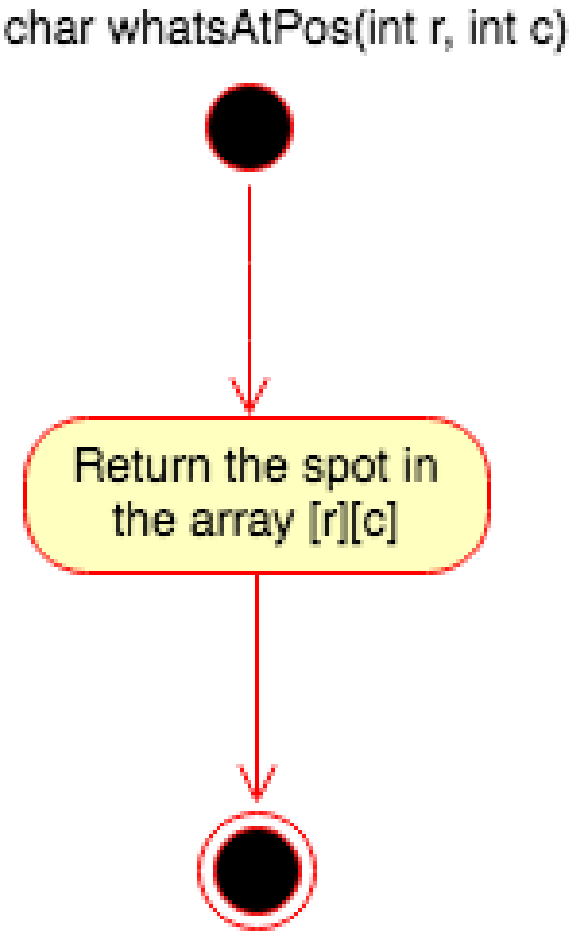
checkVertWin (default method)



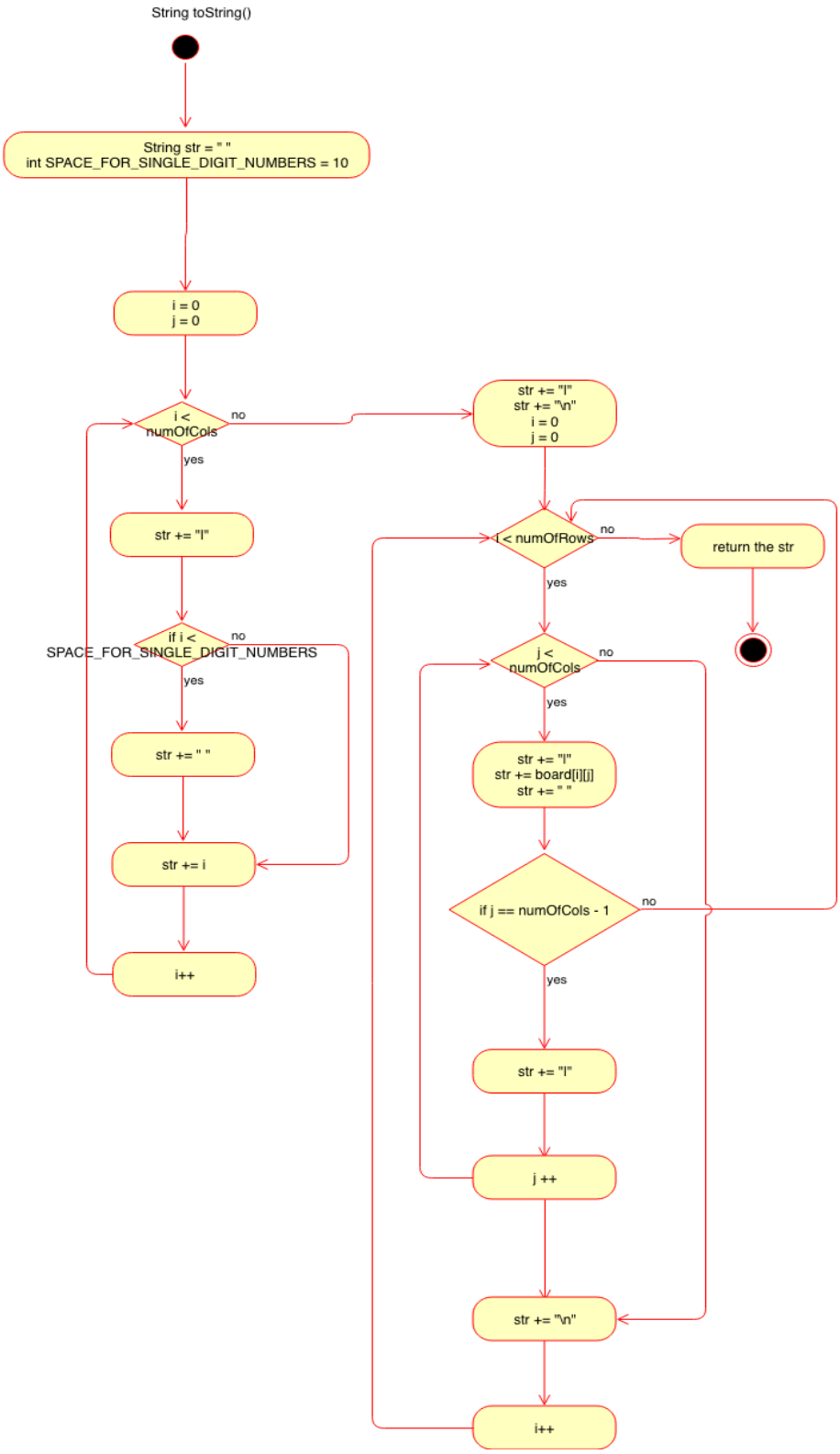
checkDiagWin (default method)



