CPSC 2150 Project Two 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 the marker type start with a value of X at the start of a game for consistent playthroughs.
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 I need indication of inappropriate input so that I may correct the mistake.
5. As a player I need an updating display of the board so I may watch the game progress.
6. As a player I need to know when win conditions have been satisfied so the game may conclude.
7. As a player I need to know when the game has been drawn so I may be aware to restart the game.
8. As a player I need to be prompted after concluding to game to begin another, so I don't need to rerun the application.
9. As a player I need the ability to place and store a market at desired positions to play the game.
10. As a player I need the board to be designed as a 5 X 8 grid to play a specific variety of tic-tac-toe.
11. As a player I need the win condition to be a line of 5 adjacent similar markings so to play a specific variety of tic-tac-toe.
12. As a player I need the gameboard display to be expressed in a readable manner so I may bear greater witness.
13. As a player, the unmarked locations of the board should possess the default value, ' ' so it may be clear where available places positions are.
14. As a player, the placed marker value should be either 'X', or 'O', so player progress may be clearly seen.

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, GameScreen, and BoardPosition.
5. The class GameBoard must exclusively use methods prescribed within assignment documentation.
6. Attributes of the class, BoardPosition, must exclusively be accessible by getter methods.
7. All U/I interaction will be exclusively preformed within the GameScreen method.
8. BoardPosition will have only one constructor method.
9. BoardPosition attributes may only be set within the constructor.
10. BoardPosition must have a methods overriding the equals() and toString() methods.
11. All attributes of GameBoard must be private unless they are static and final.
12. Gameboard is of size 5 X 8
    - 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.
13. X always goes first
14. Five in a row is the win condition
15. GameBoard will extend AbsGameboard.
16. AbsGameBoard implements IGameBoard.
17. Board element (0,0) should be top position of the board
18. No dead code should be present in the project
19. Makefile should have targets : default, run, and clean

makefile instructions:

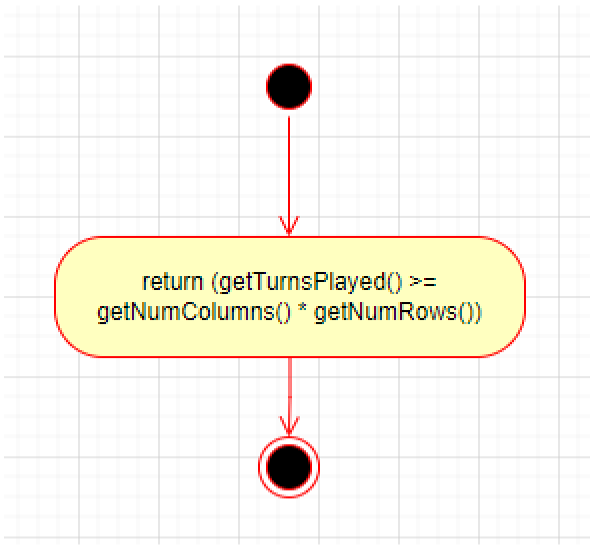
make: Compiles GameBoard, AbsGameBoard, IGameBoard, BoardPosition, and GameScreen

make run: Executes GameScreen.class

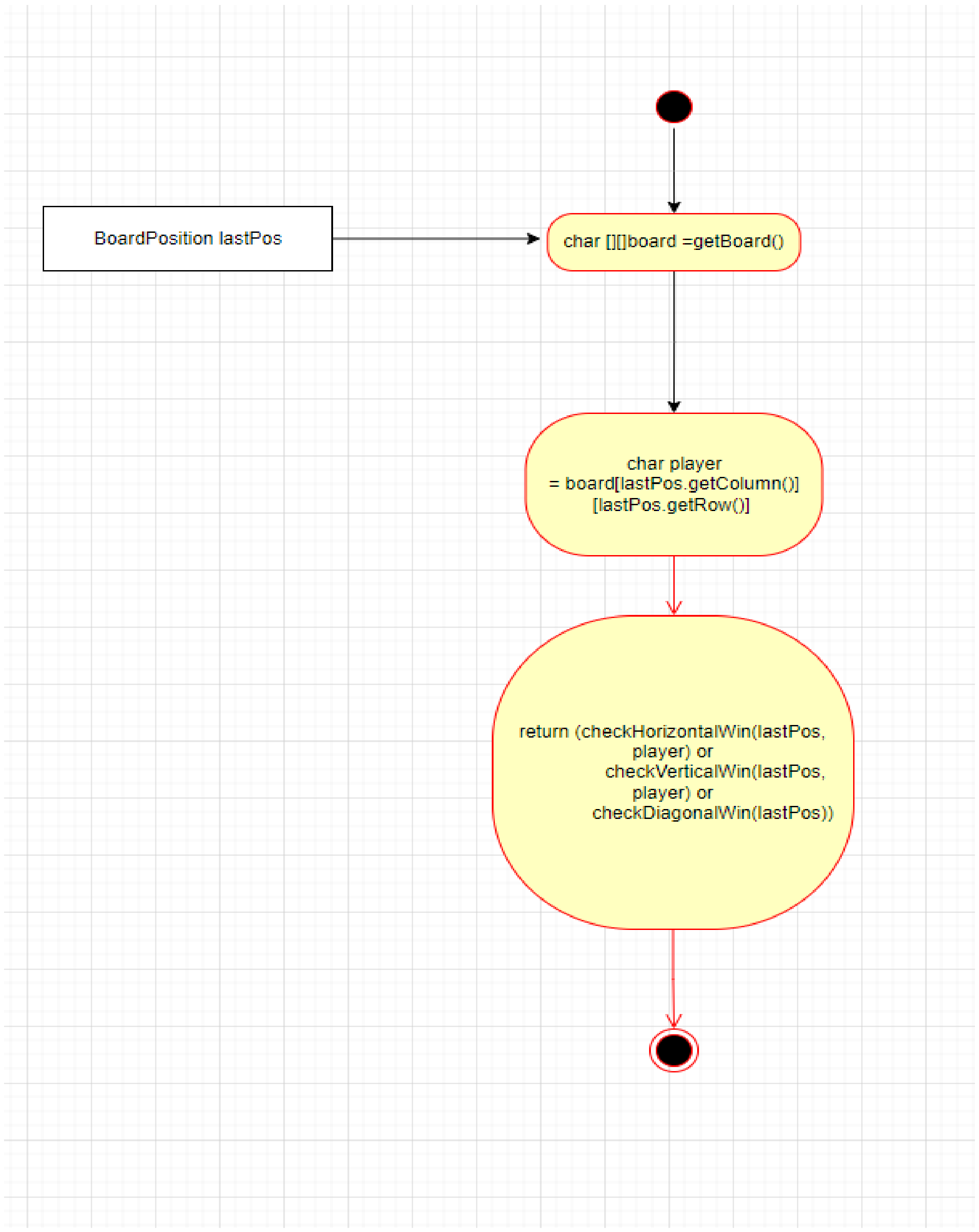
make clean: Deletes the GameScreen.class file, and deletes all class files in the models directoy

