Mike Brandin

Dr. Yu-Shan Sun

CPSC 2150 Section 002

April 4th, 2021

Project Report:

Project 4

Requirements Analysis

Functional Requirements

Gamescreen.java

- As a user, I can input a column number, so I can mark a spot on the game board and progress the game.
- As a user, I can input a Y or an N after the game has concluded, to start a new game or terminate the game session, respectively.
- As a user, I can place four markers touching horizontally, to conclude the match and receive a congratulatory message and a request to play again.
- As a user, I can place four markers touching vertically, to conclude the match and receive a congratulatory message and a request to play again.
- As a user, I can place four markers touching diagonally, to conclude the match and receive a congratulatory message and a request to play again.
- As a user, I can fill the board fully with markers and not win, to conclude the match with a tie and a request to play again.
- As a user, I can input a column value not within the confines of the gameboard and receive an error message followed by a prompt to re-enter the value.
- As a user, I can input an integer value for the number of players, so that I the number of players can be established.
- As a user, I can input a character value for each player, to represent the player's tokens on the gameboard.
- As a user, I can input a row length, to establish the length of the rows on the gameboard.
- As a user, I can input a column length, to establish the length of the columns on the gameboard.
- As a user, I can input a number to win, to establish the desired number needed to achieve in order to win the game.
- As a user, I can input an f/F or m/M, to decide whether or not to run the gameboard as a fast implementation or a more memory efficient implementation.

Gameboard.java

- As a user, I must be able to input a position value and receive character value, to know what player is in that position.
- As a user, I must be able to input a column number, to input a token into the highest available row in that column.
- As a user, I must be able to call a function to receive the value of the number of rows.
- As a user, I must be able to call a function to receive the value of the number of columns.
- As a user, I must be able to call a function to receive the value of the number needed to win.

GameboardMem.java

- As a user, I must be able to input a position value and receive character value, to know what player is in that position.
- As a user, I must be able to input a column number, to input a token into the highest available row in that column.

- As a user, I must be able to call a function to receive the value of the number of rows.
- As a user, I must be able to call a function to receive the value of the number of columns.
- As a user, I must be able to call a function to receive the value of the number needed to win.

AbsGameboard.java

- As a user, I must be able to request a fully formatted string representation of the gameboard, to visualize the current gameboard.

IGameboard.java

- As a user, I must be able to input a column number and receive a true or false value, to know whether or not a column is free for more tokens.
- As a user, I must be able to input a column number and receive a true or false value, to know whether or not the last placed token resulted in a win.
- As a user, I must be able to input a column number and receive a true or false value, to know whether or not the last placed token resulted in a tie (a full board).
- As a user, I must be able to input a character token and a position value and receive a true or false value, to know whether or not the last placed token resulted in a horizontal win.
- As a user, I must be able to input a character token and a position value and receive a true or false value, to know whether or not the last placed token resulted in a vertical win.
- As a user, I must be able to input a character token and a position value and receive a true or false value, to know whether or not the last placed token resulted in a diagonal win.
- As a user, I must be able to input a character representing a player and a position value and receive a true or false value, to know whether or not that player is in that position.

BoardPosition.java

- As a user, I must be able to receive a string output to display the row and column coordinates of the position.
- As a user, I must be able to request the Row variables value, to know the value of the board position's row.
- As a user, I must be able to request the Column variables value, to know the value of the board position's column.
- As a user, I must be able to compare two BoardPosition variables, to know whether their positional values are equivalent.

Nonfunctional Requirements

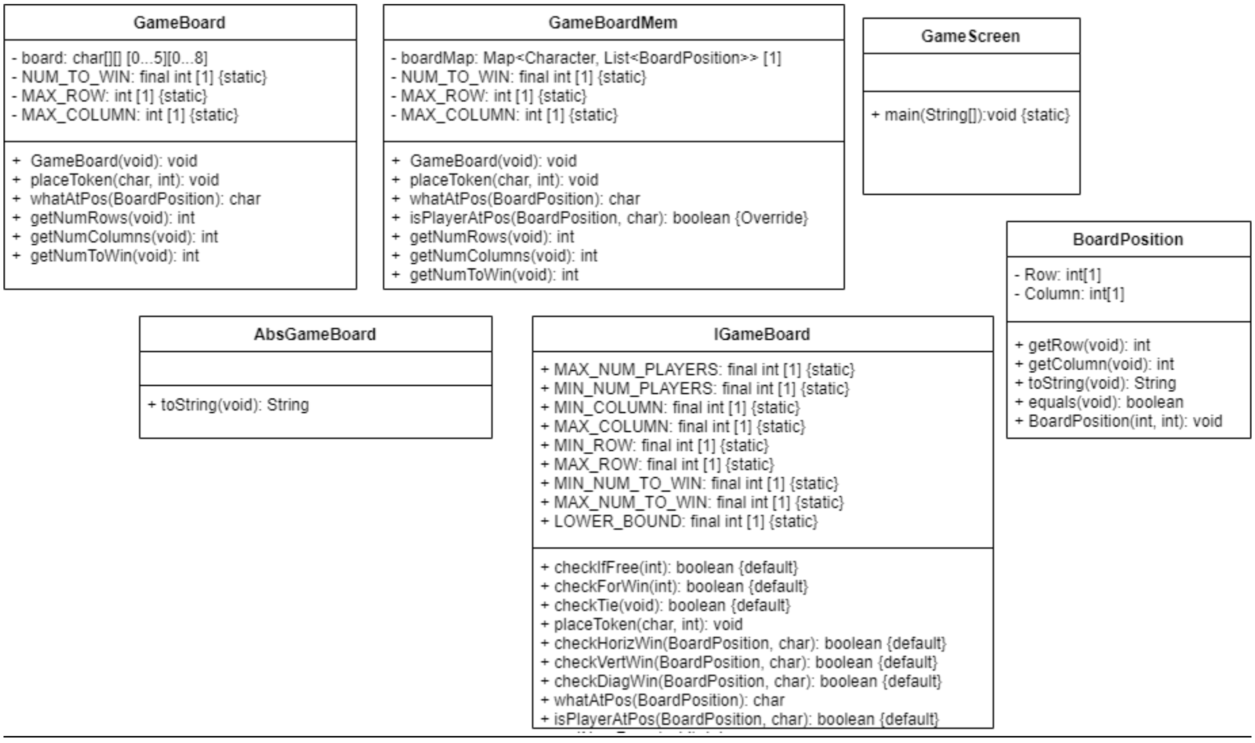
- Must have a device that supports Java.
- Must have a keyboard to play the game.
- Must run on the Schools of Computing's virtual machine.
- Must have adequate memory to allocate towards objects.
- Must handle all I/O in GameScreen.java
- Gameboard size must not exceed 100 columns nor 100 rows.
- Gameboard size must not be less than 3 columns nor 3 rows.
- Players must take turns in the order they selected their character at the start of the game.

Make File Instructions

To use the make file, open the terminal in the Project4 directory. Type **make** into the terminal followed by enter to compile the project files. Then, type **make run** into the terminal followed by enter to run the program. If you would like to begin, first start by entering **test** into the terminal to compile all test files then if you would like to run the GameBoard fast implementation enter **make testGB** into the terminal and if you would like to run the Gameboard Memory implementation enter **make testGBmem** into the terminal. Once you are finished, type **make clean** into the terminal followed by enter to remove all compiled files from the extendedConnectX package directory.

