CPSC 2150 Project Report

Ethan Coffey

Requirements Analysis

Functional Requirements: (Describe an Action, Something the user can do in or with a system, or something the system can do for the user).

1. As a player I want to choose the number of Rows so that I can the game a different way.
2. As a player I want to choose the number of Columns so that I can the game a different way.
3. As a player I want to choose the number of tokens to win so that I can the game a different way.
4. As a Player I want to be able to place a Marker in columns 0 through the number of columns chosen so that I can try and win the game.
5. As a Player I need to have a turn after the previous player unless the player won, tied or entered an invalid state so that I can have my turn.
6. As a Player, If I enter an invalid choice, I should be prompted to try again so that I don't lose my turn.
7. As a Player I need to place the number of tokens that specified to win in a row Diagonally so that I can win the game.
8. As a Player I need to place the number of tokens that specified to win in a row Horizontally so that I can win the game.
9. As a Player I need to place the number of tokens that specified to win in a row Vertically so that I can win the game.
10. As a Player, once the game has ended, I should be able to start a new game by pressing any button, so that I can play again.
11. As a Player If I fill the entire game board with no Winning options the game should result in a tie ,so that I can play again.
12. As a Player, depending on board parameters, the game should decide to be either a fast implementation or memory implementation, so that I can ConnectX more effectively.

Non-Functional Requirements: (They may affect our functional requirements by adding restrictions on them, They may be hardware or software restrictions on the system, They may be requirements on the way the developer must implement the system).

1. The game must be implemented in Java.
2. The game must alternate between by the amount of the players specified each time.
3. The game must run using a GUI implementation
4. Inputs must using a GUI system by clicking buttons over a columns.
5. The game Must have a minimum of 3 Rows and 3 Columns with a Maximum of 20 Rows and 20 Columns.
6. The game must have a minimum of 3 tokens in a row to win and a maximum of 20 tokens in a row to win.
7. The number of tokens to win can not exceed the columns given by the players.
8. The constructor must take in a parameter in order of rows, columns, and number of tokens needed to win for it to be set.
9. The game will print messages if invalid options are entered from Players.
10. The game must run without running into errors.

Deployment Instructions

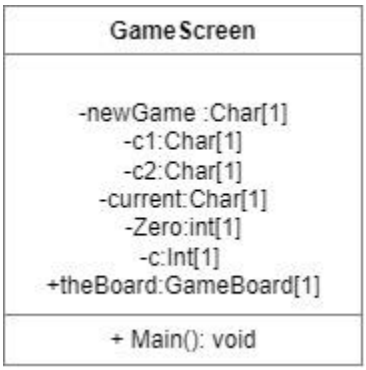
How to Run the Program of extendedConnectX:

1. To get the program to, it must be ran on IntelliJ. Navigate the project folder onto an intellj program.
2. Create an application for the ConnectXApp, look to the top right to edit and add a configuration, under the “Main Class” click the 3 dots on the side (…) and add ConnectXApp to run the app file.
3. Once the application is made click the green play button, with that the game should be playable!

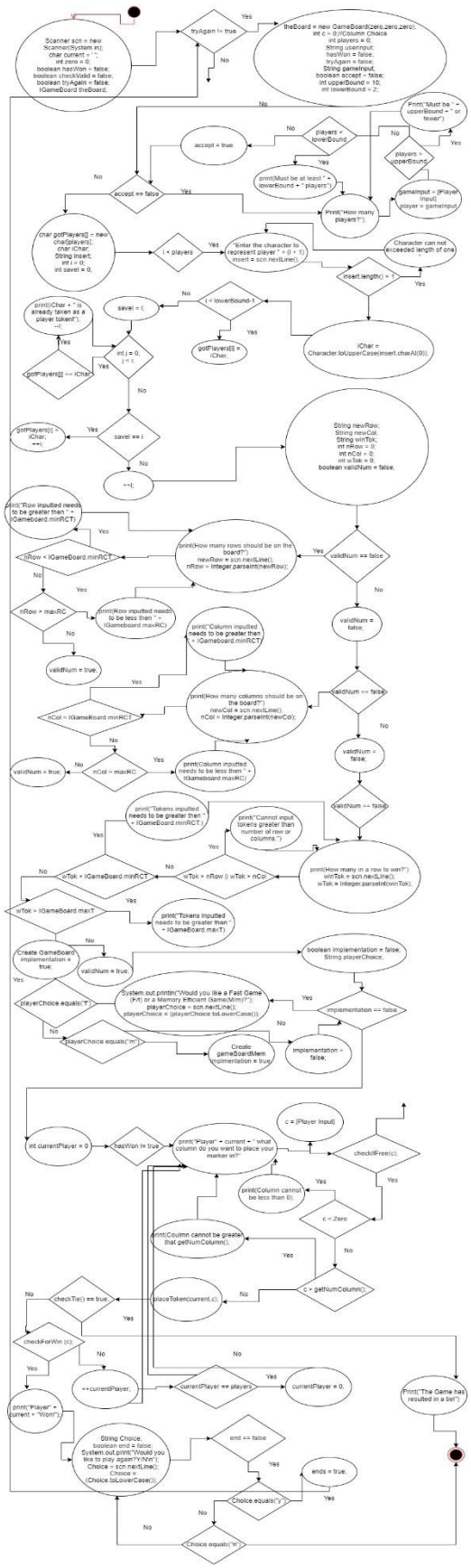
System Design

Class 1: GameScreen

Class diagram

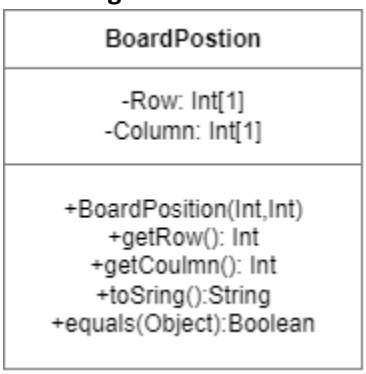


Game Screen Activity diagram(Sorry For This, But you have to Zoom in!):

