CPSC 2150 Project Report

Jason Rodgers

Requirements Analysis

Functional Requirements:

1. As a player, I need to know if I am X's or O's, so that I know if I am going first or not.
2. As a player, I need to know where I can move on the board, so that I can make the best possible next move.
3. As a player, I need to know if it is my turn to make a move, so that I can so that I can place my token on the board.
4. As a player, I want to be able see the board, so that I can see how I am doing
5. As a player, I need to know what spots are taken on the board, so that I don't move there on the board.
6. As a player, I need to know if I won or loss, so that I know when the game is over.
7. As a player, I want to know if I can play again, so that I can restart the game.
8. As a player, I want to be able to see all the moves made so far on the board, so that I can see if I need to play defense or offense.
9. As a player, I need to know if there are no possible moves left on the board, so that I know the game ended in a tie.
10. As a player, I want to see the winning combination that won the game, so that I can see how I won or loss.
11. As a player, I want to know what the columns are listed as, so that I ensure I place my token in the right column.
12. As a player, I need to identify where my tokens are, so that I can spot out the many ways for me to win the game.
13. As a player, I need to the know different combinations in which I can win, so that I know how to play the game.
14. As a player, I want to know how many rows and columns there are, so that I can make a strategy of how I am going win.
15. As a player, I want to know if I can change the rows and columns of the game, so that I can change how many tokens I would need in a row to win the game.
16. As a player, I want to know if I can change my token position after I place it, so that I can change my mistake if I misplace my token.
17. As a player, I want to know If I can change the token names from X and O, so that I can have some customizability in the game.
18. As a player, I want to enter the number of rows and columns that the game board will have, so that I have some control over the game.

19. As a player, I want to be able to see why that I cannot place my token at a certain position, so that I can put it at another valid position.
20. I want to get a winning message when I win the game, so that I feel rewarded after winning.
21. As a player, I can place a marker in a column, so I can claim a space.
22. As a player, if I get 5 in a row horizontally, I will win the game so that I can win the game.
23. As a player, if I get 5 in a row vertically, I will win the game so that I can win the game.
24. As a player, if I get 5 in a row diagonally, I will win the game so that I can win the game.
25. As a player, I can choose to play again, so I can play again.
26. As a player, I can choose the number of rows, so that I can play with the game board that I want.
27. As a player, I can choose the number of columns, so that I can play extended connect X the way that I want.
28. As a player, I can choose how many numbers are needed to win, so that I can play the game how I want to play.
29. As a player, I can choose the number of players I want to play the game, so that I can play with as many people as I want to.
30. As a player, I want to choose the token character that I want to play with, so that I am able to easily identify which token is my token.
31. As a player, I want to be able to click where I would like to place my token, so that I can get the full experience of the game.
32. As a player, I want to enter rows, columns, and numbers to win, so that I can generate a board that the players are able to play on.

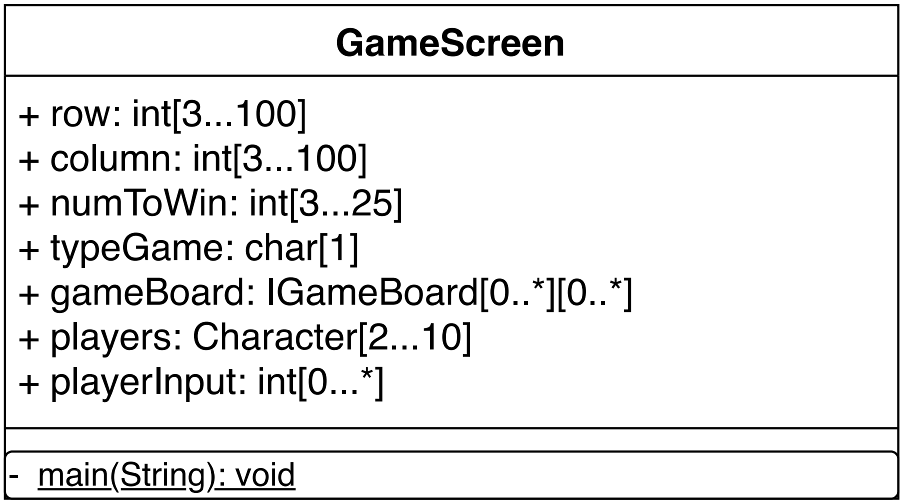
Non-Functional Requirements

1. The system must be written in Java.
2. The system must be able to gather input from the user.
3. The system must be able to generate as many rows as the player wants.
4. The system must be able to generate as many columns as the player wants.
5. The system must be able to handle multiple players playing the game.
6. The system must be able to generate more than the average numbers to get a win in the original game of connect 4.
7. The system must utilize algorithms to decrease run time and increase performance and efficiency.
8. Position 0,0 is at the bottom left of the game board.
9. The system must be able to generate a GUI for extendedConnectX.

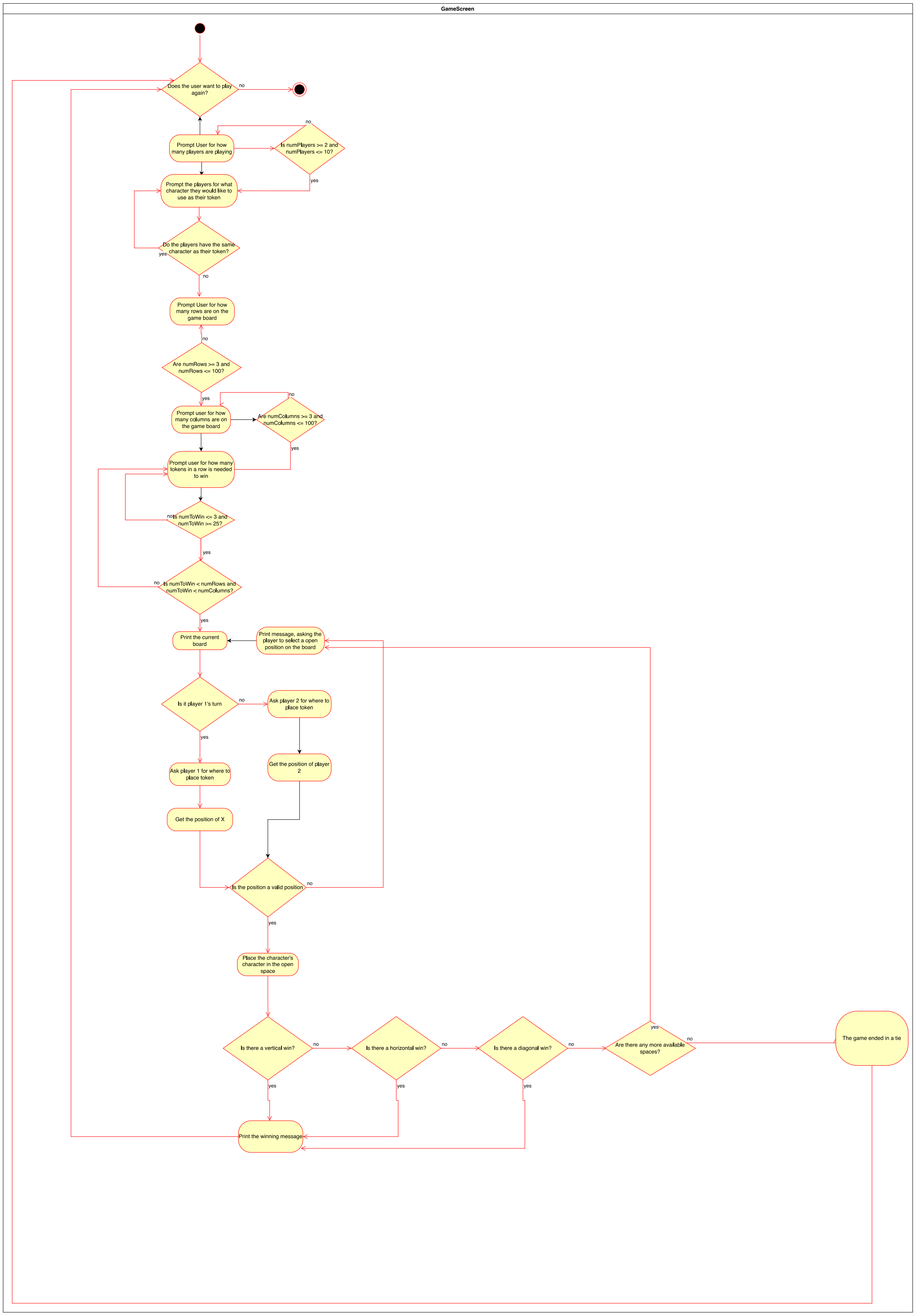
Deployment Instructions: This is an IntelliJ Project.

