Griffin Smith
CPSC 2150
March 24, 2021
Requirements Analysis
System Requirements and Operation

- As a player, I have a device with java 11 installed so that the program will be able to run
- As a player, I can not place the token outside of the board so that my turn will not result in an invalid token.
- As a player, I can not place the token in a full column.
- As a player, I can run the program on this device by following the instructions under deployment
  As a player, I can not input a desired number of rows outside of 3 and 100
- As a player, I can not input a desired number of Columns outside of 3 and 100
- As a player, I can not input a desired number in a row to win outside of 3 and 25
- As a player, I can not input a desired number of players outside of 2 and 10
- Program runs on Linux
- 0,0 is the bottom left of the gameboard
- GameBoard is a 100x100 or smaller
- GamerBoard is a 3x3 or larger

Game Start

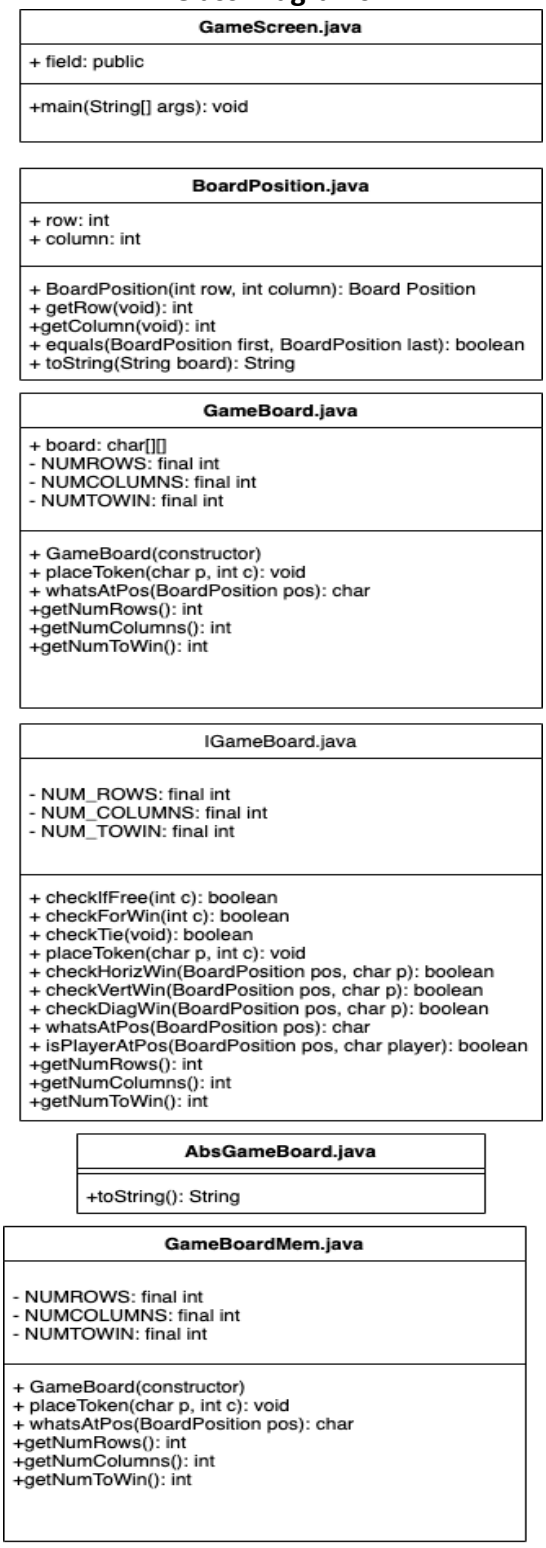
- The user will input the desired rules (players, tokens, columns, rows, number to win)
- Players will take turns placing their respected tokens in a column of their choice
    - Repeats until the game board is full (tie) or there is a winner.
- There will be a congratulating message for the winner
- The user will be asked if they want to play again
- The user must input 'Y' for yes or 'N' for no.

Deployment

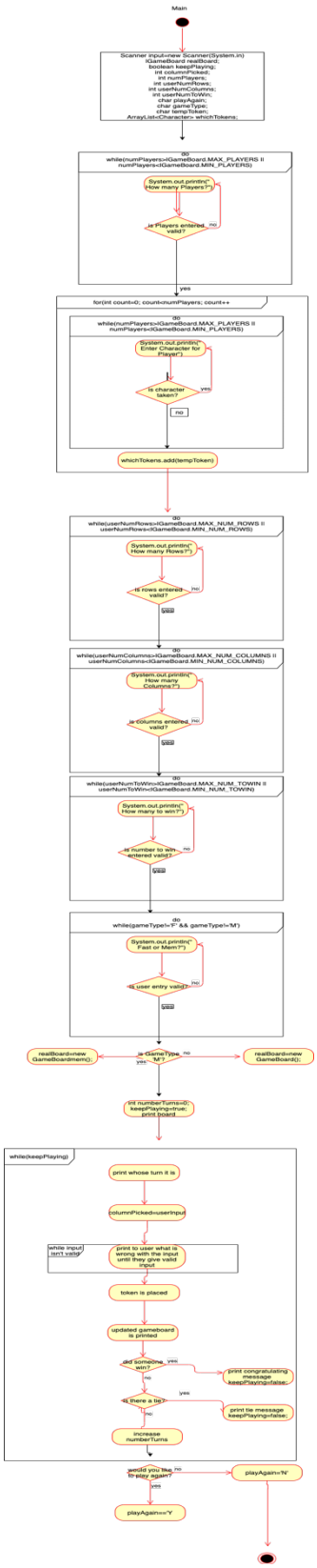
- User will open a terminal
- User will navigate to the correct location of the program in the terminal.
- User will compile program by inputting <make>.
- User will run program by inputting <make run>.
- User will delete .class files by inputting <make clean>
- User will compile test cases by inputting <make test>
- User will run test cases by inputting <make testGB> or make<testGBmem>

DESIGN:

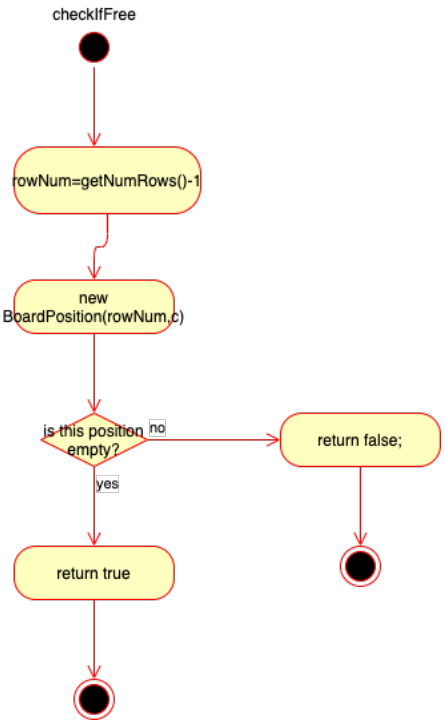
Class Diagrams

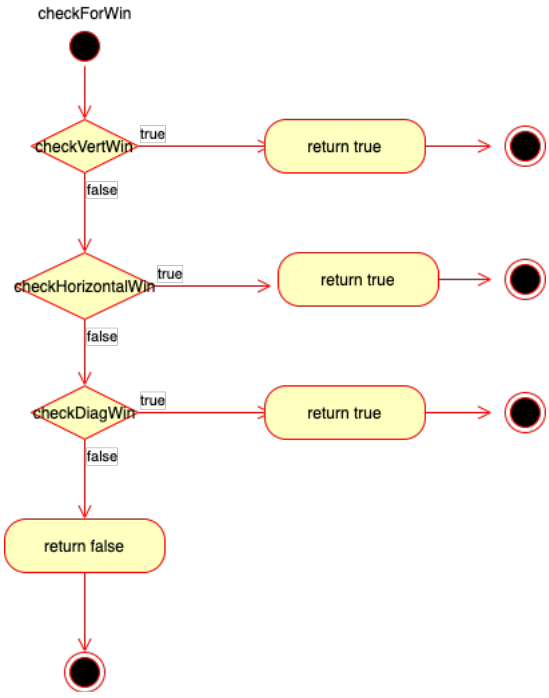


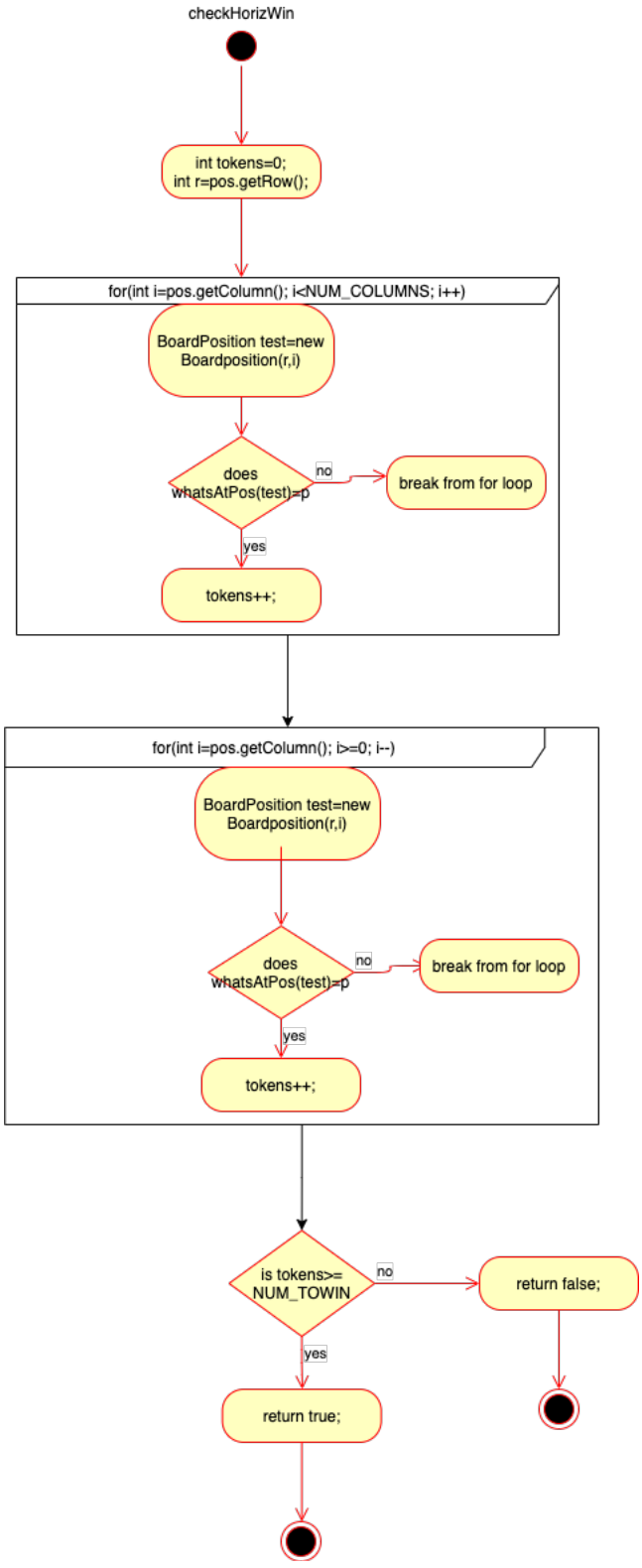
GameScreen.java

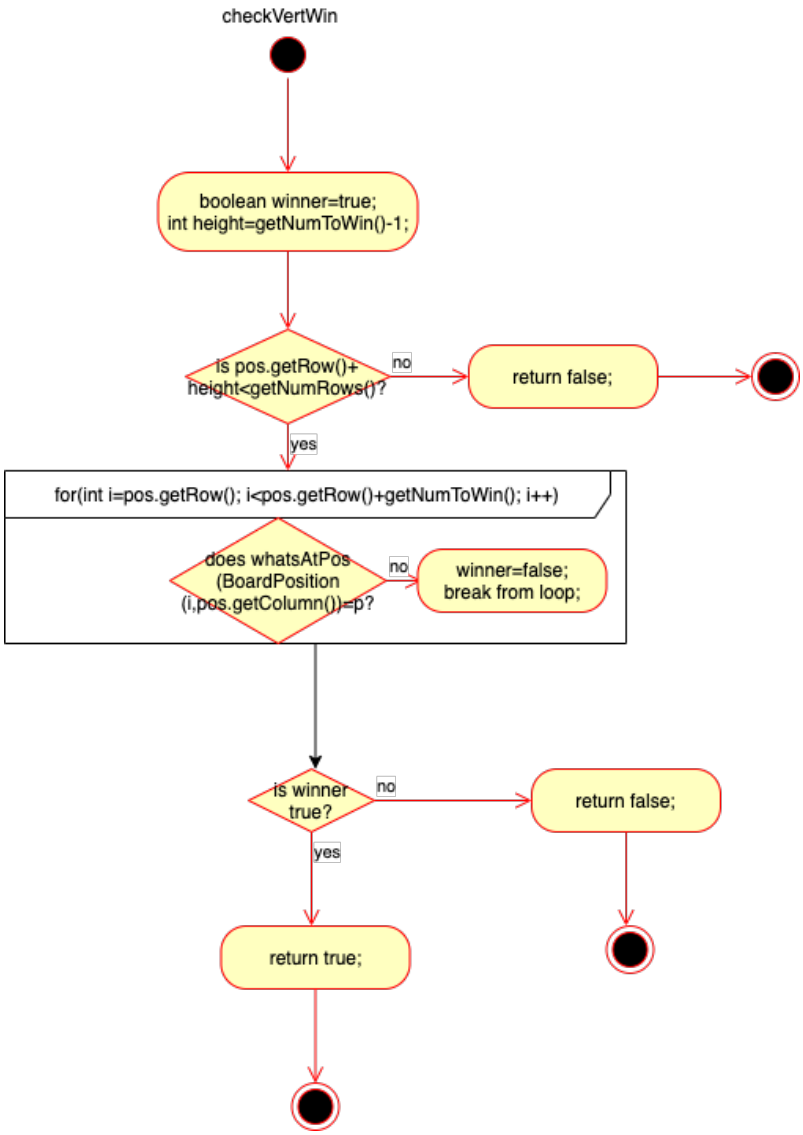


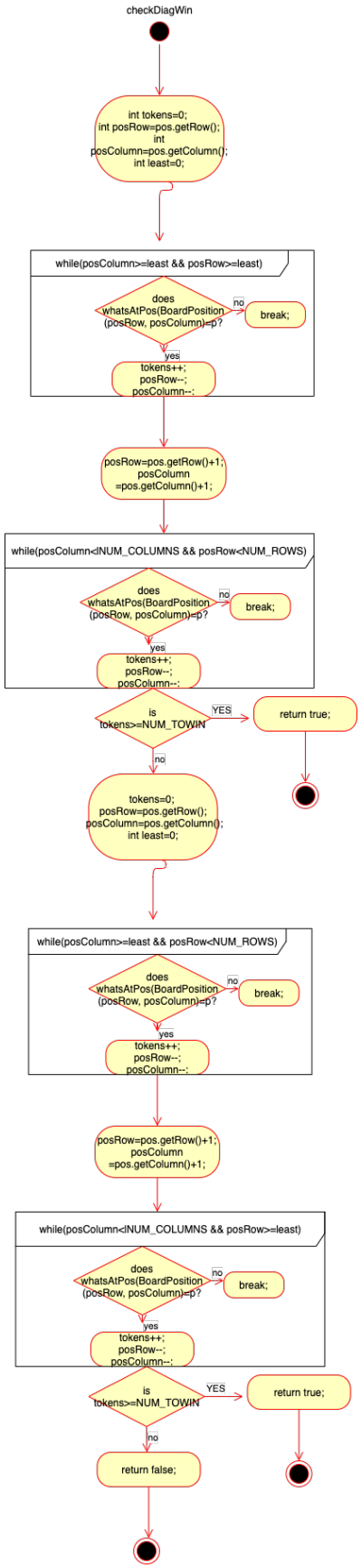
IGameBoard.java

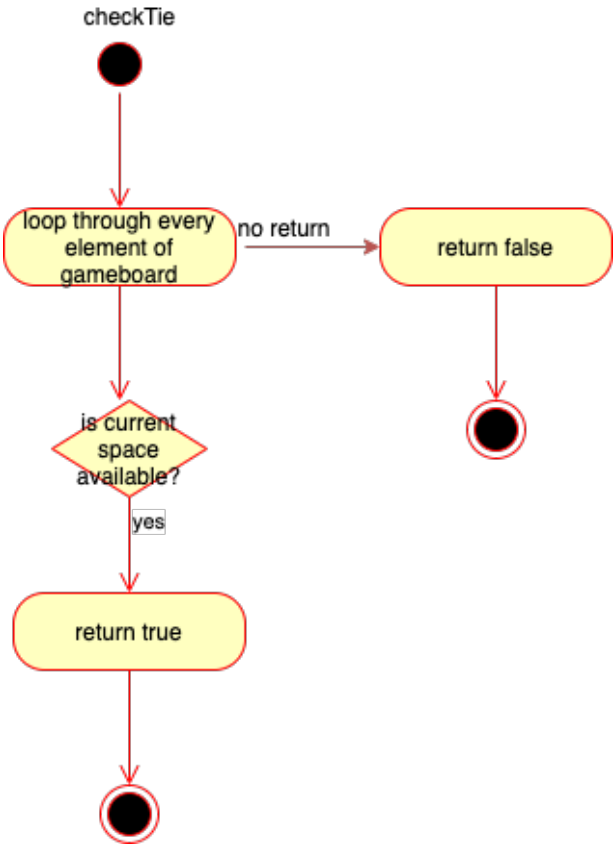

checkIfFree
rowNum=getNumRows()-1
new BoardPosition(rowNum,c)
is this position empty?
no
return false;
yes
return true

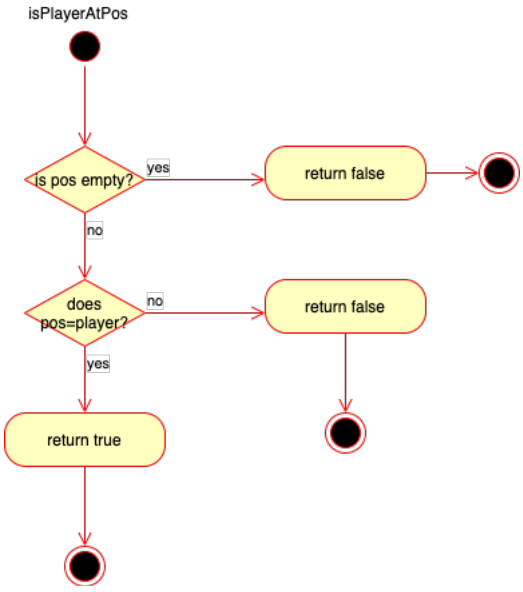

checkForWin
checkVertWin
true
return true
false
checkHorizontalWin
true
return true
false
checkDiagWin
true
return true
false
return false


checkHorizWin
int tokens=0;
int r=pos.getRow();
for(int i=pos.getColumn(); i<NUM_COLUMNS; i++)
BoardPosition test=new Boardposition(r,i)
does whatsAtPos(test)=p
no
break from for loop
yes
tokens++;
for(int i=pos.getColumn(); i>=0; i--)
BoardPosition test=new Boardposition(r,i)
does whatsAtPos(test)=p
no
break from for loop
yes
tokens++;
is tokens>= NUM_TOWIN
no
return false;
yes
return true;

