

# CPSC 2150 Project Report

Joel Miller

## Requirements Analysis

### Functional Requirements:

1. Enumerated list of functional requirements
2. **PLAYER**
  - 1) As a game player, I expect to provide the number of rows, columns, and tokens in a row needed to win the game at the start of each new game. The maximum number of rows and columns for all GameBoards should be 100, and the maximum number needed to win will be 25 for all GameBoards. The minimum number of rows, columns, and tokens in a row needed to win are all 3.
  - 2) As a game player, I should not be able to provide a "number of tokens needed to win" larger than the number of rows or the number of columns.
  - 3) As a game player, I expect the formatting of the displayed board to allow double-digit numbers.
  - 4) As a game player, if I decide to play again, I should be able to change the size and number of tokens needed to win settings from the previous game.
  - 5) As a game player, I expect the program to validate the number of rows, columns, and tokens needed to win that I provide.
  - 6) As a game player, I expect to be able to specify the number of players for each game at the start of each new game. I must enter at least 2 players and at most 10 players.
  - 7) As a game player, I assume that the game will pick the characters for the game [XOABCDEFGH] since there are a max of 10 players
  - 8) As a game player, I should be able to change the number of players and their characters if I choose to start a new game.
  - 9) As a game player, I expect there to be a GUI where the player can choose the settings of the game and interact easily with the game

### Non-Functional Requirements

- Enumerated list of Non-functional requirements
1. The game code must be written in Java.
  2. The game must be compatible with a command line interface.
  3. The game must have three classes: GameScreen.java, BoardPosition.java, and GameBoard.java.
    - **GameScreen.java:**
      - Must contain the main method.
      - This class will interface with the player{It will alternate the game between players, say whose turn it is, get the column they would like, and place their marker}.
    - **BoardPosition.java:**
      - Will keep track of the row position and column position.

- Only 1 constructor which takes in an int for row and an int for column.
- Must also have some getter functions getColumn and getrow.
- **GameBoard.java:**
  - All attributes must be private.
  - Functions must be made public.
  - Each position will have a blank character.
  - The constructors for the game boards should take parameters in the order of rows, columns, and the number of tokens needed to win.
  - The maximum number of rows and columns for all GameBoards is 100, and the maximum number needed to win will be 25 for all GameBoards.
  - The minimum number of rows, columns, and tokens in a row needed to win are all 3.
  - GameBoard.java must still use a 2D array to represent its game board, but it's ok if most of the array is ignored for smaller boards.
  - The user should not be able to provide a "number of tokens needed to win" larger than the number of rows or the number of columns.
  - The program must validate the number of rows, columns, and tokens needed to win that is provided by the user.
  - The size and number of tokens needed to win settings can be changed if the users decide to play again.
  - At the start of each new game, the users will be able to specify the number of players for each game. They must enter at least 2 players and, at most, 10 players.
  - The number of players can change if they choose to start a new game.
  - Our game must now have two different implementations of our IGameBoard interface.

4. **GameBoardMem** class should be created, which will extend AbsGameBoard and implement the IGameBoard interface for memory-efficient implementation.
  - This new implementation will be slower but more memory efficient.
  - To represent the game board, a Map will be used. The key field of the Map will be a Character that represents the player so that each player will get



## BoardPosition

-Row:int

-Column:int

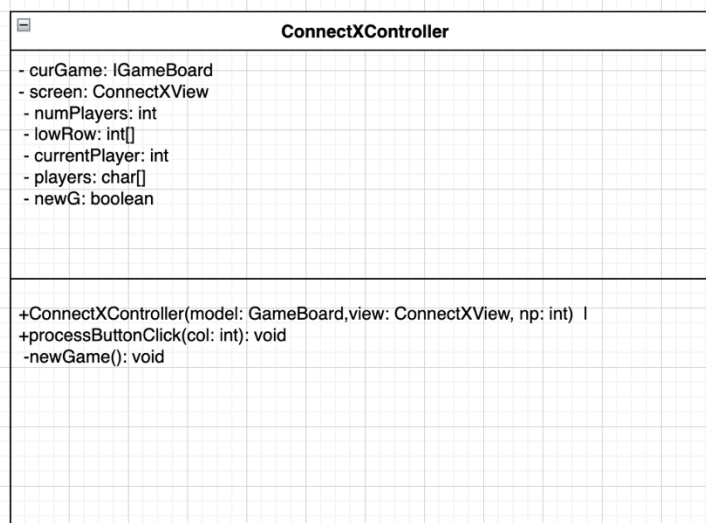
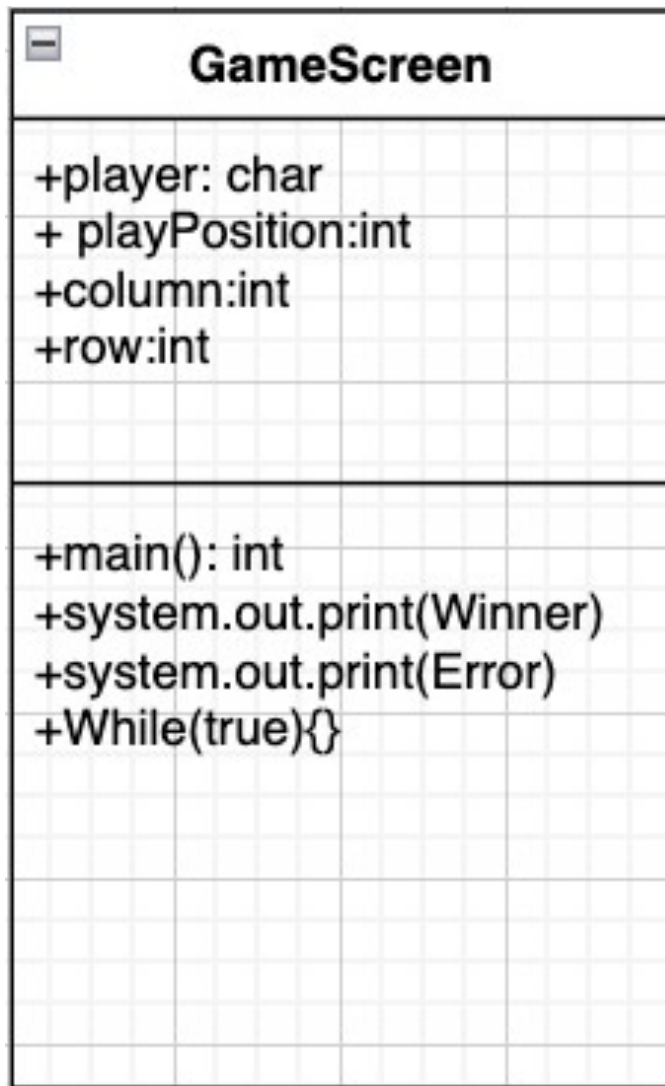
+BoardPosition(int ro, int colu)

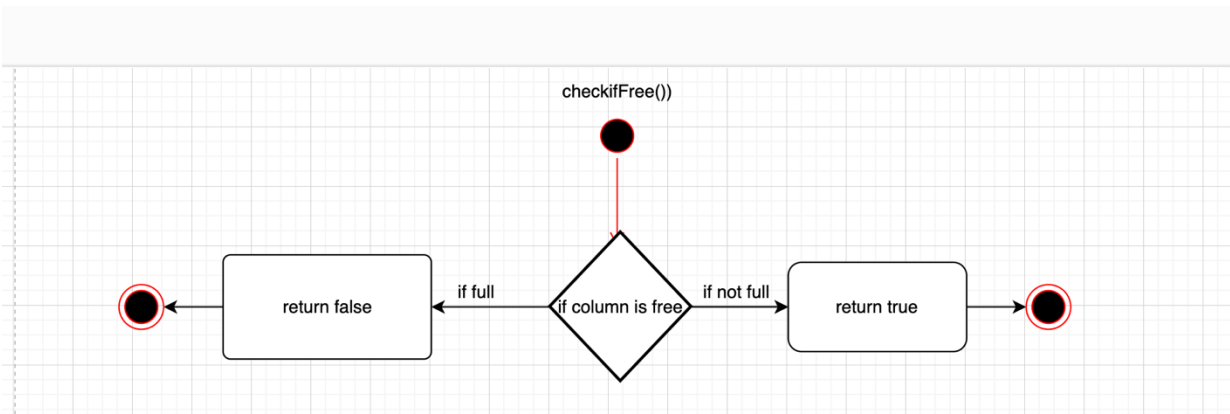
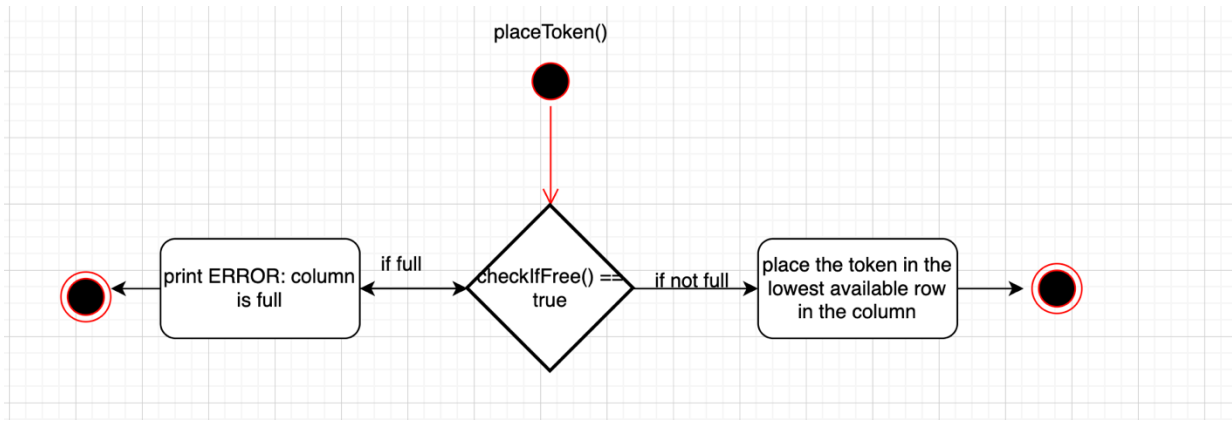
+getRow():int