UML activity Diagrams:

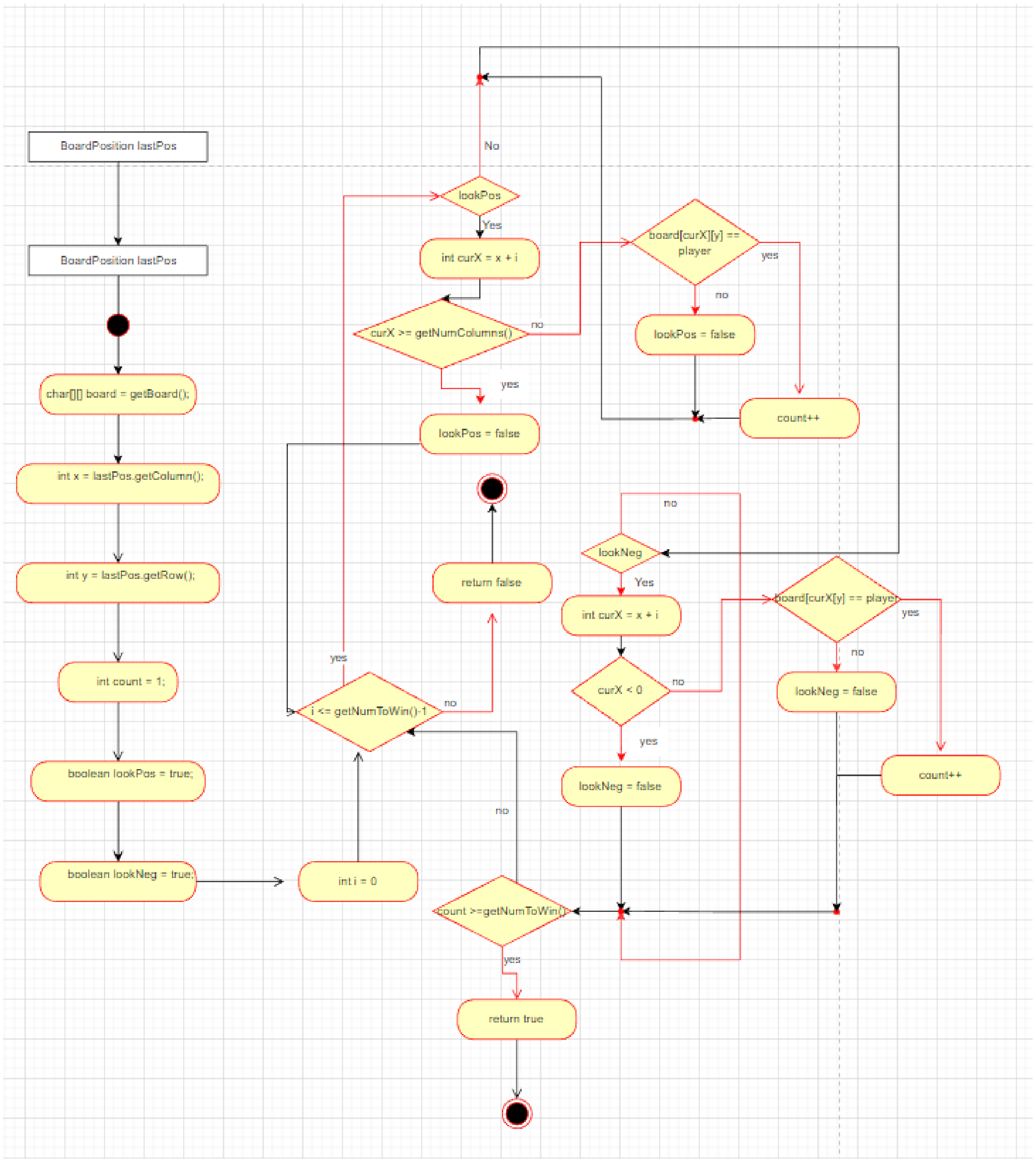
checkForDraw



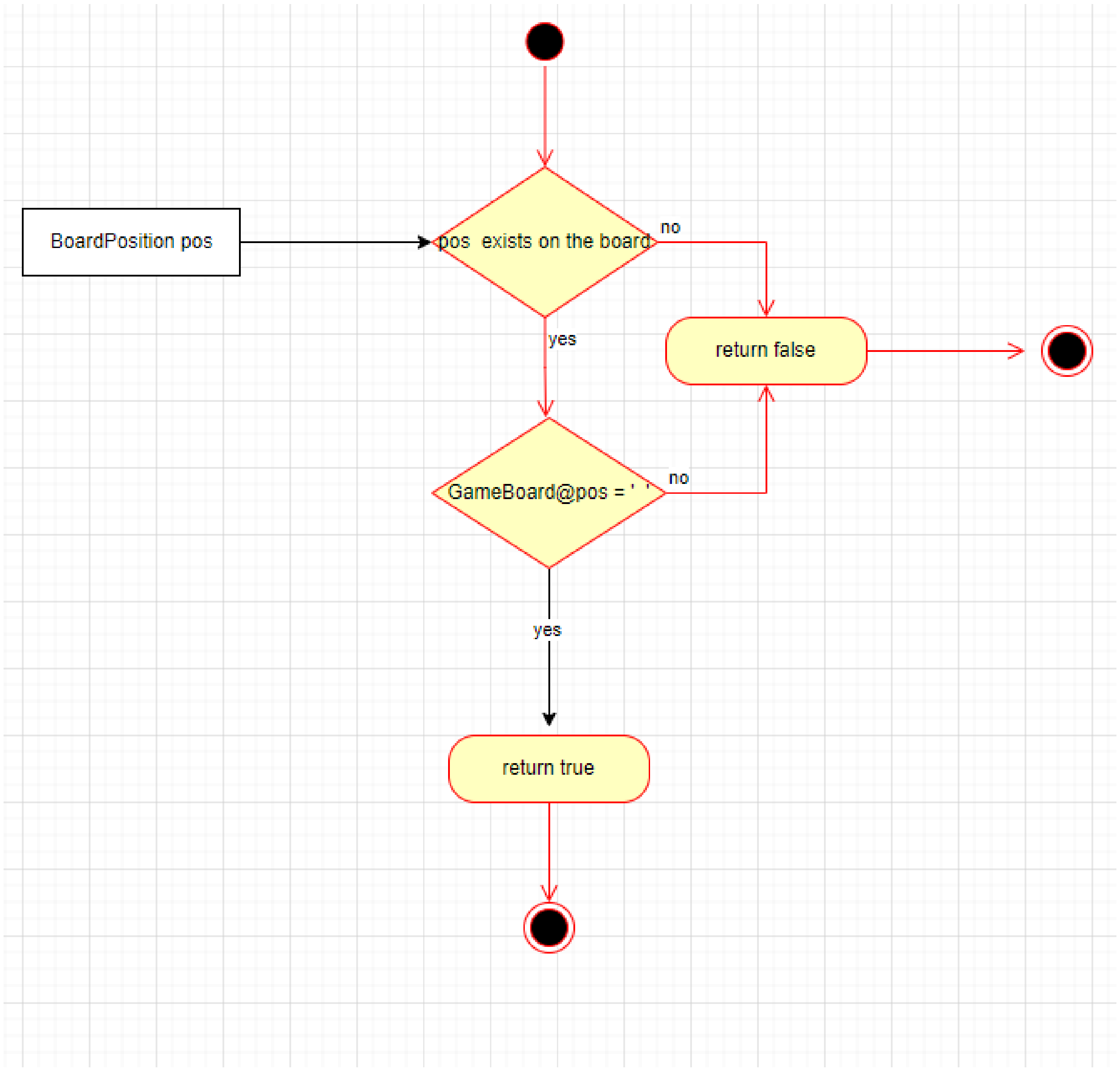
checkForWin


BoardPosition lastPos
char [][]board =getBoard()
char player = board[lastPos.getColumn()][lastPos.getRow()]
return (checkHorizontalWin(lastPos, player) or checkVerticalWin(lastPos, player) or checkDiagonalWin(lastPos))