Class 1: GameScreen

Class Diagram:

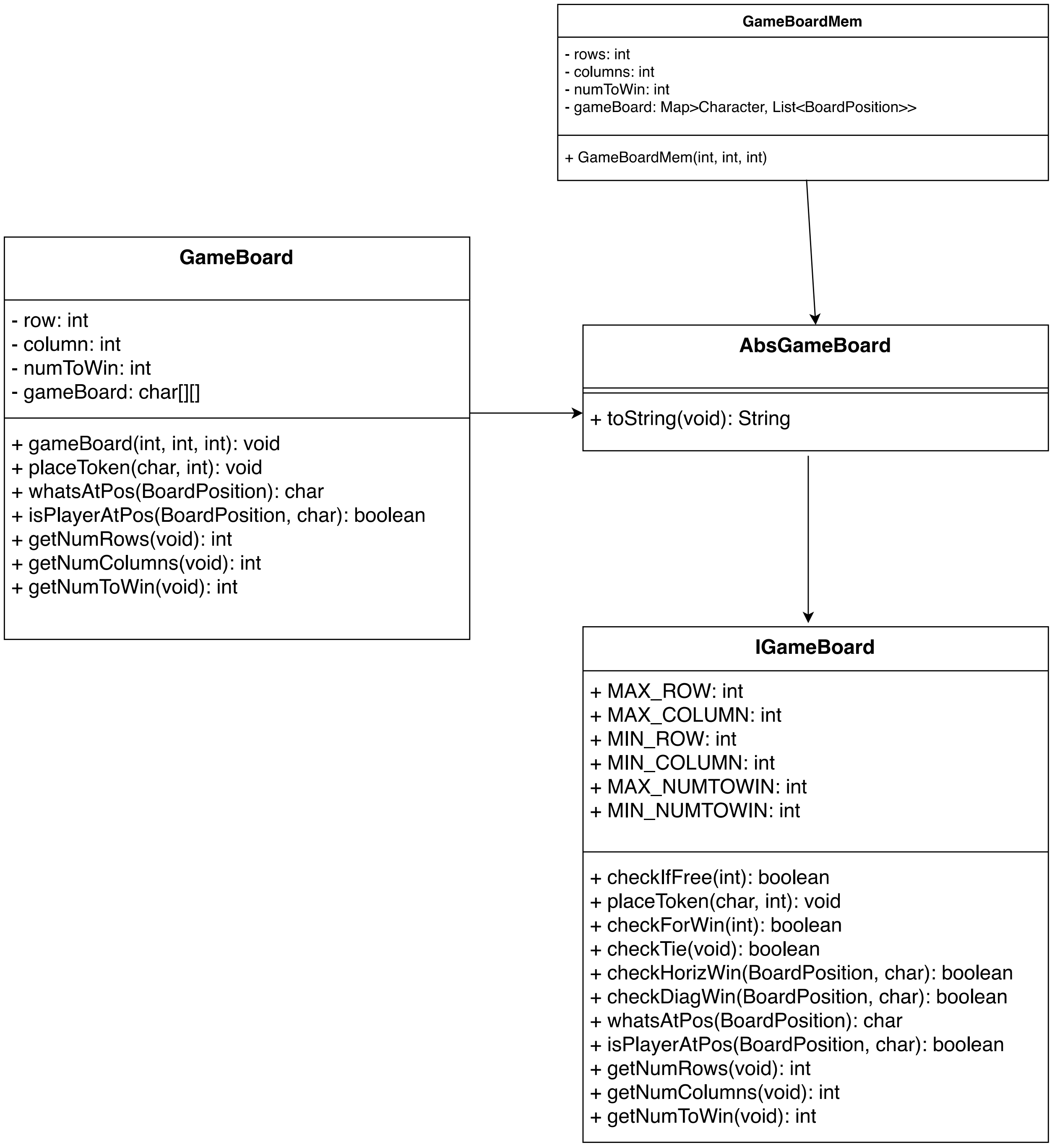


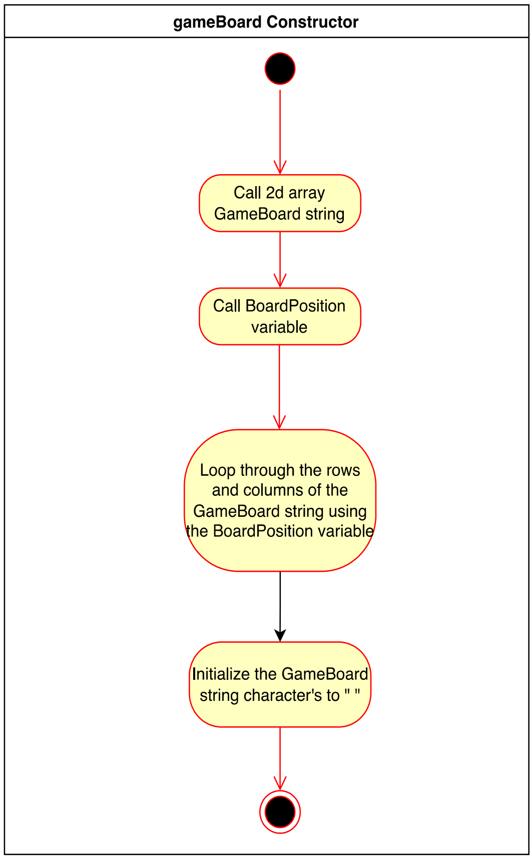
Activity diagrams (UML Diagram)

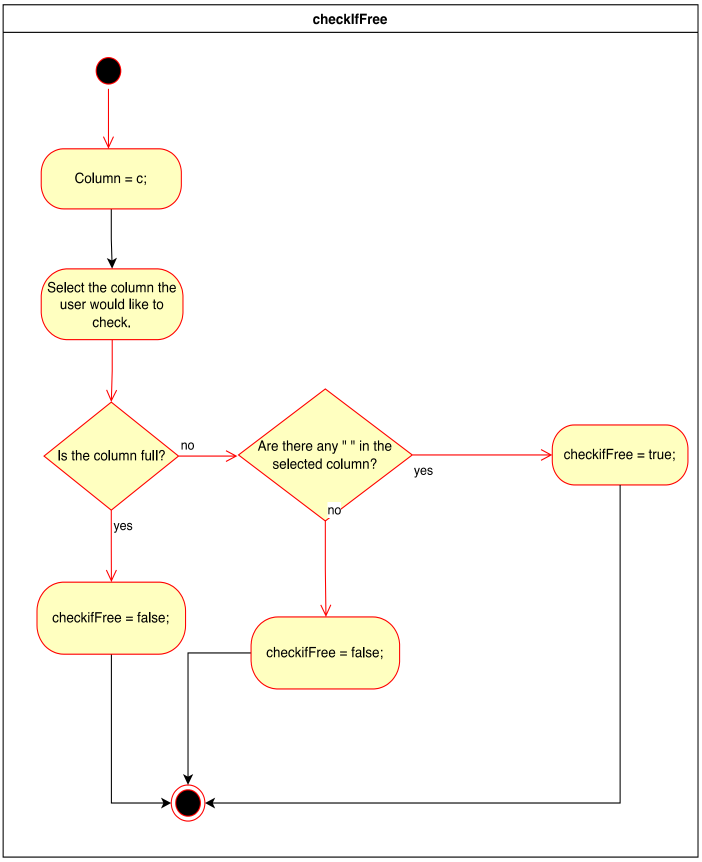

GameScreen
Does the user want to play again?
no
Prompt User for how many players are playing
Is numPlayers >= 2 and numPlayers <= 10?
no
yes
Prompt the players for what character they would like to use as their token
Do the players have the same character as their token?
yes
no
Prompt User for how many rows are on the game board
no
Are numRows >= 3 and numRows <= 100?
yes
Prompt user for how many columns are on the game board
Are numColumns >= 3 and numColumns <= 100?
no
yes
Prompt user for how many tokens in a row is needed to win
no
Is numToWin <= 3 and numToWin >= 25?
yes
no
Is numToWin < numRows and numToWin < numColumns?
yes
Print the current board
Print message, asking the player to select a open position on the board
Is it player 1's turn
no
yes
Ask player 2 for where to place token
Get the position of player 2
Ask player 1 for where to place token
Get the position of X
Is the position a valid position
no
yes
Place the character's character in the open space
Is there a vertical win?
no
Is there a horizontal win?
no
Is there a diagonal win?
no
Are there any more available spaces?
yes
no
The game ended in a tie
yes
yes
yes
Print the winning message

