CPSC 2150 Project Report

Mauricio Fortune

Requirements Analysis

Functional Requirements:

1. As a player I can pick a precise row and column on the tic tac toe board so that I can play against my opponents
2. As a player I can see the updated tic tac toe board after each round so that I can make a calculated next move
3. As a player I need row validation so that I do not pick a row that does not exist
4. As a player I need column validation so that I do not pick a column that does not exist
5. As a player I need the rows present on the printed tic tac toe board so that I know exactly where to make my next move
6. As a player I need the columns present on the printed tic tac toe board so that I know exactly where to make my next move
7. As a player I need to know who wins the match so that I can determine when the game is over
8. As a player I need to get prompted if I want to play again so that I can end the program when I am tired of playing or keep playing against my friends
9. As a player I need to be prompted to select the row in which I will make my move so that I can make a move
10. As a player I need to be prompted to select the column in which I will make my move so that I can make a move
11. As a player I can input my desired column using numbers and not words (6 not six).
12. As a player I need to win the game when I have enough markers in a row
13. As a player I need to lose the game when my opponent gets enough markers in a row
14. As a player I can pick which marker I want to use
15. As a player I can choose the size of the grid
16. As a player I can play against more than several opponents at once
17. As a player I can choose whether to use the fast method or the memory efficient method
18. As a player, if I pick an incorrect space (whether taken or out of bounds), I must be prompted to enter a new space
19. As a player, if there are no more spaces on the board, I must be informed that the game ended in a tie
20. As a player I can play against more than 2 opponents
21. As a player, my opponents and I should all take turns each round

Non-Functional Requirements

1. The system must be coded in Java
2. The system must prompt in the order that the players were given
3. The system must determine a win/loss or draw
4. The system must check if the player's placements are valid
5. The system must have a gameboard of a size selected by the user
6. The system must run on unix
7. The system must be a command line application
8. The system must prompt the player in order to enter the number of rows to win
9. The system must have 2 implementations, the fast one and the memory efficient one
10. The system must have a GUI that the player can press on to play
11.

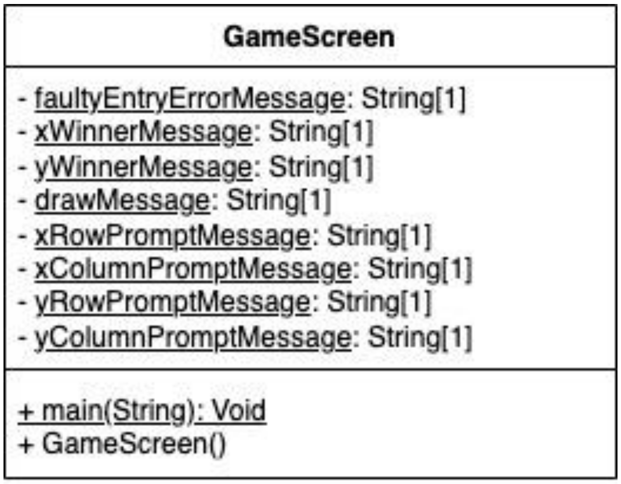
Deployment Instructions:

In order to run this program, you must use the makefile. In order to use the makefile, begin with running the command “make” this will compile all of the necessary files. Next use the command “make run” this will run the program. This command will allow you to begin playing tic tac toe. Once you are done, run the command “make clean” this will remove all of the class files that were created in order for this program to run.

System Design

Class 1: Game Screen

Class Diagram



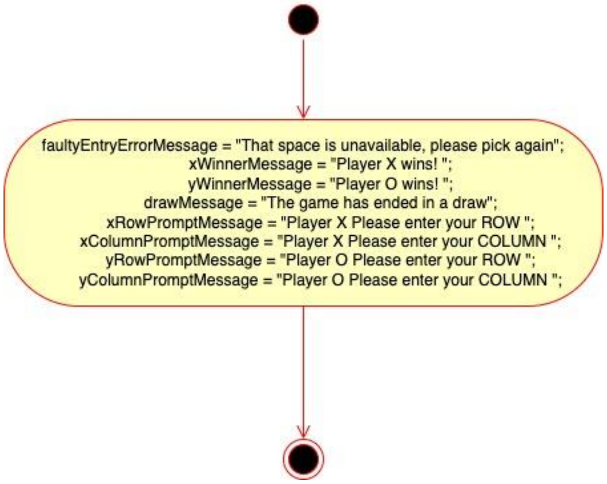
GameScreen

- faultyEntryErrorMessage: String[1]
- xWinnerMessage: String[1]
- yWinnerMessage: String[1]
- drawMessage: String[1]
- xRowPromptMessage: String[1]
- xColumnPromptMessage: String[1]
- yRowPromptMessage: String[1]
- yColumnPromptMessage: String[1]

+ main(String): Void
+ GameScreen()

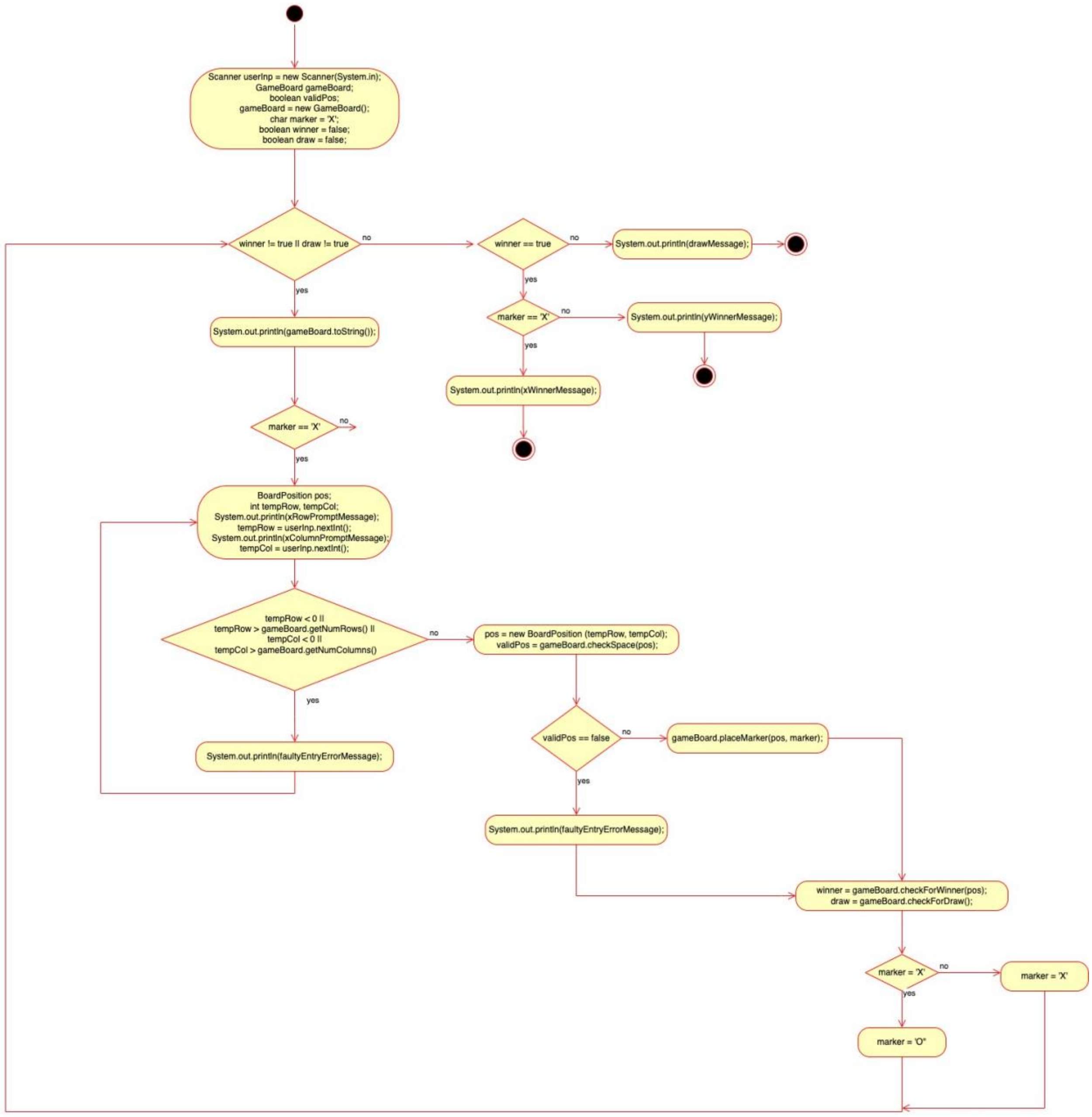
Activity Diagrams:

```
Public GameScreen()
```



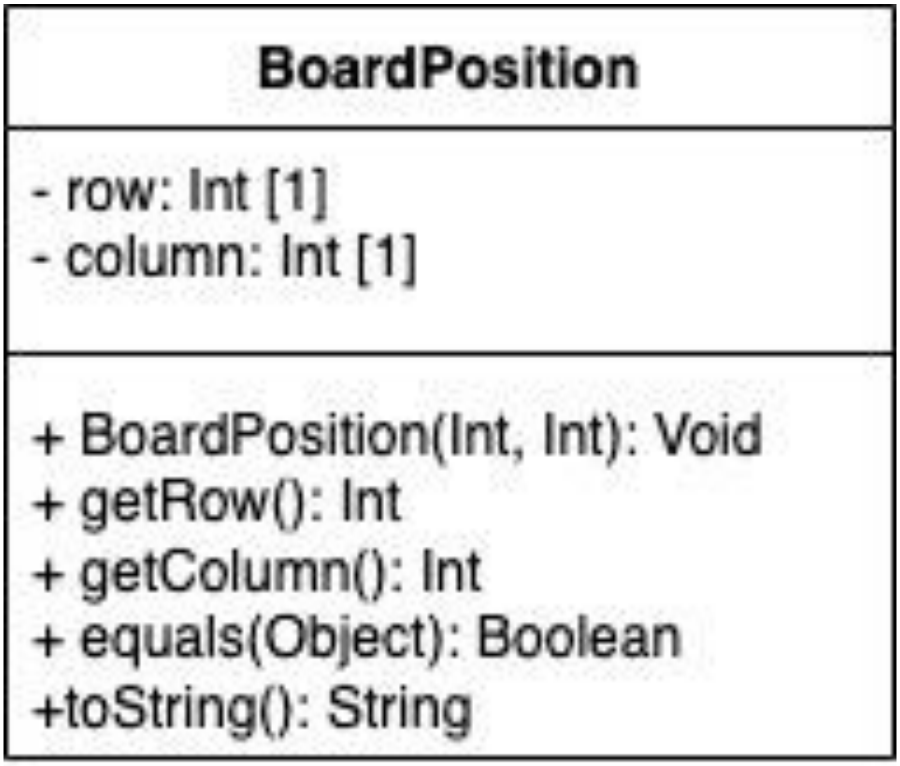
```
faultyEntryErrorMessage = "That space is unavailable, please pick again";
xWinnerMessage = "Player X wins! ";
yWinnerMessage = "Player O wins! ";
drawMessage = "The game has ended in a draw";
xRowPromptMessage = "Player X Please enter your ROW ";
xColumnPromptMessage = "Player X Please enter your COLUMN ";
yRowPromptMessage = "Player O Please enter your ROW ";
yColumnPromptMessage = "Player O Please enter your COLUMN ";
```

