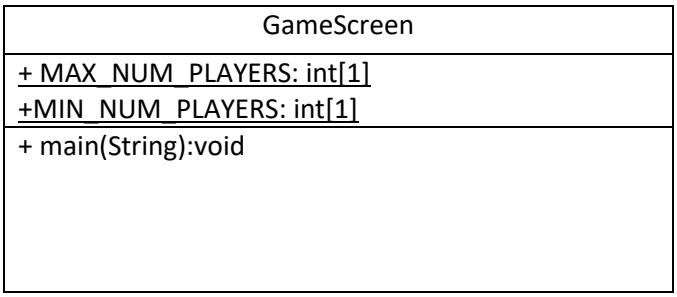
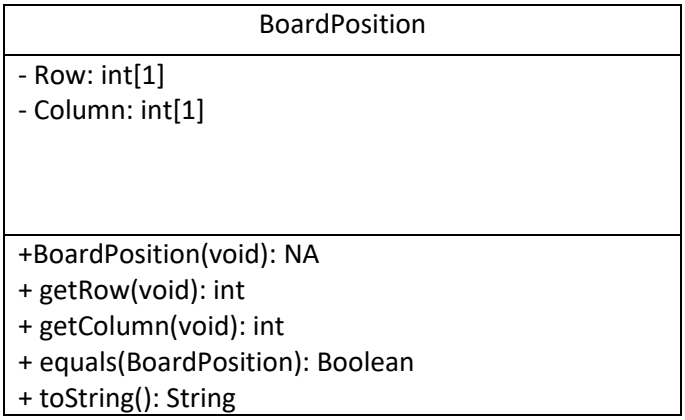
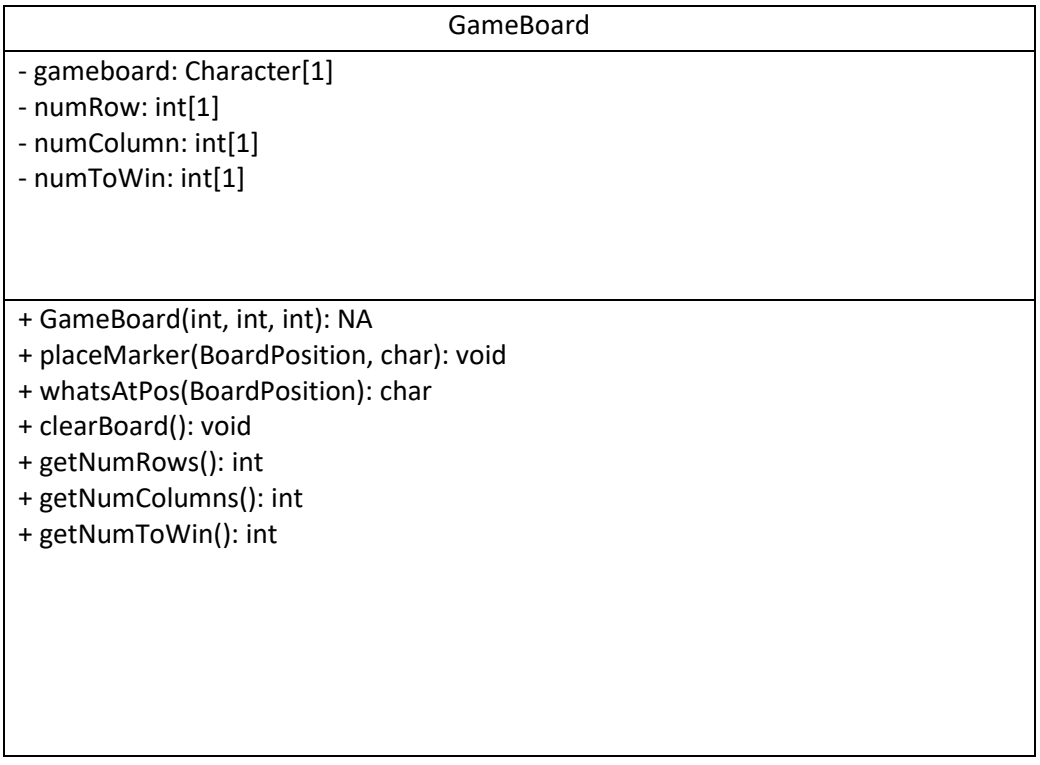
