CPSC 2150 Project Report

Wilson Murphy

Requirements Analysis

Functional Requirements:

1. As a player, I can choose the number of columns to play with, so that I can customize the game
2. As a player, I can choose the number of rows to play with, so that I can customize the game
3. As a player, I can choose the number of tokens in a row to win, so that the game can be won accordingly
4. As a player, I can select a column to place my token in, so that I can play the game.
5. As a player, I can see the current game board, so that I know the status of the game.
6. As a player, I can be informed who won, so that I know the outcome of the game.
7. As a player, I can be informed if the game is a tie, so that I know the outcome of the game.
8. As a player, I can select a column to drop my token in so that I can make a move.
9. As a player, I can be asked if I want to play again after a game ends so that I can choose to continue playing.
10. As a player, I can be informed if I select an invalid column (e.g. full column) so that I can make a valid move.

Non-Functional Requirements

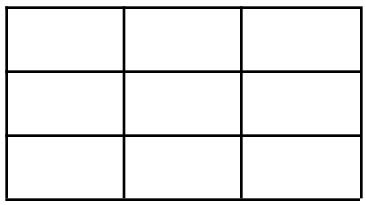
1. Enumerated list of Non-functional requirements
2. The game must be programmed using java
3. the game board must be atleast 3x3
4. the game board can be at most 20 x 20
5. the size of the board is decided by the player
6. the number of tokens in a row to win must be decided by the player

7. the number of players must be decided by the player
8. the minimum number of players is 2
9. the maximum number of players is 10
10. The game must alternate turns between players
11. the game must be able to be won via a vertical, horizontal, or diagonal line of the same token, using the given amount needed to win
12. The game must have a gui
13. The game must be designed using UML diagrams for classes and methods and writing contracts for each method and class.
14. The game must strictly follow the specified class and method names and function signatures as outlined in the project instructions.
15. The program must check for a win or tie after each move.
16. The program must handle errors if a player selects an invalid column.

Test:

Constructor

1. Gameboard(int r, int c, int w)

Input:	Output:	Reason:
r = 3 c = 3 w = 3	 board.getNumToWin() = 3	*test case that checks whether the GameBoardMem constructor can create a board with the smallest possible size (3 rows x 3 columns).* Function Name: testConstructor_minimum_size()

2. Gameboard(int r, int c, int w)

Input:	Output:	Reason:
r = 100 c = 100	board (100 x 100) board.getNumToWn() = 25	*test case that checks whether the GameBoardMem constructor can create a board with the largest*

w = 25		*possible size (100 rows x 100 columns).* Function Name: testConstructor_maximum_size()

3. Gameboard(int r, int c, int w)

Input: r = 30 c = 20 w = 3	Output Board (30 x 20) board.getNumToWin() = 3	Reason: *test case that checks whether the GameBoardMem constructor can create a board with unequal rows and columns (30 rows x 20 columns).* Function Name: testConstructor_mismatch_rows_and_columns()

checkIfFree

1. boolean checkIfFree(int c)

Input: [ ][ ][ ][ ][ ] [ ][ ][ ][ ][ ] [ ][ ][ ][ ][ ] [ ][ ][ ][ ][ ] [ ][ ][ ][ ][ ]	Output: checkIfFree(2) = true board remains the same	Reason: *test checks if the checkIfFree method correctly identifies a free column on an empty board. It initializes an empty board, places a token in a different column and checks that the method returns true for the empty column.* Function Name: testCheckIfFree_empty()

2. boolean checkIfFree(int c)

<table>
<tr>
<td>Input:
<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>X</td></tr></table></td>
<td>Output:

checkIfFree(4) = true

Board remains same</td>
<td>Reason:
test checks if the checkIfFree method correctly identifies a free column on a board with one token. It initializes an empty board, places a token in one column, checks that the method returns true for a different column and that the board has the expected token in the correct column.

Function Name:
testCheckIfFree_single_token()</td>
</tr>
</table>

3. boolean checkIfFree(int c)

<table>
<tr>
<td>Input:
<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(4) = false
Board remains same</td>
<td>Reason:
test checks if the checkIfFree method correctly identifies a full column on a board with tokens. It initializes a board with a full column of tokens and checks that the method returns false for that column.

Function Name:
testCheckIfFree_full_column()</td>
</tr>
</table>

placeToken

1. void placeToken(char p, int c)

<table>
<tr>
<td>Input:
<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
<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>X</td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
test case tests if the placeToken() method can add a token to the first available column on the board and returns the correct board configuration.

Function Name
testPlaceToken_bottom_left_position()
:</td>
</tr>
</table>

2. void placeToken(char p, int c)

<table>
<tr>
<td>Input:
<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
<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>X</td></tr></table></td>
<td>Reason:
test case tests if the placeToken() method can add a token to the last available column on the board and returns the correct board configuration.

Function Name:
test_PlaceToken_bottom_right_position()</td>
</tr>
</table>

3. void placeToken(char p, int c)

<table>
<tr>
<td>Input:
<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
<table><tr><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></tr><tr><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></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr></table></td>
<td>Reason:
test case tests if the placeToken() method can add many tokens to the board repeatedly until the board is completely filled, and returns the correct board configuration.

Function Name:
testPlaceToken_fill_entire_board()</td>
</tr>
</table>

4. void placeToken(char p, int c)

<table>
<tr>
<td>Input:
<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
<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>X</td><td>O</td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
makes sure that the function can add different tokens (represented by letters like 'X' and 'O') to the board.

Function Name:
testPlaceToken_different_characters()</td>
</tr>
</table>

5. void placeToken(char p, int c)

Input:	Output	Reason: *makes sure that the function*

<table>
<tr>
<td>
<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td>
<table>
<tr><td>w</td><td>w</td><td>w</td><td>w</td></tr>
<tr><td>i</td><td>i</td><td>i</td><td>i</td></tr>
<tr><td>l</td><td>l</td><td>l</td><td>l</td></tr>
<tr><td>s</td><td>s</td><td>s</td><td>s</td></tr>
<tr><td>o</td><td>o</td><td>o</td><td>o</td></tr>
<tr><td>n</td><td>n</td><td>n</td><td>n</td></tr>
</table>
</td>
<td>
can add a pattern of tokens to the board, where each row has a different letter.

Function Name:
testPlaceToken_fill_different_characters()
</td>
</tr>
</table>

checkVertWin

1. boolean checkVertWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>board.getNumToWin() = 3
Pos = (2, 2)
P = X</td>
<td>Output

checkVertWin(pos p) = false

board remains same</td>
<td>Reason: test case checks if the checkVertWin method returns false when there is no vertical win on the game board. It creates a 5x5 game board with no tokens placed on it and then calls the checkVertWin method with a position in the middle of the board and a token 'X'. The expected outcome is that the method returns false and the game board remains unchanged.

Function Name: testCheckVerticalWin_no_win ()</td>
</tr>
</table>

2.boolean checkVertWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>X</td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td></tr></table>board.getNumToWin()= 3
Pos = (2, 2)</td>
<td>Output

checkVertWin(pos, p) = true

board remains same</td>
<td>Reason:
test case checks if the checkVertWin method correctly detects a vertical win on the game board. It creates a 5x5 game board with three 'X' tokens in a column and calls the checkVertWin method with the position of the middle token and a token 'X'. The expected outcome is that the method returns true and the game board remains unchanged.</td>
</tr>
</table>

p = X		Function Name: testCheckVerticalWin_win()

3.boolean checkVertWin(BoardPosition pos, char p)

Input: <table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td></tr></table> board.getNumToWin() = 3 pos = (2, 2) p = X	Output checkVertWin(pos, p) = true board remains same	Reason: *test case checks if the checkVertWin method correctly detects a vertical win on the game board when there are more tokens in the column than needed for a win. It creates a 5x5 game board with four 'X' tokens in a column and calls the checkVertWin method with the position of the bottom token and a token 'X'. The expected outcome is that the method returns true and the game board remains unchanged.* Function Name: testCheckVerticalWin_more_than_needed()

4.boolean checkVertWin(BoardPosition pos, char p)

Input: <table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td>X<td></td><td></td></tr><tr><td></td><td></td>O<td></td><td></td></tr><tr><td></td><td></td>X<td></td><td></td></tr></table>	Output checkVertWin(pos, p) = false board remains same	Reason: *test case checks if the checkVertWin method correctly detects that there is no vertical win on the game board when the tokens in the column are not all the same. It creates a 5x5 game board with three 'X' tokens and one 'O' token in a column and calls the checkVertWin*

<table><tr><td></td><td></td><td>X</td><td></td><td></td></tr></table> board.getNumToWin() = 3 Pos = (2, 2) P = X		*method with the position of the bottom token and a token 'X'. The expected outcome is that the method returns false and the game board remains unchanged.* Function Name: testCheckVerticalWin_incomplete()

checkHorizWin

1. boolean checkHorizWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>pos = (2, 2)
p = X
board.checkNumToWin() = 3</td>
<td>Output

checkHorizWin(pos, p) = false

board remains same</td>
<td>Reason:
test checks if the checkHorizWin method works correctly when there is no horizontal win on an empty game board. It creates an empty 5x5 game board, places no tokens, and asserts that checkHorizWin returns false and the board is still empty.

Function Name:
testCheckHorizontalWin_empty()</td>
</tr>
</table>

2. boolean checkHorizWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>X</td><td>X</td><td>X</td><td>X</td><td></td></tr></table>board.getNumToWin() = 3
pos = (2, 2)</td>
<td>Output
checkHorizWin(pos, p) = true

board remains same</td>
<td>Reason:
test checks if the checkHorizWin method correctly detects a horizontal win on a game board with four X tokens in a row on the top row, which is one more than the three required for a win. It creates a 5x5 game board, places four X tokens on the top row, and asserts that checkHorizWin returns true and the game board matches the expected state</td>
</tr>
</table>

P = X		*with four X tokens in a row on the top row.* Function Name: testCheckHorizontalWin_more_than_needed()

3. boolean checkHorizWin(BoardPosition pos, char p)

Input: board.getNumToWin() = 3 pos = (2, 2) P = X	Output checkHorizWin(pos, p) = true board remains same	Reason: *test checks if the checkHorizWin method correctly detects a horizontal win on a game board with three X tokens in a row on the top row. It creates a 5x5 game board, places three X tokens on the top row, and asserts that checkHorizWin returns true and the game board matches the expected state with three X tokens in a row on the top row.* Function Name: testCheckHorizontalWin_normal_test()

Input:

X	X	X		

4. boolean checkHorizWin(BoardPosition pos, char p)

Input: P = X	Output checkHorizWin(pos, p) = false board remains same	Reason: *test checks if the checkHorizWin method correctly detects an incomplete horizontal win on a game board with three X tokens in a row on the top row, but one token in the middle is an O instead of an X. It creates a 5x5 game board, places three X tokens and one O token on the top*

Input:

X	X	O	X	

board.getNumToWin() = 3 pos = (2, 2) P = X		*row, and asserts that checkHorizWin returns false and the game board matches the expected state with three X tokens in a row on the top row, but the middle token is an O.* Function Name: testCheckHorizontalWin_incomplete()

checkDiagWin

1. checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>board.getNumToWin() = 3
pos = (0, 0)
P = X</td>
<td>Output

checkDiagWin(pos, p) = false

board remains same</td>
<td>Reason:test case creates an empty board and checks whether checkDiagWin correctly identifies that there is no diagonal win for player 'X' starting from the top-left corner.