Design

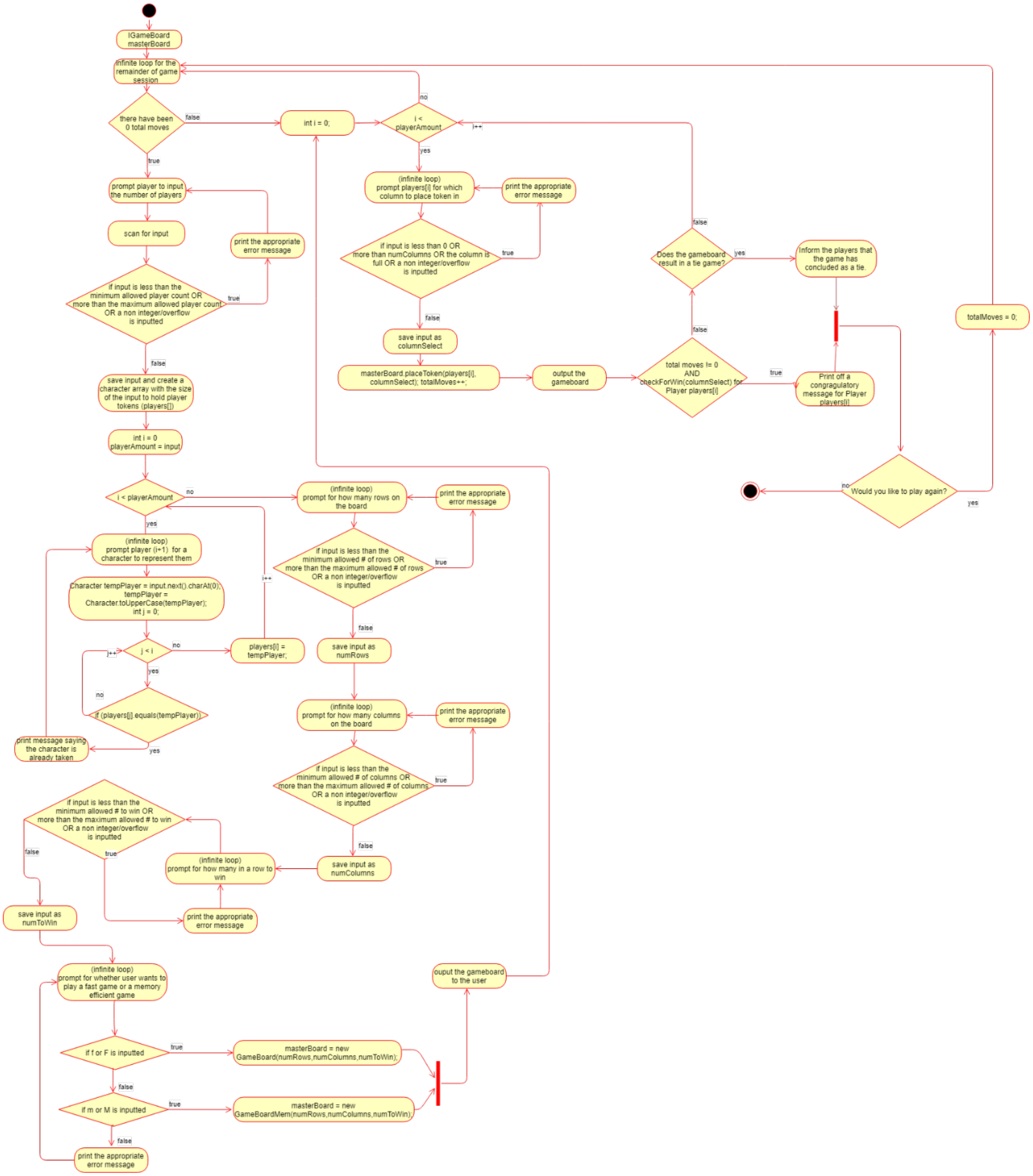
UML Class Diagrams


GameBoard
- board: char[][] [0...5][0...8]
- NUM_TO_WIN: final int [1] {static}
- MAX_ROW: int [1] {static}
- MAX_COLUMN: int [1] {static}
+ GameBoard(void): void
+ placeToken(char, int): void
+ whatAtPos(BoardPosition): char
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int
GameBoardMem
- boardMap: Map<Character, List<BoardPosition>> [1]
- NUM_TO_WIN: final int [1] {static}
- MAX_ROW: int [1] {static}
- MAX_COLUMN: int [1] {static}
+ GameBoard(void): void
+ placeToken(char, int): void
+ whatAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean {Override}
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int
GameScreen
+ main(String[]):void {static}
BoardPosition
- Row: int[1]
- Column: int[1]
+ getRow(void): int
+ getColumn(void): int
+ toString(void): String
+ equals(void): boolean
+ BoardPosition(int, int): void
AbsGameBoard
+ toString(void): String
IGameBoard
+ MAX_NUM_PLAYERS: final int [1] {static}
+ MIN_NUM_PLAYERS: final int [1] {static}
+ MIN_COLUMN: final int [1] {static}
+ MAX_COLUMN: final int [1] {static}
+ MIN_ROW: final int [1] {static}
+ MAX_ROW: final int [1] {static}
+ MIN_NUM_TO_WIN: final int [1] {static}
+ MAX_NUM_TO_WIN: final int [1] {static}
+ LOWER_BOUND: final int [1] {static}
+ checkIfFree(int): boolean {default}
+ checkForWin(int): boolean {default}
+ checkTie(void): boolean {default}
+ placeToken(char, int): void
+ checkHorizWin(BoardPosition, char): boolean {default}
+ checkVertWin(BoardPosition, char): boolean {default}
+ checkDiagWin(BoardPosition, char): boolean {default}
+ whatAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean {default}

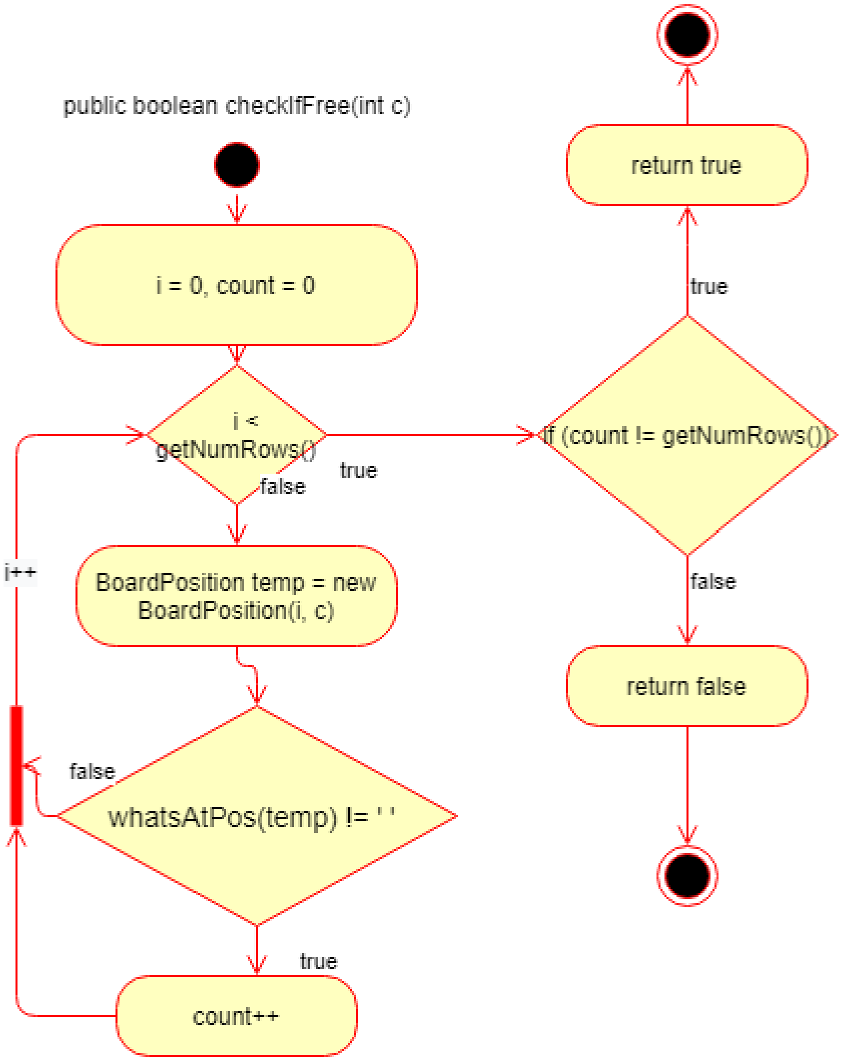
UML Activity Diagrams

- GameScreen.java:
  - main function

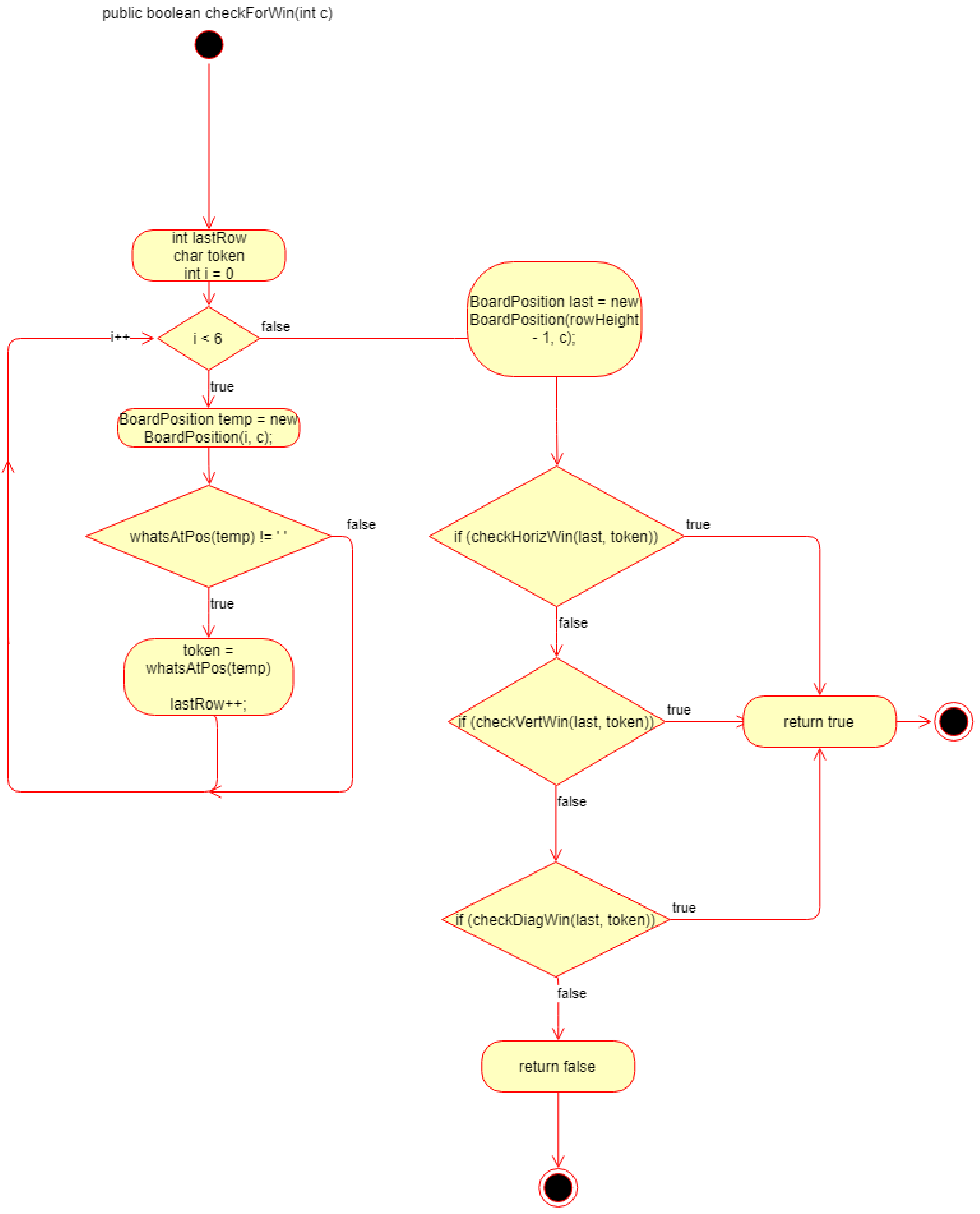
public static void main(String[] args)