Public static void main(String[] args)


Scanner userInp = new Scanner(System.in);
GameBoard gameBoard;
boolean validPos;
gameBoard = new GameBoard();
char marker = 'X';
boolean winner = false;
boolean draw = false;
winner != true ll draw != true
no
winner == true
no
System.out.println(drawMessage);
yes
marker == 'X'
no
System.out.println(yWinnerMessage);
yes
System.out.println(xWinnerMessage);
yes
System.out.println(gameBoard.toString());
marker == 'X'
no
yes
BoardPosition pos;
int tempRow, tempCol;
System.out.println(xRowPromptMessage);
tempRow = userInp.nextInt();
System.out.println(xColumnPromptMessage);
tempCol = userInp.nextInt();
tempRow < 0 ll
tempRow > gameBoard.getNumRows() ll
tempCol < 0 ll
tempCol > gameBoard.getNumColumns()
no
pos = new BoardPosition (tempRow, tempCol);
validPos = gameBoard.checkSpace(pos);
yes
System.out.println(faultyEntryErrorMessage);
validPos == false
no
gameBoard.placeMarker(pos, marker);
yes
System.out.println(faultyEntryErrorMessage);
winner = gameBoard.checkForWinner(pos);
draw = gameBoard.checkForDraw();
marker = 'X'
no
marker = 'X'
yes
marker = 'O"

Class 2: Board Position

Class Diagram:



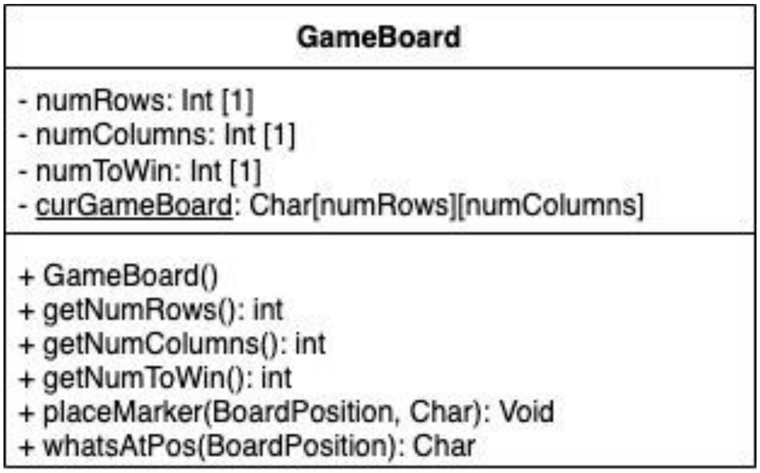
BoardPosition

- row: Int [1]
- column: Int [1]

+ BoardPosition(Int, Int): Void
+ getRow(): Int
+ getColumn(): Int
+ equals(Object): Boolean
+toString(): String

Class 3: Game Board

Class Diagram:



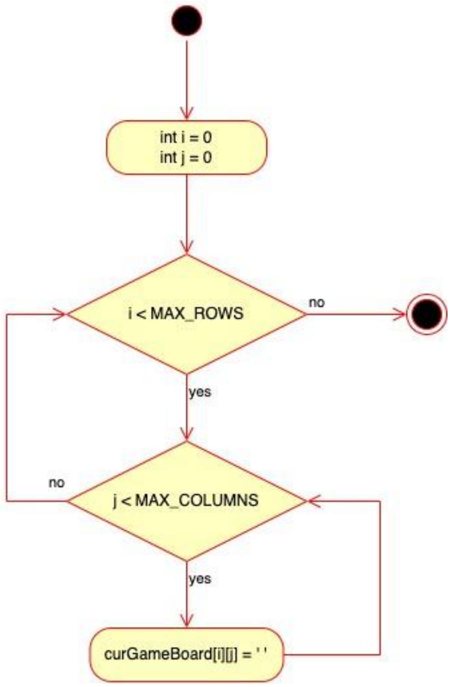
GameBoard

- numRows: Int [1]
- numColumns: Int [1]
- numToWin: Int [1]
- curGameBoard: Char[numRows][numColumns]

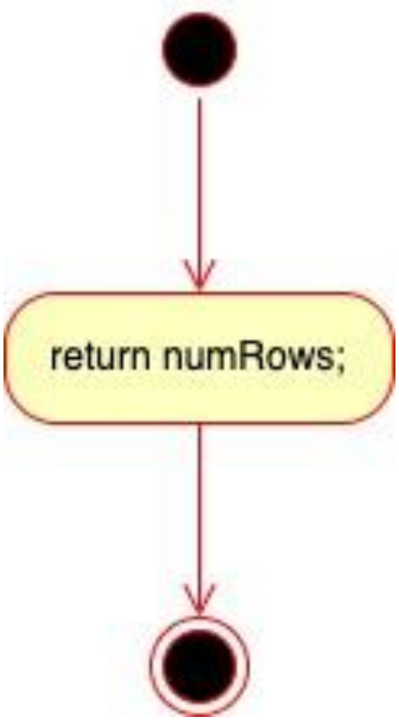
+ GameBoard()
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ placeMarker(BoardPosition, Char): Void
+ whatsAtPos(BoardPosition): Char

Activity Diagrams:

```
public GameBoard()
```

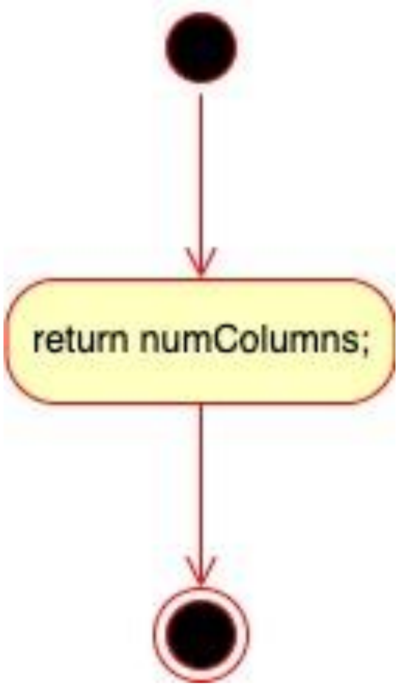


```
public int getNumRows()
```



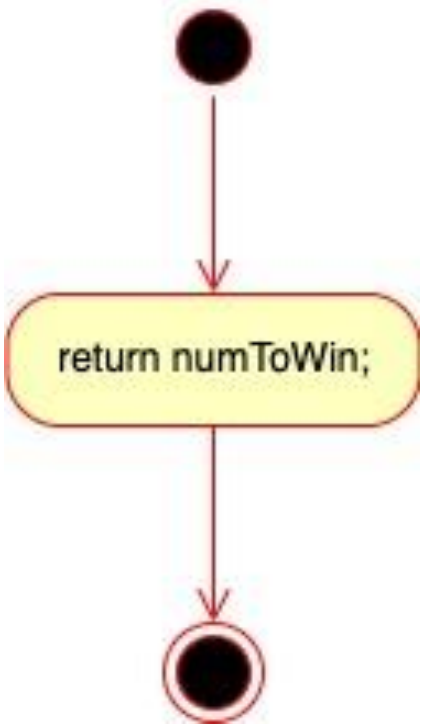
```
return numRows;
```

```
public int getNumColumns()
```

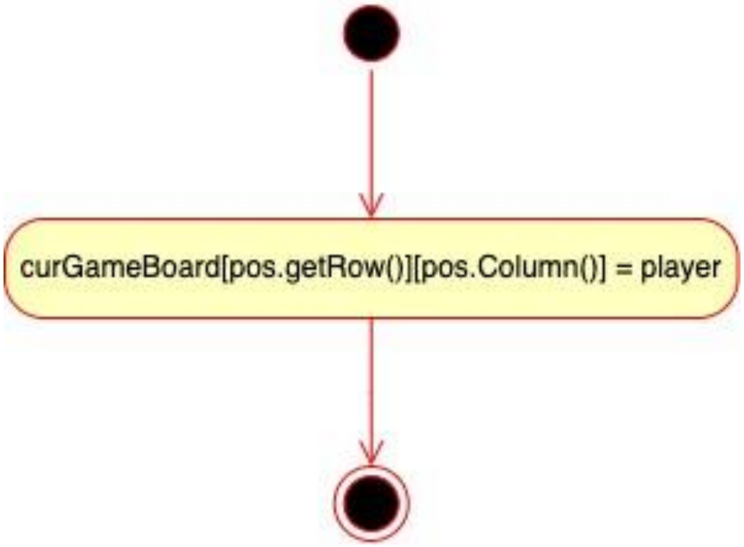


```
return numColumns;
```

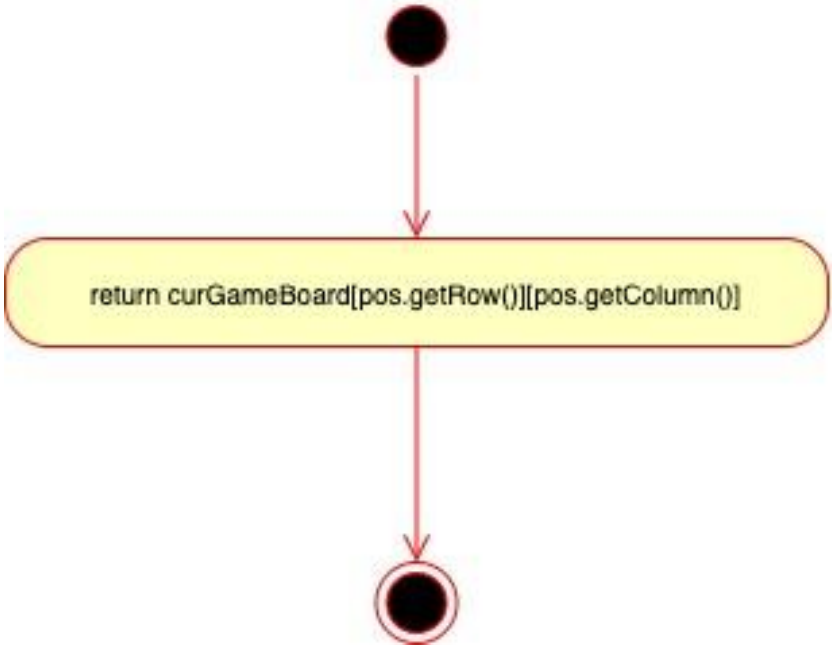
public int getNumToWin()



public void placeMarker(BoardPosition marker, char player)

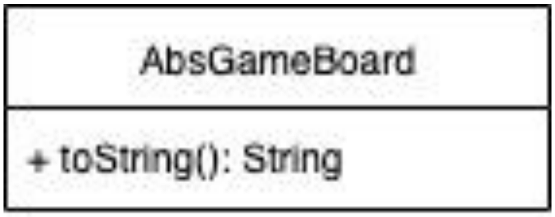


```
public char whatsAtPos(BoardPosition pos)
```