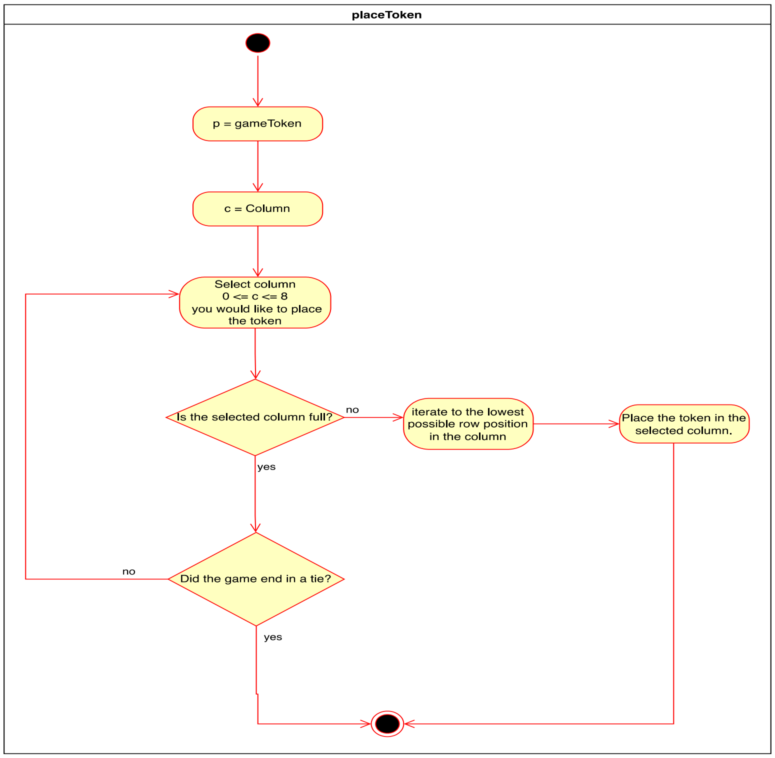
Class 2: GameBoard

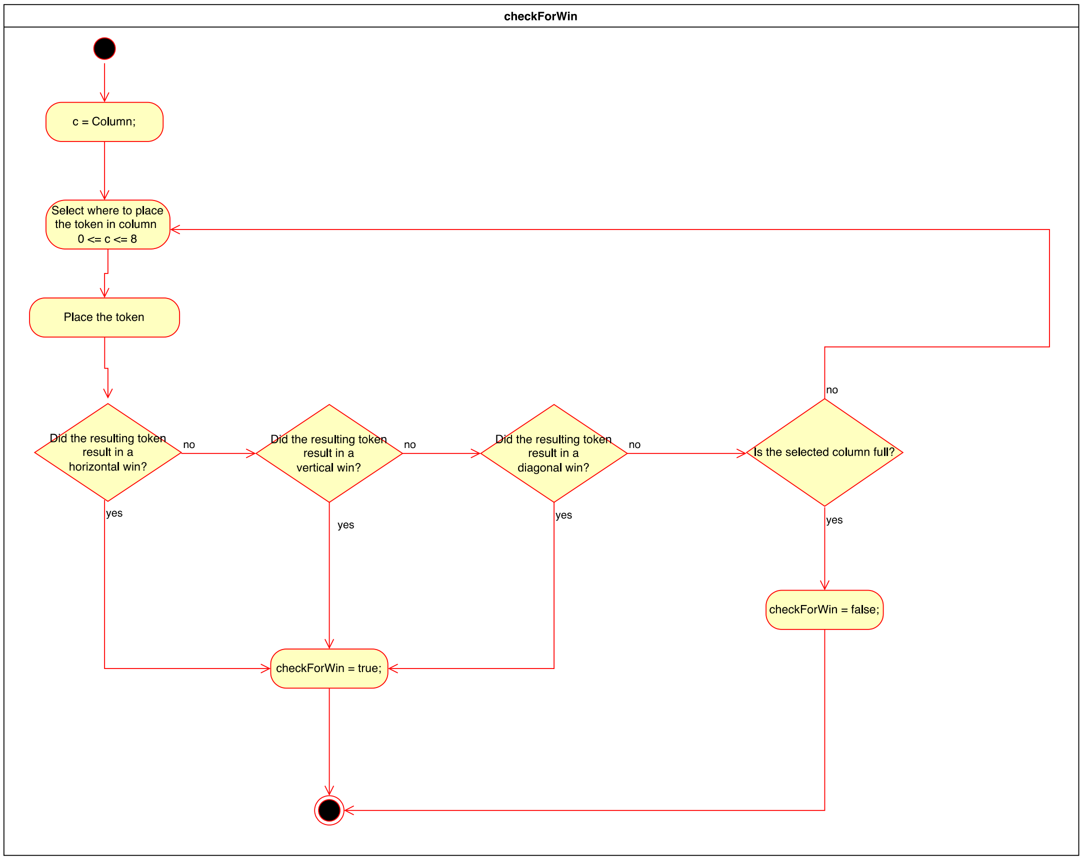
Class diagram:

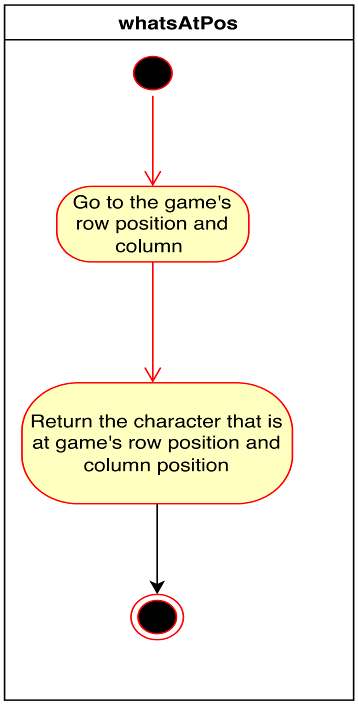

GameBoardMem
- rows: int
- columns: int
- numToWin: int
- gameBoard: Map>Character, List<BoardPosition>>
+ GameBoardMem(int, int, int)
GameBoard
- row: int
- column: int
- numToWin: int
- gameBoard: char[][]
+ gameBoard(int, int, int): void
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int
AbsGameBoard
+ toString(void): String
IGameBoard
+ MAX_ROW: int
+ MAX_COLUMN: int
+ MIN_ROW: int
+ MIN_COLUMN: int
+ MAX_NUMTOWIN: int
+ MIN_NUMTOWIN: int
+ checkIfFree(int): boolean
+ placeToken(char, int): void
+ checkForWin(int): boolean
+ checkTie(void): boolean
+ checkHorizWin(BoardPosition, char): boolean
+ checkDiagWin(BoardPosition, char): boolean
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int

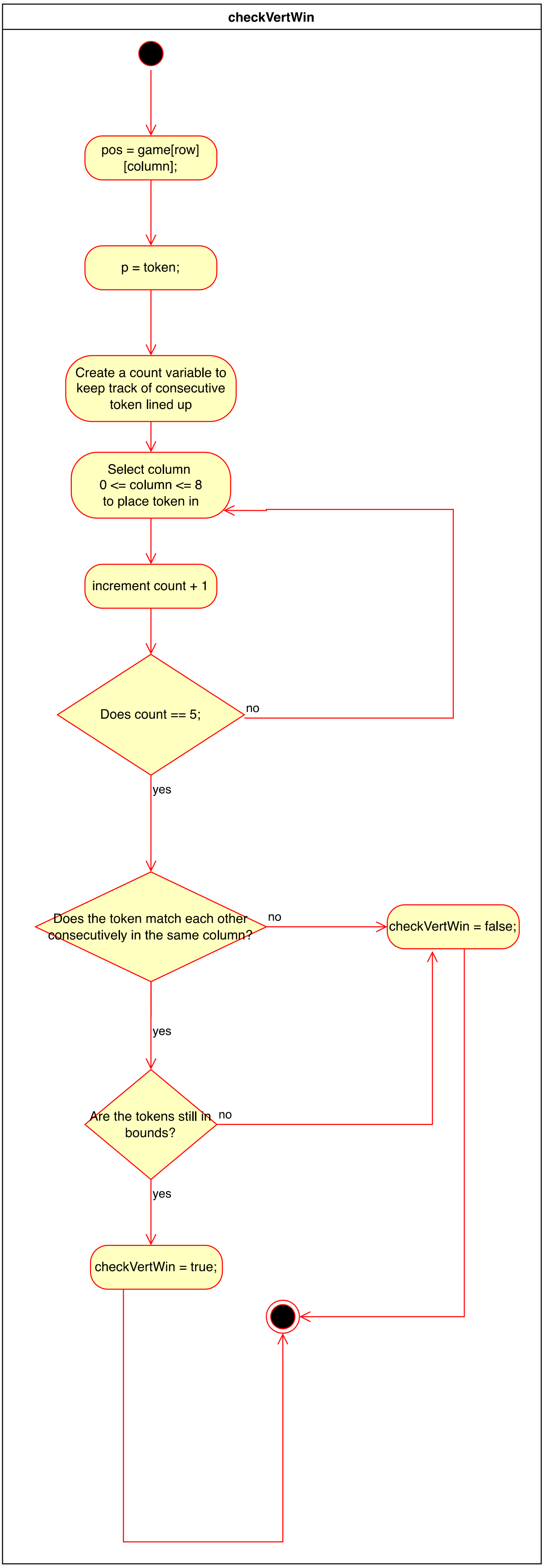

gameBoard Constructor
Call 2d array GameBoard string
Call BoardPosition variable
Loop through the rows and columns of the GameBoard string using the BoardPosition variable
Initialize the GameBoard string character's to " "