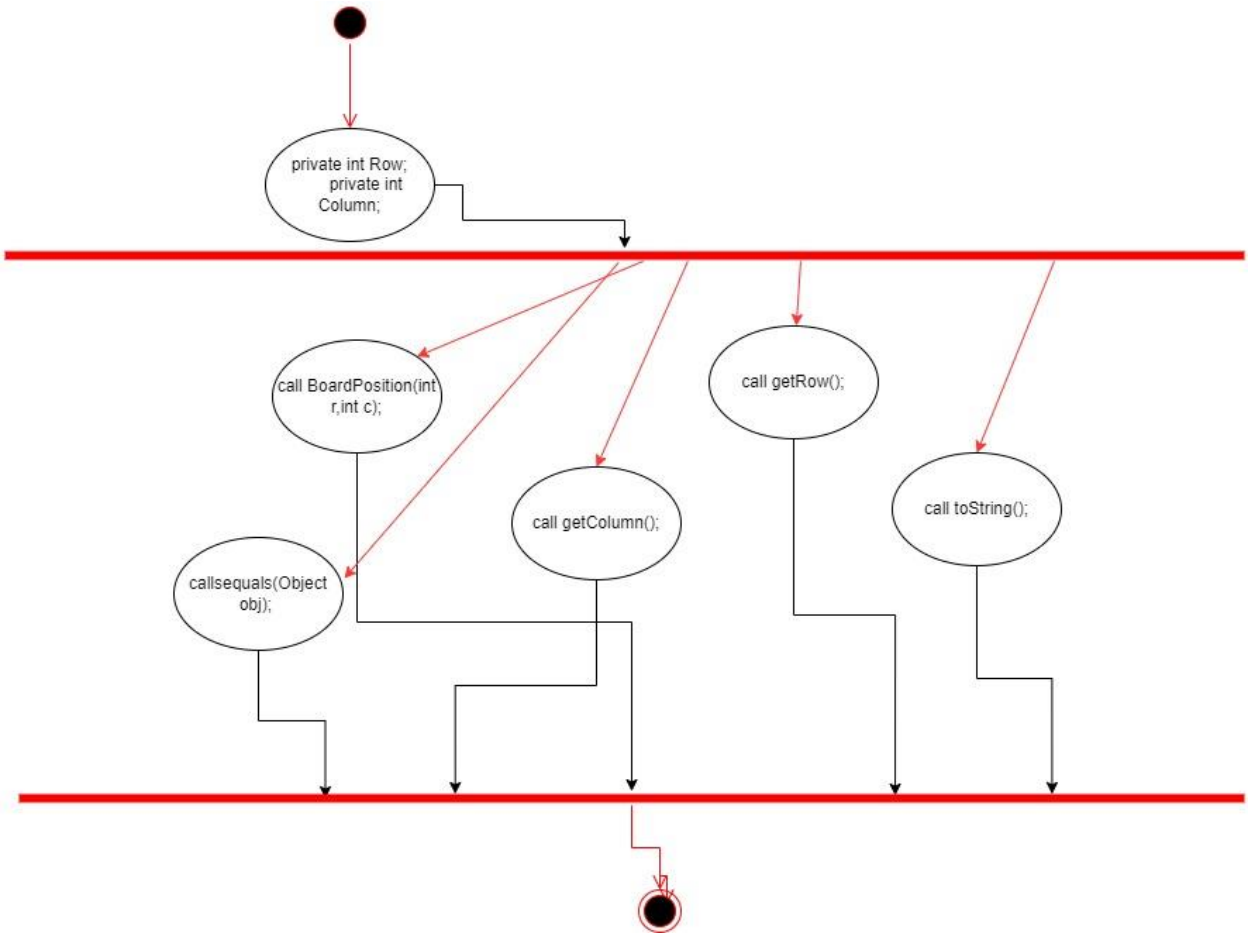

Scanner scn = new Scanner(System.in);
char current = ';';
int zero = 0;
boolean hasWon = false;
boolean checkValid = false;
boolean tryAgain = false;
IGameBoard theBoard;
tryAgain != true
Yes
No
theBoard = new GameBoard(zero,zero);
int c = 0;//Column Choice
int players = 0;
String userInput;
hasWon = false;
tryAgain = false;
String gameInput;
boolean accept = false;
int upperBound = 10;
int lowerBound = 2;
Print("Must be " + upperBound + " or fewer")
players > lowerBound
accept = true
print("Must be at least " + lowerBound + " players")
players > upperBound
gameInput = (Player Input)
player = gameInput;
accept == false
Print("How many players?")
char gotPlayers[] = new char[players];
char IChar;
String insert;
int i = 0;
int savel = 0;
i < players
"Enter the character to represent player " + (i + 1)
insert = scn.nextLine()
Character can not exceeded length of one
insert.length() > 1
i < lowerBound-1
savel = i;
print((Char + " is already taken as a player token!"), --i;
gotPlayers[i] = IChar;
IChar = Character.toUpperCase(insert.charAt(0));
int j = 0; j < i
gotPlayers[j] == IChar
gotPlayers[j] > IChar; ++i;
savel == i
++i;
String newRow;
String newCol;
String winTok;
int nRow = 0;
int nCol = 0;
int wTok = 0;
boolean validNum = false;
print("Row inputted needs to be greater than " + IGameboard.minRCT)
nRow < IGameBoard.minRCT
print(How many rows should be on the board?")
newRow = scn.nextLine();
nRow = Integer.parseInt(newRow);
validNum == false
nRow > maxRC
print(Row inputted needs to be less than " + IGameboard.maxRC)
validNum = true;
validNum = false;
print("Column inputted needs to be greater than " + IGameboard.minRCT)
nCol < IGameBoard.minRCT
print(How many columns should be on the board?")
newCol = scn.nextLine();
nCol = Integer.parseInt(newCol);
nCol > maxRC
print(Column inputted needs to be less than " + IGameboard.maxRC)
validNum = true;
print("Tokens inputted needs to be greater than " + IGameBoard.minRCT)
"print("Cannot input tokens greater than number of row or columns.")
wTok < IGameBoard.minRCT
wTok > nRow || wTok > nCol
print(How many in a row to win?")
winTok = scn.nextLine();
wTok = Integer.parseInt(winTok);
wTok > IGameBoard.maxT
print("Tokens inputted needs to be greater than " + IGameBoard.maxT)
Create GameBoard implementation = true;
validNum = true;
boolean implementation = false;
String playerChoice;
playerChoice.equals("F")
System.out.println("Would you like a Fast Game (F/f) or a Memory Efficient Game(M/m)?");
playerChoice = scn.nextLine();
playerChoice = (playerChoice.toLowerCase())
implementation == false
playerChoice.equals("m")
Create gameBoardMem implementation = true;
Implementation = false;
int currentPlayer = 0
hasWon != true
print("Player " + current + " what column do you want to place your marker in?"
c = (Player Input)
checkIfFree(c);
print(Column cannot be less than 0);
c < Zero
print(Column cannot be greater that getNumColumn();
c > getNumColumn();
checkTie() == true
placeToken(current,c);
checkForWin (c);
++currentPlayer;
currentPlayer == players
currentPlayer = 0;
Print("The Game has resulted in a tie!")
print("Player " + current + "Won!");
String Choice;
boolean end = false;
System.out.println("Would you like to play again?(Y/N/e");
Choice = scn.nextLine();
Choice = (Choice.toLowerCase());
end == false
ends = true;
Choice.equals("y")
Choice.equals("e")

Class 2: BoardPosition

Class diagram

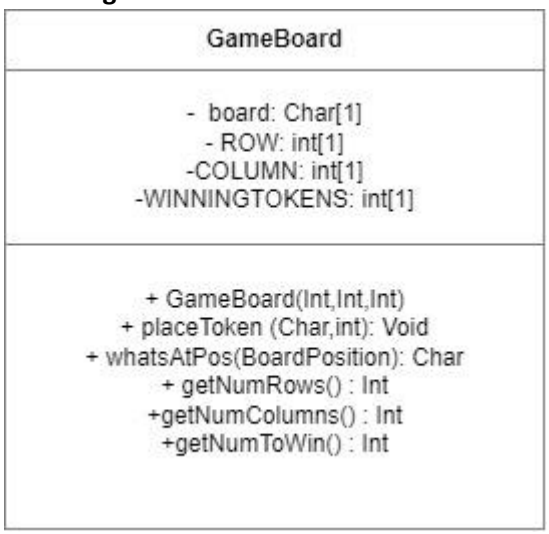


Activity diagrams

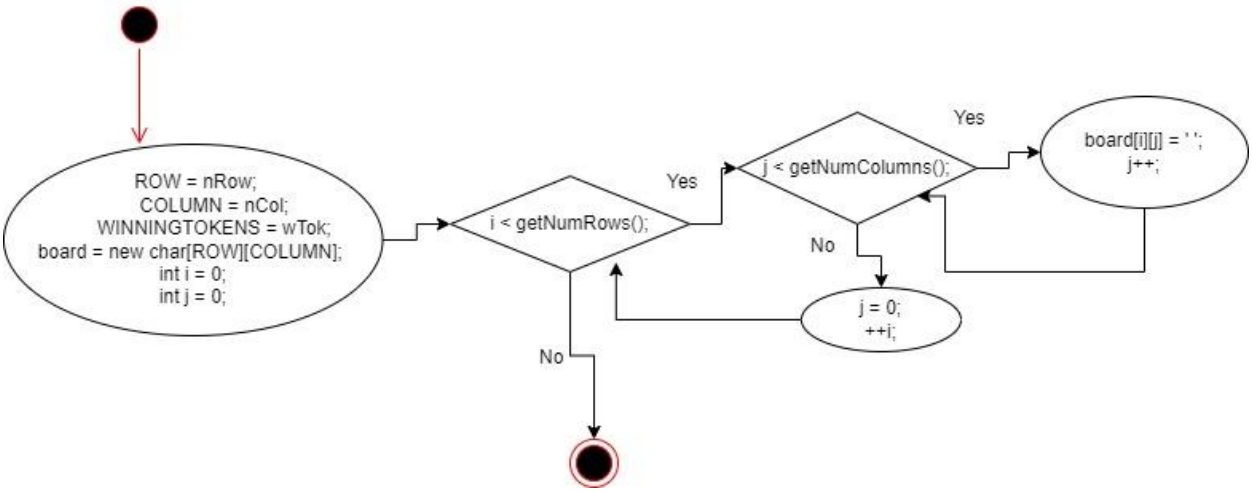


Class 3: GameBoard

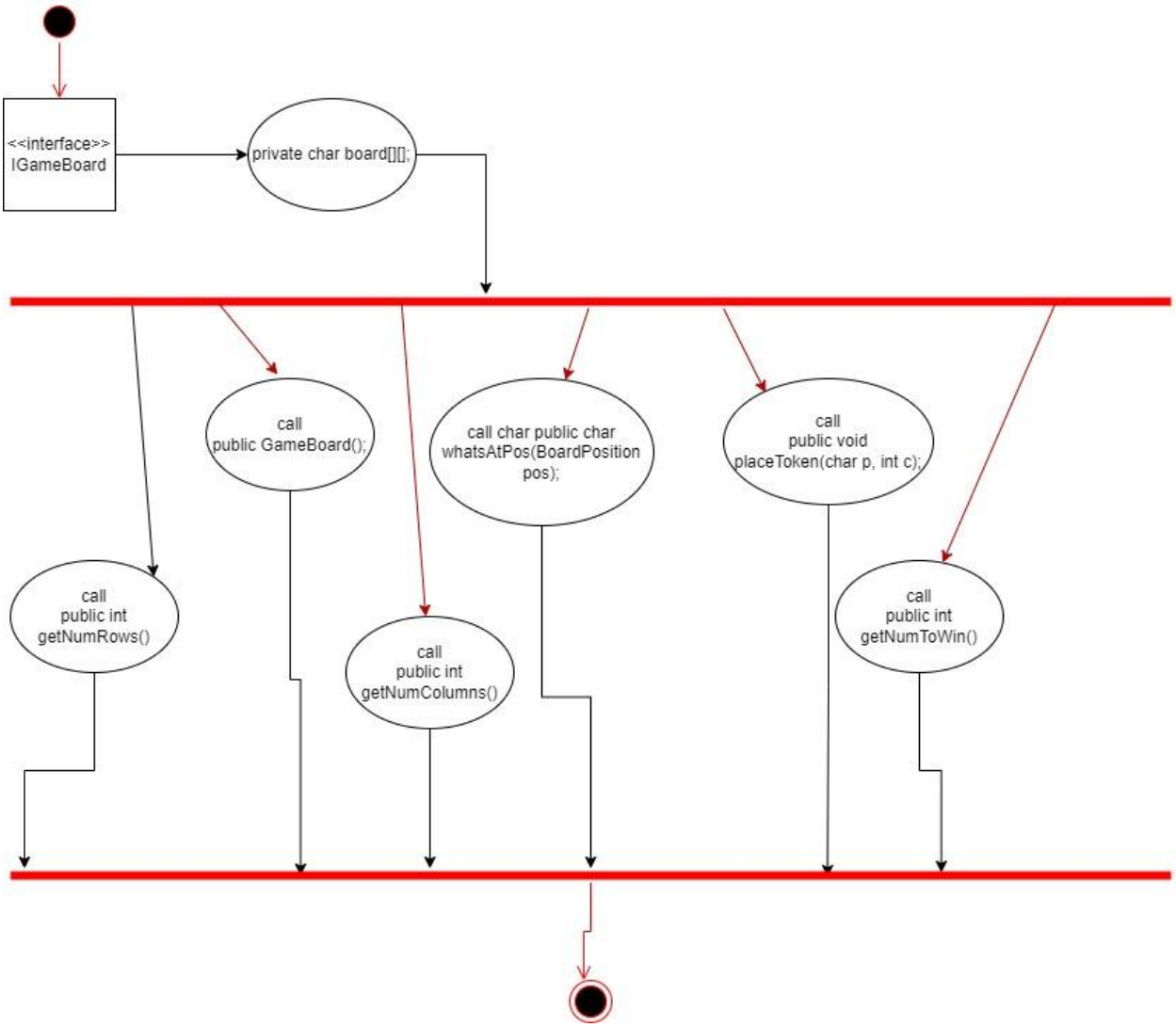
Class diagram



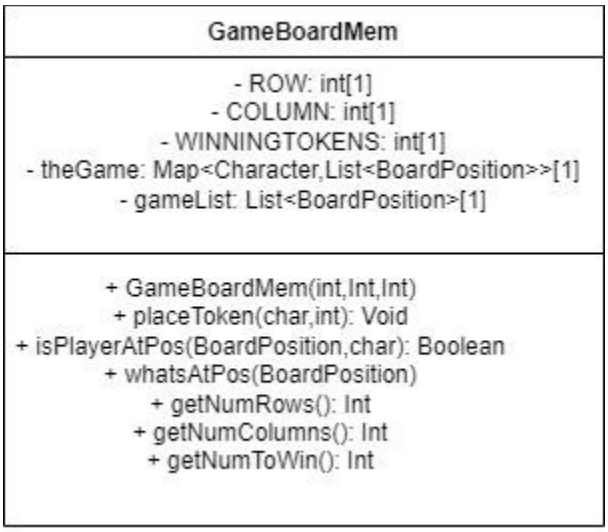
GameBoard Activity Diagram (Constructor)



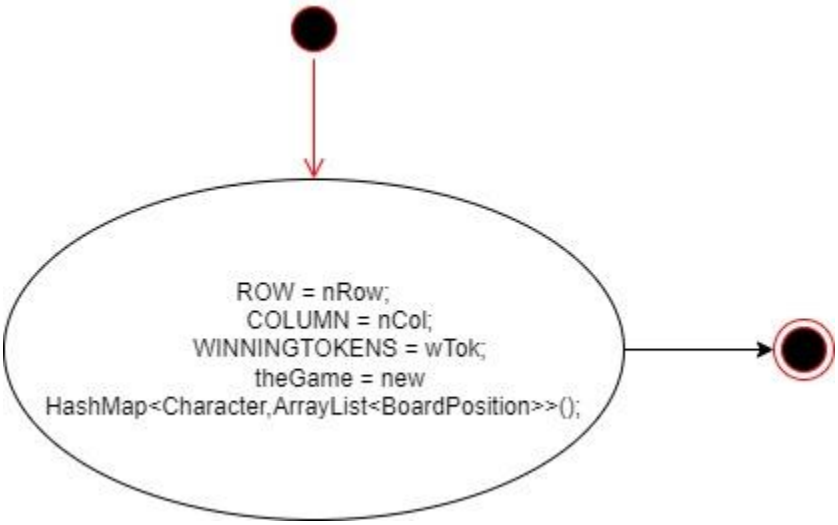
GameBoard Activity Diagram:



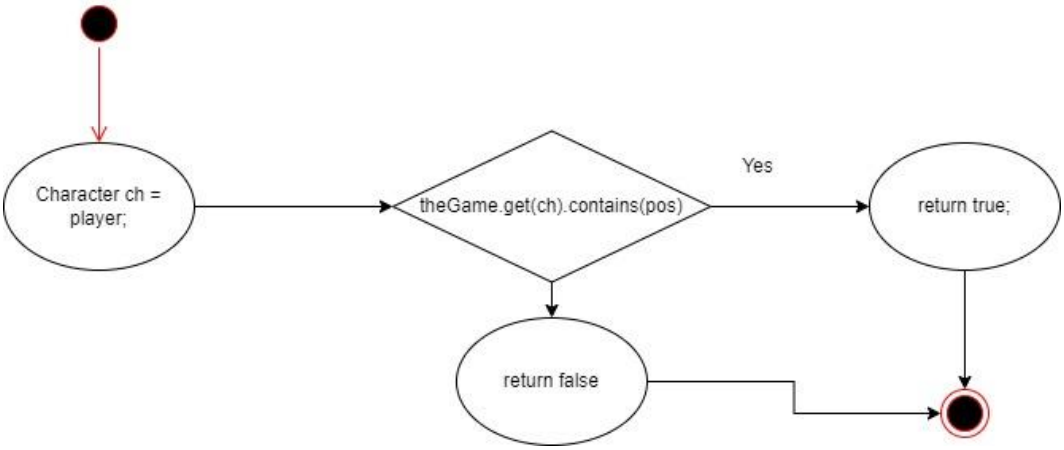
Class 4:GameBoardMem



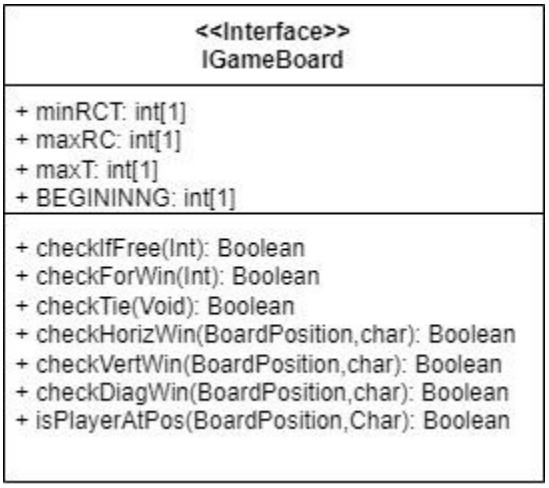
GameBoardMem Constructor Activity Diagram:



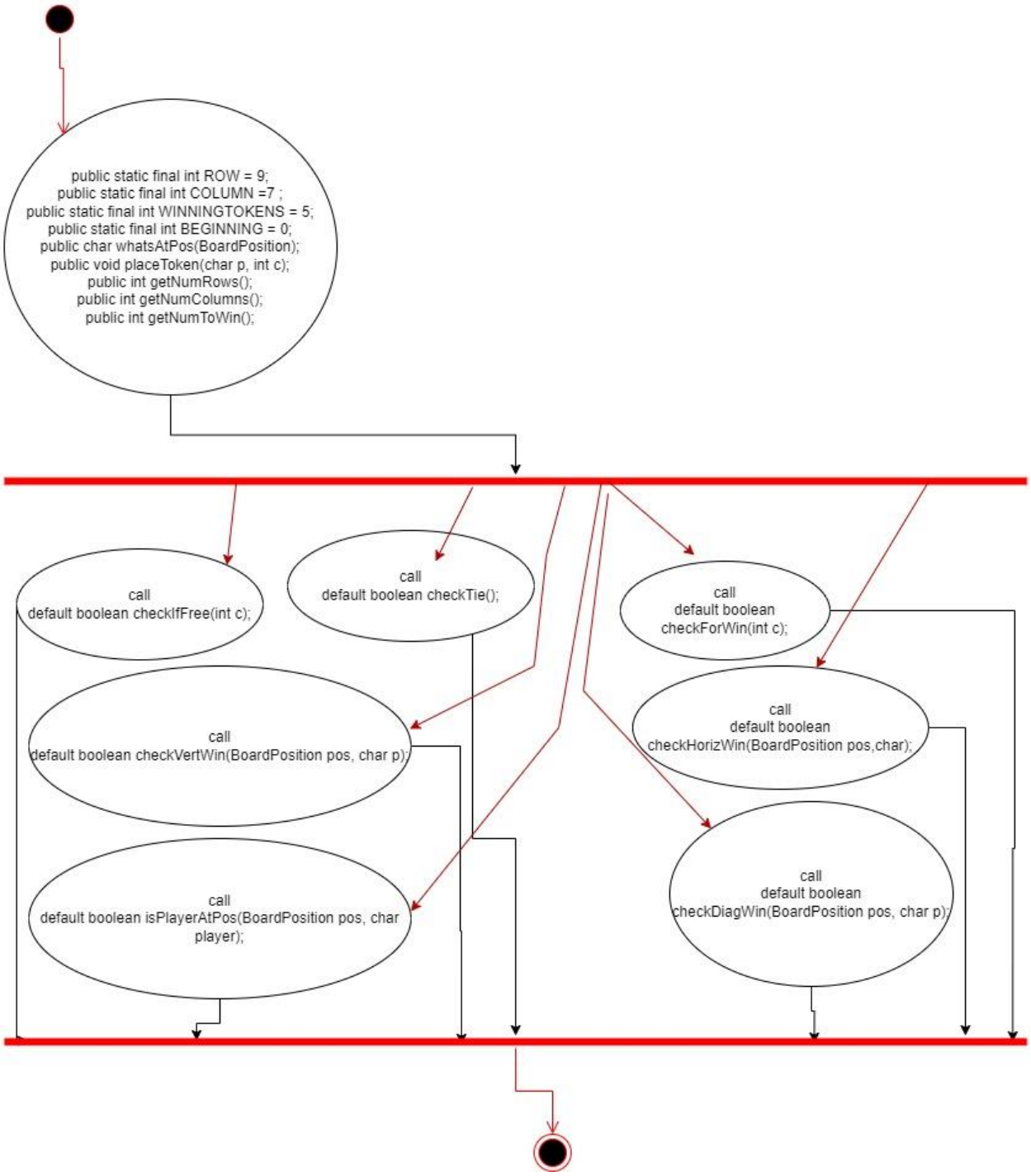
GameBoardMem isPlayerAtPos Activity Diagram