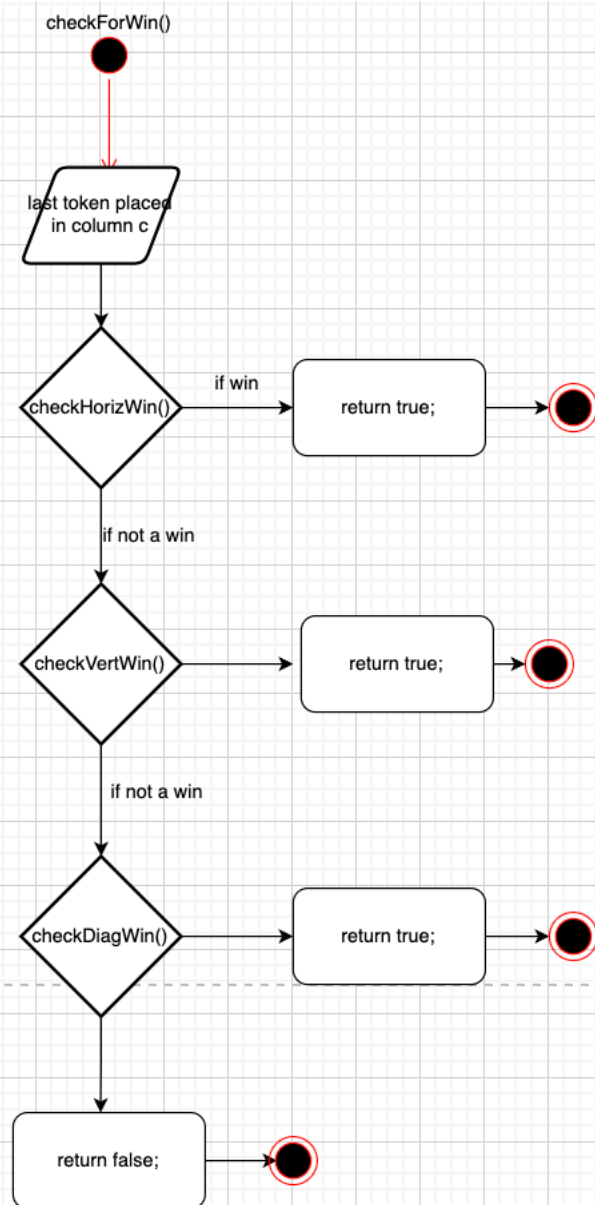
+getColumn():int

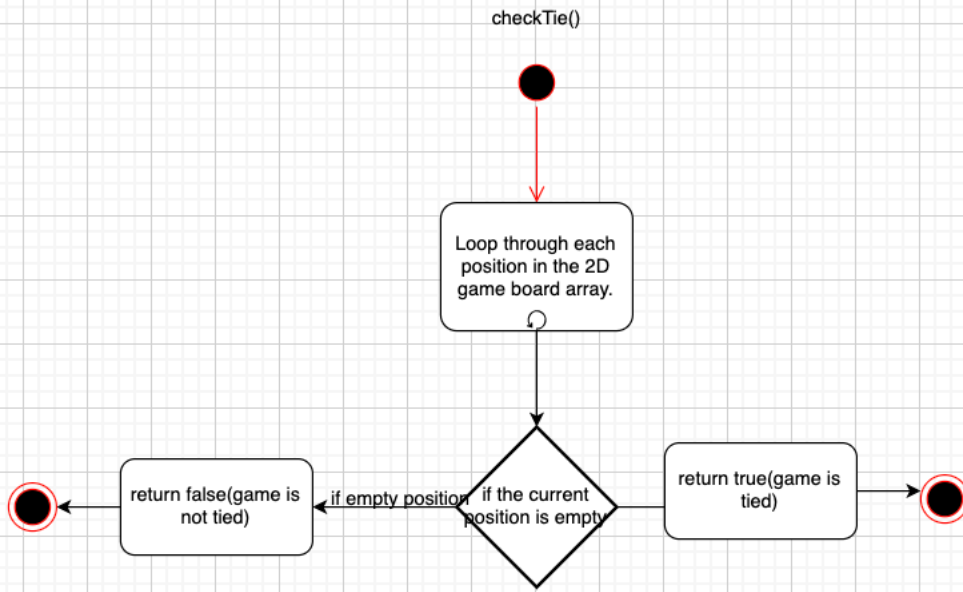
+equals() override

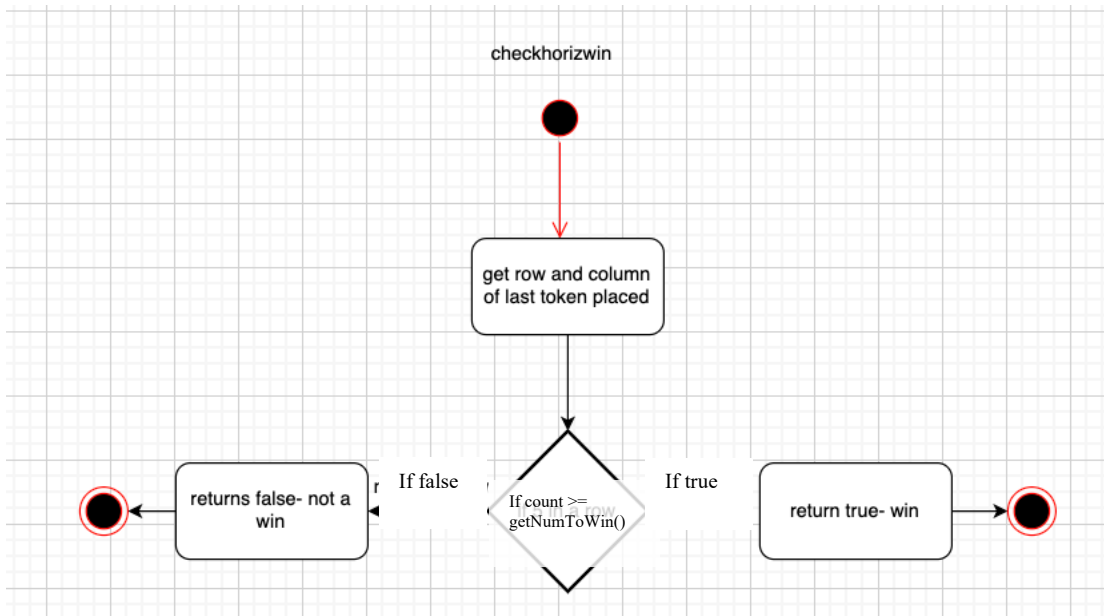
+toString() override



