# CPSC 2150 Project Three Report

Jake Macdonald

**Functional Requirements:**

1. As a player I can enter the desired column and row location for my marker to progress the game.
2. As a player I need to have a unique marker type to distinguish myself.
3. As a player I need the marker type to switch value for every other move to make this a versus game.
4. As a player up to 10 players may register into a game to make it competitive.
5. As a player two or more players must be playing the game to make it competitive.
6. As a player, players should place markers in the same ordering as player symbol selection.
7. As a player I must be able to specify the dimensions of the board for the board to be dynamic.
8. As a player the dimensions of a board must be at maximum 100 X 100 to prevent absurd playthroughs
9. As a player the dimensions of a board must be at minimum 3 X 3 to prevent absurd playthroughs
10. As a player I need indication of inappropriate input so that I may correct the mistake.
11. As a player I need an updating display of the board so I may watch the game progress.
12. As a player I need to know when win conditions have been satisfied so the game may conclude.
13. As a player I need to know when the game has been drawn so I may be aware to restart the game.
14. As a player I need to be prompted after concluding to game to begin another, so I don't need to rerun the application.
15. As a player I need the ability to place and store a marker at desired positions to play the game.
16. As a player I need the board to be designed withing a grid of specified size to play a specific variety of tic-tac-toe.
17. As a player I must be able to indicate how many rows to win for games to be dynamic.
18. As a player the number of markers I enter must be an element within the range [3,25] to prevent absurd playthroughs.
19. As a player I should be given the option to chose between a time or memory efficient run of the game so I can play better.
20. As a player I need the win condition to be a line of specified number of adjacent similar markings so to play a specific variety of tic-tac-toe.
21. As a player I need the gameboard display to be expressed in a readable manner so I may bear greater witness.
22. As a player, the unmarked locations of the board should possess the default value, ' ' so it may be clear where available places positions are.
23. As a player, the placed marker value should be any value entered by players that are capitalized alphabetical characters.

**Non-Functional Requirements**

1. The application must be developed in Java.
2. The application must function in the Ubuntu v.20 environment.
3. The class, GameScreen will possess the only main function of the program.
4. The application must exclusively use three classes: GameBoard, GameBoardMem, GameScreen, and BoardPosition.
5. The application must exclusively use three classes: GameBoard, GameBoardMem, GameScreen, and BoardPosition.
6. The IGameBoard interface must be used.
7. The AbsGameBoard abstract class must be used for the overriding toString
8. The class GameBoard must exclusively use methods prescribed within assignment documentation.
9. Attributes of the class, BoardPosition, must exclusively be accessible by getter methods.
10. All U/I interaction will be exclusively preformed within the GameScreen method.
11. BoardPosition will have only one constructor method.
12. BoardPosition attributes may only be set within the constructor.
13. BoardPosition must have a methods overriding the equals() and toString() methods.
14. All attributes of GameBoard must be private unless they are static and final.
15. Gameboard is of size user inputted size
    - I think this is more of a functional requirement than non because this requirement is made explicitly apparent while in use. But I got points off last time for this not being here.
16. GameBoard will extend AbsGameboard.
17. GameBoardMem will extend IGameboard.
18. GameBoard will implement IGameboard.
19. GameBoardMem will implement AbsGameboard.
20. AbsGameBoard implements IGameBoard.
21. Board element (0,0) should be top position of the board
22. No dead code should be present in the project
23. Makefile should have targets : default, run, and clean

24. Two implementations of IGameBoard must be usable, both possessing strength and weaknesses.

**makefile instructions:**

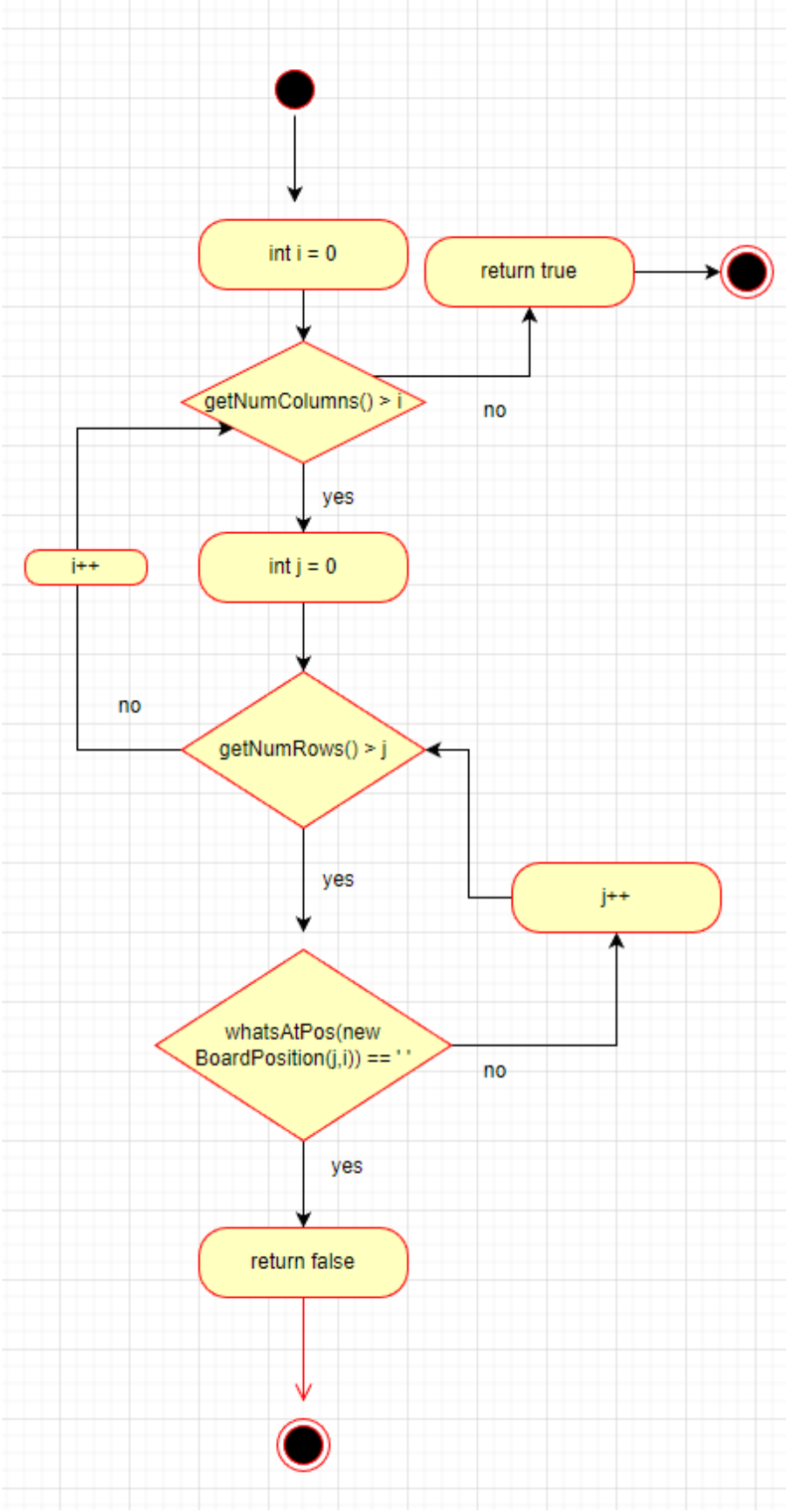
**make:** Compiles GameBoard, AbsGameBoard, IGameBoard, BoardPosition, and GameScreen