Function Name:
testCheckDiagonalWin_empty()</td>
</tr>
</table>

2. checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>X</td><td></td><td></td></tr><tr><td>O</td><td>X</td><td>X</td><td></td><td></td></tr><tr><td>X</td><td>O</td><td>O</td><td></td><td></td></tr></table>board.getNumToWin() = 3
pos = (0, 0)
P = X</td>
<td>Output
checkDiagWin(pos, p) = true

board remains same</td>
<td>Reason: test case sets up a board where player 'X' has made a diagonal sequence of three tokens starting from the top-left corner, and then checks whether checkDiagWin correctly identifies this as a win for 'X'.

Function Name:
testCheckDiagonalWin_start_left_barely_win()</td>
</tr>
</table>

3. checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>X</td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td>X</td><td>O</td><td></td></tr>
<tr><td>X</td><td>O</td><td>O</td><td></td><td></td></tr>
</table>
board.getNumToWin() = 3
pos = (2, 2)
P = X</td>
<td>Output
checkDiagWin(pos, p) = true

board remains same</td>
<td>Reason:
test case sets up a board where player 'X' has made a diagonal sequence of four tokens starting from the top-left corner, and then checks whether checkDiagWin correctly identifies this as a win for 'X'. This is distinct from the previous test case because it checks that the method correctly handles a longer diagonal sequence.

Function Name:
testCheckDiagonalWin_start_left_more_than_needed()</td>
</tr>
</table>

4. checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td></td><td></td><td>O</td><td>X</td><td></td></tr>
<tr><td>O</td><td>X</td><td>X</td><td>O</td><td></td></tr>
<tr><td>X</td><td>O</td><td>O</td><td>X</td><td></td></tr>
</table>
board.getNumToWin() = 3
pos = (2, 2)
P = X</td>
<td>Output
checkDiagWin(pos, p) = false

board remains same</td>
<td>Reason:
test case sets up a board where player 'X' has made a diagonal sequence of two tokens starting from the top-left corner, but then 'O' has interrupted the sequence by playing a token. The test then checks whether checkDiagWin correctly identifies that there is no diagonal win for 'X' starting from the given position.

Function Name:
testCheckDiagonalWin_start_left_not_enough() {</td>
</tr>
</table>

5. checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>X</td><td></td><td></td><td></td></tr>
<tr><td></td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td></td><td>O</td><td>O</td><td>X</td><td></td></tr>
</table>
board.getNumToWin() = 3
pos = (2, 2)
P = X</td>
<td>Output
checkDiagWin(pos, p) = true

board remains same</td>
<td>Reason:
test case sets up a board where player 'X' has made a diagonal sequence of three tokens starting from the top-right corner, and then checks whether checkDiagWin correctly identifies this as a win for 'X'. This is distinct from the second test case because it checks that the method correctly handles diagonal sequences that start from the top-right corner instead of the top-left corner.

Function Name:
testCheckDiagonalWin_start_right_barely_win()</td>
</tr>
</table>

6. checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>X</td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td>O</td><td>X</td><td></td></tr>
</table>
board.getNumToWin() = 3
pos = (2, 2)
P = X</td>
<td>Output
checkDiagWin(pos, p) = true

board remains same</td>
<td>Reason:
sets up a board where there is a diagonal win that starts to the right when there are more than enough tokens in a row, and then calls checkDiagWin() with the appropriate arguments to check if it returns true. Finally, it asserts that the board is in the expected state after the moves have been made.

Function Name:
testCheckDiagonalWin_start_right_extra_win()</td>
</tr>
</table>

7. checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td>O</td><td>X</td><td></td></tr>
</table>
board.getNumToWin() = 3
pos = (2, 2)
P = X</td>
<td>Output
checkDiagWin(pos, p) = true

board remains same</td>
<td>Reason:
sets up a board where the function checks to make sure a diagonal starting from the right has equivalent tokens in a row but there are not enough tokens in a row, and then calls checkDiagWin() with the appropriate arguments to check if it returns false. Finally, it asserts that the board is in the expected state after the moves have been made.

Function Name:
testCheckDiagonalWin_start_right_just_not_enough()</td>
</tr>
</table>

checkTie

1. boolean checkTie()

<table>
<tr>
<td>Input:
<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

CheckTie() = false

board remains same</td>
<td>Reason:
test case represents the standard case of there being no tie

Function Name:
testCheckTie_empty()</td>
</tr>
</table>

2.boolean checkTie()

<table>
<tr>
<td>Input:
<table><tr><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></tr><tr><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></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr></table></td>
<td>Output
CheckTie() = true

board remains same</td>
<td>Reason:
test case is for the checkTie() method of the game board class. It creates a 5x5 game board with all the cells filled with 'X'. Then it places 'X' tokens in all rows. This creates a tie game as there are no more empty cells to place tokens in. The test checks if the method checkTie() returns true for the tie game and if the actual game board matches the expected game board.

Function Name:
testCheckTie_full()</td>
</tr>
</table>

3.boolean checkTie()

<table>
<tr>
<td>Input:
<table><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr></table></td>
<td>Output
CheckTie() = false

board remains same</td>
<td>Reason:
test case is for the checkTie() method of the game board class. It creates a 5x5 game board with all the cells filled with 'X', except for the cells in the last column. Then it places 'X' tokens in all but the last row. The game board is almost full, with only one cell empty. The test checks if the method checkTie() returns false for the almost full game board and if the actual game board matches the expected game board.

Function Name:
testCheckTie_almost_full()</td>
</tr>
</table>

4.boolean checkTie()

<table>
<tr>
<td>Input:
<table><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><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></tr><tr><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></tr></table></td>
<td>Output
CheckTie() = false

board remains same</td>
<td>Reason:
test case is for the checkTie() method of the game board class. It creates a 5x5 game board with all the cells filled with 'X', except for one cell in the last row and last column. Then it places 'X' tokens in all but the last row. It places four tokens in the last row to fill all cells except for the empty cell. The test checks if the method checkTie() returns false for the almost full game board and if the actual game board matches the expected game board.

Function Name:
testCheckTie_all_pos_but_one()</td>
</tr>
</table>

whatsAtPos

1. char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:
<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>Pos = (0,0)</td>
<td>Output

whatsAtPos(pos) = ‘ ‘

board remains same</td>
<td>Reason:
creates an empty 5x5 game board. The test places no tokens on the game board and checks if the method whatsAtPos() returns an empty cell for the top-left position and if the actual game board matches the expected game board.

Function Name:
testWhatsAtPos_empty()</td>
</tr>
</table>

2. char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:
<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>X</td><td></td><td></td><td></td><td></td></tr></table>Pos = (0,0)</td>
<td>Output
whatsAtPos(pos) = ‘X‘

board remains same</td>
<td>Reason:
creates an empty 5x5 game board. The test places an 'X' token in the top-left position and checks if the method whatsAtPos() returns 'X' for the top-left position and if the actual game board matches the expected game board.

Function Name:
testWhatAtPos_player_x()</td>
</tr>
</table>

3.char whatsAtPos(BoardPosition pos)

Input:	Output whatsAtPos(pos) = ‘W‘	Reason: *creates an empty 5x5 game board. The test places a 'W'*

W				

Pos = (0,0)

board remains same

token in the top-left position and checks if the method whatsAtPos() returns 'W' for the top-left position and if the actual game board matches the expected game board.

Function Name:
testWhatsAtPos_player_not_x_or_o()

4. char whatsAtPos(BoardPosition pos)

Input:

X				

Pos = (0,1)

Output
whatsAtPos(pos) = ‘ ‘

board remains same

Reason:
creates an empty 5x5 game board. The test places an 'X' token in the top-left position and checks if the method whatsAtPos() returns an empty cell for the position (1, 0) and if the actual game board matches the expected game board.

Function Name:
testWhatsAtPos_correct_position()

5. char whatsAtPos(BoardPosition pos)

Input:

Output

whatsAtPos(pos) = ‘0‘

board remains same

Reason:
test case ensures that the whatsAtPos() method returns the correct character when there are two players on the board at different positions.

<table>
<tr>
<td>
<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td></td><td></td><td></td></tr>
</table>
Pos = (0,1)
</td>
<td></td>
<td>Function Name:
testWhatsAtPos_two_players
_correct_character()</td>
</tr>
</table>

isPlayerAtPos

1. boolean isPlayerAtPos(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>pos = (0, 0)
p = ‘X’</td>
<td>Output

isPlayerAtPos = false

board remains same</td>
<td>Reason:
test case represents the standard case of an empty position on the board and ensures that the isPlayerAtPos() method returns false for a position without a player.

Function Name:testIsPlayerAtPos_empty()</td>
</tr>
</table>

2. boolean isPlayerAtPos(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
<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>X</td><td></td><td></td><td></td><td></td></tr></table>pos = (0, 0)
p = ‘X’</td>
<td>Output

isPlayerAtPos = true

board remains same</td>
<td>Reason:
test case represents the standard case of a player X being present on the board and ensures that the isPlayerAtPos() method returns true for a position with player X.