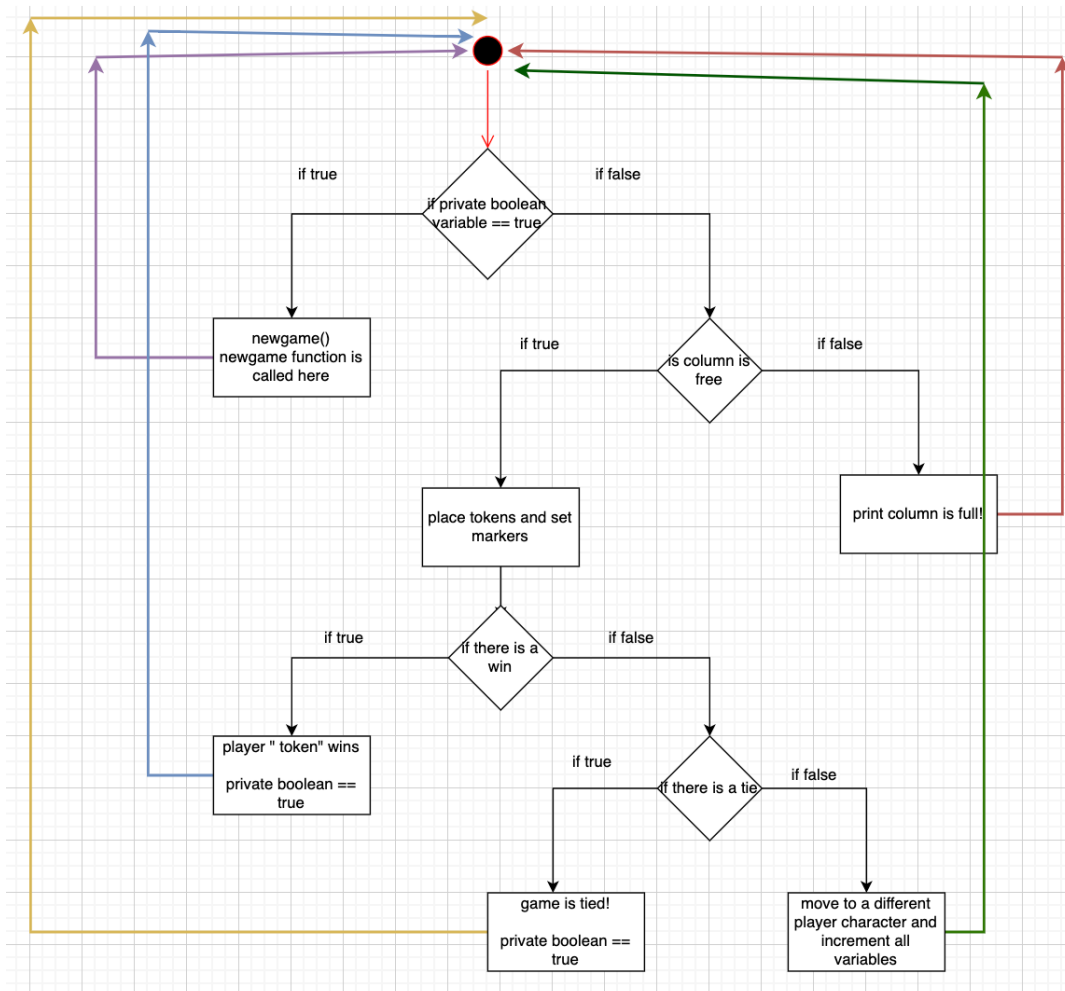




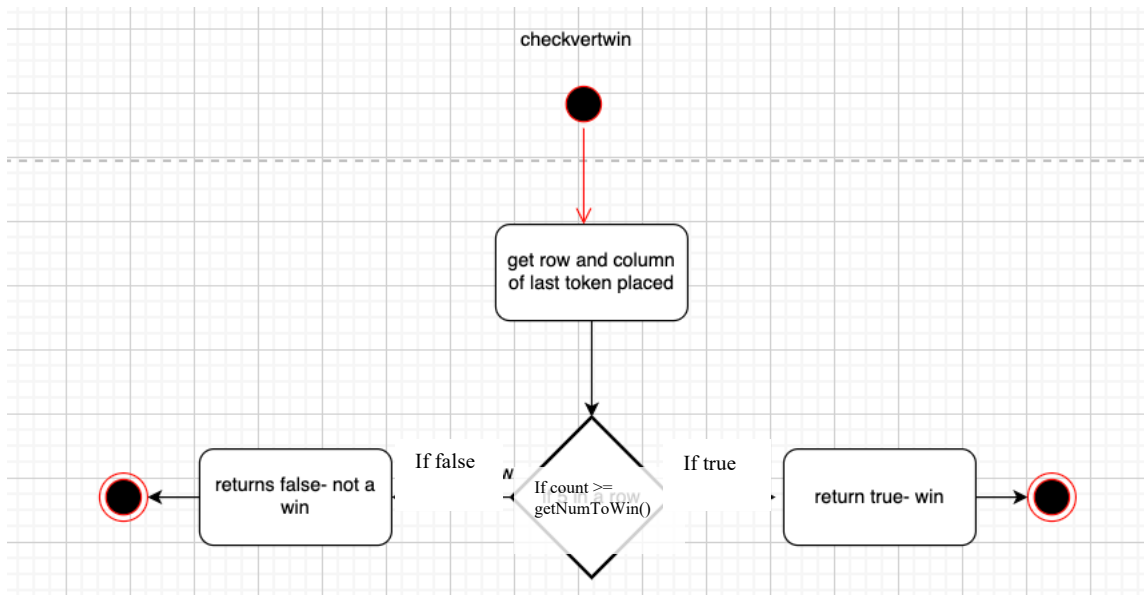




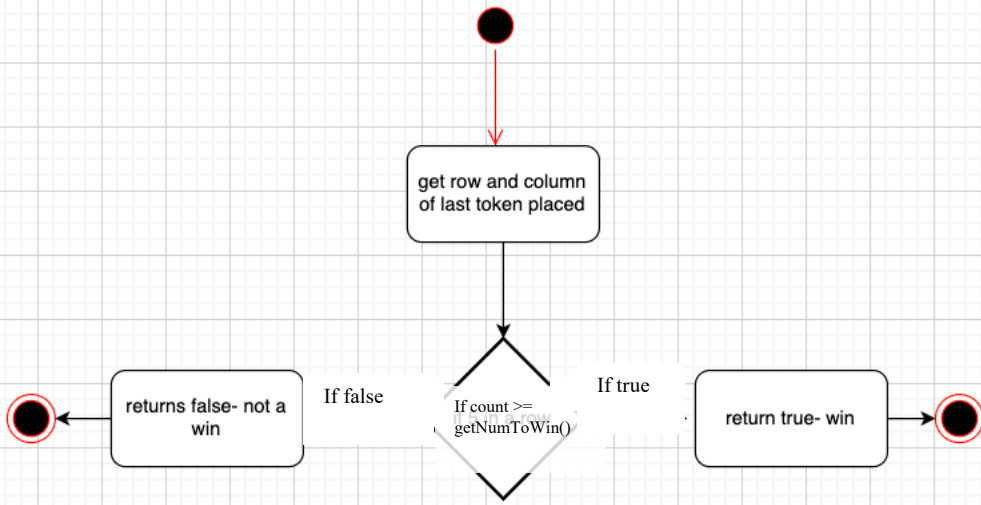
### Processbuttonclick(int column)