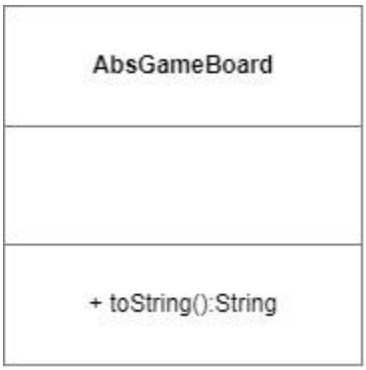
Class 5:IGameBoard



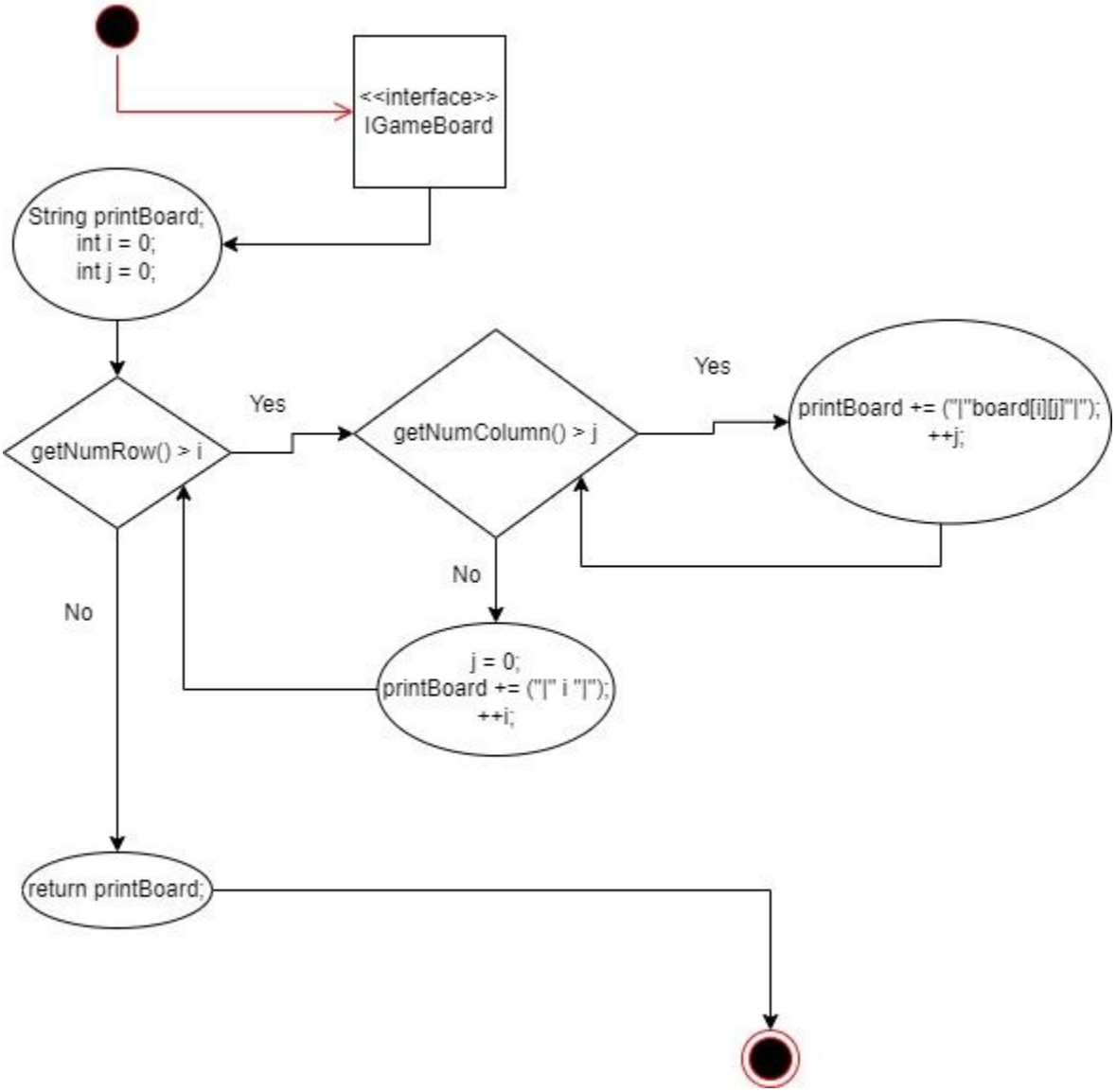
IGameBoard Activity Diagram



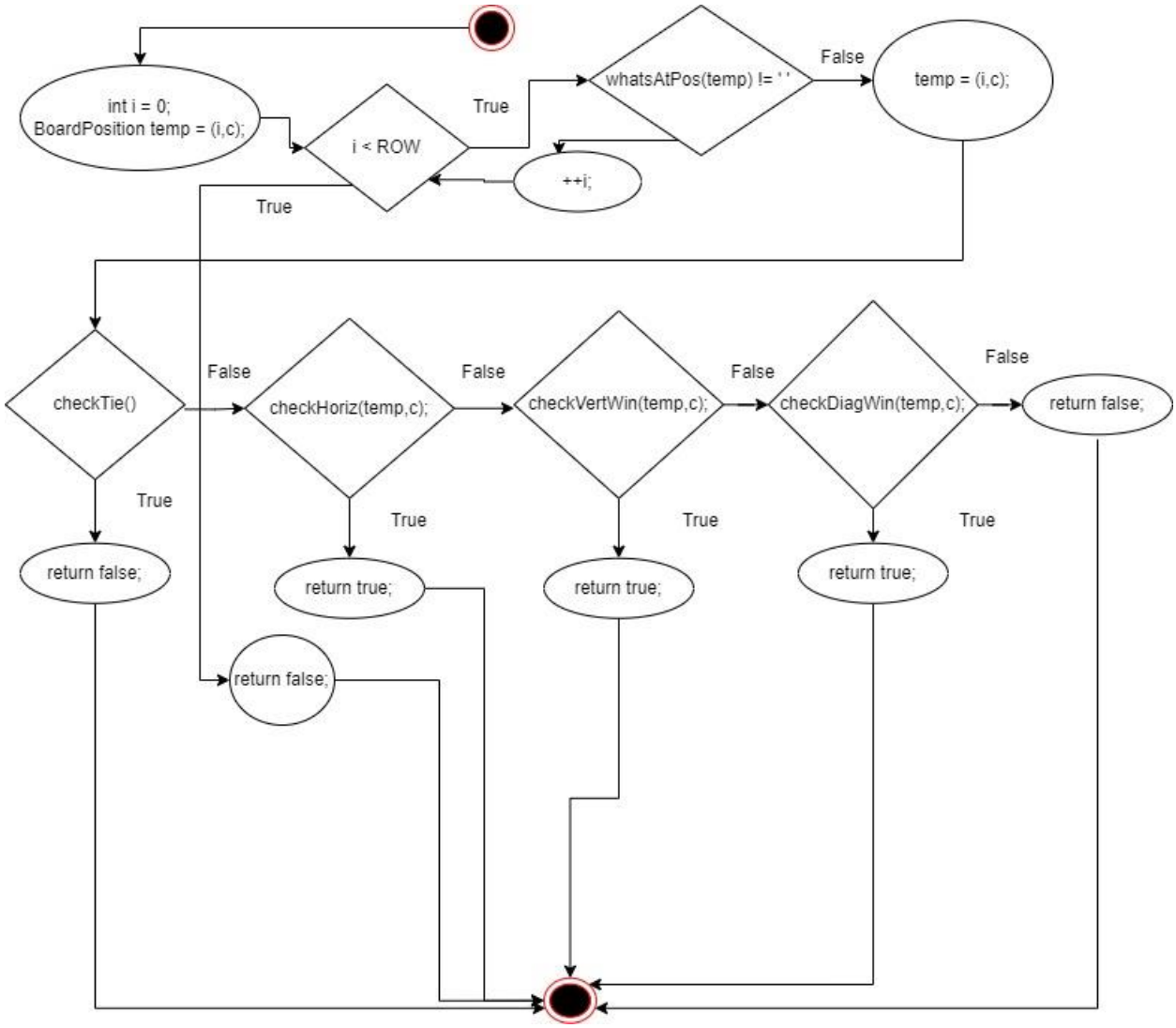
Class 5: AbsGameBoard



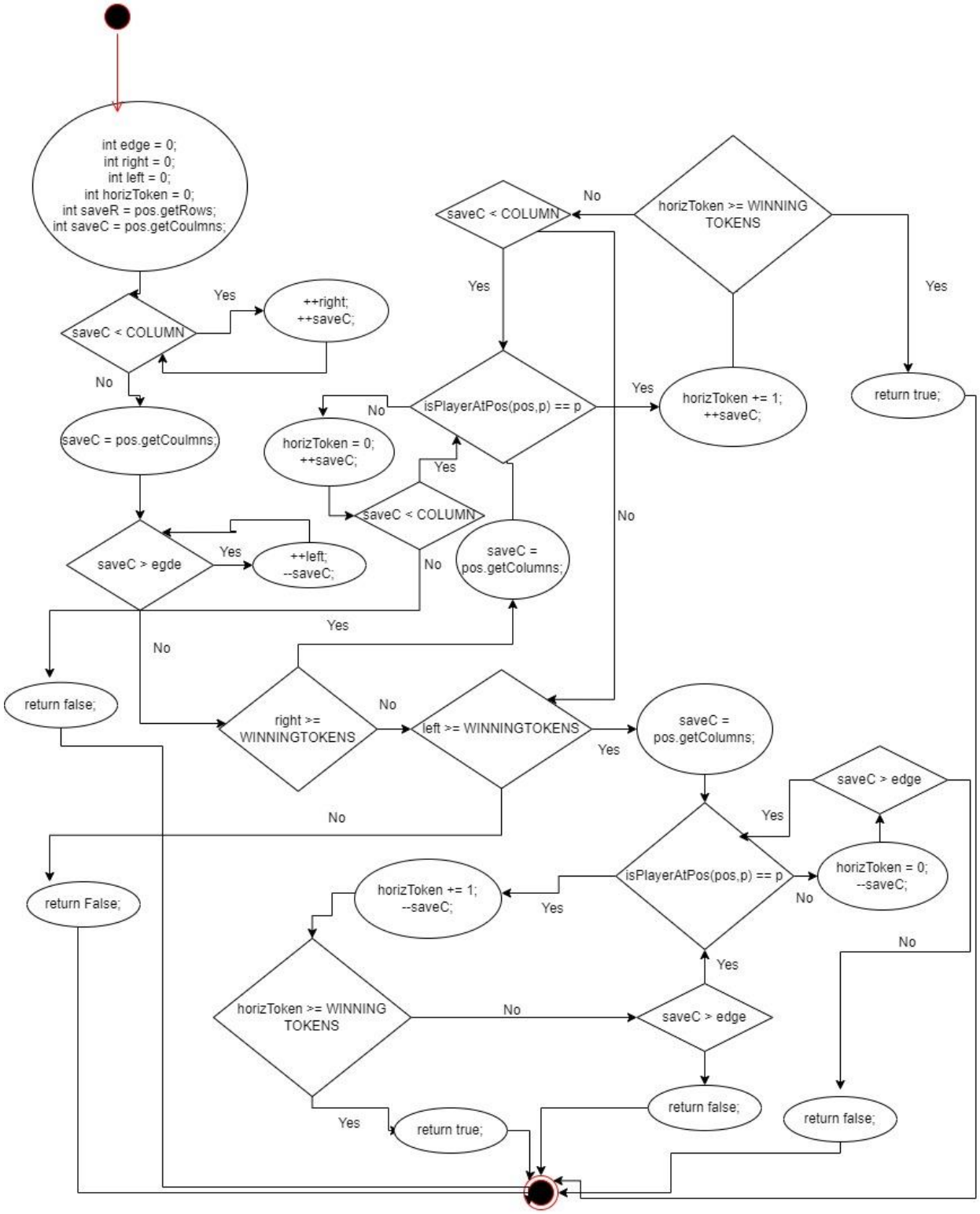
AbsGameBoard Activity diagrams



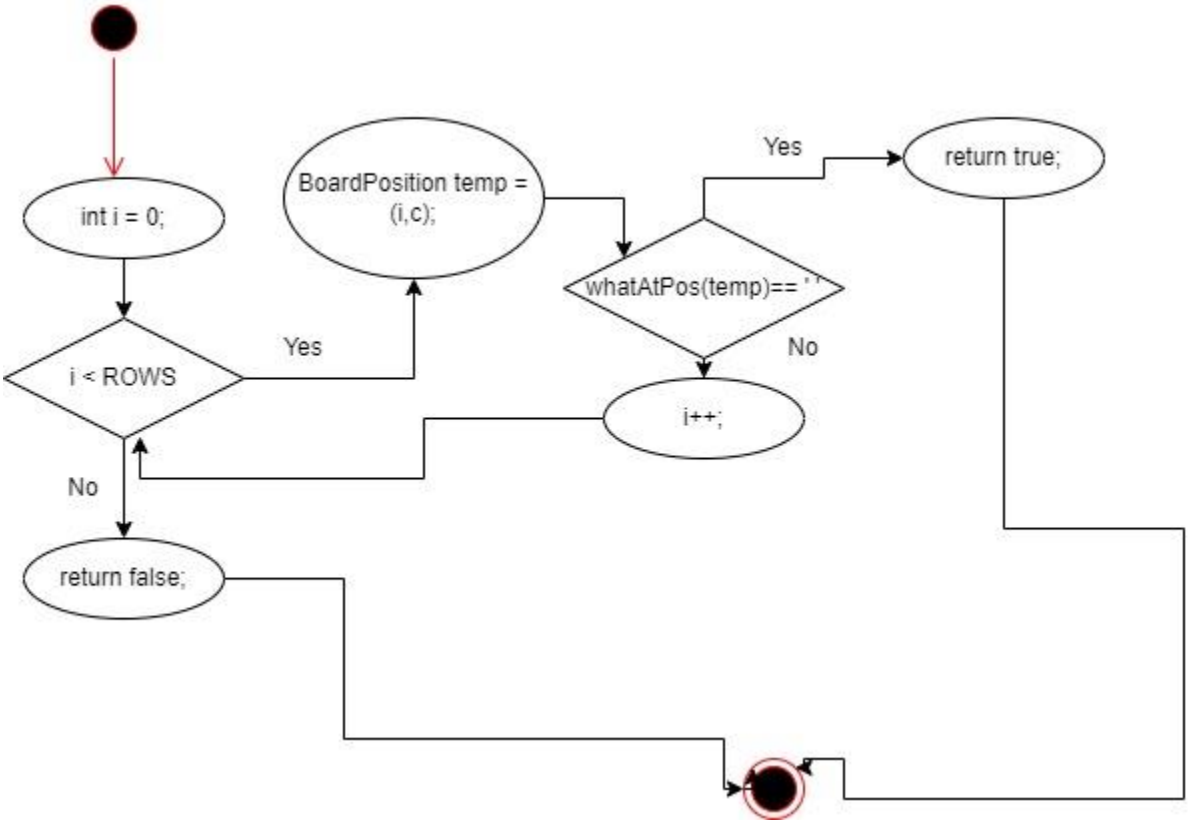
checkForWin (Default Method)



