CPSC 2150 Project 3 Report

Keenan Grant

Requirements Analysis

Functional Requirements:

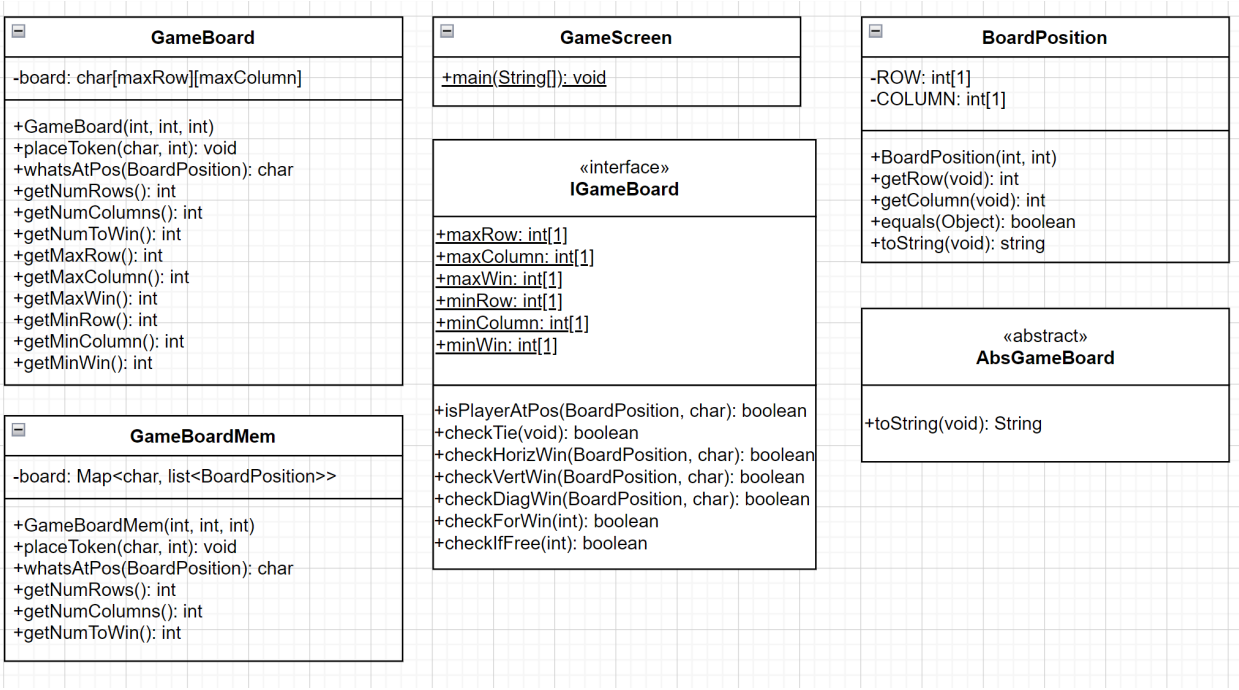
1. As a player I need to see the board after every play so I can make the right decisions.
2. As a player I want to be able to choose if I want to play the game again after it has ended so that I wouldn't have to keep restarting the program each time.
3. As a player I need to be able to distinguish between the game markers so I can know which one I'm playing as.
4. As a player I need to be able to choose which column I want to place my marker in so I can execute my strategy correctly.
5. As a player I need to be able to see how many rows and columns there are so I can execute my strategy correctly.
6. As a player I need to be able to see where my opponent's markers are so I can block my opponent from winning.
7. As a player I need to be able to pick again if I picked a full column so my turn won't be skipped.
8. As a player I need to be able to pick again if I picked a non-existent column so my turn won't be skipped.
9. As a player I need to be able to win the game horizontally, vertically, or diagonally so the game will end.
10. As a player I need to know if the game ended in a tie so that a new game could be started.
11. As a player I need to be able to place a token after my opponent so I could try to beat them before they win.
12. As a player I need to be able to choose how many players, rows, columns, and tokens in a row to win so I can make the game.

Non-Functional Requirements

1. The program must be written in Java.
2. The program must follow the rules of ConnectX.
3. The program must have overridden methods.