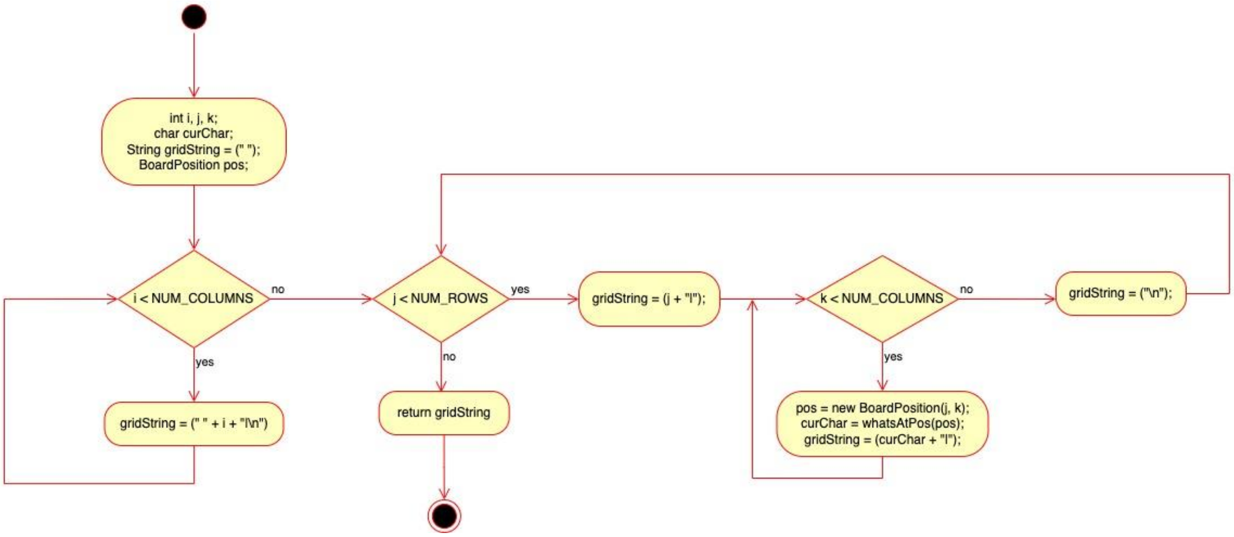
Class 4: AbsGameBoard

Class Diagram:



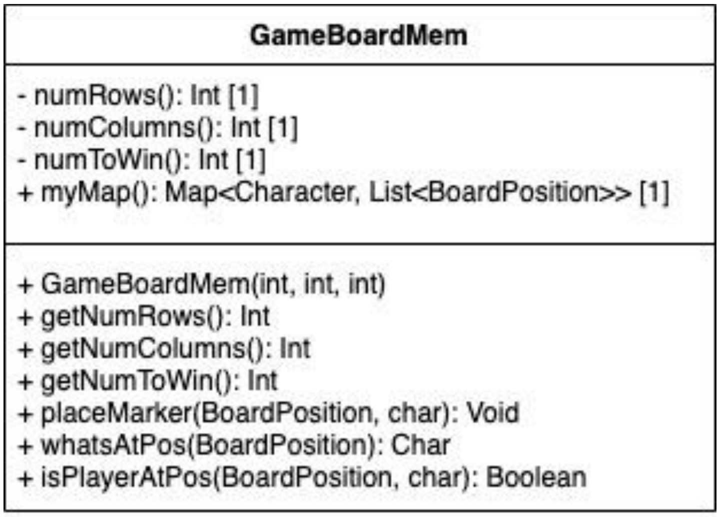
Activity Diagrams:

```
@Override
public string toString()
```



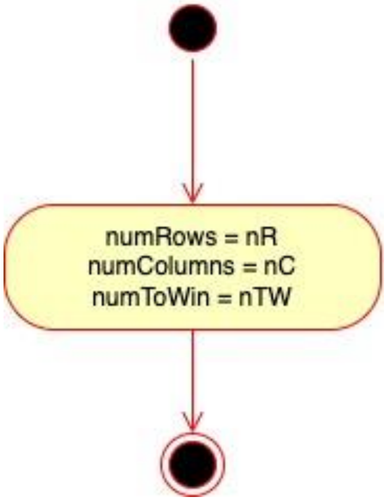
Class 5: GameBoardMem

Class Diagram:

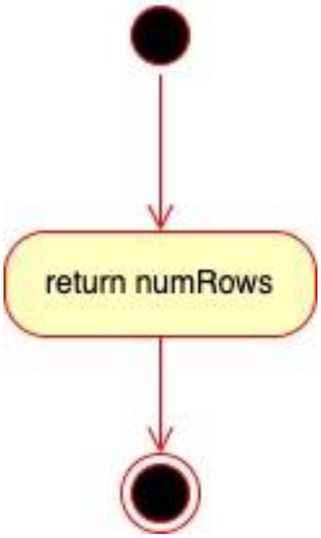

GameBoardMem
- numRows(): Int [1]
- numColumns(): Int [1]
- numToWin(): Int [1]
+ myMap(): Map<Character, List<BoardPosition>> [1]
+ GameBoardMem(int, int, int)
+ getNumRows(): Int
+ getNumColumns(): Int
+ getNumToWin(): Int
+ placeMarker(BoardPosition, char): Void
+ whatsAtPos(BoardPosition): Char
+ isPlayerAtPos(BoardPosition, char): Boolean

Activity Diagrams:

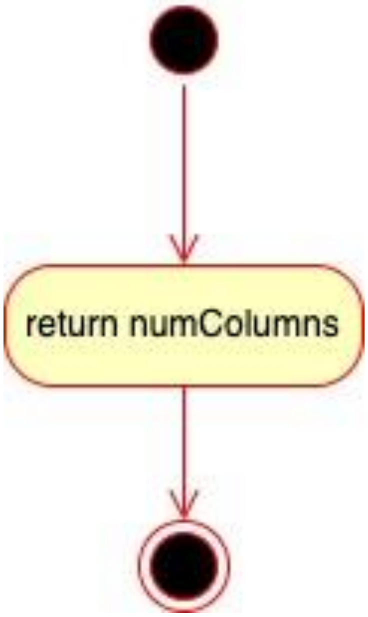
```
Public GameBoardMem(int nR, int nC, int nTW);
```