IGameBoard masterBoard
infinite loop for the remainder of game session
there have been 0 total moves
false
true
prompt player to input the number of players
scan for input
print the appropriate error message
if input is less than the minimum allowed player count OR more than the maximum allowed player count OR a non integer/overflow is inputted
true
false
save input and create a character array with the size of the input to hold player tokens (players[])
int i = 0 playerAmount = input
i < playerAmount
no
yes
(infinite loop) prompt player (i+1) for a character to represent them
Character tempPlayer = input.next().charAt(0); tempPlayer = Character.toUpperCase(tempPlayer); int j = 0;
j < i
j++
no
yes
players[i] = tempPlayer;
i++
no
if (players[j].equals(tempPlayer))
yes
print message saying the character is already taken
(infinite loop) prompt for how many rows on the board
print the appropriate error message
if input is less than the minimum allowed # of rows OR more than the maximum allowed # of rows OR a non integer/overflow is inputted
true
false
save input as numRows
(infinite loop) prompt for how many columns on the board
print the appropriate error message
if input is less than the minimum allowed # of columns OR more than the maximum allowed # of columns OR a non integer/overflow is inputted
true
false
save input as numColumns
(infinite loop) prompt for how many in a row to win
print the appropriate error message
if input is less than the minimum allowed # to win OR more than the maximum allowed # to win OR a non integer/overflow is inputted
false
true
save input as numToWin
(infinite loop) prompt for whether user wants to play a fast game or a memory efficient game
if f or F is inputted
true
masterBoard = new GameBoard(numRows,numColumns,numToWin);
false
if m or M is inputted
true
masterBoard = new GameBoardMem(numRows,numColumns,numToWin);
false
print the appropriate error message
ouput the gameboard to the user
int i = 0;
i < playerAmount
no
i++
yes
(infinite loop) prompt players[i] for which column to place token in
print the appropriate error message
if input is less than 0 OR more than numColumns OR the column is full OR a non integer/overflow is inputted
true
false
save input as columnSelect
masterBoard.placeToken(players[i], columnSelect); totalMoves++;
output the gameboard
total moves != 0 AND checkForWin(columnSelect) for Player players[i]
true
false
Does the gameboard result in a tie game?
yes
false
Inform the players that the game has concluded as a tie.
Print off a congragulatory message for Player players[i]
Would you like to play again?
no
yes
totalMoves = 0;

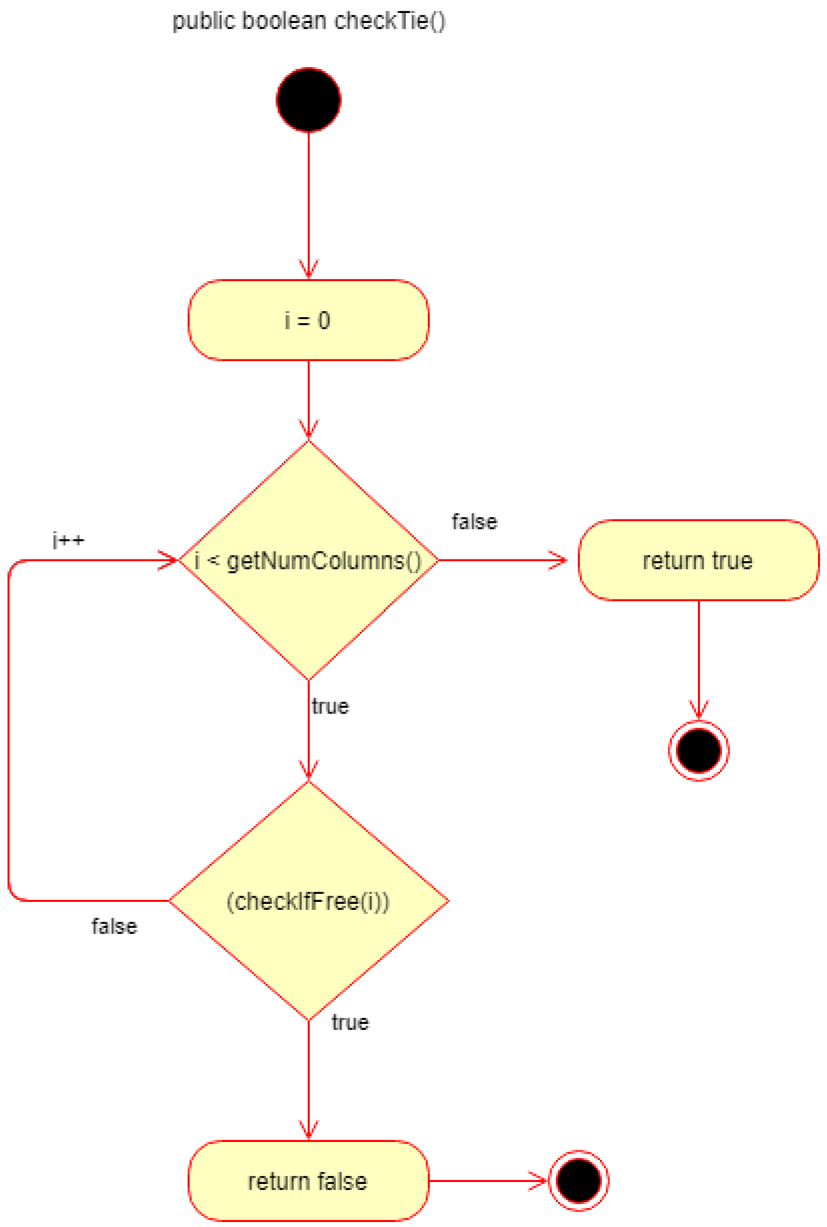
- IGameBoard.java
    - o checkIfFree method



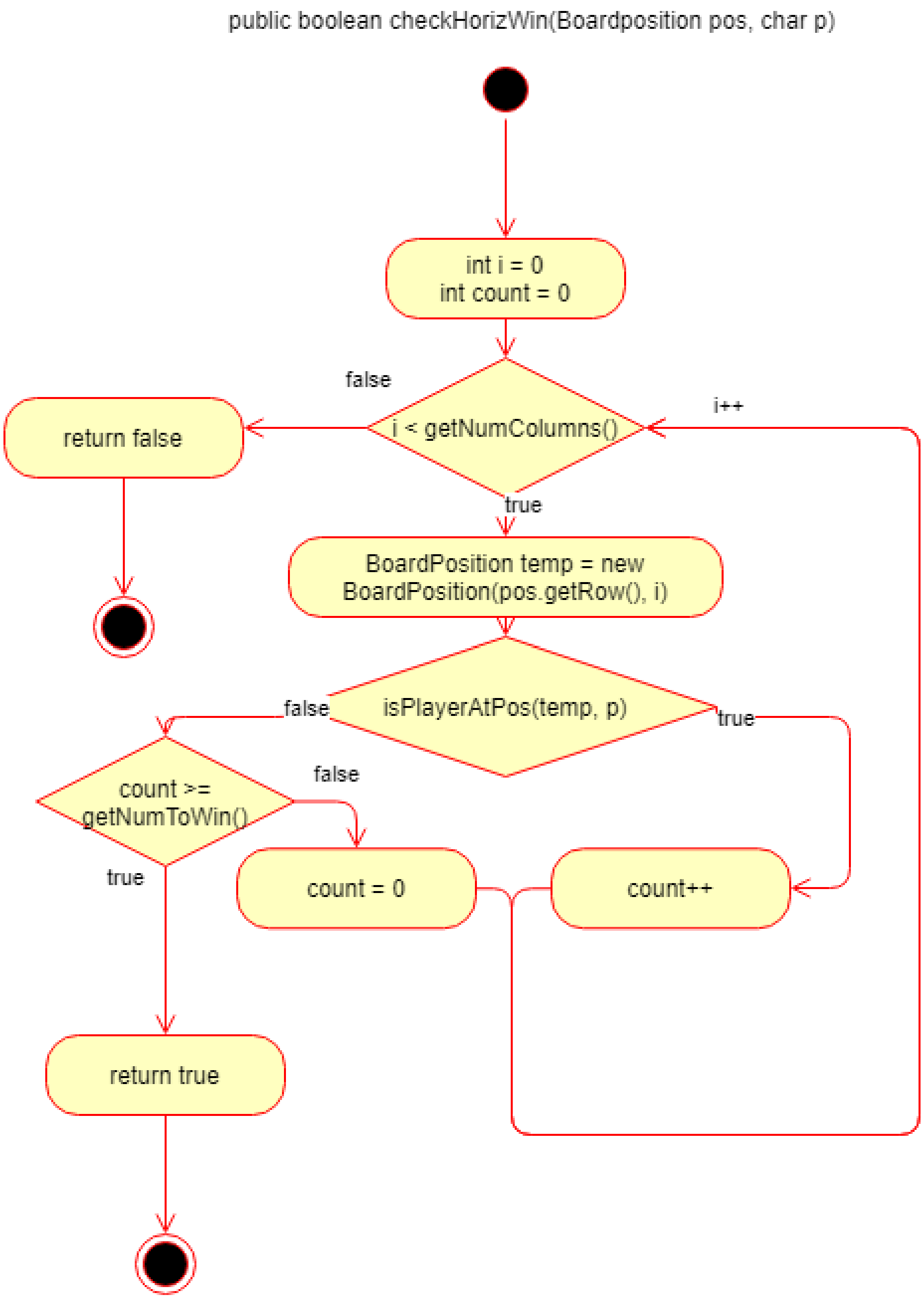
- checkForWin method


public boolean checkForWin(int c)
int lastRow
char token
int i = 0
i++
i < 6
false
true
BoardPosition last = new BoardPosition(rowHeight - 1, c);
BoardPosition temp = new BoardPosition(i, c);
whatsAtPos(temp) != ' '
false
true
token = whatsAtPos(temp)
lastRow++;
if (checkHorizWin(last, token))
true
false
if (checkVertWin(last, token))
true
return true
false
if (checkDiagWin(last, token))
true
false
return false