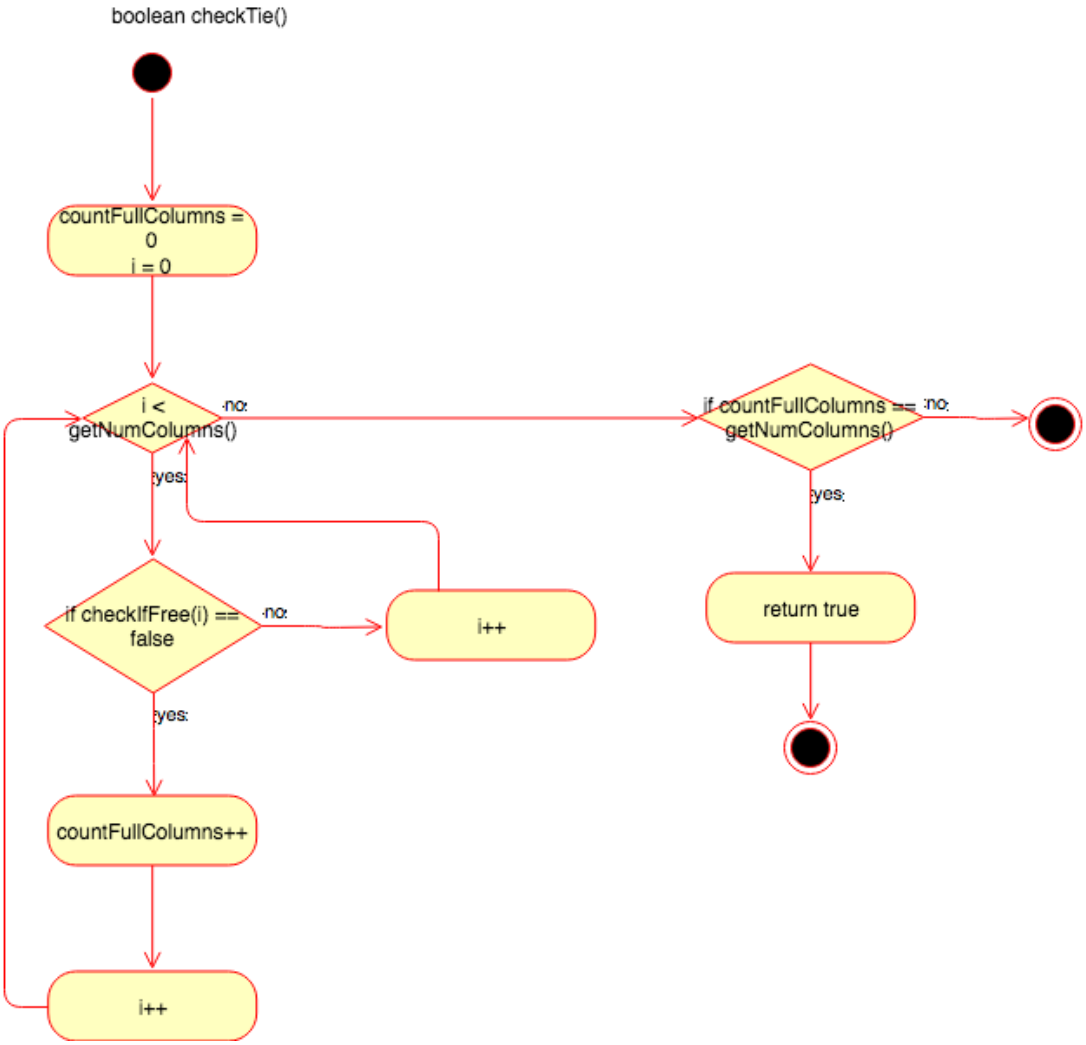
whatsAtPos (in orginal GameBoard class)