**make run:** Executes GameScreen.class

**make clean:** Deletes the GameScreen.class file, and deletes all class files in the models directoy

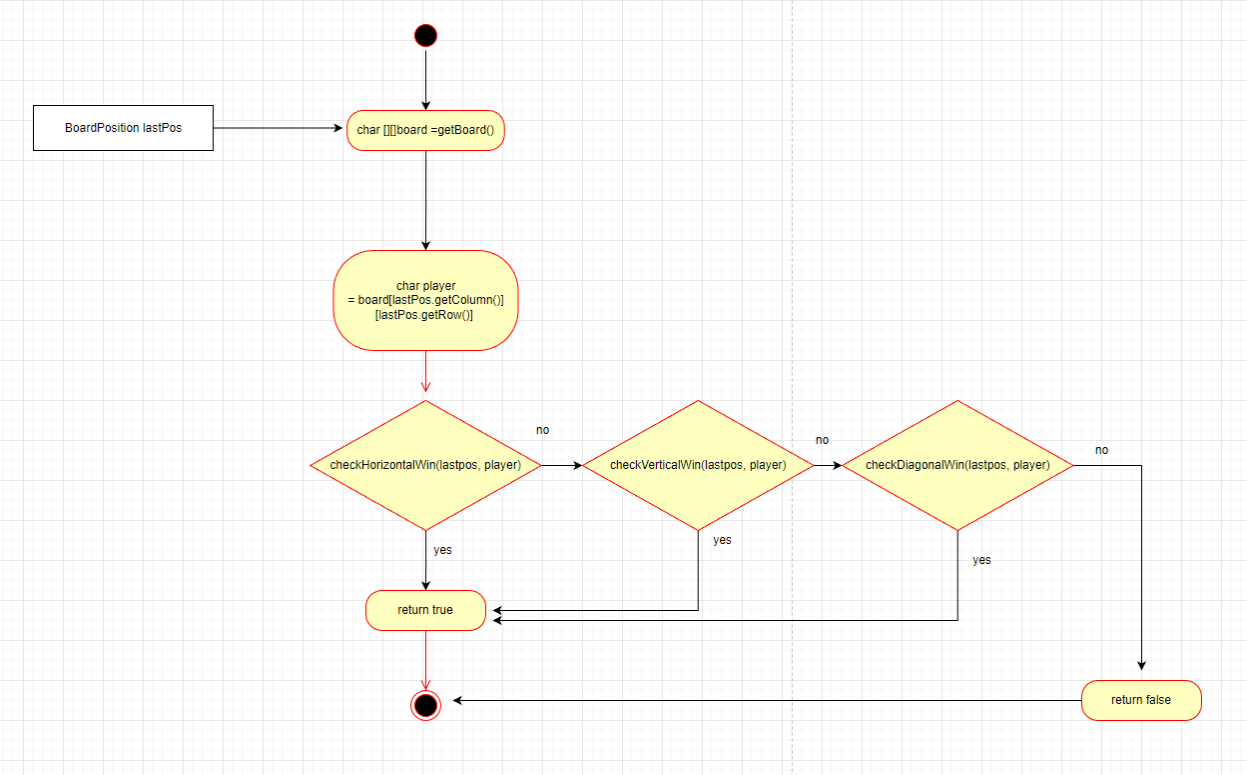
**make zip:** zips all files needed for project submission

