CPSC 2150 Project Report

Yash Patel

Requirements Analysis

Functional Requirements:

1. As a player, I need to place a marker so that I can indicate my turn
2. As a player, I want to restart the game after drawing so I can continue playing.
3. As a player, I need to alternate players so that I can have each of them have a turn playing.
4. As a programmer, I need to have a computer to play with, so I can view and test my game.
5. As a player, I can enter the number of rows and columns between 3 - 20 so I can customize the board
6. As a player, I need to be able to place my marker anywhere on a [num rows] x [num columns] grid, so then I can use my turn.
7. As a gamescreen, I need to know when all the spaces are filled with no winning side so that it can display the draw message.
8. As a player, I want to be able to quit the game so that I can stop playing
9. As a player, I need to be player 'X' if I am first so I can start game
10. As a gameboard, I need to place either 'X' or 'O', so I know the game pieces
11. As a gameboard, I need to check to see if the row/col is valid so that I can place a marker there
12. As a gameboard, I need to know if 5 of the same markers are placed in the same column to count as a vertical win
13. As a gameboard, I need to know if 5 of the same markers are placed in the same row to count as a horizontal win
14. As a gameboard, I need to know if 5 of the same markers are placed in a diagonal line to count as a diagonal win
15. As a player, I should be able to change the number of players in a game from 2-10 so I can play with more people

Non-Functional Requirements

1. The program is written in java
2. The program is played using a GUI

3. The program must have at least two players and at max 10
4. The program must be developed in IntelliJ
5. The player must know the rules of tic-tac-toe
6. the program must have a [num rows] x [num columns] board, meaning the user can choose the size from 3 to 20
7. The 'X' Player must go first

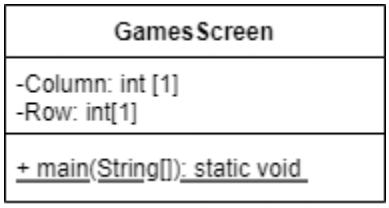
Deployment Instructions

Details in Projects 2-5.