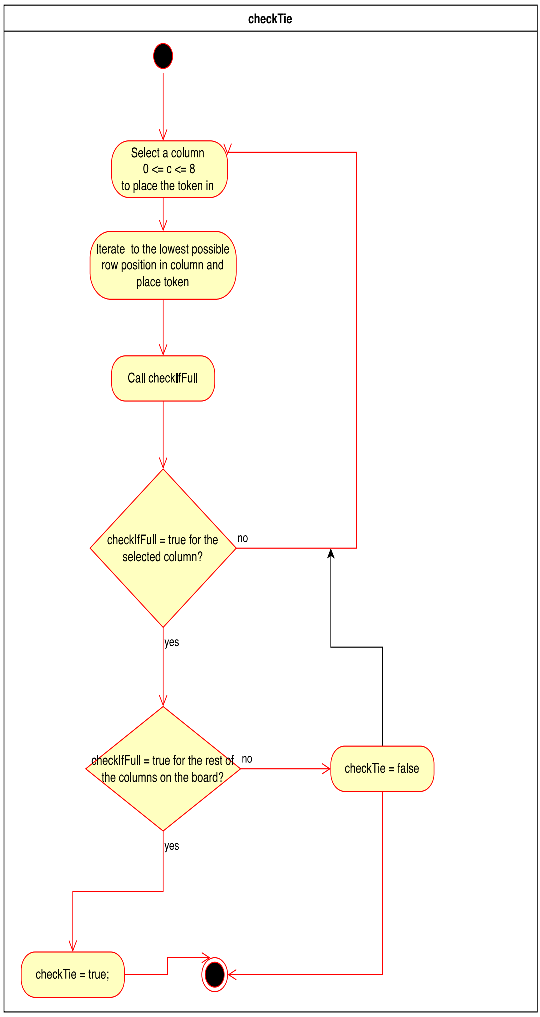

checkIfFree
Column = c;
Select the column the user would like to check.
Is the column full?
no
yes
Are there any " " in the selected column?
yes
no
checkifFree = true;
checkifFree = false;
checkifFree = false;

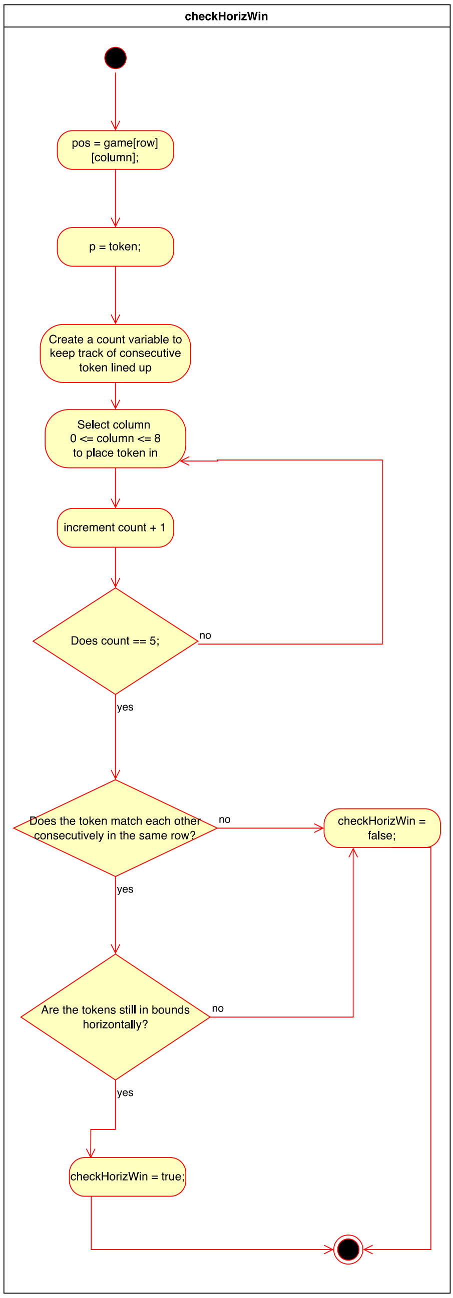

placeToken
p = gameToken
c = Column
Select column
0 <= c <= 8
you would like to place the token
Is the selected column full?
no
yes
iterate to the lowest possible row position in the column
Place the token in the selected column.
Did the game end in a tie?
no
yes

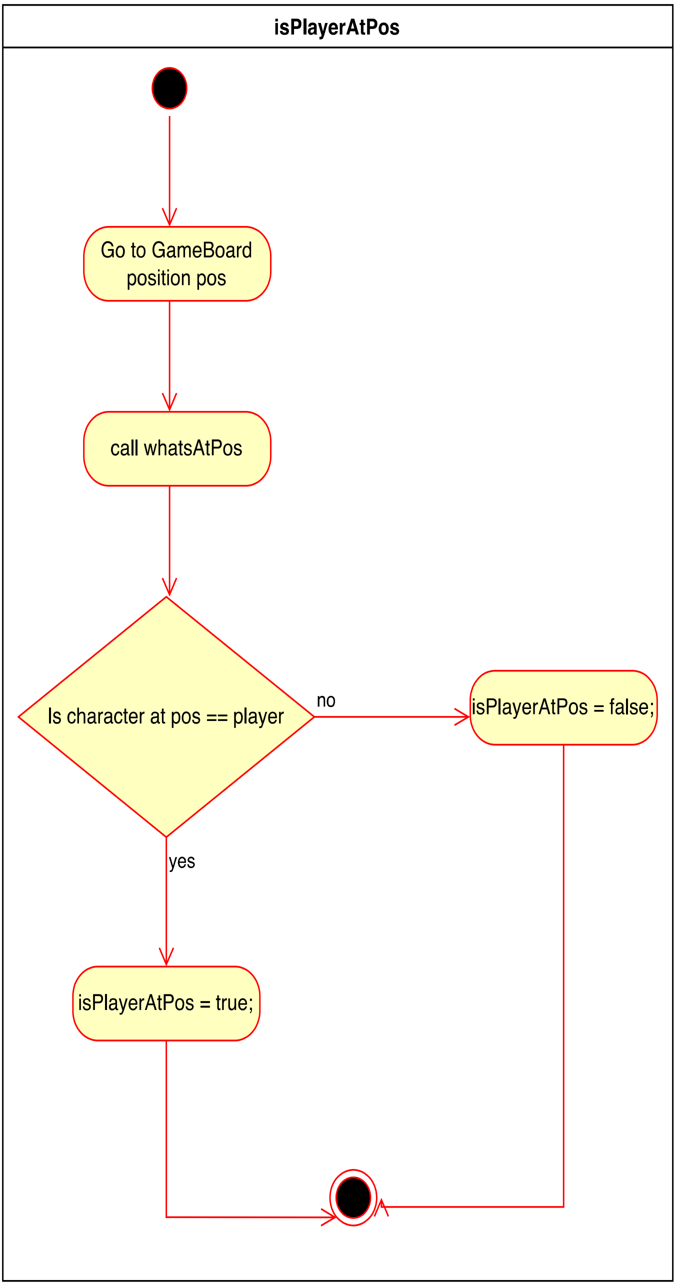

checkForWin
c = Column;
Select where to place the token in column 0 <= c <= 8
Place the token
Did the resulting token result in a horizontal win?
no
yes
Did the resulting token result in a vertical win?
no
yes
Did the resulting token result in a diagonal win?
no
yes
Is the selected column full?
no
yes
checkForWin = false;
checkForWin = true;