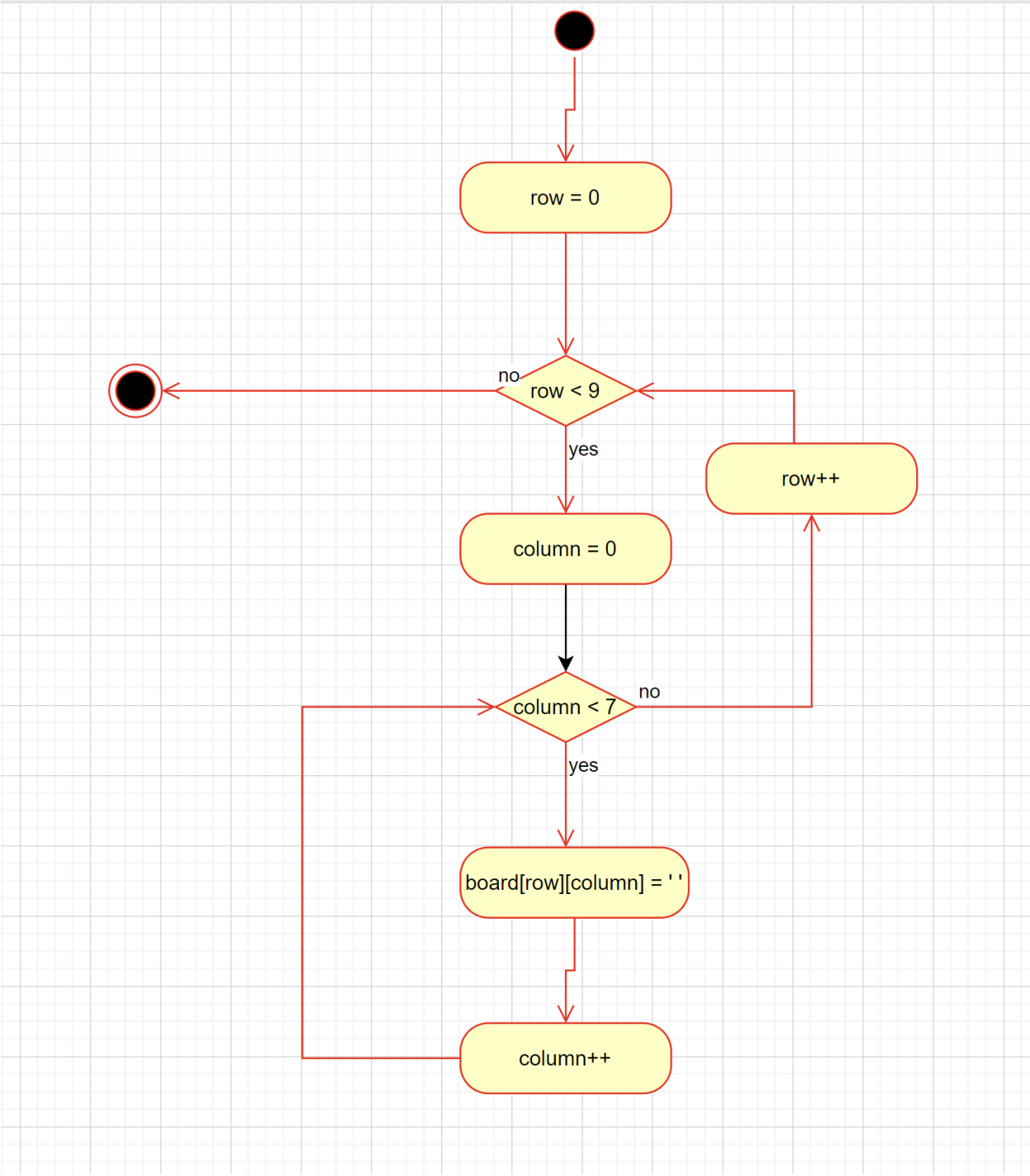
4. The program must have multiple classes.
5. The program must use a 2D array or a map for the game board.
6. The game board must not be larger than 100x100 and no smaller than 3x3.
7. The number of players must be no more than 10 and no smaller than 2.
8. The number of tokens in a row needed to win must be no larger than 25, no larger than the number of rows and columns, and no smaller than 3.
9. The program must validate the number of rows, columns, and tokens needed to win that is provided by the user.
10. After selecting the number of players, each player will be able to pick what character will represent them in the game.
11. At the start of each new game, the players can choose whether they want a fast implementation or a memory-efficient implementation.
12. The number of players and their characters can change if they choose to start a new game.
13. The choice of fast or memory-efficient implementation can change if they start a new game.