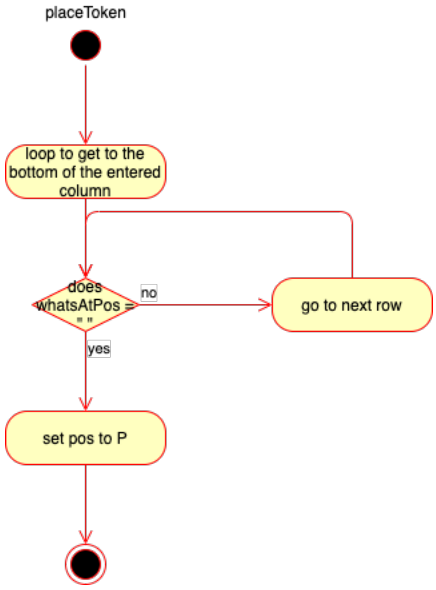

checkVertWin
boolean winner=true;
int height=getNumToWin()-1;
is pos.getRow()+
height<getNumRows()?
no
return false;
yes
for(int i=pos.getRow(); i<pos.getRow()+getNumToWin(); i++)
does whatsAtPos
(BoardPosition
(i,pos.getColumn())=p?
no
winner=false;
break from loop;
is winner
true?
no
return false;
yes
return true;

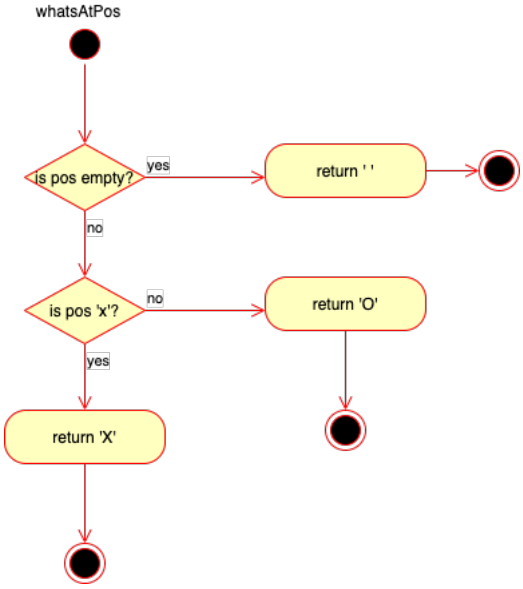

checkDiagWin
int tokens=0;
int posRow=pos.getRow();
int
posColumn=pos.getColumn();
int least=0;
while(posColumn>=least && posRow>=least)
does
whatsAtPos(BoardPosition
(posRow, posColumn)=p?
no
break;
yes
tokens++;
posRow--;
posColumn--;
posRow=pos.getRow()+1;
posColumn
=pos.getColumn()+1;
while(posColumn<lNUM_COLUMNS && posRow<NUM_ROWS)
does
whatsAtPos(BoardPosition
(posRow, posColumn)=p?
no
break;
yes
tokens++;
posRow--;
posColumn--;
is
tokens>=NUM_TOWIN
YES
return true;
no
tokens=0;
posRow=pos.getRow();
posColumn=pos.getColumn();
int least=0;
while(posColumn>=least && posRow<NUM_ROWS)
does
whatsAtPos(BoardPosition
(posRow, posColumn)=p?
no
break;
yes
tokens++;
posRow--;
posColumn--;
posRow=pos.getRow()+1;
posColumn
=pos.getColumn()+1;
while(posColumn<lNUM_COLUMNS && posRow>=least)
does
whatsAtPos(BoardPosition
(posRow, posColumn)=p?
no
break;
yes
tokens++;
posRow--;
posColumn--;
is
tokens>=NUM_TOWIN
YES
return true;
no
return false;

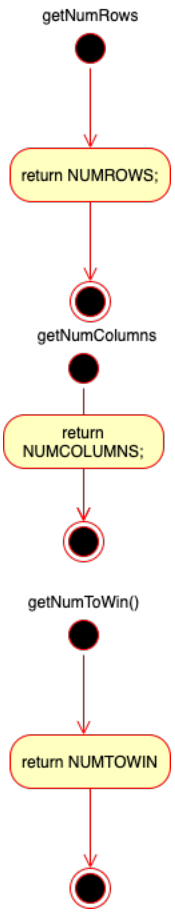

checkTie
loop through every element of gameboard
no return
return false
is current space available?
yes
return true


isPlayerAtPos
is pos empty?
yes
return false
no
does pos=player?
no
return false
yes
return true

