CPSC 2150 Project Report

Simon Hughes

Requirements Analysis

Functional Requirements:

1. As a player, I can insert a chip in a column so I can play the game.
2. As a player, I can redo my turn if I add a chip to a full column, so the game is fair.
3. As a player, I can use my turn after another player, so everyone gets a turn.
4. As a player, I can redo my turn if I insert a chip in an invalid column, so the game is fair
5. As a player, I can see the gameboard after my turn so I can play the game
6. As a player, I can see whose turn it is so I know when to play my turn
7. As a player, I can win the game by having a specified amount of pieces vertically so that the game has a winner
8. As a player, I can win the game by having a specified amount of pieces horizontally so that the game has a winner
9. As a player, I can win the game by having a specified amount of pieces diagonally so that the game has a winner
10. As a player, I can opt to play again so I can keep playing ConnectX
11. As a player, I can know when the game results in a tie, so I know when the game has ended
12. As a player, I can choose how many players will be playing the game, so everyone gets a turn
13. As a player, I can change how many rows the game will have, so I can play the game I want
14. As a player, I can change how many columns the game will have, so I can play the game I want
15. As a player, I can change how many pieces in a row to win, so I can play the game I want

Non-Functional Requirements

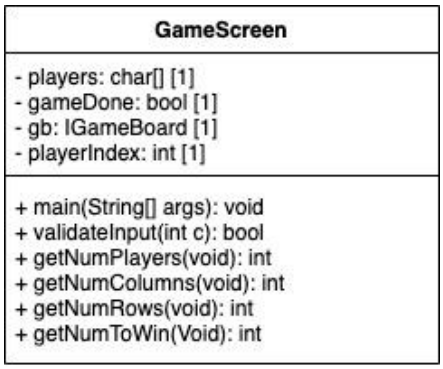
1. The program takes in arguments from the GUI
2. The program must be coded in Java
3. The game board must have a user specified amount of rows and a user specified amount of columns

4. The game board at [0][0] must be the bottom left
5. The program must display to the GUI
6. The program must continue until there is either a tie or someone has won
7. The code must follow the Model View Controller pattern

System Design

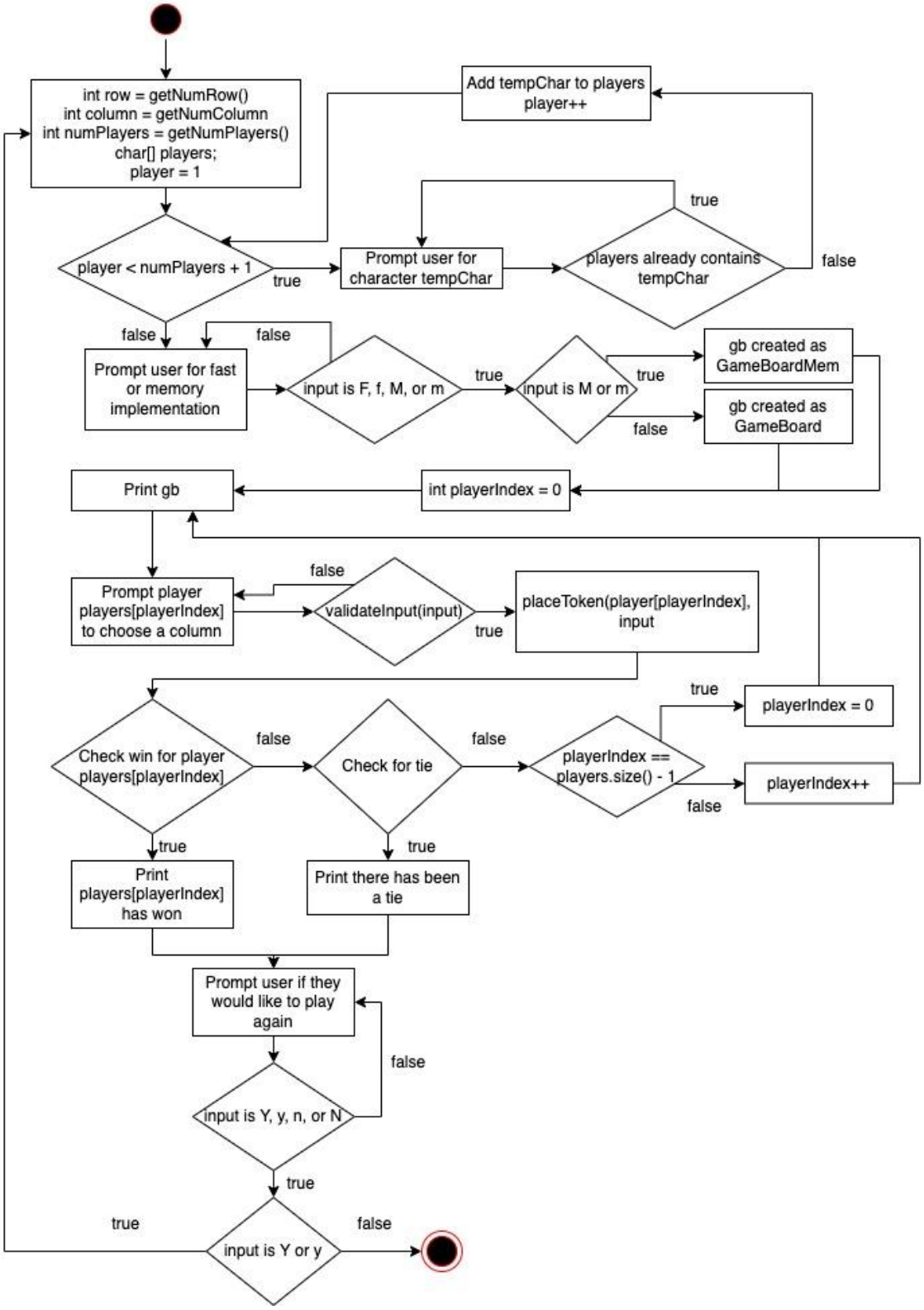
Class 1: GameScreen

Class Diagram

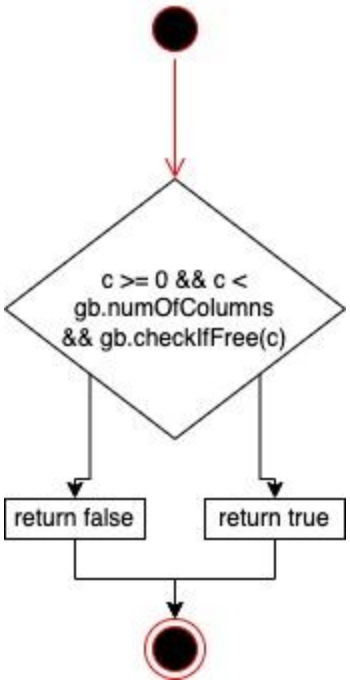


Activity Diagrams

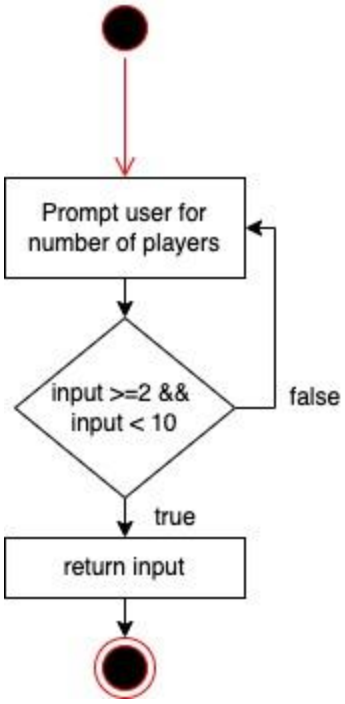
main


int row = getNumRow()
int column = getNumColumn
int numPlayers = getNumPlayers()
char[] players;
player = 1
player < numPlayers + 1
true
Prompt user for character tempChar
players already contains tempChar
true
false
Add tempChar to players
player++
false
false
Prompt user for fast or memory implementation
input is F, f, M, or m
true
input is M or m
true
gb created as GameBoardMem
false
gb created as GameBoard
int playerIndex = 0
Print gb
Prompt player players[playerIndex] to choose a column
false
validateInput(input)
true
placeToken(player[playerIndex], input
true
playerIndex = 0
Check win for player players[playerIndex]
false
Check for tie
false
playerIndex == players.size() - 1
false
playerIndex++
true
Print players[playerIndex] has won
true
Print there has been a tie
Prompt user if they would like to play again
input is Y, y, n, or N
false
true
input is Y or y
true
false