Class Diagrams

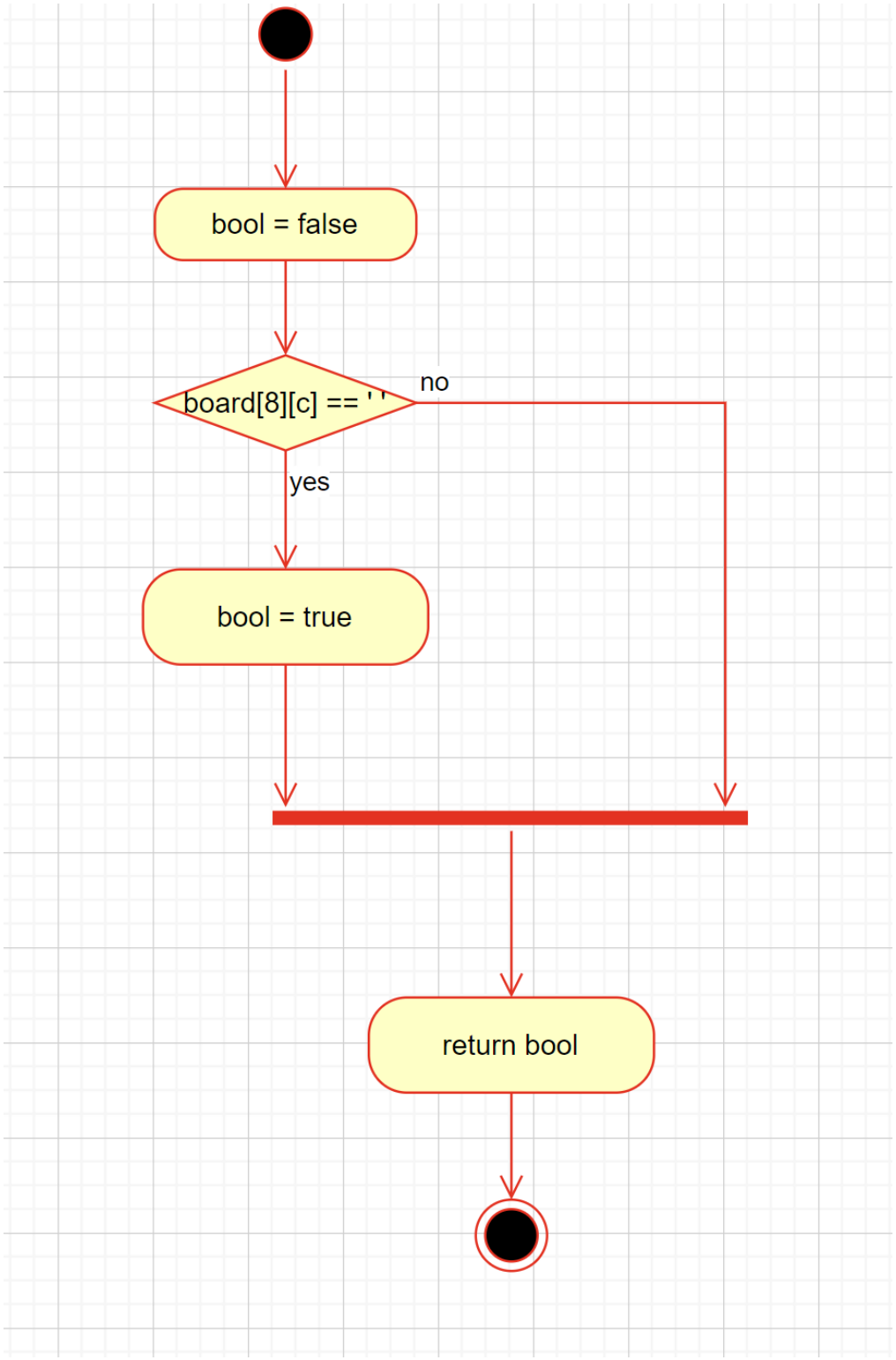


Activity Diagrams

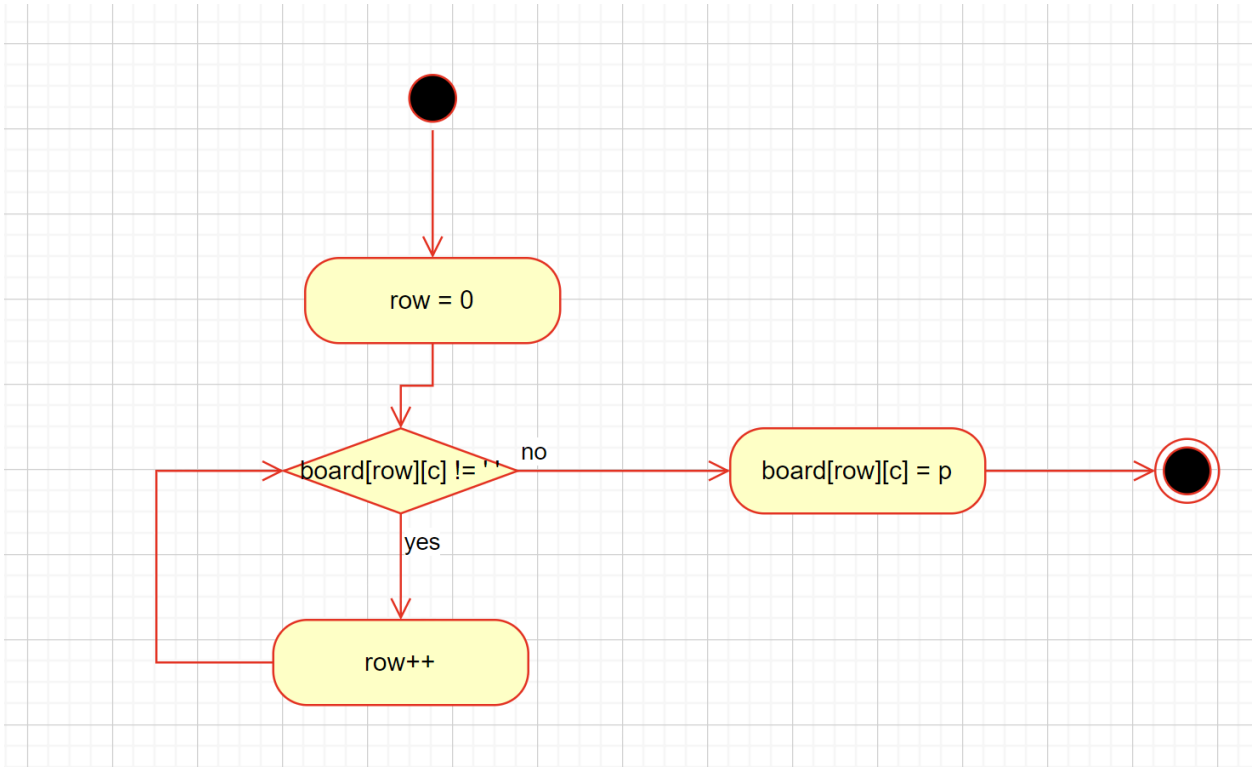