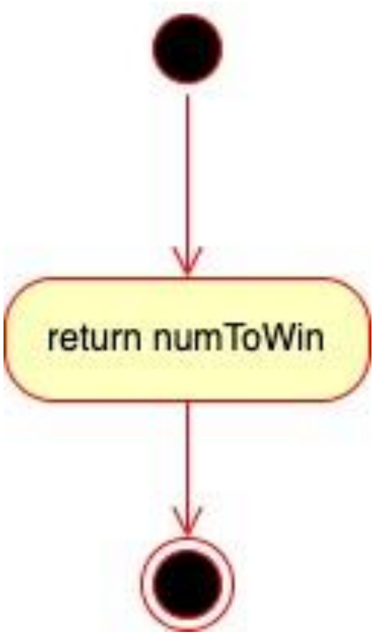
```
Public int getNumRows();
```



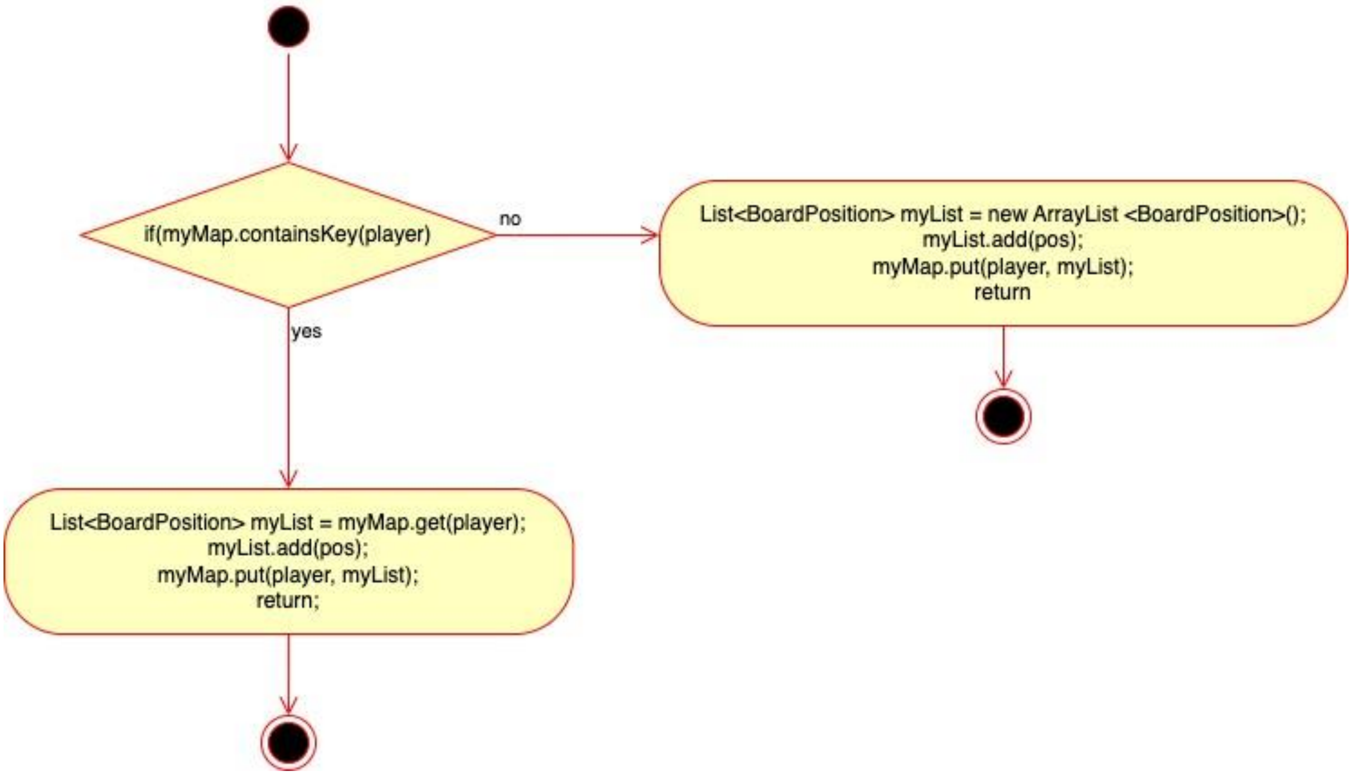
```
Public int getNumColumns();
```



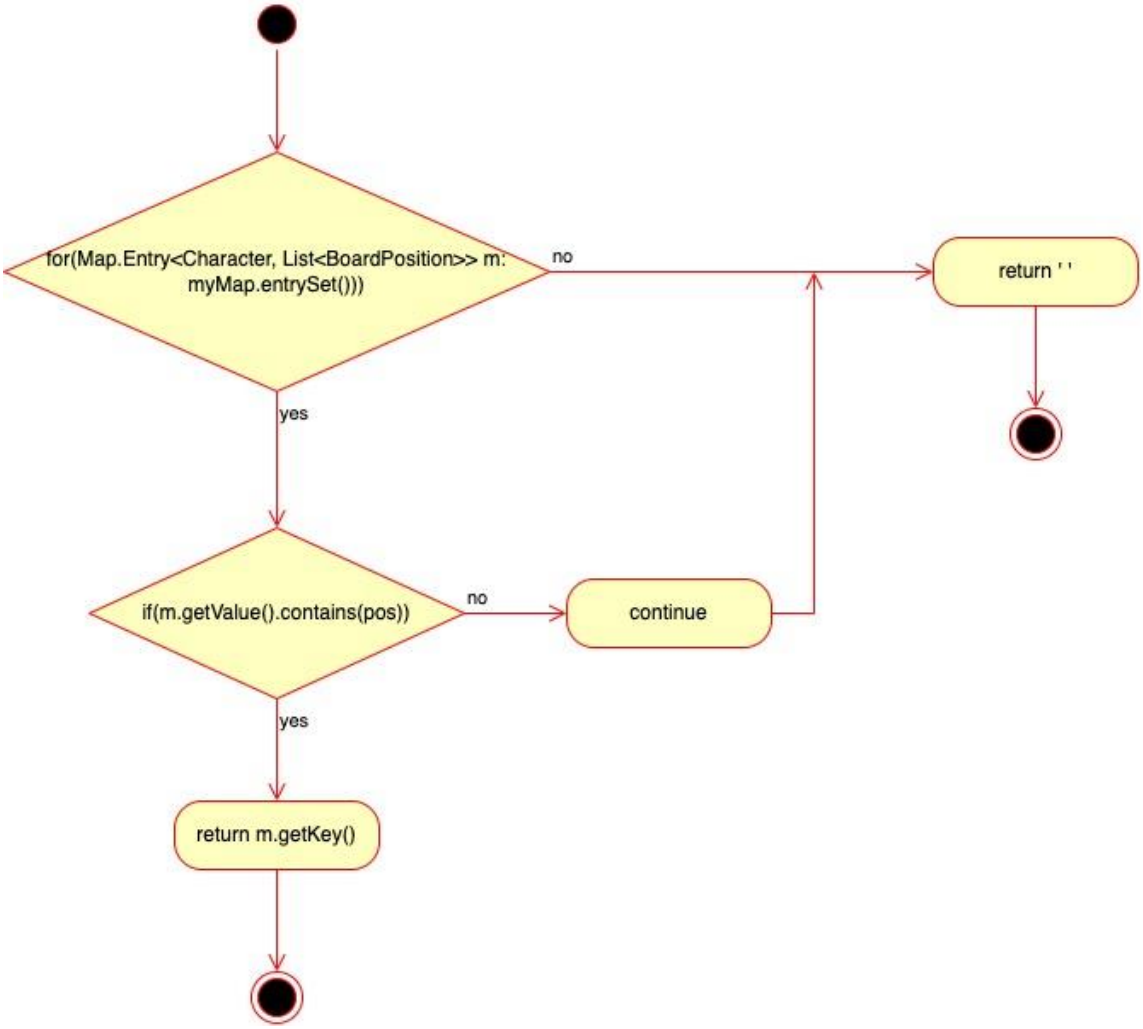
```
Public int getNumToWin();
```



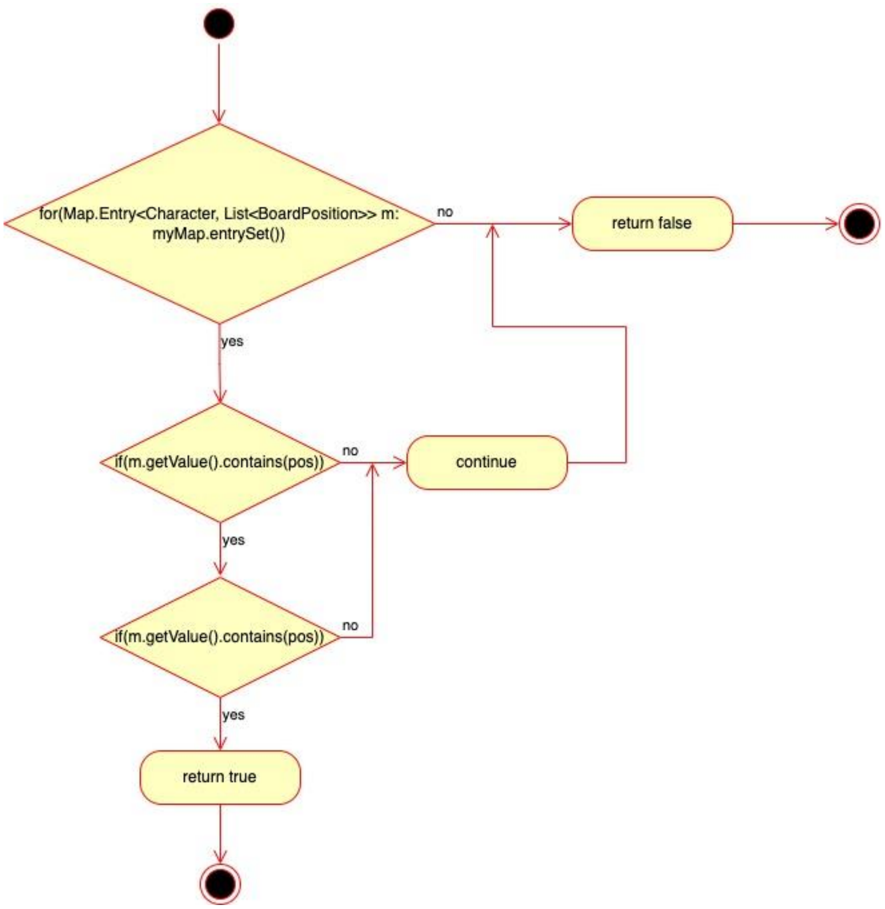
```
Public char PlaceMarker(BoardPosition pos);
```


if(myMap.containsKey(player)
no
List<BoardPosition> myList = new ArrayList <BoardPosition>();
myList.add(pos);
myMap.put(player, myList);
return
yes
List<BoardPosition> myList = myMap.get(player);
myList.add(pos);
myMap.put(player, myList);
return;

```
Public char whatsAtPos(BoardPosition pos);
```

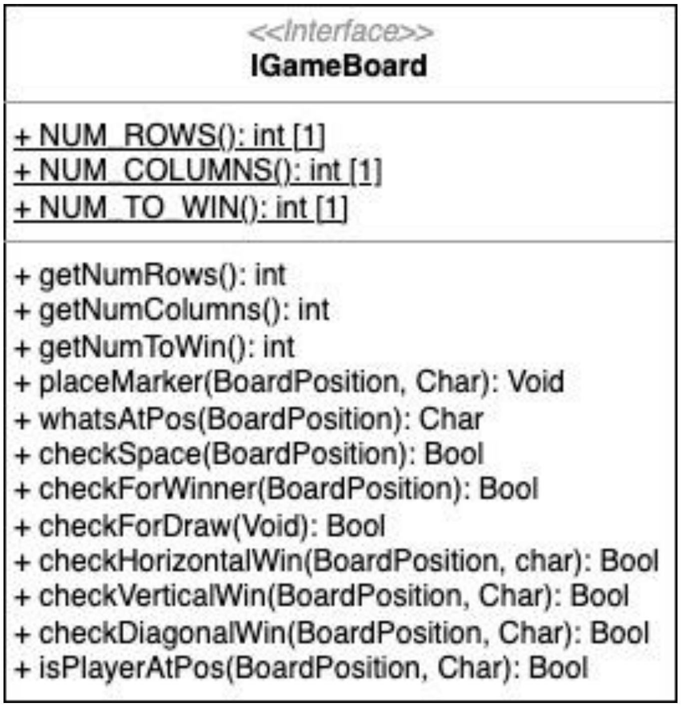

for(Map.Entry<Character, List<BoardPosition>> m: myMap.entrySet()))
no
return ' '
yes
if(m.getValue().contains(pos))
no
continue
yes
return m.getKey()

```
Public boolean isPlayerAtPos(BoardPosition pos, char player);
```


for(Map.Entry<Character, List<BoardPosition>> m: myMap.entrySet())
no
return false
yes
if(m.getValue().contains(pos))
no
continue
yes
if(m.getValue().contains(pos))
no
yes
return true

Interface 1: IGameBoard

Class Diagram:



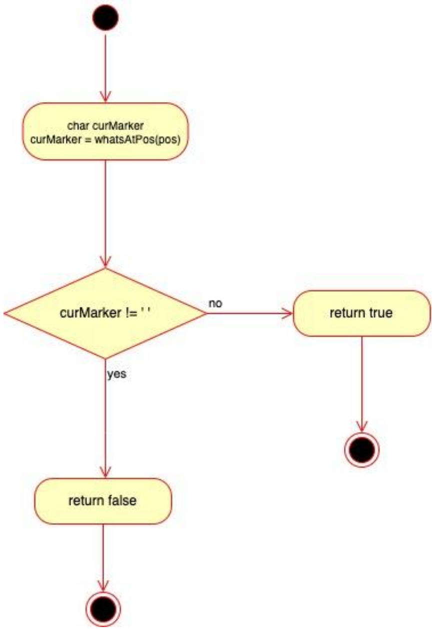
<<Interface>>
IGameBoard

+ NUM_ROWS(): int [1]
+ NUM_COLUMNS(): int [1]
+ NUM_TO_WIN(): int [1]

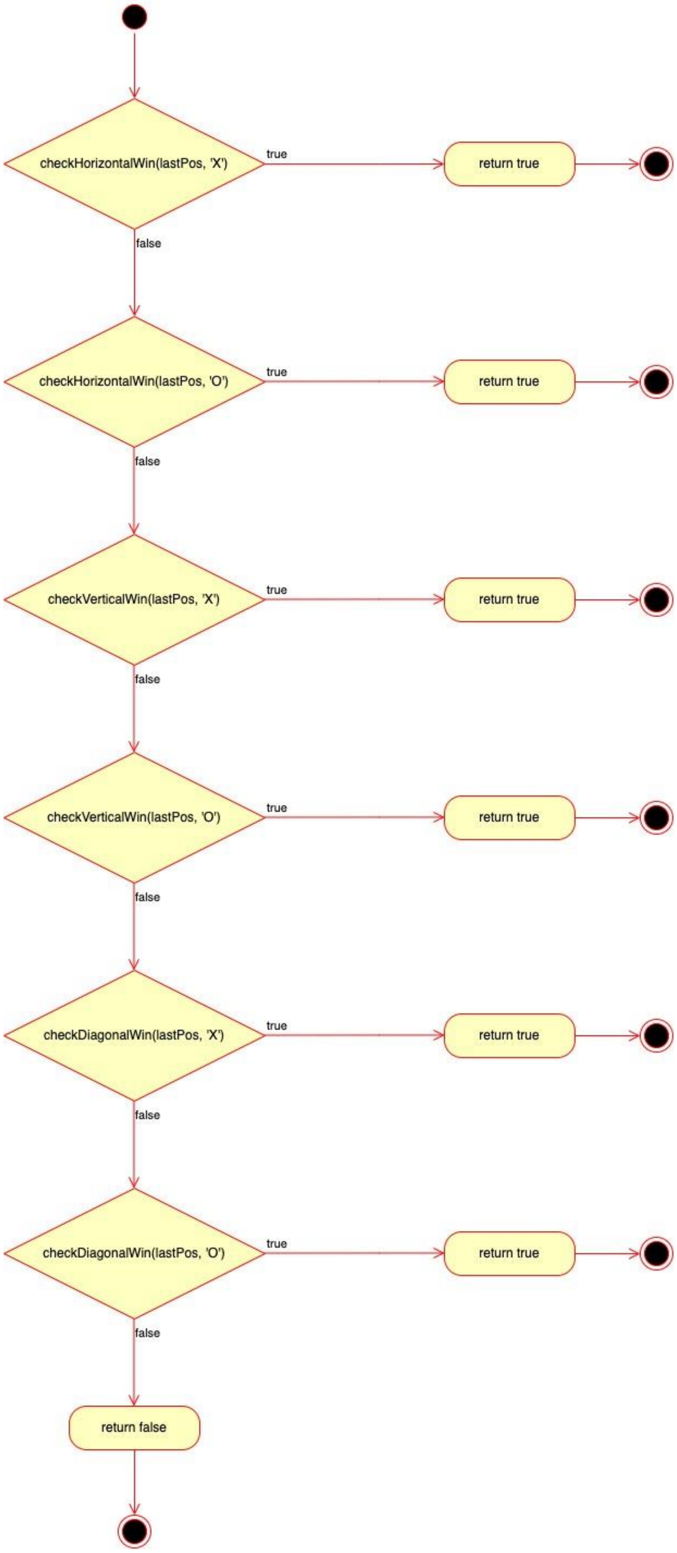
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ placeMarker(BoardPosition, Char): Void
+ whatsAtPos(BoardPosition): Char
+ checkSpace(BoardPosition): Bool
+ checkForWinner(BoardPosition): Bool
+ checkForDraw(Void): Bool
+ checkHorizontalWin(BoardPosition, char): Bool
+ checkVerticalWin(BoardPosition, Char): Bool
+ checkDiagonalWin(BoardPosition, Char): Bool
+ isPlayerAtPos(BoardPosition, Char): Bool