toString (in original GameBoard class)

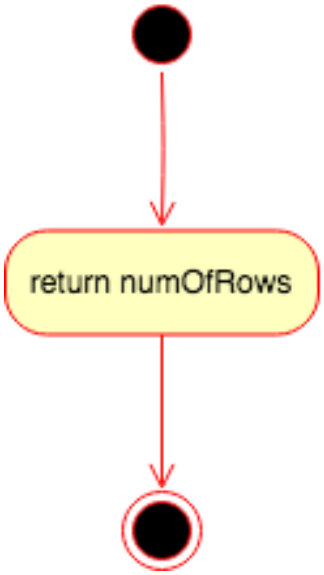


checkTie (default method)

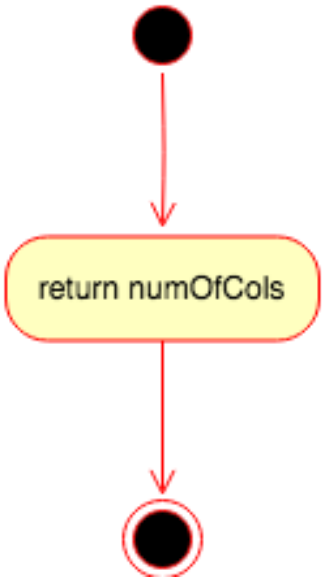


Getters (in original GameBoard class)

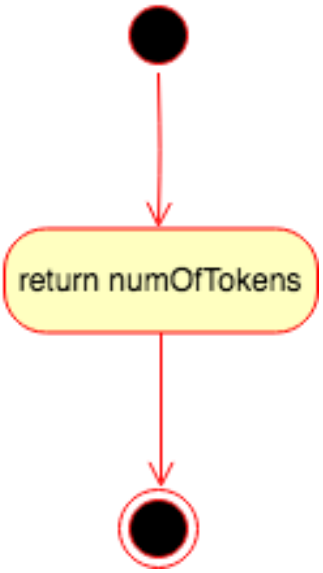
int getNumRows()



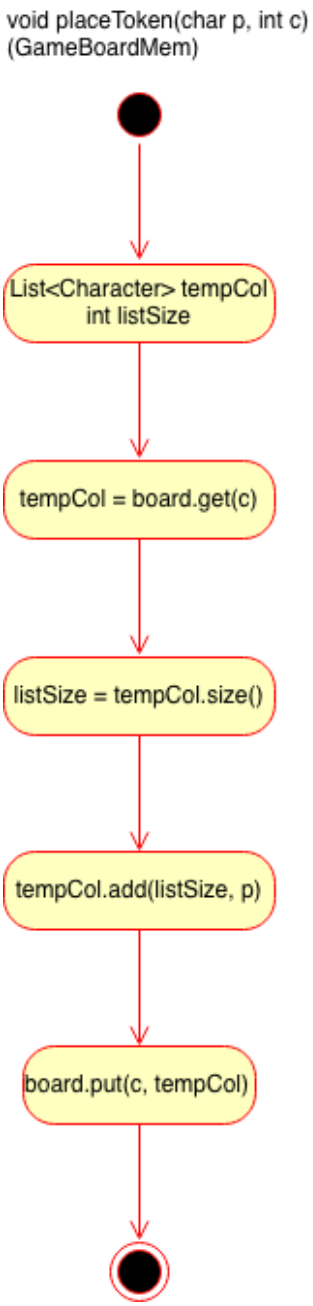
int getNumColumns()