Function Name:
testIsPlayerAtPos_player_x()</td>
</tr>
</table>

3. boolean isPlayerAtPos(BoardPosition pos, char p)

Input:	Output	Reason:
<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>W</td><td></td><td></td><td></td><td></td></tr></table> pos = (0, 0) p = ‘W’	isPlayerAtPos = true board remains same	*test case ensures that the isPlayerAtPos() method recognizes characters other than X and O, and returns true for the specified character.* Function Name: testIsPlayerAtPos_not_x_or_o()

4. boolean isPlayerAtPos(BoardPosition pos, char p)

Input:	Output	Reason:
<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>X</td><td></td><td></td><td></td><td></td></tr></table> pos = (0, 1) p = ‘X’	isPlayerAtPos = false board remains same	*test case ensures that the isPlayerAtPos() method checks the correct position on the board and returns false for a different position with the same player as the one placed.* Function Name: testIsPlayerAtPos_check_correct_position()

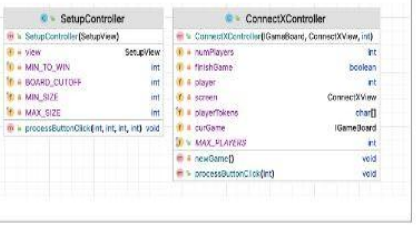
5. boolean isPlayerAtPos(BoardPosition pos, char p)

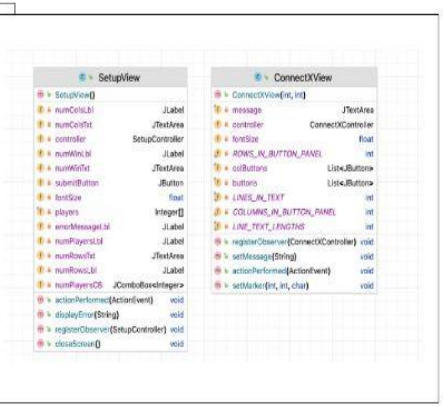
Input:	Output	Reason:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>	isPlayerAtPos(pos, p) = true board remains same	*test case ensures that the isPlayerAtPos() method looks for the correct character on the board and returns true for the specified character.* Function Name: testIsPlayerAtPos_correct_ch

X O pos = (0, 1) p = ‘O’		aracters()

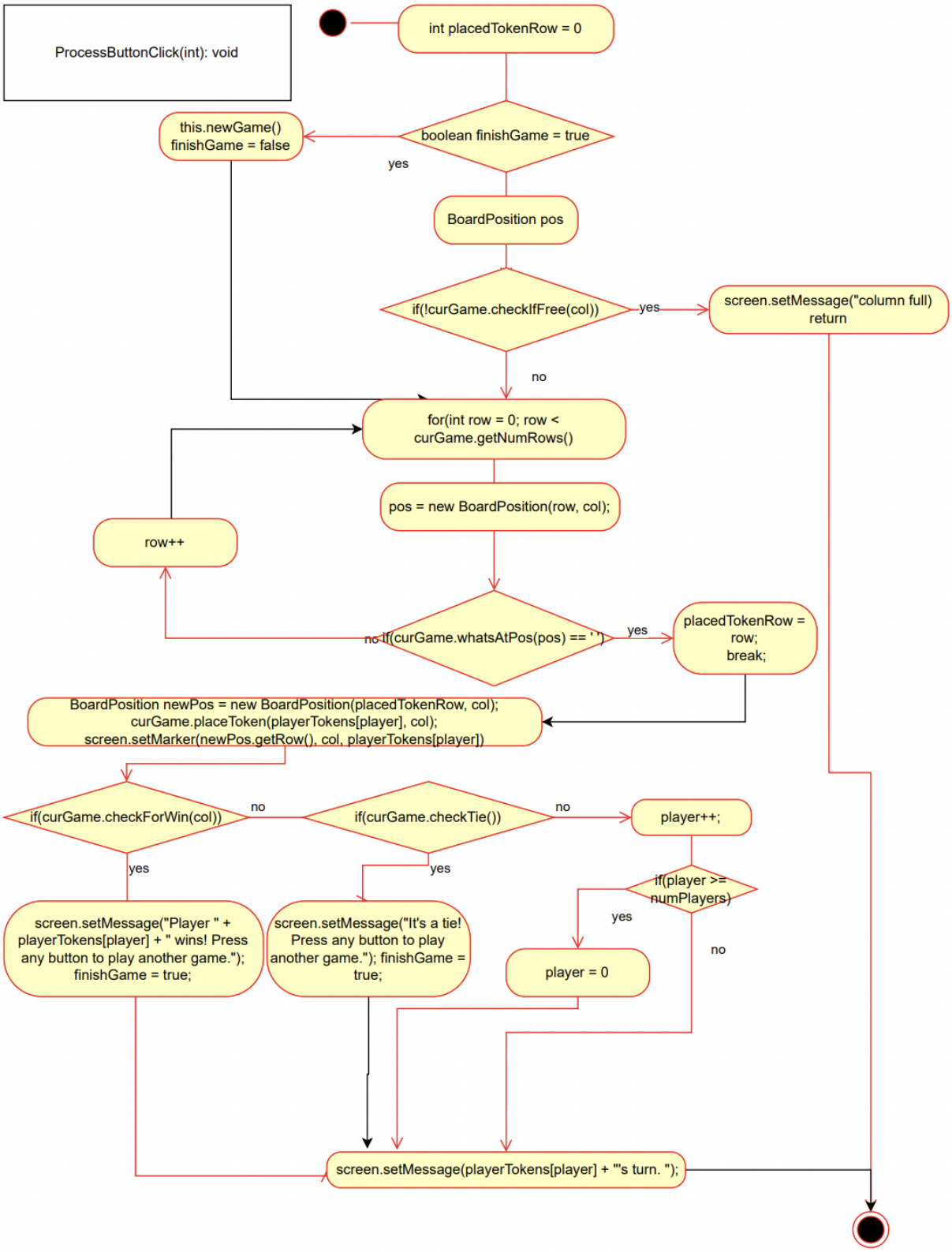
ConnectXController	
ConnectXController(IGameBoard, ConnectXView, int)	
numPlayers	int
finishGame	boolean
player	int
screen	ConnectXView
playerTokens	char[]
curGame	IGameBoard
MAX_PLAYERS	int
newGame()	void
processButtonClick(int)	void


ConnectXApp
ConnectXApp()
main(String[]) void

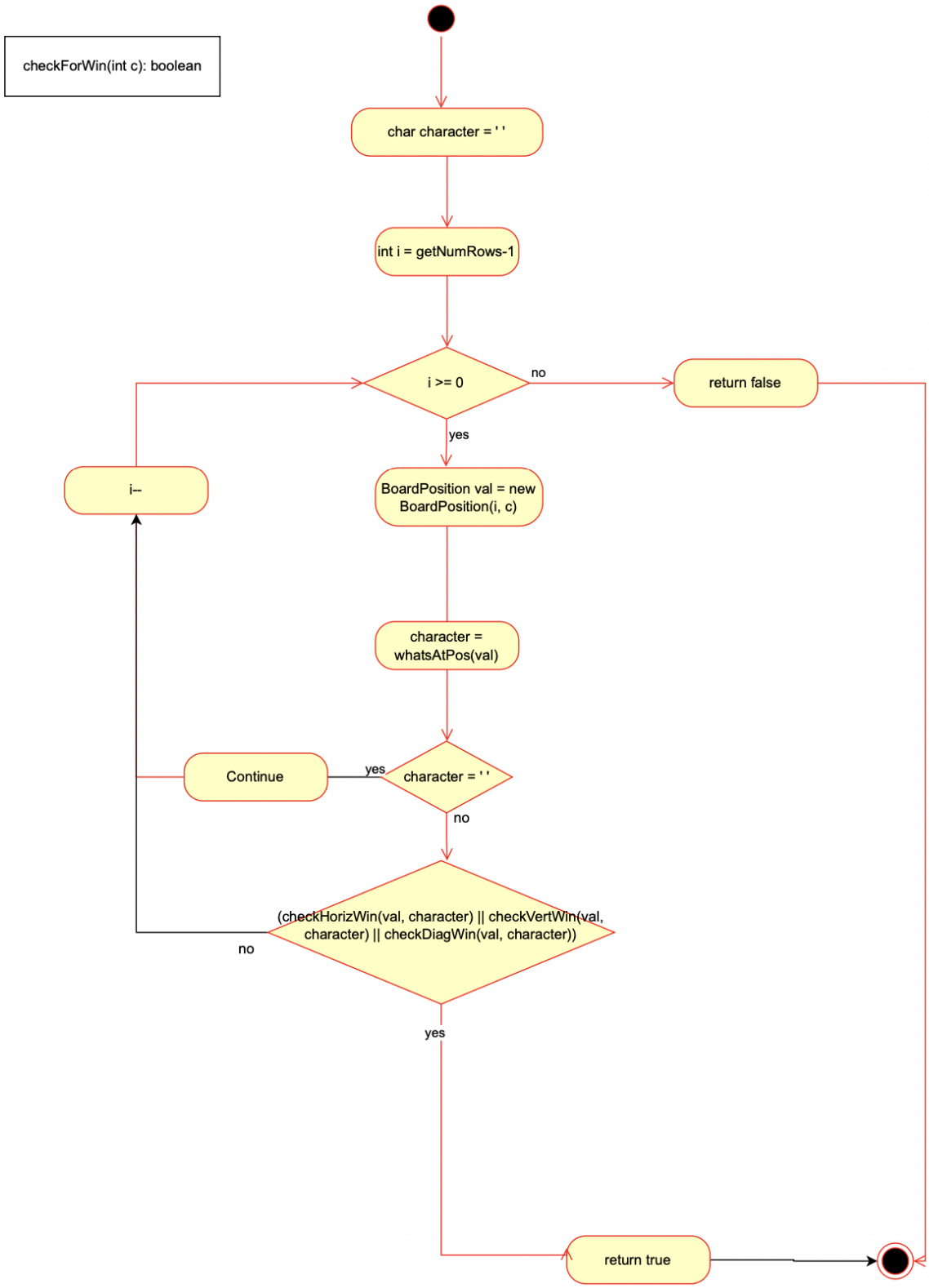

SetupController
SetupController(SetupView)
view SetupView
MIN_TO_WIN int
BOARD_CUTOFF int
MIN_SIZE int
MAX_SIZE int
processButtonClick(int, int, int, int) void
ConnectXController
ConnectXController(GameBoard, ConnectXView, int)
numPlayers int
finishGame boolean
player int
screen ConnectXView
playerTokens char[]
curGame IGameBoard
MAX_PLAYERS int
newGame() void
processButtonClick(int) void