**UML activity Diagrams:**

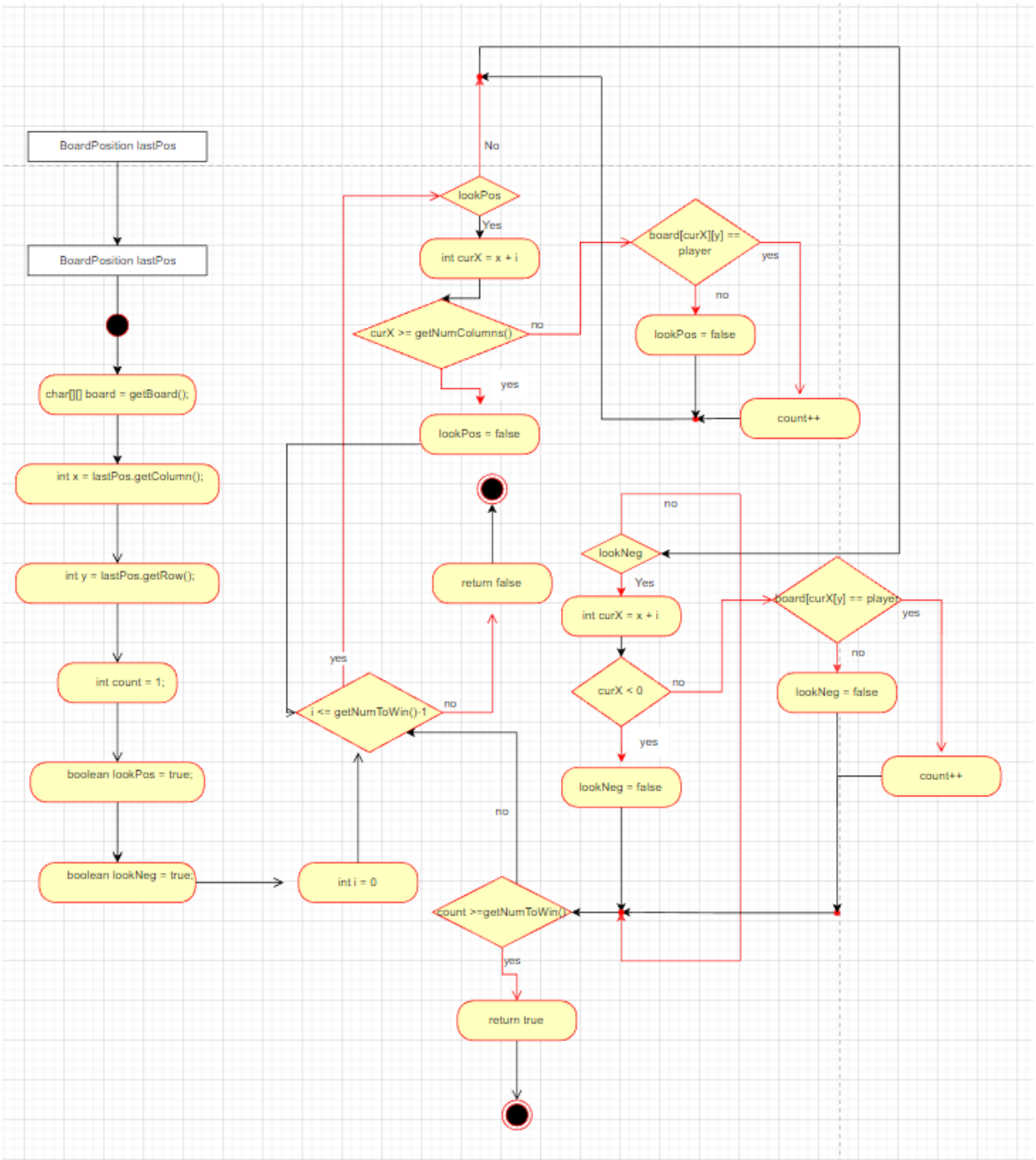



checkForDraw

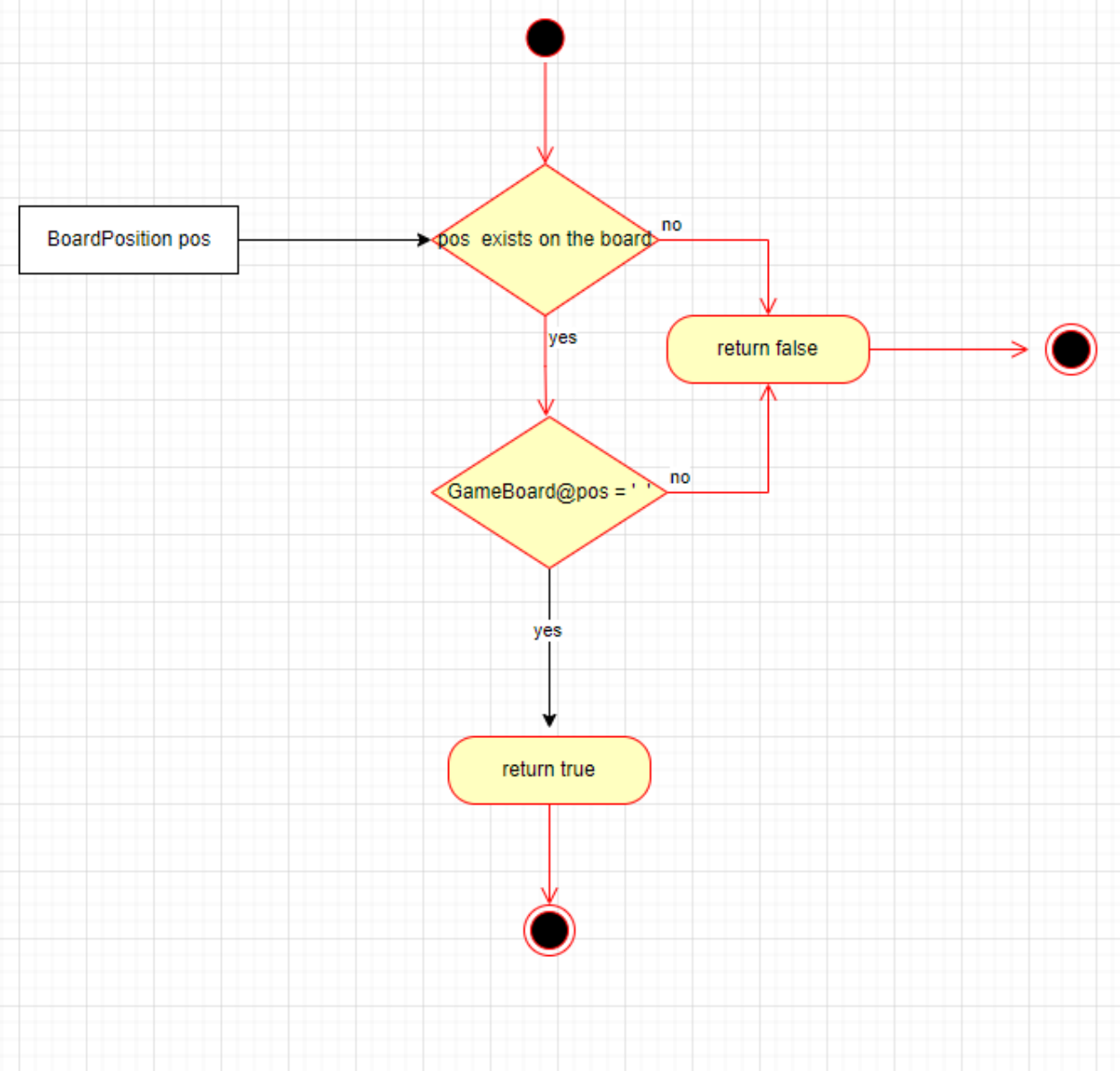
checkForWin



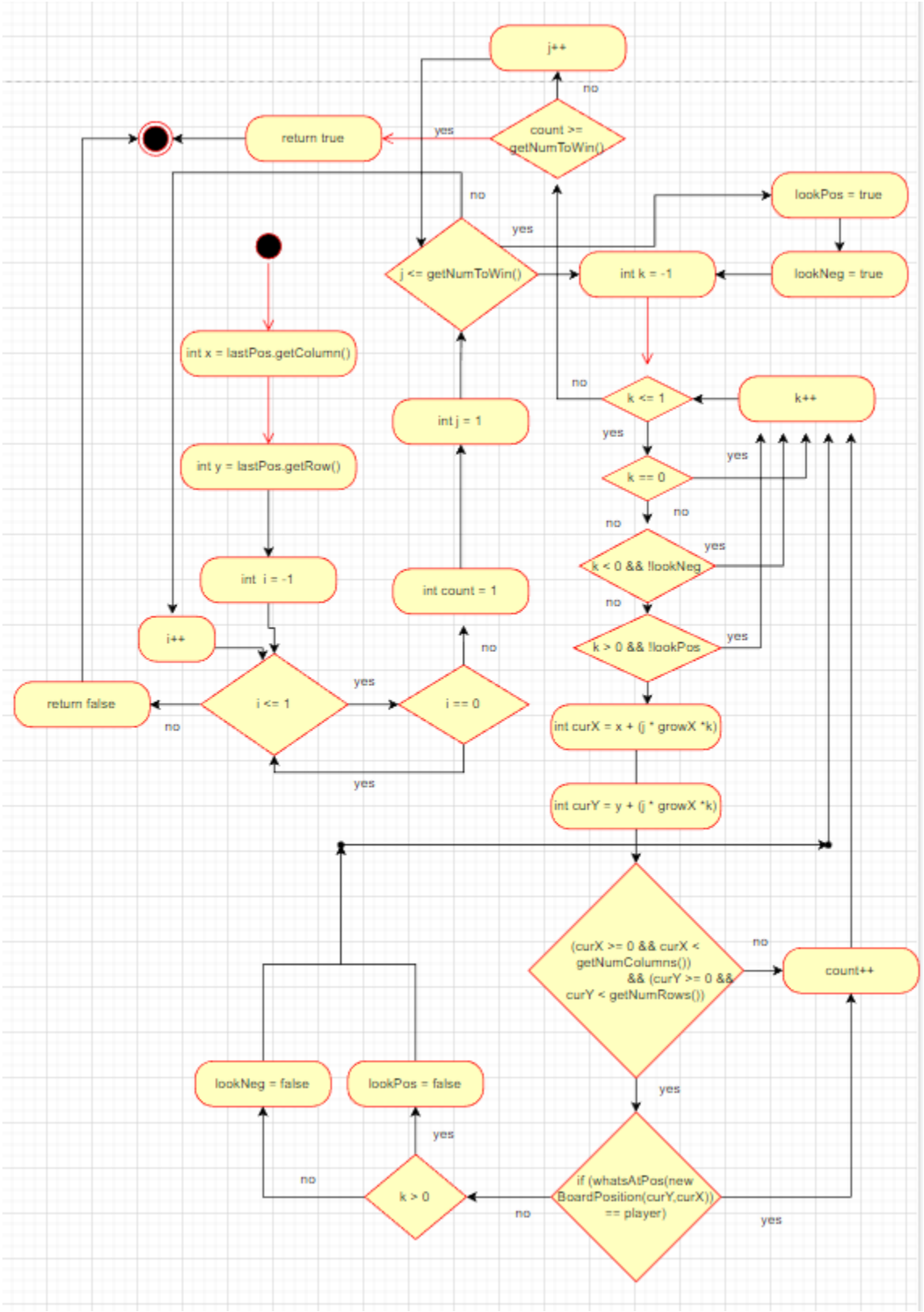
CheckHorizontalWin



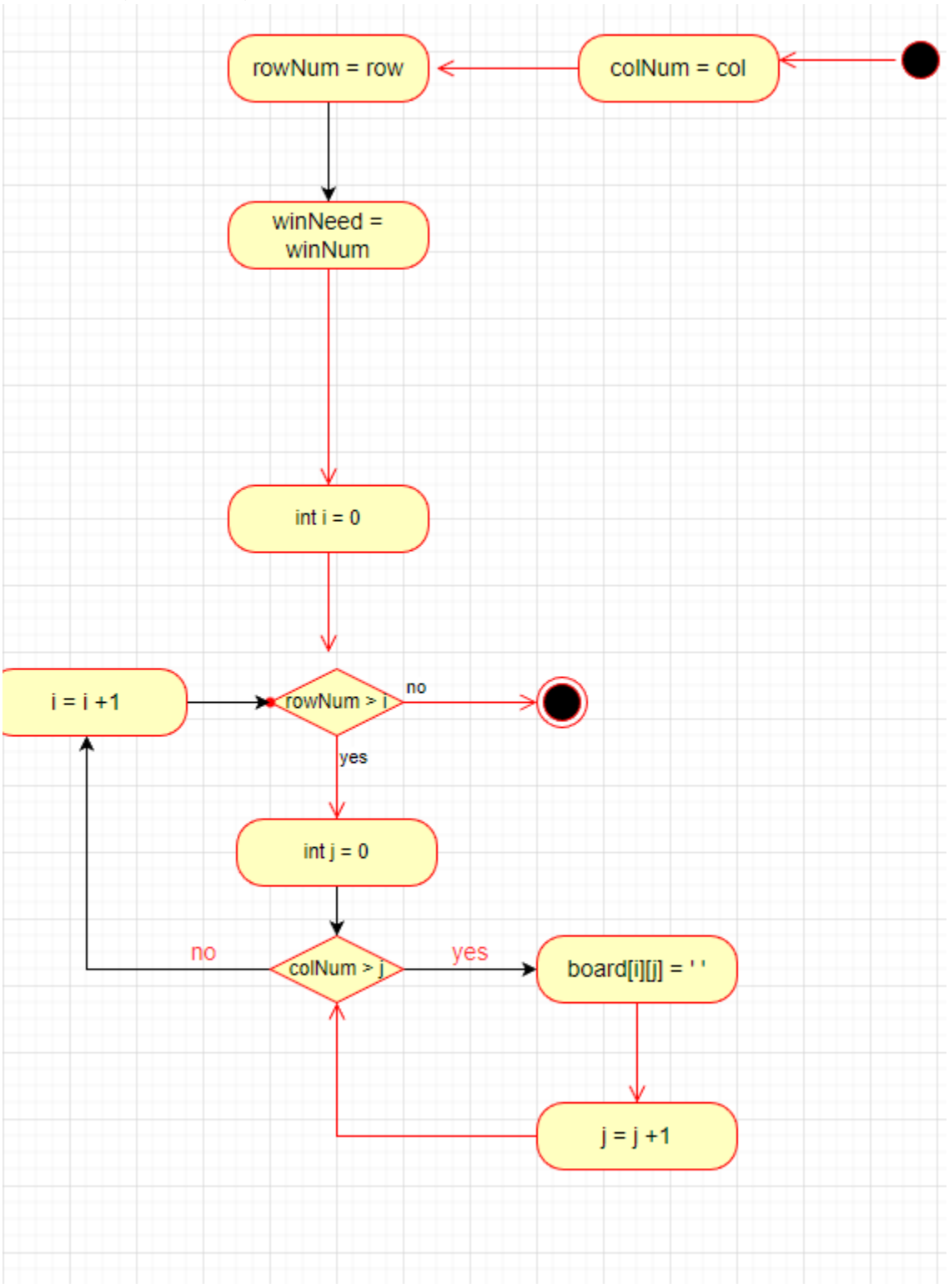
checkSpace


BoardPosition pos
pos exists on the board
no
yes
return false
GameBoard@pos = ' '
no
yes
return true