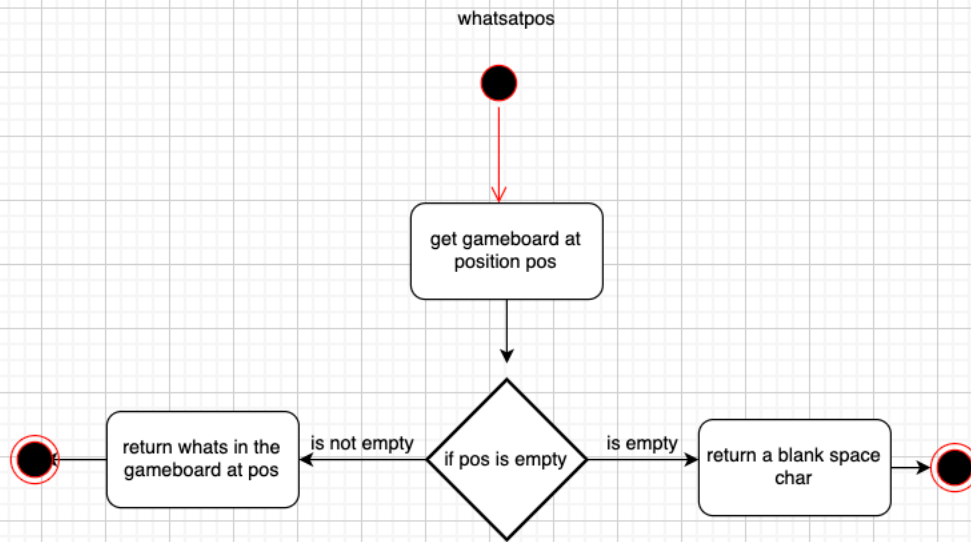


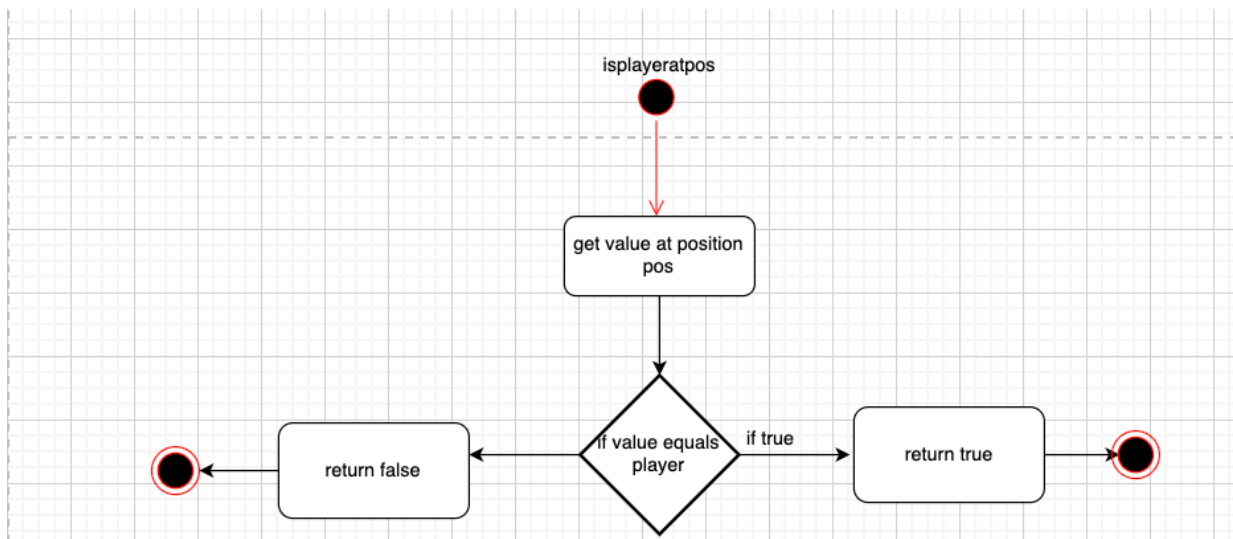




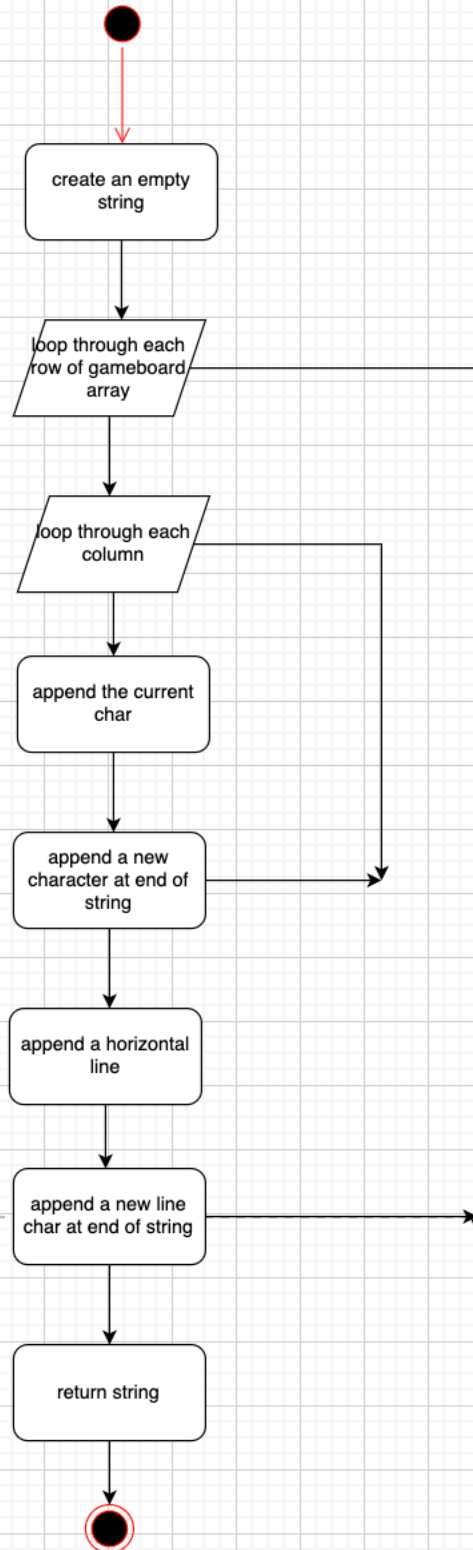
checkforwin

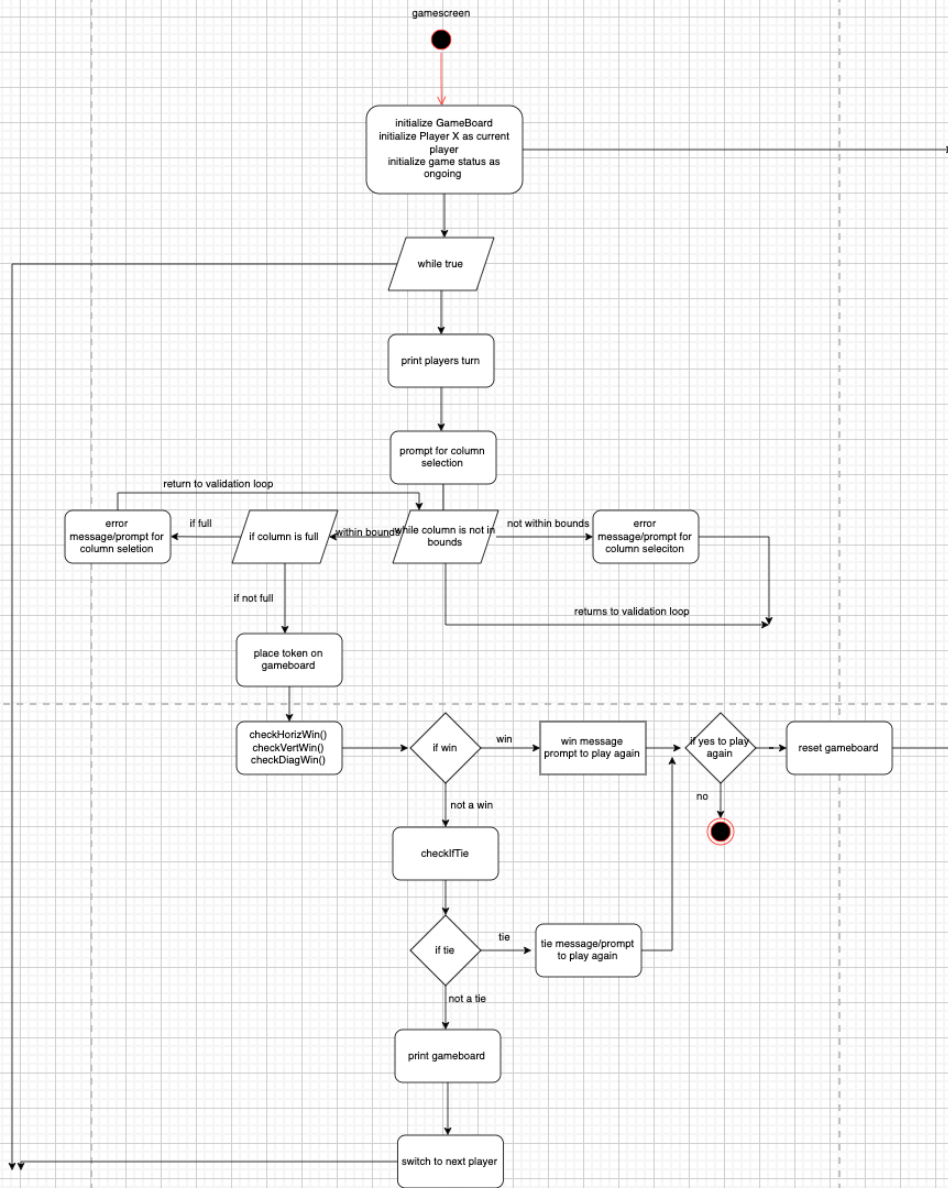






AbsGameBoard

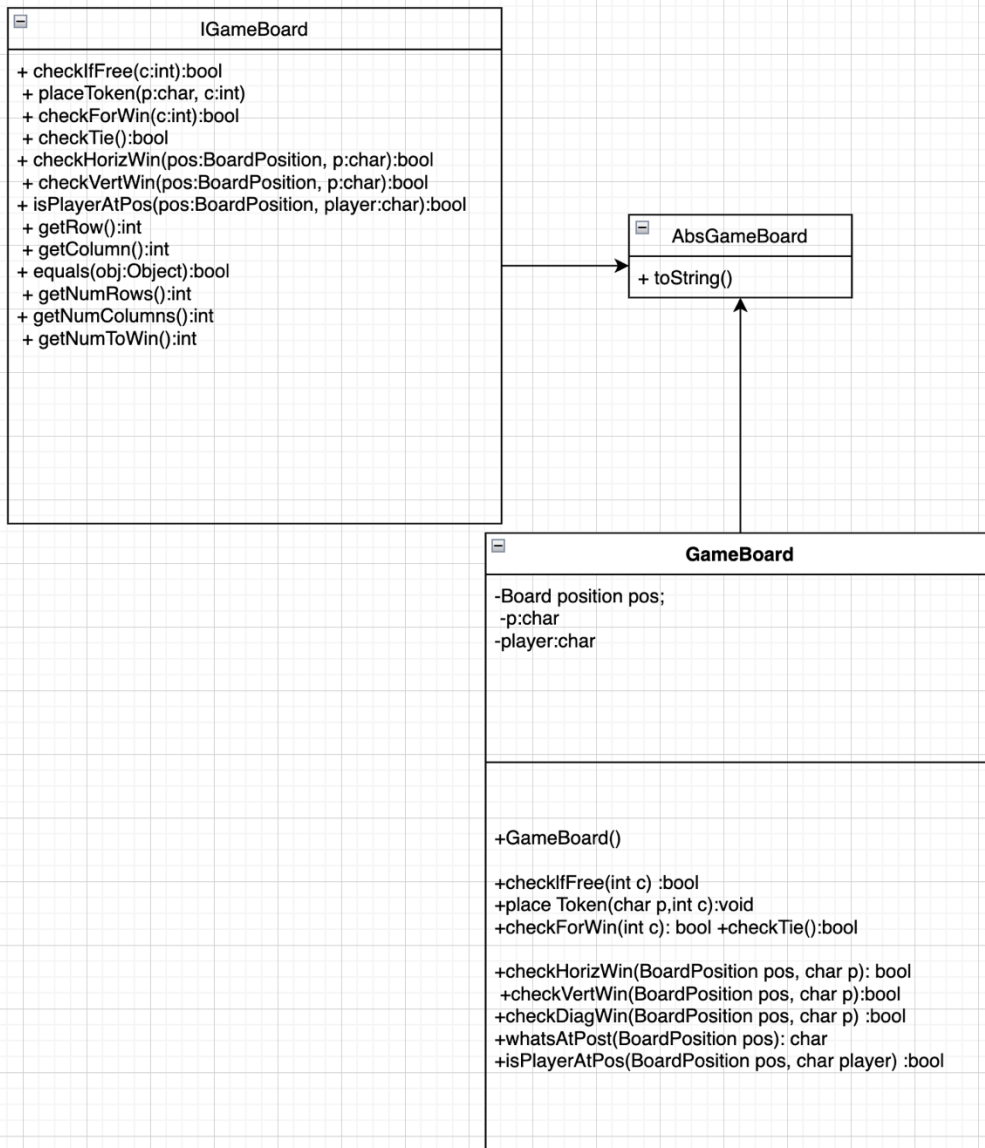




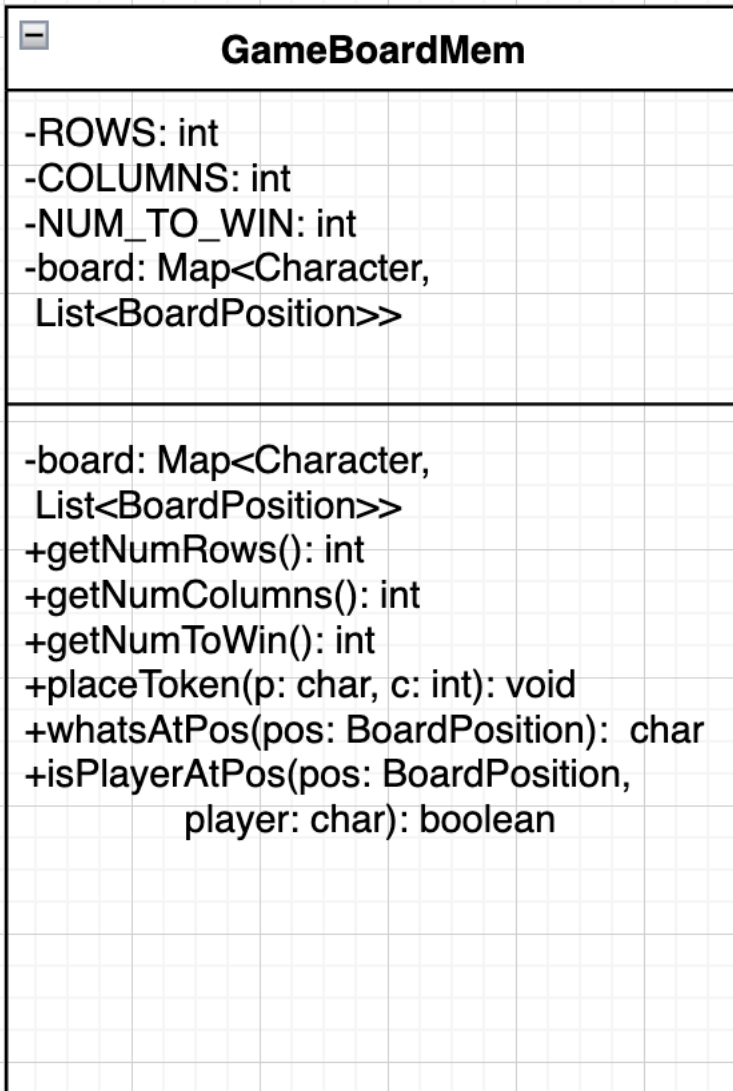


## IGameBoard

- + checkIfFree(c:int):bool
- + placeToken(p:char, c:int)
- + checkForWin(c:int):bool
- + checkTie():bool
- + checkHorizWin(pos:BoardPosition, p:char):bool
- + checkVertWin(pos:BoardPosition, p:char):bool
- + isPlayerAtPos(pos:BoardPosition, player:char):bool
- + getRow():int
- + getColumn():int
- + equals(obj:Object):bool
- + getNumRows():int
- + getNumColumns():int
- + getNumToWin():int







## Constructor GameBoard() GameboardMem()

<b>Input:</b>  <b>numRows = 3</b> <b>numColumns = 3</b> <b>numToWin = 3</b>	<b>Output:</b>  <b>State(number to win = 3)</b> <table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>  <b>Function Name:</b> testConstructor_01										<b>Reason:</b>  This test case is unique because it tests the constructor for creating a standard 3x3 board with a winning condition of 3 tokens in a row.

<b>Input:</b>  <b>numRows = 4</b> <b>numColumns = 4</b> <b>numToWin = 4</b>	<b>Output:</b>  <b>State(number to win = 4)</b> <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr></table>  <b>Function Name:</b> testConstructor_02																	<b>Reason:</b>  This test case is unique because it tests the constructor for creating a 4x4 game board with a winning condition of 4 tokens in a row. This tests the constructor's ability to handle different board sizes.

<b>Input:</b>  numRows = 5 numColumns = 5 numToWin = 3	<b>Output:</b>  State(number to win = 3) <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>  Function Name: testConstructor_03																										<b>Reason:</b>  This test case is unique because it tests the constructor for creating a 5x5 game board with a winning condition of only 3 tokens in a row. This tests the constructor's ability to handle different winning conditions in relation to the board size.

`boolean checkIfFree(int c)`

<b>Input:</b>  <b>State(number to win = 4)</b> <table><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>o</td><td>x</td><td>o</td><td>o</td></tr><tr><td>x</td><td>x</td><td>x</td><td>o</td></tr><tr><td>o</td><td>o</td><td>x</td><td>x</td></tr></table> c = 0;	o	x	o	x	o	x	o	o	x	x	x	o	o	o	x	x	<b>Output:</b>  checkIfFree = false  state of the board is unchanged  <b>Function Name:</b> testCheckIfFree_01	<b>Reason:</b>  This test case is unique because the game board is completely full, and the function should check if there is any available space in the specified column (0). Since the column is full, the function should return false.
o	x	o	x															
o	x	o	o															
x	x	x	o															
o	o	x	x															

<b>Input:</b>  <b>State(number to win = 4)</b> <table><tr><td></td><td><b>x</b></td><td><b>o</b></td><td><b>x</b></td></tr><tr><td><b>x</b></td><td><b>x</b></td><td><b>o</b></td><td><b>o</b></td></tr><tr><td>x</td><td>x</td><td>x</td><td>o</td></tr><tr><td>o</td><td>o</td><td>x</td><td>x</td></tr></table> c = 1;		<b>x</b>	<b>o</b>	<b>x</b>	<b>x</b>	<b>x</b>	<b>o</b>	<b>o</b>	x	x	x	o	o	o	x	x	<b>Output:</b>  checkIfFree = true  state of the board is unchanged  <b>Function Name:</b> testCheckIfFree_02	<b>Reason:</b>  This test case is unique because the game board has an empty spot in the specified column (1). The function should check if there is any available space in that column. Since the column has an empty spot, the function should return true.