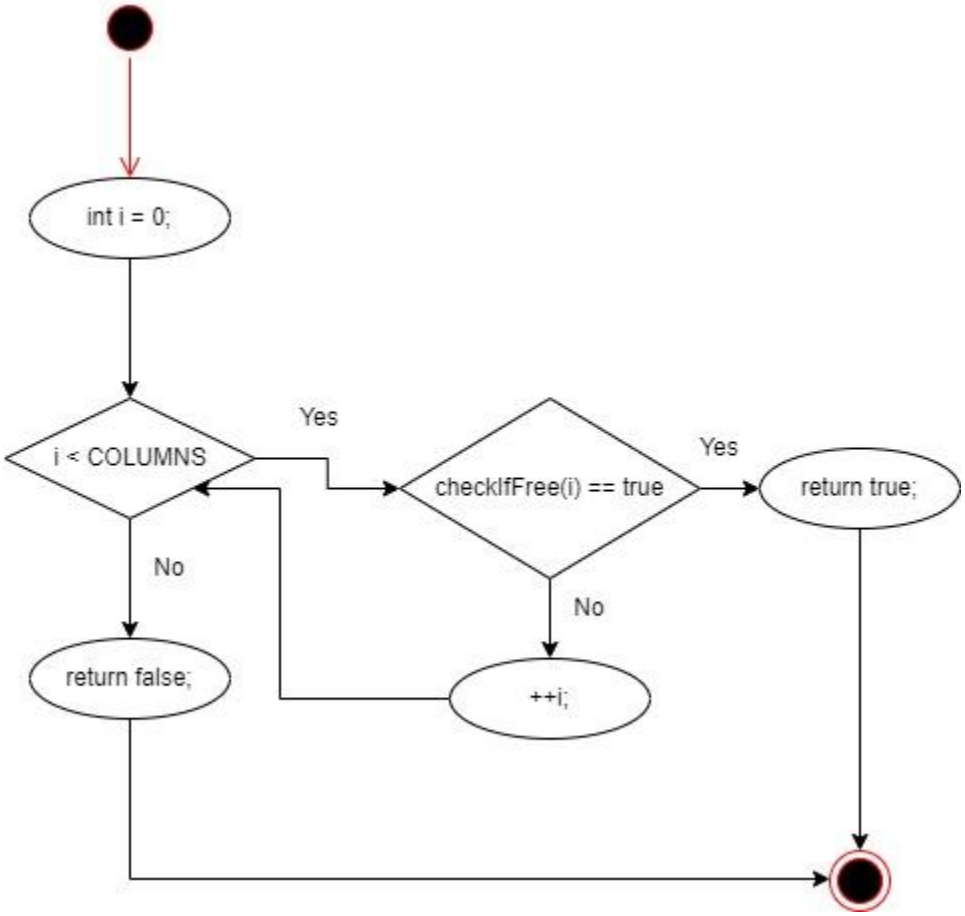
checkHorizWin (Default Method)



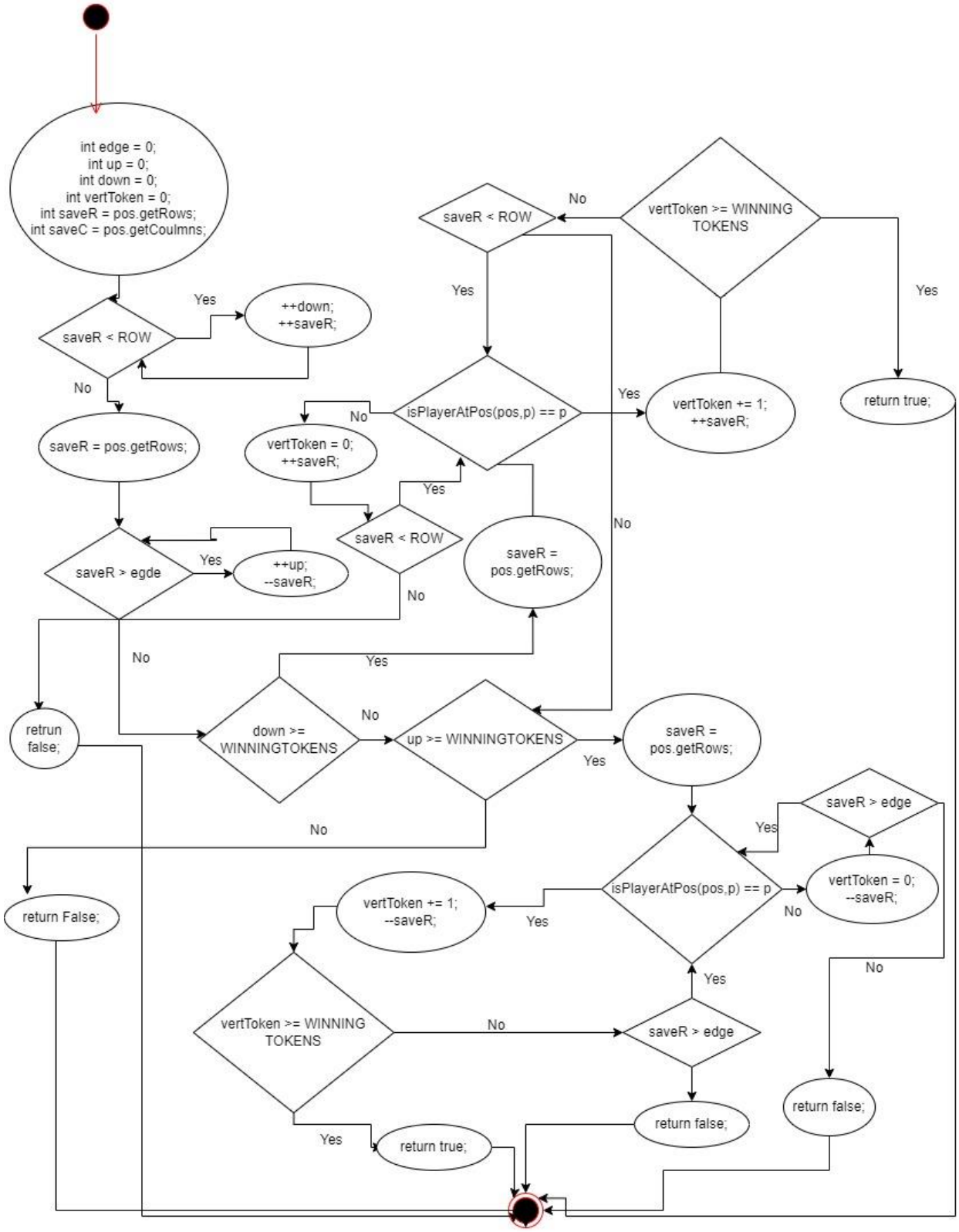
checkIfFree (Default Method)



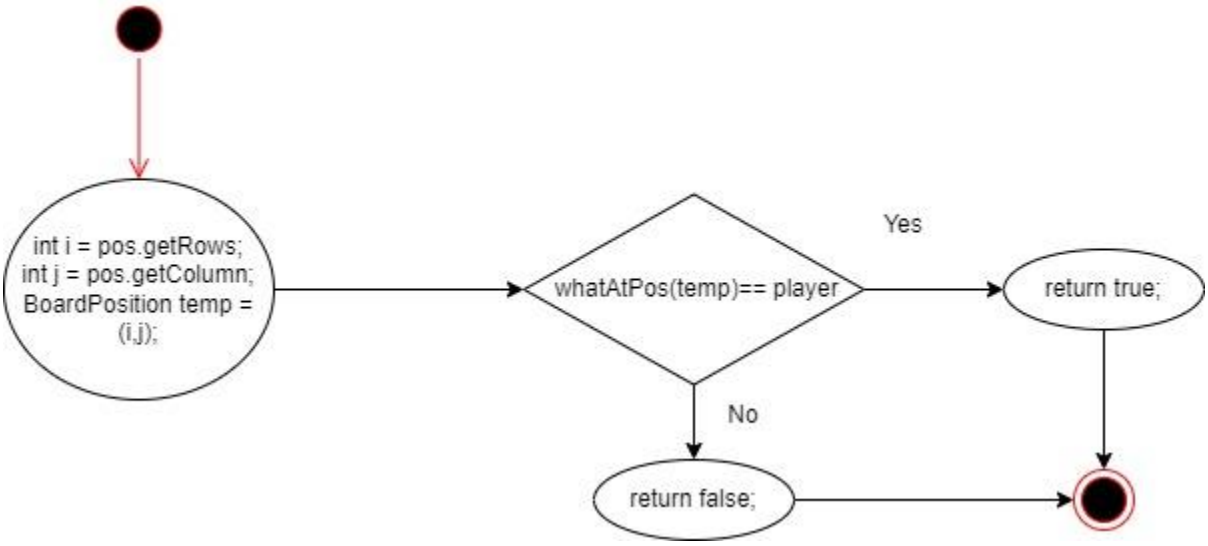
checkTie (Default Method)



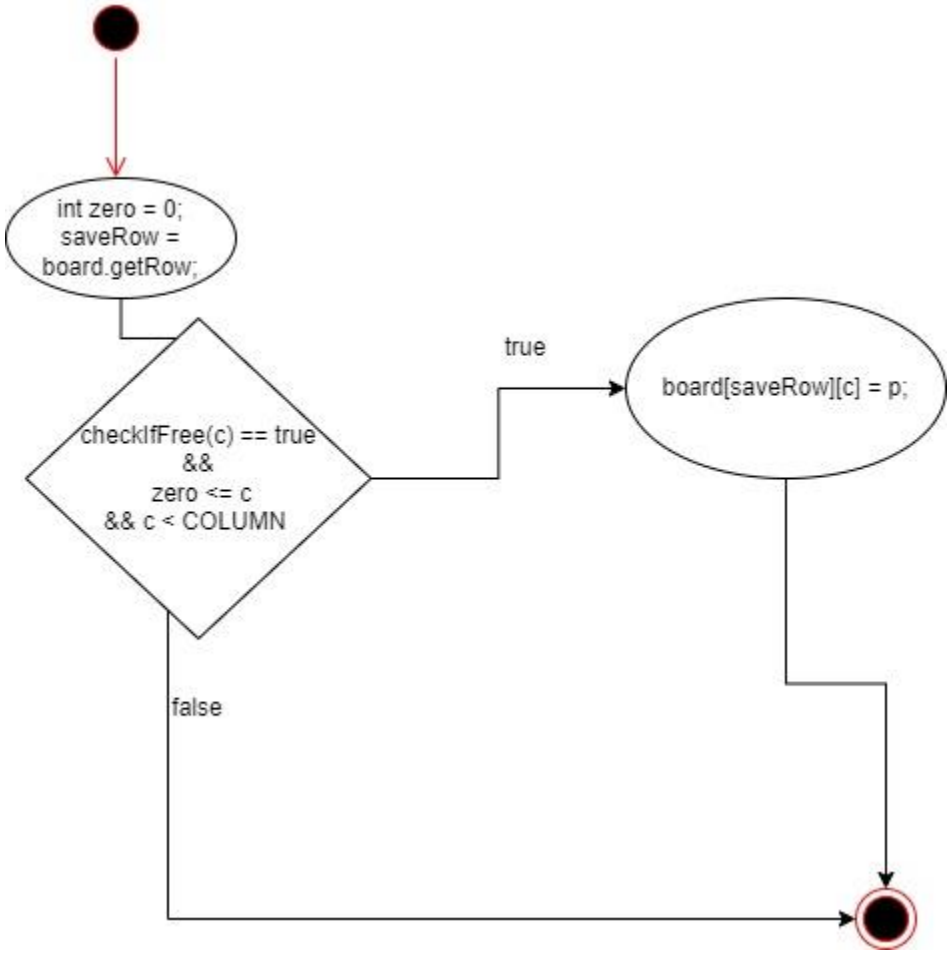
checkVertWin (Default Method)