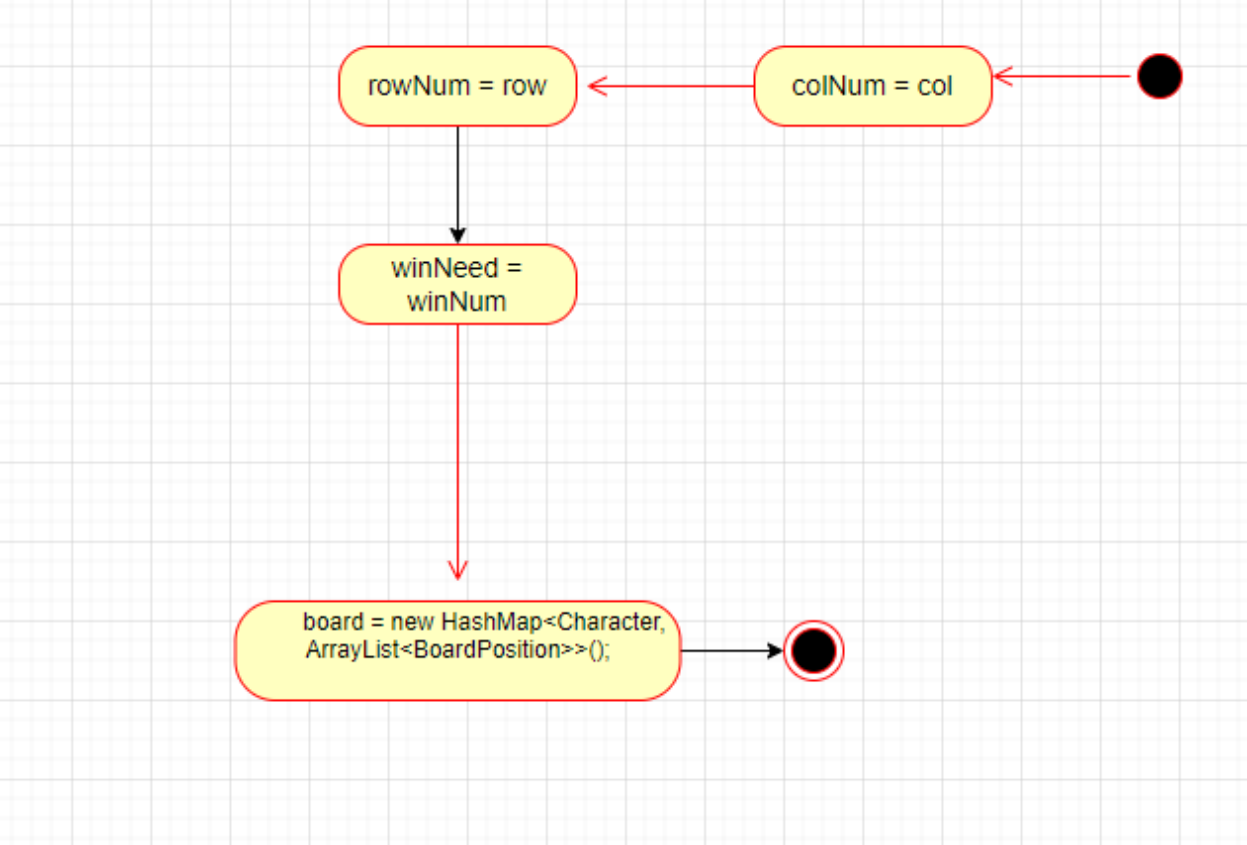
CheckDiagonalWin

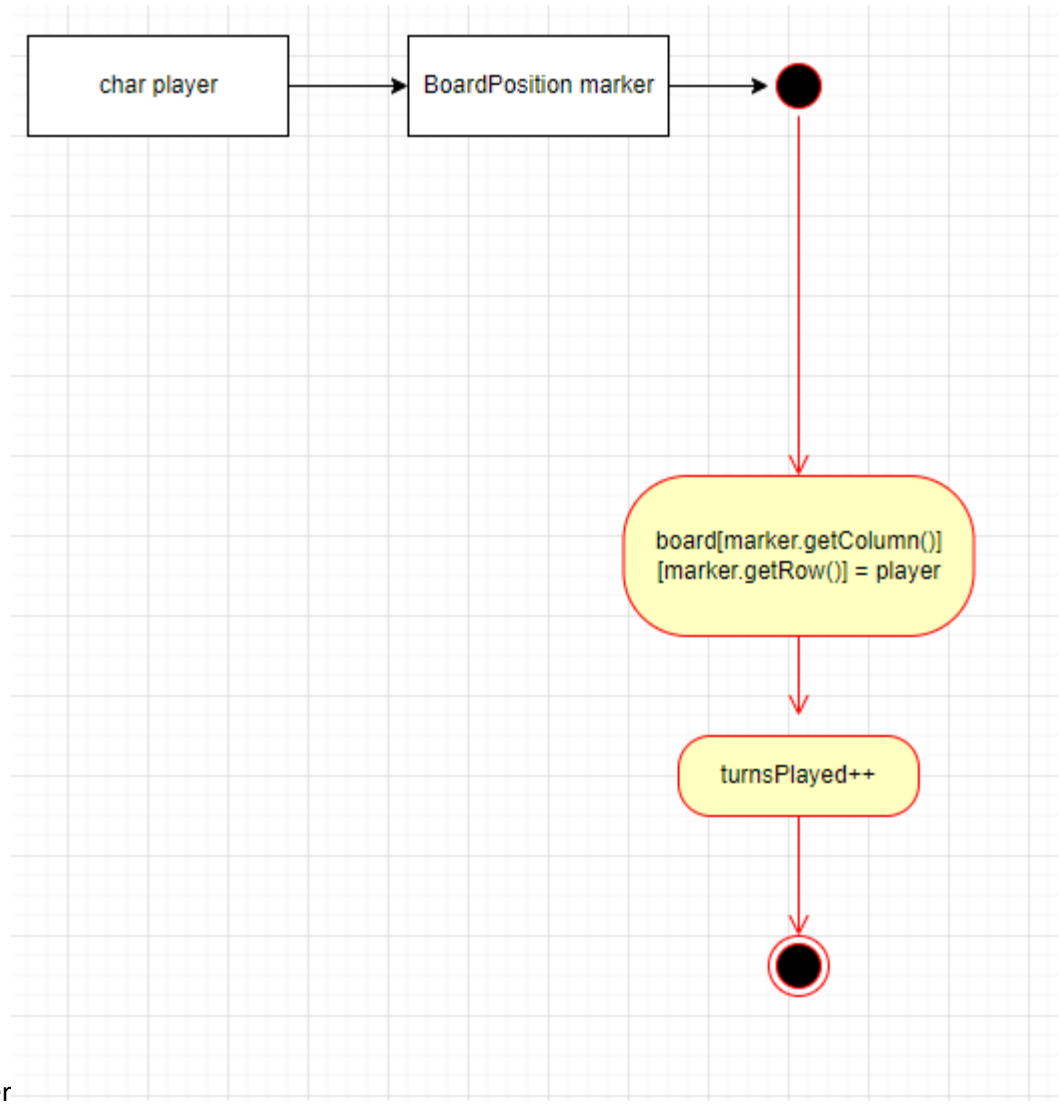


Gameboard(constructor)