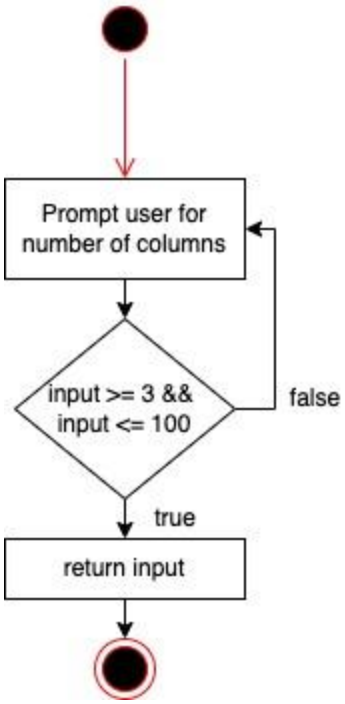
validateInput



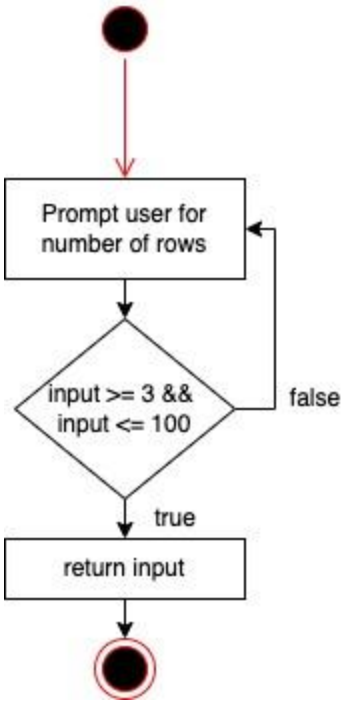
getNumPlayers



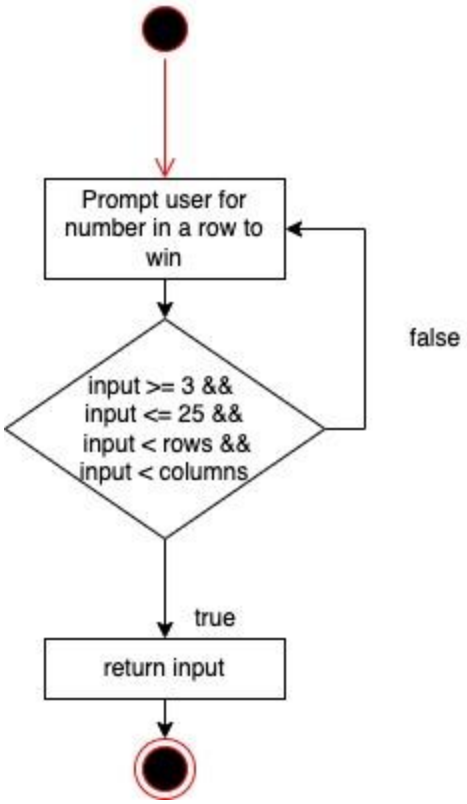
getNumColumns



getNumRows

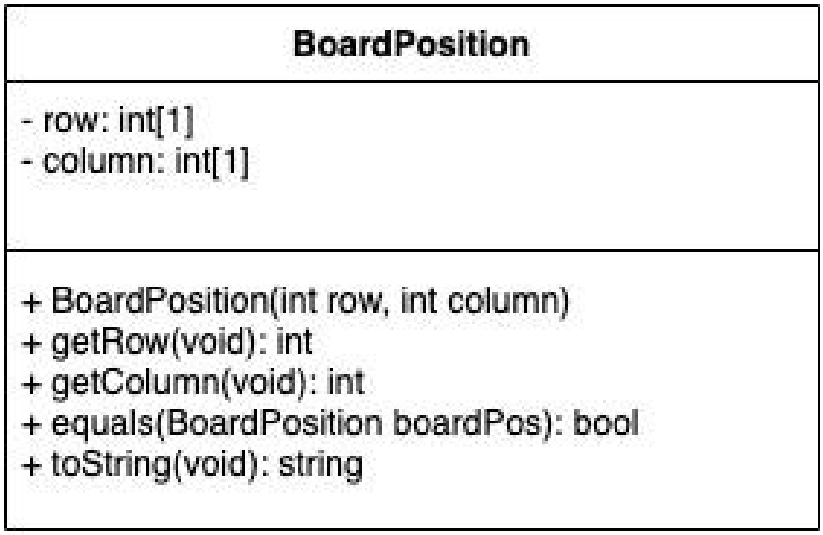


getNumToWin



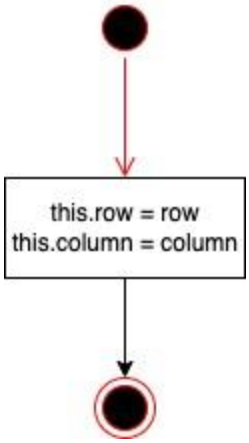
Class 2: BoardPosition

Class Diagram

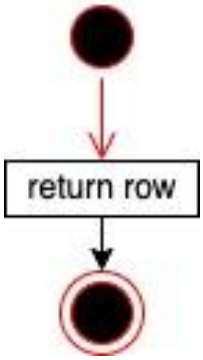


Activity Diagrams

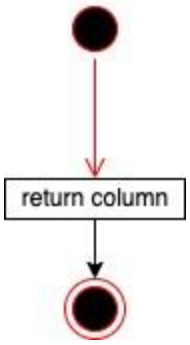
BoardPosition



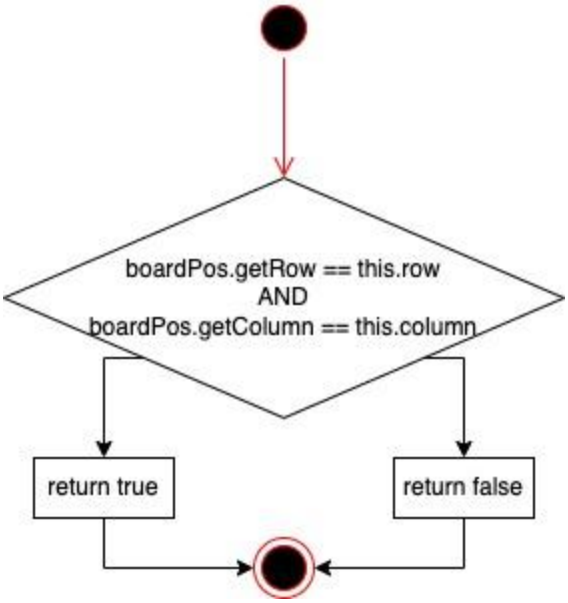
getRow



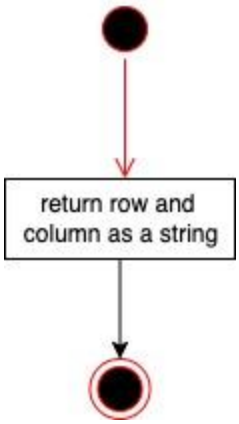
getColumn



equals



toString

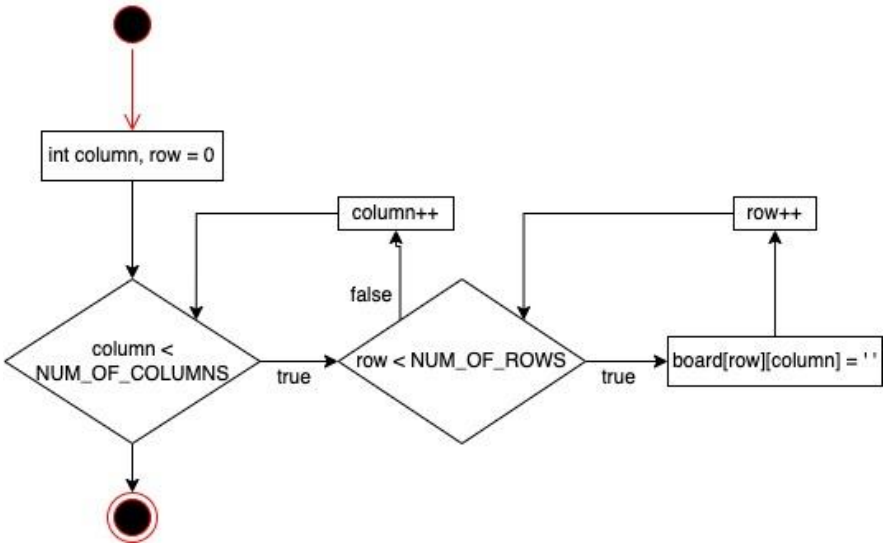


Class 3: GameBoard

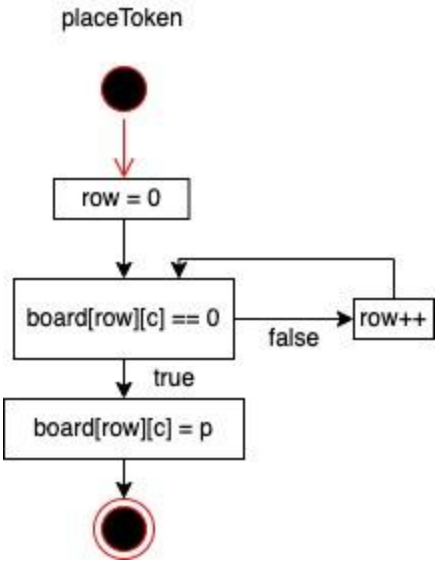