Activity Diagrams:

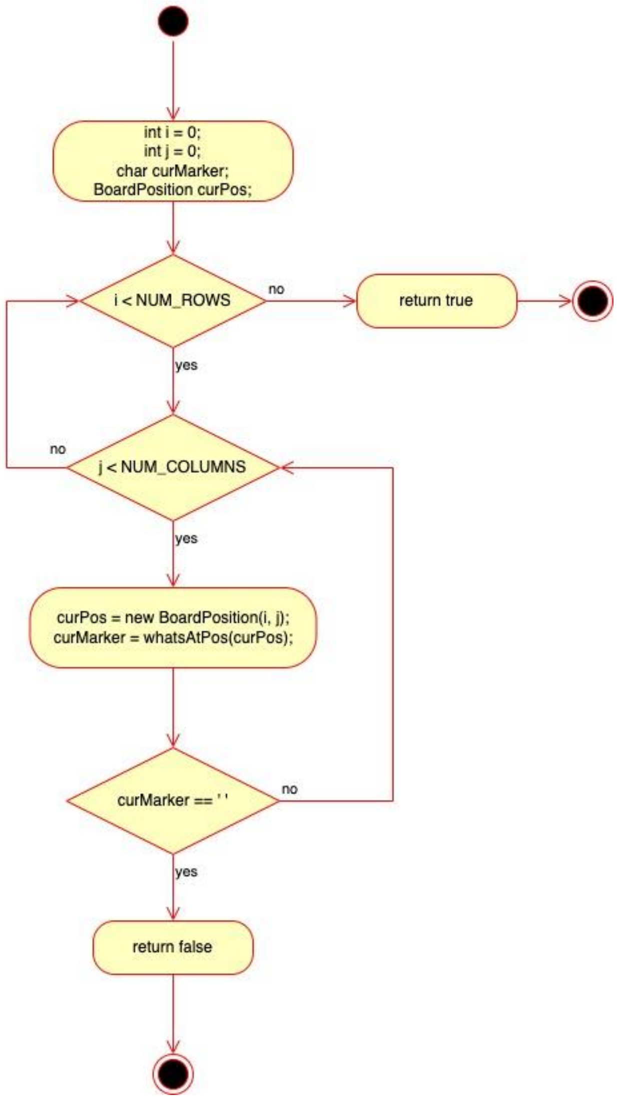
`default boolean checkSpace(BoardPosition pos)`



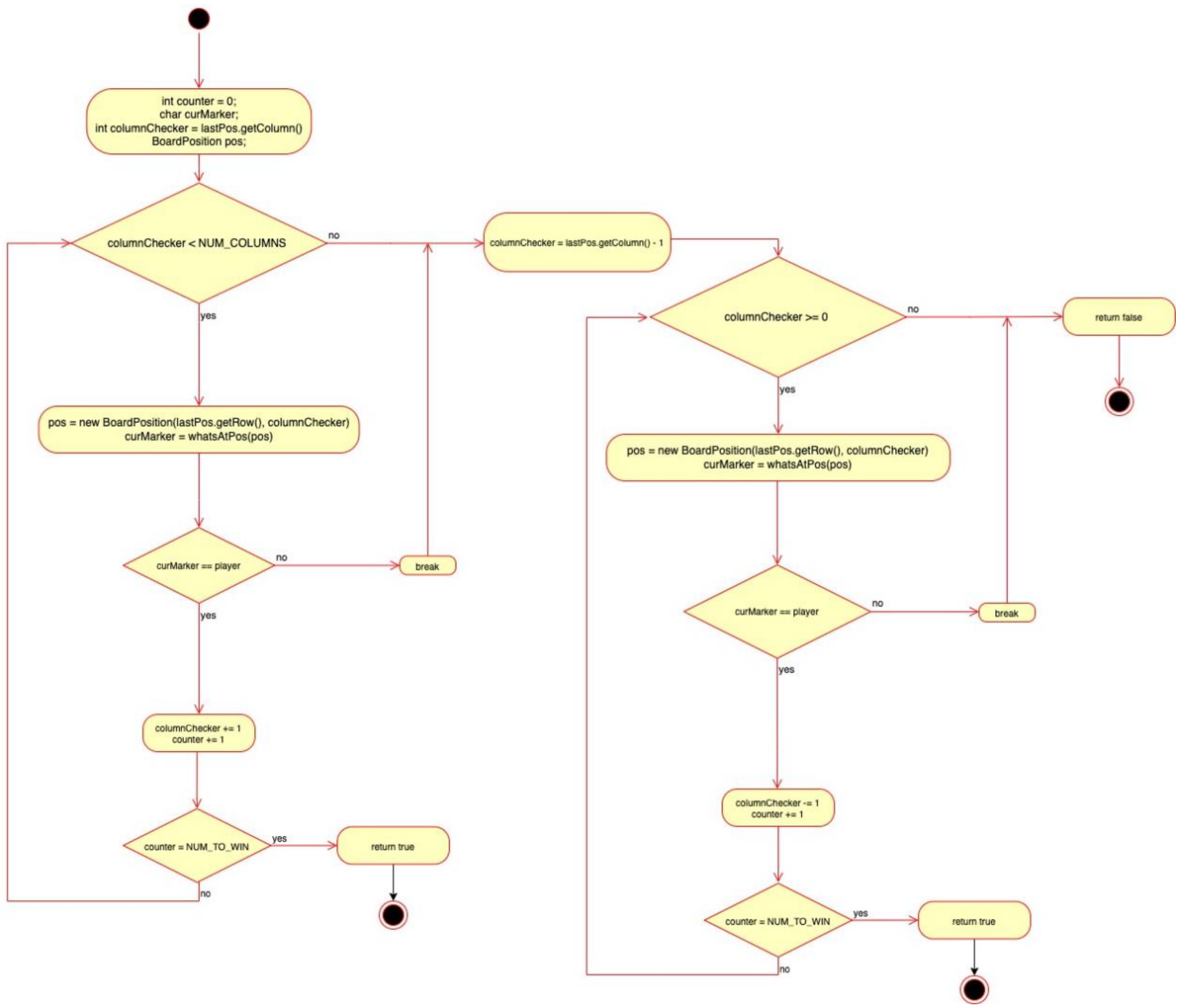
`default boolean checkForWinner(BoardPosition lastPos)`


checkHorizontalWin(lastPos, 'X')
true
return true
false
checkHorizontalWin(lastPos, 'O')
true
return true
false
checkVerticalWin(lastPos, 'X')
true
return true
false
checkVerticalWin(lastPos, 'O')
true
return true
false
checkDiagonalWin(lastPos, 'X')
true
return true
false
checkDiagonalWin(lastPos, 'O')
true
return true
false
return false

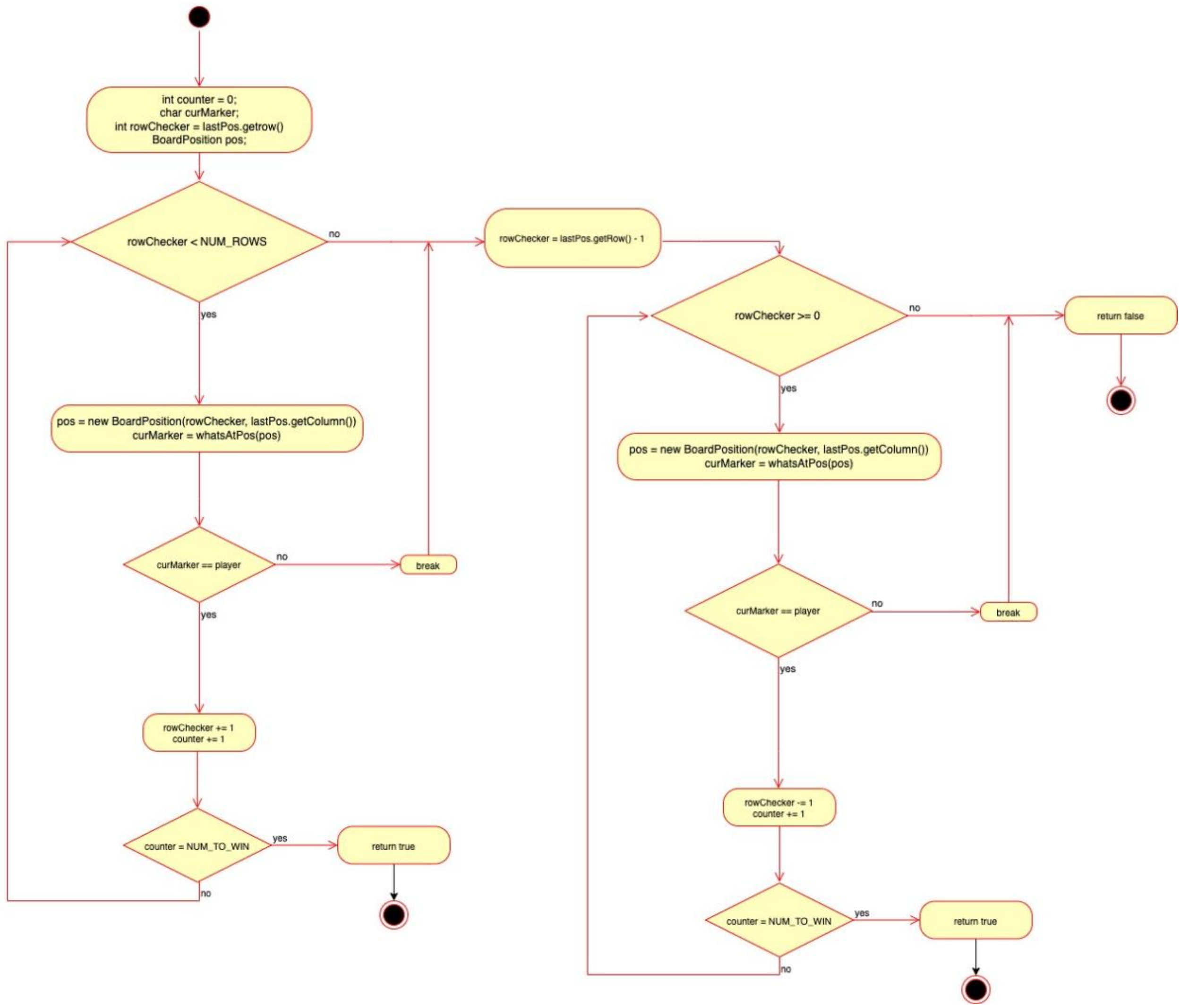
```
default boolean checkForDraw()
```


int i = 0;
int j = 0;
char curMarker;
BoardPosition curPos;
i < NUM_ROWS
no
return true
yes
no
j < NUM_COLUMNS
yes
curPos = new BoardPosition(i, j);
curMarker = whatsAtPos(curPos);
curMarker == ' '
no
yes
return false