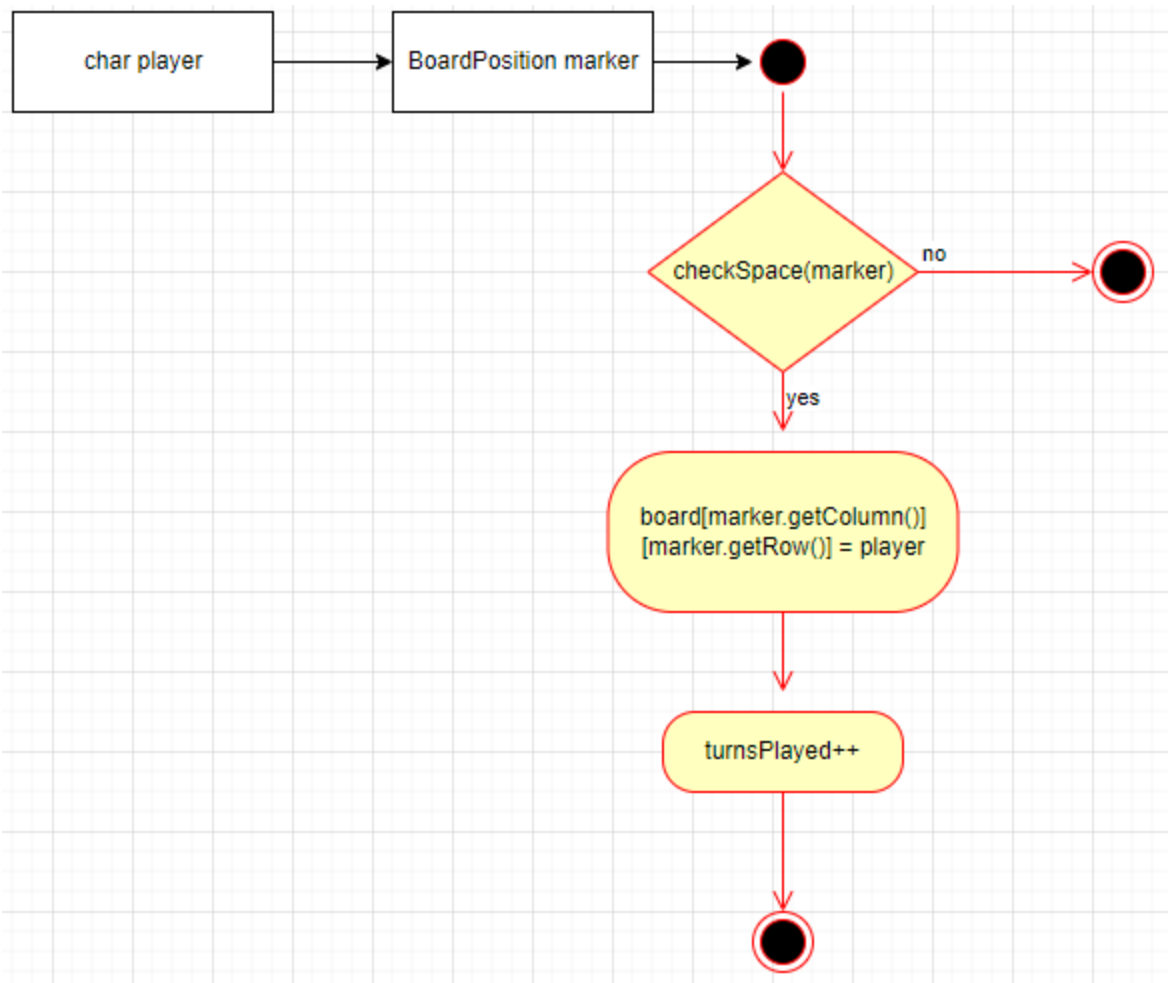


GameBoardMem(constructor)