Class Diagram

Activity Diagrams

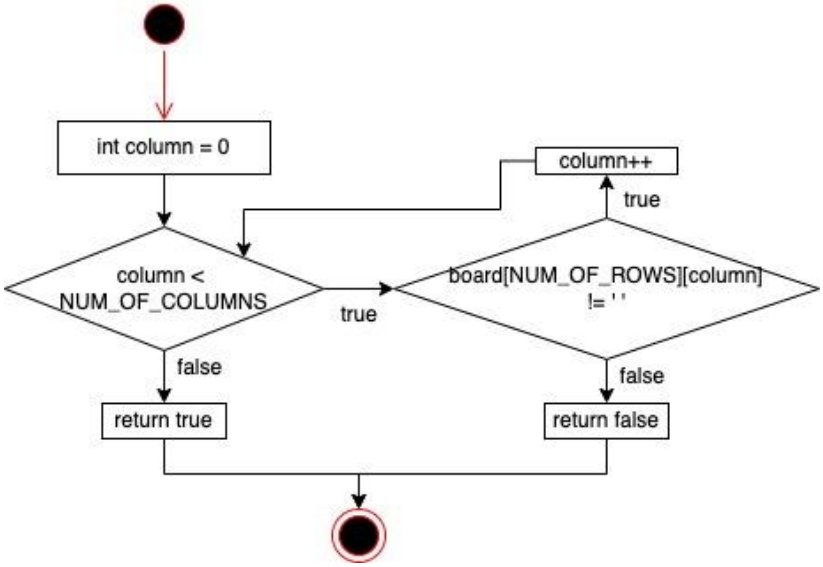
GameBoard



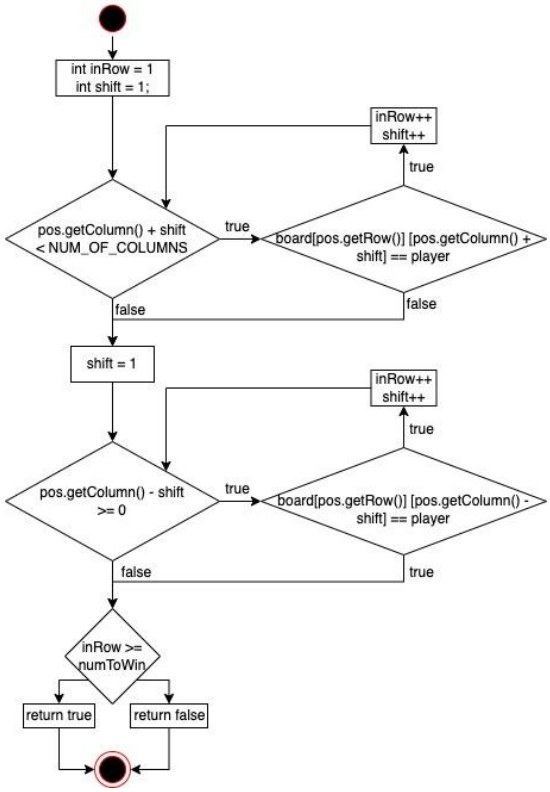
placeToken



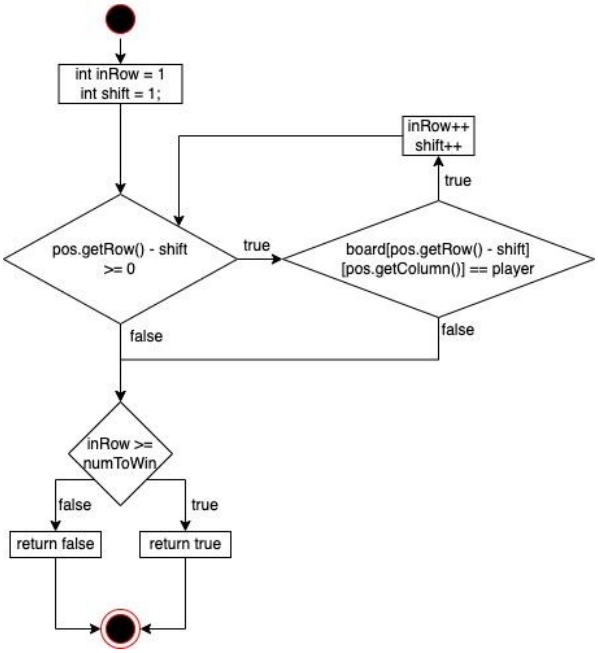
checkTie



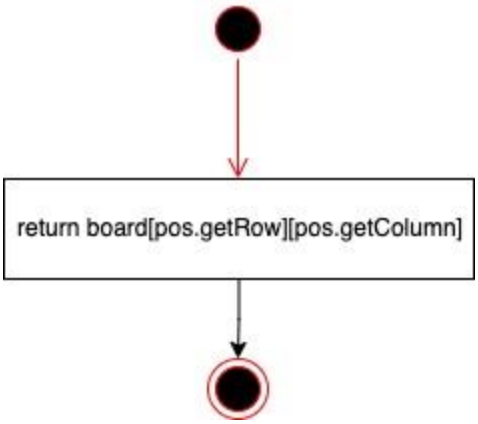
checkHorizWin



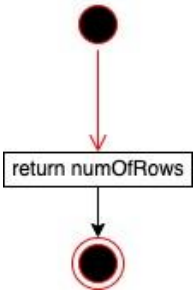
checkVertWin



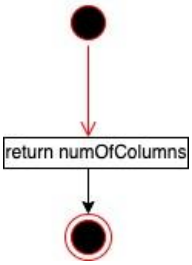
whatsAtPos



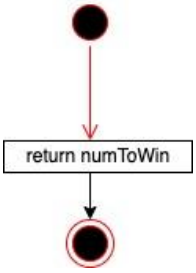
getNumRows



getNumColumns

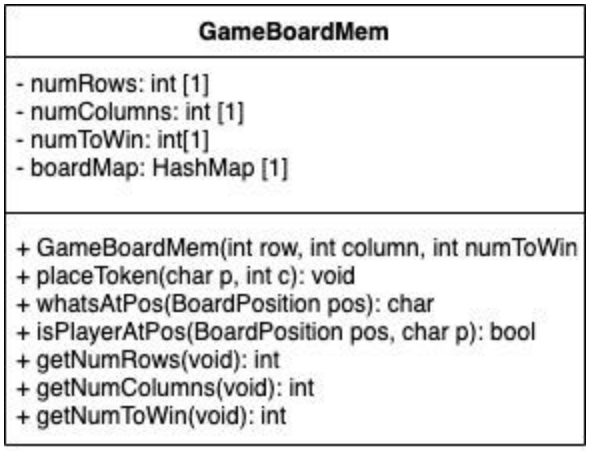


getNumToWin



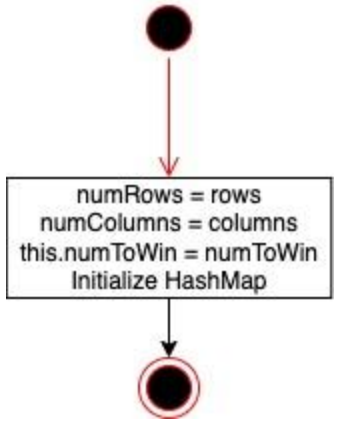
Class 4: GameBoardMem

Class Diagram:

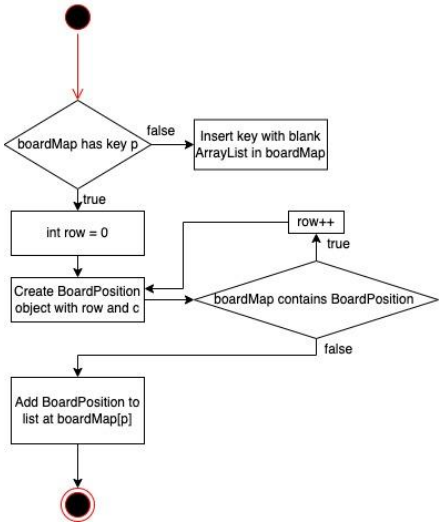


Activity Diagram:

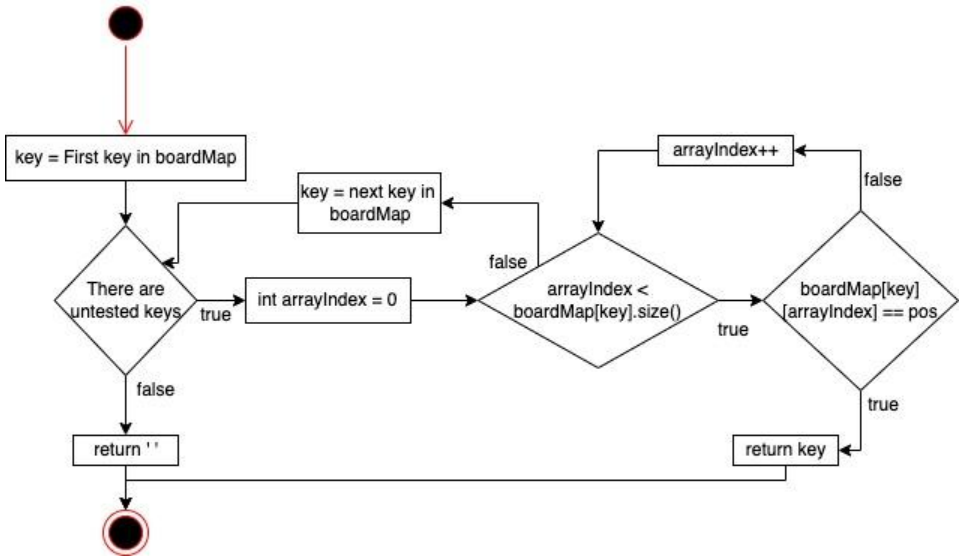
GameBoardMem



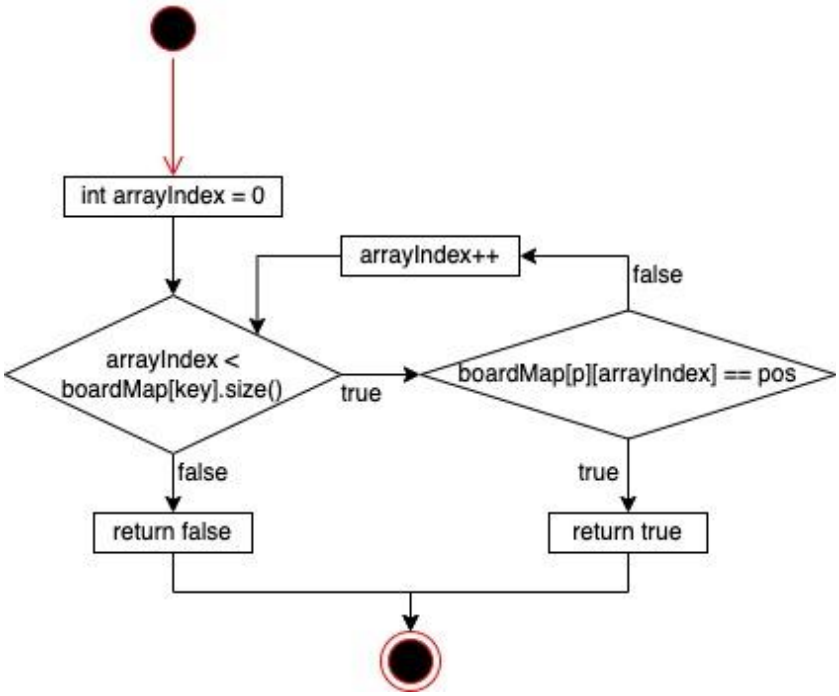
placeToken



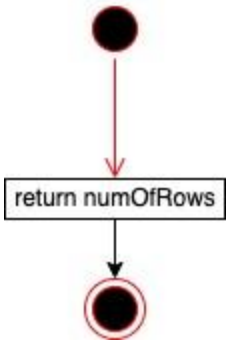
whatsAtPos



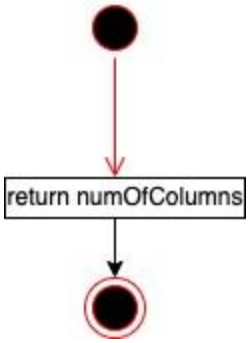
isPlayerAtPos



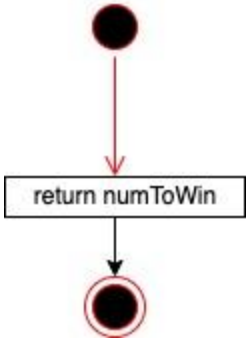
getNumRows



getNumColumns



getNumToWin



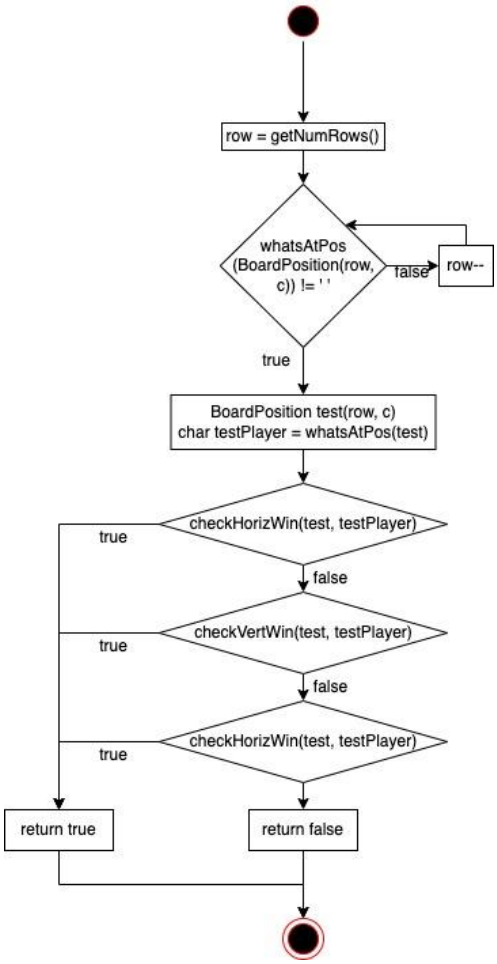
Interface

<<interface>>
IGameBoard

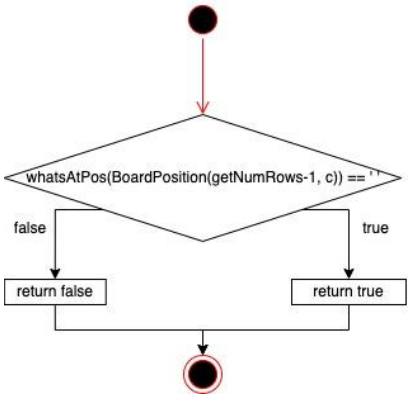
+ checkIfFree(int c): bool
+ placeToken(char p, int c): void
+ checkForWin(void): bool
+ checkTie(void): bool
+ checkVertWin(BoardPosition pos, char p): bool
+ checkHorizWin(BoardPosition pos, char p): bool
+ checkDiagWin(BoardPosition pos, char p): bool
+ whatsAtPos(BoardPosition pos): char
+ isPlayerAtPos(BoardPosition pos, char p): bool
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int

Class Diagrams:

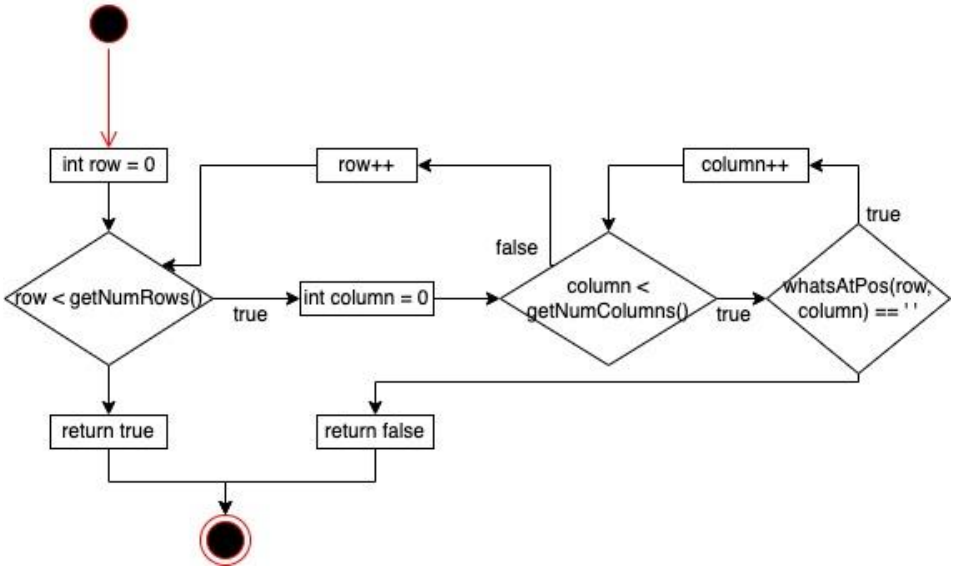
checkForWin



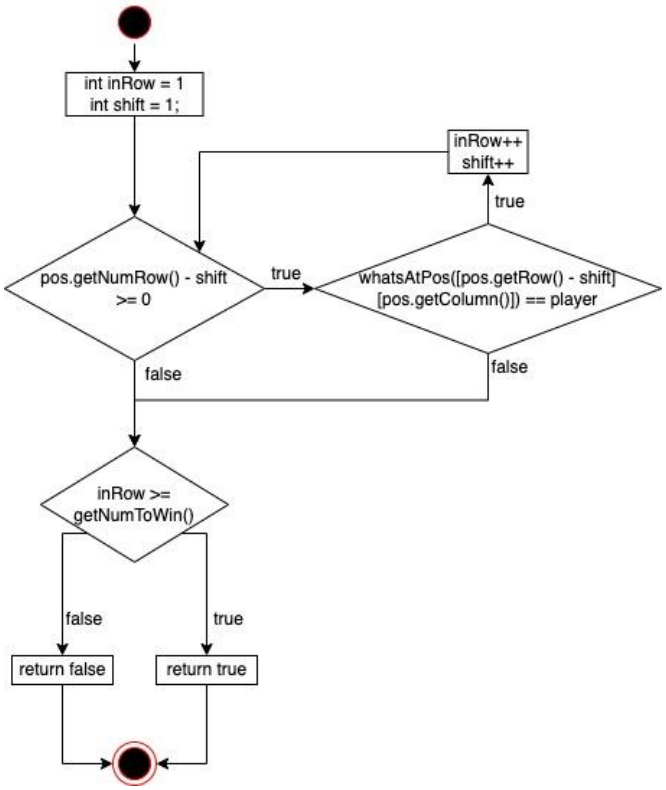
checkIfFree



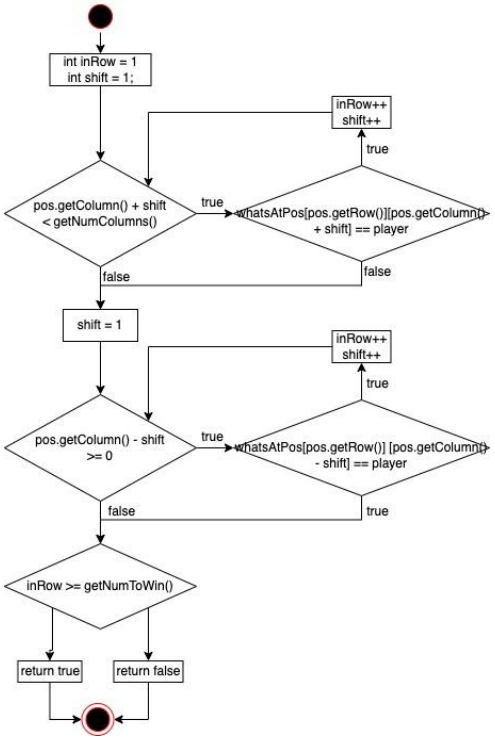
checkTie



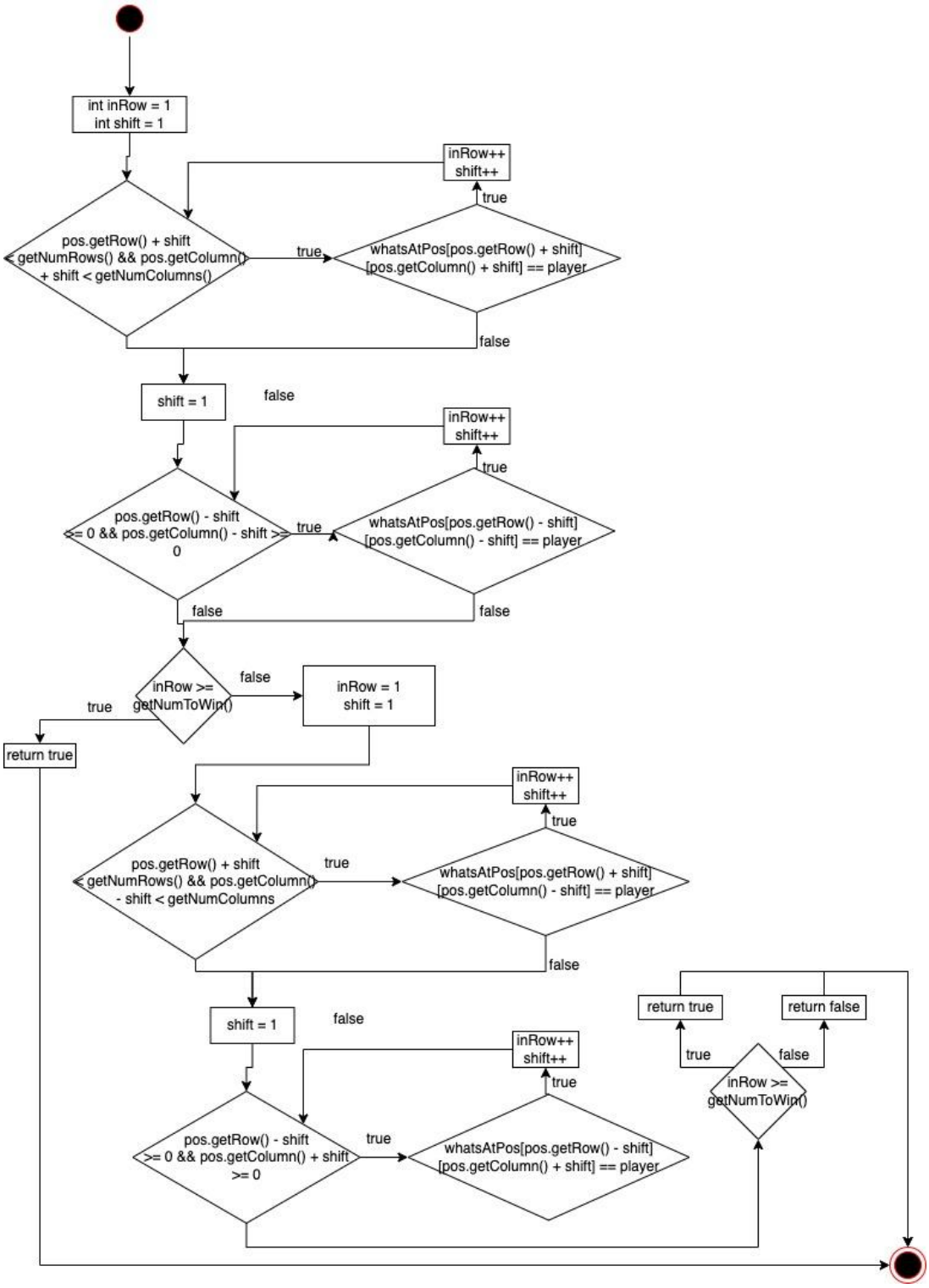
checkVertWin



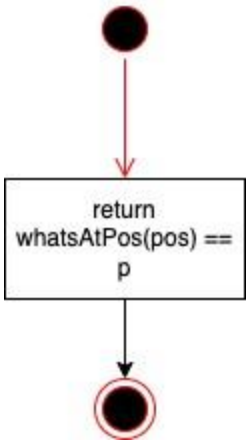
checkHorizWin



checkDiagWin

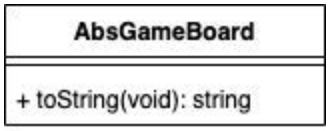


isPlayerAtPos

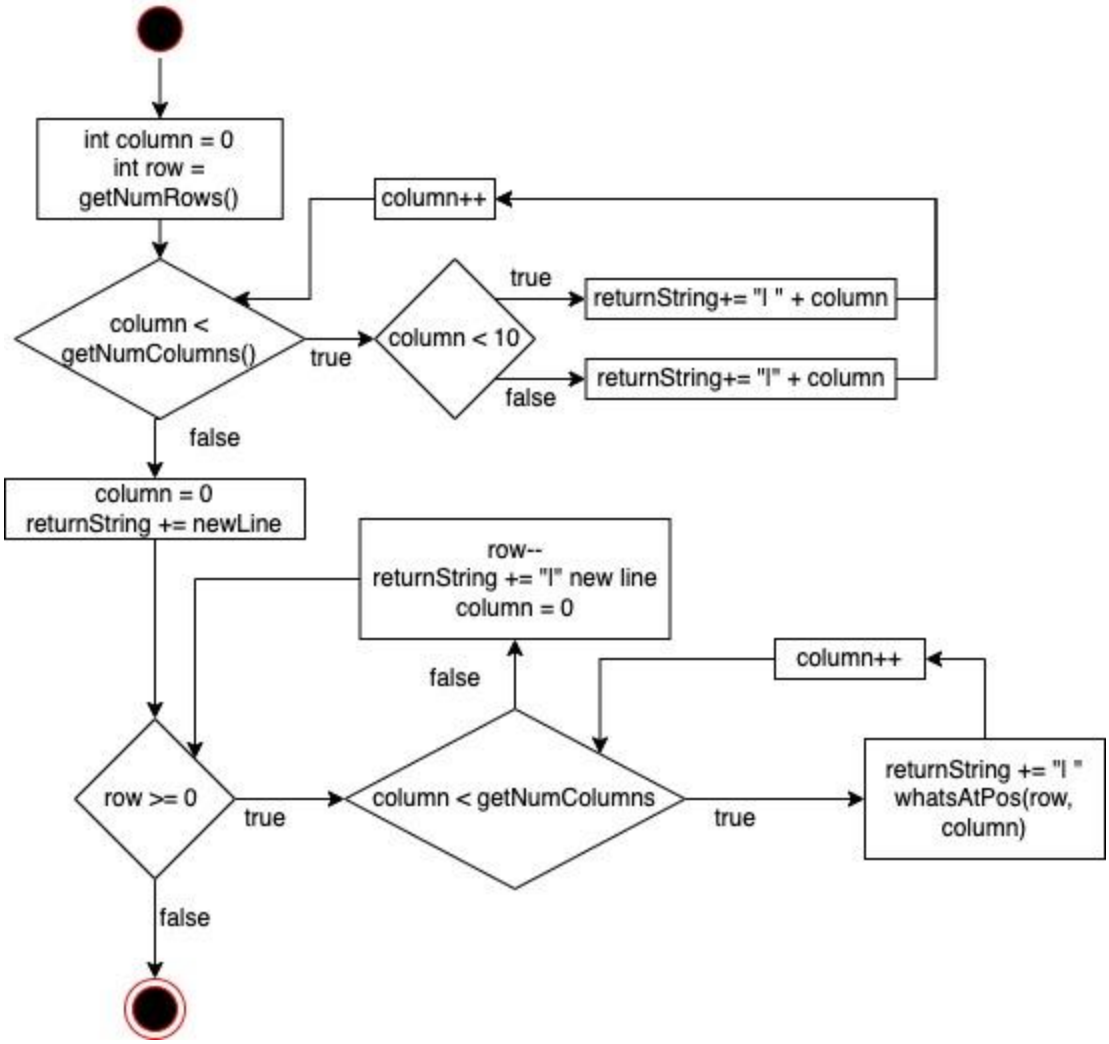


Abstract Class

Class Diagram:



Activity Diagram:

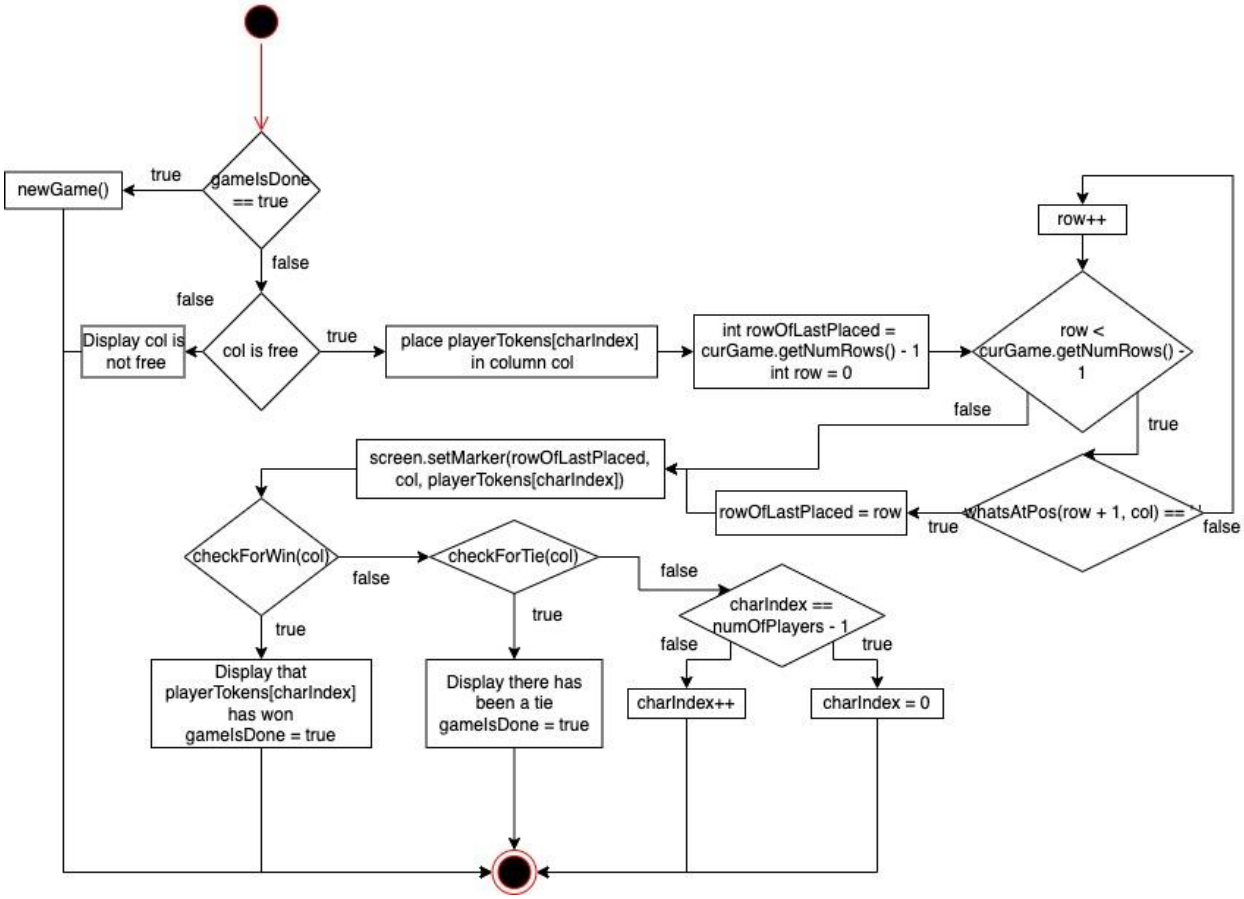


Class 5: ConnectXController

Class Diagram:

ConnectXController
- curGame: IGameBoard[1] - screen: ConnectXView[1] - charIndex: int [1] - gameIsOver: boolean [1] + MAX_PLAYERS: int[1] - playerTokens: char[] [1] - numPlayers: int[1]
+ ConnectXController(IGameBoard, ConnectXView, int) + processButtonClick(int): void - newGame(void): void

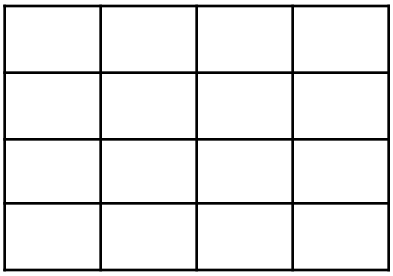
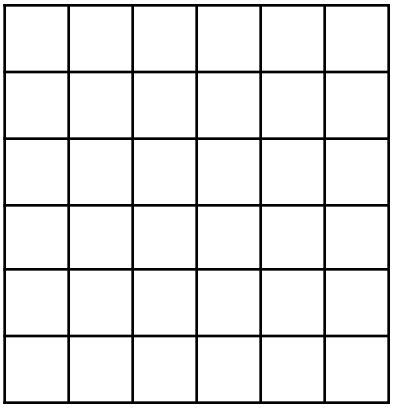
Activity Diagrams:



Deployment Instructions:
This is an IntelliJ Project, so open the file with IntelliJ. Then, press the green button in the top right.

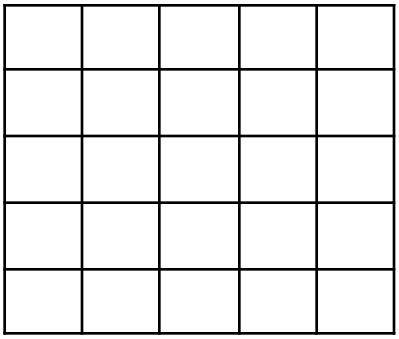
Test Cases

`GameBoard(int rows, int columns, int numToWin)`

Input: rows = 4 columns = 4 numToWin = 3	**Output:** State:  numRows = 4 numColumns = 4 this.numToWin = 3	**Reason:** This test case tests the lower boundary of the dimensions of the board. (Technically, 3x3 is lower boundary but because numToWin has to be greater than row and column and must be greater than 3, 4x4 is the smallest) **Function Name:** test_constructor_4x4_3win
Input: rows = 100 columns = 100 numToWin = 25	**Output:** State: A 100x100 empty board numRows = 100 numColumns = 100 this.numToWin = 25	**Reason:** This test case tests the upper boundary of the dimensions of the board **Function Name:** test_constructor_100x100_25win
Input: rows = 6 columns = 6 numToWin = 4	**Output:** State:  numRows = 6 numColumns = 6 this.numToWin = 4	**Reason:** This test case tests a routine construction of GameBoard **Function Name:** test_constructor_6x6_4win

`boolean checkIfFree(int c)`

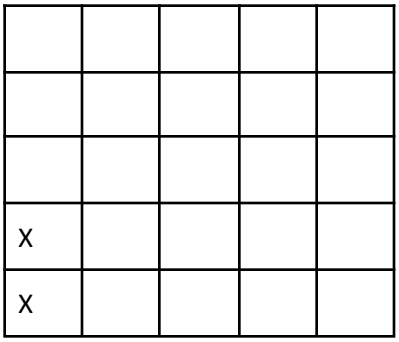
<table>
<tr>
<td>Input:

State:
 

c = 0</td>
<td>Output:

checkIfFree = true
State of board is unchanged</td>
<td>Reason: This test tests checkIfFree when the column is completely empty

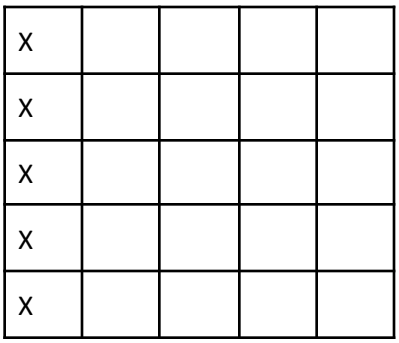
Function Name:
test_checkIfFree_empty</td>
</tr>
<tr>
<td>Input:

State:
 

c = 0</td>
<td>Output:

checkIfFree = true
State of board is unchanged</td>
<td>Reason: This test tests checkIfFree when the column is not empty but is not full

Function Name:
test_checkIfFree_populated</td>
</tr>
<tr>
<td>Input:

State:
 

c = 0</td>
<td>Output:

checkIfFree = false
State of board is unchanged</td>
<td>Reason: This test tests checkIfFree when the column is full

Function Name:
test_checkIfFree_full</td>
</tr>
</table>

`boolean checkHorizWin(BoardPosition pos, char p)`

<table>
<tr>
<td>Input:

State: (number to win = 4)

<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td></td><td></td></tr>
</table>

pos.getRow = 0
pos.getColumn = 2
p = ‘X’</td>
<td>Output:

checkHorizWin = false
State of board is unchanged</td>
<td>Reason: This test tests checkHorizWin when there are tokens of the same character in a row, but less than number to win

Function Name:
test_checkHorizWin_noWin</td>
</tr>
<tr>
<td>Input:

State: (number to win = 4)

<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr>
</table>

pos.getRow = 0
pos.getColumn = 3
p = ‘X’</td>
<td>Output:

checkHorizWin = true
State of board is unchanged</td>
<td>Reason: This test tests checkHorizWin when there are numToWin tokens in a row of the same character

Function Name:
test_checkHorizWin_win</td>
</tr>
</table>

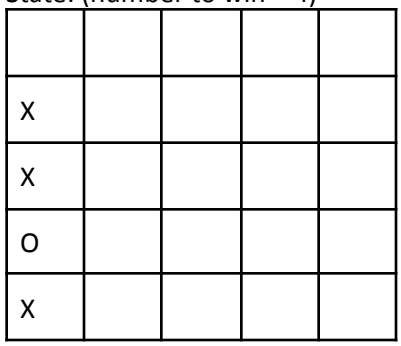
Input: State: (number to win = 4) 					 pos.getRow = 0 pos.getColumn = 2 p = ‘X’	**Output:** checkHorizWin = true State of board is unchanged	**Reason:** While this is similar to the first test, the most recent token is dropped in the middle of the row, so the method must check forwards and backwards **Function Name:** test_checkHorizWin_winInMiddle
X	X	X	X				