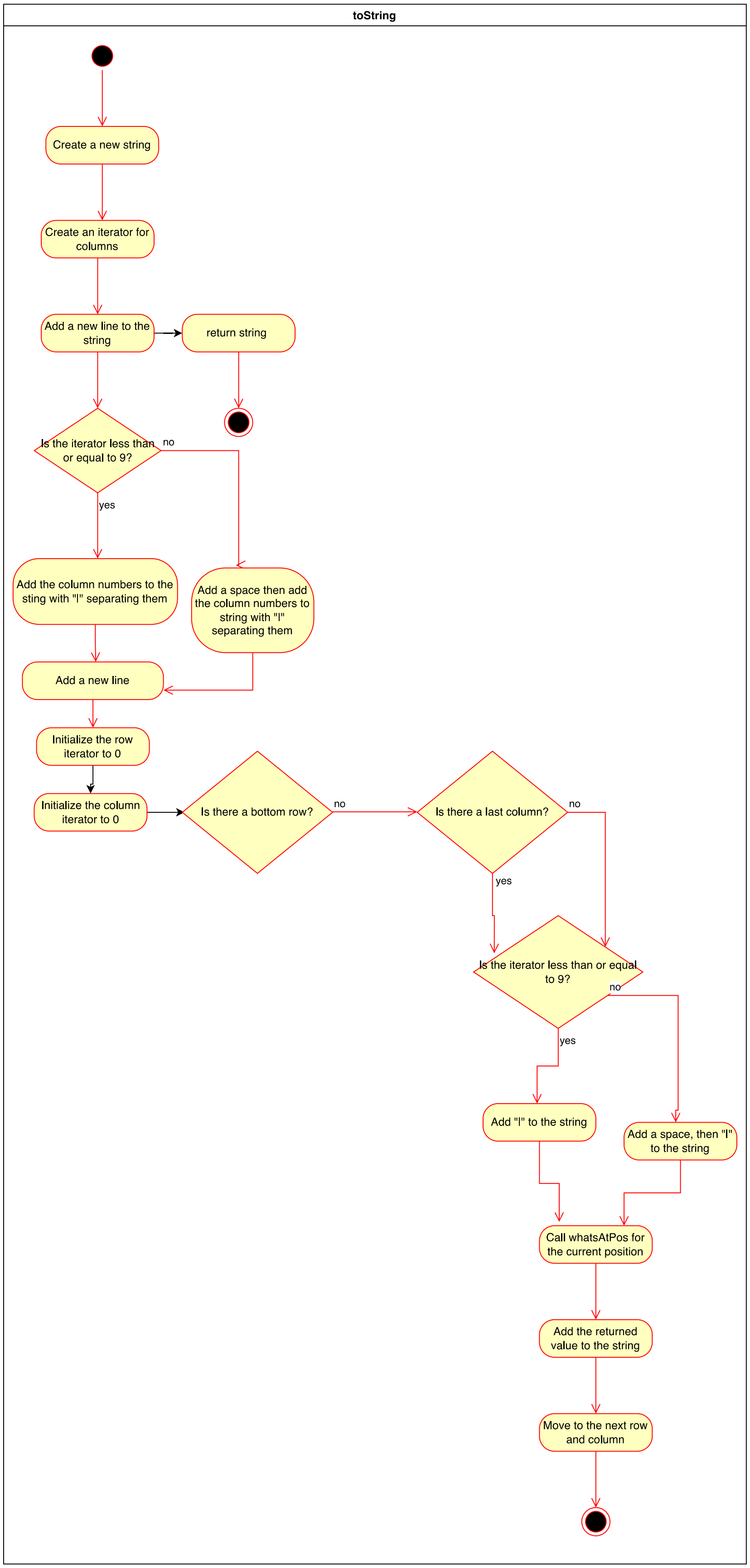

whatsAtPos
Go to the game's row position and column
Return the character that is at game's row position and column position

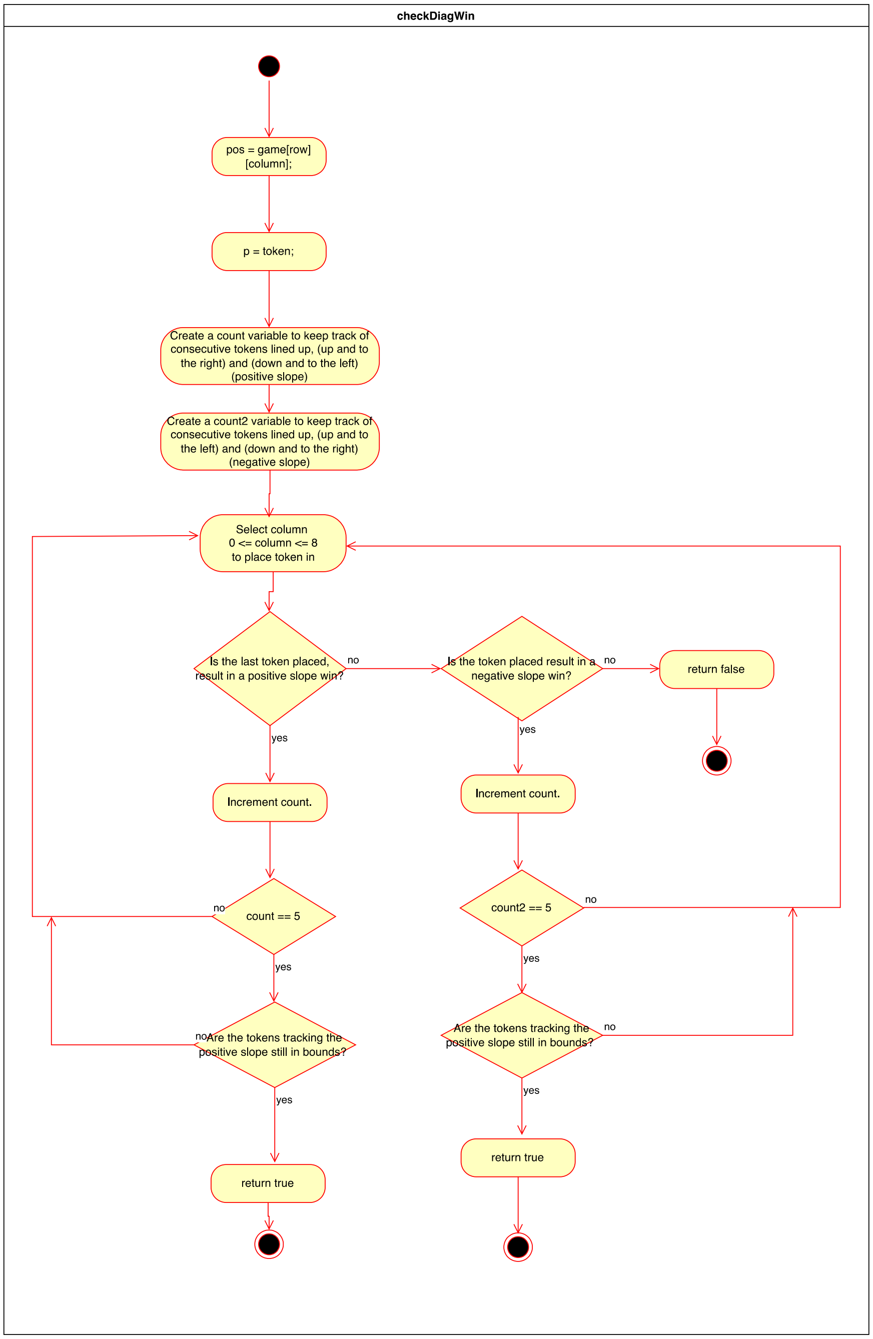

checkVertWin
pos = game[row][column];
p = token;
Create a count variable to keep track of consecutive token lined up
Select column 0 <= column <= 8 to place token in
increment count + 1
Does count == 5;
no
yes
Does the token match each other consecutively in the same column?
no
checkVertWin = false;
yes
Are the tokens still in bounds?
no
yes
checkVertWin = true;

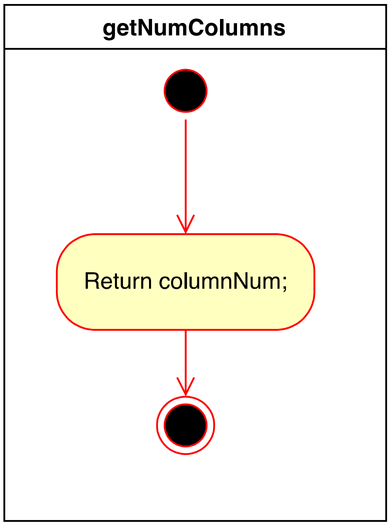

checkTie
Select a column 0 <= c <= 8 to place the token in
Iterate to the lowest possible row position in column and place token
Call checkIfFull
checkIfFull = true for the selected column?
no
yes
checkIfFull = true for the rest of the columns on the board?
no
checkTie = false
yes
checkTie = true;