System Design

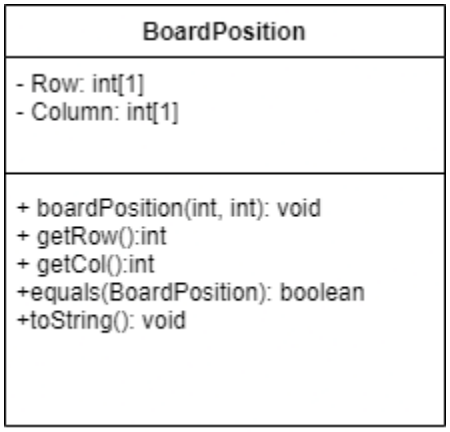
Class 1: GameScreen

Class diagram



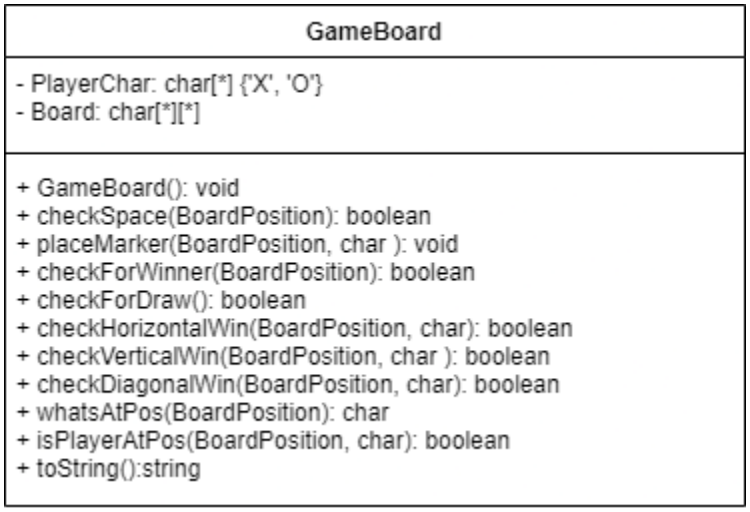
Class 2: BoardPosition

Class diagram



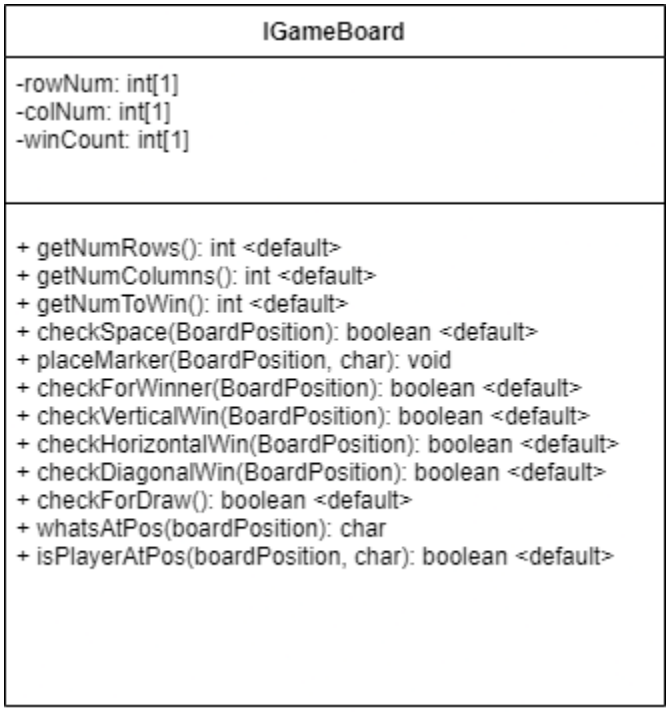
Class 3: GameBoard

Class diagram



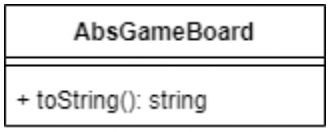
Class 4: IGameBoard

Class diagram



Class 5: AbsGameBoard

Class diagram



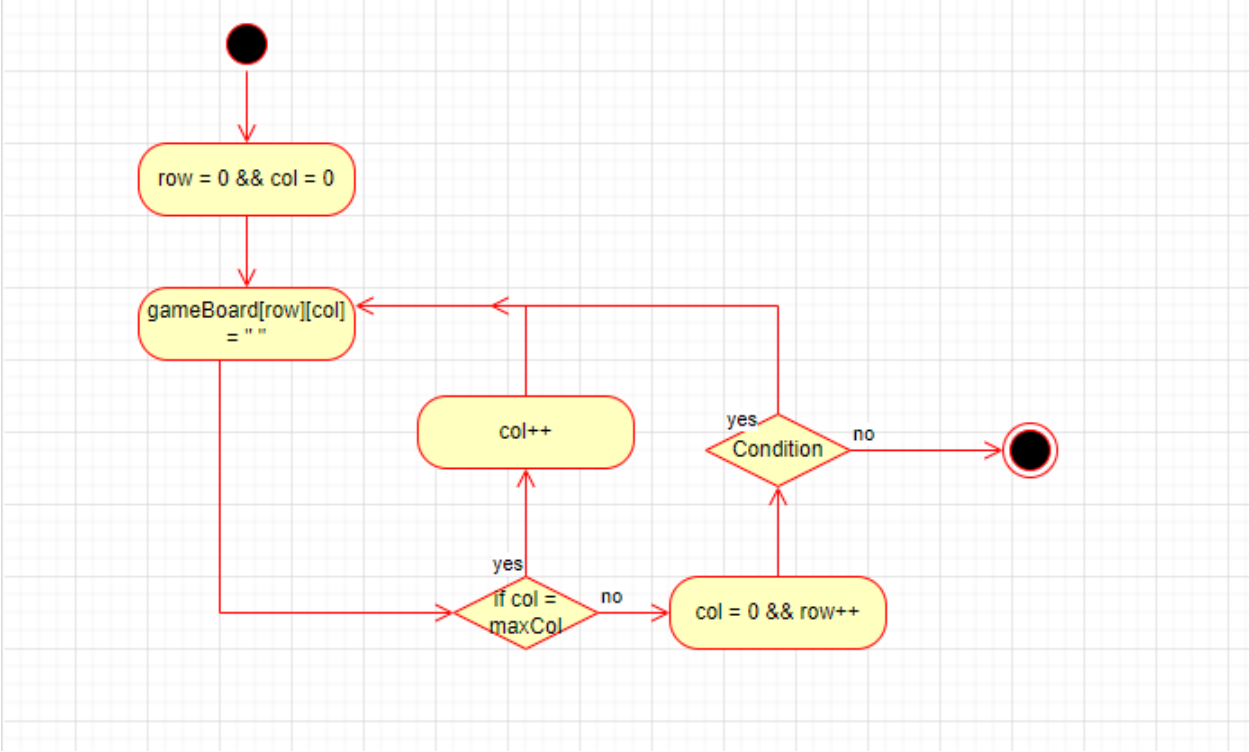
Class 6: GameBoardMem

Class diagram

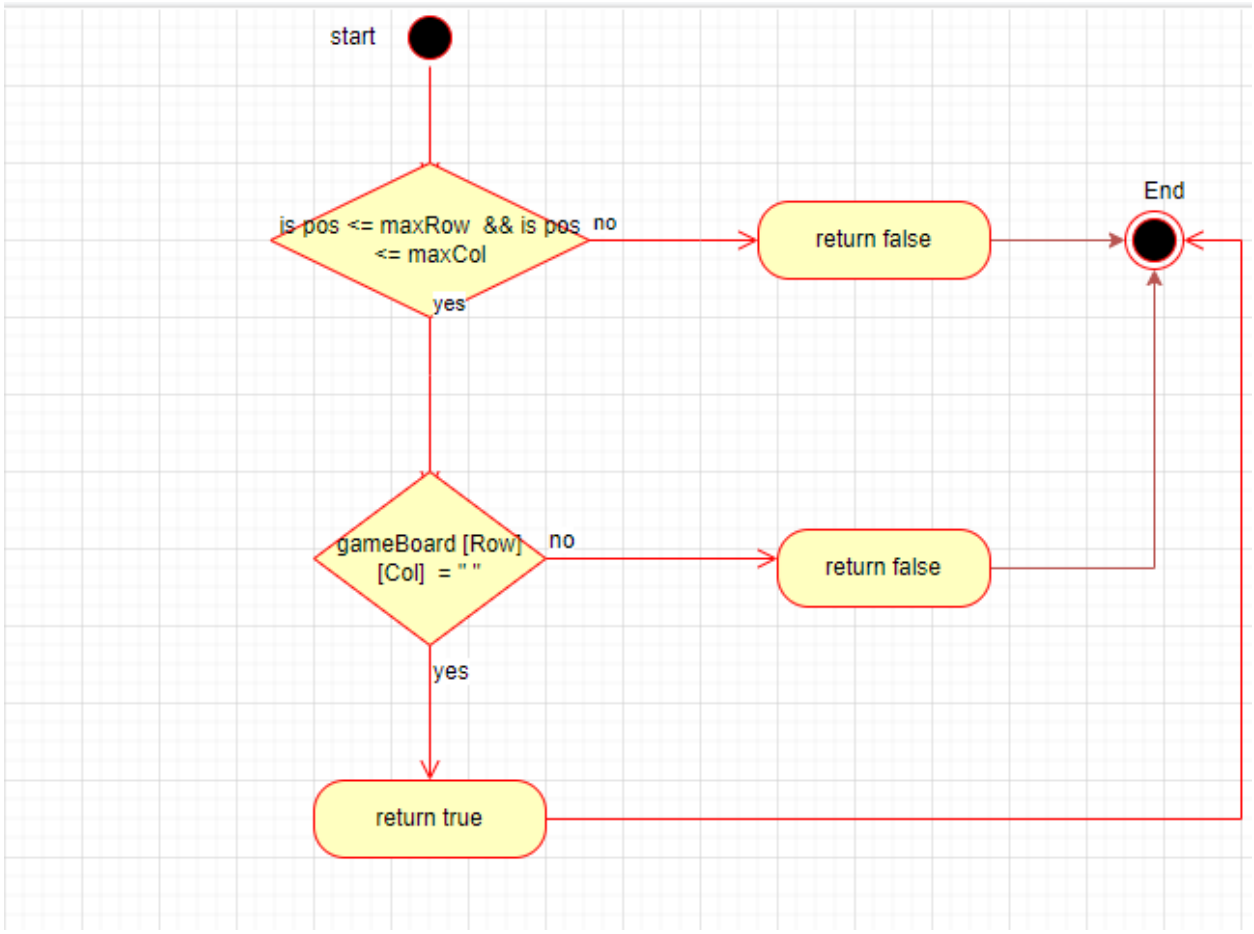
GameBoardMem
- Map<Character, List<BoardPosition>> board: board - playerChar: char - winScore: int - row: int - col: int
+ GameBoardMem(int,int,int) + placeMarker(BoardPosition,char): void + whatsAtPos(BoardPosition): char + isPlayerAtPos(BoardPosition, char): boolean + getNumRows(): int + getNumColumns(): int +getNumToWIn(): int

Activity diagrams

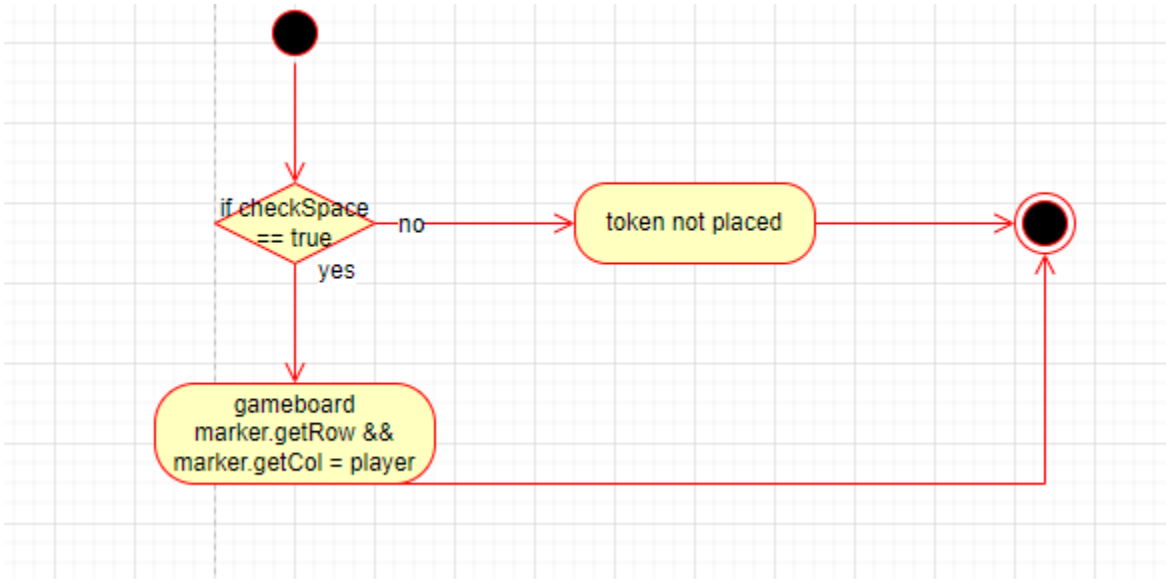
Gameboard(Constructor):



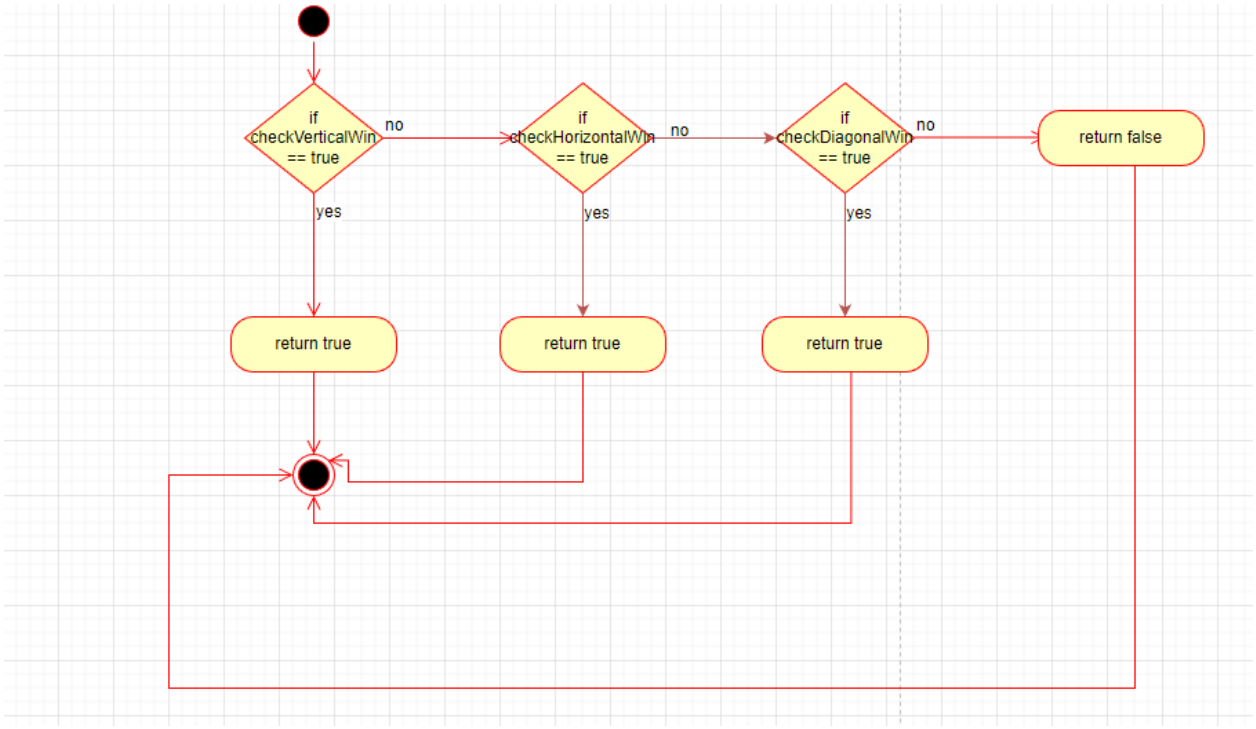
public boolean checkSpace(BoardPosition pos):