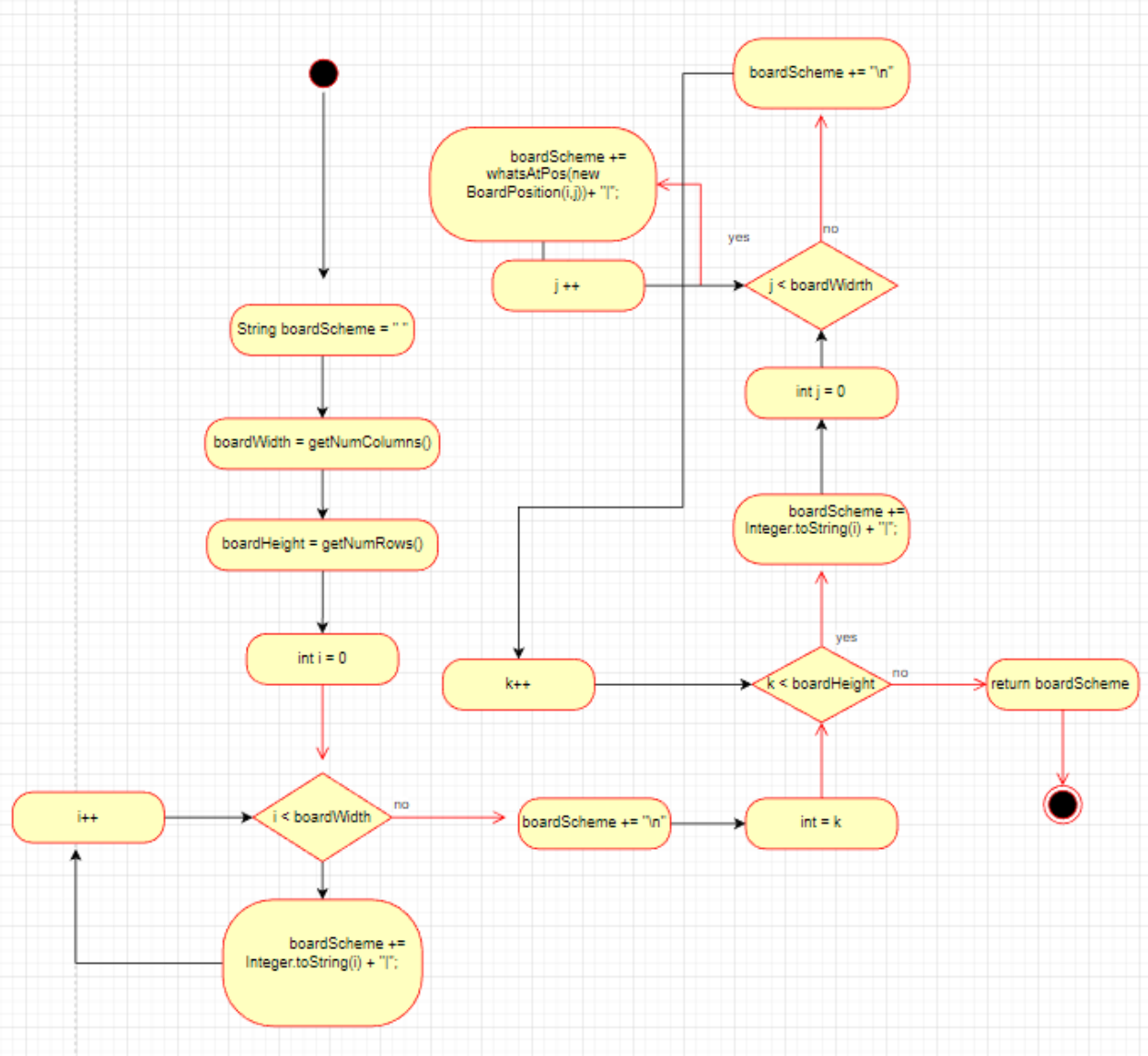




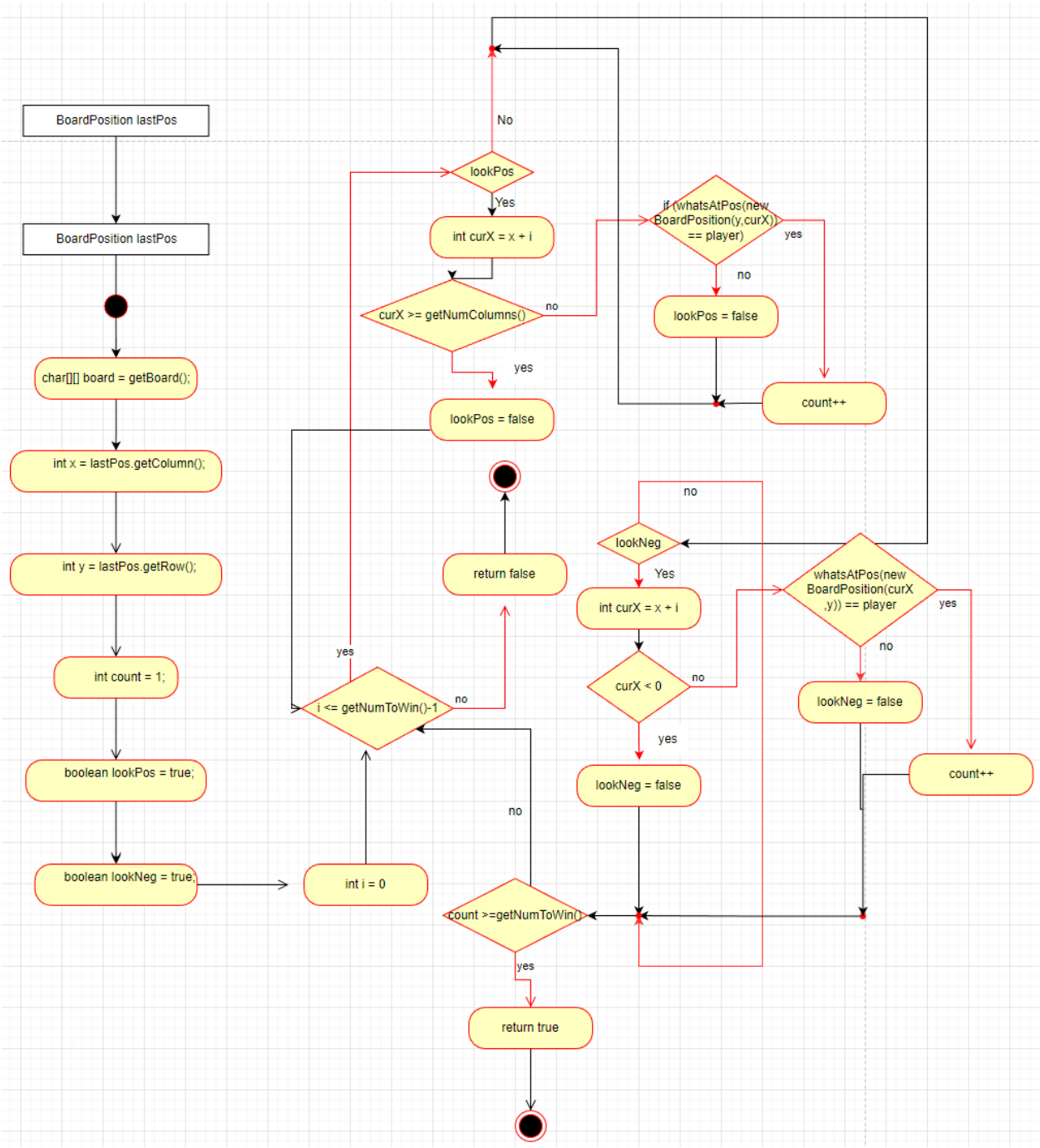

placeMarker

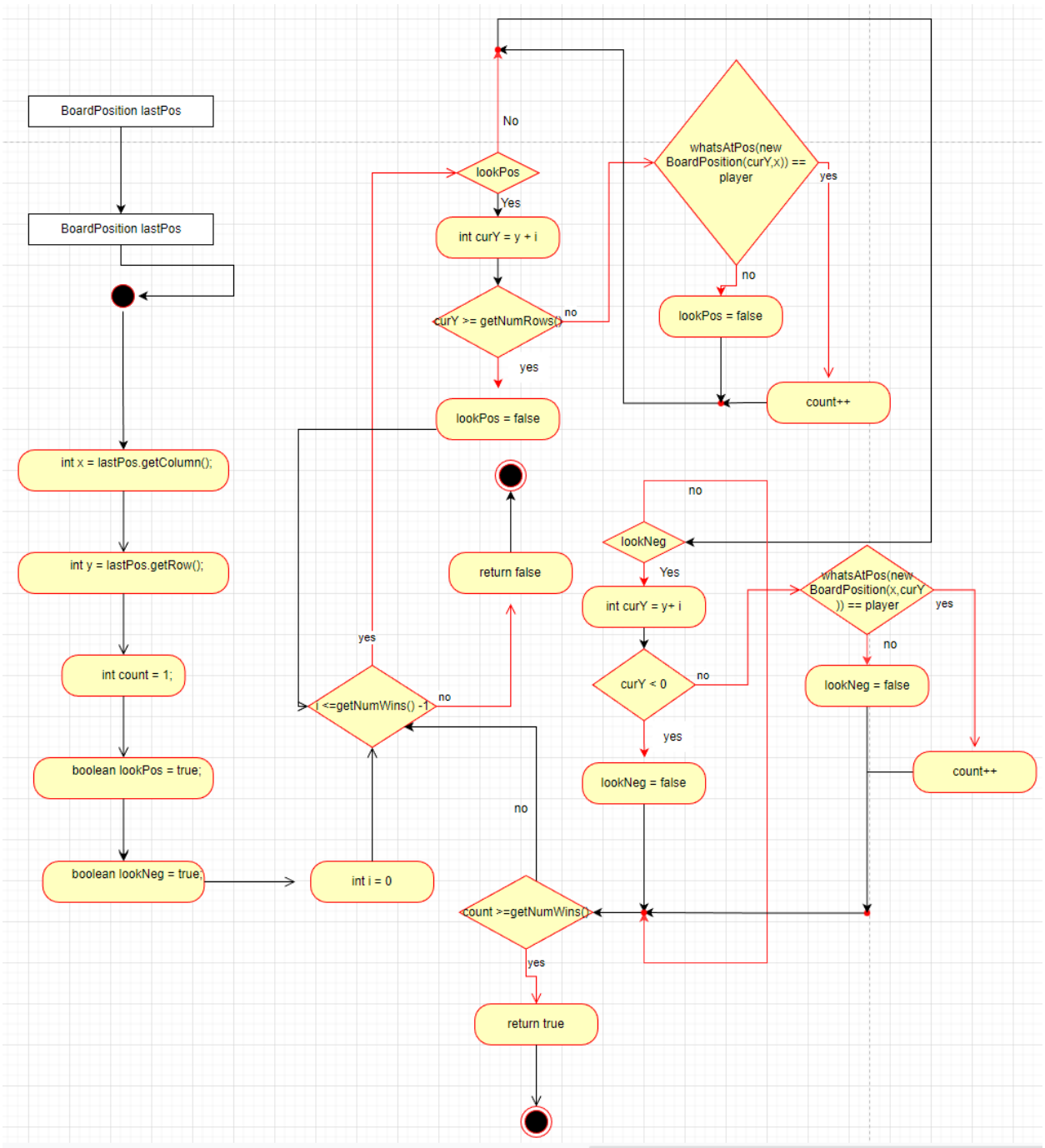

char player
BoardPosition marker
checkSpace(marker)
no
yes
board[marker.getColumn()]
[marker.getRow()] = player
turnsPlayed++

toString

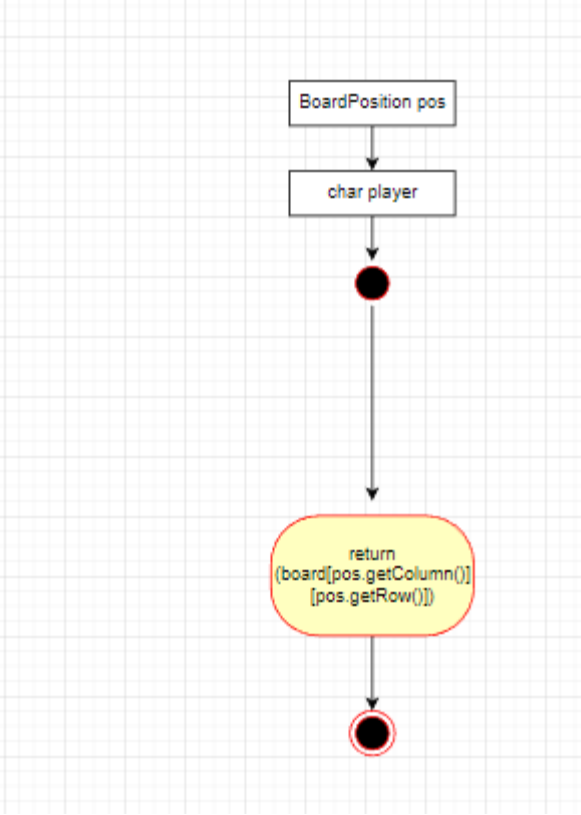

String boardScheme = " "
boardWidth = getNumColumns()
boardHeight = getNumRows()
int i = 0
i < boardWidth
yes
no
i++
boardScheme += Integer.toString(i) + "|";
boardScheme += "\n"
int = k
k < boardHeight
yes
no
k++
int j = 0
j < boardWidrth
yes
no
j ++
boardScheme += whatsAtPos(new BoardPosition(i,j))+ "|";
boardScheme += Integer.toString(i) + "|";
boardScheme += "\n"
return boardScheme