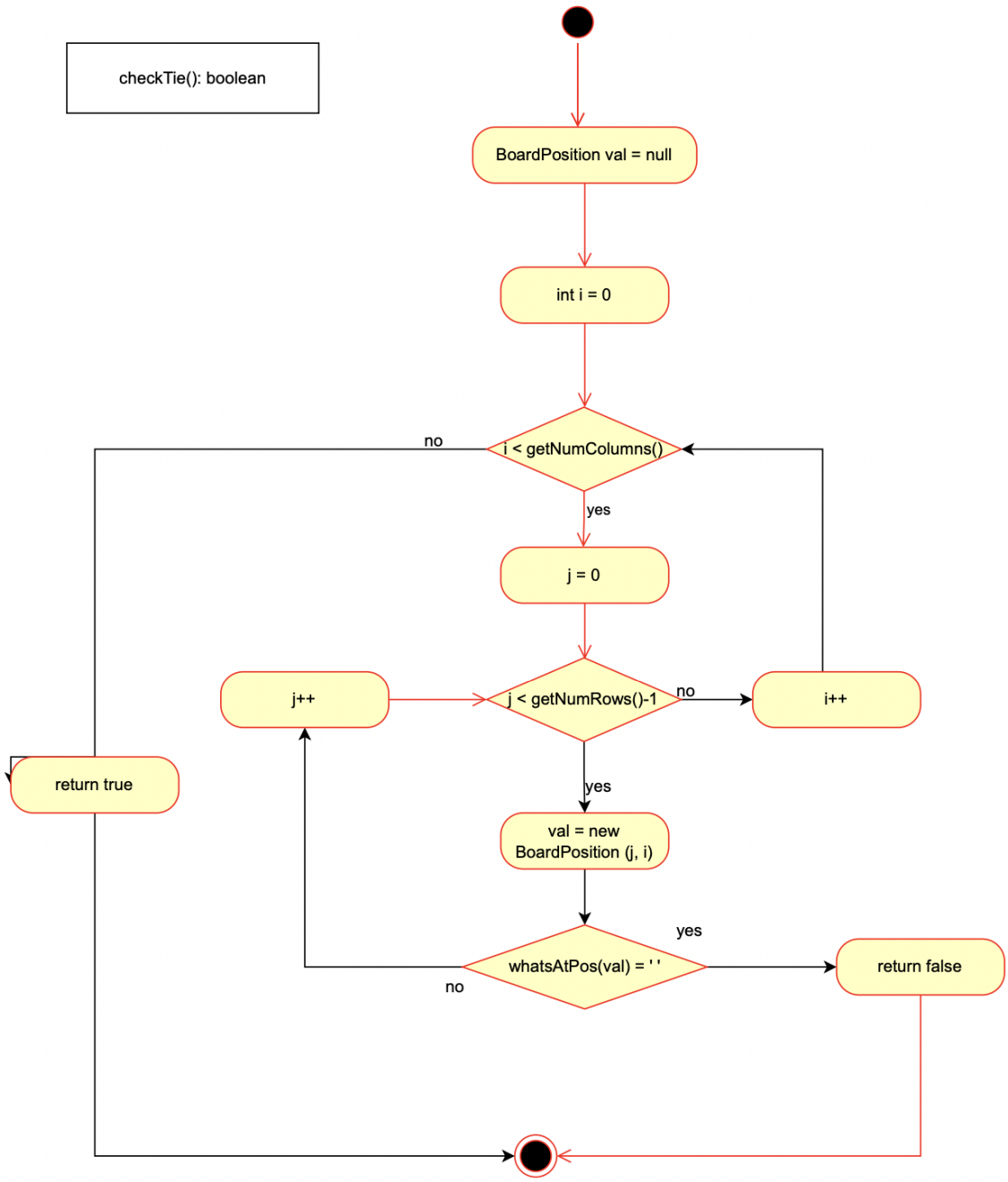

SetupView
SetupView()
numColsLbl JLabel
numColsTxt JTextArea
controller SetupController
numWinLbl JLabel
numWinTxt JTextArea
submitButton JButton
fontSize float
players Integer[]
errorMessageLbl JLabel
numPlayersLbl JLabel
numRowsTxt JTextArea
numRowsLbl JLabel
numPlayersCB JComboBox<Integer>
actionPerformed(ActionEvent) void
displayError(String) void
registerObserver(SetupController) void
closeScreen() void
ConnectXView
ConnectXView(int, int)
message JTextArea
controller ConnectXController
fontSize float
ROWS_IN_BUTTON_PANEL int
colButtons List<JButton>
buttons List<JButton>
LINES_IN_TEXT int
COLUMNS_IN_BUTTON_PANEL int
LINE_TEXT_LENGTHS int
registerObserver(ConnectXController) void
setMessage(String) void
actionPerformed(ActionEvent) void
setMarker(int, int, char) void

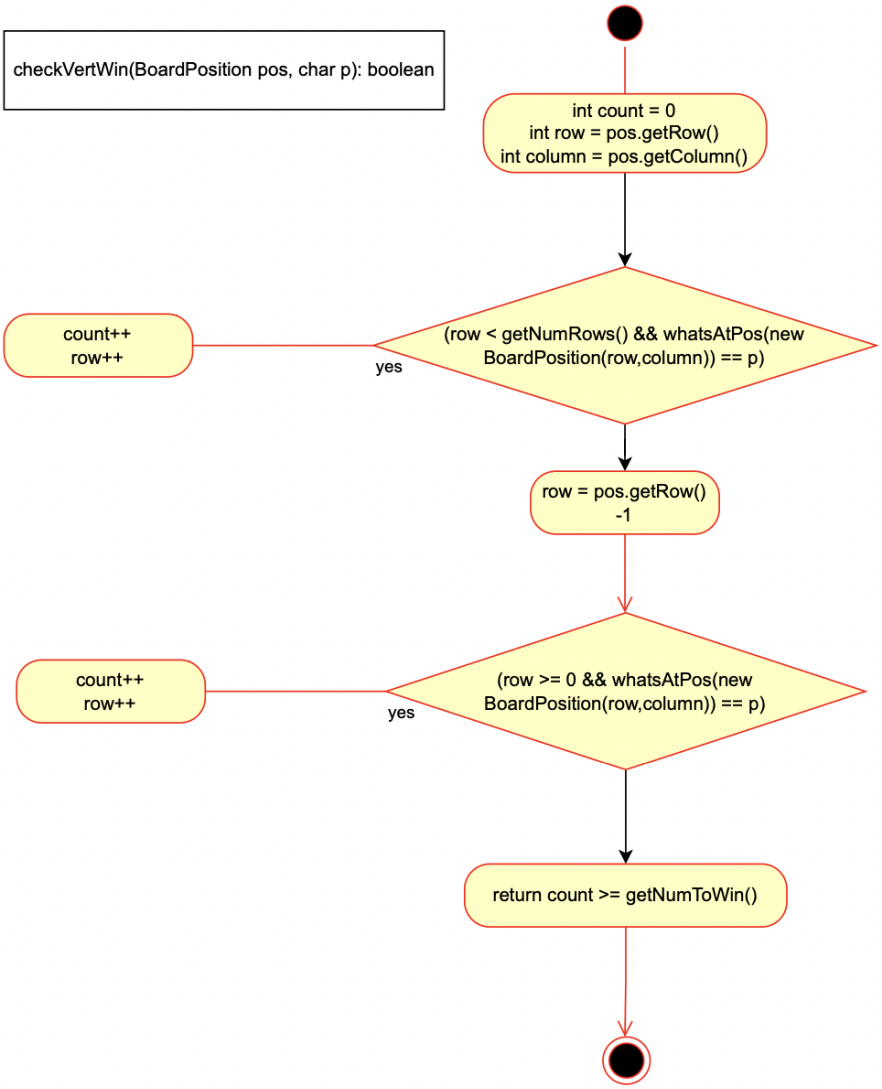

TestGameBoardMem
TestGameBoardMem()
testWhatsAtPos_correct_position() void
test_PlaceToken_bottom_right_position() void
testConstructor_minimum_size() void
testCheckDiagonalWin_start_right_barely_win() void
testIsPlayerAtPos_player_x() void
testIsPlayerAtPos_check_correct_position() void
testCheckIfFree_full_column() void
testCheckHorizontalWin_more_than_needed() void
testCheckTie_almost_full() void
testCheckDiagonalWin_start_left_more_than_needed() void
testCheckTie_all_pos_but_one() void
testIsPlayerAtPos_not_x_or_o() void
testCheckDiagonalWin_start_right_extra_win() void
createEmptyBoardArray(int, int) char[][]
testCheckVerticalWin_no_win() void
testCheckDiagonalWin_start_right_just_not_enough() void
testCheckHorizontalWin_empty() void
testConstructor_maximum_size() void
testConstructor_mismatch_rows_and_columns() void
testCheckVerticalWin_win() void
testPlaceToken_fill_entire_board() void
testPlaceToken_different_characters() void
testCheckVerticalWin_more_than_needed() void
testCheckDiagonalWin_start_left_not_enough() void
testCheckDiagonalWin_empty() void
testWhatAtPos_player_x() void
testCheckIfFree_single_token() void
testIsPlayerAtPos_correct_characters() void
testWhatsAtPos_player_not_x_or_o() void
testIsPlayerAtPos_empty() void
test_CheckTie_empty() void
testWhatsAtPos_empty() void
testCheckIfFree_empty() void
testWhatsAtPos_two_players_correct_character() void
testCheckHorizontalWin_normal_test() void
testPlaceToken_bottom_left_position() void
testCheckDiagonalWin_start_left_barely_win() void
testCheckHorizontalWin_incomplete() void
testCheckTie_full() void
boardArrayString(char[][], int, int) String
testCheckVerticalWin_incomplete() void
gb(int, int, int) IGameBoard
testPlaceToken_fill_different_characters() void
TestGameBoard
TestGameBoard()
testIsPlayerAtPos_empty() void
testCheckVerticalWin_more_than_needed() void
testCheckDiagonalWin_start_left_more_than_needed() void
createEmptyBoardArray(int, int) char[][]
test_PlaceToken_bottom_right_position() void
testCheckDiagonalWin_start_left_not_enough() void
testCheckVerticalWin_incomplete() void
testCheckDiagonalWin_empty() void
testPlaceToken_bottom_left_position() void
testPlaceToken_fill_entire_board() void
testPlaceToken_different_characters() void
testCheckIfFree_full_column() void
testWhatAtPos_player_x() void
testCheckDiagonalWin_start_right_extra_win() void
testIsPlayerAtPos_check_correct_position() void
testPlaceToken_fill_different_characters() void
gb(int, int, int) IGameBoard
testCheckTie_almost_full() void
testCheckDiagonalWin_start_right_just_not_enough() void
testCheckHorizontalWin_incomplete() void
testCheckTie_full() void
testCheckDiagonalWin_start_left_barely_win() void
testConstructor_mismatch_rows_and_columns() void
testWhatsAtPos_player_not_x_or_o() void
testCheckIfFree_empty() void
testCheckIfFree_single_token() void
testWhatsAtPos_correct_position() void
testWhatsAtPos_two_players_correct_character() void
testIsPlayerAtPos_correct_characters() void
boardArrayString(char[][], int, int) String
testWhatsAtPos_empty() void
testCheckHorizontalWin_empty() void
testCheckHorizontalWin_normal_test() void
testCheckDiagonalWin_start_right_barely_win() void
testIsPlayerAtPos_player_x() void
testCheckTie_empty() void
testCheckVerticalWin_win() void
testConstructor_minimum_size() void
testCheckVerticalWin_no_win() void
testCheckTie_all_pos_but_one() void
testCheckHorizontalWin_more_than_needed() void
testConstructor_maximum_size() void
testIsPlayerAtPos_not_x_or_o() void
IGameBoard
maxNumToWin int
minRowsCols int
maxRowsCols int
getNumRows() int
checkHorizWin(BoardPosition, char) boolean
placeToken(char, int) void
whatsAtPos(BoardPosition) char
checkForWin(int) boolean
checkDiagWin(BoardPosition, char) boolean
checkVertWin(BoardPosition, char) boolean
getNumColumns() int
getNumToWin() int
isPlayerAtPos(BoardPosition, char) boolean
checkTie() boolean
checkIfFree(int) boolean
AbsGameBoard
AbsGameBoard()
toString() String
GameBoard
GameBoard(int, int, int)
COLUMN int
Board char[][]
NUM_TO_WIN int
ROW int
placeToken(char, int) void
getNumColumns() int
whatsAtPos(BoardPosition) char
getNumRows() int
getNumToWin() int
GameBoardMem
GameBoardMem(int, int, int)
NUM_TO_WIN int
Board_Map Map<Character, List<BoardPosition>>
ROW int
COLUMN int
getNumColumns() int
getNumToWin() int
getNumRows() int
placeToken(char, int) void
whatsAtPos(BoardPosition) char
isPlayerAtPos(BoardPosition, char) boolean
BoardPosition
BoardPosition(int, int)
row int
column int
getRow() int
equals(Object) boolean
getColumn() int
toString() String