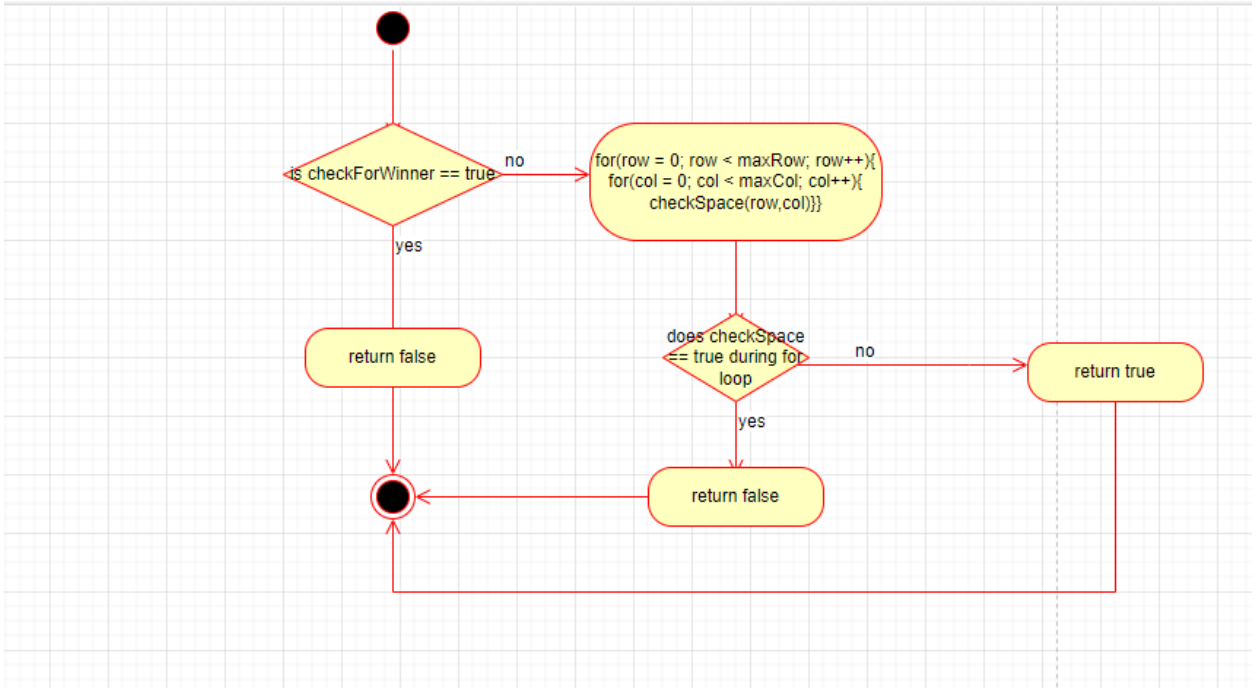
public void placeMarker(BoardPosition marker, char player):



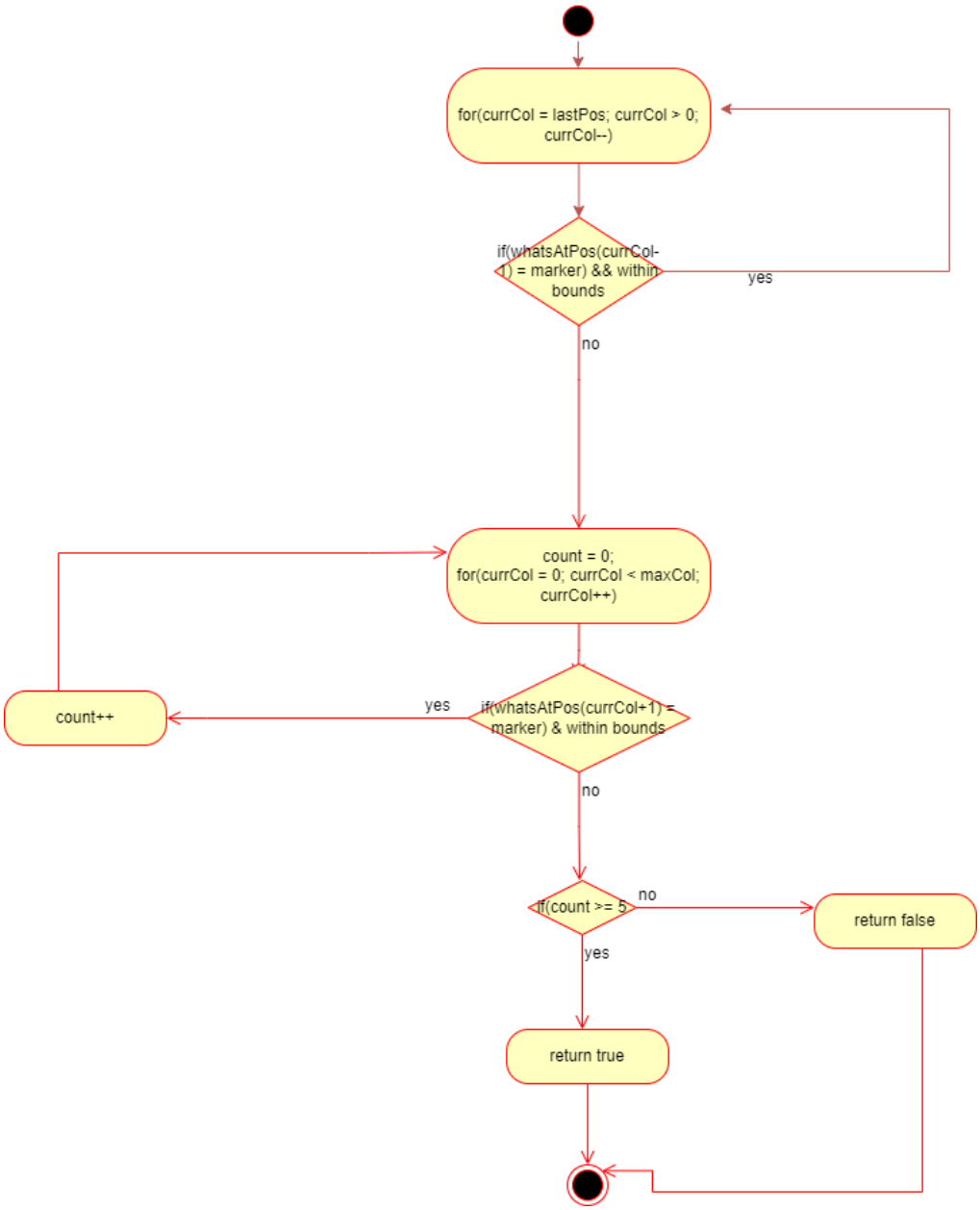
public boolean checkForWinner(BoardPosition lastPos):



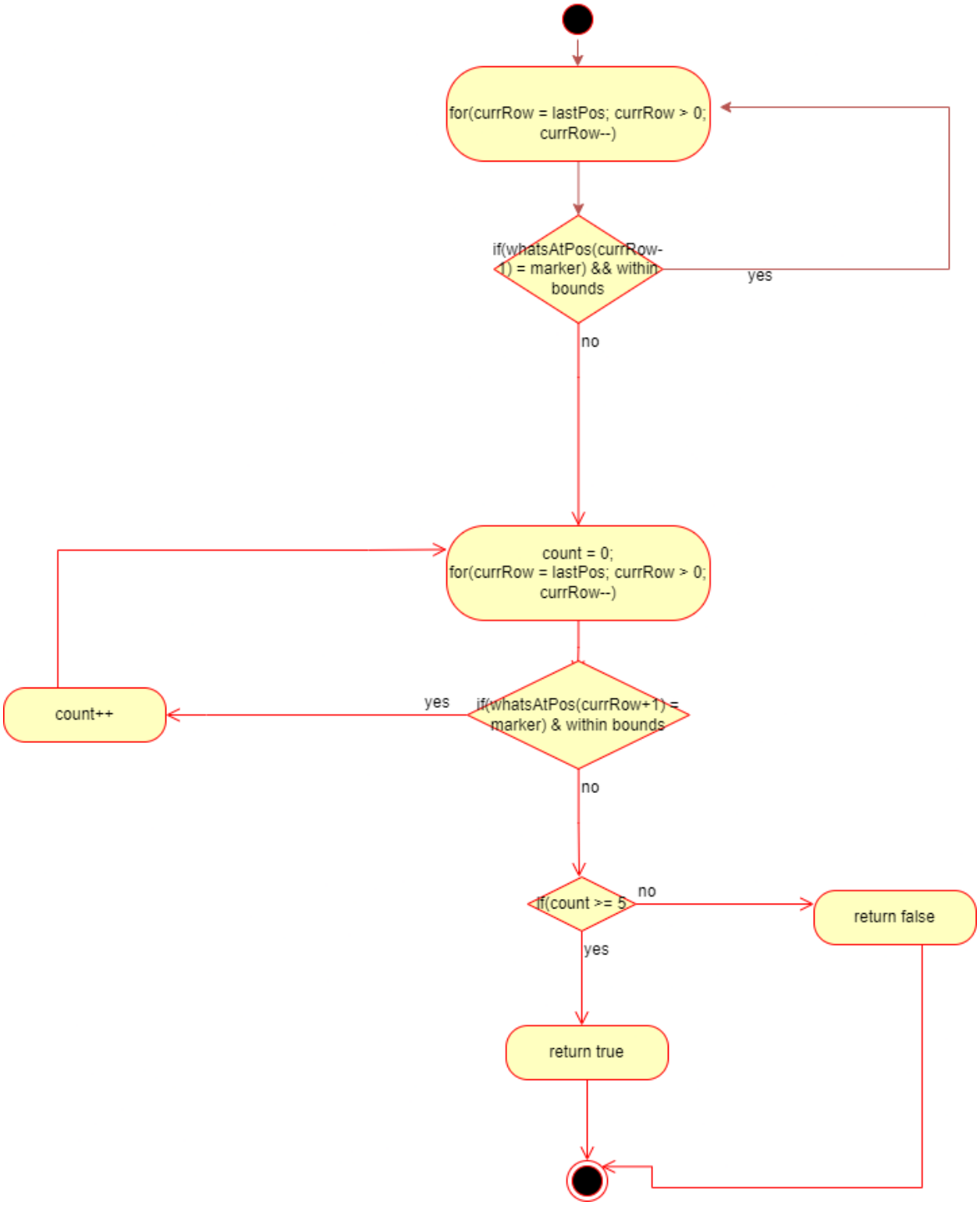
public boolean checkForDraw():