GameBoard(void):



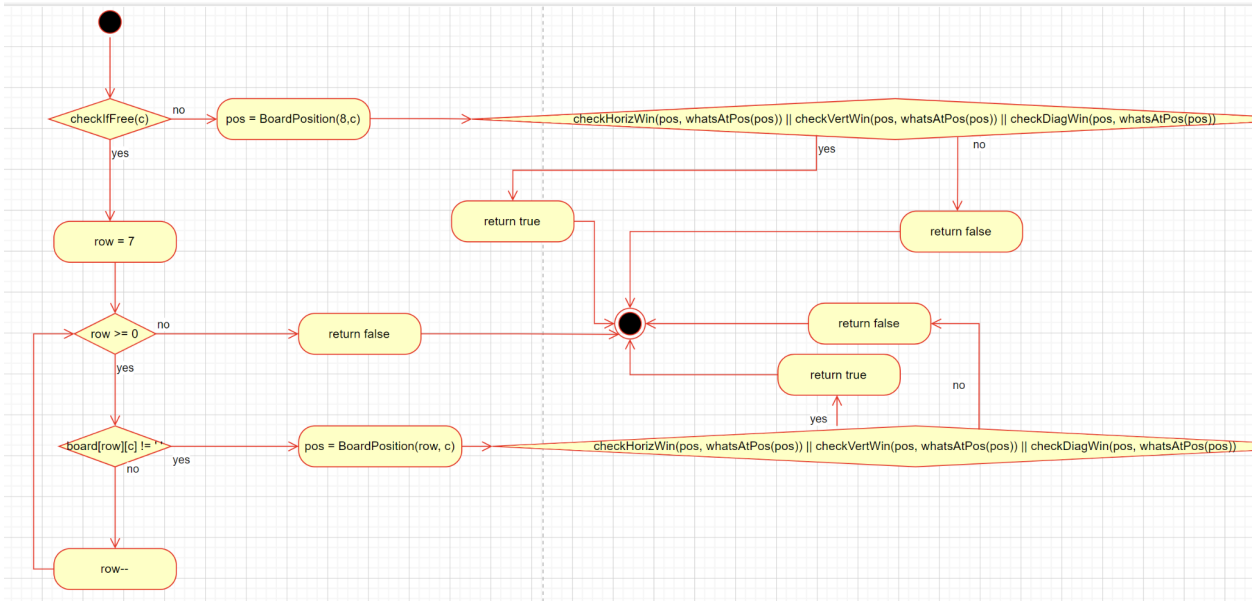
checkIfFree(int):