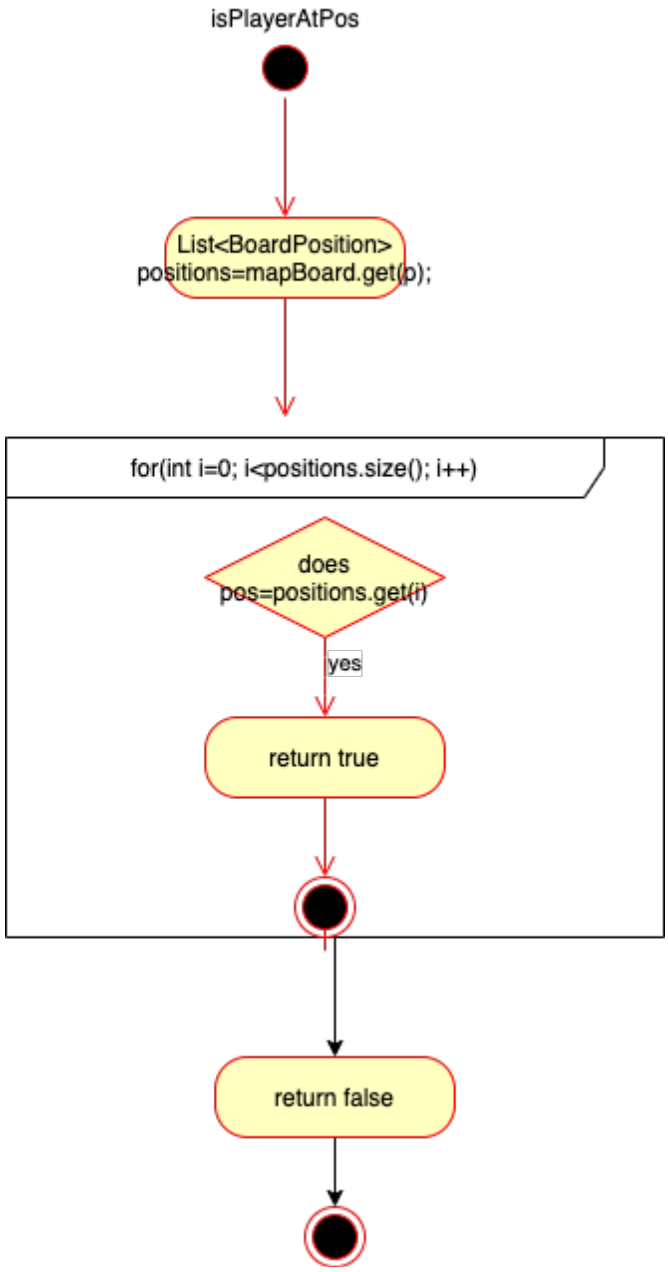
GameBoard.java

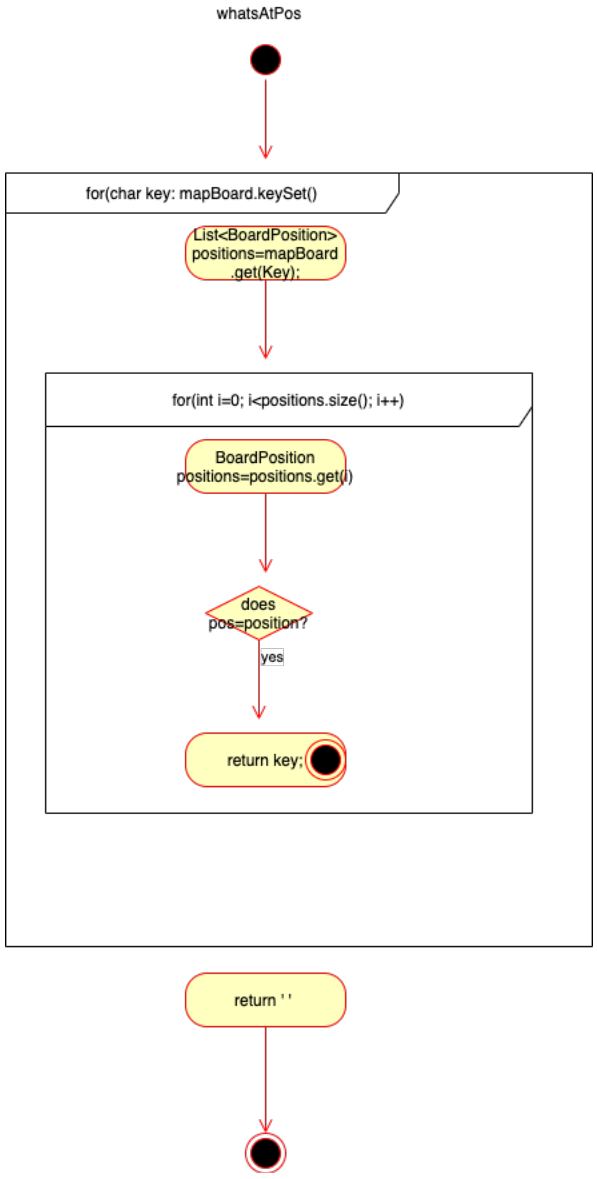


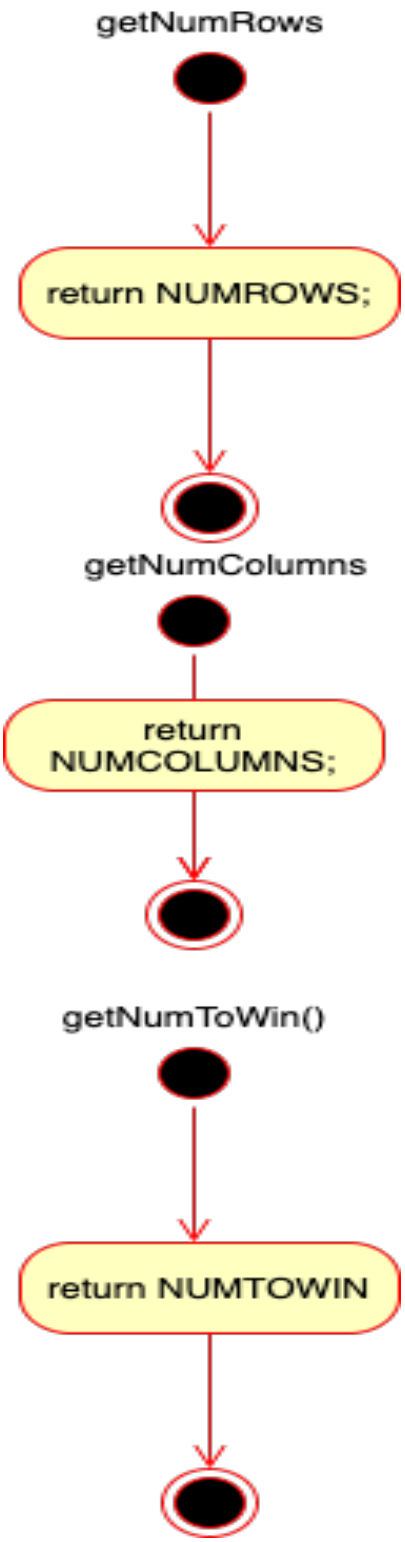


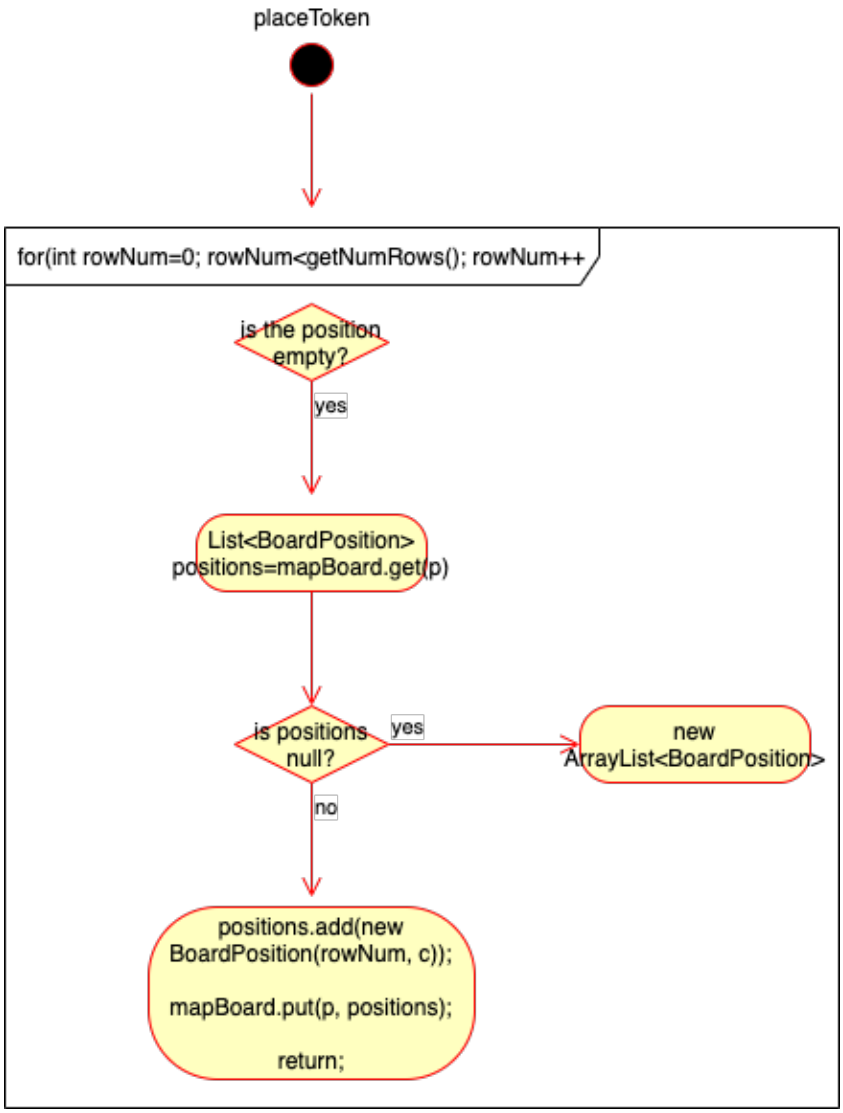

getNumRows
return NUMROWS;
getNumColumns
return NUMCOLUMNS;
getNumToWin()
return NUMTOWIN

GameBoardMem.java


isPlayerAtPos
List<BoardPosition>
positions=mapBoard.get(p);
for(int i=0; i<positions.size(); i++)
does
pos=positions.get(i)
yes
return true
return false


whatsAtPos
for(char key: mapBoard.keySet()
List<BoardPosition>
positions=mapBoard
.get(Key);
for(int i=0; i<positions.size(); i++)
BoardPosition
positions=positions.get(i)
does
pos=position?
yes
return key;
return ' '


getNumRows
return NUMROWS;
getNumColumns
return NUMCOLUMNS;
getNumToWin()
return NUMTOWIN


placeToken
for(int rowNum=0; rowNum<getNumRows(); rowNum++
is the position empty?
yes
List<BoardPosition> positions=mapBoard.get(p)
is positions null?
yes
new ArrayList<BoardPosition>
no
positions.add(new BoardPosition(rowNum, c));
mapBoard.put(p, positions);
return;

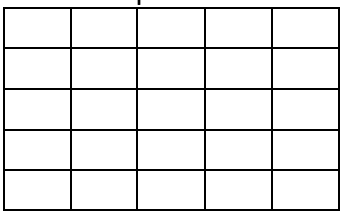
Test

public GameBoard(int userNumRows, int unserNumColumns, int userNumToWin)

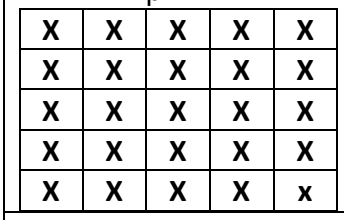
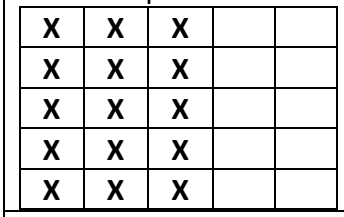
Input:	**Output:**	**Reason:**
State: userNumRows=100 userNumColumns=100 userNumToWin=25	The output is a 100x100 gameBoard. (Not Able to insert 100x100 empty table) The table is empty and every position is ' '	This test case is unique and distinct because it tests for the constructor to construct the maximum size of the game board allowed(100x100) **Function Name:** **MakeGameBoard_Max_SizeBoard()**
Input: State: userNumRows=3 userNumColumns=3 userNumToWin=3	**Output:** The 3x3 table is empty and every position is ' '	**Reason:** This test case is unique and distinct because it tests for the constructor to construct the minimum size of the game board allows(3x3) **Function Name:** **MakeGameBoard_Min_SizeBoard()**
Input: State: userNumRows=5 userNumColumns=5 userNumToWin=4	**Output:** The 3x3 table is empty and every position is ' '	**Reason:** This test case is unique and distinct because it tests for the constructor to construct the game board and tests if tokens can be placed **Function Name:** **MakeGameBoard_WholeBoard_XY()**