- checkTie method

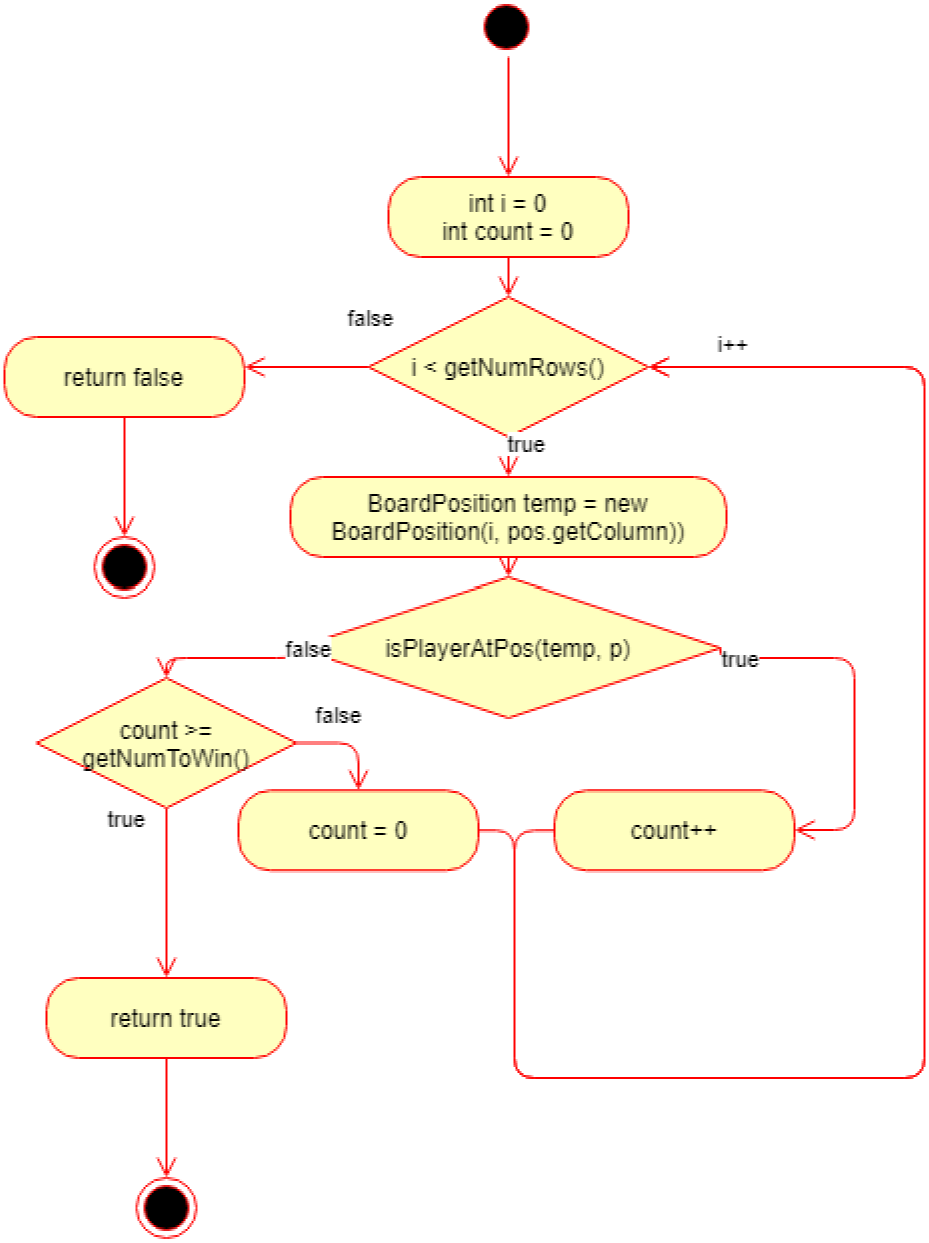


- checkHorizWin method

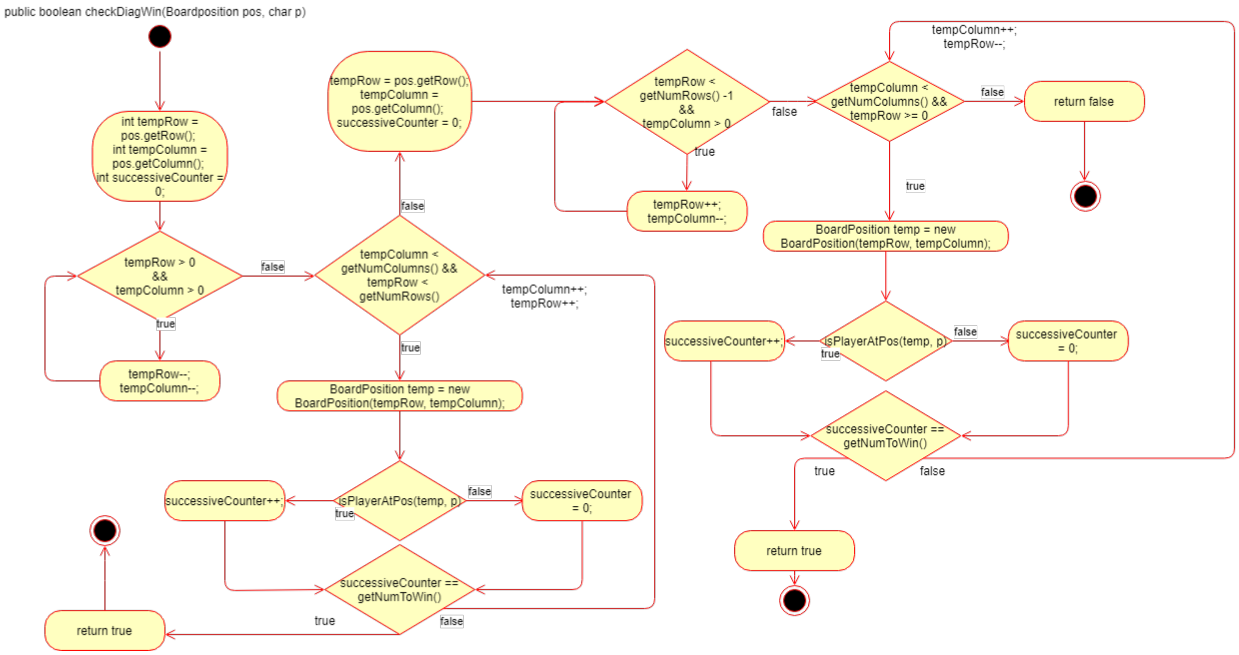


- checkVertWin method

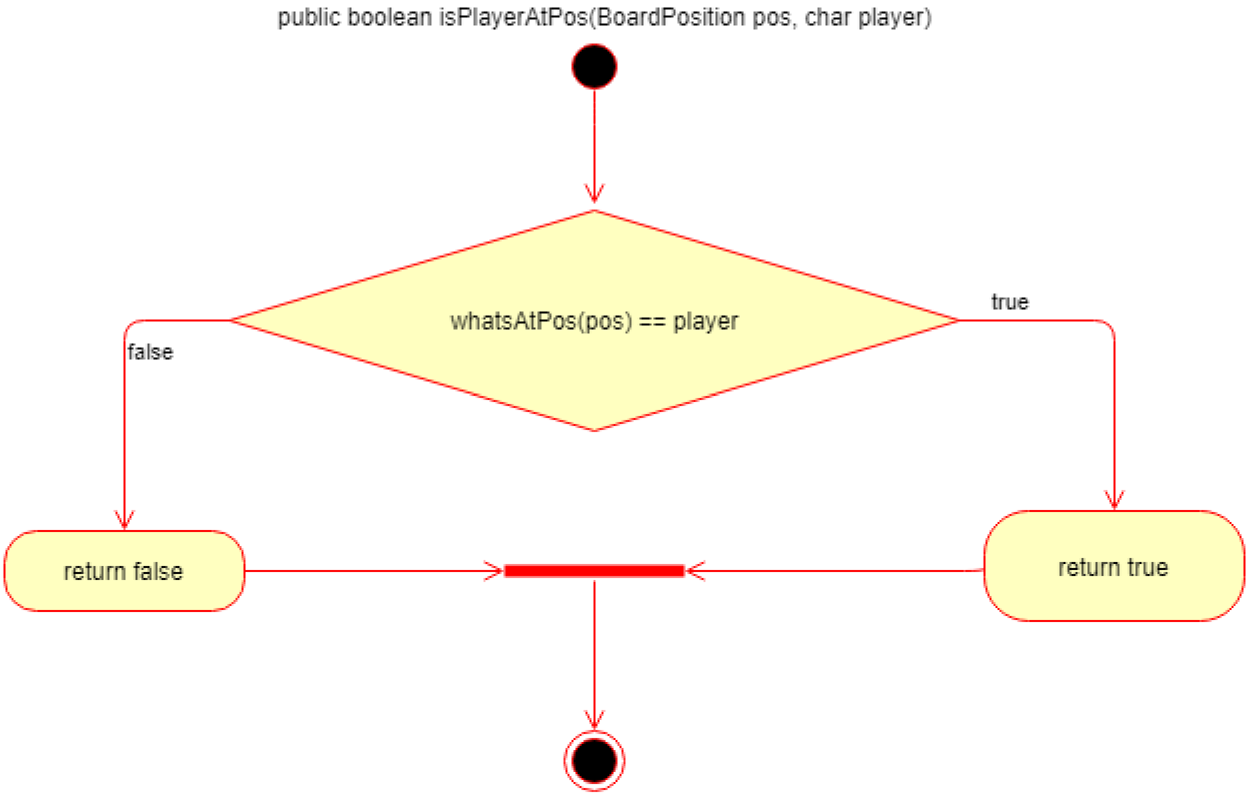
public boolean checkVertWin(Boardposition pos, char p)