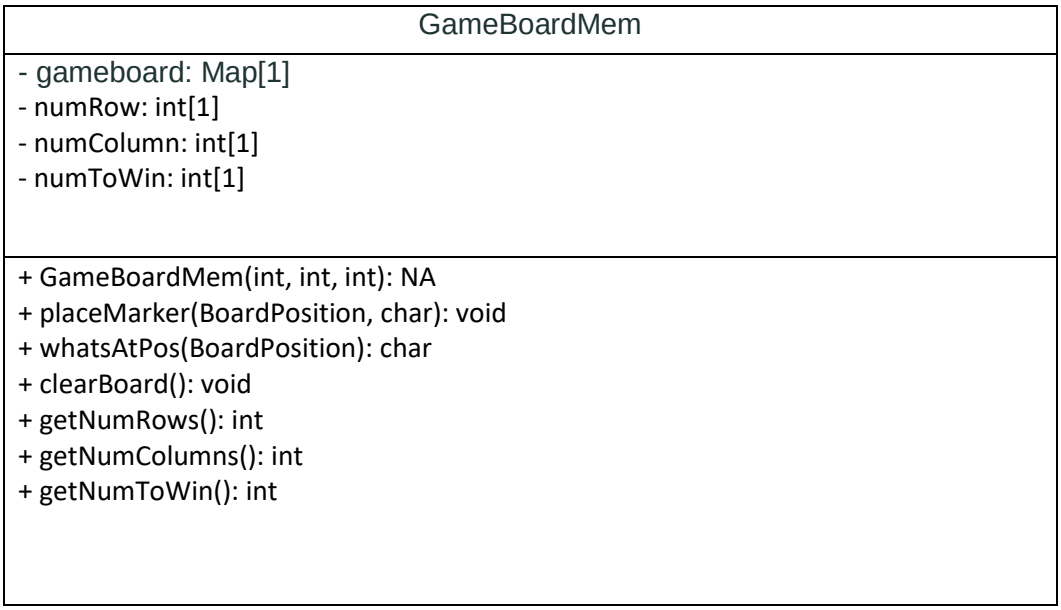
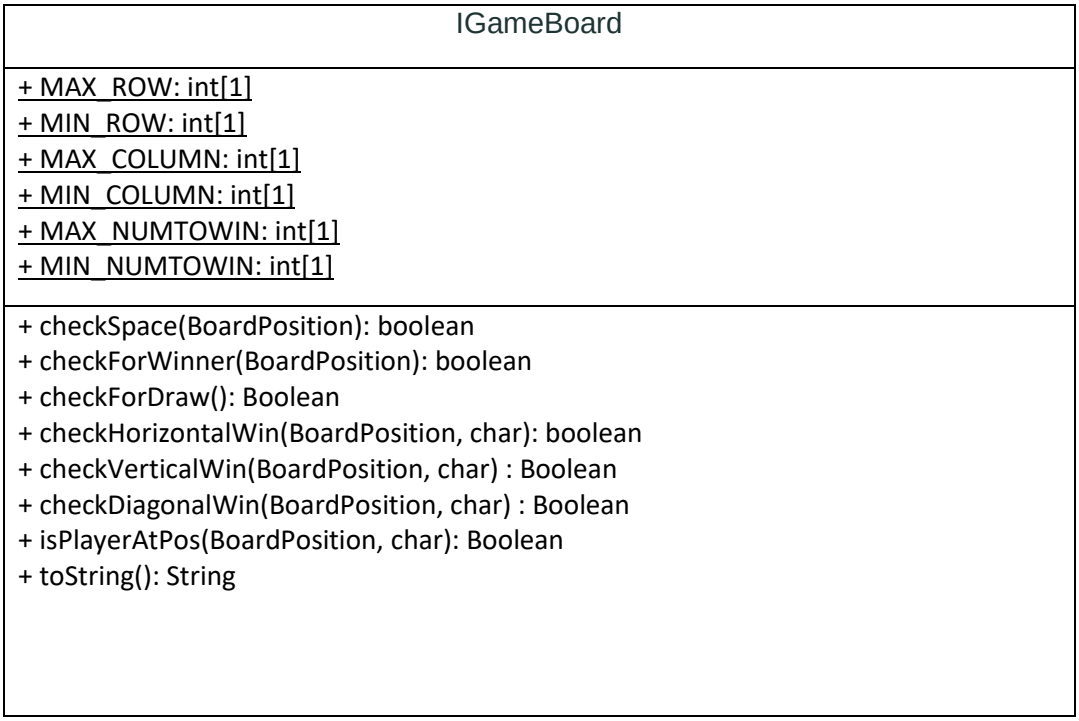