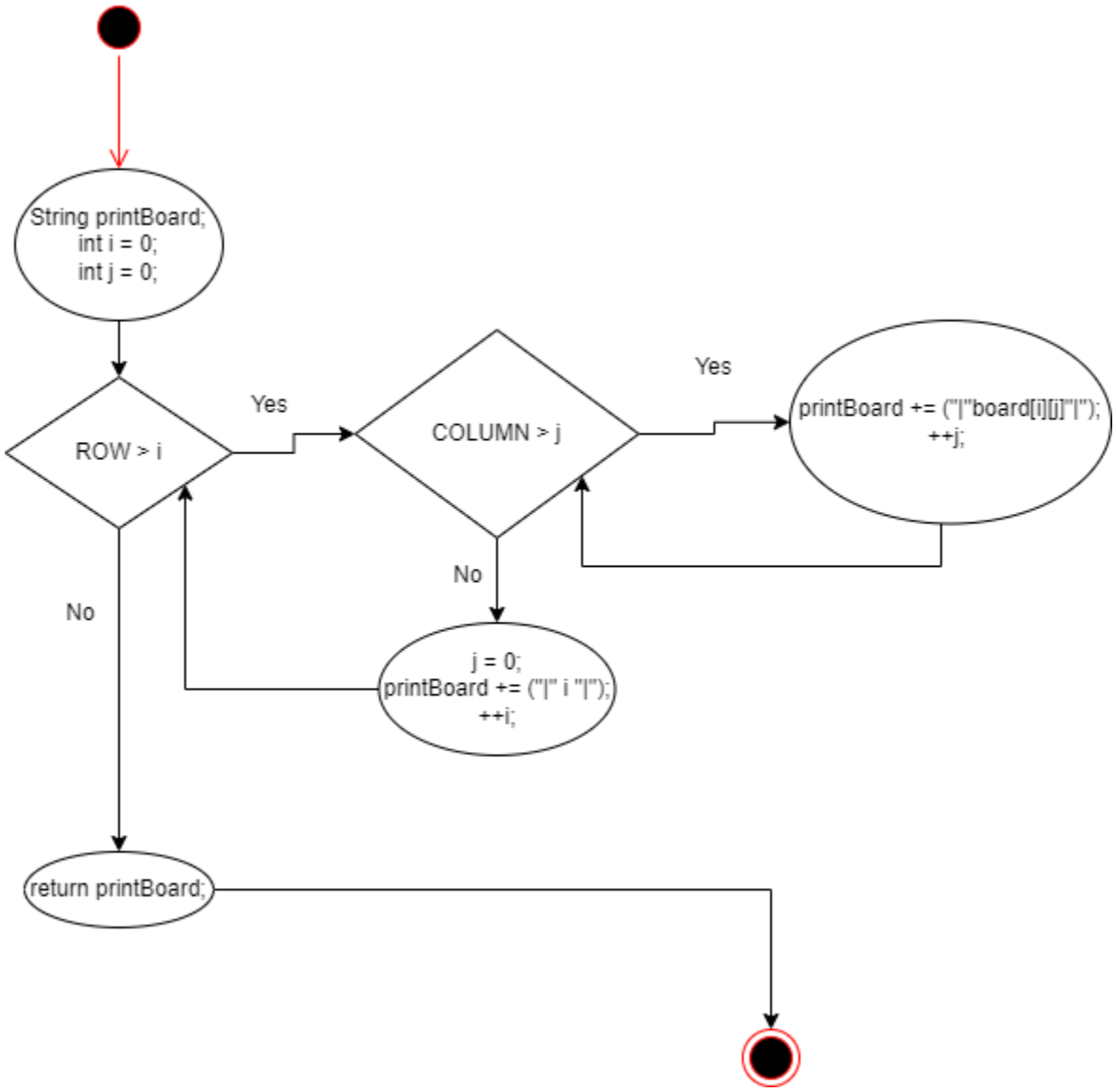
isPlayAtPos (Default Method)



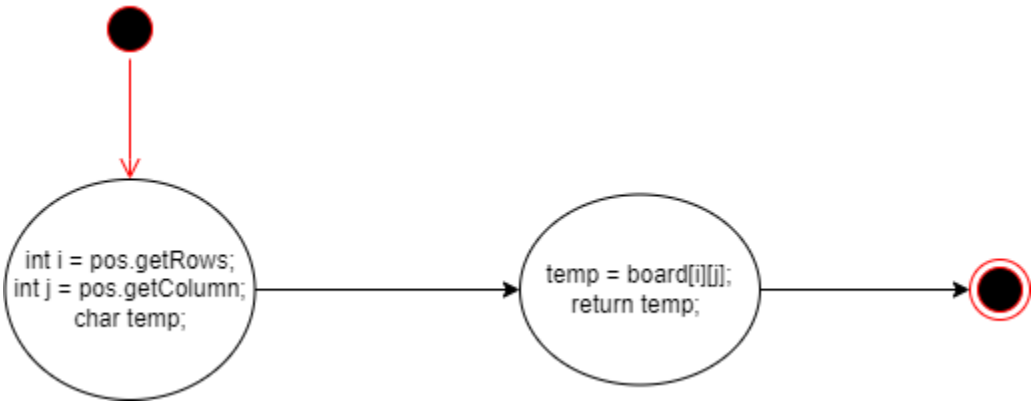
placeToken


int zero = 0;
saveRow =
board.getRow;
checkIfFree(c) == true
&&
zero <= c
&& c < COLUMN
true
board[saveRow][c] = p;
false

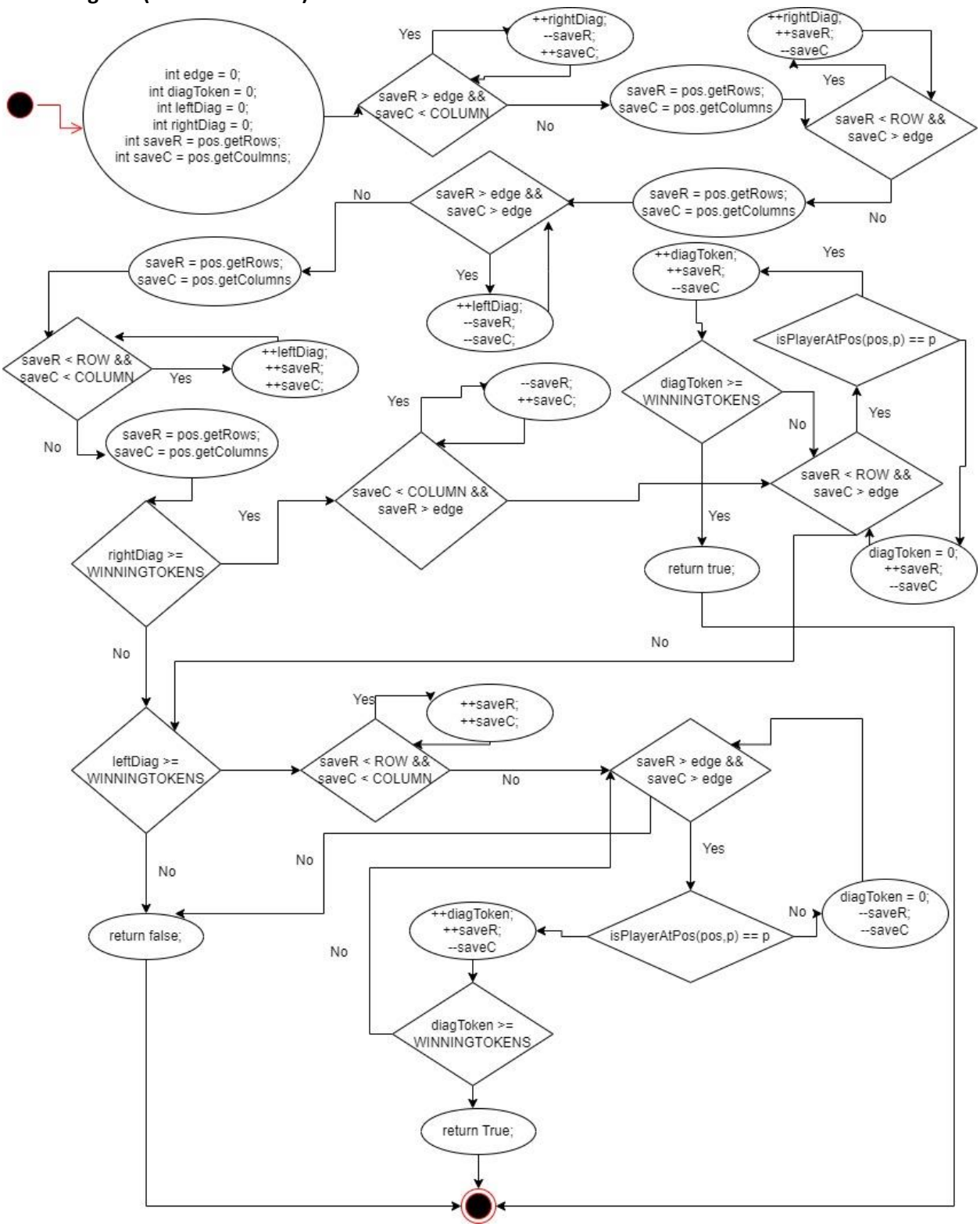
toString (*NOTE This Method has Been moved to ABSGAMEBOARD, left here to not mess anything up)



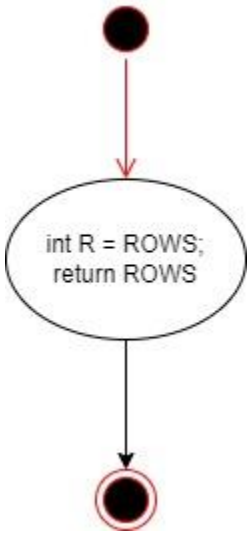
whatAtPos


int i = pos.getRows;
int j = pos.getColumn;
char temp;
temp = board[i][j];
return temp;

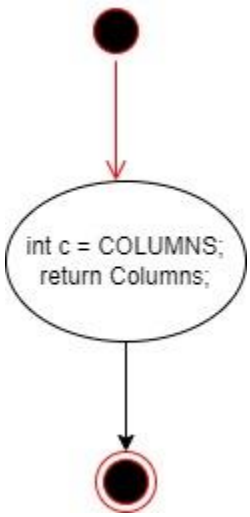
checkDiagWin (Default Method)



getNumRows()

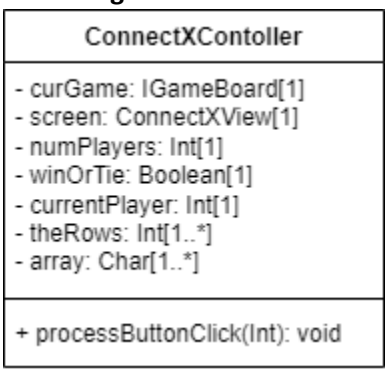


getNumColumns()