	<b>x</b>	<b>o</b>	<b>x</b>															
<b>x</b>	<b>x</b>	<b>o</b>	<b>o</b>															
x	x	x	o															
o	o	x	x															

<b>Input:</b>  <b>State(number to win = 4)</b> <table><tr><td></td><td><b>x</b></td><td><b>o</b></td><td><b>x</b></td></tr><tr><td></td><td><b>x</b></td><td><b>o</b></td><td><b>o</b></td></tr><tr><td><b>x</b></td><td><b>x</b></td><td><b>x</b></td><td><b>o</b></td></tr><tr><td><b>o</b></td><td><b>o</b></td><td><b>x</b></td><td><b>x</b></td></tr></table> c = 0;		<b>x</b>	<b>o</b>	<b>x</b>		<b>x</b>	<b>o</b>	<b>o</b>	<b>x</b>	<b>x</b>	<b>x</b>	<b>o</b>	<b>o</b>	<b>o</b>	<b>x</b>	<b>x</b>	<b>Output:</b>  checkIfFree = true  state of the board is unchanged  <b>Function Name:</b> testCheckIfFree_03	<b>Reason:</b>  This test case is unique because the game board has a partially filled specified column (0) with only one 'x' marker. The function should check if there is any available space in that column. Since the column has empty spots, the function should return true.
	<b>x</b>	<b>o</b>	<b>x</b>															
	<b>x</b>	<b>o</b>	<b>o</b>															
<b>x</b>	<b>x</b>	<b>x</b>	<b>o</b>															
<b>o</b>	<b>o</b>	<b>x</b>	<b>x</b>															

**boolean checkHorizWin(BoardPosition pos, char p)**

<b>Input:</b>  <b>State:(number to win = 3)</b> <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td>x</td><td>x</td><td>x</td><td></td></tr><tr><td>o</td><td>o</td><td>x</td><td>x</td></tr></table>  pos.getRow = 1 pos.getCol = 2 p = 'x'									x	x	x		o	o	x	x	<b>Output:</b>  checkHorizWin = true  state of the board is unchanged  <b>Function Name:</b> testCheckHorizWin_01	<b>Reason:</b> This test case is unique and distinct because the last x was placed at the end of the string so the function needs to be able to count the x's to the left
x	x	x																
o	o	x	x															

<b>Input:</b>  <b>State:(number to win = 4)</b> <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td>o</td><td>o</td><td>o</td></tr><tr><td>x</td><td>x</td><td>x</td><td>o</td></tr></table> pos.getRow = 1 pos.getCol = 2 p = 'o'									o	o	o	o	x	x	x	o	<b>Output:</b>  checkHorizWin = true  state of the board is unchanged  <b>Function Name:</b> testCheckHorizWin_02	<b>Reason:</b> This test case is unique and distinct because the last o was placed in the middle of the string of 4 consecutive o's as opposed to on the end, so the function needs to count o's on the right and left
o	o	o	o															
x	x	x	o															

<b>Input:</b>  <b>State:(number to win = 4)</b> <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>o</td><td>o</td><td>o</td><td></td></tr><tr><td>x</td><td>x</td><td>x</td><td>o</td><td>x</td></tr></table> <p>pos.getRow = 0 pos.getCol = 4 p = 'x'</p>												o	o	o		x	x	x	o	x	<b>Output:</b>  checkHorizWin = false  state of the board is unchanged  <b>Function Name:</b> testCheckHorizWin_03	<b>Reason:</b> This test case is unique because the 'x' has tokens to the left, but they are not consecutive. The function needs to count the number of consecutive 'x's to determine if there is a win condition. In this case, there are only three consecutive 'x's and o's, so there is no win condition.
	o	o	o																			
x	x	x	o	x																		