checkHorizontalWin



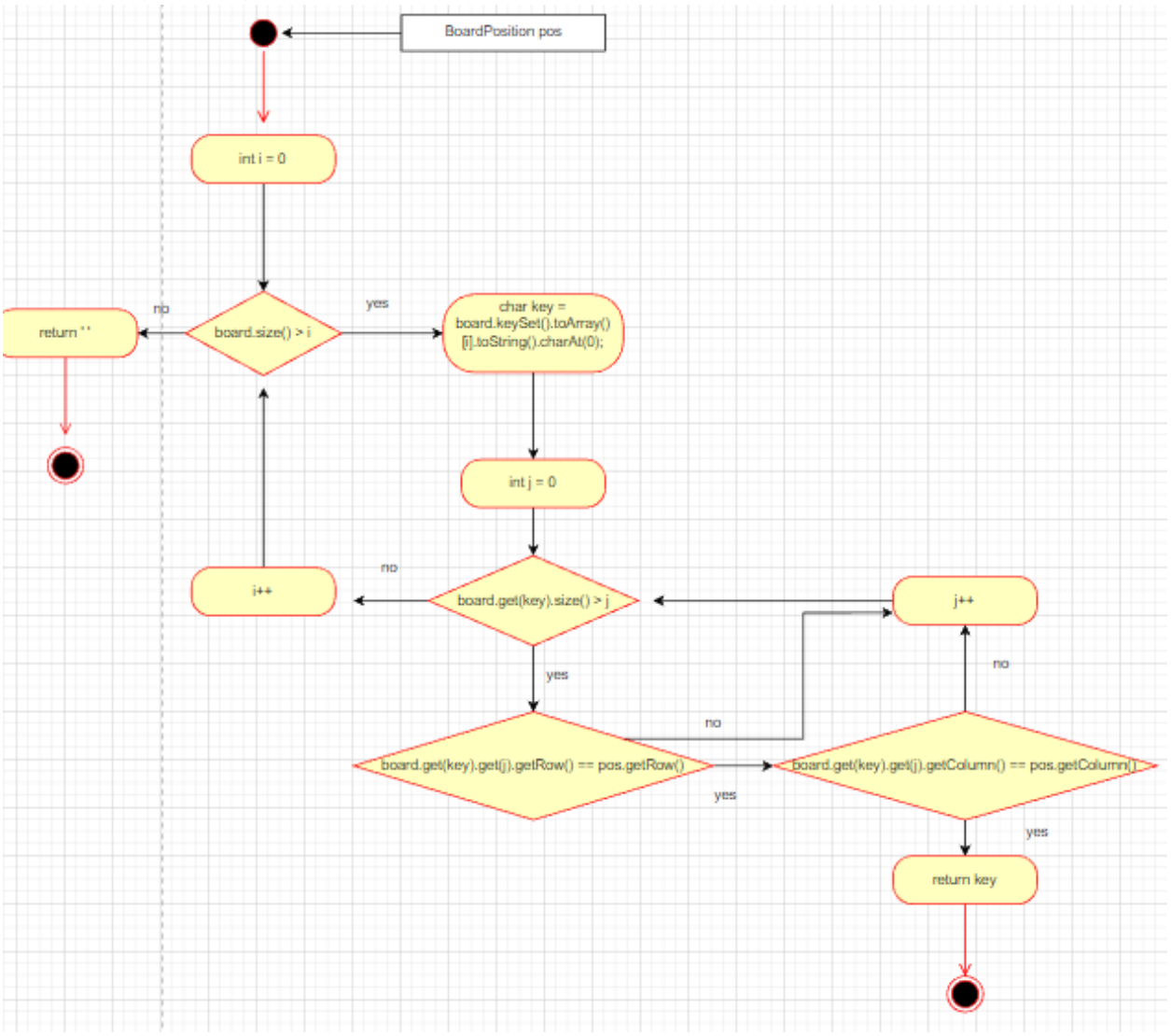


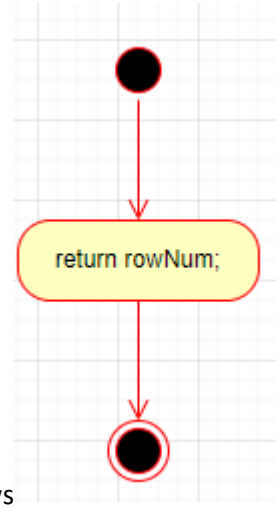

checkVerticalWin


BoardPosition pos
char player
return
(board[pos.getColumn()]
[pos.getRow()])

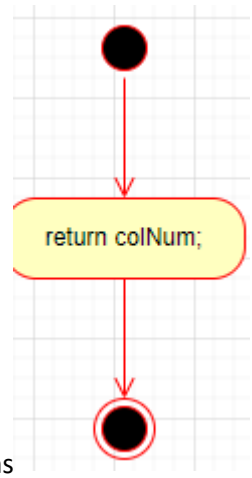
whatsAtPos

WhatsAtPos(for mem)



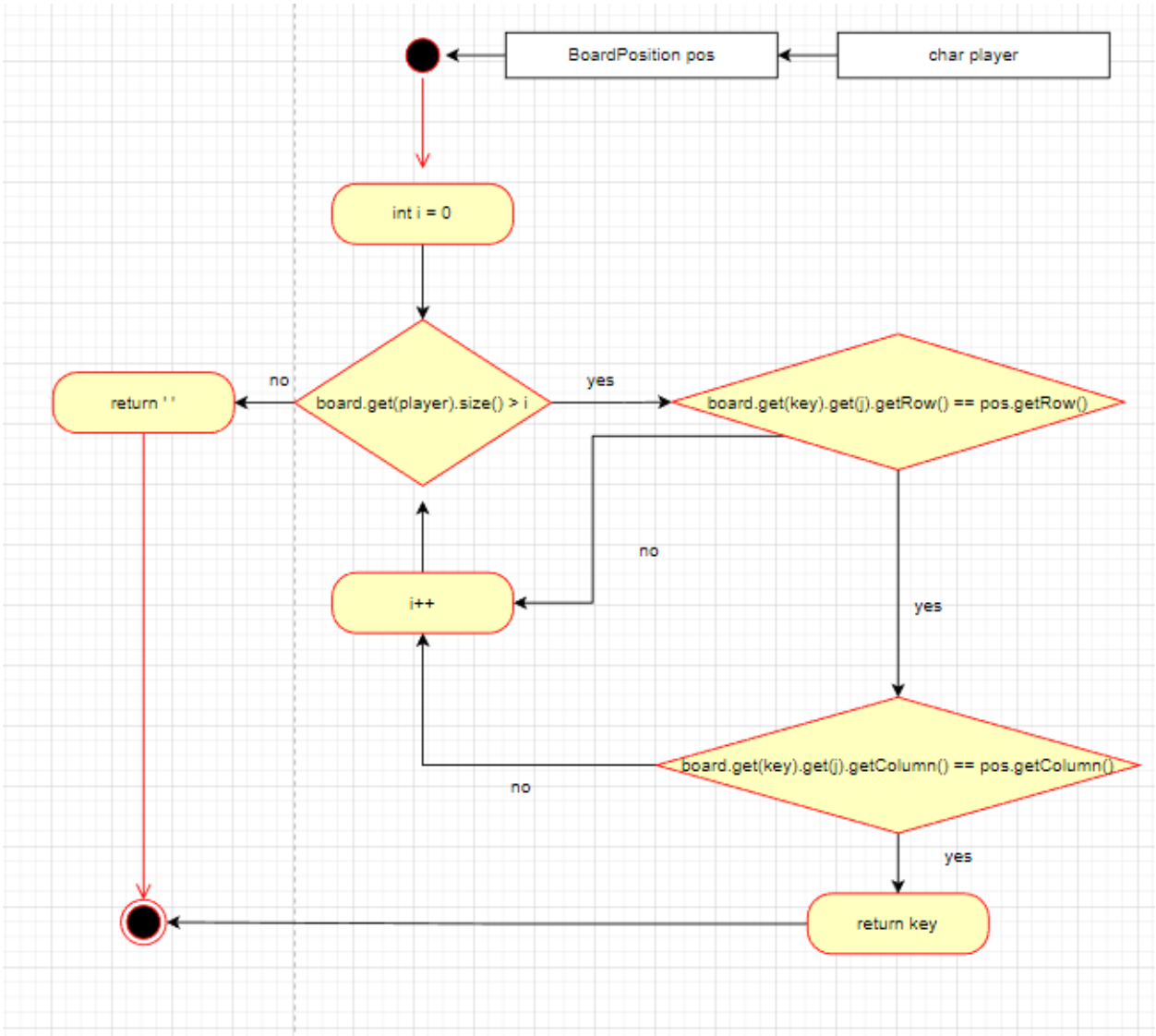



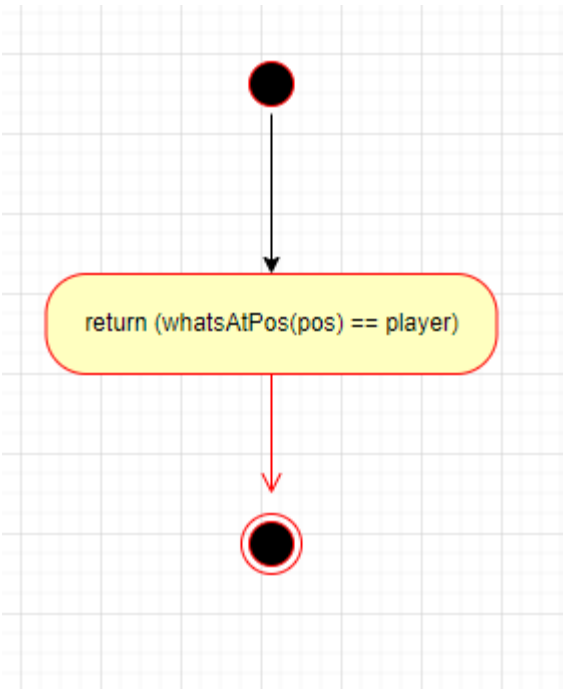

getNumRows



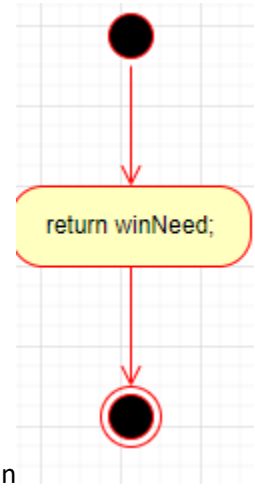

getNumColumns

isplayerAtPos (mem)