Boolean checkIfFree(int c)

Input:	**Output:**	**Reason:**
State: C=0	checkIfFree=false state of the board is unchanged	This test case is unique and distinct because it fills the left most column on the board with tokens and tests checkIfFree on the full column while the other columns are empty **Function Name:** **test_checkIfFree_LeftColumn()**
Input: State: C is looped from 0 to 4	**Output:** checkIfFree=false state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs and tests checkIfFree on every column on the gameboard to make sure every column is false

X X X X x		**Function Name: test_checkIfFree_LeftColumn()**
Input: State: C is looped from 0 to 4 	**Output:** checkIfFree=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it keeps the board empty and tests every column to make sure checkIfFree is true for every column on a new gameboard **Function Name: test_checkIfFree_LeftColumn()**

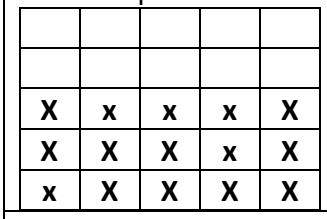
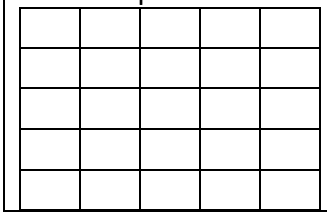
Boolean checkHorizWin(BoardPosition pos, char p)

Input: State: pos.getRow()=4 pos.getColumn()=4 p='X' 	**Output:** checkHorizWin=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs and tests checkHorizWin on the top right position of the gameboard for a horizontal win. **Function Name: test_checkHorizWin_FullBoard**
Input: State: (row and column are looped to test every position 0-4 means 0 through 4 throughout every loop iteration) pos.getRow()=0-4 pos.getColumn()=0-4 numToWin=4 p='X' 	**Output:** checkHorizWin=false state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the board with 1 less than the number to win in a row. This case tests every position on the board to make sure there is no horizontal win in any position. **Function Name: test_checkHorizWin_OneShort_ToWin**
Input: State: pos.getRow()=0 pos.getColumn()=0-4 numToWin=4	**Output:** checkHorizWin=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the bottom row of the board with Xs in a row. This tests every single position on the bottom row to