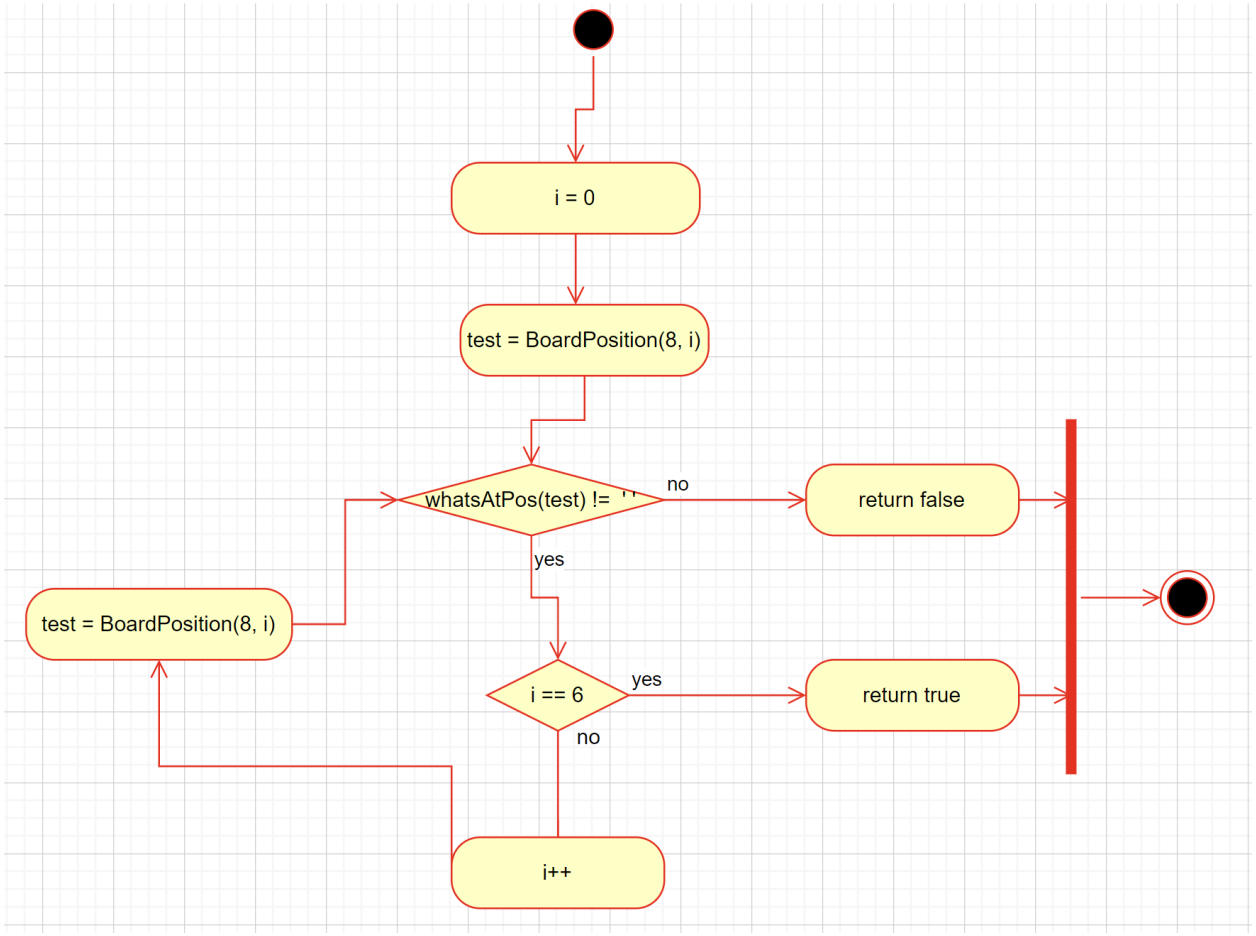
placeToken(char, int):