GameScreen
+ MAX_NUM_PLAYERS: int[1]
+MIN_NUM_PLAYERS: int[1]
+ main(String):void


BoardPosition
- Row: int[1]
- Column: int[1]
+BoardPosition(void): NA
+ getRow(void): int
+ getColumn(void): int
+ equals(BoardPosition): Boolean
+ toString(): String


GameBoard
- gameboard: Character[1]
- numRow: int[1]
- numColumn: int[1]
- numToWin: int[1]
+ GameBoard(int, int, int): NA
+ placeMarker(BoardPosition, char): void
+ whatsAtPos(BoardPosition): char
+ clearBoard(): void
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int


GameBoardMem
- gameboard: Map[1]
- numRow: int[1]
- numColumn: int[1]
- numToWin: int[1]
+ GameBoardMem(int, int, int): NA
+ placeMarker(BoardPosition, char): void
+ whatsAtPos(BoardPosition): char
+ clearBoard(): void
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int


IGameBoard
+ MAX_ROW: int[1]
+ MIN_ROW: int[1]
+ MAX_COLUMN: int[1]
+ MIN_COLUMN: int[1]
+ MAX_NUMTOWIN: int[1]
+ MIN_NUMTOWIN: int[1]
+ checkSpace(BoardPosition): boolean
+ checkForWinner(BoardPosition): boolean
+ checkForDraw(): Boolean
+ checkHorizontalWin(BoardPosition, char): boolean
+ checkVerticalWin(BoardPosition, char) : Boolean
+ checkDiagonalWin(BoardPosition, char) : Boolean
+ isPlayerAtPos(BoardPosition, char): Boolean
+ toString(): String

TicTacToeController
- currGame: IGameBoard [1] - screen: TicTacToeView [1] - numPlayers: int [1] - players: Character [1] - playerTurn: int [1] - win: boolean [1] - tie: boolean [1]
+TicTacToeController(IGameBoard, TicTacToeView, int): NA + processButtonClick(int, int): void - newGame(void): void

Functional Requirements:

As a player, I can place a token in any open spot so that I can try to get 5 in a row to win.

As a player, I can reselect a different spot if the spot I chose already contains a token.

As a player, I can reselect a different spot if I choose an invalid position.

As a player, I can choose to play again when the game ends so that I do not have to re-run the program.

As a player, I can win by getting a set number of my markers in a horizontal row.

As a player, I can win by getting a set number of my markers in a vertical row.

As a player, I can win by getting a set number of my markers in a diagonal row.

As a player, only I or my opponent can win.(Only one of us can win)

As a player, after I go, it is the next player in the rotations turn to go.

As a player, if all the spots on the board are taken, and there is still no win, it is a draw.

As a player, I can choose any character to represent my player.

As a player, I can set the side of the board, ranging from 3x3 to 100x100.

As a player, I can set the number of tokens I need in a row to win, ranging from 3 to 25.

The game will be memory efficient if the total area of the board (determined by rows * columns) is greater than MEM_CUTOFF (64).

Non-Functional Requirements:

Must be written in Java

Gameboard is of size determined by user (3 – 20) x (3 – 20)

0,0 is the top left of the board

Must be run using IntelliJ

Testing:

GameBoard(int r, int c, int w)

Input	Board is initialized with a row value of 32 a column value of 25 and a numtoWin value of 13. All positions are set equal to ' '.	This is a normal run of the constructor with effectively random values.
r = 32 c = 25 w = 13 board doesn't exist yet		

GameBoard(int r, int c, int w)

Input	Board is initialized with a row value of 3 a column value of 3 and a numtoWin value of 3. All positions are set equal to ' '.	This test case is unique because it tests with the minimum values according to the invariants.
r = 3 c = 3 w = 3 board doesn't exist yet	(numToWin = 3)	

	0	1	2
0			
1			
2			

GameBoard(int r, int c, int w)

<table>
<tr>
<td>Input

r = 100
c = 100
w = 25

board doesn’t exist yet</td>
<td>Board is initialized with a row value of 100 a column value of 100 and a numtoWin value of 25. All positions are set equal to ‘ ‘.