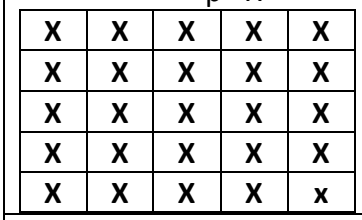
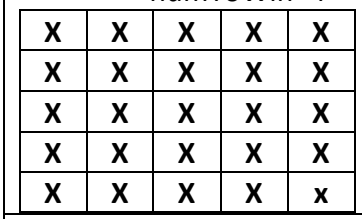
p='X' X X X X X		assert true that there is a horizontal win in every position **Function Name:** **test_checkHorizWin_BottomRow_Win**
Input: State: pos.getRow()=0-4 pos.getColumn()=0-4 numToWin=4 p='X'	**Output:** checkHorizWin=false state of the board is unchanged	**Reason:** This test case is unique and distinct because it tests an empty gameboard for a horizontal win. This test case tests every single position on the game board to determine that they are false since there are no tokens on the board **Function Name:** **test_checkHorizWin_Empty()**

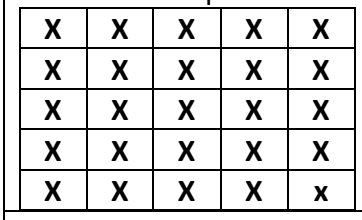
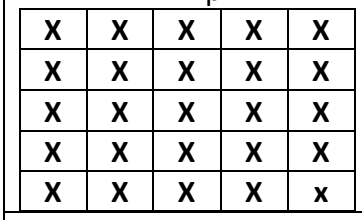
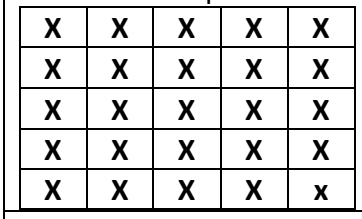
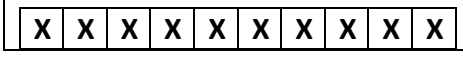
Boolean checkVertWin(BoardPosition pos, char p)

Input: State: pos.getRow()=0-4 pos.getColumn()=0-4 p='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:** checkVertWin=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs and tests checkVertWin on every single position on the board to determine that checkVertWin works on every position. **Function Name:** **test_checkVertWin_FullBoard**
Input: State: pos.getRow()=0-4 pos.getColumn()=0 numToWin=4 p='X' X X X X X	**Output:** checkVertWin=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the left most column with the same token. It tests each position in the left most column to determine that checkVertWin works on each of these positions. **Function Name:** **test_checkVertWin_FirstColumn_Win**
Input: State: pos.getRow()=0 pos.getColumn()=0-4	**Output:** checkVertWin=false	**Reason:** This test case is unique and distinct because it fills the board with 1 less than the number to win in a row vertically.

numToWin=4 p='X' 	state of the board is unchanged	This case tests every position on the board to make sure there is no vertical win in any position. **Function Name: test_checkVertWin_OneLess_NumToWin**
Input: State: pos.getRow()=0-4 pos.getColumn()=0-4 numToWin=4 p='X' 	**Output:** checkVertWin=false state of the board is unchanged	**Reason:** This test case is unique and distinct because it tests an empty gameboard for a vertical win. This test case tests every single position on the game board to determine that they are false since there are no tokens on the board **Function Name: test_checkVertWin_Empty()**

Boolean checkDiagWin(BoardPosition pos, char p)

Input: State: pos.getRow()=0 pos.getColumn()=0 numToWin=4 p='X' 	**Output:** checkDiagWin=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the bottom left to top right direction. It does this by calling checkDiagWin on the 0,0(bottom left) position. **Function Name: test_checkDiagWin_BottomLeft_UpRight**
Input: State: pos.getRow()=4(top row) pos.getColumn()=0 (pos is top left position) p='X' numToWin=4 	**Output:** checkDiagWin=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the top left to bottom right direction. It does this by calling checkDiagWin on the 4,0(top left) position. **Function Name: test_checkDiagWin_TopLeft_BottomRight**
Input:	**Output:**	**Reason:**

State: pos.getRow()=4(top row) pos.getColumn()=4 (pos is top right position) numToWin=4 p='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	checkDiagWin=true state of the board is unchanged	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the top right to bottom bottom direction. It does this by calling checkDiagWin on the 4,4(top right) position. **Function Name: test_checkDiagWin_TopRight_BottomLeft**
Input: State: pos.getRow()=0(top row) pos.getColumn()=4 (pos is bottom right position) numToWin=4 p='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:** checkDiagWin=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the bottom right to top left direction. It does this by calling checkDiagWin on the 0,4(bottom right) position. **Function Name: test_checkDiagWin_BottomRight_TopLeft**
Input: State: pos.getRow()=2 pos.getColumn()=2 (pos is middle position) numToWin=4 p='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:** checkDiagWin=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. **Function Name: test_checkDiagWin_FullBoard_TestMiddlePos**
Input: State: pos.getRow()=0 pos.getColumn()=0 (pos is bottom left position) p='X' numToWin=10  X X X X X X X X X X	**Output:** checkDiagWin=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin when the number to win is equal to the number of rows and number of columns in the board.

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		**Function Name:** **test_checkDiagWin_WinNum_** **EqualBoardSize_FullWin**
Input: State: pos.getRow()=0-4 pos.getColumn()=0-4 numToWin=4 p='X' X O Y Q W X O Y Q W X O Y Q W X O Y Q W X O y Q W	**Output:** checkDiagWin= false state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs and Ys in a checker board pattern. This case tests every position on the game board to determine that no positions result in a diagonal win. **Function Name:** **test_checkDiagWin_CheckPattern_No Win**

Boolean checkTie()

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	**Output:** checkTie=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs in order to make the board full and then it calls checkTie to confirm that the board is full, therefore there is a tie. **Function Name:** **test_checkTie_Full_Board**
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	**Output:** checkTie=false state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs except for one position. Then it tests checkTie in order to confirm that there is not yet a tie since a token can still be placed. **Function Name:** **test_checkTie_Full_Board**
Input: State: X X X X	**Output:** checkTie=false	**Reason:** This test case is unique and distinct because it fills the whole board with Xs except for one column. Then it tests

X X X X X X X X X X X X X X X X	state of the board is unchanged	checkTie in order to confirm that there is not yet a tie since a token can still be placed in the open column. Confirms that checkTie still works when a full column is open. **Function Name: test_checkTie_Missing_RightColumn**
Input userNumRows=100 userNumColumns=100 userNumToWin=25 100x100 gameboard is filled with X and Ys in a checkerboard pattern. (Not Able to insert 100x100 empty table)	checkTie=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs in order to make the board full and then it calls checkTie to confirm that the board is full, therefore there is a tie. This test case is distinct because it fills up the maximum size gameboard(100x100) confirming that no matter how large the size, the checkTie function will still work. **Function Name: test_checkTie_Max_SizeBoard_Full**