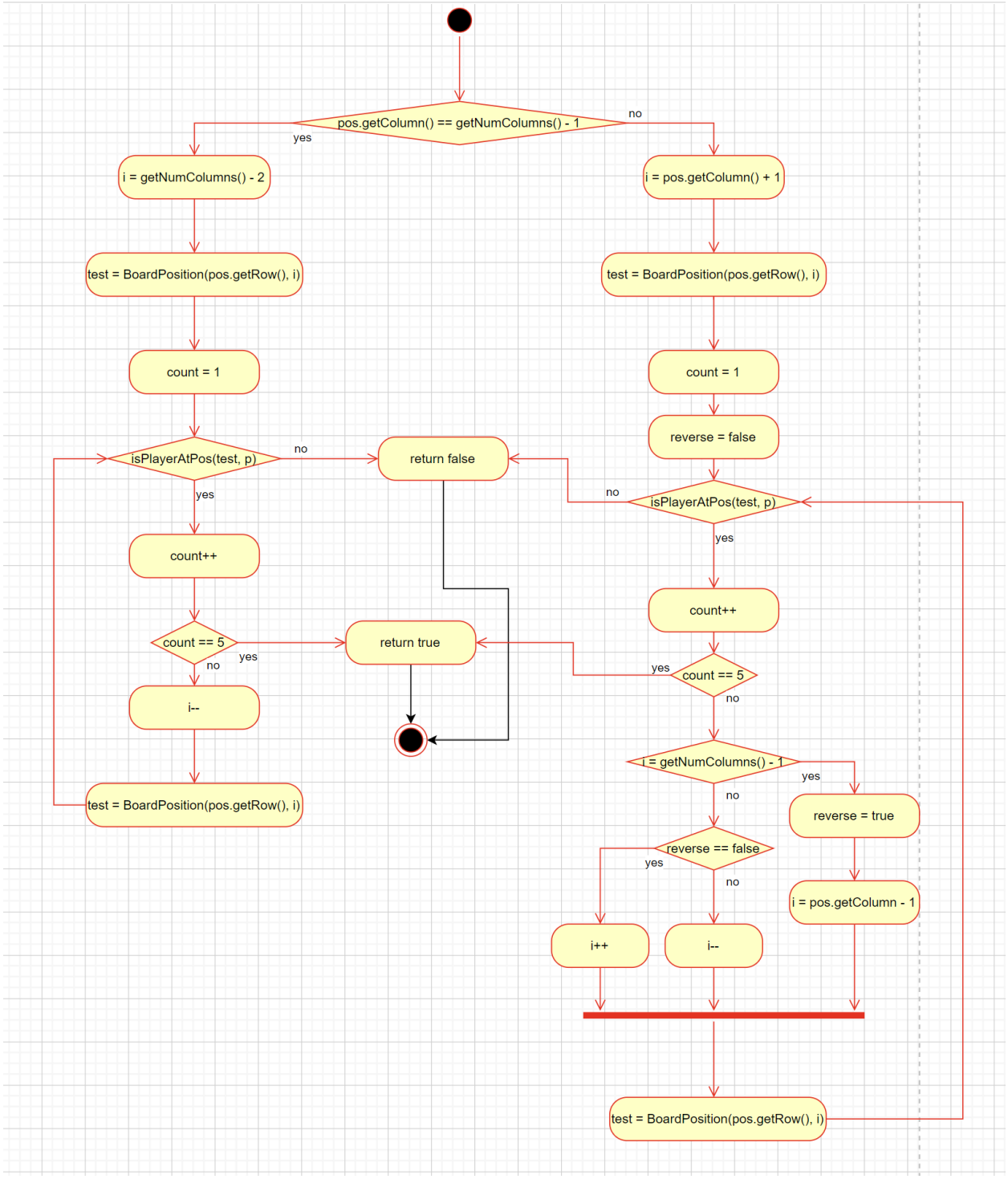
checkForWin(int):