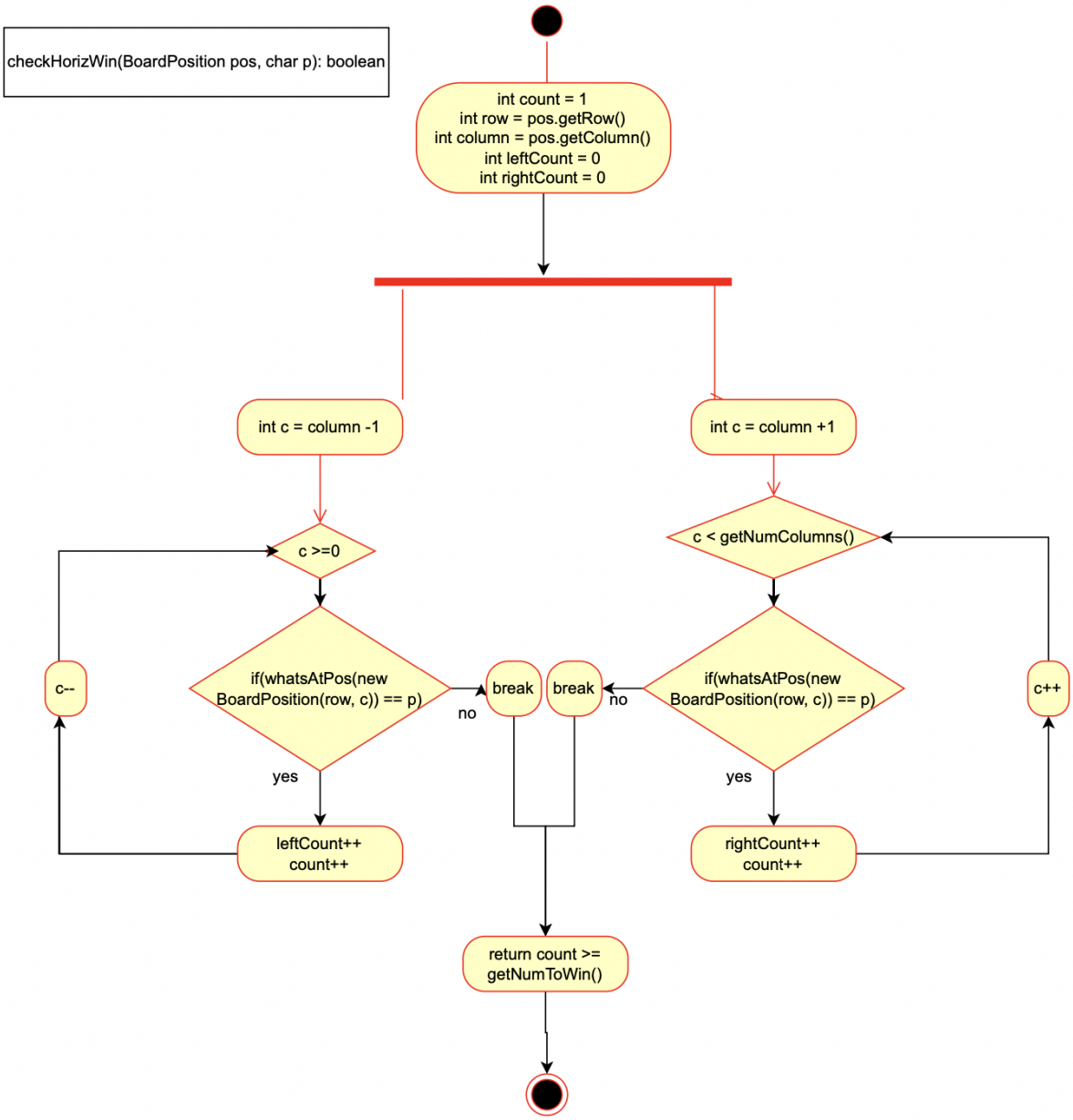

ProcessButtonClick(int): void
int placedTokenRow = 0
boolean finishGame = true
yes
this.newGame()
finishGame = false
BoardPosition pos
if(!curGame.checkIfFree(col))
yes
screen.setMessage("column full)
return
no
for(int row = 0; row <
curGame.getNumRows()
pos = new BoardPosition(row, col);
row++
no
if(curGame.whatsAtPos(pos) == ' ')
yes
placedTokenRow =
row;
break;
BoardPosition newPos = new BoardPosition(placedTokenRow, col);
curGame.placeToken(playerTokens[player], col);
screen.setMarker(newPos.getRow(), col, playerTokens[player])
if(curGame.checkForWin(col))
no
if(curGame.checkTie())
no
player++;
yes
yes
if(player >=
numPlayers)
yes
no
screen.setMessage("Player " +
playerTokens[player] + " wins! Press
any button to play another game.");
finishGame = true;
screen.setMessage("It's a tie!
Press any button to play
another game."); finishGame =
true;
player = 0
screen.setMessage(playerTokens[player] + "'s turn. ");

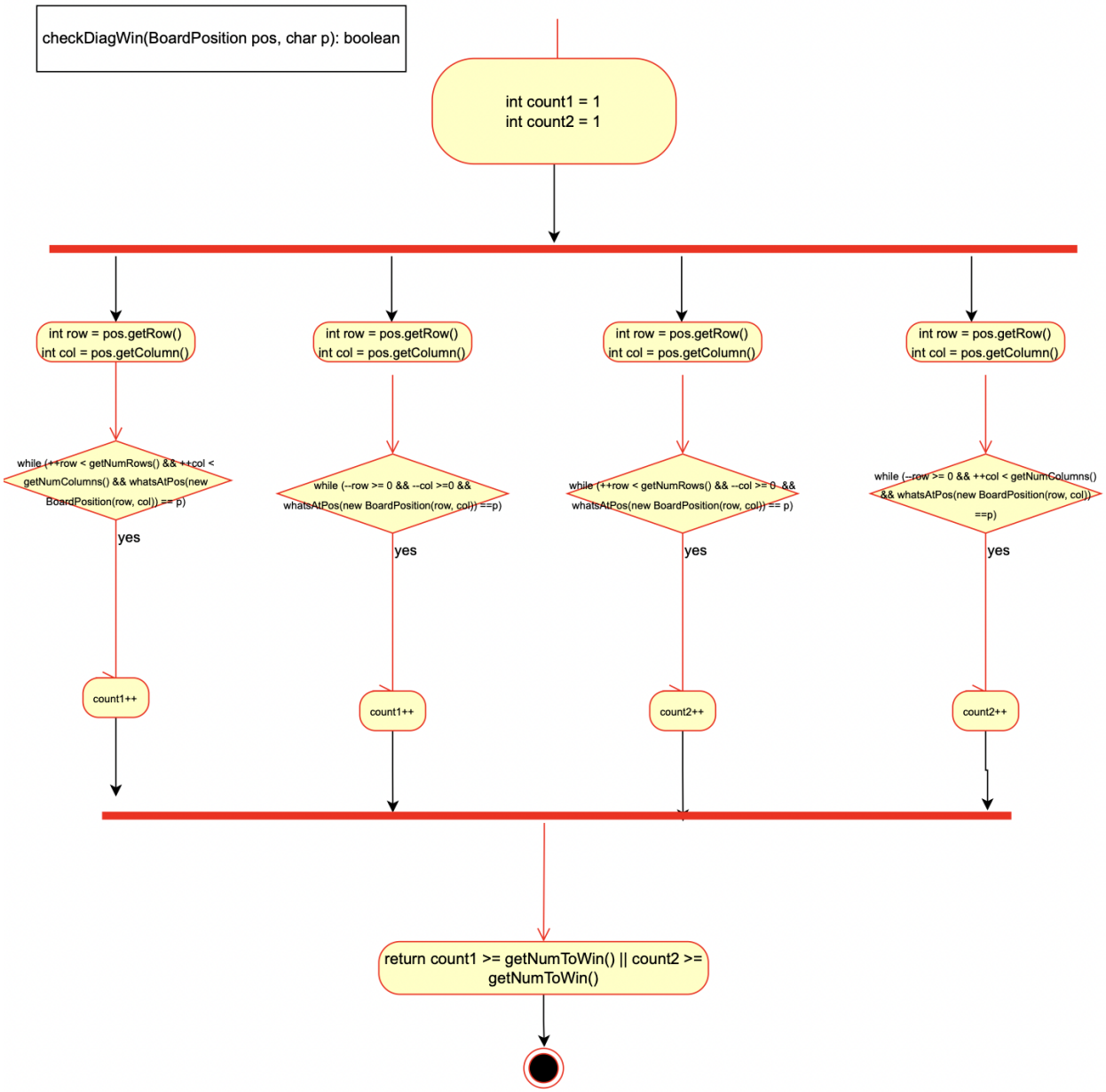
IGameBoard:


checkForWin(int c): boolean
char character = ' '
int i = getNumRows-1
i >= 0
no
return false
yes
i--
BoardPosition val = new BoardPosition(i, c)
character = whatsAtPos(val)
Continue
yes
character = ' '
no
(checkHorizWin(val, character) || checkVertWin(val, character) || checkDiagWin(val, character))
no
yes
return true