(numToWin = 25)
<table><tr><td></td><td>0</td><td>1</td><td>…</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>…</td><td></td><td></td><td></td></tr></table></td>
<td>This test case is unique because it tests with the maximum values according to the invariants. Board was too big to display in word.</td>
</tr>
</table>

Boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td>Input

(numToWin = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>
Pos.getRow = 0
Pos.getColumn = 0</td>
<td>Output

checkSpace = false

state of the board is unchanged</td>
<td>This test case is unique because it checks the lowest row and column numbers possible according to the invariants.</td>
</tr>
</table>

Boolean checkSpace(BoardPosition pos)

Input	Output	This test case is unique because it checks the highest row and column numbers possible according to the invariants.

(numToWin = 3)

	0	1	2	3	4
0					
1					
2					
3					
4					X

Pos.getRow = 4
Pos.getColumn = 4

checkSpace = false

state of the board is unchanged

Boolean checkSpace(BoardPosition pos)

Input	Output	This test case is unique because it checks the highest row and column numbers possible according to the invariants, but the space is free.

(numToWin = 3)

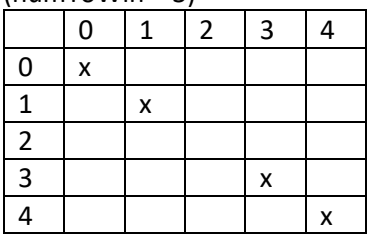
	0	1	2	3	4
0					
1					
2					
3					
4					

Pos.getRow = 4
Pos.getColumn = 4

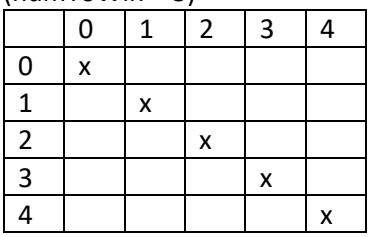
checkSpace = true

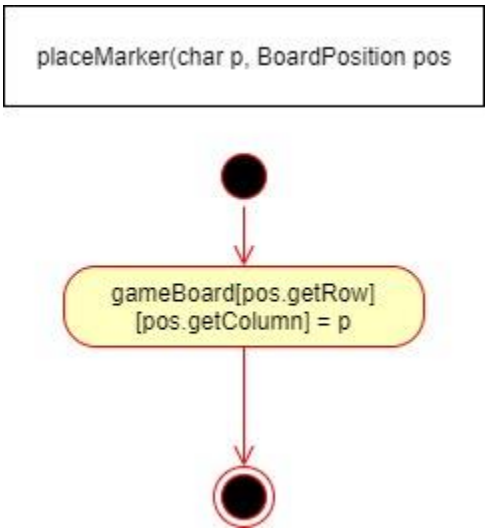
state of the board is unchanged

Boolean checkHorizontalWin(BoardPosition pos, char p)

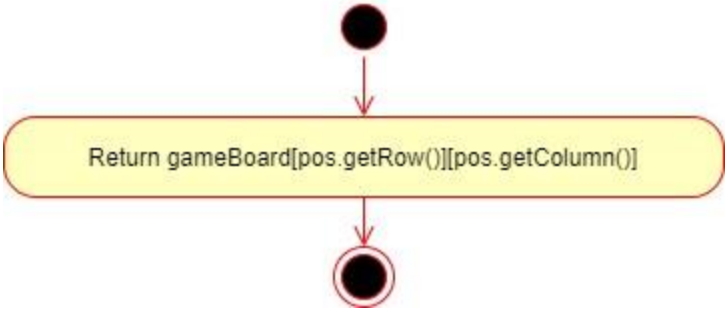
Input	Output	This test case is unique because it checks a split win from the left and from the right and determines it to be false.
(numToWin = 3)  Pos1.getRow = 4 Pos1.getColumn = 4 Pos2.getRow = 0 Pos2.getColumn = 0	checkHorizontalWin = false state of the board is unchanged	