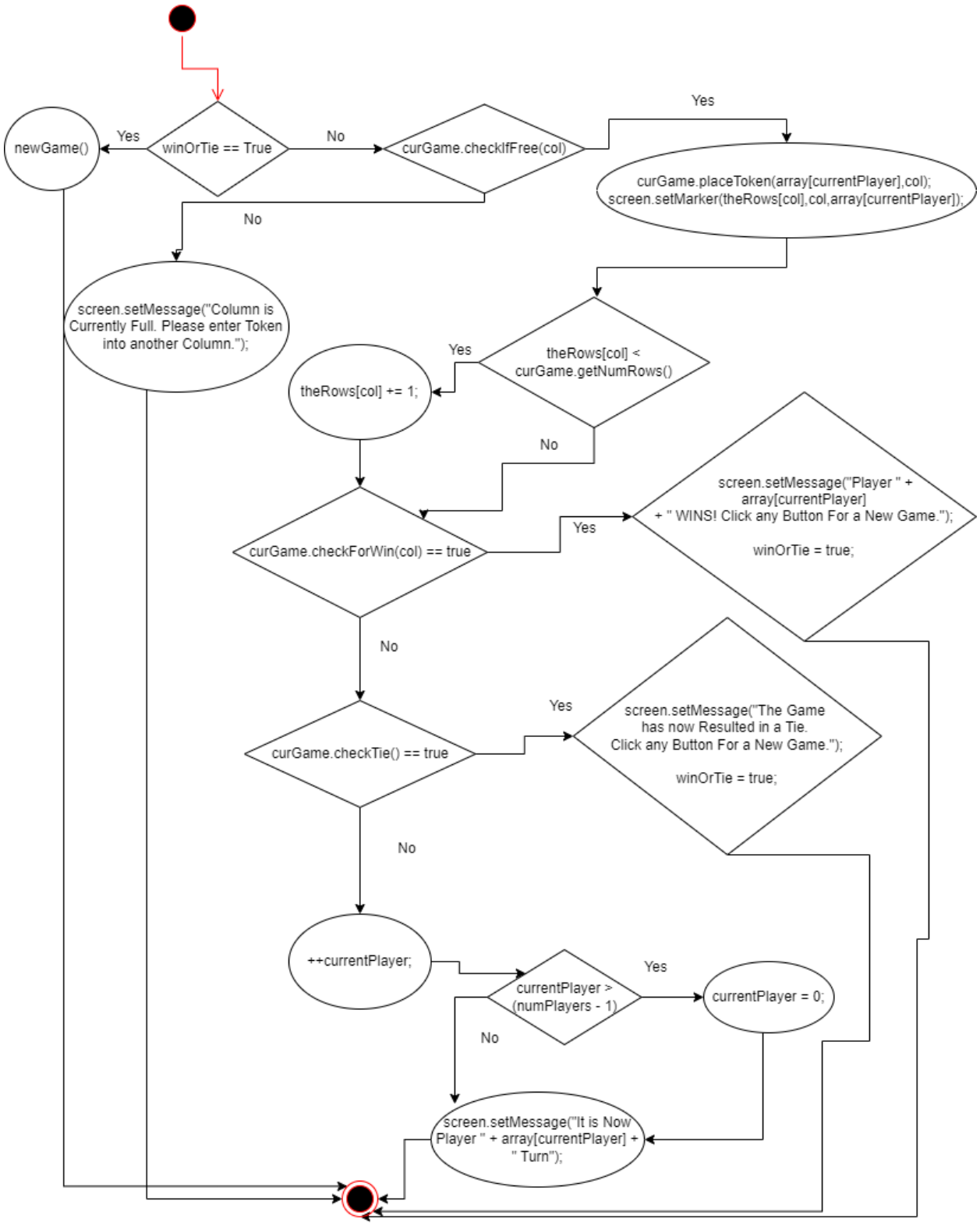


getNumToWin()

Class Diagram ConnectXController

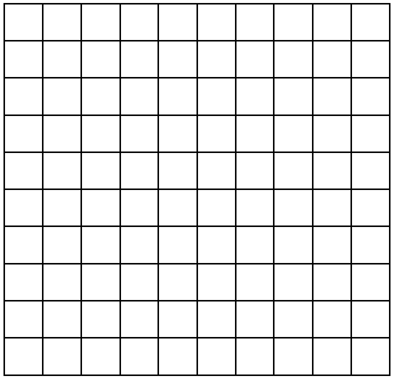


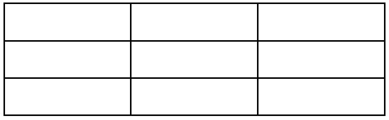
Activity Diagram ProcessButtonClick

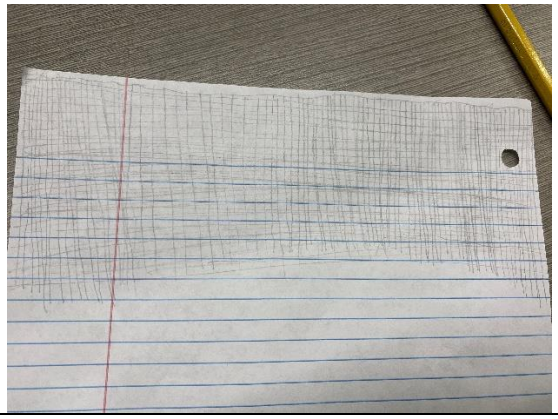


Test Cases (PROJECT 4)

Constructor (int nRow, int nCol, int wTok)

Input: nRow = 10 nCol = 10 wTok = 5	**Output**: ROW = 10 COLUMN = 10 WINNINGTOKENS = 5 State of Board (num to win = 5): 	**Reason**: This test is unique and distinct due to this board not being at either edge case (3x3 or 100x100) and the same goes for the amount of winning tokens (3 or 25). Function Name: testConstructor_Board_Ten_By_Ten_Tok_Five

Input: nRow = 3 nCol = 3 wTok = 3	**Output**: ROW = 3 COLUMN = 3 WINNINGTOKENS = 3 State of Board (num to win = 3): 	**Reason**: This test is unique and distinct due to this board at an edge case (3x3) and winning tokens aswell(3). Function Name: testConstructor_Board_Three_By_Three_Tok_Three

Input:	**Output**:	**Reason**: This test is unique and distinct due to this board at an edge case (100x100) and winning tokens aswell(100).
nRow = 100 nCol = 100 wTok = 25	ROW = 100 COLUMN = 100 WINNINGTOKENS = 25 (*May Not be Drawn to Exact Scale) State of Board(Num to win 25): 	Function Name: testConstructor_Board_Hundred_By_Hundred_Tok_TwentyFive

boolean checkIfFree(int c)

Input:	**Output**:	**Reason**:
State of Board(Num: to Win 4): [ ] O X O [ ] [ ] c = 3	checkIfFree = true; **State of the Board will remain unchanged** (checkIfFree doesn't add anything to the board)	This test is unique and distinct due to it checking a completely empty column(and not one with an entry already inside). Function Name: testcheckIfFree_Test_One Empty_Column_

Input:	**Output**:	**Reason**:
State of Board(Num: to Win 4): [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] O [ ] X O O X X O X O X O X O X O X c = 2	checkIfFree = true; **State of the Board will remain unchanged** (checkIfFree doesn't add anything to the board)	This test is unique and distinct due to it checking a column that Is slightly filled with tokens. Function Name: testcheckIfFree_Test_Two Slightly_Filled_Column

<table>
<tr>
<td>

Input:
State of Board(Num: to Win 4):

X	O			
X	O			
X	O			
O	X			
O	X			

c = 0

</td>
<td>

Output:

checkIfFree = fasle;

State of the Board will remain unchanged (checkIfFree doesn't add anything to the board)

</td>
<td>

Reason:

This test is unique and distinct due to it checking a column that Is fully filled with tokens.

Function Name:
testcheckIfFree_Test_Three Fully_Filled_Column

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

boolean checkHorizWin(BoardPosition pos, char p)

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 3):

		O
X	X	X
X	O	O

pos.getRow = 1
pos.getColumn = 0
p = 'X'

</td>
<td>

Output:

checkHorizWin = true;

state of the board is unchanged

</td>
<td>

Reason:

This test is unique and distinct due to it checking a winning horizontal row (completely full)

Function Name:
testcheckHoriz_Test_One Winning_Horizontal_Mid

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

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 3):

O	X	O	
X	O	X	

pos.getRow = 1
pos.getColumn = 2
p = 'O'

</td>
<td>

Output:

checkHorizWin = false;

state of the board is unchanged

</td>
<td>

Reason:

This test is unique and distinct due to it checking a Horizontal row (non full and non-winning Horizontal).

Function Name:
testcheckHoriz_Test_Two NonWinning_Horizontal_

</td>
</tr>
<tr>
<td>

Input:

State of Board(Num: to Win 3):

O	X	X	
X	O	O	O
O	X	O	O
X	O	X	X

Pos.getRow = 2
Pos.getColumn = 3

p = 'O'

</td>
<td>

Output:

checkHorizWin = true;

state of the board is unchanged

</td>
<td>

Reason:

This test is unique and distinct due to it checking a Horizontal row that is from a nearly full Board and resulting in a win

Function Name:
testcheckHoriz_Test_Three NFull_Winning_Horizontal_

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

<table>
<tr>
<td>Input:

State of Board(Num: to Win 3):
<table>
<tr><td></td><td></td><td>O</td><td>O</td></tr>
<tr><td>X</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>X</td><td>O</td><td>X</td><td>O</td></tr>
</table>
Pos.getRow = 2
Pos.getColumn = 0

p = ‘X’</td>
<td>Output:

checkHorizWin = false;

state of the board is unchanged.</td>
<td>Reason:

This test is unique and distinct due to it checking for a win horizontal, although it totals up to three, it is not in a row, thus resulting in returning false

Function Name:
testcheckHoriz_Test_Four GoodCount_NotRow_Check</td>
</tr>
</table>

Boolean checkVertWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:

State of Board(Num: to Win 3):
<table>
<tr><td>X</td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td></td></tr>
<tr><td>X</td><td>O</td><td></td></tr>
</table>
pos.getRow = 2
pos.getColumn = 0
p = ‘X’</td>
<td>Output:

checkVertWin = true;

state of the board is unchanged</td>
<td>Reason:

This test is unique and distinct due to it checking for a vertical win with the minimum amount of tokens (a full column of three in a row).

Function Name:
testcheckVert_Test_One Winning_Min</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State of Board(Num: to Win 3):
<table>
<tr><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td></td><td></td></tr>
</table>
pos.getRow = 2
pos.getColumn = 1
p = ‘X’</td>
<td>Output:

checkVertWin = false;

state of the board is unchanged</td>
<td>Reason:

This test is unique and distinct due to it checking for a vertical, but the current board doesn’t have one

Function Name:
testcheckVert_Test_Two NonWinning_Game_Min</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State of Board(Num: to Win 4):
<table>
<tr><td></td><td>X</td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>O</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>O</td><td>X</td><td>X</td><td>O</td></tr>
</table>
pos.getRow = 3
pos.getColumn = 1
p = ‘X’</td>
<td>Output:

checkVertWin = true;

state of the board is unchanged</td>
<td>Reason:

This test is unique and distinct due to it winning a vertically game with an almost filled board while not on an edge.

Function Name: testcheckVert_Test_Three Winning_Game_NearMax</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State of Board(Num: to Win 4):
<table>
<tr><td>X</td><td>O</td><td></td><td>O</td></tr>
<tr><td>O</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>O</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>O</td><td>X</td><td>O</td><td>X</td></tr>
</table>
pos.getRow = 3
pos.getColumn = 3
p = ‘O’</td>
<td>Output:

checkVertWin = false;

state of the board is unchanged</td>
<td>Reason:

This test is unique and distinct due to checking an almost full board, checking above and below the token, however its not 4 in a row, resulting in it being false

Function Name: testcheckVert_Test_Four NonWinning_Game_NearMax</td>
</tr>
</table>

Boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:

State of Board(Num: to Win 3):
<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td></tr>
<tr><td>X</td><td>X</td><td>O</td><td></td></tr>
<tr><td>X</td><td>O</td><td>O</td><td></td></tr>
</table>
pos.getRow = 2
pos.getColumn = 2</td>
<td>Output:

checkDiagWin = true;

state of the board is unchanged</td>
<td>Reason:

This test is unique and distinct due to it checking the right diagonal(from current pos to left corner), finding 3 in a row, resulting in checkDiagWin being true.