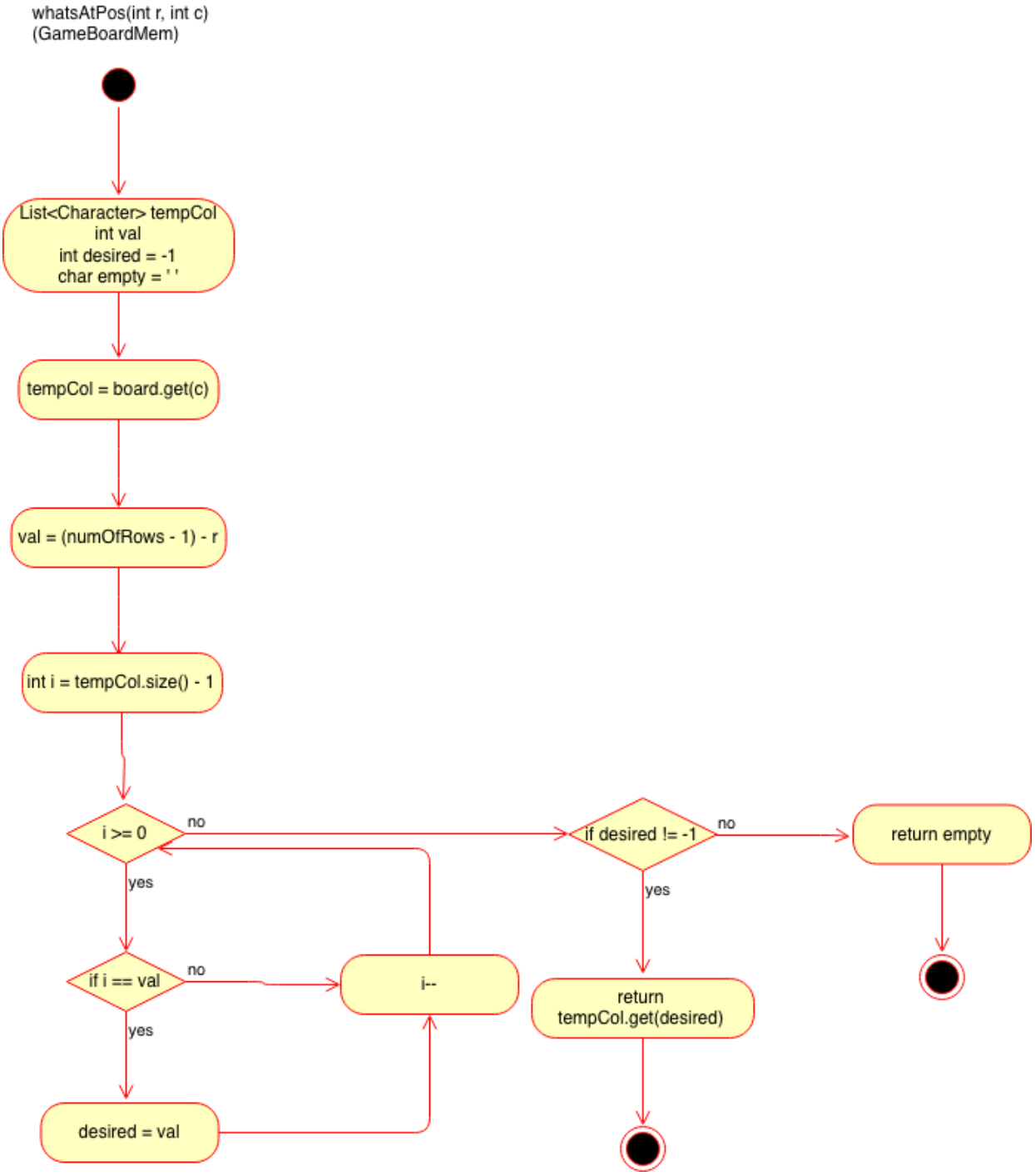
int getNumToWin()