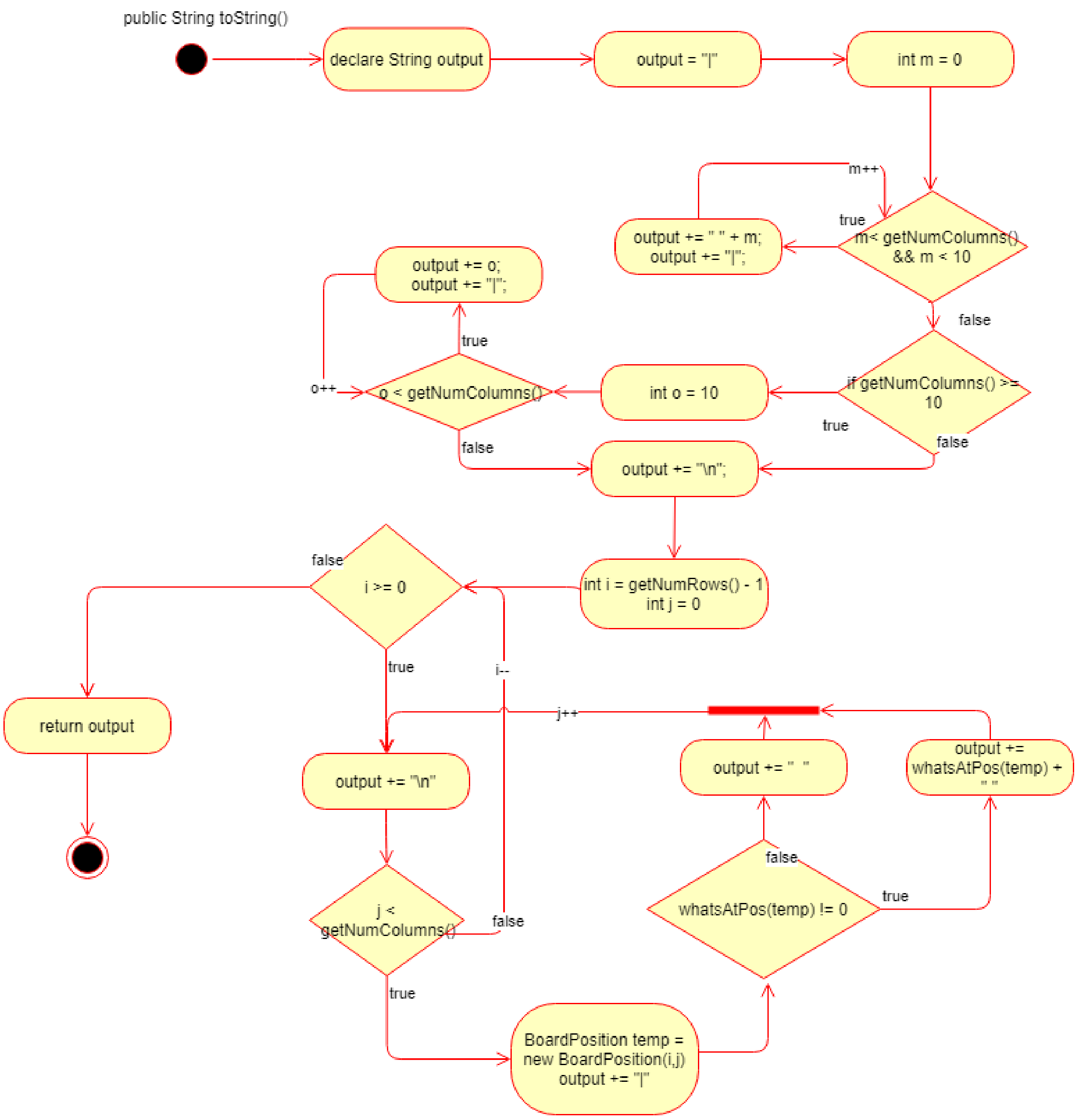
- ○ checkDiagWin method


public boolean checkDiagWin(Boardposition pos, char p)
int tempRow = pos.getRow(); int tempColumn = pos.getColumn(); int successiveCounter = 0;
tempRow > 0 && tempColumn > 0
true
false
tempRow--; tempColumn--;
tempColumn < getNumColumns() && tempRow < getNumRows()
false
true
tempRow = pos.getRow(); tempColumn = pos.getColumn(); successiveCounter = 0;
tempColumn++; tempRow++;
BoardPosition temp = new BoardPosition(tempRow, tempColumn);
successiveCounter++;
isPlayerAtPos(temp, p)
true
false
successiveCounter = 0;
successiveCounter == getNumToWin()
true
false
return true
tempRow < getNumRows() -1 && tempColumn > 0
false
true
tempRow++; tempColumn--;
tempColumn++; tempRow--;
tempColumn < getNumColumns() && tempRow >= 0
false
return false
true
BoardPosition temp = new BoardPosition(tempRow, tempColumn);
successiveCounter++;
isPlayerAtPos(temp, p)
true
false
successiveCounter = 0;
successiveCounter == getNumToWin()
true
false
return true

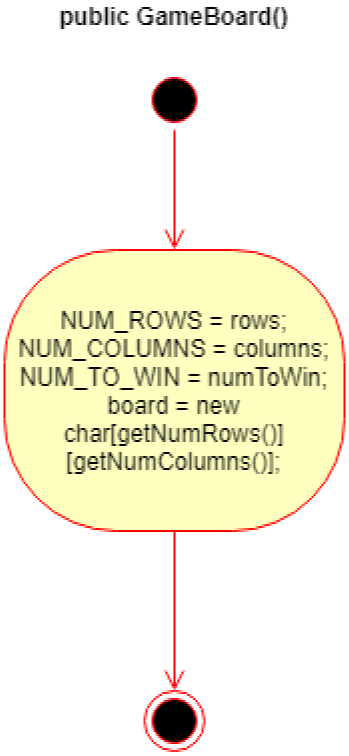
- isPlayerAtPos method


public boolean isPlayerAtPos(BoardPosition pos, char player)
whatsAtPos(pos) == player
false
true
return false
return true

- AbsGameBoard.java
    - toString method

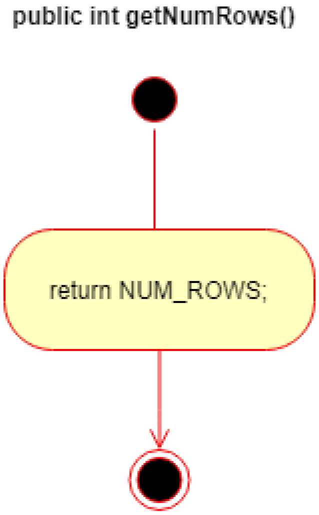


- GameBoard.java
  - GameBoard constructor



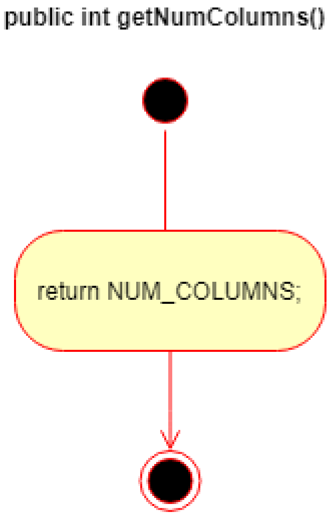
```
NUM_ROWS = rows;
NUM_COLUMNS = columns;
NUM_TO_WIN = numToWin;
board = new
char[getNumRows()]
[getNumColumns()];
```