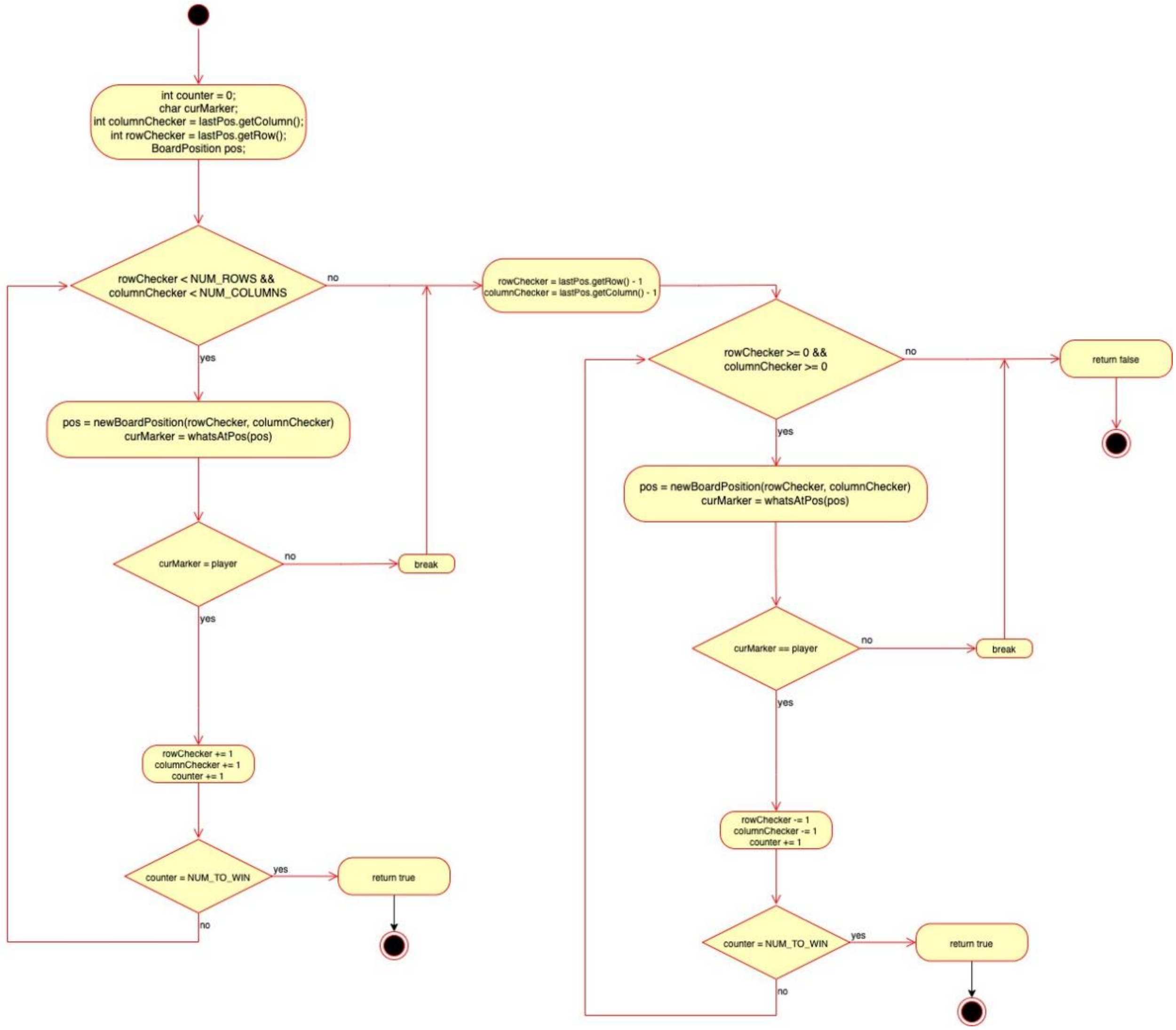
```
default boolean checkHorizontalWin(BoardPosition lastPos, char
player)
```



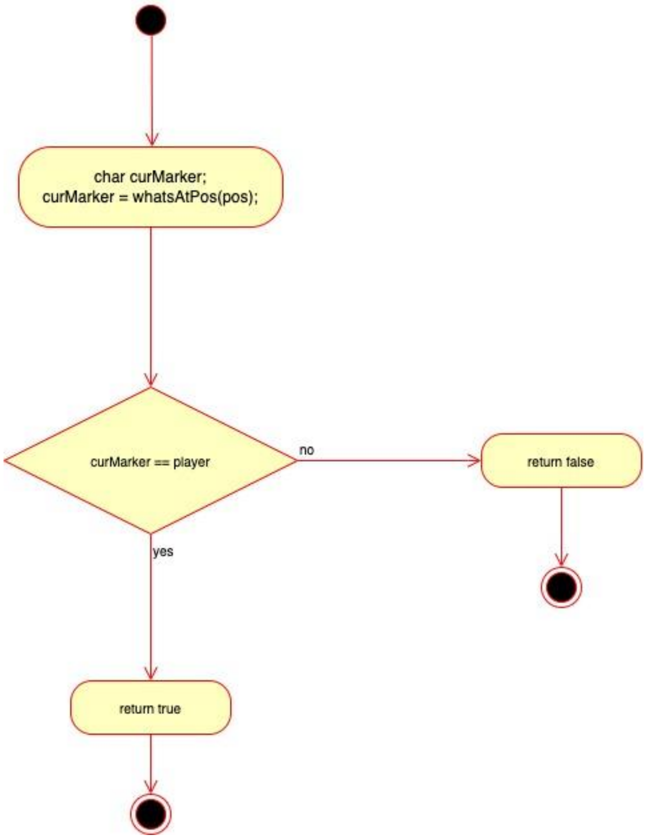
```
default boolean checkVerticalWin(BoardPosition lastPos, char
player)
```



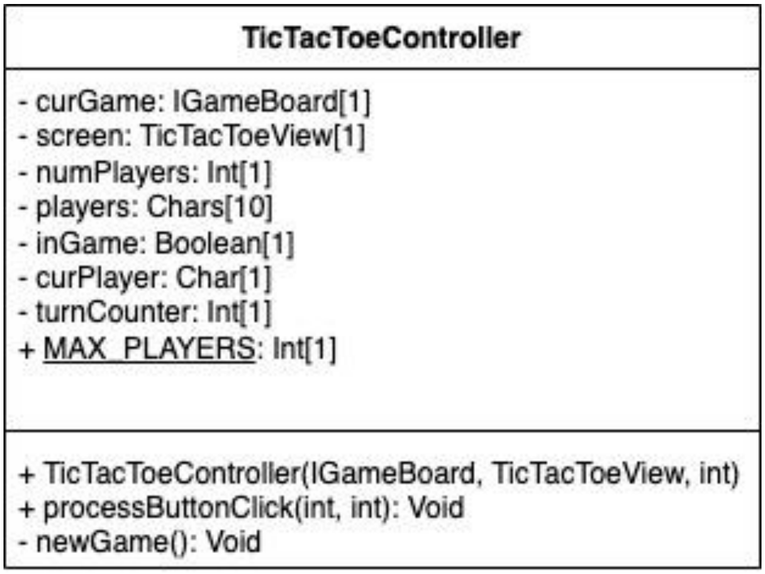
```
default boolean checkDiagonalWin(BoardPosition lastPos, char
player)
```



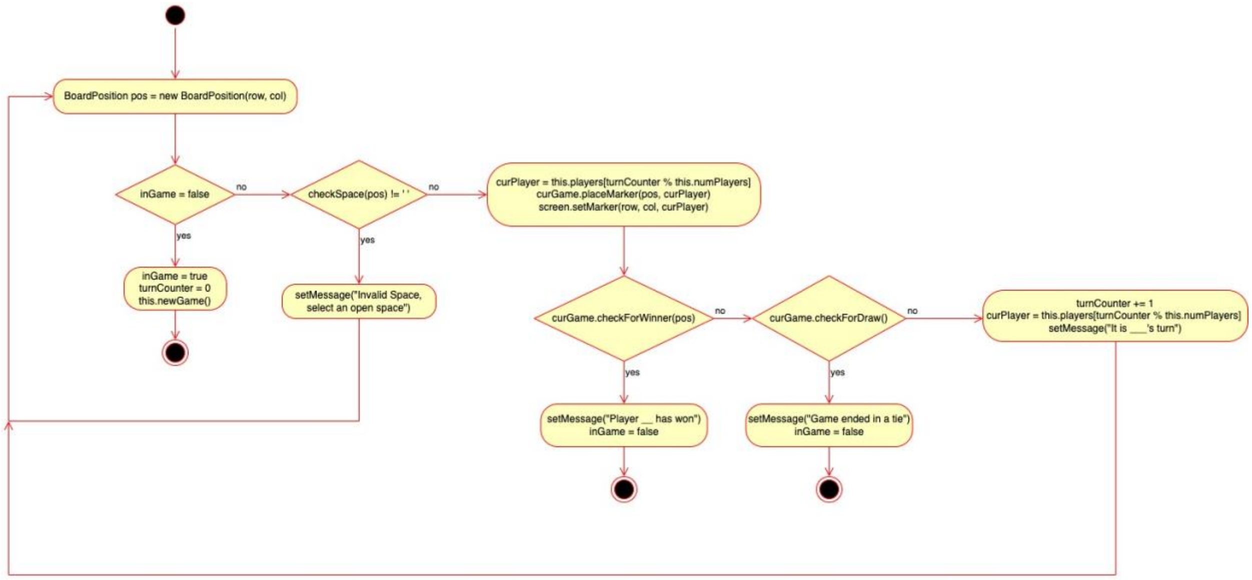
```
default boolean isPlayerAtPos(BoardPosition pos, char player)
```



Class Diagram:



public void processButtonClick(int row, int col);


BoardPosition pos = new BoardPosition(row, col)
inGame = false
no
yes
inGame = true
turnCounter = 0
this.newGame()
checkSpace(pos) != ' '
no
yes
setMessage("Invalid Space, select an open space")
curPlayer = this.players[turnCounter % this.numPlayers]
curGame.placeMarker(pos, curPlayer)
screen.setMarker(row, col, curPlayer)
curGame.checkForWinner(pos)
no
yes
setMessage("Player __ has won")
inGame = false
curGame.checkForDraw()
no
yes
setMessage("Game ended in a tie")
inGame = false
turnCounter += 1
curPlayer = this.players[turnCounter % this.numPlayers]
setMessage("It is ___'s turn")

Test Cases:

Constructor:
public GameBoard(int nR, int nC, int nTW){
- Create 3 distinct test cases for the constructor

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

State: No GameBoard created yet.

numRows = 3</td>
<td><b>Output:</b>

<b>State:</b>
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th></tr>
<tr><th>0</th><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td><b>Reason:</b>
Check if a board of the minimum size will be made

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

<table>
<tr>
<td>numColumns = 3
numToWin = 3</td>
<td>
<table>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td>values</td>
</tr>
</table>

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

State: No GameBoard created yet.

numRows = 100
numColumns = 100
numToWin = 25</td>
<td><b>Output:</b>

<b>State:</b>
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td><b>Reason:</b>
Check if a board of the maximum size will be made

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

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

State: No GameBoard created yet.

numRows = 30
numColumns = 40
numToWin = 10</td>
<td><b>Output:</b>

<b>State:</b>
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td><b>Reason:</b>
Check if a board of the normal size will be made

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

checkSpace

```
default boolean checkSpace(BoardPosition pos){
```

- Create 3 distinct test cases for checkSpace

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td>X</td><td>X</td></tr>
<tr><td><b>2</b></td><td></td><td>O</td><td></td></tr>
</table>

pos.getRow(1)
pos.getColumn(0)
checkSpace(pos){</td>
<td><b>Output:</b>

checkSpace = false

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td>X</td><td>X</td></tr>
<tr><td><b>2</b></td><td></td><td>O</td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkSpace when there is just a full space on an min board

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

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>49</th><th>99</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td>X</td></tr>
<tr><td><b>49</b></td><td></td><td>O</td><td></td></tr>
<tr><td><b>99</b></td><td>X</td><td></td><td></td></tr>
</table>

pos.getRow(1)
pos.getColumn(0)
checkSpace(pos){</td>
<td><b>Output:</b>

checkSpace = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>49</th><th>99</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td>X</td></tr>
<tr><td><b>49</b></td><td></td><td>O</td><td></td></tr>
<tr><td><b>99</b></td><td>X</td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkSpace when there is just a full space on a max board

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

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>5</th><th>17</th><th>32</th></tr>
<tr><td><b>10</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>13</b></td><td></td><td>X</td><td></td></tr>
<tr><td><b>27</b></td><td></td><td></td><td>O</td></tr>
</table>

pos.getRow(1)
pos.getColumn(0)
checkSpace(pos){</td>
<td><b>Output:</b>

checkSpace = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>5</th><th>17</th><th>32</th></tr>
<tr><td><b>10</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>13</b></td><td></td><td>X</td><td></td></tr>
<tr><td><b>27</b></td><td></td><td></td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkSpace when there is just a full space on a reg board

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

checkHorizontalWin

default boolean checkHorizontalWin(BoardPosition lastPos, char player)

- Create 4 distinct test cases

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>X</td><td>X</td></tr>
<tr><td><b>1</b></td><td>O</td><td></td><td>O</td></tr>
<tr><td><b>2</b></td><td></td><td></td><td>O</td></tr>
</table>
pos.getRow(0)
pos.getColumn(2)
marker = ‘X’
checkHorizontalWin(pos, marker);
</td>
<td>
<b>Output:</b>

checkHorizontalWin = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>X</td><td>X</td></tr>
<tr><td><b>1</b></td><td>O</td><td></td><td>O</td></tr>
<tr><td><b>2</b></td><td></td><td></td><td>O</td></tr>
</table>
</td>
<td>
<b>Reason:</b>
Check to see what is returned by checkHorizontalWin when there is a win on the min board

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

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>X</td><td></td></tr>
<tr><td><b>1</b></td><td>O</td><td></td><td>O</td></tr>
<tr><td><b>2</b></td><td></td><td>X</td><td>O</td></tr>
</table>
pos.getRow(0)
pos.getColumn(0)
marker = ‘X’
checkHorizontalWin(pos, marker);
</td>
<td>
<b>Output:</b>

checkHorizontalWin = false

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>X</td><td>X</td></tr>
<tr><td><b>1</b></td><td>O</td><td></td><td>O</td></tr>
<tr><td><b>2</b></td><td></td><td></td><td>O</td></tr>
</table>
</td>
<td>
<b>Reason:</b>
Check to see what is returned by checkHorizontalWin when there is a loss on the min board

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

Input:

State: (number to win = 25)

	0	25	42	72
0	X…	…X	O	O
10			O	O
15				O

```
pos.getRow(0)
pos.getColumn(24)
marker = ‘X’
checkHorizontalWin(pos,
marker);
```

Output:

checkHorizontalWin = true

State: state of board remains unchanged:

	0	25	42	72
0	X…	…X	O	O
10			O	O
15				O

Reason:
Check to see what is returned by checkHorizontalWin when there is a win on the max board

Function Name:
testCheckHorizontalWin_max_board_win

Input:

State: (number to win = 25)

	0	3	4	5	25	72
0	X…	…X	O	X…	…X	O
10			O			O
15						O

```
pos.getRow(0)
pos.getColumn(24)
marker = ‘X’
checkHorizontalWin(pos,
marker);
```

Output:

checkHorizontalWin = false

State: state of board remains unchanged:

	0	3	4	5	25	72
0	X…	…X	O	X…	…X	O
10			O			O
15						O

Reason:
Check to see what is returned by checkHorizontalWin when there is a loss on the max board

Function Name:
testCheck HorizontalWin_max_board_win

checkVerticalWin
default boolean checkVerticalWin(BoardPosition lastPos, char player)
- Create 4 distinct test cases

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>O</td><td></td></tr>
<tr><td><b>1</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>2</b></td><td>X</td><td>O</td><td>O</td></tr>
</table>

pos.getRow(2)
pos.getColumn(o)
marker = ‘X’
checkHorizontalWin(pos, marker);</td>
<td><b>Output:</b>

checkHorizontalWin = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>O</td><td></td></tr>
<tr><td><b>1</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>2</b></td><td>X</td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkVerticalWin when there is a win on the min board

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

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>O</td><td></td></tr>
<tr><td><b>1</b></td><td>X</td><td></td><td>X</td></tr>
<tr><td><b>2</b></td><td></td><td>O</td><td>O</td></tr>
</table>

pos.getRow(0)
pos.getColumn(0)
marker = ‘X’
checkHorizontalWin(pos, marker);</td>
<td><b>Output:</b>

checkHorizontalWin = false

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>O</td><td></td></tr>
<tr><td><b>1</b></td><td>X</td><td></td><td>X</td></tr>
<tr><td><b>2</b></td><td></td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkVerticalWin when there is no win on the min board

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

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

<b>State:</b> (number to win = 25)
<table>
<tr><th></th><th>0</th><th>10</th><th>15</th></tr>
<tr><td><b>0</b></td><td>X…</td><td></td><td></td></tr>
<tr><td><b>24</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>42</b></td><td>O…</td><td>…O</td><td></td></tr>
<tr><td><b>72</b></td><td>O…</td><td>…O…</td><td>…O</td></tr>
</table>

pos.getRow(24)
pos.getColumn(0)
marker = ‘X’
checkHorizontalWin(pos, marker);</td>
<td><b>Output:</b>

checkHorizontalWin = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>10</th><th>15</th></tr>
<tr><td><b>0</b></td><td>X…</td><td></td><td></td></tr>
<tr><td><b>24</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>42</b></td><td>O…</td><td>…O</td><td></td></tr>
<tr><td><b>72</b></td><td>O…</td><td>…O…</td><td>…O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkVerticalWin when there is a win on the max board

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

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

<b>State:</b> (number to win = 25)
<table>
<tr><th></th><th>0</th><th>10</th><th>15</th></tr>
<tr><td><b>0</b></td><td>X…</td><td></td><td></td></tr>
<tr><td><b>3</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>4</b></td><td>O…</td><td>…O</td><td></td></tr>
<tr><td><b>5</b></td><td>X…</td><td></td><td></td></tr>
<tr><td><b>24</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>72</b></td><td>O…</td><td>…O…</td><td>…O</td></tr>
</table>

pos.getRow(24)
pos.getColumn(0)
marker = ‘X’
checkHorizontalWin(pos, marker);</td>
<td><b>Output:</b>

checkHorizontalWin = false

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>10</th><th>15</th></tr>
<tr><td><b>0</b></td><td>X…</td><td></td><td></td></tr>
<tr><td><b>3</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>4</b></td><td>O…</td><td>…O</td><td></td></tr>
<tr><td><b>5</b></td><td>X…</td><td></td><td></td></tr>
<tr><td><b>24</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>72</b></td><td>O…</td><td>…O…</td><td>…O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkVerticalWin when there is a loss on the max board

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

checkDiagonalWin
default boolean checkDiagonalWin(BoardPosition lastPos, char player)
- Create 7 distinct test cases
- Note: the different diagonals are distinct

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>O</td><td>O</td><td>X</td></tr>
<tr><td><b>1</b></td><td></td><td>X</td><td></td></tr>
<tr><td><b>2</b></td><td>X</td><td>O</td><td></td></tr>
</table>

pos.getRow(0)
pos.getColumn(2)
marker = ‘X’
checkHorizontalWin(pos, marker);</td>
<td><b>Output:</b>

checkHorizontalWin = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>O</td><td>O</td><td>X</td></tr>
<tr><td><b>1</b></td><td></td><td>X</td><td></td></tr>
<tr><td><b>2</b></td><td>X</td><td>O</td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkDiagonalWin when there is a win on the min board

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

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>O</td><td>O</td></tr>
<tr><td><b>1</b></td><td></td><td>X</td><td>O</td></tr>
<tr><td><b>2</b></td><td></td><td></td><td>X</td></tr>
</table>

pos.getRow(2)
pos.getColumn(2)
marker = ‘X’
checkHorizontalWin(pos, marker);</td>
<td><b>Output:</b>

checkHorizontalWin = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>X</td><td>O</td><td>O</td></tr>
<tr><td><b>1</b></td><td></td><td>X</td><td>O</td></tr>
<tr><td><b>2</b></td><td></td><td></td><td>X</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkDiagonalWin when there is a win on the min board

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

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>O</td><td>O</td><td>X</td></tr>
<tr><td><b>1</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>2</b></td><td>X</td><td>O</td><td></td></tr>
</table>

pos.getRow(0)
pos.getColumn(2)
marker = ‘X’
checkHorizontalWin(pos, marker);</td>
<td><b>Output:</b>

checkHorizontalWin = false

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td><b>0</b></td><td>O</td><td>O</td><td>X</td></tr>
<tr><td><b>1</b></td><td>X</td><td></td><td></td></tr>
<tr><td><b>2</b></td><td>X</td><td>O</td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkDiagonalWin when there is a no win on the min board

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

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

<b>State:</b> (number to win = 25)
<table>
<tr><th></th><th>0</th><th>10</th><th>15</th><th>25</th></tr>
<tr><td><b>0</b></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td><b>10</b></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td><b>15</b></td><td></td><td></td><td>X</td><td></td></tr>
<tr><td><b>25</b></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td><b>26</b></td><td>O…</td><td>…O</td><td></td><td></td></tr>
<tr><td><b>72</b></td><td>O…</td><td>…O…</td><td>…O</td><td></td></tr>
</table>

pos.getRow(24)
pos.getColumn(24)
marker = ‘X’
checkDiagonalWin(pos, marker);</td>
<td><b>Output:</b>

checkDiagonalWin = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>10</th><th>15</th><th>25</th></tr>
<tr><td><b>0</b></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td><b>10</b></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td><b>15</b></td><td></td><td></td><td>X</td><td></td></tr>
<tr><td><b>25</b></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td><b>26</b></td><td>O…</td><td>…O</td><td></td><td></td></tr>
<tr><td><b>72</b></td><td>O…</td><td>…O…</td><td>…O</td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkDiagonalWin when there is a win on the max board