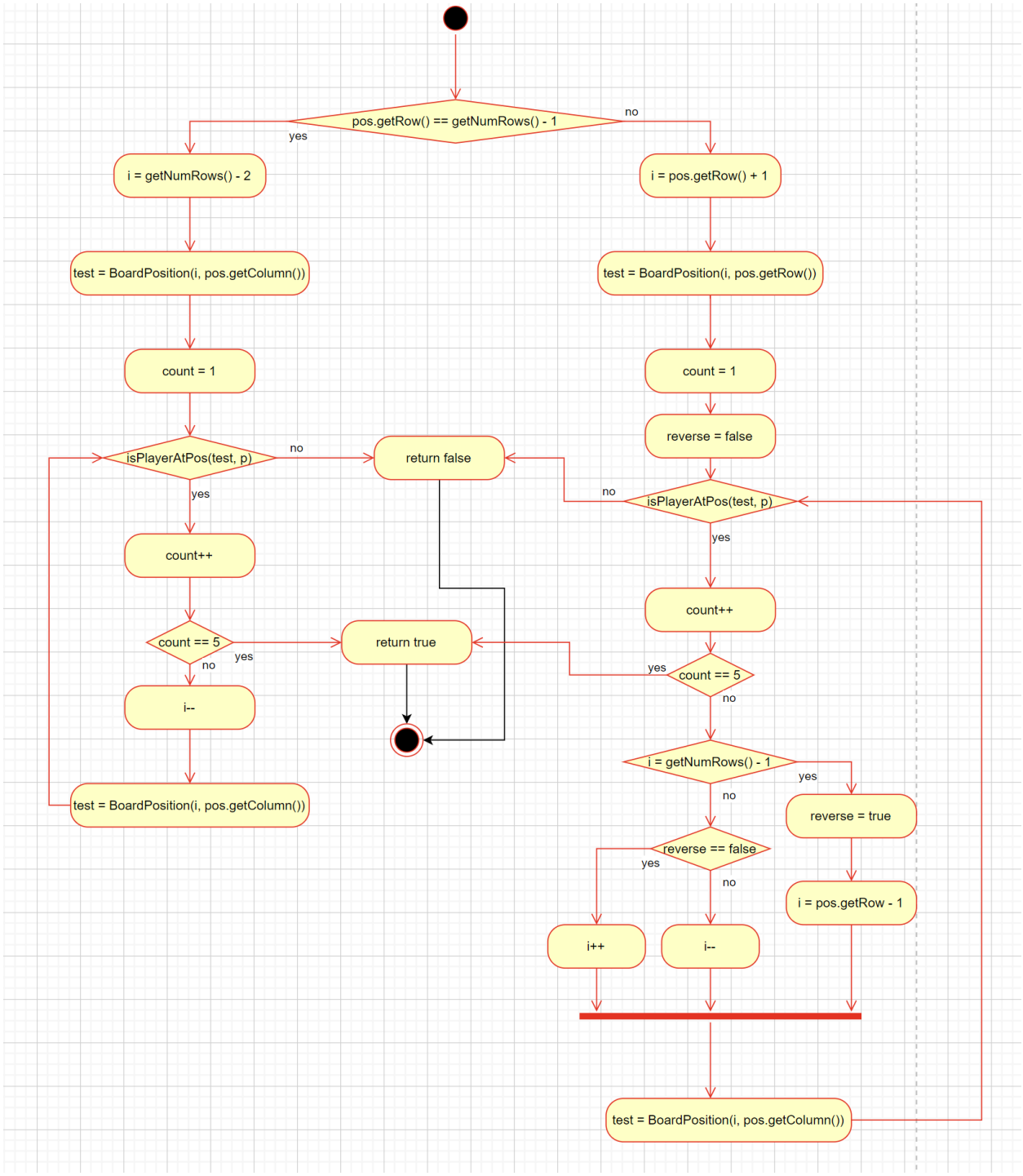
checkTie(void):



checkHorizWin(BoardPosition, char):

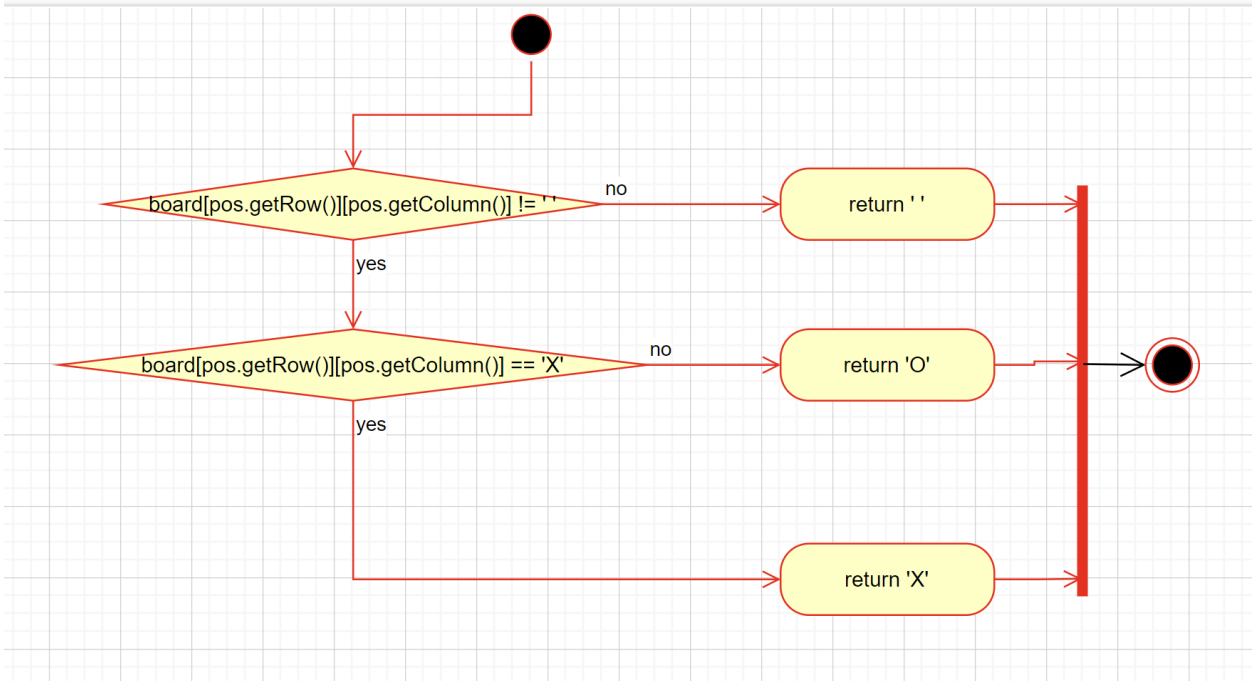


checkVertWin(BoardPosition, char):