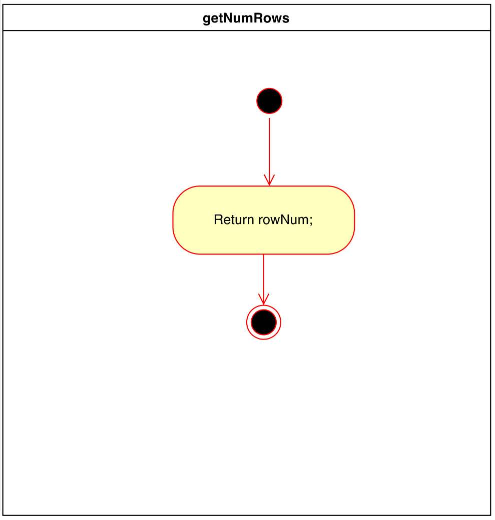

checkHorizWin
pos = game[row] [column];
p = token;
Create a count variable to keep track of consecutive token lined up
Select column 0 <= column <= 8 to place token in
increment count + 1
Does count == 5;
no
yes
Does the token match each other consecutively in the same row?
no
checkHorizWin = false;
yes
Are the tokens still in bounds horizontally?
no
yes
checkHorizWin = true;

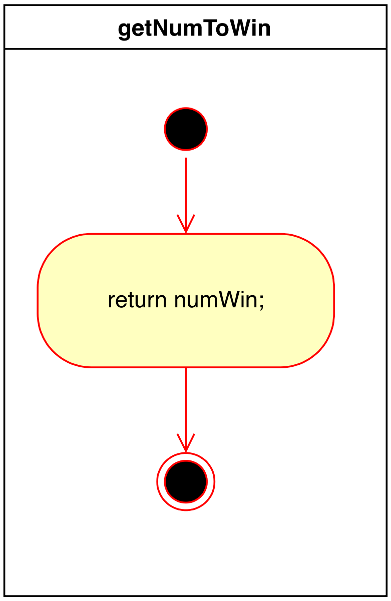

isPlayerAtPos
Go to GameBoard position pos
call whatsAtPos
Is character at pos == player
no
isPlayerAtPos = false;
yes
isPlayerAtPos = true;


toString
Create a new string
Create an iterator for columns
Add a new line to the string
return string
Is the iterator less than or equal to 9?
no
yes
Add the column numbers to the sting with "|" separating them
Add a space then add the column numbers to string with "|" separating them
Add a new line
Initialize the row iterator to 0
Initialize the column iterator to 0
Is there a bottom row?
no
Is there a last column?
no
yes
Is the iterator less than or equal to 9?
no
yes
Add "|" to the string
Add a space, then "|" to the string
Call whatsAtPos for the current position
Add the returned value to the string
Move to the next row and column


checkDiagWin
pos = game[row][column];
p = token;
Create a count variable to keep track of consecutive tokens lined up, (up and to the right) and (down and to the left) (positive slope)
Create a count2 variable to keep track of consecutive tokens lined up, (up and to the left) and (down and to the right) (negative slope)
Select column 0 <= column <= 8 to place token in
Is the last token placed, result in a positive slope win?
no
Is the token placed result in a negative slope win?
no
return false
yes
yes
Increment count.
Increment count.
no
count == 5
count2 == 5
no
yes
yes
no
Are the tokens tracking the positive slope still in bounds?
Are the tokens tracking the positive slope still in bounds?
no
yes
yes
return true
return true

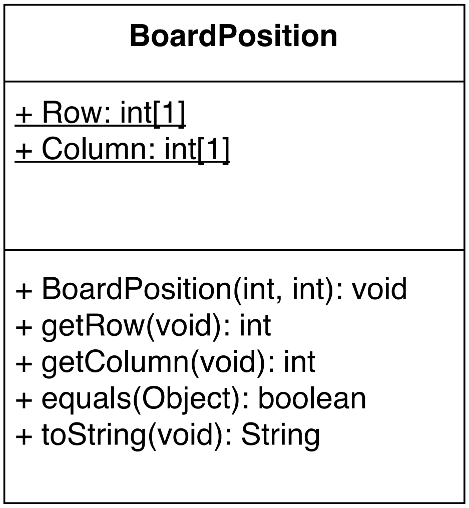

getNumColumns
Return columnNum;


getNumRows
Return rowNum;


getNumToWin
return numWin;

Class 3: BoardPosition

Class diagram:

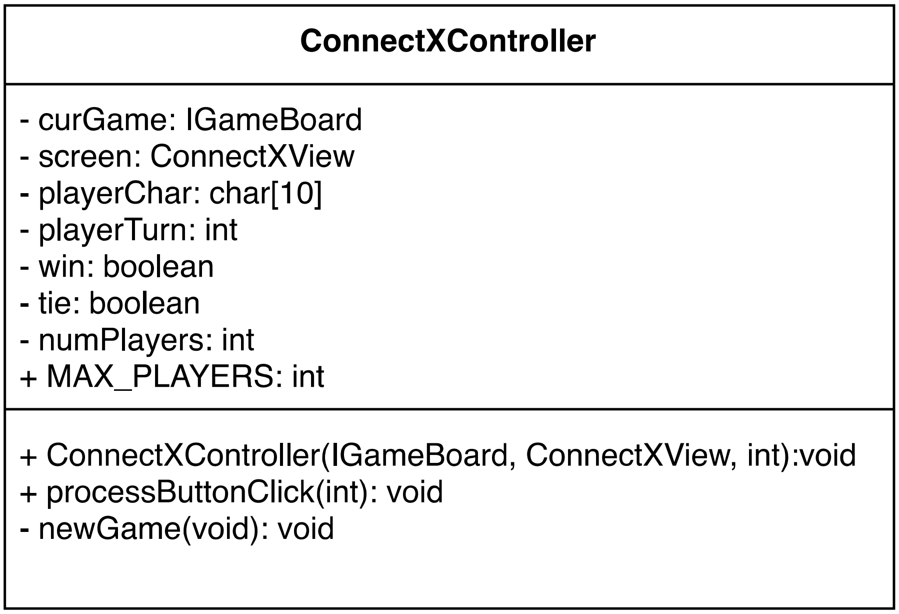


BoardPosition

+ Row: int[1]
+ Column: int[1]

+ BoardPosition(int, int): void
+ getRow(void): int
+ getColumn(void): int
+ equals(Object): boolean
+ toString(void): String