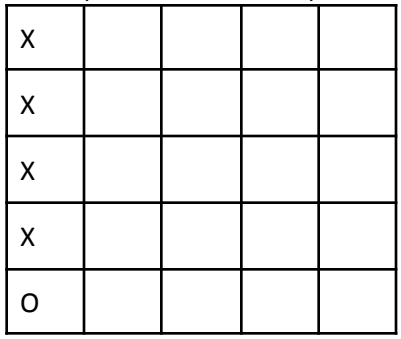
Input: State: (number to win = 4) 					 pos.getRow = 0 pos.getColumn = 3 p = ‘X’	**Output:** checkHorizWin = false State of board is unchanged	**Reason:** This test tests checkHorizWin when there are numToWin tokens in a row, but the tokens are different players **Function Name:** test_checkHorizWin_multiplePlayers
O	X	X	X				

`boolean checkVertWin(BoardPosition pos, char p)`

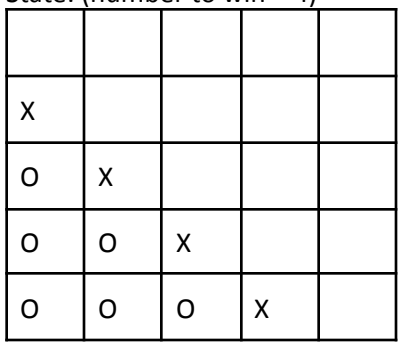
Input: State: (number to win = 4) 					**Output:** checkVertWin = false State of board is unchanged	**Reason:** This test tests checkVertWin when there are tokens in a row vertically, but not >= numToWin. **Function Name:** test_checkVertWin_noWin
X						
X						
X						
pos.getRow = 2 pos.getColumn = 0 p = 'X'						

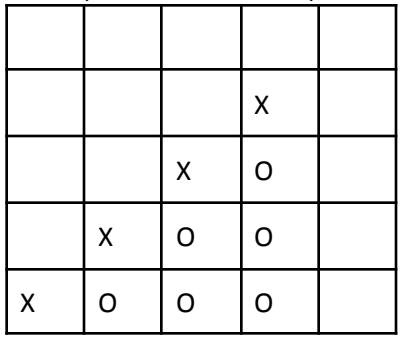
Input: State: (number to win = 4) 					**Output:** checkVertWin = true State of board is unchanged	**Reason:** This test tests checkVertWin when there are numToWin tokens in a row of the same player **Function Name:** test_checkVertWin_win
X						
X						
X						
X						
pos.getRow = 3 pos.getColumn = 0 p = 'X'						

Input: State: (number to win = 4)  pos.getRow = 3 pos.getColumn = 0 p = ‘X’	**Output:** checkVertWin = false State of board is unchanged	**Reason:** This test tests checkVertWin when there are numToWin tokens in a row but the tokens are not the same player **Function Name:** test_checkVertWin_multiplePlayers

Input: State: (number to win = 4)  pos.getRow = 4 pos.getColumn = 0 p = ‘X’	**Output:** checkVertWin = true State of board is unchanged	**Reason:** This test is similar to the second test, but checkVertWin must check above it (out of bounds) which is a challenging case **Function Name:** test_checkVertWin_winAtTopRow

`boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 4)  pos.getRow = 3 pos.getColumn = 0 p = 'X'	**Output:** checkDiagWin = true State of board is unchanged	**Reason:** This test case is unique because it is checking a win going up and left **Function Name:** test_checkDiagWin_upLeft

Input: State: (number to win = 4)  pos.getRow = 3 pos.getColumn = 3 p = 'X'	**Output:** checkDiagWin = true State of board is unchanged	**Reason:** This test case is unique because it is checking a win going down and left **Function Name:** test_checkDiagWin_downLeft

Input:

State: (number to win = 4)

			X	
		X	O	
	X	O	O	
X	O	O	O	

pos.getRow = 0
pos.getColumn = 0
p = 'X'

Output:

checkDiagWin = true
State of board is unchanged

Reason: This test case is unique because it is checking for a win up and right.

Function Name:
test_checkDiagWin_upRight

Input:

State: (number to win = 4)

X				
O	X			
O	O	X		
O	O	O	X	

pos.getRow = 0
pos.getColumn = 3
p = 'X'

Output:

checkDiagWin = true
State of board is unchanged

Reason: This test case is unique because it is checking a win going up and left

Function Name:
test_checkDiagWin_downLeft

Input:

State: (number to win = 4)

X				
O	X			
O	O	X		
O	O	O	X	

pos.getRow = 2
pos.getColumn = 1
p = ‘X’

Output:

checkDiagWin = true
State of board is unchanged

Reason: This test case is unique because the last X placed is at 2,1 which means the method has to check both up and left and down and right

Function Name:
test_checkDiagWin_winInMiddle_upLeft_downRight

Input:

State: (number to win = 4)

			X	
		X	O	
	X	O	O	
X	O	O	O	

pos.getRow = 2
pos.getColumn = 2
p = ‘X’

Output:

checkDiagWin = true
State of board is unchanged

Reason: This test case is unique because the last X placed is at 2,2, which means the method has to check both up and right and down and left

Function Name:
test_checkDiagWin_upRight_downLeft

Input:

State: (number to win = 4)

		X		
	X	O		
X	O	O		

pos.getRow = 2
pos.getColumn = 2
p = ‘X’

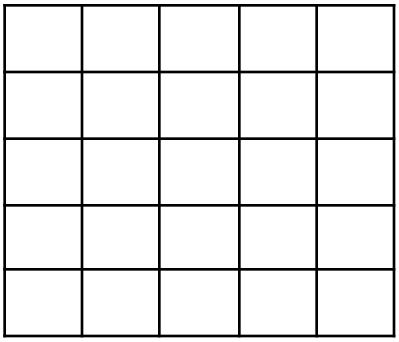
Output:

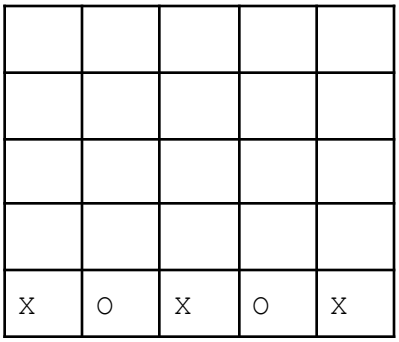
checkDiagWin = false
State of board is unchanged

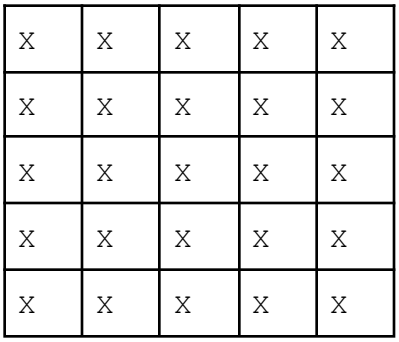
Reason: This test case is unique because it there are not numToWin tokens in a row diagonally

Function Name:
test_checkDiagWin_noWin

`boolean checkTie()`

Input: State: 	**Output:** checkTie = false State of board is unchanged	**Reason:** This test case is unique because it is checking for a tie when the board is empty **Function Name:** test_checkTie_empty

Input: State: 	**Output:** checkTie = false State of board is unchanged	**Reason:** This test case is unique because it is checking for a tie when the board is not empty but not full **Function Name:** test_checkTie_populated

Input: State: 	**Output:** checkTie = true State of board is unchanged	**Reason:** This test case is unique because it is checking for a tie when the board is completely full **Function Name:** test_checkTie_full

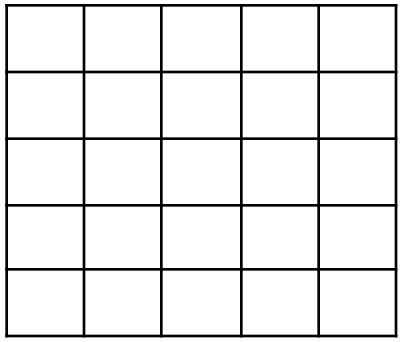
<table>
<tr>
<td>Input:

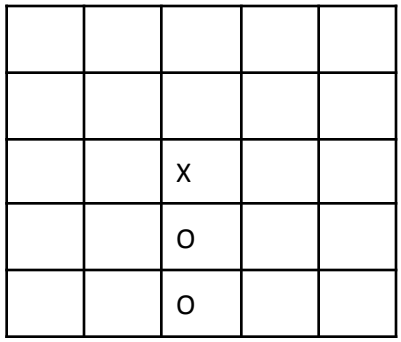
State:
<table>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table></td>
<td>Output:

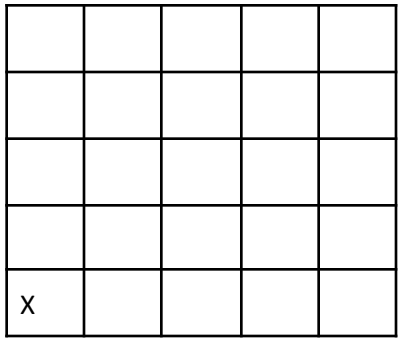
checkTie = false
State of board is unchanged</td>
<td>Reason: This test case is unique because it is checking for a tie when the board is one token away from being completely full, i.e. a boundary case.