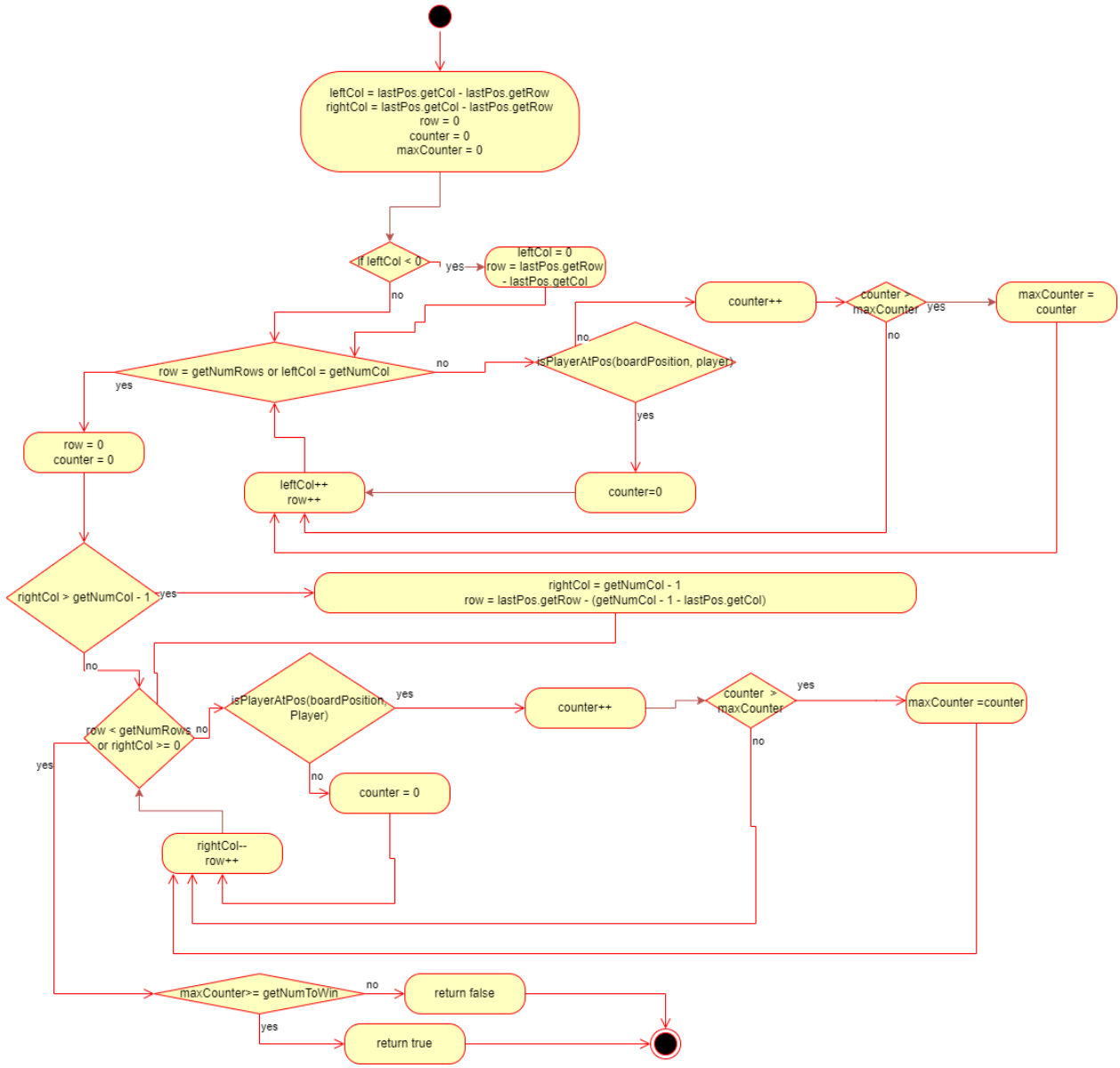
public boolean checkHorizontalWin(BoardPosition lastPos, char player):



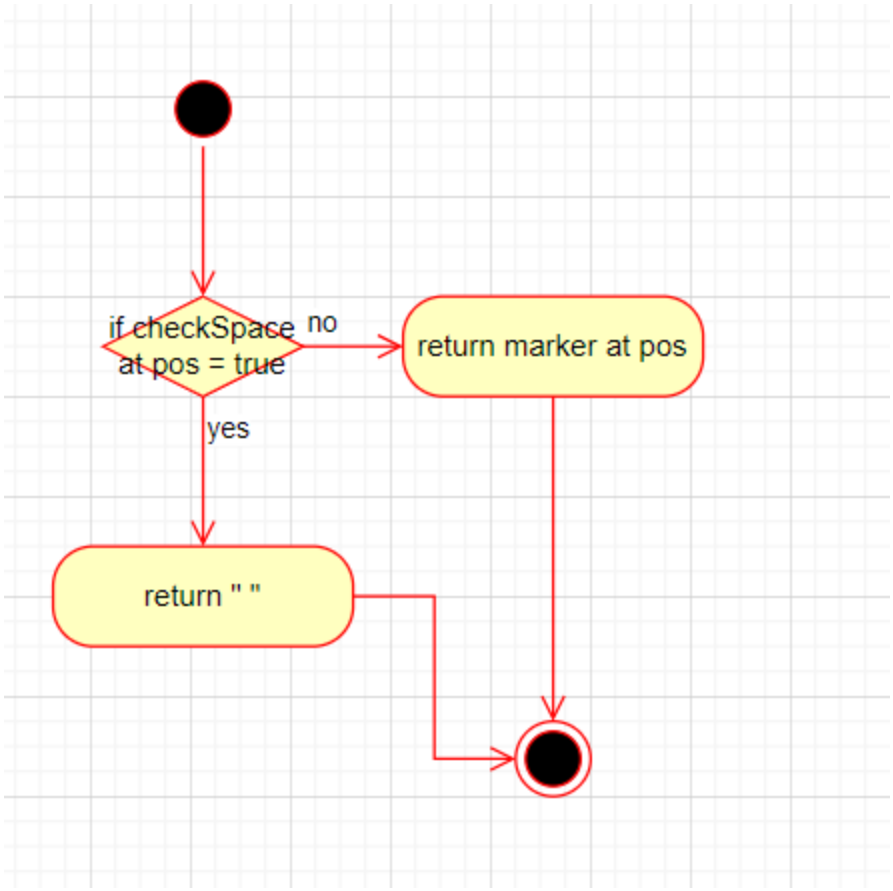
public boolean checkVerticalWin(BoardPosition lastPos, char player):



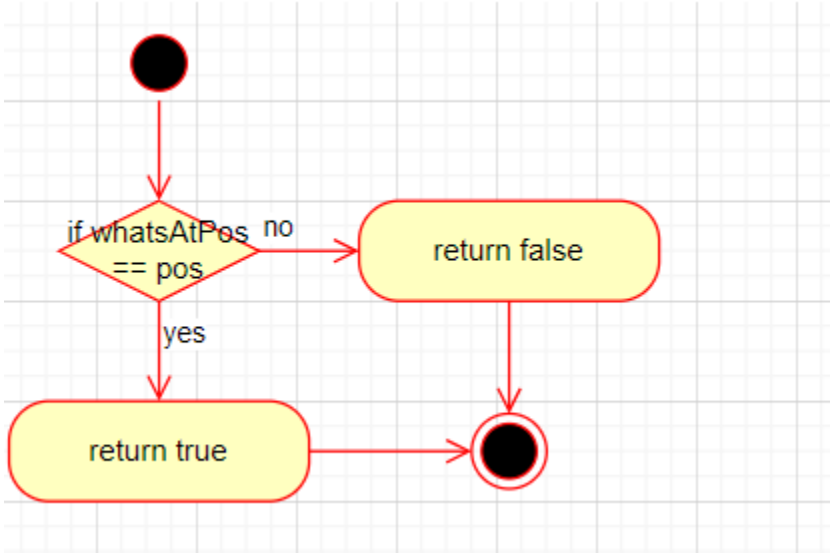
public boolean checkDiagonalWin(BoardPosition lastPos, char player):