- ○ getNumRows method

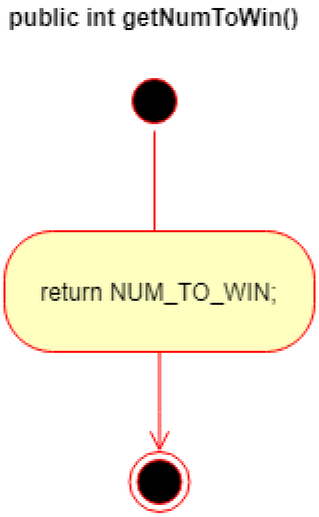


```
return NUM_ROWS;
```

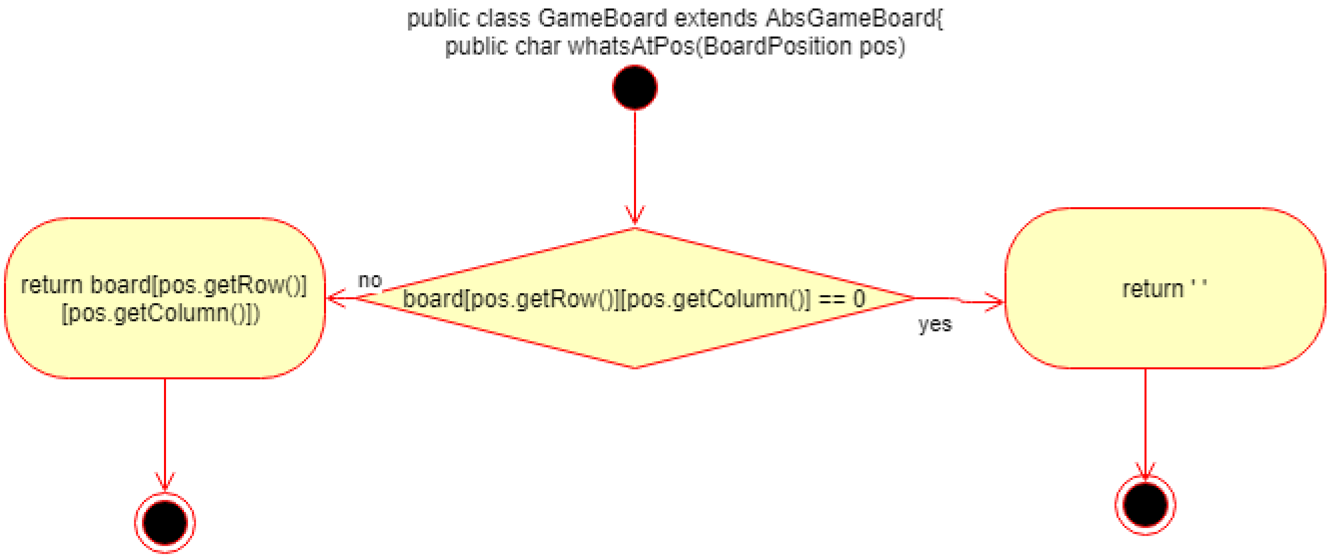
- ○ getNumColumns method



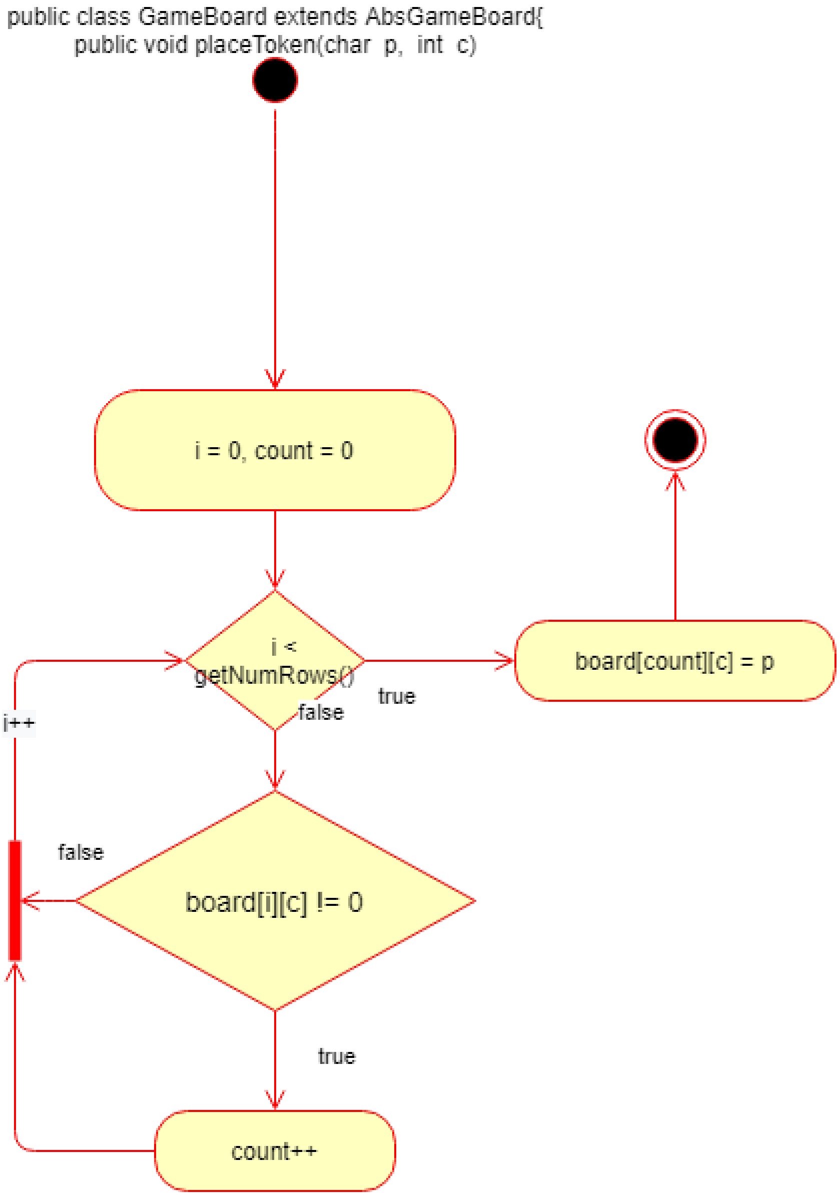
- ○ getNumToWin method



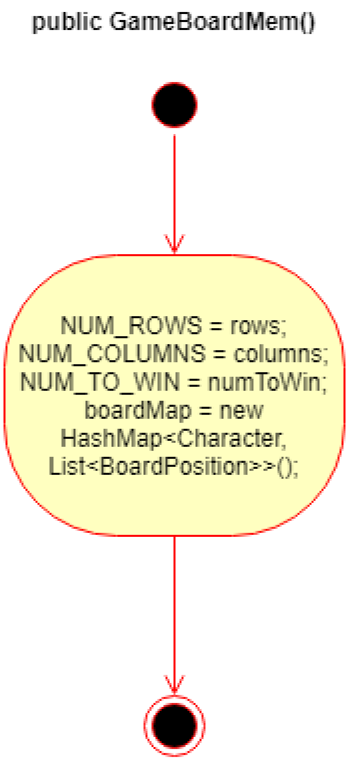
- ○ whatsAtPos method


public class GameBoard extends AbsGameBoard{
public char whatsAtPos(BoardPosition pos)
board[pos.getRow()][pos.getColumn()] == 0
no
yes
return board[pos.getRow()][pos.getColumn()])
return ' '

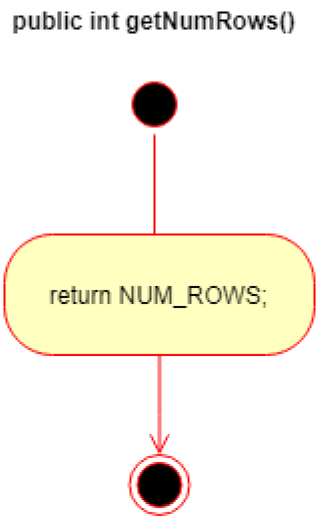
- placeToken method



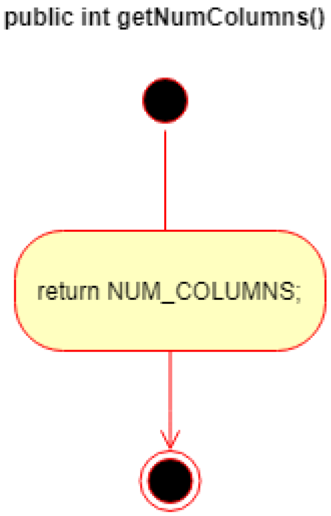
- GameBoardMem.java
    - GameBoardMem constructor