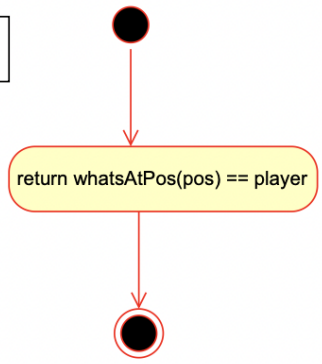

checkTie(): boolean
BoardPosition val = null
int i = 0
i < getNumColumns()
no
yes
j = 0
j++
j < getNumRows()-1
no
i++
return true
yes
val = new
BoardPosition (j, i)
yes
whatsAtPos(val) = ' '
return false
no

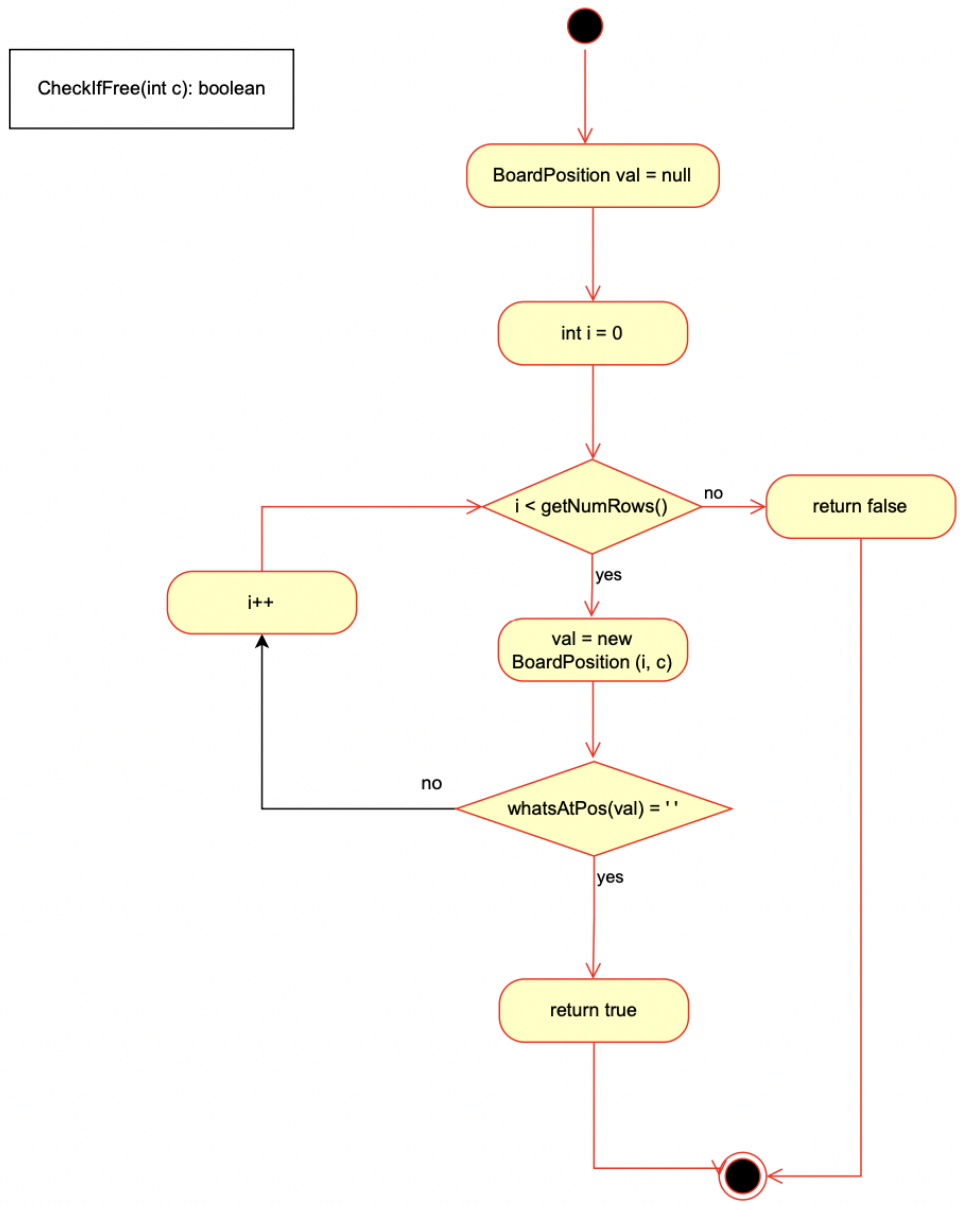

checkVertWin(BoardPosition pos, char p): boolean
int count = 0
int row = pos.getRow()
int column = pos.getColumn()
(row < getNumRows() && whatsAtPos(new BoardPosition(row,column)) == p)
yes
count++
row++
row = pos.getRow()
-1
(row >= 0 && whatsAtPos(new BoardPosition(row,column)) == p)
yes
count++
row++
return count >= getNumToWin()


checkHorizWin(BoardPosition pos, char p): boolean
int count = 1
int row = pos.getRow()
int column = pos.getColumn()
int leftCount = 0
int rightCount = 0
int c = column -1
int c = column +1
c >=0
c < getNumColumns()
c--
if(whatsAtPos(new BoardPosition(row, c)) == p)
break
break
if(whatsAtPos(new BoardPosition(row, c)) == p)
c++
no
no
yes
yes
leftCount++
count++
rightCount++
count++
return count >= getNumToWin()


checkDiagWin(BoardPosition pos, char p): boolean
int count1 = 1
int count2 = 1
int row = pos.getRow()
int col = pos.getColumn()
while (++row < getNumRows() && ++col < getNumColumns() && whatsAtPos(new BoardPosition(row, col)) == p)
yes
count1++
int row = pos.getRow()
int col = pos.getColumn()
while (--row >= 0 && --col >=0 && whatsAtPos(new BoardPosition(row, col)) ==p)
yes
count1++
int row = pos.getRow()
int col = pos.getColumn()
while (++row < getNumRows() && --col >= 0 && whatsAtPos(new BoardPosition(row, col)) == p)
yes
count2++
int row = pos.getRow()
int col = pos.getColumn()
while (--row >= 0 && ++col < getNumColumns() && whatsAtPos(new BoardPosition(row, col)) ==p)
yes
count2++
return count1 >= getNumToWin() || count2 >= getNumToWin()

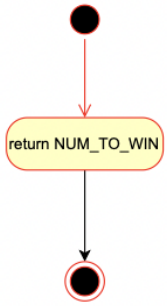
isPlayerAtPos(BoardPosition pos, char player):boolean



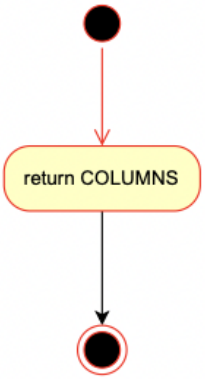

CheckIfFree(int c): boolean
BoardPosition val = null
int i = 0
i < getNumRows()
no
return false
yes
i++
val = new
BoardPosition (i, c)
no
whatsAtPos(val) = ' '
yes
return true

GameBoard:

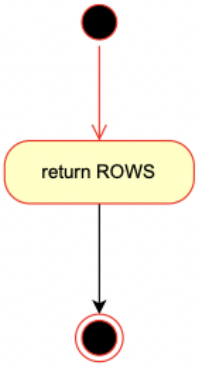
getNumToWin():int



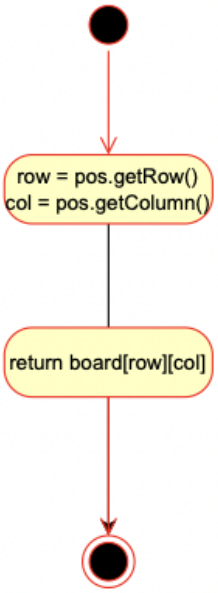
getNumColumns():int



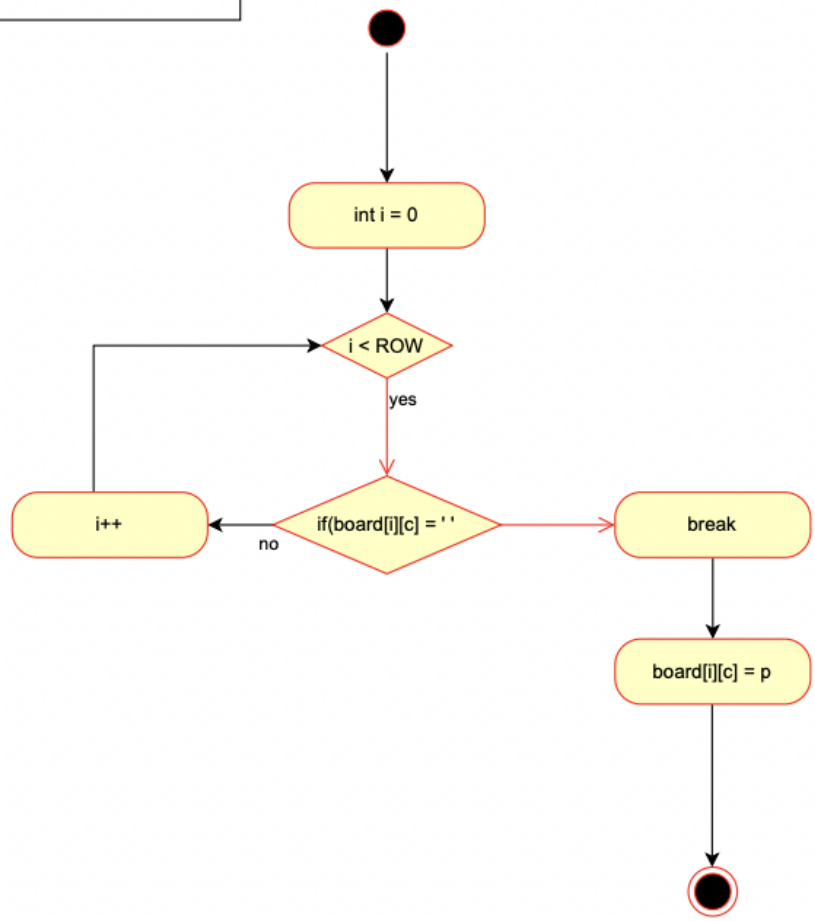
getNumRows():int

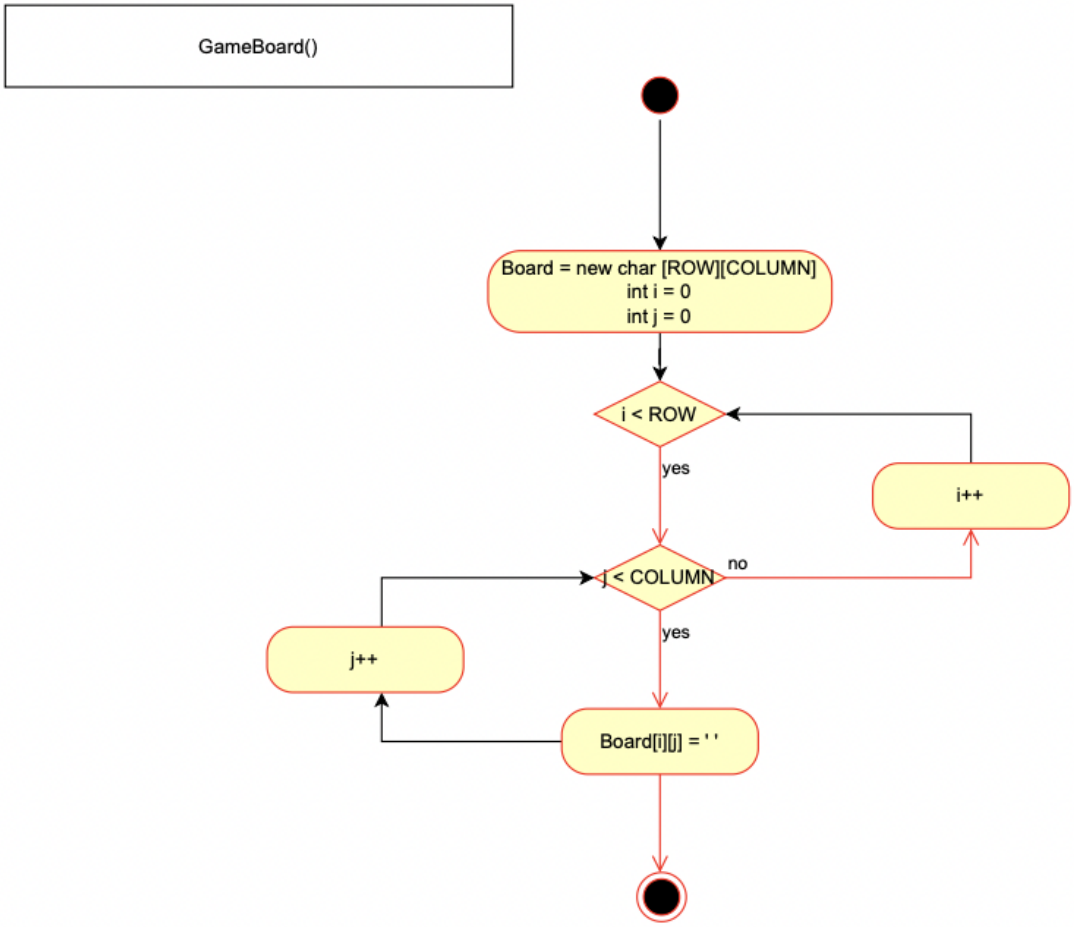


whatsAtPos(BoardPosition pos): char

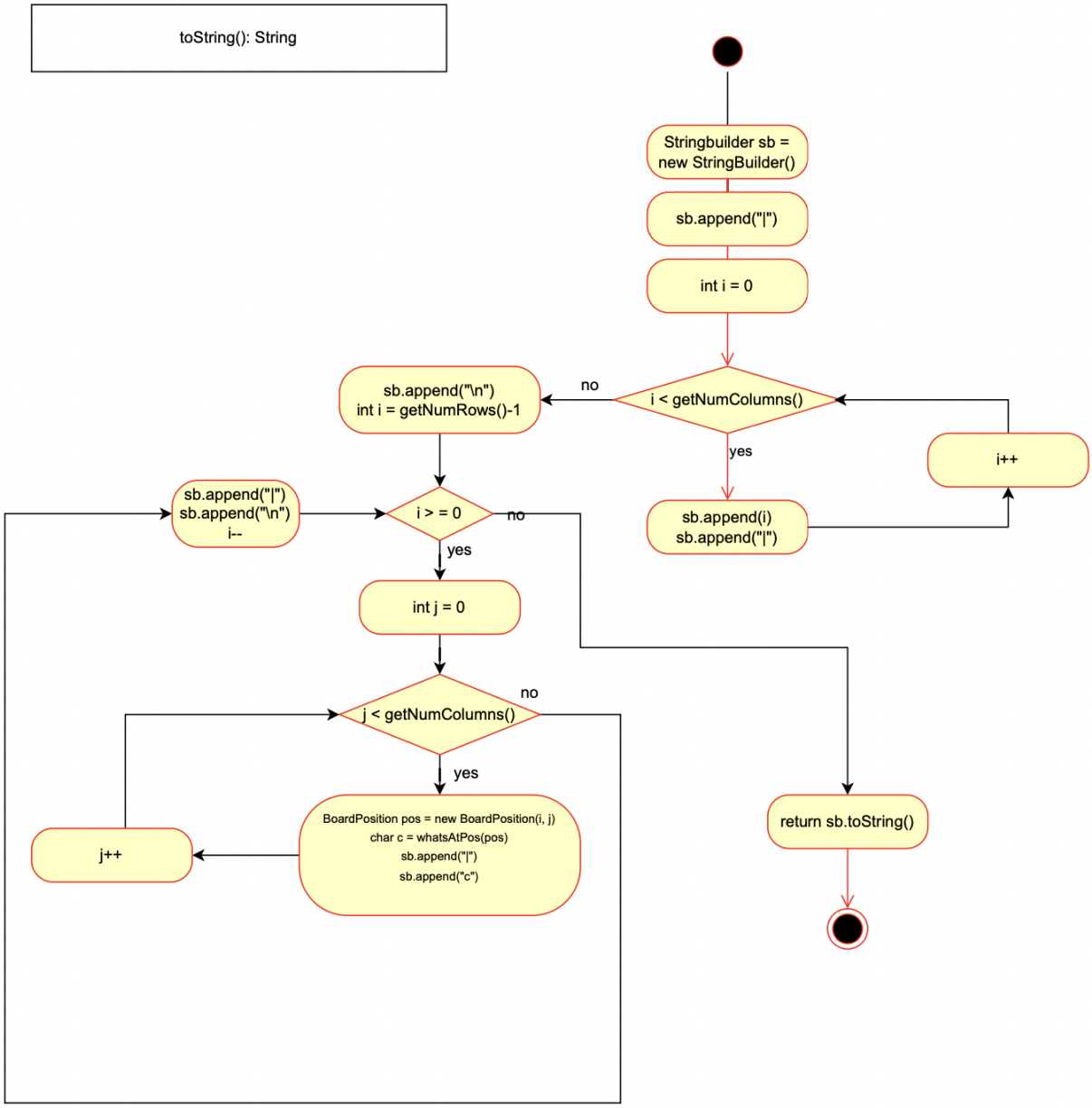


placeToken(char p, int c): void

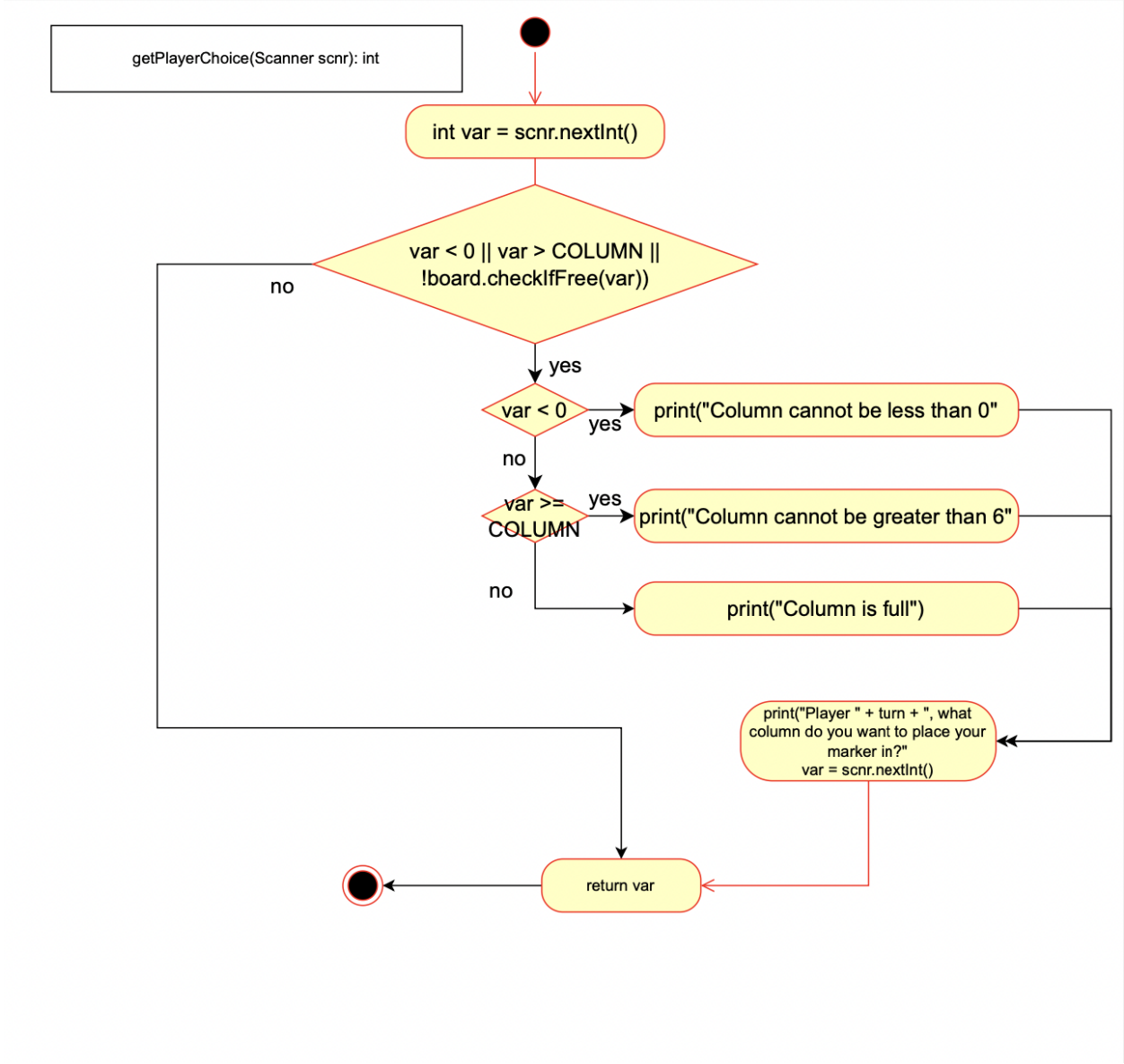


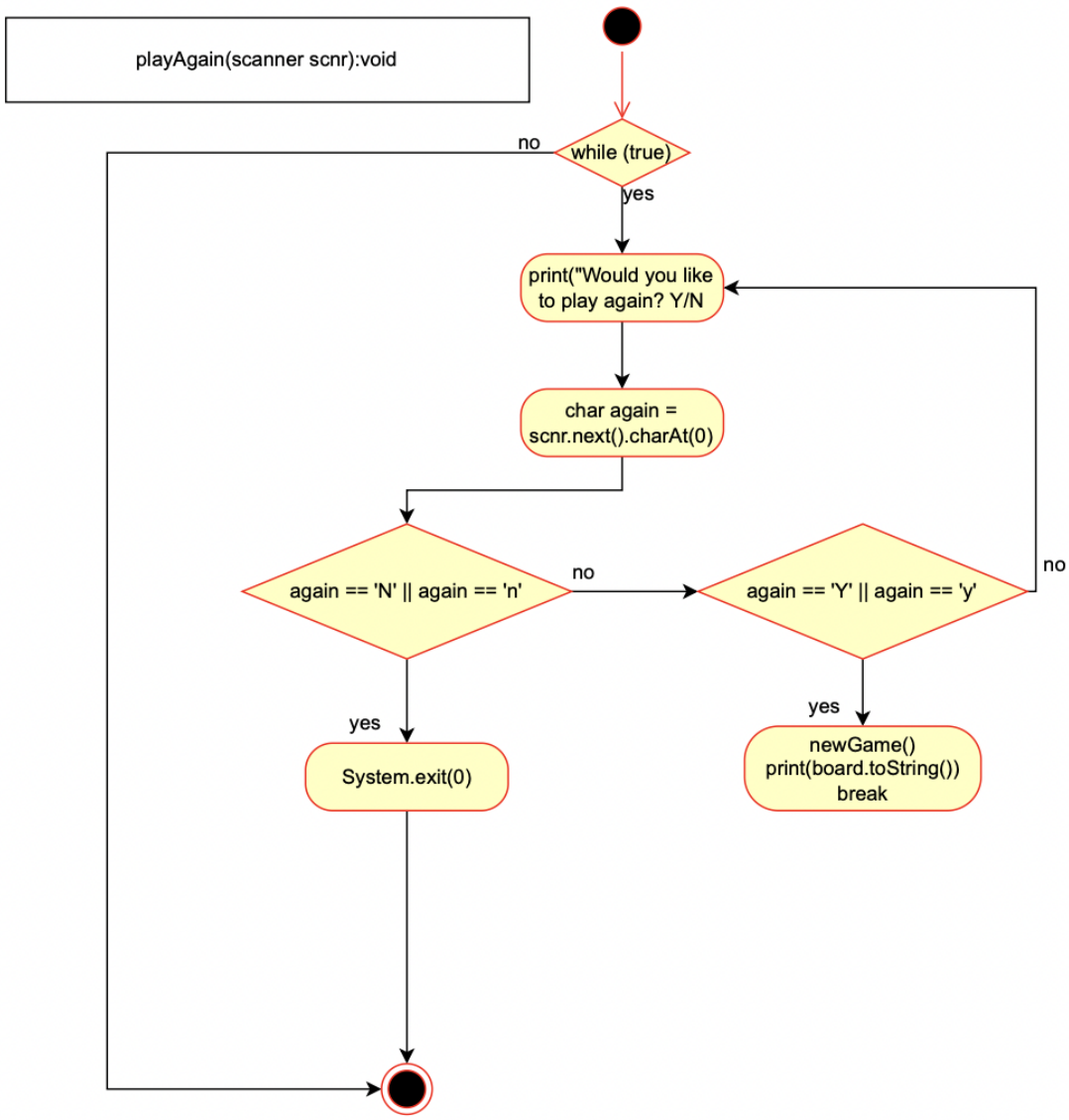

GameBoard()
Board = new char [ROW][COLUMN]
int i = 0
int j = 0
i < ROW
yes
i++
j < COLUMN
no
yes
j++
Board[i][j] = ' '

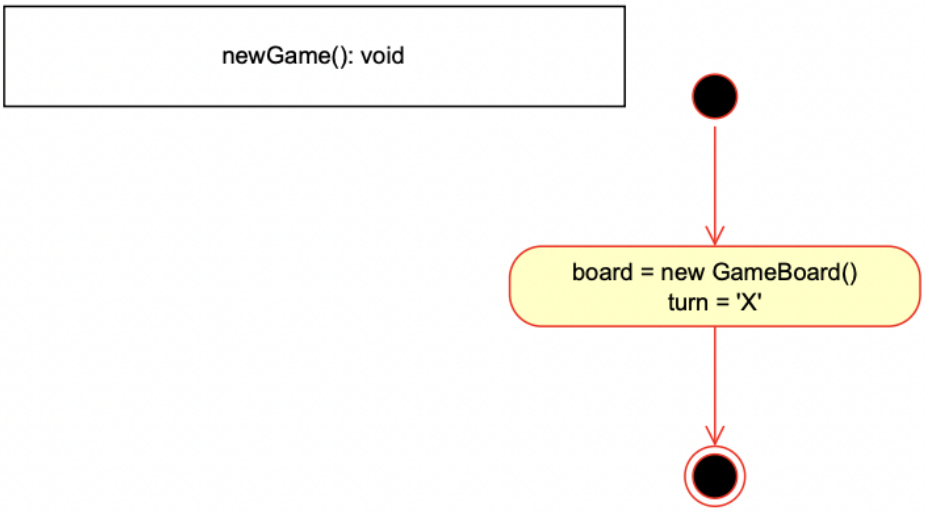
AbsGameBoard:

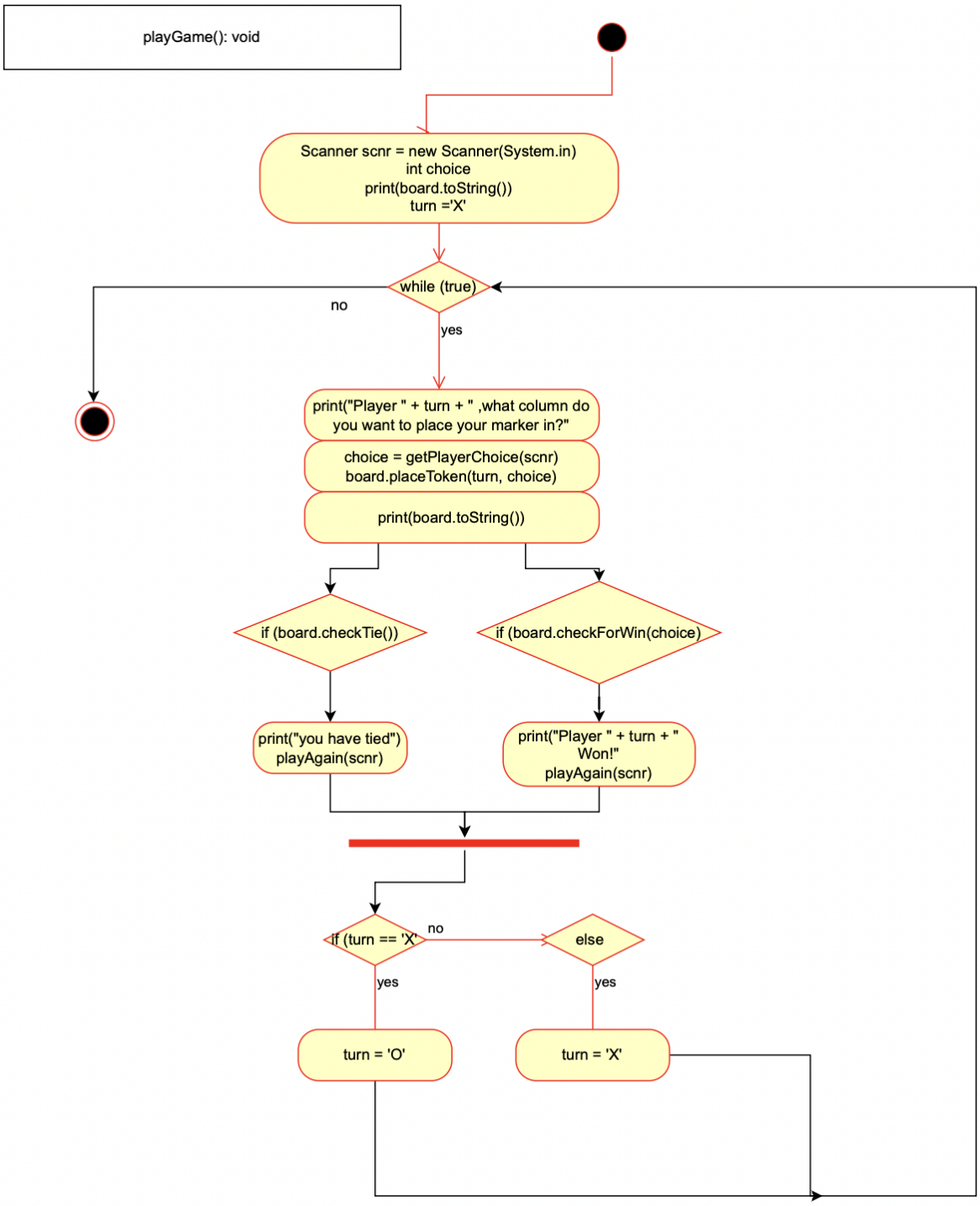


GameScreen:


getPlayerChoice(Scanner scnr): int
int var = scnr.nextInt()
var < 0 || var > COLUMN ||
!board.checkIfFree(var))
no
yes
var < 0
yes
print("Column cannot be less than 0"
no
var >=
COLUMN
yes
print("Column cannot be greater than 6"
no
print("Column is full")
print("Player " + turn + ", what
column do you want to place your
marker in?"
var = scnr.nextInt()
return var


playAgain(scanner scnr):void
while (true)
no
yes
print("Would you like to play again? Y/N
char again = scnr.next().charAt(0)
again == 'N' || again == 'n'
no
again == 'Y' || again == 'y'
no
yes
System.exit(0)
yes
newGame()
print(board.toString())
break


newGame(): void
board = new GameBoard()
turn = 'X'


playGame(): void
Scanner scnr = new Scanner(System.in)
int choice
print(board.toString())
turn ='X'
while (true)
no
yes
print("Player " + turn + " ,what column do you want to place your marker in?"
choice = getPlayerChoice(scnr)
board.placeToken(turn, choice)
print(board.toString())
if (board.checkTie())
if (board.checkForWin(choice)
print("you have tied")
playAgain(scnr)
print("Player " + turn + " Won!"
playAgain(scnr)
if (turn == 'X'
no
else
yes
yes
turn = 'O'
turn = 'X'