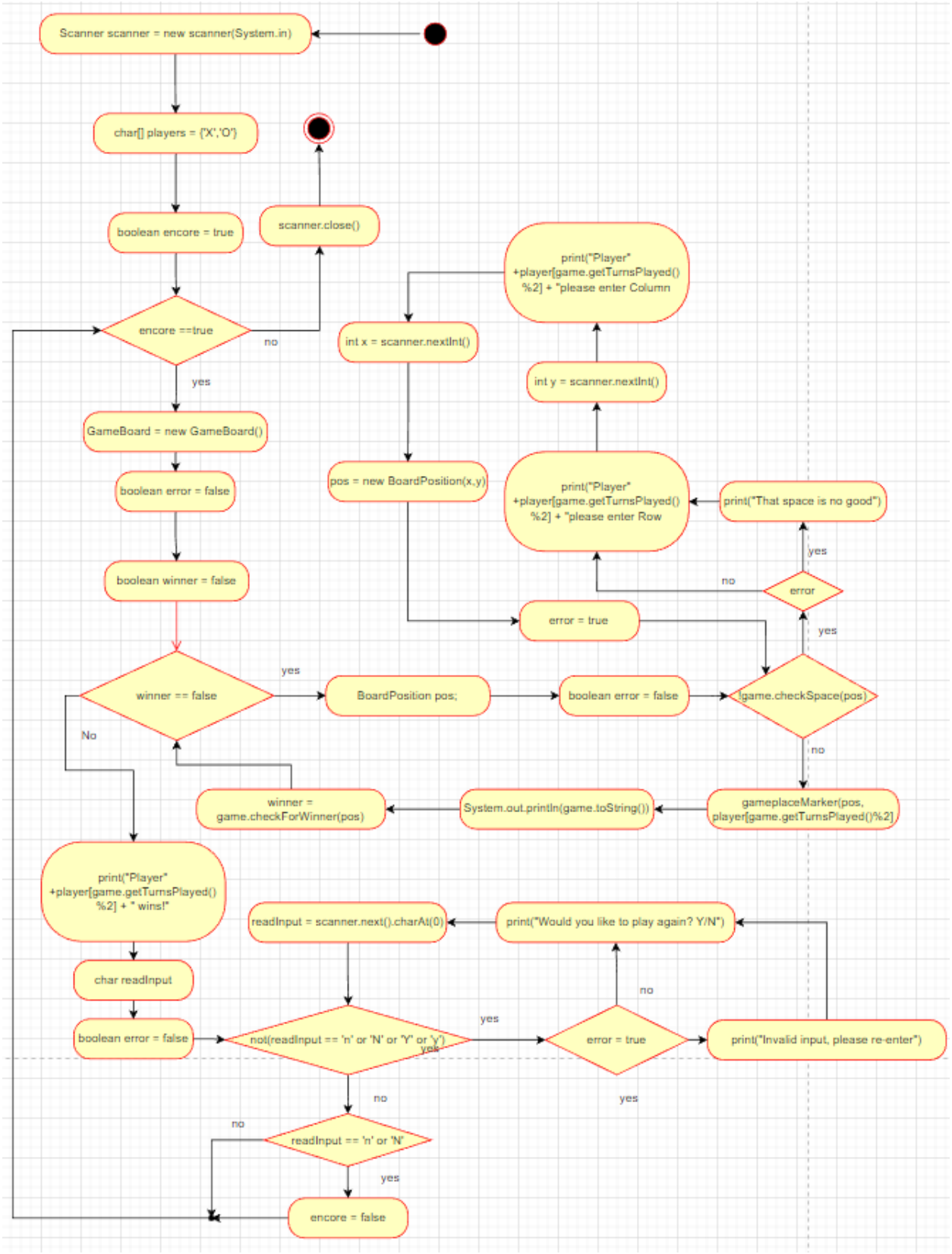


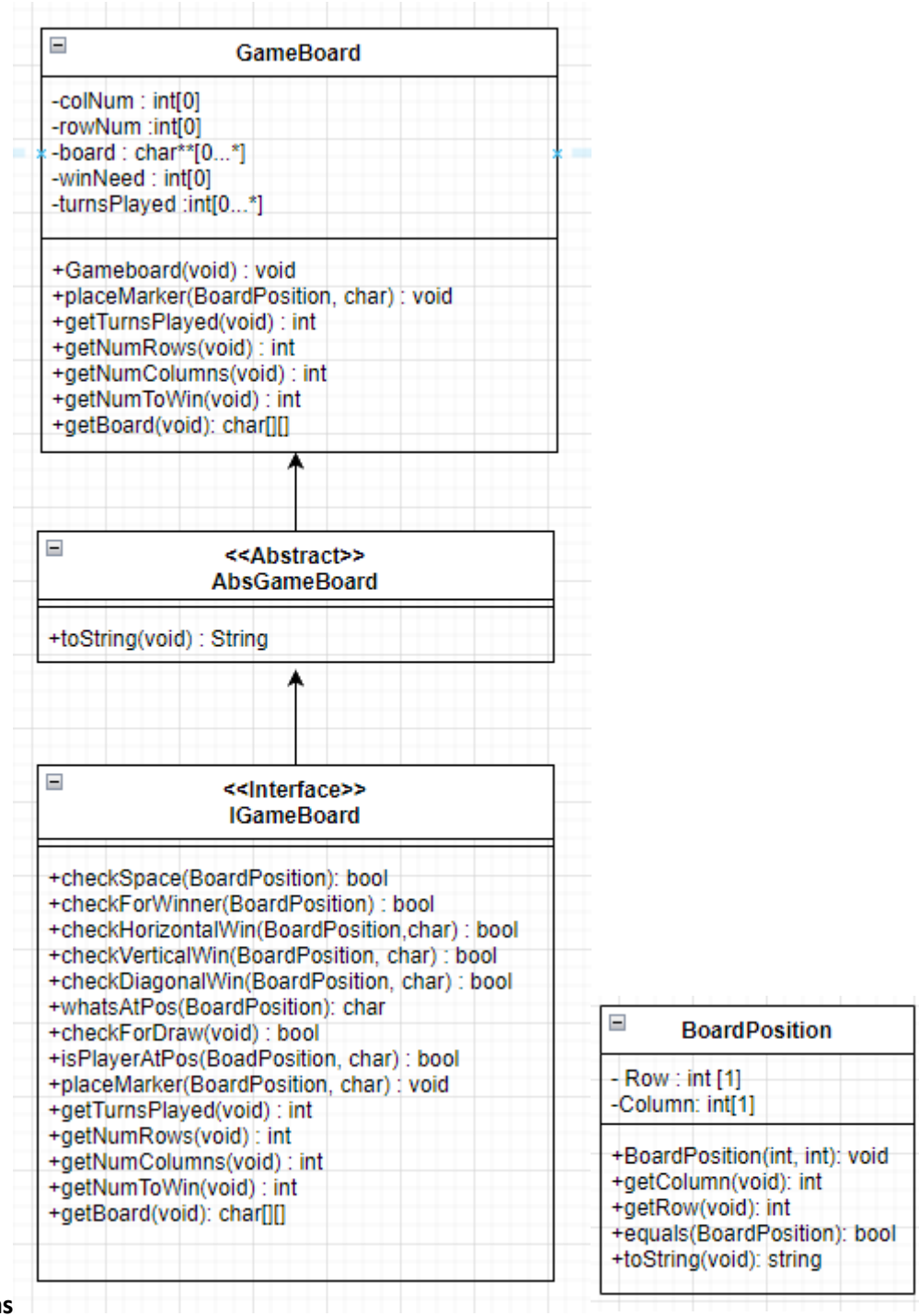

isPlayerAtPos




getNumToWin

main


Scanner scanner = new scanner(System.in)
char[] players = {'X','O'}
boolean encore = true
scanner.close()
encore ==true
no
yes
GameBoard = new GameBoard()
boolean error = false
boolean winner = false
winner == false
yes
No
print("Player" +player[game.getTurnsPlayed() %2] + " wins!"
char readInput
boolean error = false
int x = scanner.nextInt()
print("Player" +player[game.getTurnsPlayed() %2] + "please enter Column
int y = scanner.nextInt()
pos = new BoardPosition(x,y)
print("Player" +player[game.getTurnsPlayed() %2] + "please enter Row
print("That space is no good")
error
yes
no
error = true
BoardPosition pos;
boolean error = false
!game.checkSpace(pos)
yes
no
gameplaceMarker(pos, player[game.getTurnsPlayed()%2]
System.out.println(game.toString())
winner = game.checkForWinner(pos)
readInput = scanner.next().charAt(0)
print("Would you like to play again? Y/N")
not(readInput == 'n' or 'N' or 'Y' or 'y')
yes
error = true
print("Invalid input, please re-enter")
no
yes
readInput == 'n' or 'N'
no
yes
encore = false




**UML Class Diagrams**