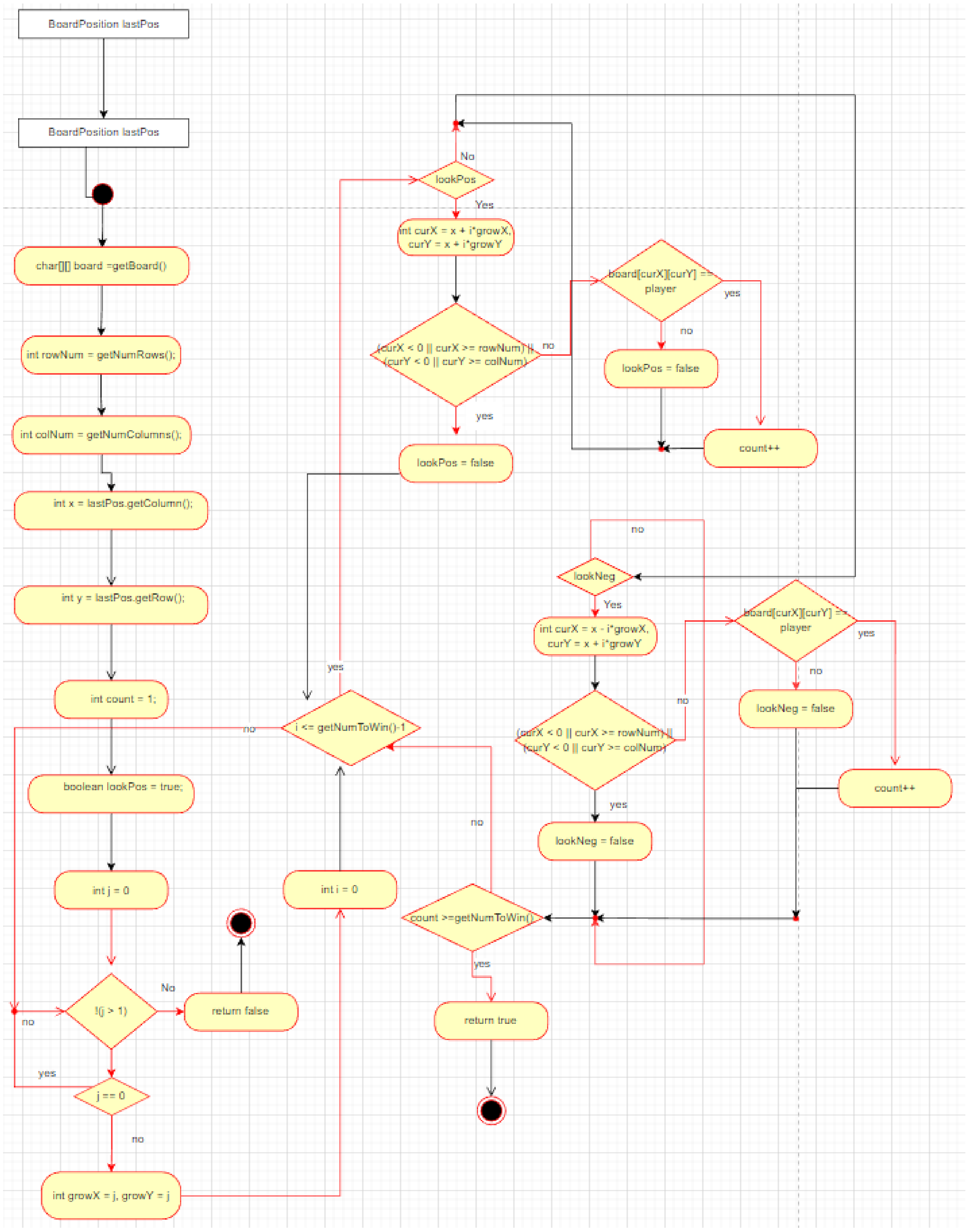
CheckHorizontalWin



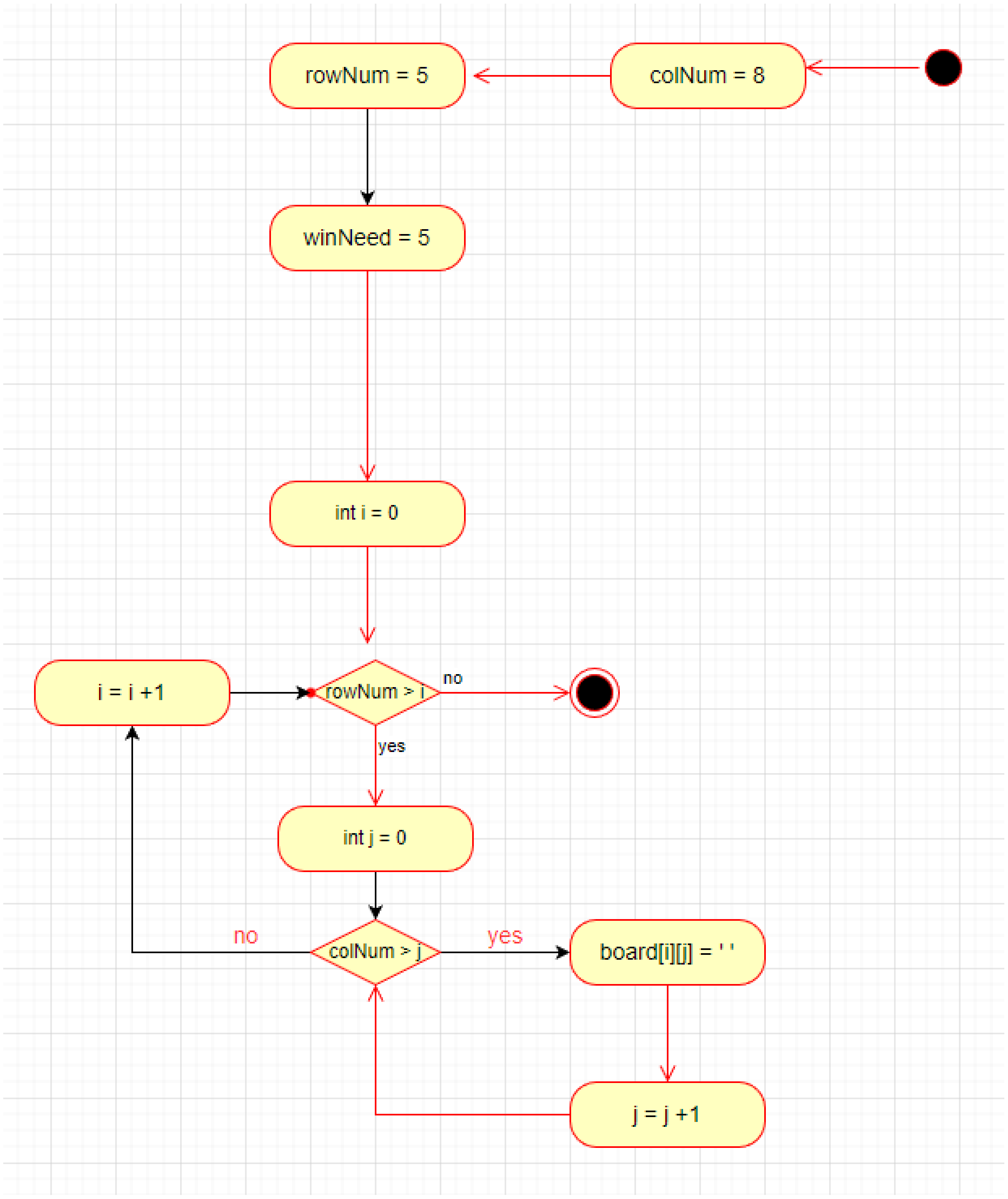
checkSpace


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

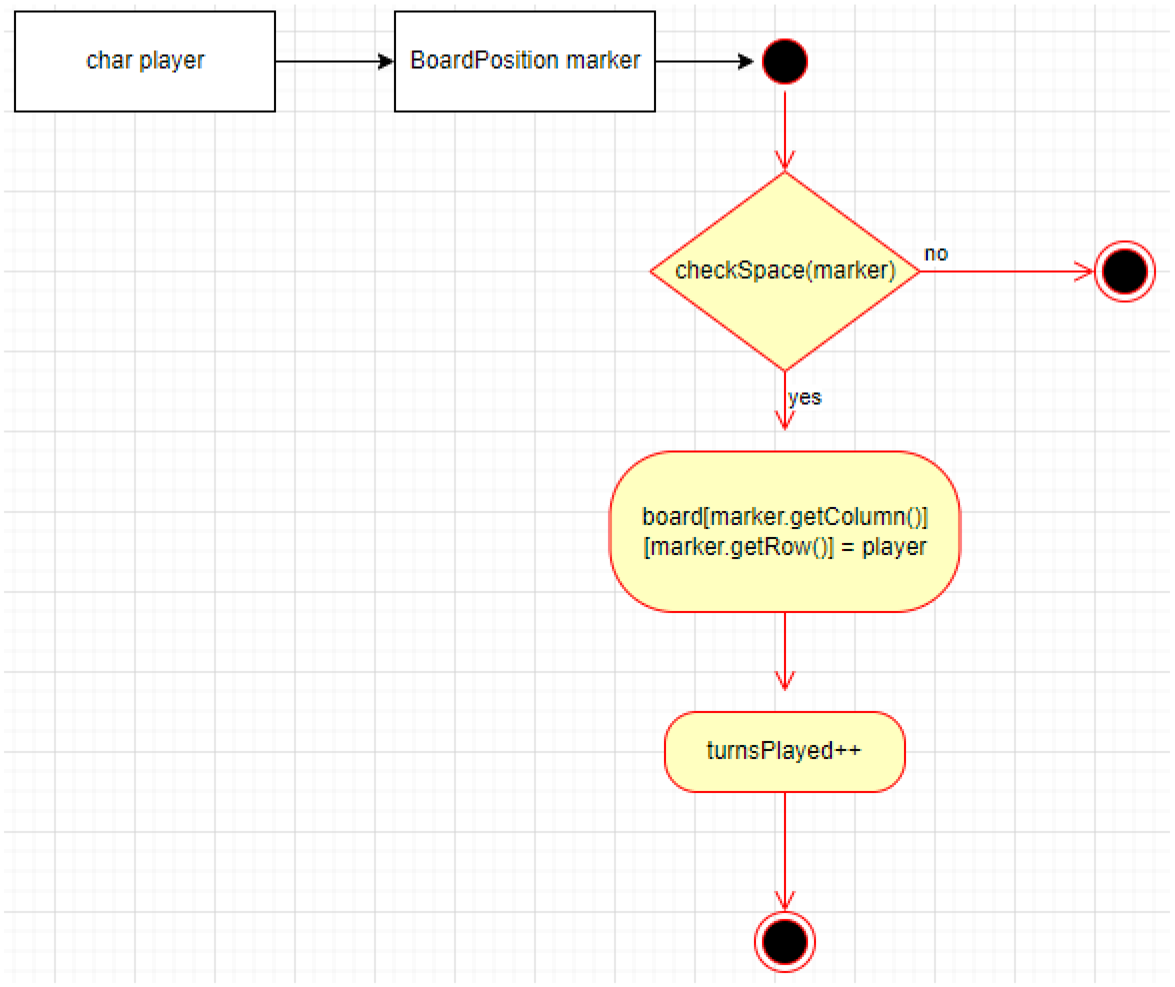
CheckDiagonalWin


BoardPosition lastPos
BoardPosition lastPos
char[][] board =getBoard()
int rowNum = getNumRows();
int colNum = getNumColumns();