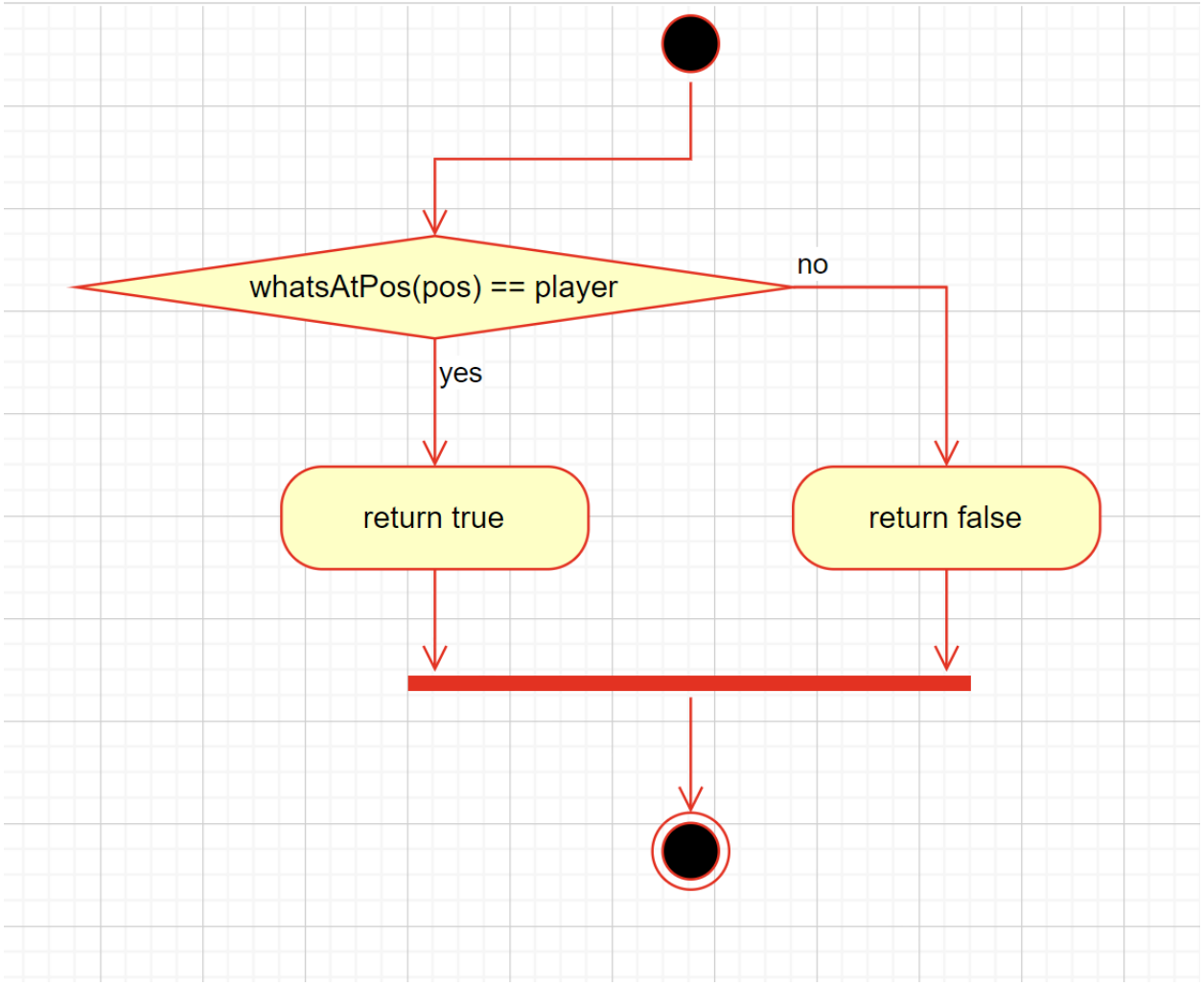


checkDiagWin(BoardPosition, char):

whatsAtPos(BoardPosition):



isPlayerAtPos(BoardPosition, char):



toString(void):