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

p = ‘X’		testcheckDiagWin_Test_One Right_Diag_Win

Input: State of Board(Num: to Win 3): [table below] pos.getRow = 2 pos.getColumn = 2 p = ‘O’	**Output**: checkDiagWin = false; state of the board is unchanged	**Reason**: This test is unique and distinct due to it checking the right diagonal(from current pos to left corner), **not** finding 3 in a row, resulting in checkDiagWin being false. Function Name: testcheckDiagWin_Test_Two Right_Diag_Lose

		O	
X	O	X	
X	O	O	

Input: State of Board(Num: to Win 3): [table below] pos.getRow = 2 pos.getColumn = 1 p = ‘O’	**Output**: checkDiagWin = true; state of the board is unchanged	**Reason**: This test is unique and distinct due to it checking the Left diagonal(from current pos to right corner), finding 3 in a row, resulting checkDiagWin being true. Function Name: testcheckDiagWin_Test_Three Left_Diag_Win

	O		
	X	O	X
	X	O	O

Input: State of Board(Num: to Win 3): [table below] pos.getRow = 2 pos.getColumn = 1 p = ‘X’	**Output**: checkDiagWin = false; state of the board is unchanged	**Reason**: This test is unique and distinct due to it checking the Left diagonal (from current pos to right corner), **not** finding 3 in a row, resulting in checkDiagWin being false. Function Name: testcheckDiagWin_Test_Four Left_Diag_Lose

	X		
	O	O	X
	X	O	X

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 4):

X				
O	X			
X	O	X		
O	X	O	X	
X	O	O	X	O

pos.getRow = 4
pos.getColumn = 0
p = ‘X’

</td>
<td>

Output:

checkDiagWin = true;

state of the board is unchanged

</td>
<td>

Reason:

This test is unique and distinct due to it checking the Left diagonal (from current pos, in top left corner), finding 4 in a row, resulting in checkDiagWin being true.

Function Name:
testcheckDiagWin_Test_Five Left_Diag__Top_Corner_Win

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

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 4):

				X
			X	O
			O	X
		O	X	O
	O	X	O	X

pos.getRow = 0
pos.getColumn = 1
p = ‘X’

</td>
<td>

Output:

checkDiagWin = true;

state of the board is unchanged

</td>
<td>

Reason:

This test is unique and distinct due to it checking the right diagonal (from current pos to right edge), finding 4 in a row, resulting in checkDiagWin being true.

Function Name:
testcheckDiagWin_Test_Six Right_Diag__Edge_Win

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

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 4):

X				
X	O			
O	X	O		
X	O	O	O	

pos.getRow = 0
pos.getColumn = 3
p = ‘X’

</td>
<td>

Output:

checkDiagWin = false;

state of the board is unchanged

</td>
<td>

Reason:

This test is unique and distinct due to it checking the Left diagonal (from current pos to top left edge), and **not** finding 4 in a row, resulting in checkDiagWin being false.

Function Name:
testcheckDiagWin_Test_Seven Left_Diag__Edge_Lose

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

Boolean checkTie()

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 4):

A	H	B	F
B	G	A	E
C	F	J	D
D	E	I	C

</td>
<td>

Output:

checkTie = true;

State of the board remains unchanged

</td>
<td>

Reason:

This test is unique and distinct due to the entire board being full with no winning player, resulting in checkTie being true.

Function Name: testcheckTie_Test_One Full_Board_TenPlayers

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

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 3):

X	O	

</td>
<td>

Output:

checkTie = false;

State of the board remains unchanged

</td>
<td>

Reason:

This test is unique and distinct due to the board being bare an empty, not full enough to be a tie.

Function Name: testcheckTie_Test_Two NonFull_Board_TwoPlayers

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

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 4):

V	B	V	
C	N	C	N
Z	M	Z	M
X	A	X	A

</td>
<td>

Output:

checkTie = false;

State of the board remains unchanged

</td>
<td>

Output:
This test is unique and distinct due to the board nearly being full, but not completely, meaning there is no tie.
Function Name: testcheckTie_Test_Three NonFullTwo_Board_EightPlayers

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

Input:	**Output**:	**Output**:

<table>
<tr>
<td>State of Board(Num: to Win 7):
<table>
<tr><td>A</td><td>A</td><td>J</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr>
<tr><td>S</td><td>S</td><td>A</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td></tr>
<tr><td>D</td><td>D</td><td>S</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td></tr>
<tr><td>F</td><td>F</td><td>D</td><td>F</td><td>F</td><td>F</td><td>F</td><td>F</td><td>F</td></tr>
<tr><td>G</td><td>G</td><td>F</td><td>G</td><td>G</td><td>G</td><td>G</td><td>G</td><td>G</td></tr>
<tr><td>H</td><td>H</td><td>G</td><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td></tr>
<tr><td>J</td><td>J</td><td>H</td><td>J</td><td>J</td><td>J</td><td>J</td><td>J</td><td>J</td></tr>
</table>
</td>
<td>checkTie = true;

State of the board remains unchanged</td>
<td>This test is unique and distinct due to the entire board being full, with board also being a lot bigger with more players, is filled resulting in checkTie being true.

Function Name: testcheckTie_Test_Four FullTwo_Board_SevenPlayers</td>
</tr>
</table>

Char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:

State of Board(Num: to Win 3):
<table>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td></td></tr>
</table>
pos.getRow = 0
pos.getColumn = 1</td>
<td>Output:

whatAtPos = O;

State of the board remains unchanged</td>
<td>Reason:

This test is unique and distinct due to whatsAtpos returning the char O on a very small board.

Function Name: testwhatsAtPos_Test_One CharO_SB</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State of Board(Num: to Win 3):
<table>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td></td></tr>
</table>
pos.getRow = 0
pos.getColumn = 0</td>
<td>Output:

whatAtPos = X;

State of the board remains unchanged</td>
<td>Reason:

This test is unique and distinct due to whatsAtpos returning the char X, in the lower left corner of a very small board.

Function Name: testwhatsAtPos_Test_Two CharX_SB</td>
</tr>
</table>

<table>
<tr>
<td>Input:

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

pos.getRow = 0
pos.getColumn = 2</td>
<td>Output:

whatAtPos = ‘ ‘;

State of the board remains unchanged</td>
<td>Reason:

This test is unique and distinct due to whatsAtpos returning a blank space on a very small board.

Function Name: testwhatsAtPos_Test_Three BlankS_SB</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State of Board(Num: to Win 6):
<table>
<tr><td>O</td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>X</td><td></td><td></td><td></td><td>X</td><td>O</td></tr>
<tr><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>X</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr>
</table>

pos.getRow = 5
pos.getColumn = 5</td>
<td>Output:

whatAtPos = O;

State of the board remains unchanged</td>
<td>Reason:

This test is unique and distinct due to whatsAtpos returning the O from the top right of the board of a decently size board.

Function Name: testwhatsAtPos_Test_Four LeftCharO_DB</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State of Board(Num: to Win 6):
<table>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td></td><td></td><td></td><td></td></tr>
</table>

pos.getRow = 5
pos.getColumn = 0</td>
<td>Output:

whatAtPos = X;

State of the board remains unchanged</td>
<td>Reason:

This test is unique and distinct due to whatsAtpos returning the X from the top left of the board

Function Name: testwhatsAtPos_Test_Five LeftCharX_DB</td>
</tr>
</table>

Boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 3):

	O	
X	O	X

pos.getRow = 1
pos.getColumn = 1
p = ‘O’

</td>
<td>

Output:

isPlayerAtPos = true;

State of the board remains unchanged

</td>
<td>

Reason:

This test is unique and distinct due to the checking of the pos (1,1) matching the char p (O)

Function Name:
testisPlayerAtPos_Test_One
PIAP_SB

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

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 3):

	O	
X	O	X

pos.getRow = 0
pos.getColumn = 1
p = ‘X’

</td>
<td>

Output:

isPlayerAtPos = false;

State of the board remains unchanged

</td>
<td>

Reason:

This test is unique and distinct due to the checking of the pos (0,1) **is not** matching the char p (X)

Function Name:
testisPlayerAtPos_Test_Two
PINOTAP_SB

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

<table>
<tr>
<td>

Input:

State of Board(Num: to Win 5):

				V
				L
				K
J	K	L	V	J

pos.getRow = 0
pos.getColumn = 2
p = ‘L’

</td>
<td>

Output:

isPlayerAtPos = true;

State of the board remains unchanged

</td>
<td>

Reason:

This test is unique and distinct due to the checking of the pos (0,3) matching the char p (L), while there being 4 players present on the board.

Function Name:
testisPlayerAtPos_Test_Three
PIAPFiveByFive_MB

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

<table>
<tr>
<td>Input:

State of Board(Num: to Win 5):
<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>V</td></tr>
<tr><td></td><td></td><td></td><td></td><td>L</td></tr>
<tr><td></td><td></td><td></td><td></td><td>K</td></tr>
<tr><td>J</td><td>K</td><td>L</td><td>V</td><td>J</td></tr>
</table>
pos.getRow = 3
pos.getColumn = 4
p = ‘L’</td>
<td>Output:

isPlayerAtPos = false;

State of the board remains unchanged</td>
<td>Reason:

This test is unique and distinct due to the checking of the pos (3,4) is not matching the char p (L), while there being 4 players present on the board.

Function Name:
testisPlayerAtPos_Test_Four PINAPFiveByFive_MB</td>
</tr>
</table>

<table>
<tr>
<td>Input:

State of Board(Num: to Win 5):
<table>
<tr><td>J</td><td>J</td><td>V</td><td>J</td><td>J</td></tr>
<tr><td>L</td><td>L</td><td>L</td><td>V</td><td>V</td></tr>
<tr><td>V</td><td>V</td><td>K</td><td>L</td><td>L</td></tr>
<tr><td>K</td><td>K</td><td>J</td><td>K</td><td>K</td></tr>
<tr><td>J</td><td>K</td><td>L</td><td>V</td><td>J</td></tr>
</table>
pos.getRow = 4
pos.getColumn = 0
p = ‘J’</td>
<td>Output:

isPlayerAtPos = true;

State of the board remains unchanged</td>
<td>Reason:

This test is unique and distinct due to the checking of the pos (4,0, top Left of the board) matching the char p (J), while there being 4 players present on the board and the Board is full.

Function Name:
testisPlayerAtPos_Test_Five PIAPFULLFiveByFive_MB</td>
</tr>
</table>

placeToken(char p, int c)

<table>
<tr>
<td>Input:

State of Board:
<table>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td></tr>
</table>
p = ‘X’
c = 1</td>
<td>Output

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td></td></tr>
</table></td>
<td>Reason:

This test is unique and distinct due to it entering a unique char (X) into a new column.

Function name:
testplaceToken_Test_One PT_Unique_Char</td>
</tr>
</table>

Input:

State of Board:

O	O	
X	X	X
O	O	X

p = ‘O’
c = 2

Output

State of Board:

O	O	O
X	X	X
O	O	X

Reason:

This test is unique and distinct due to it entering the char (O) making it a full Board.

Function name:
testplaceToken_Test_Two
PT_Full_Board

Input:

State of Board:

p = ‘X’
c = 1

Output

State of Board:

X		

Reason:

This test is unique and distinct due to it entering a char (X) to a blank board.

Function name:
testplaceToken_Test_Three
PT_Empty_Board

Input:

State of Board:

	C	X	V	
X	C	V	X	C

p = ‘V’
c = 2

Output

State of Board:

		V		
	C	X	V	
X	C	V	X	C

Reason:
This test is unique and distinct due to it entering the char (V), its slightly larger board and is placed on the second row, second column
Function name:
testplaceToken_Test_Four
PT_Bigger_Board_HalfFull

<table>
<tr>
<td>Input:

State of Board:
<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>X</td><td></td><td></td><td></td><td></td></tr>
</table>

p = ‘V’
c = 0</td>
<td>Output

State of Board:
<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>V</td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td>Reason:

This test is unique and distinct due to it entering the char (V), its slightly larger board and is placed on top of another char in a column.

Function name:
testplaceToken_Test_Five PT_Bigger_Board_Ontop</td>
</tr>
</table>