int x = lastPos.getColumn();
int y = lastPos.getRow();
int count = 1;
boolean lookPos = true;
int j = 0
!(j > 1)
No
return false
no
yes
j == 0
no
int growX = j, growY = j
int i = 0
i <= getNumToWin()-1
yes
no
lookPos
No
Yes
int curX = x + i*growX, curY = x + i*growY
(curX < 0 || curX >= rowNum) || (curY < 0 || curY >= colNum)
no
yes
lookPos = false
board[curX][curY] == player
yes
no
lookPos = false
count++
lookNeg
no
Yes
int curX = x - i*growX, curY = x + i*growY
(curX < 0 || curX >= rowNum) || (curY < 0 || curY >= colNum)
no
yes
lookNeg = false
board[curX][curY] == player
yes
no
lookNeg = false
count++
count >=getNumToWin()
no
yes
return true

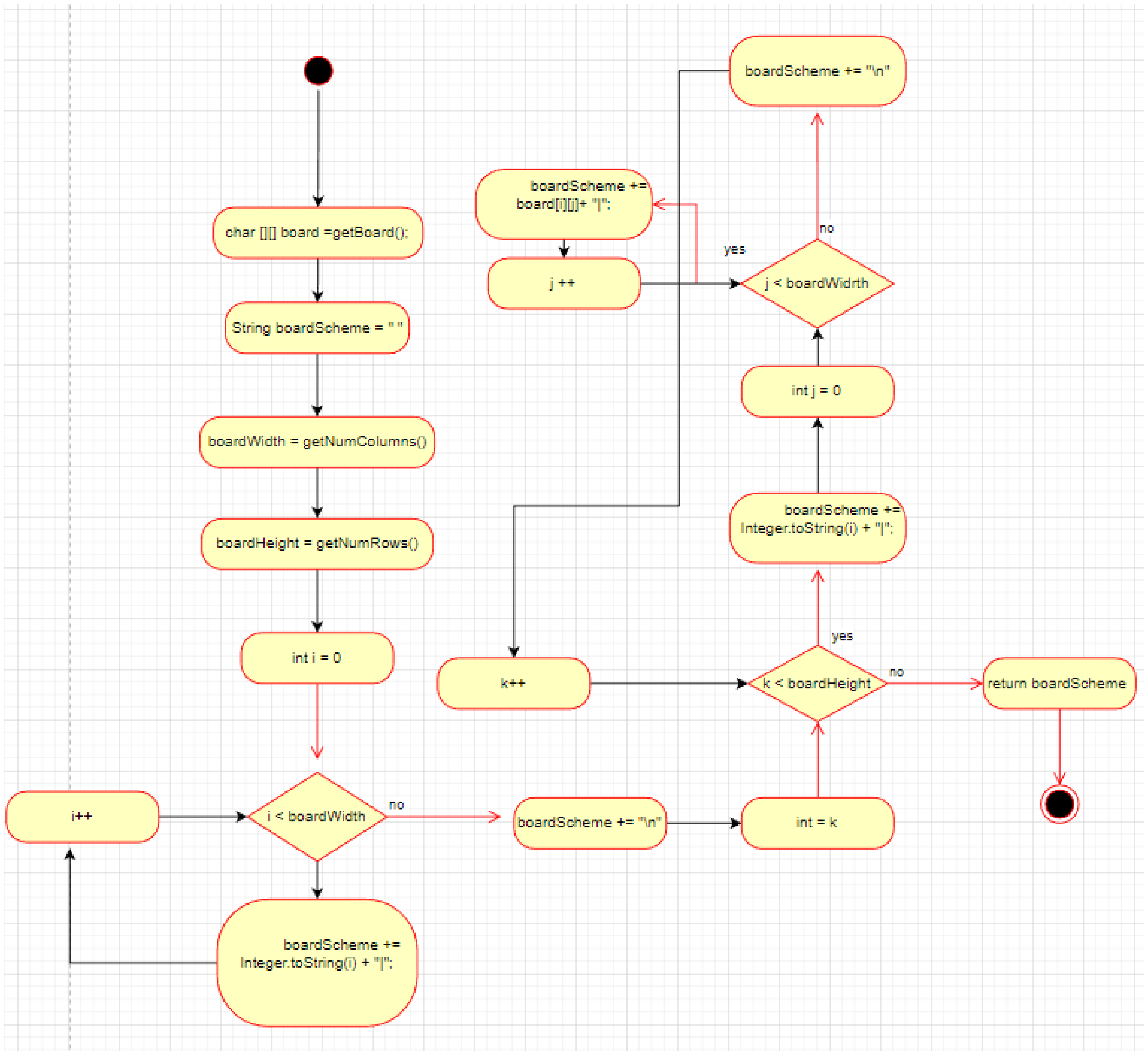
Gameboard(constructor)