<b>Input:</b>  <b>State:(number to win = 4)</b> <table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td></td><td>o</td><td></td><td></td><td></td></tr><tr><td>o</td><td>o</td><td>x</td><td>x</td><td>x</td><td>x</td></tr></table> pos.getRow = 0 pos.getCol = 3 p = 'x'																									o		o				o	o	x	x	x	x	<b>Output:</b>  checkHorizWin = true  state of the board is unchanged   <b>Function Name:</b> testCheckHorizWin_04	<b>Reason:</b> This test case is unique because the 'x' just placed is in the middle of a string of four consecutive 'x's, and the function needs to check for a win condition by counting the number of 'x's to the left and right of the newly placed 'x'.
o		o																																				
o	o	x	x	x	x																																	

**boolean checkVertWin(BoardPosition pos, char p)**

<b>Input:</b>  <b>State:(number to win = 3)</b>  <table border="1"> <tr><td></td><td>x</td><td></td></tr> <tr><td>o</td><td>x</td><td></td></tr> <tr><td>o</td><td>x</td><td></td></tr> </table> pos.getRow = 2 pos.getCol = 1 p = 'x'		x		o	x		o	x		<b>Output:</b>  checkVertWin = true  state of the board is unchanged   <b>Function Name:</b> testCheckVertWin_01	<b>Reason:</b> This test case is unique because the 'x' just placed is part of a string of three consecutive 'x's in the second column from the left, and the function needs to count the number of 'x's below to determine a win condition. Since there are three 'x's in a row vertically, <b>checkVertWin</b> should return <b>true</b> for a win condition.
	x										
o	x										
o	x										

<b>Input:</b>  <b>State:(number to win = 5)</b>  <table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>x</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>x</td><td></td></tr><tr><td>s</td><td>s</td><td>f</td><td>o</td><td>o</td><td>x</td><td></td></tr><tr><td>o</td><td>f</td><td>f</td><td>f</td><td>s</td><td>x</td><td></td></tr><tr><td>o</td><td>f</td><td>o</td><td>s</td><td>s</td><td>x</td><td>o</td></tr></table>  pos.getRow = 4 pos.getCol = 5 p = 'x'													x							x		s	s	f	o	o	x		o	f	f	f	s	x		o	f	o	s	s	x	o	<b>Output:</b>  checkVertWin = true  state of the board is unchanged    <b>Function Name:</b> testCheckVertWin_02	<b>Reason:</b> This test case is unique because the 'x' just placed is part of a string of five consecutive 'x's in a column, and the function needs to count the number of consecutive 'x's to determine a win condition. In this case, there are five consecutive 'x's so checkVertWin should return true.
					x																																							
					x																																							
s	s	f	o	o	x																																							
o	f	f	f	s	x																																							
o	f	o	s	s	x	o																																						

<b>Input:</b>  <b>State:(number to win = 4)</b> <table><tr><td></td><td></td><td><b>o</b></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td><b>x</b></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td><b>x</b></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td><b>o</b></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td><b>o</b></td><td><b>x</b></td><td></td><td></td></tr><tr><td></td><td></td><td><b>o</b></td><td><b>x</b></td><td></td><td></td></tr></table> <p>pos.getRow = 5 pos.getCol = 2 p = 'o'</p>			<b>o</b>						<b>x</b>						<b>x</b>						<b>o</b>						<b>o</b>	<b>x</b>					<b>o</b>	<b>x</b>			<b>Output:</b>  checkVertWin = false  state of the board is unchanged      <b>Function Name:</b> testCheckVertWin_03	<b>Reason:</b> This test case is unique because the 'o' just placed is not part of a string of four consecutive 'o's, and the function should return <b>false</b> for a win condition. Although there are four 'o's in the column, they are not in a consecutive pattern, so <b>checkVertWin</b> should return <b>false</b> .
		<b>o</b>																																				
		<b>x</b>																																				
		<b>x</b>																																				
		<b>o</b>																																				
		<b>o</b>	<b>x</b>																																			
		<b>o</b>	<b>x</b>																																			

<b>Input:</b>  <b>State(number to win = 5)</b>  <table><tr><td></td><td></td><td></td><td></td><td></td><td><b>x</b></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td><b>o</b></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td><b>x</b></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td><b>o</b></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td><b>x</b></td><td></td><td></td><td></td><td></td></tr></table>  pos.getRow = 4 pos.getCol = 5 p = 'x'						<b>x</b>										<b>o</b>										<b>x</b>										<b>o</b>										<b>x</b>					<b>Output:</b>  checkVertWin = false  state of the board is unchanged   <b>Function Name:</b> testCheckVertWin_04	<b>Reason:</b> This test case is unique because the 'x' just placed is at the top of a string of five consecutive tokens and the function needs to check for a win condition by counting the number of 'x's. Since there are no 5 consecutive tokens checkVertWin should return false.
					<b>x</b>																																															
					<b>o</b>																																															
					<b>x</b>																																															
					<b>o</b>																																															
					<b>x</b>																																															

```
public default boolean checkDiagWin(BoardPosition pos, char p)
```

<b>Input:</b>  <b>State(number to win = 3)</b> <table border="1"> <tr><td><b>x</b></td><td></td><td></td></tr> <tr><td><b>o</b></td><td><b>x</b></td><td></td></tr> <tr><td><b>o</b></td><td><b>o</b></td><td><b>x</b></td></tr> </table> pos.getRow = 3 pos.getCol = 1 p = 'x'	<b>x</b>			<b>o</b>	<b>x</b>		<b>o</b>	<b>o</b>	<b>x</b>	<b>Output:</b>  checkDiagWin = true  state of the board is unchanged   <b>Function Name:</b> testCheckDiagWin_01	<b>Reason:</b> This test case is unique because the 'x' just placed is in the middle of a string of three consecutive 'x's in a diagonal direction. The function needs to check for a win condition by counting the number of 'x's in both diagonal directions (top-left to bottom-right and top-right to bottom-left). Since there are three consecutive 'x's in the top-left to bottom-right diagonal direction, <b>checkDiagWin</b> should <b>return true</b> .
<b>x</b>											
<b>o</b>	<b>x</b>										
<b>o</b>	<b>o</b>	<b>x</b>									

<b>Input:</b>  <b>State(number to win = 5)</b>  <table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>x</td></tr><tr><td></td><td></td><td></td><td>x</td><td>x</td></tr><tr><td></td><td></td><td>x</td><td>o</td><td>o</td></tr><tr><td></td><td>x</td><td>o</td><td>o</td><td>x</td></tr><tr><td>x</td><td>o</td><td>o</td><td>o</td><td>x</td></tr></table> pos.getRow = 2 pos.getCol = 2 p = 'x'										x				x	x			x	o	o		x	o	o	x	x	o	o	o	x	<b>Output:</b>  checkDiagWin = true  state of the board is unchanged   <b>Function Name:</b> testCheckDiagWin_02	<b>Reason:</b> This test case is unique because the 'x' just placed is in the middle of a diagonal string of five consecutive tokens, and the function needs to check for a win condition by counting the number of 'x's in both diagonal directions. There are five consecutive 'x's in the top-left to bottom-right diagonal direction, so checkDiagWin should return true
				x																												
			x	x																												
		x	o	o																												
	x	o	o	x																												
x	o	o	o	x																												

<b>Input:</b>  <b>State(number to win = 3)</b>  <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>x</td><td></td></tr><tr><td></td><td>x</td><td>o</td><td></td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr></table>  <p>pos.getRow = 0 pos.getCol = 0 p = 'x'</p>											x			x	o		x	o	x	o	<b>Output:</b>  checkDiagWin = true  state of the board is unchanged   <b>Function Name:</b> testCheckDiagWin_03	<b>Reason:</b> This test case is unique because the 'x' just placed is at the beginning of a string of three consecutive 'x's in a diagonal direction. The function needs to check for a win condition by counting the number of 'x's in both diagonal directions (top-left to bottom-right and top-right to bottom-left). Since there are three consecutive 'x's in the top-right to bottom-left diagonal direction, checkDiagWin should return true
		x																				
	x	o																				
x	o	x	o																			

<b>Input:</b>  State(number to win = 4)  <table><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>x</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>o</td><td>x</td><td></td><td></td><td></td></tr><tr><td></td><td>x</td><td>o</td><td>x</td><td></td><td></td></tr><tr><td></td><td>o</td><td>x</td><td>o</td><td>x</td><td></td></tr></table>  pos.getRow = 3 pos.getCol = 1 p = 'x'														x						o	x					x	o	x				o	x	o	x		<b>Output:</b>  checkDiagWin = true  state of the board is unchanged   <b>Function Name:</b> testCheckDiagWin _04	<b>Reason:</b>  This test case is unique because the 'x' just placed is at the beginning of a string of four consecutive 'x's in a diagonal direction. The function needs to check for a win condition by counting the number of 'x's in both diagonal directions (top-left to bottom-right and top-right to bottom-left). Since there are four consecutive 'x's in the top-right to bottom-left diagonal direction, checkDiagWin should return true.
	x																																					
	o	x																																				
	x	o	x																																			
	o	x	o	x																																		