Boolean checkHorizontalWin(BoardPosition pos, char p)

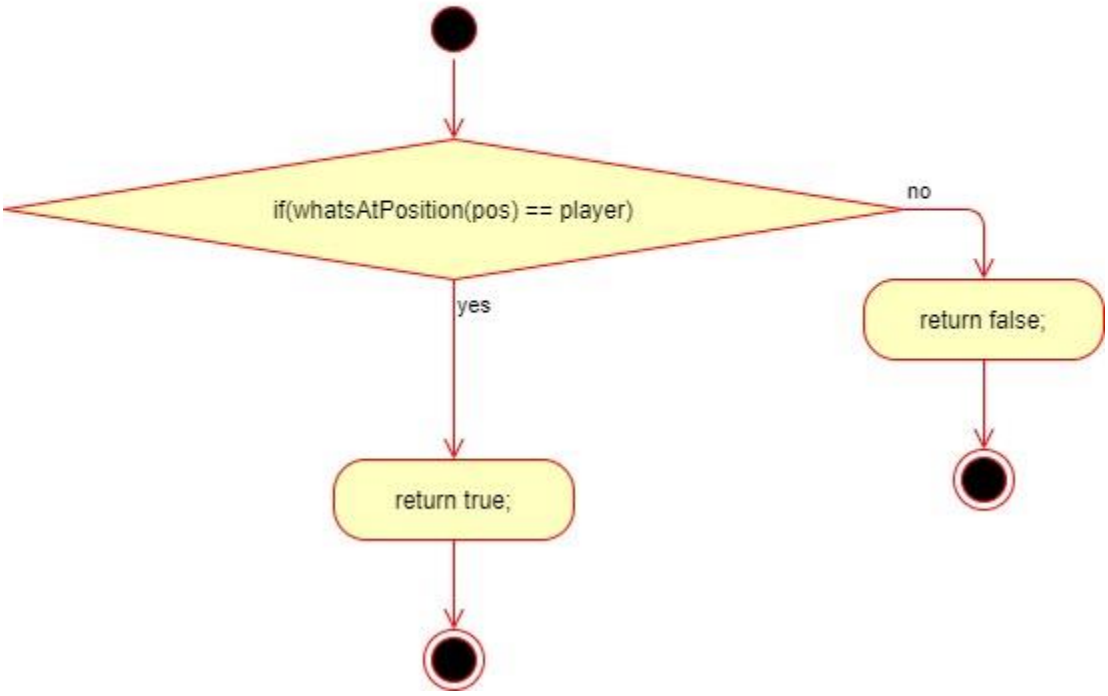
Input	Output	This test case is unique because it checks a win from the left and from the right and determines it to be true.
(numToWin = 3)  Pos1.getRow = 4 Pos1.getColumn = 4 Pos2.getRow = 0 Pos2.getColumn = 0	checkHorizontalWin = true state of the board is unchanged	