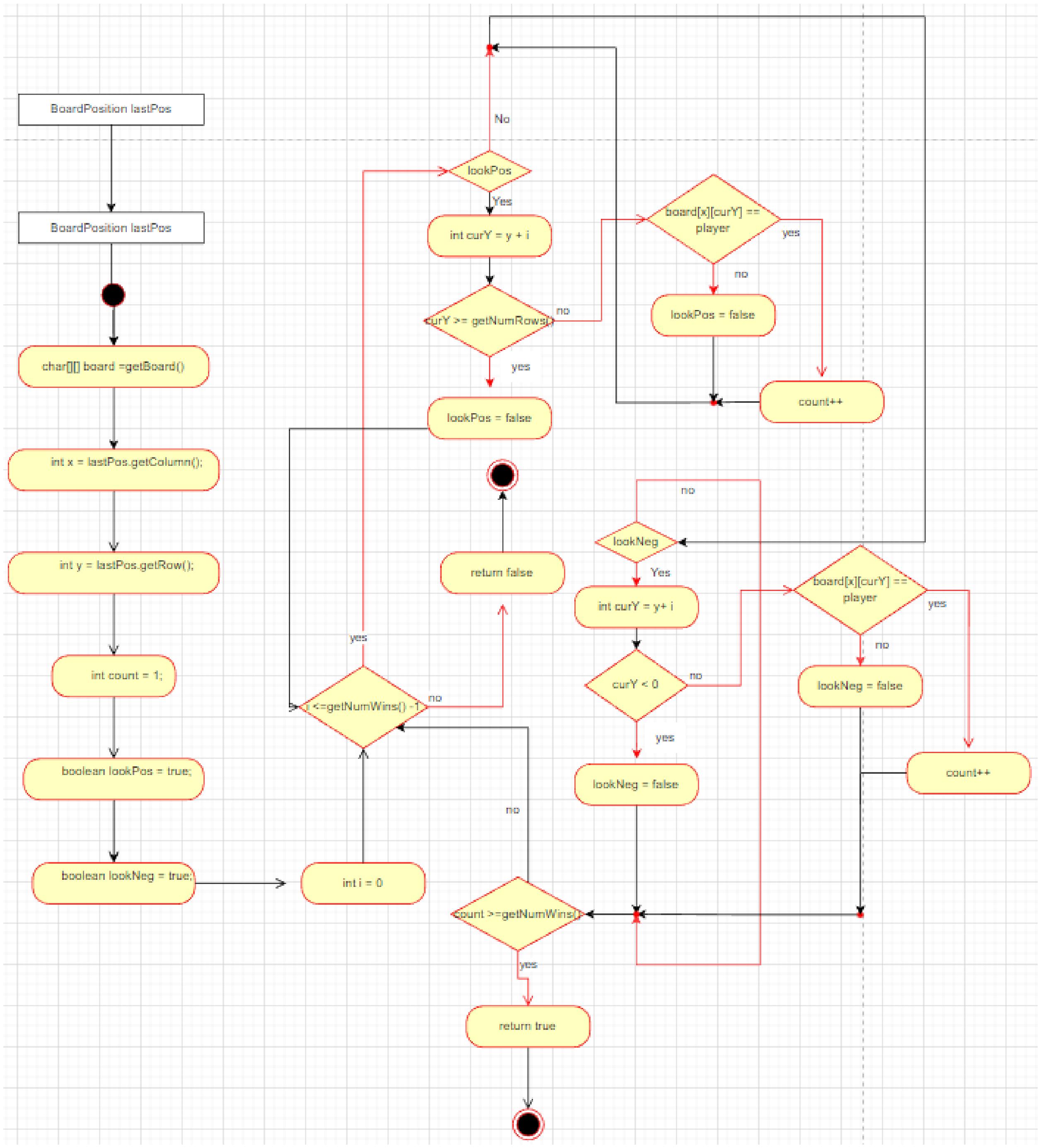
placeMarker

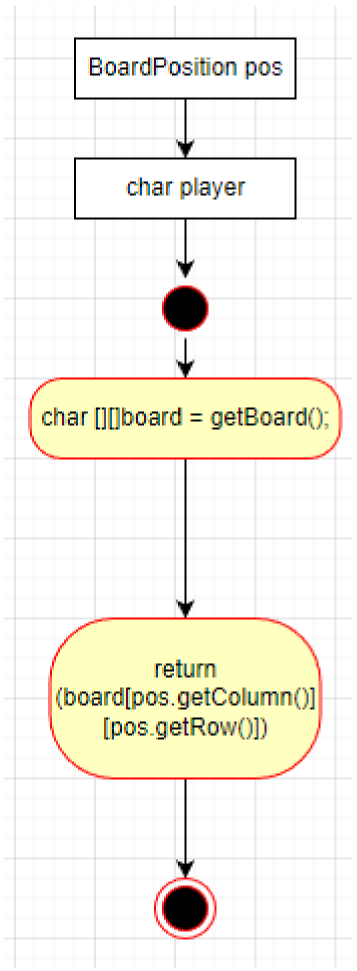


toString

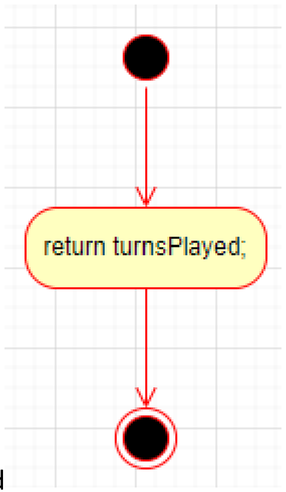

char [][] board =getBoard();
String boardScheme = " "
boardWidth = getNumColumns()
boardHeight = getNumRows()
int i = 0
i++
i < boardWidth
no
boardScheme += Integer.toString(i) + "|";
boardScheme += "\n"
int = k
k < boardHeight
yes
no
return boardScheme
k++
boardScheme += Integer.toString(i) + "|";
int j = 0
j < boardWidrth
yes
no
boardScheme += board[i][j]+ "|";
j ++
boardScheme += "\n"

checkHorizontalWin

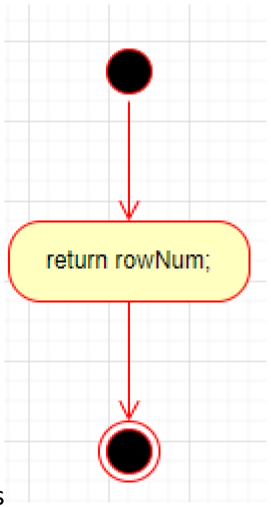



BoardPosition pos
char player
char [][]board = getBoard();
return (board[pos.getColumn()][pos.getRow()])