<b>Input:</b>  State(number to win = 4)  <table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>O</td><td>X</td></tr><tr><td></td><td></td><td></td><td></td><td>O</td><td>O</td><td>X</td></tr><tr><td></td><td></td><td></td><td>O</td><td>X</td><td>X</td><td>X</td></tr><tr><td></td><td>O</td><td>X</td><td>O</td><td>X</td><td>X</td><td>O</td></tr></table>  pos.getRow = 0 pos.getCol = 1 p = 'x'														X						O	X					O	O	X				O	X	X	X		O	X	O	X	X	O	<b>Output:</b>  checkDiagWin = false  state of the board is unchanged          <b>Function Name:</b> testCheckDiagWin_05	<b>Reason:</b> This test case is unique because the 'x' just placed is part of a diagonal string of 'x's, but the number of consecutive 'x's is not enough to meet the win condition. The function needs to check for a win condition by counting the number of 'x's in both diagonal directions (top-left to bottom-right and top-right to bottom-left). Since there are only three consecutive 'x's in the top-right to bottom-left diagonal direction and the number to win is 4, checkDiagWin should return false.
						X																																						
					O	X																																						
				O	O	X																																						
			O	X	X	X																																						
	O	X	O	X	X	O																																						

<b>Input:</b>  State(number to win = 5)  <table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>O</td><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>O</td><td>O</td><td>X</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>O</td><td>X</td><td>O</td><td>O</td><td>X</td><td></td><td></td></tr></table>  pos.getRow = 3 pos.getCol = 3 p = 'x'																																								X									O	X								O	O	X						O	X	O	O	X			<b>Output:</b>  checkDiagWin = false  state of the board is unchanged         <b>Function Name:</b> testCheckDiagWin _06	<b>Reason:</b>  This test case is unique because the 'x' just placed is part of a diagonal string of 'x's, but the number of consecutive 'x's is not enough to meet the win condition. The function needs to check for a win condition by counting the number of 'x's in both diagonal directions (top-left to bottom-right and top-right to bottom-left). Since there are only four consecutive 'x's in the top-left to bottom-right diagonal direction and the number to win is 5, checkDiagWin should return false
			X																																																																							
			O	X																																																																						
			O	O	X																																																																					
		O	X	O	O	X																																																																				



Input:	Output:	Reason:																																																	
<p>State(number to win = 7)</p> <table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>o</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>o</td><td>o</td><td>x</td><td></td><td></td></tr><tr><td></td><td>o</td><td>x</td><td>o</td><td>x</td><td>x</td><td></td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td><td>o</td><td>o</td><td>x</td></tr></table> <p>pos.getRow = 3 pos.getCol = 3 p = 'x'</p>																									o						o	o	x				o	x	o	x	x		o	x	o	x	o	o	x	<p>checkDiagWin = false</p> <p>state of the board is unchanged</p> <p><b>Function Name:</b> testCheckDiagWin_07</p>	<p>This test case is unique because the 'x' just placed is part of a diagonal string of 'x's, but the number of consecutive 'x's is not enough to meet the win condition. The function needs to check for a win condition by counting the number of 'x's in both diagonal directions (top-left to bottom-right and top-right to bottom-left). Since the longest diagonal string of consecutive 'x's is four, and the number to win is 7, checkDiagWin should return false.</p>
			o																																																
		o	o	x																																															
	o	x	o	x	x																																														
o	x	o	x	o	o	x																																													

**boolean checkTie()**

Input:	Output:	Reason:																
<p>State(number to win = 4)</p> <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td></tr><tr><td>O</td><td>O</td><td>X</td><td>X</td></tr></table>											X		O	O	X	X	<p>checkTie = false</p> <p>state of the board is unchanged</p> <p><b>Function Name:</b> testCheckTie_01</p>	<p>This test case is unique because the game board has several empty spots, and it is easy to identify that the game is not tied. The function should return false as there are still available moves.</p>
		X																
O	O	X	X															

<b>Input:</b>  <b>State(number to win = 4)</b>  <table><tr><td></td><td>x</td><td>o</td><td>x</td></tr><tr><td>o</td><td>o</td><td>o</td><td>x</td></tr><tr><td>X</td><td>X</td><td>X</td><td>O</td></tr><tr><td>o</td><td>o</td><td>X</td><td>X</td></tr></table>		x	o	x	o	o	o	x	X	X	X	O	o	o	X	X	<b>Output:</b>  checkTie = false  state of the board is unchanged  <b>Function Name:</b> testCheckTie_02	<b>Reason:</b>  This test case is unique because the game board appears to be almost full, with only one empty spot. However, since there is still an available move, the function should return false, indicating that the game is not tied.
	x	o	x															
o	o	o	x															
X	X	X	O															
o	o	X	X															

<b>Input:</b>  State(number to win = 4) <table><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>o</td><td>x</td><td>o</td><td>o</td></tr><tr><td>x</td><td>x</td><td>x</td><td>o</td></tr><tr><td>o</td><td>o</td><td>x</td><td>x</td></tr></table>	o	x	o	x	o	x	o	o	x	x	x	o	o	o	x	x	<b>Output:</b>  checkTie = true  state of the board is unchanged  <b>Function Name:</b> testCheckTie_03	<b>Reason:</b>  This test case is unique because the game board is completely full, and there are no available moves left. The function should return true, indicating that the game is tied.
o	x	o	x															
o	x	o	o															
x	x	x	o															
o	o	x	x															

<b>Input:</b>  State(number to win = 4)  <table><tr><td>x</td><td></td><td></td><td></td></tr><tr><td>o</td><td>x</td><td>o</td><td></td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>o</td></tr></table>	x				o	x	o		x	o	x	o	o	x	o	o	<b>Output:</b>  checkTie = false  state of the board is unchanged  <b>Function Name:</b> testCheckTie_04	<b>Reason:</b>  This test case is unique because the game board has an irregular distribution of 'x' and 'o' markers, but there are still empty spots available. The function should return false, indicating that the game is not tied.
x																		
o	x	o																
x	o	x	o															
o	x	o	o															

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

<b>Input:</b>  State(number to win = 4) <table><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>x</td><td>x</td><td>o</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr></table> BoardPosition (row = 0, col = 0)	x	o	x	o	o	x	o	x	x	x	o	o	o	x	o	x	<b>Output:</b>  whatsAtPos = 'o'  state of the board is unchanged  <b>Function Name:</b> testCheckWhatsAtPos_01	<b>Reason:</b>  This test case is unique because the function should return the marker at the specified BoardPosition (row = 0, col = 0). In this case, the marker is 'o', so the function should return 'o'.
x	o	x	o															
o	x	o	x															
x	x	o	o															
o	x	o	x															

<b>Input:</b>  State(number to win = 4)  <table><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr></table>  BoardPosition (row = 2, col = 1)	x	o	x	o	o	x	o	x	x	o	x	o	o	x	o	x	<b>Output:</b>  whatsAtPos = 'x'  state of the board is unchanged  <b>Function Name:</b> testCheckWhatsAtPos_02	<b>Reason:</b>  This test case is unique because the function should return the marker at the specified BoardPosition (row = 2, col = 1). In this case, the marker is 'x', so the function should return 'x'.
x	o	x	o															
o	x	o	x															
x	o	x	o															
o	x	o	x															

<b>Input:</b>  <b>State(number to win = 4)</b>  <table><tr><td>x</td><td>o</td><td></td><td>o</td></tr><tr><td>o</td><td>o</td><td>o</td><td>x</td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr></table>  BoardPosition (row = 3 col = 2)	x	o		o	o	o	o	x	x	o	x	o	o	x	o	x	<b>Output:</b>  whatsAtPos = ''  state of the board is unchanged  <b>Function Name:</b> testCheckWhatsAtPos_03	<b>Reason:</b>  This test case is unique because the function should return the marker at the specified BoardPosition (row = 3, col = 2). In this case, there is no marker, so the function should return '' (empty space).
x	o		o															
o	o	o	x															
x	o	x	o															
o	x	o	x															

<b>Input:</b>  <b>State(number to win = 4)</b> <table><tr><td>x</td><td>o</td><td></td><td>o</td></tr><tr><td>o</td><td>x</td><td></td><td>x</td></tr><tr><td>x</td><td>o</td><td></td><td>o</td></tr><tr><td>o</td><td>x</td><td></td><td>x</td></tr></table>  BoardPosition (row = 0, col = 2)	x	o		o	o	x		x	x	o		o	o	x		x	<b>Output:</b>  whatsAtPos = ''  state of the board is unchanged  <b>Function Name:</b> testCheckWhatsAtPos_04	<b>Reason:</b>  This test case is unique because the function should return the marker at the specified BoardPosition (row = 0, col = 2). In this case, the marker is '', so the function should return ''(empty space) This is also a unique case because of how the whole board is full except one column.
x	o		o															
o	x		x															
x	o		o															
o	x		x															

<b>Input:</b>  <b>State(number to win = 4)</b> <table border="1"><tr><td>x</td><td></td><td></td><td></td></tr><tr><td>o</td><td></td><td></td><td></td></tr><tr><td>x</td><td></td><td></td><td></td></tr><tr><td>o</td><td></td><td></td><td></td></tr></table> <b>BoardPosition (row = 3, col = 3)</b>	x				o				x				o				<b>Output:</b>  whatsAtPos = ''  state of the board is unchanged  <b>Function Name:</b> testCheckWhatsAtPos_05	<b>Reason:</b>  This test case is unique because the function should return the marker at the specified BoardPosition (row = 3, col = 3). In this case, the marker is '', so the function should return ''(empty space). This is also a unique case because of how the whole board is empty except one column.
x																		
o																		
x																		
o																		