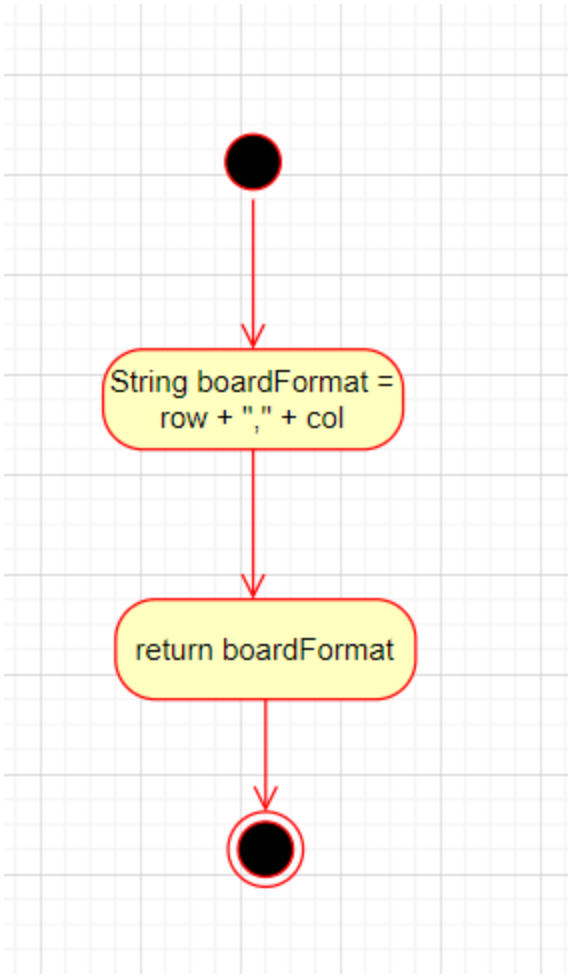
public char whatsAtPos(BoardPosition pos):



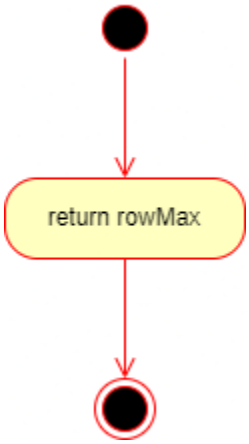
public boolean isPlayerAtPos(BoardPosition pos, char player):



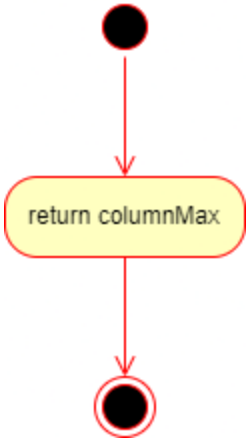
toString():



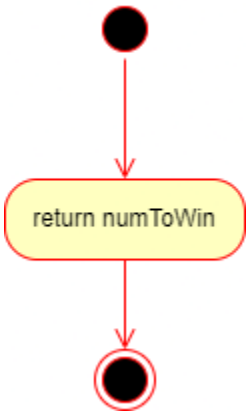
getNumRows():



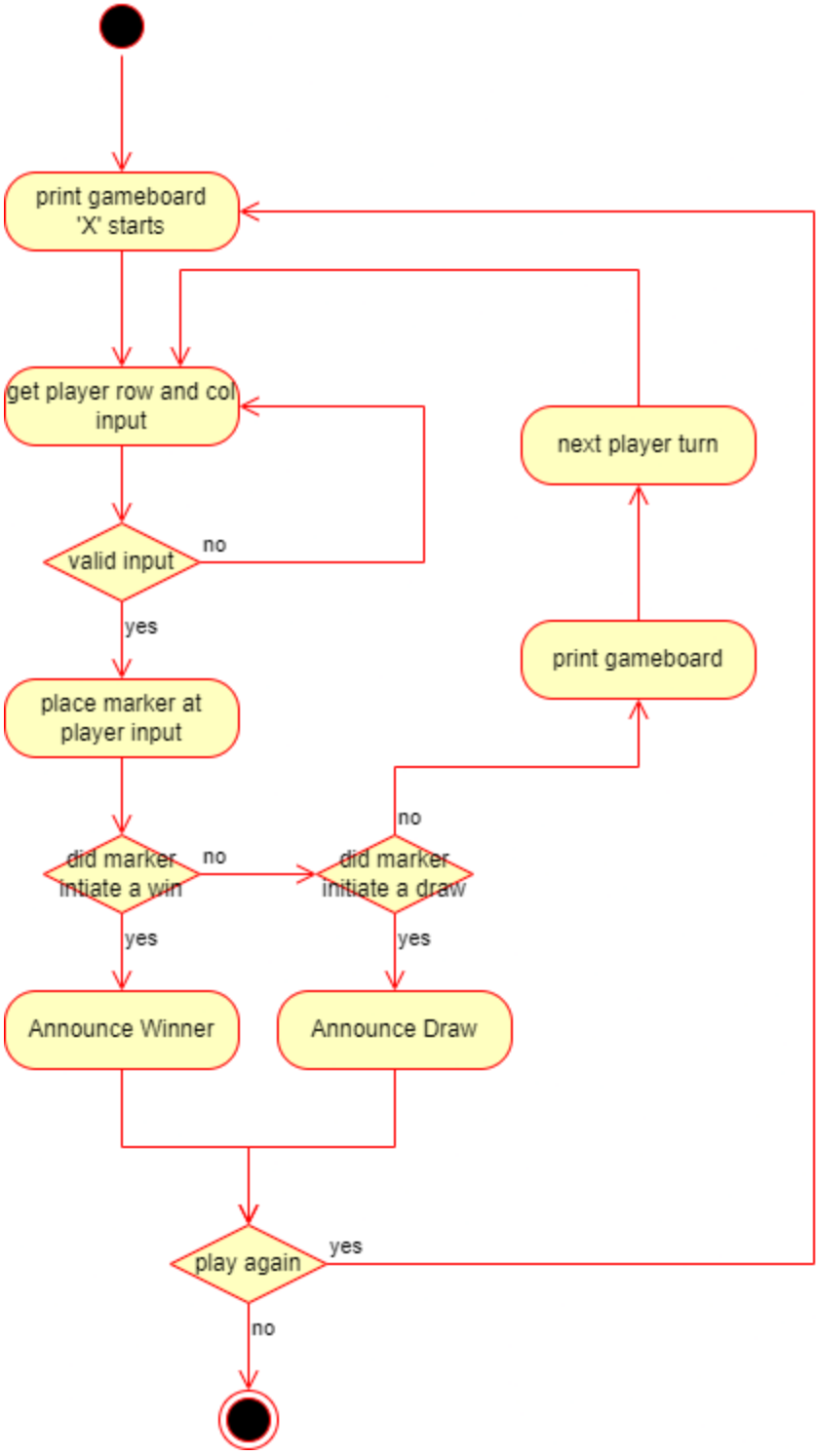
getNumCol();



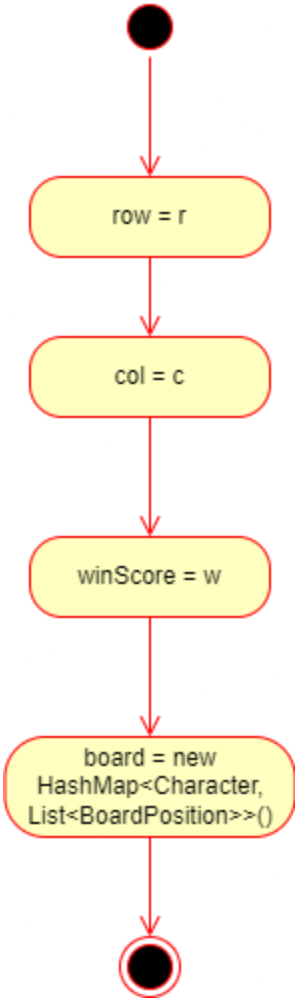
getNumToWin():



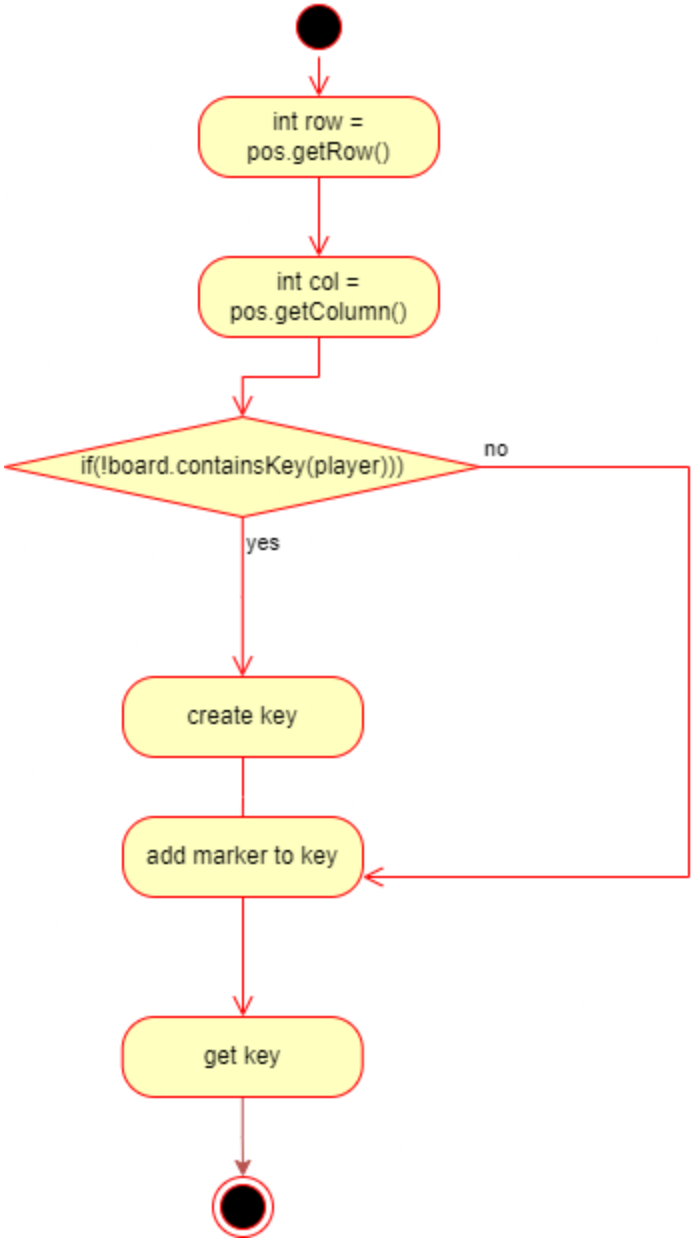
main():