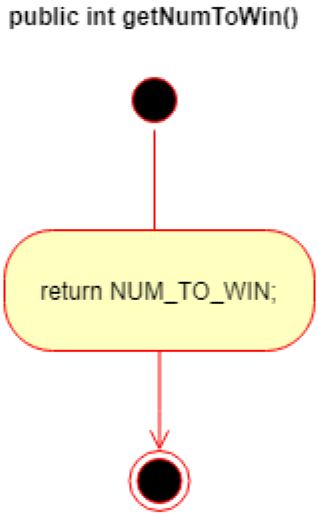
- getNumRows method



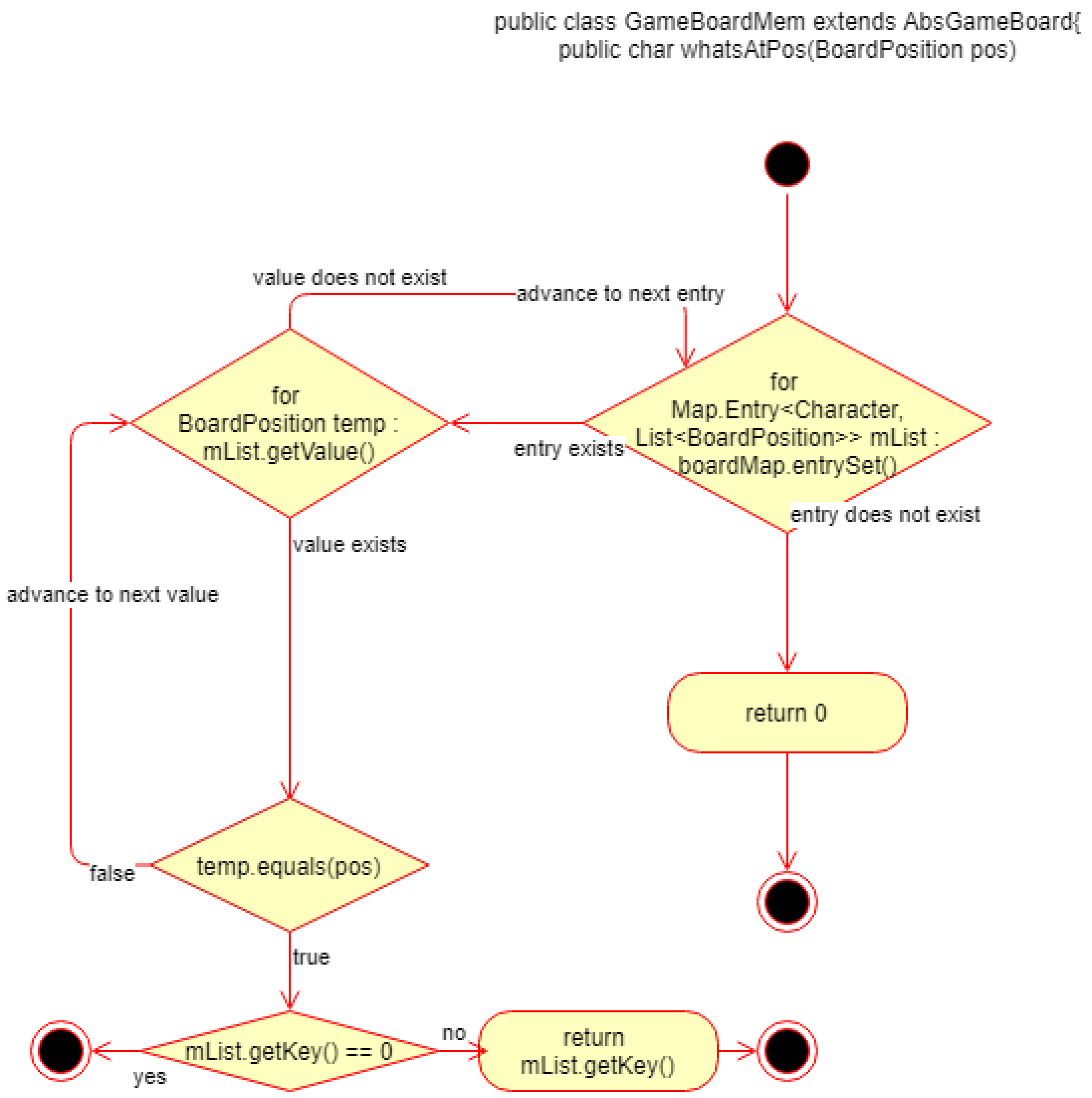
- getNumColumns method



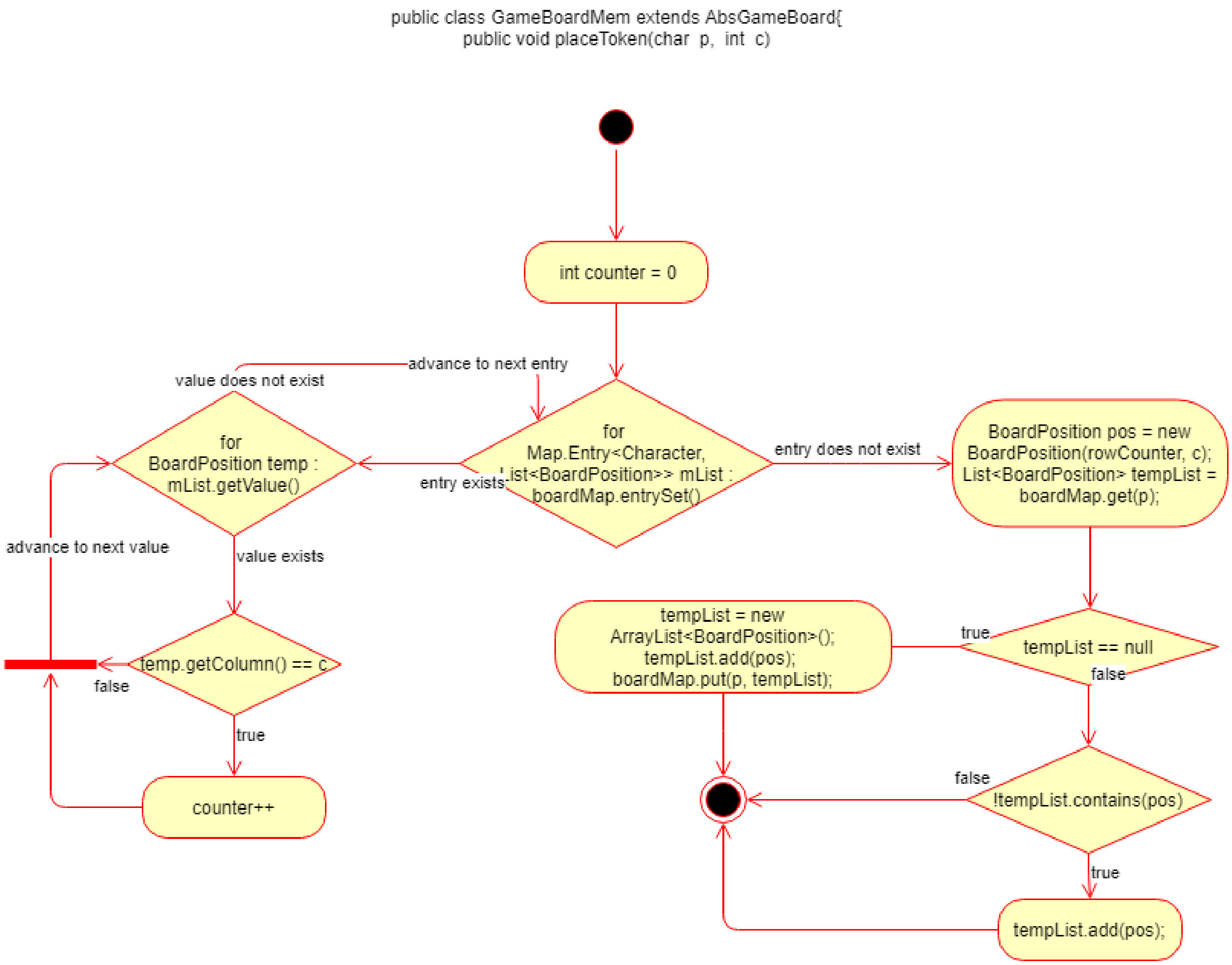
- getNumToWin method



- whatsAtPos method

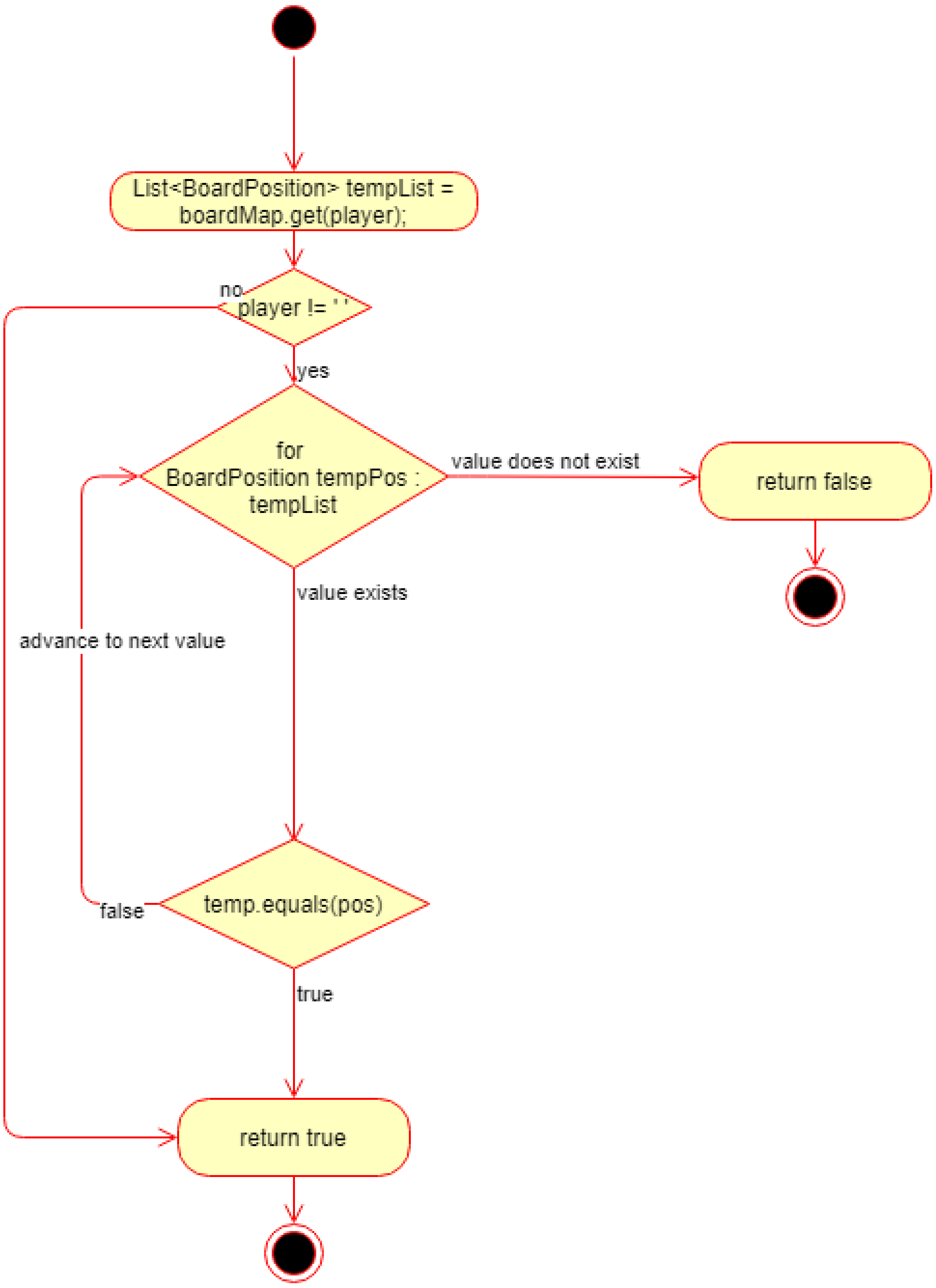


- placeToken method



- isPlayerAtPos method

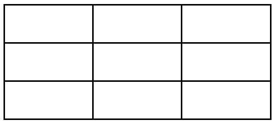
public class GameBoardMem extends AbsGameBoard{
public boolean isPlayerAtPos(BoardPosition pos, char player)



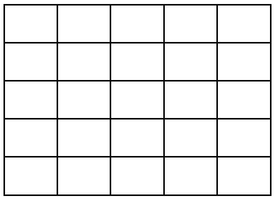
Testing

GameBoard(int rows, int columns, int numToWin)

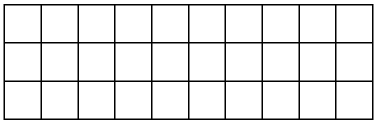
Input:	**Output:**	**Reason:**

rows = 3
columns = 3
numToWin = 3
	State:
	
		This test case is unique and distinct because I am testing the smallest possible row, columns, and numToWin inputs.
		Function Name:
		testConstructor_rows3_columns3_numToWin3

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

rows = 100
columns = 100
numToWin = 25
	State:
	
	(actual column and row size is 20 times larger)
		This test case is unique and distinct because I am testing the largest possible row, columns, and numToWin inputs.
		Function Name:
		testConstructor_rows100_columns100_numToWin25

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

rows = 3
columns = 100
numToWin = 3
	State:
	
	(actual column size is 10 times larger)
		This test case is unique and distinct because I am testing the largest possible column input with the smallest possible row and numToWin inputs.
		Function Name:
		testConstructor_rows3_columns100_numToWin3

boolean checkIfFree(int c)

<table>
<tr>
<td><b>Input:</b>

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

c = 5</td>
<td><b>Output:</b>

checkIfFree = true

state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because I am testing an empty board and thus testing an empty no boundary column.

<b>Function Name:</b>
testCheckIfFree_EmptyBoard_Col5</td>
</tr>
</table>

<table>
<tr>
<td><b>Input:</b>

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

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

checkIfFree = false

state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because I am testing a full board and testing the lower boundary input to make sure it returns false.

<b>Function Name:</b>
testCheckIfFree_FilledBoard_Col0</td>
</tr>
</table>

Input:

State:

X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X

c = 9

Output:

checkIfFree = true

state of the board is unchanged

Reason:
This test case is unique and distinct because a partially full board in the one column (that is also a boundary input) that has a free place to ensure that checkIfFree returns true.

Function Name:
testCheckIfFree_PartiallyFilledBoard_EmptyCol9Row9

boolean checkHorizWin(BoardPosition pos, char p)

Input:

State: (number to win = 5)

		X	X	X	X	X			

pos.getRow = 0
pos.getColumn = 4

p = ‘X’

Output:

checkHorizWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is the middle of the tokens and it involves no boundary cases.

Function Name:
testCheckHorizWin_NonBoundary_LastPlacedMiddle

Input:

State: (number to win = 5)

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

pos.getRow = rows(10) - 1
pos.getColumn = 2

p = ‘X’

Output:

checkHorizWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is the middle of the tokens the top left boundary place.

Function Name:
testCheckHorizWin_TopLeft_LastPlacedMiddle

Input:

State: (number to win = 5)

					X	X	X	X	X

pos.getRow = 0
pos.getColumn = columns(10) - 1

p = ‘X’

Output:

checkHorizWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is on the right and also on the lower bottom boundary.