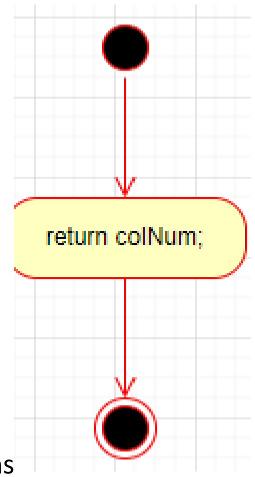
whatsAtPos



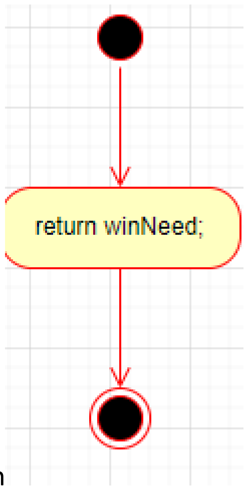
getTurnsPlayed



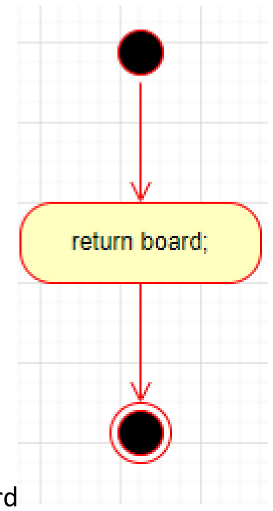
getNumRows



getNumColumns

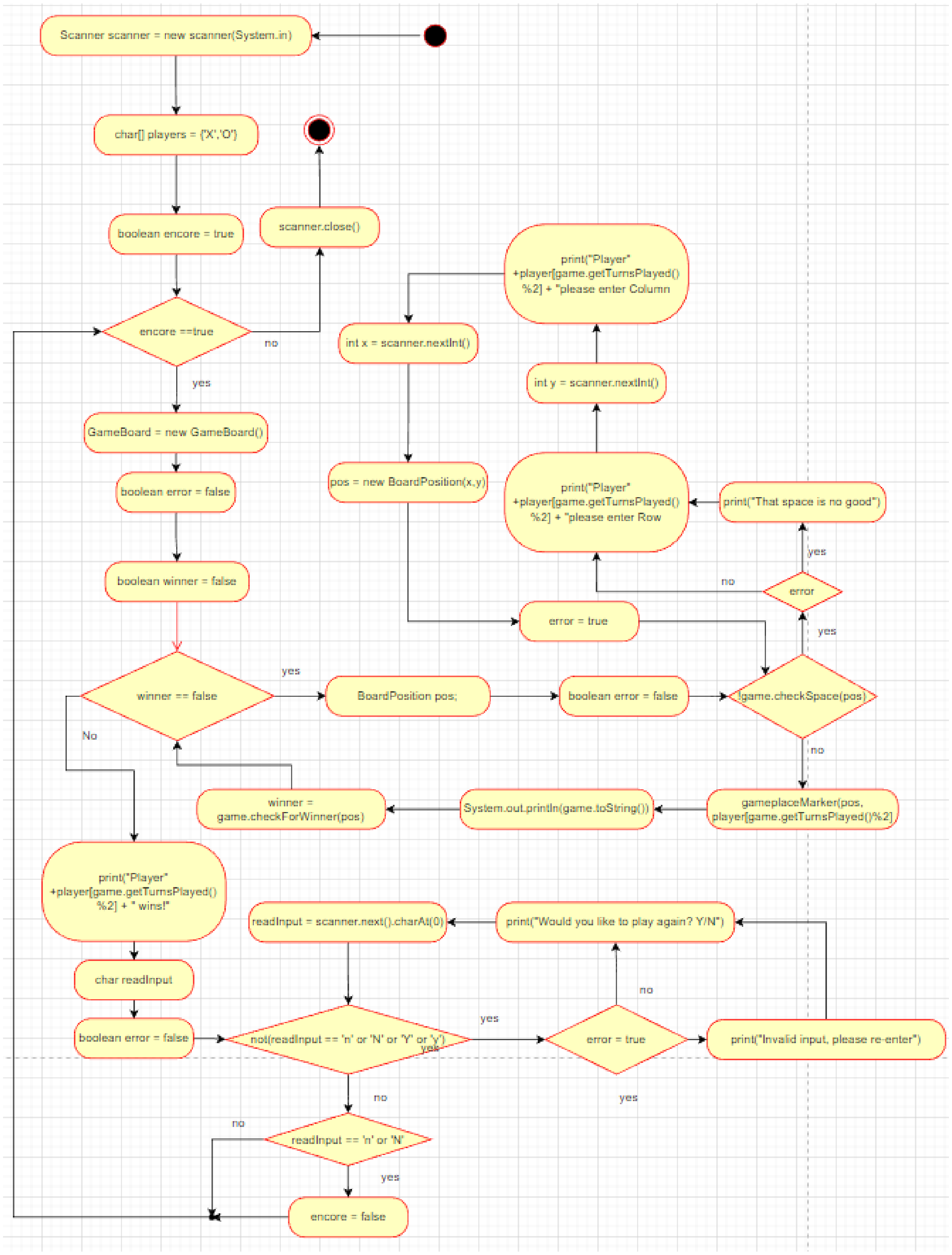


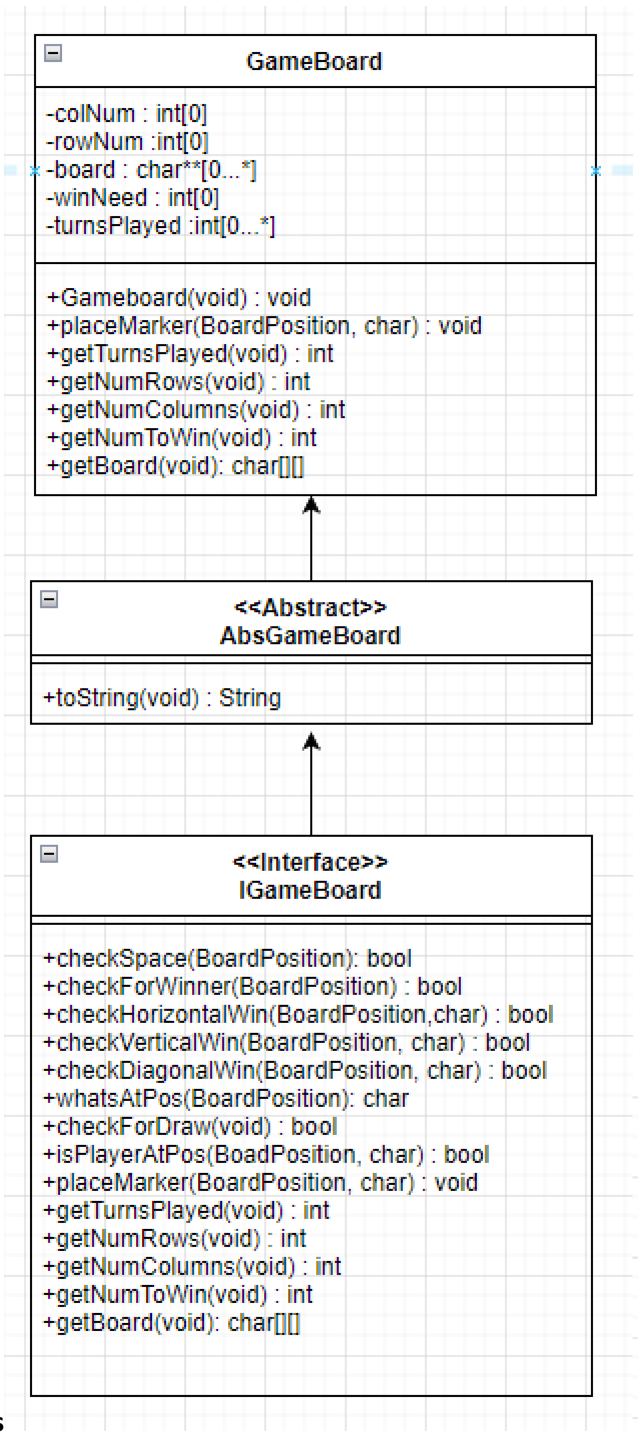
getNumToWin

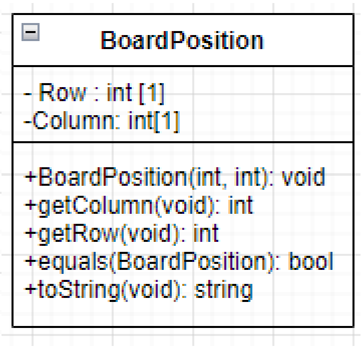


getBoard

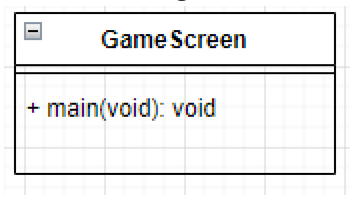
main


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


GameBoard
-colNum : int[0]
-rowNum :int[0]
-board : char**[0...*]
-winNeed : int[0]
-turnsPlayed :int[0...*]
+Gameboard(void) : void
+placeMarker(BoardPosition, char) : void
+getTurnsPlayed(void) : int
+getNumRows(void) : int
+getNumColumns(void) : int
+getNumToWin(void) : int
+getBoard(void): char[][]
<<Abstract>>
AbsGameBoard
+toString(void) : String
<<Interface>>
IGameBoard
+checkSpace(BoardPosition): bool
+checkForWinner(BoardPosition) : bool
+checkHorizontalWin(BoardPosition,char) : bool
+checkVerticalWin(BoardPosition, char) : bool
+checkDiagonalWin(BoardPosition, char) : bool
+whatsAtPos(BoardPosition): char
+checkForDraw(void) : bool
+isPlayerAtPos(BoadPosition, char) : bool
+placeMarker(BoardPosition, char) : void
+getTurnsPlayed(void) : int
+getNumRows(void) : int
+getNumColumns(void) : int
+getNumToWin(void) : int
+getBoard(void): char[][]



UML Class Diagrams


GameScreen
+ main(void): void