Char whatsAtPos(BoardPosition pos)

Input: State: Pos.row=4 Pos.col=4 (top right position) X X X X O X X X X X X X X X X X X X X X X X X X x	**Output:** whatsAtPos='O' state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs except for one position. Then it tests whatsAtPos to determine that the function works for the position that is not an X in the top right position on the Gameboard. **Function Name: test_whatsAtPos_Top_Right**
Input: State: pos.getRow()=0-4 pos.getColumn()=0-4	whatsAtPos=' ' state of the board is unchanged	**Reason:** This test case is unique and distinct because it calls an empty gameboard. It uses whatsAtPos to determine that every position in the gameboard is initialized to ' ' **Function Name: test_whatsAtPos_EmptyBoard**
Input userNumRows=100	whatsAtPos=X'	**Reason:**

userNumColumns=100 userNumToWin=25 100x100 gameboard is filled with Xs. (Not Able to insert 100x100 empty table) Pos.getRow()=0-99 Pos.getColumn()=0-99	state of the board is unchanged	This test case is unique and distinct because it uses a gameboard that is the maximum size (100x100) and fills it with Xs. whatsAtPos is tested on every position to determine that each position = 'X' **Function Name:** **test_whatsAtPos_Max_Size_FullBoard**
Input: State: pos.getRow()=0-4 pos.getColumn()=0-4 X X X X X X X x X	whatsAtPos=X ' state of the board is unchanged	**Reason:** This test case is unique and distinct because it uses a gameboard that is the minimum size (3x3) and fills it with Xs. whatsAtPos is tested on every position to determine that each position = 'X' **Function Name:** **test_whatsAtPos_Min_Size_FullBoard**
Input: State: Pos.getRow()=0 Pos.getColumn()=0 **X X X X** **X X X X** **X X X X** **X X X X** **X X X X**	whatsAtPos=X ' state of the board is unchanged	**Reason:** This test case is unique and distinct because it does not fill the gameboard all the way, but calls the function in the bottom left corner to determine that the function works when the gameboard is not full. **Function Name:** **test_whatsAtPos_BottomLeftX**

Boolean isPlayerAtPos(BoardPosition pos, char p)

Input: State: Pos.row=4 Pos.col=4 P='O' (top right position) **X X X X O** **X X X X X** **X X X X X** **X X X X X** **X X X X x**	**Output:** isPlayerAtPos=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it fills the whole board with Xs except for one position. Then it tests isPlayerAtPos to determine that the function returns true for the position that is not an X in the top right position on the Gameboard. **Function Name:** **test_isPlayerAtPos_Top_Right**
Input: State: Pos.getRow()=0 Pos.getColumn()=0	isPlayerAtPos=true state of the board is unchanged	**Reason:** This test case is unique and distinct because it does not fill the gameboard all the way, but calls the function in the

<table>
<tr>
<td>P='X'
<table><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr></table></td>
<td></td>
<td>bottom left corner to determine that the function works when the gameboard is not full and that the function correctly identifies the correct player token.
Function Name:
test_isPlayerAtPos_BottomLeftX</td>
</tr>
<tr>
<td>Input:
State:
pos.getRow()=0-4
pos.getColumn()=0-4
p='X'
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>isPlayerAtPos=false

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it calls an empty gameboard. It uses isPlayerAtPos to determine that every position in the gameboard is initialized to ' ' and not to a character token
Function Name:
test_isPlayerAtPos_EmptyBoard</td>
</tr>
<tr>
<td>Input
userNumRows=100
userNumColumns=100
userNumToWin=25
100x100 gameboard is filled with Xs.
(Not Able to insert 100x100 empty table)
Pos.getRow()=0-99
Pos.getColumn()=0-99
P='X'</td>
<td>isPlayerAtPos=true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it uses a gameboard that is the maximum size (100x100) and fills it with Xs. isPlayerAtPos is tested on every position to determine that each position returns true.
Function Name:
test_isPlayerAtPos_Max_Size_FullBoard</td>
</tr>
<tr>
<td>Input:
State:
pos.getRow()=0-4
pos.getColumn()=0-4
p='X'
<table><tr><td>X</td><td>X</td><td>X</td></tr><tr><td>X</td><td>X</td><td>X</td></tr><tr><td>X</td><td>x</td><td>X</td></tr></table></td>
<td>isPlayerAtPos=true

state of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it uses a gameboard that is the minimum size (3x3) and fills it with Xs. isPlayerAtPos is tested on every position to determine that each position returns true.
Function Name:
test_isPlayerAtPos_Min_Size_FullBoard</td>
</tr>
</table>

Void placeToken(char p, int c)

<table>
<tr>
<td>Input:
State:
P='X'
C=0-4</td>
<td>Output:
<table><tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr></table></td>
<td>Reason:
This test case is unique and distinct because I placed the X marker in every position on the gameboard.</td>
</tr>
</table>

	X X X X X **X X X X X** **X X X X x**	**Function Name:** **test_placeToken_Full_Board**
Input: State: P='X' C=1	**Output:** **X**	**Reason:** This test case is unique and distinct because I placed the X marker in a column that is empty. This is also unique because it is the first token placed on the newly initialized board **Function Name:** **test_placeToken_newBoard**
Input: State: P='X','Y','Z','W','Q' C=0-4	**Output:** **X Y Z W Q** **X Y Z W Q** **X Y Z W Q** **X Y Z W Q** **X Y Z W Q**	**Reason:** This test case is unique and distinct because I place different markers. This shows that the function works when placing different markers. **Function Name:** **test_placeToken_Full_BoardXYZWQ**
Input userNumRows=100 userNumColumns=100 userNumToWin=25 100x100 gameboard is filled with Xs. (Not Able to insert 100x100 empty table) C=0-99 P='X'	**Output** Output is a 100x100 gameboard completely filled with Xs	**Reason:** This test case is unique and distinct because I use a maximum sized board. This shows that the function continues working throughout the length of the program, no matter how big a board the user wants. **Function Name:** **test_placeToken_MaxBoard**
Input: State: C=0-2 p='X'	**Output** X X X X X X X x X	**Reason:** This test case is unique and distinct because I use a minimum sized board. This shows that the function continues working throughout the length of the program, no matter how small a board the user wants. **Function Name:** **test_placeToken_MinBoard**