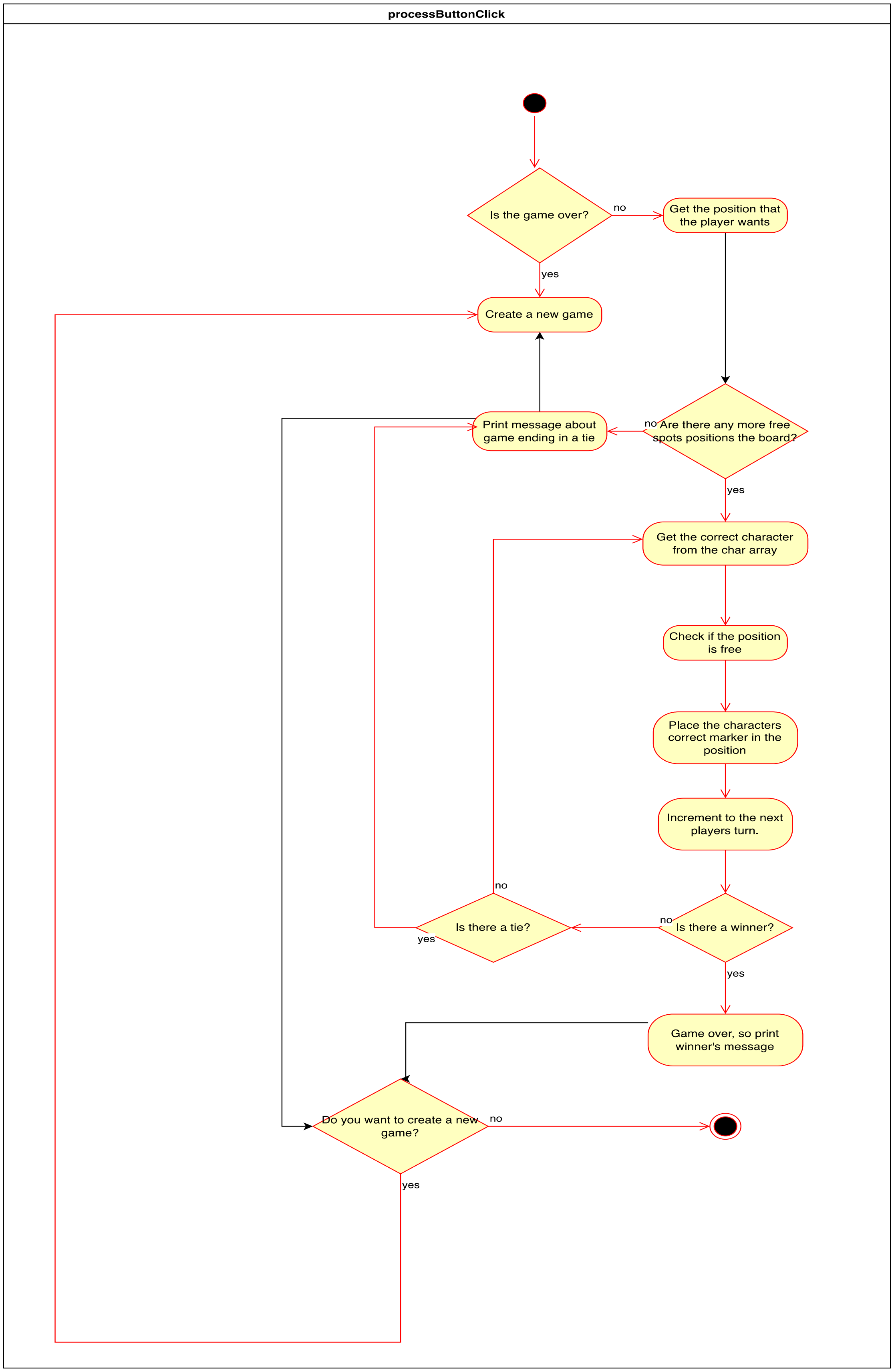
Class 4: ConnectXController



ConnectXController

- curGame: IGameBoard
- screen: ConnectXView
- playerChar: char[10]
- playerTurn: int
- win: boolean
- tie: boolean
- numPlayers: int
+ MAX_PLAYERS: int

+ ConnectXController(IGameBoard, ConnectXView, int):void
+ processButtonClick(int): void
- newGame(void): void


processButtonClick
Is the game over?
no
Get the position that the player wants
yes
Create a new game
no
Are there any more free spots positions the board?
Print message about game ending in a tie
yes
Get the correct character from the char array
Check if the position is free
Place the characters correct marker in the position
Increment to the next players turn.
no
Is there a tie?
yes
no
Is there a winner?
yes
Game over, so print winner's message
Do you want to create a new game?
no
yes

Test Cases

public GameBoard(int row, int column, int numWin)

Input:

State: Uninitialized

Row = 3
Column = 3
numWIn = 3

Output:

State:	**0**	**1**	**2**
2			
1			
0			

Reason:

This test case is unique and distinct because it tests that the board row, column, and numToWin works with the minimum precondition case.

Function Name:

test_constructor_SquareMinimum_BoardSize

Input:

State: Uninitialized

Row = 100
Column = 100
numWIn = 25

Output:

State:	**0**	**1**	**2**	**3**	**…**	**50…**	**99**
99							
50…							
…							
2							
1							
0							

Reason:

This test case is unique and distinct because it tests that the board row, column and numToWin works with the maximum precondition case.

Function Name:

test_constructor_SquareMax_BoardSize

Input:

State: Uninitialized

Row = 50
Column = 50
numWIn = 25

Output:

State:	**0**	**1**	**2**	**…**	**25…**	**49**
49						
48						
25…						
…						
2						
1						
0						

Reason:

This test case is unique and distinct because it tests that the board works a random row input and column input.

Function Name:

test_constructor_Square50x50_BoardSize

boolean checkIfFree(int c)

Input:

State:

	0	1	2	3	4	5	6	7
7								
6								
5								
4								
3								
2								
1								
0								

c = 0;

Output:

checkIfFree = true;

The state of the board is unchanged.

Reason:

This test case is unique and distinct because it tests that the checkIfFree is able to detect that there is not a character at the selected position.

Function Name:

test_checkIfFree_columnEmpty

Input:

State:

	0	1	2	3	4	5	6	7
7								
6	X							
5	X							
4	X							
3	X							
2	X							
1	X							
0	X							

c = 0;

Output:

checkIfFree = true;

The state of the board is unchanged.

Reason:

This test case is unique and distinct because it tests that the checkIfFree is able to detect that the column is not full at the selected number when the column is not full.

Function Name:

test_checkIfFree_column_not_full

Input:

State:

	0	1	2	3	4	5	6	7
7	X							
6	X							
5	X							
4	X							
3	X							
2	X							
1	X							
0	X							

c = 0;

Output:

checkIfFree = false;

The state of the board is unchanged.

Reason:

This test case is unique and distinct because it tests that the checkIfFree is able to detect that there is a character at the selected position when the selected is full.

Function Name:

test_checkIfFree_columnFull

boolean checkHorizWin(BoardPosition pos, char p)

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr>
</table>

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

checkHorizWin = true;

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

This test case is unique and distinct because it tests that the checkHorizWin is true and able to detect character X on the first three rows of the game board. This also test the minimum case of numToWin.

<b>Function Name:</b>

test_checkHorizWin_beginning_row0</td>
</tr>
</table>

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>

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

checkHorizWin = true;

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

This test case is unique and distinct because it tests that the checkHorizWin is true and able to detect character X on the first five rows of the game board. This test case also tests a random numToWin number.

<b>Function Name:</b>

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

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>O</td><td>O</td><td>O</td></tr>
</table>

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

checkHorizWin = false;

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

This test case is unique and distinct because it tests that the checkHorizWin is false for character X even though checkHorizWin for character O returns true.

<b>Function Name:</b>

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

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td></td><td>X</td><td>X</td><td>O</td><td>O</td></tr>
</table>

pos.getRow = 0
pos.getColumn = 4
p = ‘o’</td>
<td><b>Output:</b>

checkHorizWin = false;

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

This test case is unique and distinct because it tests that the checkHorizWin is false at a position that is not at the beginning of the row. Makes sure that the boundary of checkHorizWin is correct.

<b>Function Name:</b>

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

boolean checkVertWin(BoardPosition pos, char p)

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>X</td><td></td><td></td><td></td><td></td></tr>
</table>

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

checkVertWin = true;

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

This test case is unique and distinct because it test that checkVertWin is true at the column 0 after placing 3 X characters in the first column.

<b>Function Name:</b>

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

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td>O</td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>O</td><td></td><td></td><td></td><td></td></tr>
</table>

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

checkVertWin = false;

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

This test case is unique and distinct because it tests that checkVertWin is false at the column 0 after placing both X and O in the first column of the board. Makes sure that the boundary of checkVertWin is correct.

<b>Function Name:</b>

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

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td>O</td><td></td><td></td></tr>
<tr><td>0</td><td></td><td></td><td>O</td><td></td><td></td></tr>
</table>

pos.getRow = 3
pos.getColumn = 0
p = ‘X’</td>
<td><b>Output:</b>
checkVertWin = false;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkVertWin is false with the 2 characters in the middle column to show that checkVertWin is able to work the same in a different column.
<b>Function Name:</b>
test_checkVertWin_false_top_with_spots_below</td>
</tr>
</table>

boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

pos.getRow = 0
pos.getColumn = 0
p = ‘X’</td>
<td><b>Output:</b>
checkDiagWin = false;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkDiagWin is able to return false with no tokens on the board and an empty board.
<b>Function Name:</b>
test_checkDiagWin_false_with_empty board</td>
</tr>
</table>

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td>X</td><td></td><td></td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr>
</table>

pos.getRow = 2
pos.getColumn = 2
p = ‘X’</td>
<td><b>Output:</b>
checkDiagWin = true;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkDiagWin is able to detect character X in a win from left to top right.
<b>Function Name:</b>
test_checkDiagWin_left_to_topRight</td>
</tr>
</table>

Input:

State:

	0	1	2	3	4
4					
3					
2	X				
1	X	X			
0	X	X	X		

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

Output:

checkDiagWin = true;

The state of the board is unchanged.

Reason:

This test case is unique and distinct because it tests that checkDiagWin is able to detect character X in a diagonal win from right to top left.

Function Name:

test_checkDiagWin_right_to_topLeft

Input:

State:

	0	1	2	3	4
4				X	
3			X	X	X
2		X	X	X	X
1	O	O	O	O	O
0	O	O	O	O	O

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

Output:

checkDiagWin = true;

The state of the board is unchanged.

Reason:

This test case is unique and distinct because it shows that checkDiagWin works with X being in the middle and top of the game board.

Function Name:

test_checkDiagWin_bottom_left_to_top_right_filled_under

Input:

State:

	0	1	2	3	4
4					
3					
2					
1		X			
0	X	X			

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

Output:

checkDiagWin = false;

The state of the board is unchanged.