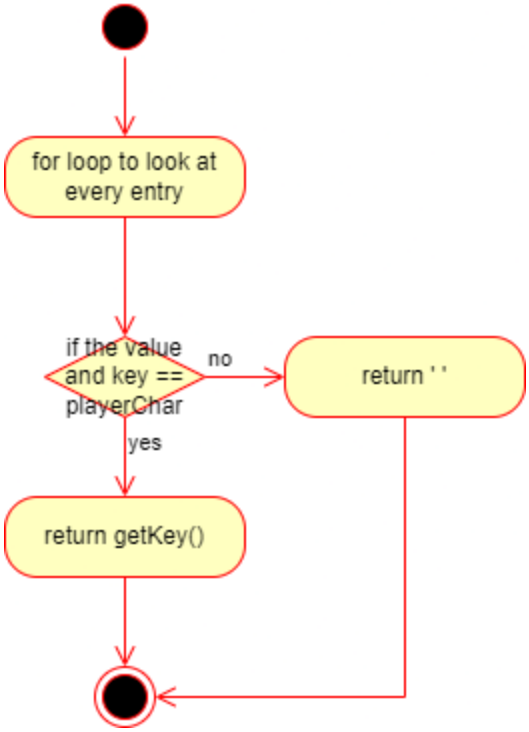
GameBoardMem(int r, int c, int w):



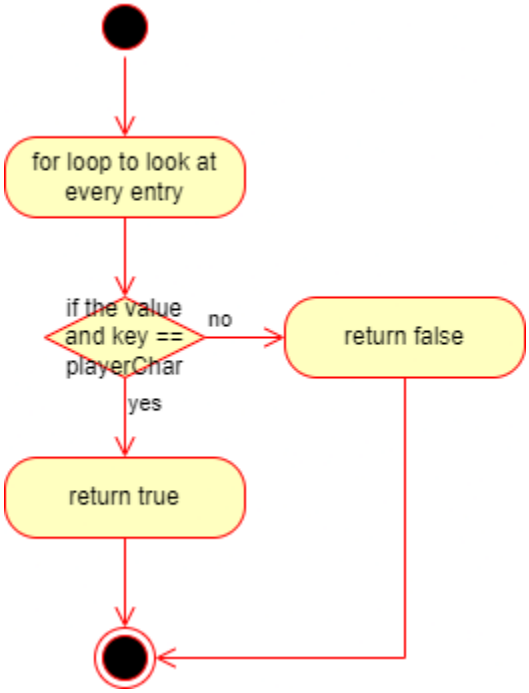
placeMarker(BoardPosition pos, char Player):



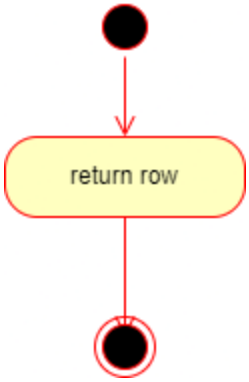
whatsAtPos(BoardPosition pos):



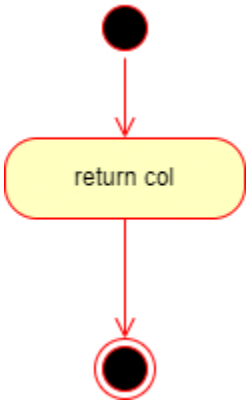
@Override isPlayerPos(BoardPosition pos, char playerChar):



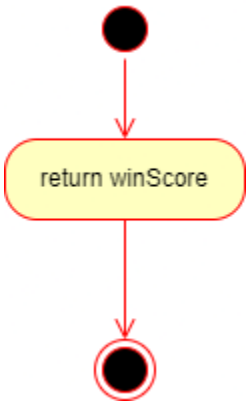
getNumRows():



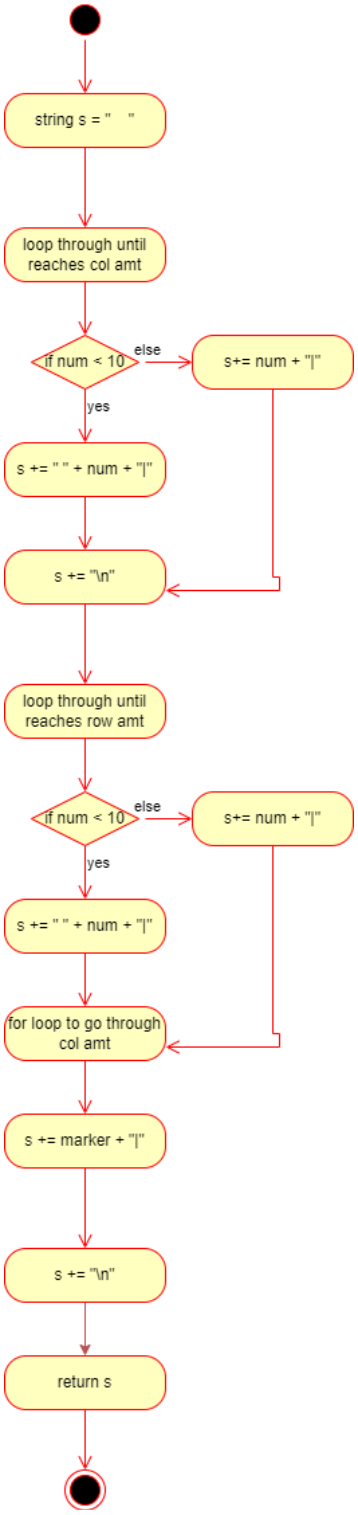
getNumColumns();



getNumToWin():



toString():