Function Name:
testCheckHorizWin_BottomRight_LastPlacedRight

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 5)
<table>
<tr><td></td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><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><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
</table>
pos.getRow = 2
pos.getColumn = columns(10) - 1
p = ‘X’</td>
<td><b>Output:</b>
checkHorizWin = true
state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because the last placed position is on the right it tests both the left and right boundaries by taking up the entire row. Also, the number of consecutive X’s exceeds the numToWin to test.
<b>Function Name:</b>
testCheckHorizWin_R2_EntireRow_LastPlacedLeft</td>
</tr>
</table>

boolean checkVertWin(BoardPosition pos, char p)

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 5)
<table>
<tr><td>X</td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
</table>
pos.getRow = rows(10) - 1
pos.getColumn = 0
p = ‘X’</td>
<td><b>Output:</b>
checkVertWin = true
state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because the last placed position is on the upper left boundary of the game board in a partially filled board.
<b>Function Name:</b>
testCheckVertWin_TopLeft_C0</td>
</tr>
</table>

Input:

State: (number to win = 5)

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

pos.getRow = rows(10) - 1
pos.getColumn = column(10) - 1

p = ‘X’

Output:

checkVertWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is on the upper right boundary of the game board in a partially filled board.

Function Name:
testCheckVertWin_TopRight_C9

Input:

State: (number to win = 5)

									X
									X
									X
									X
									X

pos.getRow = numToWin(5) - 1
pos.getColumn = column(10) - 1

p = ‘X’

Output:

checkVertWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position right boundary and it involves the lower right boundary.

Function Name:
testCheckVertWin_BottomRight_C9

<table>
<tr><td><b>Input:</b>
State: (number to win = 5)
<table>
<tr><td></td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
</table>
pos.getRow = 7
pos.getColumn = 4
p = ‘X’</td>
<td><b>Output:</b>
checkVertWin = true
state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because the last placed position involves no boundary cases and is approximately in the center.
<b>Function Name:</b>
testCheckVertWin_MiddleCenter_C4</td></tr>
</table>

boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr><td><b>Input:</b>
State: (number to win = 5)
<table>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow = rows(10) - 1
pos.getColumn = 0
p = ‘X’</td>
<td><b>Output:</b>
checkDiagWin = true
state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because the last placed position is the top left boundary position, and the function must check in the NW/SE direction.
<b>Function Name:</b>
testCheckDiagWin_LastPlacedTopLeft_C0_NWSE</td></tr>
</table>

Input:

State: (number to win = 5)

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

pos.getRow = rows(10)-1

pos.getColumn = columns(10) -1

p = ‘X’

Output:

checkDiagWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is the top right boundary position, and the function must check in the NE/SW direction.

Function Name:
testCheckDiagWin_LastPlacedTopRight_C9_NESW

Input:

State: (number to win = 5)

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

pos.getRow = rows(10)-1

pos.getColumn = 0

p = ‘X’

Output:

checkDiagWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is the bottom right boundary position, and the function must check in the NW/SE direction.

Function Name:
testCheckDiagWin_LastPlacedBottomRight_C9_NWSE

Input:

State: (number to win = 5)

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

pos.getRow = 0

pos.getColumn = 0

p = ‘X’

Output:

checkDiagWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is the bottom left boundary position, and the function must check in the NE/SW direction.

Function Name:
testCheckDiagWin_LastPlacedBottomLeft_C0_NESW

Input:

State: (number to win = 5)

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

pos.getRow = 0

pos.getColumn = 0

p = ‘X’

Output:

checkDiagWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is the bottom left boundary position, and the top right boundary is also involved, and the function must check in the NE/SW direction.

Function Name:
testCheckDiagWin_LastPlacedLeftMost_C0_NESW_TopRight

Input:

State: (number to win = 5)

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

pos.getRow = 0

pos.getColumn = columns(10)-1

p = ‘X’

Output:

checkDiagWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is the bottom right boundary position, and the top left boundary position is also involved, and the function must check in the NW/SE direction.

Function Name:
testCheckDiagWin_LastPlacedRightMost_C9_NWSE_TopLeft

Input:

State: (number to win = 5)

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

pos.getRow = 6

pos.getColumn = 5

p = ‘X’

Output:

checkDiagWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last placed position is a nonboundary in the middle of a diagonal of X’s, and none of the X’s touch a boundary, and the function must check in the NW/SE direction.

Function Name:
testCheckDiagWin_LastPlacedRightMost_C9_NWSE_TopLeft

boolean checkTie()

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td></td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></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><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td><b>Output:</b>
checkTie = false
state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct we are testing an empty board for a tie.
<b>Function Name:</b>
testCheckTie_EmptyBoard</td>
</tr>
</table>

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><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><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><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><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><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><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><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><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><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><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>
</td>
<td><b>Output:</b>
checkTie = true
state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct we are testing a full board for a tie.
<b>Function Name:</b>
testCheckTie_FullBoard</td>
</tr>
</table>

Input:

State:

X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X

Output:

checkTie = false

state of the board is unchanged

Reason:
This test case is unique and distinct we are testing a nearly full board for a tie and this one empty spot is a bottom left boundary case.

Function Name:
testCheckTie_PartiallyFullBoard_EmptyPlaceTopRight

Input:

State:

X									

Output:

checkTie = false

state of the board is unchanged

Reason:
This test case is unique and distinct we are testing a nearly empty board for a tie and this one occupied spot is a bottom left boundary case.

Function Name:
testCheckTie_PartiallyFullBoard_OccupiedSpace_BottomLeft

char whatsAtPos(BoardPosition pos)

Input:

State:

pos.getRow = rows(10)-1

pos.getColumn = columns(10)-1

Output:

whatsAtPos = ‘ ‘

state of the board is unchanged

Reason:
This test case is unique and distinct because it tests the top right boundary case on an empty board.

Function Name:
testWhatsAtPos_EmptyBoard_TopRight

Input:

State:

X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X

pos.getRow = rows(10)-1

pos.getColumn = columns(10)-1

Output:

whatsAtPos = ‘X‘

state of the board is unchanged

Reason:
This test case is unique and distinct because it tests the top right boundary case on a full board.

Function Name:
testWhatsAtPos_FilledBoard_TopRight

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><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><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><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><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><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><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><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><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><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><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>
pos.getRow = 0
pos.getColumn = 0</td>
<td><b>Output:</b>
whatsAtPos = ‘X‘
state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests the bottom left boundary case on a full board.
<b>Function Name:</b>
testWhatsAtPos_FilledBoard_BottomLeft</td>
</tr>
</table>

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><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><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><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><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><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><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><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><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><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><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>
pos.getRow = rows(10)-1
pos.getColumn = columns(10)-1</td>
<td><b>Output:</b>
whatsAtPos = ‘ ‘
state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests the top right boundary case on a partially full board.
<b>Function Name:</b>
testWhatsAtPos_PartiallyFilledBoard_EmptyTopRight</td>
</tr>
</table>

<table>
<tr><td><b>Input:</b>

State:

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

pos.getRow = rows(10)-1

pos.getColumn = 0</td>
<td><b>Output:</b>

whatsAtPos = ‘O‘

state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests the top left boundary case on a partially filled board.

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

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr><td><b>Input:</b>

State:

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

pos.getRow = rows(10)-1

pos.getColumn = 0

player = ‘O’</td>
<td><b>Output:</b>

isPlayerAtPos = true

state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests the top left boundary for an occupied O when the column has alternating X’s and O’s on a partially full board.

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

<table>
<tr>
<td><b>Input:</b>

State:

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

pos.getRow = rows(10)-1

pos.getColumn = column(10)-1

player = ‘ ’</td>
<td><b>Output:</b>

isPlayerAtPos = true

state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests the top right boundary for an empty space when the rest of the gameboard is full of alternating X’s and O’s.

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

<table>
<tr>
<td><b>Input:</b>

State:

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

pos.getRow = rows(10)-1

pos.getColumn = 0

player = ‘X’</td>
<td><b>Output:</b>

isPlayerAtPos = true

state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests the top left boundary for an occupied X when the entire board is full of alternating X’s and O’s.

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

<table>
<tr><td><b>Input:</b>
State:
<table><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr></table>pos.getRow = 0
pos.getColumn = columns(10) - 1
player = ‘ ’</td><td><b>Output:</b>
isPlayerAtPos = true
state of the board is unchanged</td><td><b>Reason:</b>
This test case is unique and distinct because it tests the top left boundary for an empty space on an empty board.
<b>Function Name:</b>
testIsPlayerAtPos_EmptyBoard_TopLeft</td></tr>
</table>

void placeToken(char p, int c)

<table>
<tr><td><b>Input:</b>
State:
<table><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr></table>c = 0
p = ‘X’</td><td><b>Output:</b>
State:
<table><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td><td> </td><td> </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><td> </td><td> </td><td> </td><td> </td></tr></table></td><td><b>Reason:</b>
This test case is unique and distinct because it tests the bottom left boundary on an empty board of a nonboundary touching gameboard.
<b>Function Name:</b>
testPlaceToken_BottomLeft_rows10_columns10</td></tr>
</table>

Input:

State:

c = 2

p = ‘X’

Output:

State:

		x

Reason: This test case is unique and distinct because it tests the bottom right boundary on an empty board of minimum size.

Function Name: testPlaceToken_BottomRigh_rows3_columns3

Input:

State:

									O
									X
									O
									X
									O
									X
									O
									X
									O

(actual size of gameboard is 10x larger)

c = columns(10) - 1

p = ‘X’

Output:

State:

									X
									O
									X
									O
									X
									O
									X
									O
									X
									O

(actual size of gameboard is 10x larger)

Reason: This test case is unique and distinct because it tests the top right boundary on a partially full board of a max sized gameboard.

Function Name: testPlaceToken_TopRight_rows100_columns100

Input:

State:

O		
X		
O		
X		
O		
X		
O		
X		
O		

c = 0

p = ‘X’

Output:

State:

X		
O		
X		
O		
X		
O		
X		
O		
X		
O		

Reason: This test case is unique and distinct because it tests the top left boundary on a nearly full column where the gameboard is has the minimum size number of columns.

Function Name: testPlaceToken_TopLeft_rows10_columns_3

Input:

State:

					X				
					O				
					X				
					O				

c = columns(10)/2 = 5

p = ‘O’

Output:

State:

					O				
					X				
					O				
					X				
					O				

Reason: This test case is unique and distinct because it tests the center row and center column on a partially full board of a nonboundary touching gameboard size.

Function Name: testPlaceToken_Row5_Col5