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

<b>Input:</b>  <b>State(number to win = 4)</b> <table><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr></table> BoardPosition (row = 0, col = 0), player = 'o'	x	o	x	o	o	x	o	x	x	o	x	o	o	x	o	x	<b>Output:</b>  isPlayerAtPos = true  state of the board is unchanged  <b>Function Name:</b> testCheckIsPlayerAtPos_01	<b>Reason:</b>  This test case is unique because the function should return true if the specified player ('o') is at the specified BoardPosition (row = 0, col = 0). In this case, the player 'o' is at the given position, so the function should return true.
x	o	x	o															
o	x	o	x															
x	o	x	o															
o	x	o	x															

<b>Input:</b>  <b>State(number to win = 4)</b> <table><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr></table> BoardPosition (row = 2, col = 1), player = 'x'	x	o	x	o	o	x	o	x	x	o	x	o	o	x	o	x	<b>Output:</b>  isPlayerAtPos = true  state of the board is unchanged  <b>Function Name:</b> testCheckIsPlayerAtPos_02	<b>Reason:</b>  This test case is unique because the function should return true if the specified player ('x') is at the specified BoardPosition (row = 2, col = 1). In this case, the player 'x' is at the given position, so the function should return true.
x	o	x	o															
o	x	o	x															
x	o	x	o															
o	x	o	x															

<b>Input:</b>  <b>State(number to win = 4)</b> <table><tr><td></td><td>o</td><td>o</td><td>x</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr></table> BoardPosition (row = 3, col = 0), player = 'o'		o	o	x	o	x	o	x	x	o	x	o	o	x	o	x	<b>Output:</b>  isPlayerAtPos = false  state of the board is unchanged  <b>Function Name:</b> testCheckIsPlayerAtPos_03	<b>Reason:</b>  This test case is unique because the function should return false if the specified player ('o') is not at the specified BoardPosition (row = 3, col = 0). In this case, there is no marker at the given position, so the function should return false.
	o	o	x															
o	x	o	x															
x	o	x	o															
o	x	o	x															

<b>Input:</b>  <b>State(number to win = 4)</b>  <table border="1"><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td><b>x</b></td><td><b>o</b></td><td></td></tr><tr><td>x</td><td>x</td><td>o</td><td></td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td></tr></table>  BoardPosition (row = 1, col = 2), player = 'x'						<b>x</b>	<b>o</b>		x	x	o		o	x	o	x	<b>Output:</b>  isPlayerAtPos = false  state of the board is unchanged   <b>Function Name:</b> testCheckIsPlayerAtPos_04	<b>Reason:</b>  This test case is unique because the function should return false if the specified player ('x') is not at the specified BoardPosition (row = 1, col = 2). In this case, the player 'o' is at the given position, so the function should return false.
	<b>x</b>	<b>o</b>																
x	x	o																
o	x	o	x															

<b>Input:</b>  <b>State(number to win = 4)</b>  <table><tr><td></td><td></td><td>x</td><td></td></tr><tr><td></td><td></td><td>x</td><td></td></tr><tr><td></td><td></td><td>x</td><td></td></tr><tr><td></td><td></td><td>o</td><td></td></tr></table>  BoardPosition (row = 3, col = 3), player = 'o'			x				x				x				o		<b>Output:</b>  isPlayerAtPos = false  state of the board is unchanged  <b>Function Name:</b> testCheckIsPlayerAtPos_05	<b>Reason:</b>  This test case is unique because the function should return false if the specified player ('o') is not at the specified BoardPosition (row = 3, col = 3). In this case, the player 'x' is at the given position, so the function should return false.
		x																
		x																
		x																
		o																

```
void placeToken(char p, int c)
```

<b>Input:</b>	<b>Output:</b>	<b>Reason:</b>																																																		
<b>State:</b> <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><b>o</b></td><td></td><td></td></tr><tr><td><b>x</b></td><td><b>o</b></td><td><b>x</b></td><td></td><td></td></tr></table> <p>p = 'x' c = 1</p>																		<b>o</b>			<b>x</b>	<b>o</b>	<b>x</b>			<b>State:</b> <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>x</td><td>o</td><td></td><td></td></tr><tr><td>x</td><td>o</td><td>x</td><td></td><td></td></tr></table> <p><b>Function Name:</b> testPlace_token_01</p>																	x	o			x	o	x			<p>This test case is unique because I am placing my marker in a column that was not empty, and also was not close to being full.</p>
		<b>o</b>																																																		
<b>x</b>	<b>o</b>	<b>x</b>																																																		
	x	o																																																		
x	o	x																																																		

<div><div>Input:</div><div>State:</div><table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>x</td><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td></td><td></td><td></td><td></td></tr></table><div><p>p = 'x'</p><p>c = 0</p></div></div> <div><div>Output:</div><div>State:</div><table><tr><td>x</td><td></td><td></td><td></td><td></td></tr><tr><td>x</td><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td></td><td></td><td></td><td></td></tr></table><div><div>Function Name:</div><div>testPlace_token_02</div></div></div> <div><div>Reason:</div><div><p>This test case is unique because it involves placing the 'x' marker in a column that was not empty, and also was not close to being full. Additionally, this placement completely fills row 1 of the game board, not allowing any more tokens to be placed in that row.</p></div></div>						x					o					o					o					x					x					o					o					o				
x																																																		
o																																																		
o																																																		
o																																																		
x																																																		
x																																																		
o																																																		
o																																																		
o																																																		

<b>Input:</b>  <b>State:</b> <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><b>o</b></td><td><b>x</b></td><td></td><td><b>x</b></td><td><b>o</b></td></tr></table> <p>p = 'x' c = 2</p>																					<b>o</b>	<b>x</b>		<b>x</b>	<b>o</b>	<b>Output:</b>  <b>State:</b> <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><b>o</b></td><td><b>x</b></td><td><b>x</b></td><td><b>x</b></td><td><b>o</b></td></tr></table> <b>Function Name:</b> testPlace_token_03																					<b>o</b>	<b>x</b>	<b>x</b>	<b>x</b>	<b>o</b>	<b>Reason:</b>  This test case is unique because I am placing my marker in a column that was empty, and also was not close to being full and also completely filling row 0 of the gameboard not allowing anymore tokens to be placed on that row.
<b>o</b>	<b>x</b>		<b>x</b>	<b>o</b>																																																
<b>o</b>	<b>x</b>	<b>x</b>	<b>x</b>	<b>o</b>																																																

<b>Input:</b>	<b>Output:</b>	<b>Reason:</b>																																																		
<b>State:</b> <table><tr><td>x</td><td></td><td></td><td></td><td></td></tr><tr><td>x</td><td>x</td><td></td><td></td><td></td></tr><tr><td>o</td><td>o</td><td></td><td></td><td></td></tr><tr><td>o</td><td>x</td><td>x</td><td>x</td><td></td></tr><tr><td>o</td><td>x</td><td>x</td><td>x</td><td>x</td></tr></table> <p>p = 'x' c = 2</p>	x					x	x				o	o				o	x	x	x		o	x	x	x	x	<b>State:</b> <table><tr><td>x</td><td></td><td></td><td></td><td></td></tr><tr><td>x</td><td>x</td><td></td><td></td><td></td></tr><tr><td>o</td><td>o</td><td>x</td><td></td><td></td></tr><tr><td>o</td><td>x</td><td>x</td><td>x</td><td></td></tr><tr><td>o</td><td>x</td><td>x</td><td>x</td><td>x</td></tr></table> <b>Function Name:</b> testPlace_token_04	x					x	x				o	o	x			o	x	x	x		o	x	x	x	x	<p>This test case is unique because it involves placing the 'x' marker in a column that was not empty and has a mix of 'x' and 'o' markers. The board is also not close to being full, and the placement results in a more balanced distribution of tokens across the rows, allowing for more strategic play in future moves. Also this completes a diagonal row of x's</p>
x																																																				
x	x																																																			
o	o																																																			
o	x	x	x																																																	
o	x	x	x	x																																																
x																																																				
x	x																																																			
o	o	x																																																		
o	x	x	x																																																	
o	x	x	x	x																																																

<b>Input:</b>  <b>State:</b> <table><tr><td>x</td><td>o</td><td>x</td><td>o</td><td></td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td><td>x</td></tr></table>  p = 'x' c = 4	x	o	x	o		o	x	o	x	o	x	o	x	o	x	o	x	o	x	o	x	o	x	o	x	<b>Output:</b>  State: <table><tr><td>x</td><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td><td>x</td></tr><tr><td>o</td><td>x</td><td>o</td><td>x</td><td>o</td></tr><tr><td>x</td><td>o</td><td>x</td><td>o</td><td>x</td></tr></table>  <b>Function Name:</b> testPlace_token_05	x	o	x	o	x	o	x	o	x	o	x	o	x	o	x	o	x	o	x	o	x	o	x	o	x	<b>Reason:</b>  This test case is unique because it involves placing the 'x' marker in a column that is almost full, with a mix of 'x' and 'o' markers, and has only one spot left for the token to be placed. After placing the 'x' token, the game board becomes completely full.
x	o	x	o																																																	
o	x	o	x	o																																																
x	o	x	o	x																																																
o	x	o	x	o																																																
x	o	x	o	x																																																
x	o	x	o	x																																																
o	x	o	x	o																																																
x	o	x	o	x																																																
o	x	o	x	o																																																
x	o	x	o	x																																																