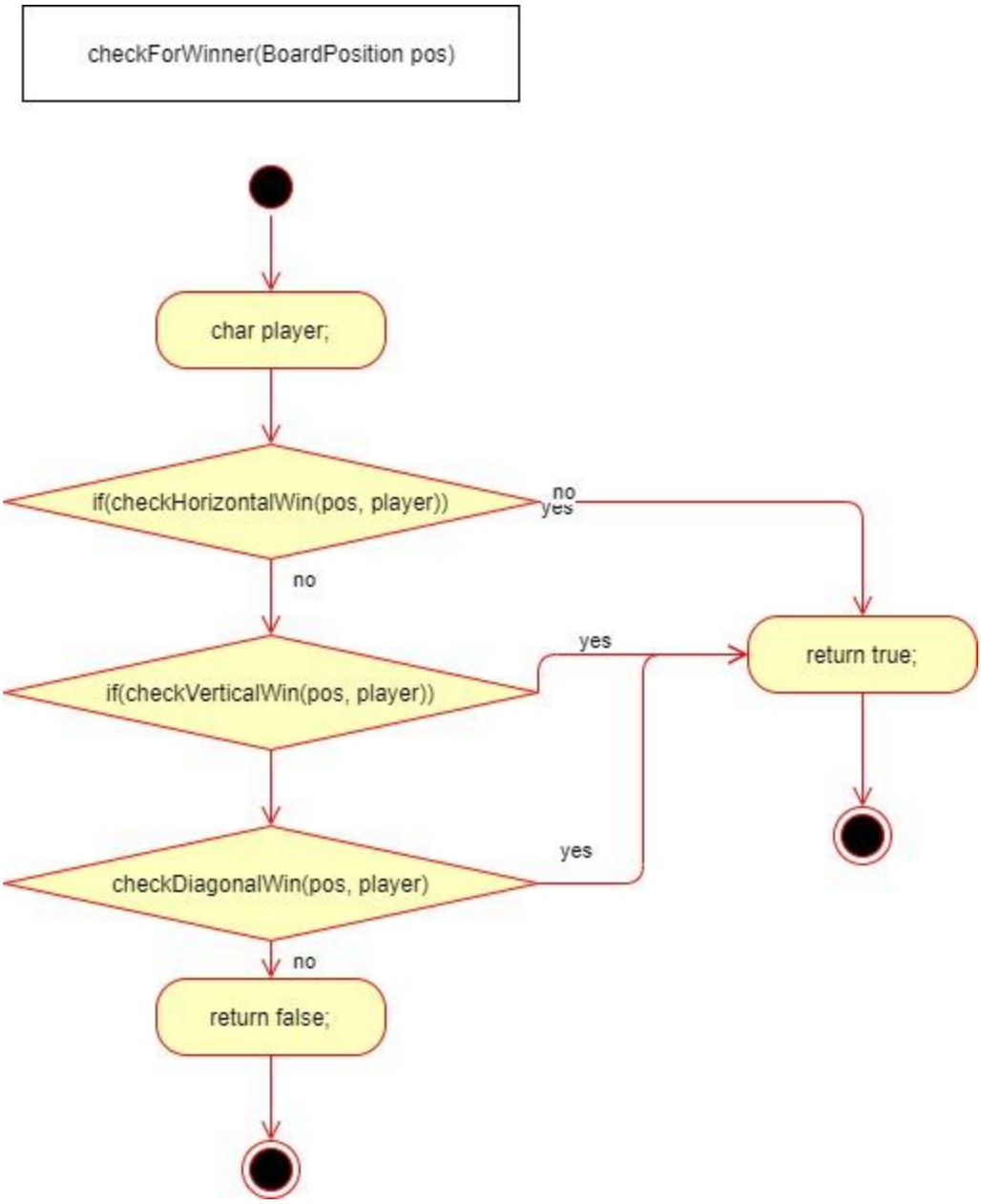

placeMarker(char p, BoardPosition pos
gameBoard[pos.getRow]
[pos.getColumn] = p

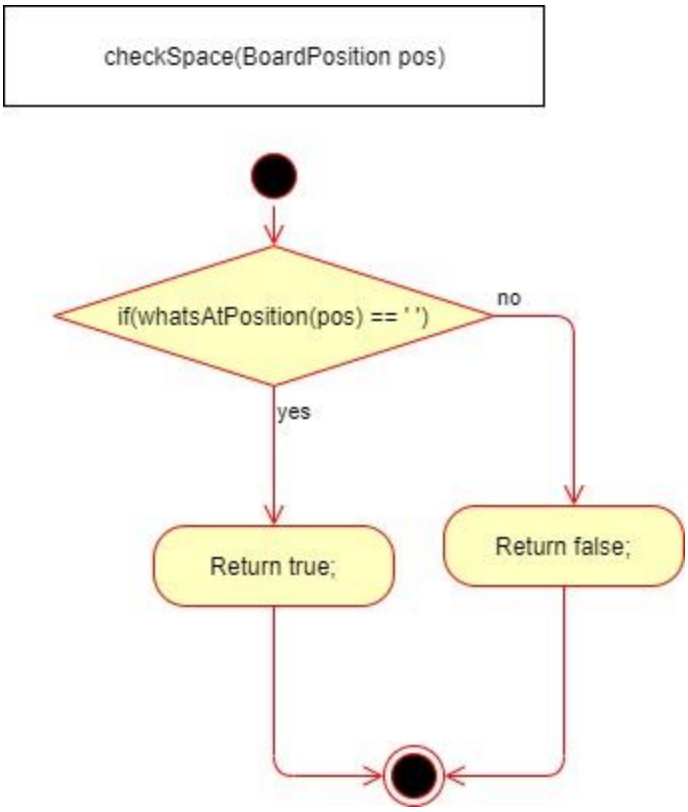
whatsAtPos(BoardPosition pos)