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

Input:

State: (number to win = 25)

	0	10	15	25
15				
26	O…	…O		
72	O…	…O…	…O	
75				X
85			X	
90		X		
99	X			

```
pos.getRow(99)
pos.getColumn(0)
marker = 'X'
checkDiagonalWin(pos,
marker);
```

Output:

checkDiagonalWin = true

State: state of board remains unchanged:

	0	10	15	25
15				
26	O…	…O		
72	O…	…O…	…O	
75				X
85			X	
90		X		
99	X			

Reason:
Check to see what is returned by checkDiagonalWin when there is a win on the max board

Function Name:
testCheckDiagonalWin_max_board_win_SWtoNE

Input:

State: (number to win = 25)

	0	10	15	25
0	X			
10		X		
15			X	
25				X
26	O…	…O		
28		X		
37	X			
72	O…	…O…	…O	

```
pos.getRow(0)
pos.getColumn(0)
marker = 'X'
checkDiagonalWin(pos,
marker);
```

Output:

checkDiagonalWin = false

State: state of board remains unchanged:

	0	10	15	25
0	X			
10		X		
15			X	
25				X
26	O…	…O		
28		X		
37	X			
72	O…	…O…	…O	

Reason:
Check to see what is returned by checkDiagonalWin when there is a win on the max board

Function Name:
testCheckDiagonalWin_max_board_loss

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

<b>State:</b> (number to win = 10)
<table>
<tr><th></th><th>0</th><th>8</th><th>10</th><th>15</th></tr>
<tr><td><b>0</b></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td><b>10</b></td><td></td><td></td><td>X</td><td></td></tr>
<tr><td><b>12</b></td><td>O…</td><td>…O</td><td></td><td></td></tr>
<tr><td><b>26</b></td><td>O…</td><td>…O</td><td></td><td></td></tr>
<tr><td><b>28</b></td><td></td><td></td><td>X</td><td></td></tr>
</table>
pos.getRow(0)
pos.getColumn(0)
marker = ‘X’
checkDiagonalWin(pos, marker);</td>
<td><b>Output:</b>

checkDiagonalWin = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>8</th><th>10</th><th>15</th></tr>
<tr><td><b>0</b></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td><b>10</b></td><td></td><td></td><td>X</td><td></td></tr>
<tr><td><b>12</b></td><td>O…</td><td>…O</td><td></td><td></td></tr>
<tr><td><b>26</b></td><td>O…</td><td>…O</td><td></td><td></td></tr>
<tr><td><b>28</b></td><td></td><td></td><td>X</td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkDiagonalWin when there is a win on the reg board

<b>Function Name:</b>
testCheckDiagonal Win_reg_board_win</td>
</tr>
</table>

checkForDraw
default boolean checkForDraw()
- Create 4 distinct test cases

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

<b>State:</b> (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
</table>
checkForDraw();</td>
<td><b>Output:</b>

checkForDraw = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkForDraw when it is a min board draw

<b>Function Name:</b>
testCheckForDraw_ min_board_true</td>
</tr>
</table>

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

<b>State:</b> (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>100</td></tr>
<tr><td>0</td><td></td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>100</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
</table>
checkForDraw();</td>
<td><b>Output:</b>

checkForDraw = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>100</td></tr>
<tr><td>0</td><td></td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>100</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkForDraw when it is a max board draw

<b>Function Name:</b>
testCheckForDraw_ max_board_true</td>
</tr>
</table>

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

<b>State:</b> (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>100</td></tr>
<tr><td>0</td><td>0</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td></tr>
<tr><td>100</td><td></td><td></td><td></td><td></td></tr>
</table>
checkForDraw();</td>
<td><b>Output:</b>

checkForDraw = false

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>100</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td></tr>
<tr><td>100</td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by checkForDraw when it is a max board no draw

<b>Function Name:</b>
testCheckForDraw_ max_board_false</td>
</tr>
</table>

whatsAtPos
public char whatsAtPos(BoardPosition pos);
- Create 5 distinct test cases

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

<b>State:</b> (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td>X</td><td>O</td><td>O</td></tr>
</table>
pos.getRow(0)
pos.getColumn(0)
whatsAtPos(pos);</td>
<td><b>Output:</b>

whatsAtPos = ‘X‘

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td>X</td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by whatsAtPos when a position is filled and min board

<b>Function Name:</b>
testWhatsAtPos_ min_board_ populated</td>
</tr>
</table>

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

<b>State:</b> (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td>X</td><td>O</td><td>O</td></tr>
</table>
pos.getRow(1)
pos.getColumn(1)
whatsAtPos(pos);</td>
<td><b>Output:</b>

whatsAtPos = ‘ ‘

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td>X</td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by whatsAtPos when a position is empty and min board

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

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

<b>State:</b> (number to win = 25)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>100</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td></tr>
<tr><td>100</td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow(2)
pos.getColumn(1)
whatsAtPos(pos);</td>
<td><b>Output:</b>

whatsAtPos = ‘X’

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>100</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td></tr>
<tr><td>100</td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by whatsAtPos when a position is full and max board

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

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

<b>State:</b> (number to win = 25)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>100</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td></tr>
<tr><td>100</td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow(0)
pos.getColumn(0)
whatsAtPos(pos);</td>
<td><b>Output:</b>

whatsAtPos = ‘ ’

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>100</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td></tr>
<tr><td>100</td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by whatsAtPos when a position is empty and max board

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

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

<b>State:</b> (number to win = 10)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>40</th></tr>
<tr><td>0</td><td></td><td>O</td><td>X</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td></tr>
<tr><td>30</td><td></td><td></td><td>O</td><td>O</td></tr>
</table>
pos.getRow(0)
pos.getColumn(1)
whatsAtPos(pos);</td>
<td><b>Output:</b>

whatsAtPos = ‘O’

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>40</th></tr>
<tr><td>0</td><td></td><td>O</td><td>X</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td></tr>
<tr><td>30</td><td></td><td></td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by whatsAtPos when a position is populated and reg board

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

isPlayerAtPos
default boolean isPlayerAtPos(BoardPosition pos, char player)
- Create 5 distinct test cases

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>0</td><td></td><td></td><td></td><td>O</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td>O</td><td>O</td><td>X</td></tr>
</table>
pos.getRow(1)
pos.getColumn(1)
player = ‘X’
whatsAtPos(pos,marker)</td>
<td><b>Output:</b>

isPlayerAtPos = false

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>0</td><td></td><td></td><td></td><td>O</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td>O</td><td>O</td><td>X</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by isPlayerAtPos when a position with no marker in it and no markers around it is checked. isPlayerAtPos(pos, marker) should return false

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

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

<b>State:</b> (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>O</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>2</td><td>O</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td></td><td>O</td><td>O</td><td>O</td></tr>
</table>
pos.getRow(2)
pos.getColumn(1)
player = ‘X’
isPlayerAtPos(pos, marker);</td>
<td><b>Output:</b>

isPlayerAtPos = false

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>O</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>O</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>2</td><td>O</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td></td><td>O</td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by isPlayerAtPos when a position with a marker in it and only opposing markers around it. isPlayerAtPos(pos, marker) should return false

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

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

<b>State:</b> (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow(3)
pos.getColumn(0)
marker= ‘X’
isPlayerAtPos(pos);</td>
<td><b>Output:</b>

isPlayerAtPos = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by isPlayerAtPos when a position with a marker in it and it is on the corner of the board. Making sure there are no out of bounds issues. isPlayerAtPos(pos, marker) should return true

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

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

<b>State:</b> (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td>X</td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td>X</td></tr>
<tr><td>3</td><td>O</td><td></td><td></td><td></td></tr>
</table>
pos.getRow(1)
pos.getColumn(2)
marker= O
isPlayerAtPos(pos, marker);</td>
<td><b>Output:</b>

isPlayerAtPos = true

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td>X</td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td>X</td></tr>
<tr><td>3</td><td>O</td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by whatsAtPos when a position with a marker in it and there is a win on the board. isPlayerAtPos(pos, marker) should return true

<b>Function Name:</b>
testIsPlayerAtPos_ with_full_pos_ and_diag_win</td>
</tr>
</table>

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

<b>State:</b> (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
</table>
pos.getRow(0)
pos.getColumn(0)
marker= ‘X’
isPlayerAtPos(pos, marker);</td>
<td><b>Output:</b>

isPlayerAtPos = false

<b>State:</b> state of board remains unchanged:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is returned by isPlayerAtPos when a position with an incorrect marker in it and there is a draw on the board. isPlayerAtPos(pos, marker) should return false

<b>Function Name:</b>
testIsPlayerAtPos_ with_full_pos_ and_draw</td>
</tr>
</table>

placeMarker
public void placeMarker(BoardPosition pos, char player);
- Create 5 distinct test cases

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

<b>State:</b> (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow(1)
pos.getColumn(1)
marker = ‘X’
placeMarker(pos, marker);</td>
<td><b>Output:</b>

<b>State:</b> state of board changes:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is how the state of the board changes when a new board with nothing on it get its first position filled

<b>Function Name:</b>
testPlaceMarker_ with_first_move</td>
</tr>
</table>

Input:

State: (number to win = 3)

	0	1	2	3
0				
1		O	X	
2			O	
3				

```
pos.getRow(0)
pos.getColumn(3)
marker = ‘X’
placeMarker(pos,
marker);
```

Output:

State: state of board changes:

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

Reason:
Check to see what is how the state of the board changes when a corner move is made. Checks to make sure that it is considered in bounds

Function Name:
testPlaceMarker_
with_corner_move

Input:

State: (number to win = 3)

	0	1	2	3
0				
1		O	X	
2			O	
3				

```
pos.getRow(1)
pos.getColumn(1)
marker = ‘X’
placeMarker(pos,
marker);
```

Output:

Prompt user to enter a valid board position

State: state of board remains unchanged:

	0	1	2	3
0				
1		O	X	
2			O	
3				

Reason:
Check to see what is how the state of the board changes when an invalid board position is entered. Checks to make sure that no former moves can be altered

Function Name:
testPlaceMarker_
with_player_full_
space

Input:

State: (number to win = 3)

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

```
pos.getRow(0)
pos.getColumn(0)
marker = ‘O’
placeMarker(pos,
marker);
```

Output:

State: state of board remains unchanged:

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

Reason:
Check to see what is how the state of the board changes when a winning move is entered.

Function Name:
testPlaceMarker_
with_diagonal_win

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

<b>State:</b> (number to win = 3)
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>0</td><td></td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
</table>

pos.getRow(0)
pos.getColumn(0)
marker = ‘O’
placeMarker(pos, marker);</td>
<td><b>Output:</b>

<b>State:</b> state of board remains unchanged:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>0</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
</table></td>
<td><b>Reason:</b>
Check to see what is how the state of the board changes the final move is entered to tie the game.

<b>Function Name:</b>
testPlaceMarker_ with_tie_placement</td>
</tr>
</table>