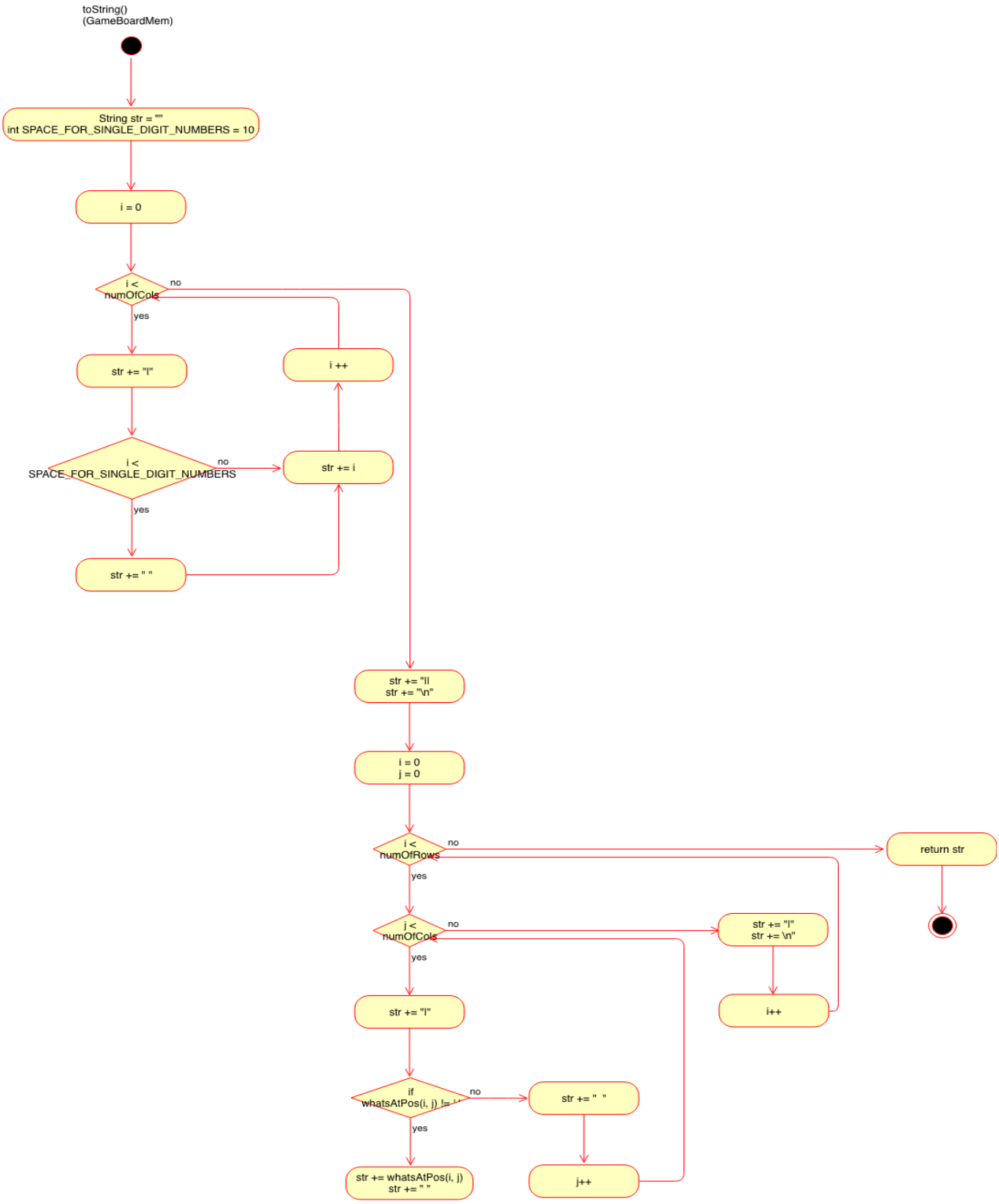
placeToken (GameBoardMem)



whatsAtPos (GameBoardMem)



toString (GameBoardMem)



Deployment

In order to run my program, you first have to unzip the PaigeClemonsHW4 folder. Then navigate to that folder in your terminal. Once you in the PaigeClemonsHW4 folder, there should only be a cpsc2150 folder, the makefile, and my report in the folder. After you check that stay in that folder, you can compile my program by typing "make" into the terminal. Then to run the program all you have to do is type "make run" then you can play the Connect Four game as usual. If you wish to compile the test cases, then type "make test" into the terminal. If you wish to run the test cases, then type "make runtest" into the termial. If you wish to delete the .class files, then all you have to do is type "make clean" into the terminal.