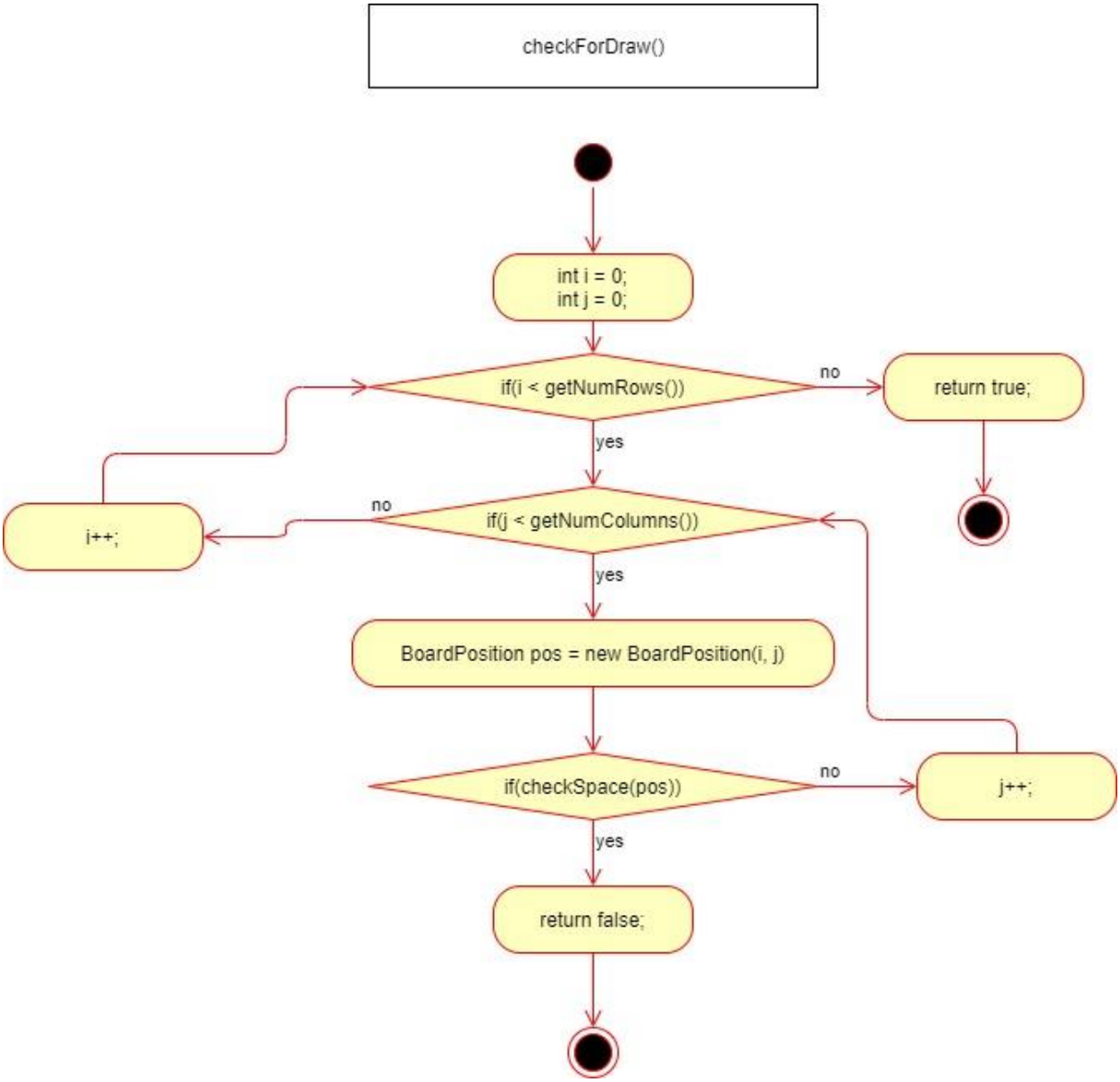


isPlayerAtPos(BoardPosition pos, char player)