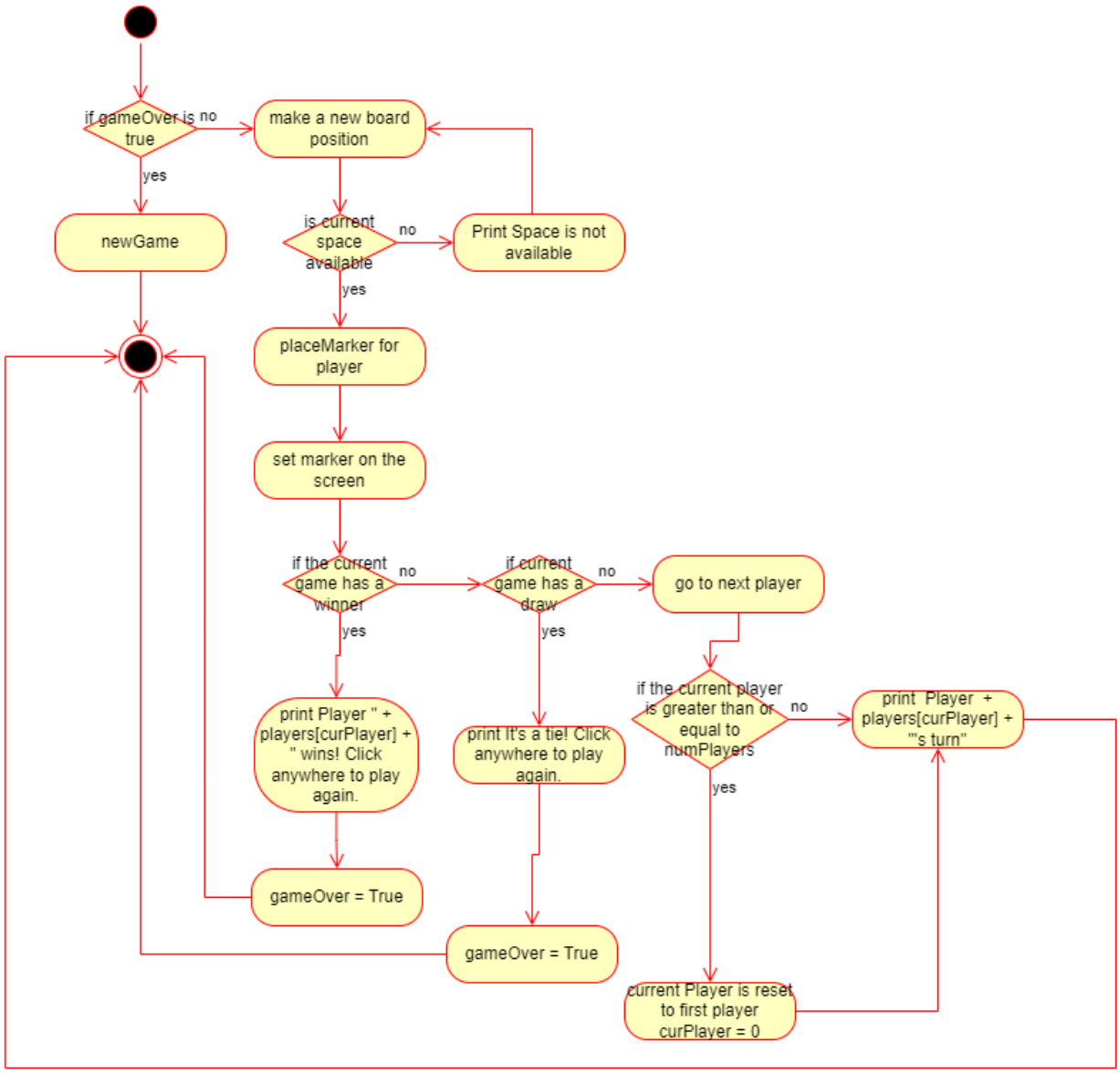


Class 7: TicTacToeController

Class Diagram

TicTacToeController
- curGame: IGameBoard[1] - screen: TicTacToeView[1] + MAX_Players;Static final int[1] - numPlayers: int [1] - players: char[10] - curPlayer: int [1] - gameOver: boolean[1]
+ TicTacToeController(IGameBoard, TicTacToeView, int) + processButtonClick(int, int) : void - newGame(): void

Activity Diagram:
ProcessButtonClick



How To Run:
make: compiles the program
make run: runs the entire program
make clean: deletes all the .class files
make test: complies the program with the test files
make testGB: runs and uses the test file only for the Gamboard class
make testGBmem: runs and uses the test file only for the GameboardMem class.

Test Cases
Details in Project 4.
Constructor:

<table>
<tr>
<td>Input:
row = 3
col = 3
numToWin = 3</td>
<td>Output:
State:
<table><tr><td></td><td>0</td><td>1</td><td>2</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>2</td><td></td><td></td><td></td></tr></table>getNumRows = 3
getNumColumns = 3
getNumToWin = 3</td>
<td>Reason:
This test is unique and distinct because it tests the minimum amount of board requirements needed for a table
Function name:
testGameBoard_min_input</td>
</tr>
<tr>
<td>input:
row = 5
col = 5
numToWin = 5</td>
<td>Output:
<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></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>getNumRows = 5
getNumColumns = 5
getNumToWin = 5</td>
<td>Reason:
This test is unique and distinct because it tests bounds of valid values thats not the min or max
Function name:
testGameBoard_valid</td>
</tr>
</table>

<table>
<tr>
<td>input:
row = 100
col = 100
numToWin = 25</td>
<td>Output:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>…</td><td>99</td></tr>
<tr><td>0</td><td></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>…</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>99</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
getNumRows = 100
getNumColumns = 100
getNumToWin = 25</td>
<td>Reason:
This test is unique and distinct because it tests the maximum amount of board requirements needed for a table
Function name:
testGameBoard_max_input</td>
</tr>
</table>

checkSpace:

<table>
<tr>
<td>Input:
State: 3 to win
<table>
<tr><td></td><td>0</td><td>1</td><td>2</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>2</td><td></td><td></td><td></td></tr>
</table>
pos.getRow = 0
pos.getCol = 2</td>
<td>Output:
checkspace = true

state of the board is unchanged</td>
<td>Reason:
This test is unique and distinct because it is checking a space on an unpopulated board
Function name:
testCheckSpace_empty</td>
</tr>
<tr>
<td>Input:
State: 3 to win
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td></td><td>O</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td></td><td></td></tr>
</table>
pos.getRow = 5
pos.getCol = 5</td>
<td>Output:
checkSpace = false

State: unchanged</td>
<td>Reason:
This space is unique because it is in a spot where a marker is out of bounds
Function:
testCheckSpace_bounds</td>
</tr>
</table>

Input:
State: 3 to win

	0	1	2
0	X	O	X
1	O	X	O
2	X	X	O

pos.getRow = 2
pos.getCol = 1

Output:
checkSpace = false

State: unchanged

Reason:
This space is unique because the board is already full
Function:
testCheckSpace_full

checkHorizontalWin:

Input:
State: 4 to win

	0	1	2	3	4
0					
1					
2	X	X	X	X	
3	O	O	O	X	O
4					

pos.getRow = 2;
pos.getCol = 2;
pos = 'X'

Output:
checkHorizontalWin = true
state is unchanged

Reason:
This test case is unique and distinct because the last X was placed in the middle of the string of 4 consecutive X's as opposed to on the end, so the function needs to count X's on the right and left

Function:
testCheckHorizontalWin_
_Middle

<table>
<tr>
<td>Input:
State: 4 to win
<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></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>X</td><td>X</td><td>X</td><td>X</td><td></td></tr>
<tr><td>3</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow = 2;
pos.getCol = 3;
pos = ‘X’</td>
<td>Output:
checkHorizontalWin = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X was placed at the end of the string of 4 consecutive X’s, so the function needs to count X’s on the left

Function:
testCheckHorizontalWin_End</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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></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>X</td><td>X</td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td>O</td><td>O</td><td>O</td><td>X</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow = 3;
pos.getCol = 3;
pos = ‘X’</td>
<td>Output:
checkHorizontalWin = False
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X was placed at a spot that would not give it a win, thus resulting in a false value
Function:
testCheckHorizontalWin_No_Win</td>
</tr>
</table>

Input: State: 4 to win <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></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>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>3</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>pos.getRow = 2; pos.getCol = 0; pos = 'X'	**Output:** checkHorizontalWin = true state is unchanged	**Reason:** This test case is unique and distinct because the last X was placed at the start of the string of 4 consecutive X's, so the function needs to count X's on the right **Function:** testCheckHorizontalWin_start

checkVerticalWin:

Input: State: 4 to win <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>O</td><td></td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>O</td><td></td><td></td><td></td></tr><tr><td>2</td><td>X</td><td>O</td><td></td><td></td><td></td></tr><tr><td>3</td><td>X</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 = 3; pos.getCol = 0; pos = 'X'	**Output:** checkVerticalWin = true state is unchanged	**Reason:** This test case is unique and distinct because the last marker was at the end of the string of 4 consecutive X's, so the function needs to count X's from the bottom to the top. **Function:** testCheckVerticalWin_end

<table>
<tr>
<td>Input:
State: 4 to win
<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>O</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td>X</td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>X</td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</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.getCol = 0;
pos = ‘X’</td>
<td>Output:
checkVerticalWin = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last marker was at the start of the string of 4 consecutive X’s, so the function needs to count X’s from the top to the bottom.
Function:
testCheckVerticalWin_start</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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>O</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td>X</td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>X</td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</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 = 2;
pos.getCol = 0;
pos = ‘X’</td>
<td>Output:
checkVerticalWin = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last marker was at the middle of the string of 4 consecutive X’s, so the function needs to count X’s from both the top to the bottom and bottom to top.
Function:
testCheckVerticalWin_mid</td>
</tr>
</table>

Input:
State: 4 to win

	0	1	2	3	4
0	X	O			
1	X	O			
2		O			
3	X				
4	X				

pos.getRow = 4;
pos.getCol = 0;
pos = 'X'

Output:
checkVerticalWin = false
state is unchanged

Reason:
This test case is unique and distinct because the last marker was at the end of the string but also does not out put 4 consecutive markers for the string so it would result in a false

Function:
testCheckVerticalWin_No_Win

CheckDiagonalWin:

Input:
State: 4 to win

	0	1	2	3	4
0					
1		X			
2			X		
3	O	O	O	X	
4					X

pos.getRow = 1;
pos.getCol = 1;
pos = 'X'

Output:
checkDiagonalWin = true
state is unchanged

Reason:
This test case is unique and distinct because the last X was placed at the start of the string of 4 diagonal X's, so the function needs to count from the last placed marker, then going down and to the right

Function:
testCheckDiagonalWin_start

<table>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td>O</td><td>O</td><td>O</td><td>X</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td>O</td><td></td><td>X</td></tr>
</table>
pos.getRow = 4;
pos.getCol = 4;
pos = ‘X’</td>
<td>Output:
checkDiagonalWin = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X was placed at the end of the string of 4 diagonal X’s, so the function needs to count from the last placed marker, then going up and to the left

Function:
testCheckDiagonalWin_end</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td>O</td><td>O</td><td>O</td><td>X</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td>O</td><td></td><td>X</td></tr>
</table>
pos.getRow = 2;
pos.getCol = 2;
pos = ‘X’</td>
<td>Output:
checkDiagonalWin = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X was placed at the middle of the string of 4 diagonal X’s, so the function needs to count from the last placed marker, then both going down and to the right and up and to the left