Reason:

This test case is unique and distinct because it tests that checkDiagWin is false when there are not enough tokens to fill the criteria of checkDiagWin top left to bottom right. It shows that the boundaries of checkDiagWin top left to bottom right are valid.

Function Name:

test_checkDiagWin_top_left_to_bottom_right_insufficient_chars

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</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><tr><td>1</td><td></td><td>X</td><td></td><td></td><td></td></tr><tr><td>0</td><td></td><td>X</td><td>X</td><td></td><td></td></tr></table>
pos.getRow = 1
pos.getColumn = 1
p = ‘X’</td>
<td><b>Output:</b>
checkDiagWin = false;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkDiagWin is false when there are not enough tokens to fill the criteria of checkDiagWin top left to bottom right. It shows that the boundaries of checkDiagWin top left to bottom right are valid.
<b>Function Name:</b>
test_checkDiagWin_false_top_left_to_bottom_right_insufficient_chars</td>
</tr>
</table>

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>4</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>3</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr><tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>1</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr><tr><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr></table>
pos.getRow = 2
pos.getColumn = 2
p = ‘X’</td>
<td><b>Output:</b>
checkDiagWin = false;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkDiagWin is false when there is a full tied board. X fills all of the even numbered rows while O fill all of the odd numbered rows which will not result in a diagonal win.
<b>Function Name:</b>
test_checkDiagWin_false_full_tied_board</td>
</tr>
</table>

boolean checkTie()

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>4</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>3</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>0</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;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkTIe recognizes that there are no more empty spaces on the board and that it recognizes the token that is filling up the board.
<b>Function Name:</b>
test_checkTie_true_full_board</td>
</tr>
</table>

<table>
<tr><td><b>Input:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>0</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;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkTIe recognizes that there are no more empty spaces on the board and that it recognizes the token that is filling up the board.
<b>Function Name:</b>
test_checkTie_true_full_board</td></tr>
</table>

<table>
<tr><td><b>Input:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Output:</b>
checkTie = false;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkTIe recognizes that there are empty spaces on the game board.
<b>Function Name:</b>
test_checkTie_false_empty_board</td></tr>
</table>

<table>
<tr><td><b>Input:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr>
</table></td>
<td><b>Output:</b>
checkTie = false;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkTIe is able to recognize that although there are some full columns filled with the character X, empty spaces still exist on the game board and the board has not filled up yet.
<b>Function Name:</b>
test_checkTie_some_full_columns</td></tr>
</table>

<table>
<tr><td><b>Input:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>0</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;
The state of the board is unchanged.</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests that checkTIe recognizes that there are no empty spaces on the game board with more than one token on the board. X is placed in the even numbered rows and O is placed in the odd numbered rows.
<b>Function Name:</b>
test_checkTie_full_alternating_board</td></tr>
</table>

char whatsAtPos(BoardPosition pos)

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

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

whatsAtPos == ‘ ‘;

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

This test case is unique and distinct because it tests that whatsAtPos recognizes that there is an empty space at the current position of the board.

<b>Function Name:</b>

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

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>

pos.getRow = 1
pos.getColumn = 0</td>
<td><b>Output:</b>

whatsAtPos == ‘ ‘;

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

This test case is unique and distinct because it tests that whatsAtPos correctly traverses through the game board to get to the correct location. The bottom row of the game board is filled with X.

<b>Function Name:</b>

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

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>

pos.getRow = 4
pos.getColumn = 4</td>
<td><b>Output:</b>

whatsAtPos = ‘ ‘;

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

This test case is unique and distinct because it tests that whatsAtPos correctly identifies what is at the last row and column of the game board with the whole board except for the last row position and column position filled.

<b>Function Name:</b>

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

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

State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</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><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0</td><td></td><td></td><td></td><td>X</td><td></td></tr></table>
pos.getRow = 0
pos.getColumn = 3</td>
<td><b>Output:</b>

whatsAtPos = ‘X‘;

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

This test case is unique and distinct because it tests that whatsAtPos recognizes that there is only one character on the board and is able to recognize the correct placement of the character token and return the correct character token present.

<b>Function Name:</b>

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

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

State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>0</td><td>O</td><td></td><td></td><td></td><td></td></tr></table>
pos.getRow = 1
pos.getColumn = 0</td>
<td><b>Output:</b>

whatsAtPos = ‘X‘;

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

This test case is unique and distinct because it tests that whatsAtPos recognizes that there is an empty space at the current position of the board.

<b>Function Name:</b>

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

boolean isPlayerAtPos(BoardPosition pos, char player)

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

State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</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><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr></table>
pos.getRow = 0
pos.getColumn = 0
player = X</td>
<td><b>Output:</b>

IsPlayerAtPos = false;

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

This test case is unique and distinct because it tests that isPlayerAtPos recognizes that there is not a token at the current board position of the board. The game board is empty.

<b>Function Name:</b>

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

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td>O</td></tr>
</table>

pos.getRow = 0
pos.getColumn = 4
player = O</td>
<td><b>Output:</b>

IsPlayerAtPos = true;

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

This test case is unique and distinct because it tests that isPlayerAtPos recognizes that there is a token at the current board position of the board. isPlayerAtPos has one valid position on the board where it can return true.

<b>Function Name:</b>

test_isPlayerAtPos_one_char_on _board</td>
</tr>
</table>

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>

pos.getRow = 0
pos.getColumn = 1
player = X</td>
<td><b>Output:</b>

IsPlayerAtPos = true;

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

This test case is unique and distinct because it tests that isPlayerAtPos recognizes that there is a token at the current board position of the board. With the first row of the game board filled, isPlayerAtPos is able to recognize that the current position of the game board is filled with a token.

<b>Function Name:</b>

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

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>

pos.getRow = 4
pos.getColumn = 4
player = X</td>
<td><b>Output:</b>

IsPlayerAtPos = false;

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

This test case is unique and distinct because it tests that isPlayerAtPos recognizes that there is a token at the current board position of the board. With every position on the board filled with a token, isPlayerAtPos is able to recognize the only character on the board that is not filled with token O which makes isPlayerAtPos return false.

<b>Function Name:</b>

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

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>

pos.getRow = 4
pos.getColumn = 4
player = X</td>
<td><b>Output:</b>

IsPlayerAtPos = true;

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

This test case is unique and distinct because it tests that isPlayerAtPos recognizes that there is a token at the current board position of the board when the game board is completely full. The token X fills the whole game board.

<b>Function Name:</b>

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

void placeToken(char p, int c)

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

p = ‘X’
c = 0</td>
<td><b>Output:</b>

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>X</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 that placeToken I am placing token X in an empty board, and after placing the token in column 0, it was the only token that was in the board.

<b>Function Name:</b>

test_placeToken_on _empty_board</td>
</tr>
</table>

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

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>O</td><td></td><td></td><td></td><td></td></tr>
</table>

p = ‘X’
c = 0</td>
<td><b>Output:</b>

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</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>
<tr><td>1</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>O</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 that placeToken I am placing token X in a column that is already filled with a token, and also the column was not close to being full.

<b>Function Name:</b>

test_placeToken_in _partly_filled_column</td>
</tr>
</table>

Input:

State:

	0	1	2	3	4
4					
3	O				
2	O				
1	O				
0	O				

p = ‘X’
c = 0

Output:

State:

	0	1	2	3	4
4	X				
3	O				
2	O				
1	O				
0	O				

Reason:

This test case is unique and distinct because it tests that placeToken I am placing token X in a column that is almsot full, and also after placing the token in column 0, it filled up the column.

Function Name:

test_placeToken_fill _up_column

Input:

State:

	0	1	2	3	4
4					
3					
2					
1					
0	O	O	O	O	

p = ‘X’
c = 4

Output:

State:

	0	1	2	3	4
4					
3					
2					
1					
0	O	O	O	O	X

Reason:

This test case is unique and distinct because it tests that placeToken I am placing token X in a row that is almsot full, and also after placing the token in row 4, it fills up the row.

Function Name:

test_placeToken_fill _up_row

Input:

State:

	0	1	2	3	4
4	O	O	O	O	
3	O	O	O	O	O
2	O	O	O	O	O
1	O	O	O	O	O
0	O	O	O	O	O

p = ‘X’
c = 4

Output:

State:

	0	1	2	3	4
4	O	O	O	O	X
3	O	O	O	O	O
2	O	O	O	O	O
1	O	O	O	O	O
0	O	O	O	O	O

Reason:

This test case is unique and distinct because it tests that placeToken I am placing token X in a board that is almost full, and after placing the token in column 4, the entire board will be full.

Function Name:

test_placeToken_to _fill_board