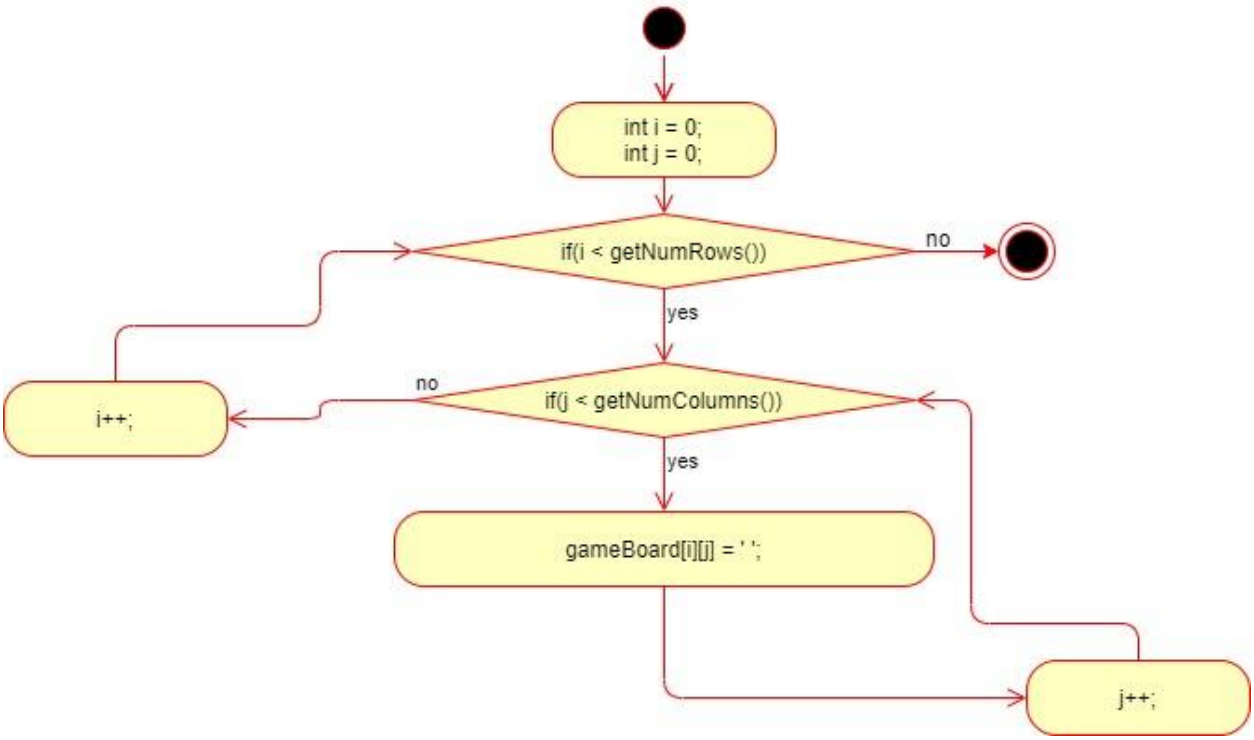



checkForWinner(BoardPosition pos)
char player;
if(checkHorizontalWin(pos, player))
no
yes
no
if(checkVerticalWin(pos, player))
yes
return true;
checkDiagonalWin(pos, player)
yes
no
return false;


checkSpace(BoardPosition pos)
if(whatsAtPosition(pos) == ' ')
no
yes
Return true;
Return false;


checkForDraw()
int i = 0;
int j = 0;
if(i < getNumRows())
no
return true;
yes
if(j < getNumColumns())
no
i++;
yes
BoardPosition pos = new BoardPosition(i, j)
if(checkSpace(pos))
no
j++;
yes
return false;

clearBoard()


int i = 0;
int j = 0;
if(i < getNumRows())
no
yes
if(j < getNumColumns())
no
i++;
yes
gameBoard[i][j] = ' ';
j++;