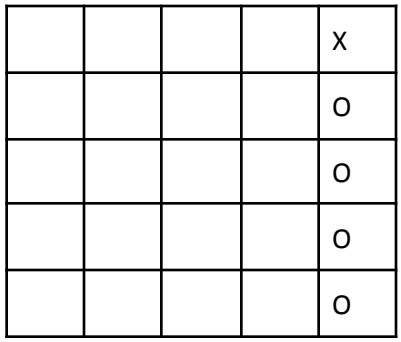
Function Name:
test_checkTie_almostFulll</td>
</tr>
</table>

`char whatsAtPos(BoardPosition pos)`

Input:	**Output:**	**Reason:** This test case is unique because the board is empty
State:  pos.getRow = 2 pos.getColumn = 2	whatsAtPos = ‘ ‘ State of board is unchanged	**Function Name:** test_whatsAtPos_empty

Input:	**Output:**	**Reason:** This test case is unique because the character at pos is X. This is a routine case
State:  pos.getRow = 2 pos.getColumn = 2	whatsAtPos = X State of board is unchanged	**Function Name:** test_whatsAtPos_X

Input: State:  pos.getRow = 0 pos.getColumn = 0	**Output:** whatsAtPos = X State of board is unchanged	**Reason:** This test case is unique because it is testing the lower boundary (pos.getRow >= 0, pos.getColumn >= 0) **Function Name:** test_whatsAtPos_lowerBoundary

Input: State:  pos.getRow = 4 pos.getColumn = 4	**Output:** whatsAtPos = X State of board is unchanged	**Reason:** This test case is unique because it is testing the higher boundary (pos.getRow < numRows, pos.getColumn < numColumns) **Function Name:** test_whatsAtPos_higherBoundary

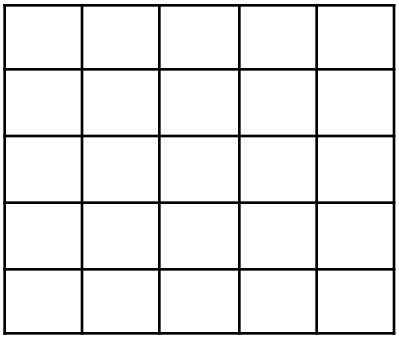
<table>
<tr>
<td>Input:

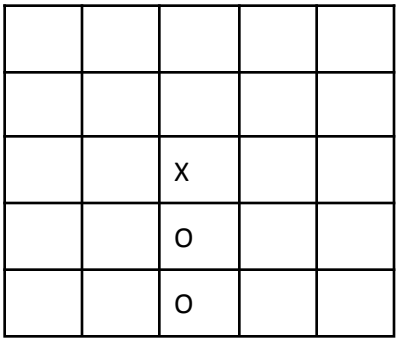
State:
<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>
pos.getRow = 1
pos.getColumn = 2</td>
<td>Output:

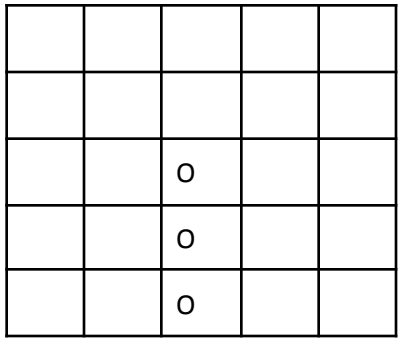
whatsAtPos = ‘ ‘
State of board is unchanged</td>
<td>Reason: This test case is unique because the board is not empty but the cell at pos is

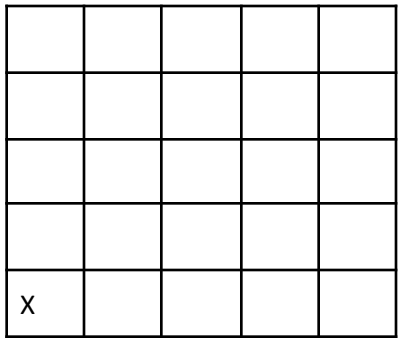
Function Name:
test_whatsAtPos_populated_emptyPos</td>
</tr>
</table>

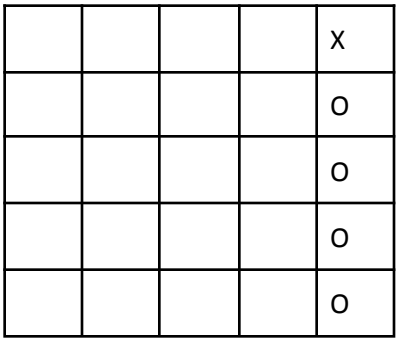
`boolean isPlayerAtPos(BoardPosition pos, char p)`

Input: State:  pos.getRow = 0 pos.getColumn = 0 p = X	**Output:** isPlayerAtPos = false State of board is unchanged	**Reason**: This test case is unique because it is using isPlayerAtPos on an empty board **Function Name:** test_isPlayerAtPos_empty

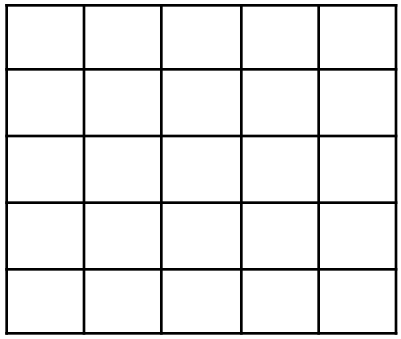
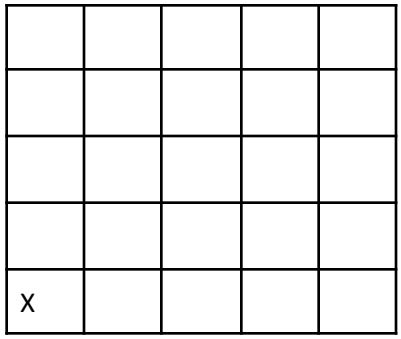
Input: State:  pos.getRow = 2 pos.getColumn = 2 p = X	**Output:** isPlayerAtPos = true State of board is unchanged	**Reason**: This test case is unique because the character getting checked is at pos. This is a routine case. **Function Name:** test_isPlayerAtPos_xAt22

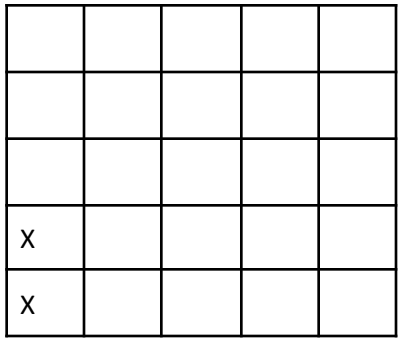
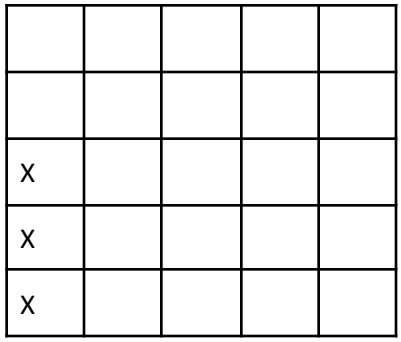
Input: State:  pos.getRow = 2 pos.getColumn = 2 p = X	**Output:** isPlayerAtPos = false State of board is unchanged	**Reason**: This test case is unique because there is a character at the pos being tested, but it is not the right one **Function Name:** test_isPlayerAtPos_oAt22

Input: State:  pos.getRow = 0 pos.getColumn = 0 p = X	**Output:** isPlayerAtPos = true State of board is unchanged	**Reason:** This test case is unique because it is testing the lower bounds of isPlayerAtPos (pos.getColumn must be >= 0 and pos.getRow must be >= 0) **Function Name**: test_isPlayerAtPos_xAt00

Input: State:  pos.getRow = 4 pos.getColumn = 4 p = X	**Output:** isPlayerAtPos = true State of board is unchanged	**Reason:** This test case is unique because it is testing the higher bounds of isPlayerAtPos (pos.getColumn must be <= numOfColumns and pos.getRow must be <= numOfRows) **Function Name**: test_isPlayerAtPos_xAt44

`void placeToken(char p, int c)`

Input: State:  c = 0 p = X	**Output:** State: 	**Reason:** This test is unique because it is testing place token on an empty board **Function Name:** test_placeToken_empty

Input: State:  c = 0 p = X	**Output:** State: 	**Reason:** This test is unique because it is testing placeToken on a non-empty column **Function Name:** test_placeToken_populated

Input:

State:

X				
X				
X				
X				

c = 0
p = X

Output:

State:

X				
X				
X				
X				
X				

Reason: This test is unique because it is testing placeToken on a column that is one token away from being full, meaning after execution the column will be full

Function Name:
test_placeToken_almostFull

Input:

State:

O				
X				

c = 0
p = X

Output:

State:

X				
O				
X				

Reason: This test is unique because it is testing placeToken on a non-empty column with the token under a different character

Function Name:
test_placeToken_populated_multiplePlayers

Input:

State:

				X

c = 4
p = X

Output:

State:

				X
				X

Reason: This test is unique because it is testing placeToken on its high boundary (c < numColumn where c is numColumns - 1)

Function Name:
test_placeToken_highBoundary