Function:
testCheckDiagonalWin_middle</td>
</tr>
</table>

<table>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>X</td><td></td><td>O</td><td>O</td></tr>
<tr><td>4</td><td>X</td><td></td><td></td><td>O</td><td>O</td></tr>
</table>
pos.getRow = 1;
pos.getCol = 3;
pos = ‘X’</td>
<td>Output:
checkDiagonalWin = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X was placed at the start of the string of 4 diagonal X’s, so the function needs to count from the last placed marker, then going down and to the left

Function:
testCheckDiagonalWin_start_opposite</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>X</td><td></td><td>O</td><td>O</td></tr>
<tr><td>4</td><td>X</td><td></td><td></td><td>O</td><td>O</td></tr>
</table>
pos.getRow = 4;
pos.getCol = 0;
pos = ‘X’</td>
<td>Output:
checkDiagonalWin = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X was placed at the end of the string of 4 diagonal X’s, so the function needs to count from the last placed marker, then going up and to the right

Function:
testCheckDiagonalWin_end_opposite</td>
</tr>
</table>

<table>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>X</td><td></td><td>O</td><td>O</td></tr>
<tr><td>4</td><td>X</td><td></td><td></td><td>O</td><td>O</td></tr>
</table>
pos.getRow = 2;
pos.getCol = 2;
pos = ‘X’</td>
<td>Output:
checkDiagonalWin = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X was placed at the middle of the string of 4 diagonal X’s, so the function needs to count from the last placed marker, then both going down and to the left and up and to the right

Function:
testCheckDiagonalWin_middle_opposite</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td>X</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>X</td><td></td><td>O</td><td>O</td></tr>
<tr><td>4</td><td>X</td><td></td><td></td><td>O</td><td>O</td></tr>
</table>
pos.getRow = 0;
pos.getCol = 4;
pos = ‘X’</td>
<td>Output:
checkDiagonalWin = false
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last marker was at the end of the string but also does not out put 4 consecutive markers for the string so it would result in a false

Function:
testCheckDiagonalWin_No_Win</td>
</tr>
</table>

checkForDraw:

<table>
<tr>
<td>Input:
State: 4 to win
<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></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></td>
<td>Output:
checkForDraw = false
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the board is empty, so the function only would result in a false value
Function name:
testCheckForDraw_empty</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr><tr><td>2</td><td>O</td><td>X</td><td>X</td><td>O</td><td>X</td></tr><tr><td>3</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>4</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr></table></td>
<td>Output:
checkForDraw = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the board is full and there is no win condition through either marker, so the function only would result in a true value
Function name:
testCheckForDraw_full</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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></td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr><tr><td>2</td><td>O</td><td>X</td><td>X</td><td>O</td><td>X</td></tr><tr><td>3</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>4</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr></table>pos.getRow = 0;
pos.getCol = 0;
pos = ‘X’</td>
<td>Output:
checkForDraw = false
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the board’s first position is empty, so the function would result in a false value.
Function name:
testCheckForDraw_first_empty</td>
</tr>
</table>

Input:
State: 4 to win

	0	1	2	3	4
0	X	O	X	O	X
1	O	X	O	X	O
2	O	X	X	O	X
3	X	X	O	O	X
4	O	X	O	X	X

pos.getRow = 3;
pos.getCol = 1;
pos = 'X'

Output:
checkForDraw = false
state is unchanged

Reason:
This test case is unique and distinct because the boards full but has player X won, so the function would result in a false value.
Function name:
testCheckForDraw_Full_Win

whatsAtPos:

Input:
State: 4 to win

	0	1	2	3	4
0					
1					
2					
3					
4					

pos.getRow = 1
pos.getCol = 1

Output:
wharsAtPos = ' '
state is unchanged

Reason:
This test case is unique and distinct because the board is empty, which will make the function return a space.
Function:
testWhatsAtPos_empty

<table>
<tr>
<td>Input:
State: 4 to win
<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>O</td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td>X</td><td></td><td></td><td></td><td></td></tr></table>pos.getRow = 0
pos.getCol = 1</td>
<td>Output:
wharsAtPos = ‘ ‘
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the space is next to a filled column, which will make the function return a space.
Function:
testWhatsAtPos_adjacent_space</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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></td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr><tr><td>2</td><td>O</td><td>X</td><td>X</td><td>O</td><td>X</td></tr><tr><td>3</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>4</td><td>O</td><td>X</td><td>O</td><td>X</td><td>X</td></tr></table>pos.getRow = 0;
pos.getCol = 0;</td>
<td>Output:
wharsAtPos = ‘ ‘
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the board is filled except one, which will make the function return a space.
Function:
testWhatsAtPos_single_space</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</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 = 1
pos.getCol = 1</td>
<td>Output:
wharsAtPos = ‘X ‘
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the board is empty except for one, which will make the function return the marker.
Function:
testWhatsAtPos_Marker</td>
</tr>
</table>

Input: State: 4 to win	**Output:**	**Reason:**
	wharsAtPos = ‘O ‘ state is unchanged	This test case is unique and distinct because the space on the board is filled, which will make the function return the marker. **Function:** testWhatsAtPos_full

	0	1	2	3	4
0	O	O	X	O	X
1	O	X	O	X	O
2	O	X	X	O	X
3	X	O	X	O	X
4	O	X	O	X	X

pos.getRow = 0;
pos.getCol = 0;

<u>isPlayerAtPos:</u>

Input: State: 4 to win	**Output:**	**Reason:**
	isPlayerAtPos = false state is unchanged	This test case is unique and distinct because the board is empty, which will test to see if a single marker is in an empty board. **Function:** testIsPlayerAtPos_empty

	0	1	2	3	4
0					
1					
2					
3					
4					

pos.getRow = 1
pos.getCol = 1
pos = ‘X’

<table>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</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 = 1
pos.getCol = 1
pos = ‘X’</td>
<td>Output:
isPlayerAtPos = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the board will test to see if a single marker is in position
Function:
testIsPlayerAtPos_Marker</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td>O</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 = 2
pos.getCol = 3
pos = ‘O’</td>
<td>Output:
isPlayerAtPos = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the board will test to see if the function can differentiate between two different markers
Function:
testIsPlayerAtPos_Two_Markers</td>
</tr>
</table>

<table>
<tr>
<td>Input:
State: 4 to win
<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></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td>O</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 = 2
pos.getCol = 3
pos = ‘X’</td>
<td>Output:
isPlayerAtPos = false
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the pos is different and marker is already taken by a different marker
Function:
testIsPlayerAtPos_Two_Markers_False</td>
</tr>
<tr>
<td>Input:
State: 5 to win
<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>A</td><td>S</td><td>D</td><td>F</td><td>G</td></tr>
<tr><td>1</td><td>S</td><td>D</td><td>F</td><td>G</td><td>A</td></tr>
<tr><td>2</td><td>D</td><td>F</td><td>G</td><td>A</td><td>S</td></tr>
<tr><td>3</td><td>F</td><td>G</td><td>A</td><td>S</td><td>D</td></tr>
<tr><td>4</td><td>A</td><td>S</td><td>D</td><td>F</td><td>G</td></tr>
</table>
pos.getRow = 3
pos.getCol = 3
pos = ‘S’</td>
<td>Output:
isPlayerAtPos = true
state is unchanged</td>
<td>Reason:
This test case is unique and distinct because the pos is filled and the board is filled
Function:
testIsPlayerAtPos_filled</td>
</tr>
</table>

placeMarker:

Input:
State: 4 to win

	0	1	2	3
0				
1				
2		X		
3	O			

pos.getRow = 1;
pos.getCol = 2;
p = 'A'

Output:
State:

	0	1	2	3
0				
1			A	
2		X		
3	O			

Reason:
This test case is unique and distinct because I am placing a marker representing a player who has not been placed on this board before.

Function:
testPlaceMarker_col_not_empty

Input:
State: 4 to win

	0	1	2	3
0				
1				
2				
3				

pos.getRow = 1;
pos.getCol = 2;
p = 'X'

Output:
State:

	0	1	2	3
0				
1		X		
2				
3				

Reason:
This test case is unique and distinct because I am placing a marker representing a player when the board is empty.

Function:
testPlaceMarker_empty

Input:
State: 4 to win

	0	1	2	3
0	X	O	X	O
1	O	X		X
2	X	O	X	O
3	O	X	X	O

pos.getRow = 1;
pos.getCol = 2;
p = 'O'

Output:
State:

	0	1	2	3
0	X	O	X	O
1	O	X	O	X
2	X	O	X	O
3	O	X	X	O

Reason:
This test case is unique and distinct because I am placing a marker representing a player when the board is filled except one space.
Function:
testPlaceMarker_single_space

<table>
<tr>
<td>Input:
State: 4 to win
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>X</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>pos.getRow = 0;
pos.getCol = 0;
p = ‘X’</td>
<td>Output:
State is unchanged</td>
<td>Reason:
This test case is unique and distinct because I am placing a marker in the corner

Function:
testPlaceMarker_Corner</td>
</tr>
<tr>
<td>Input:
State: 4 to win
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td>X</td><td></td><td></td></tr><tr><td>3</td><td>O</td><td></td><td></td><td></td></tr></table>pos.getRow = 1;
pos.getCol = 2;
p = ‘X’</td>
<td>Output:
State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td>X</td><td></td><td></td></tr><tr><td>3</td><td>O</td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
This test case is unique and distinct because I am placing a marker representing a player who has been placed on this board before.

Function:
testPlaceMarker_same_marker</td>
</tr>
</table>