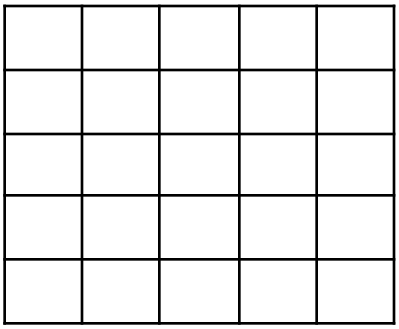
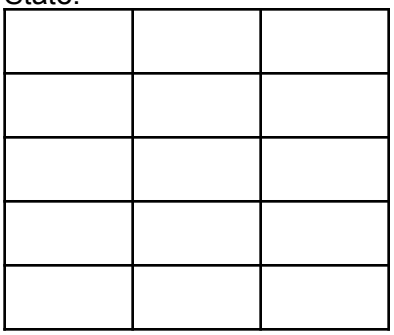
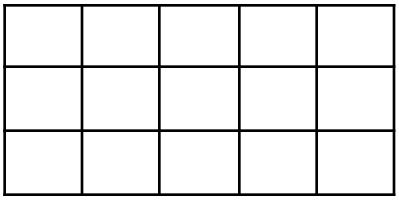
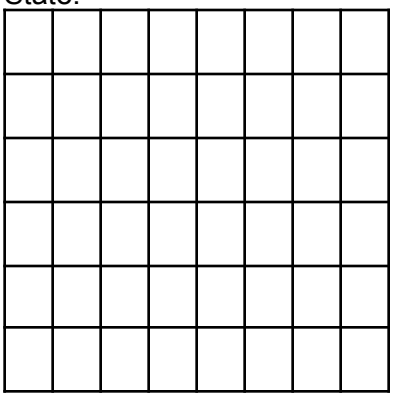
`Constructor(int width, int height, int numToWin)`

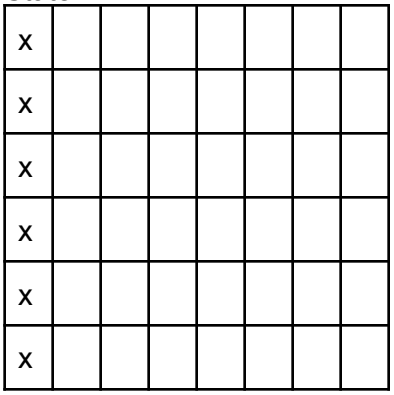
Input:	**Output:**	**Reason:** This test case is unique and distinct because the board may return differently if the height and the width are equal. **Function Name:** `testConstructor_height_and_width_equal`
width = 5 height = 5 numToWin = 3	State: 	

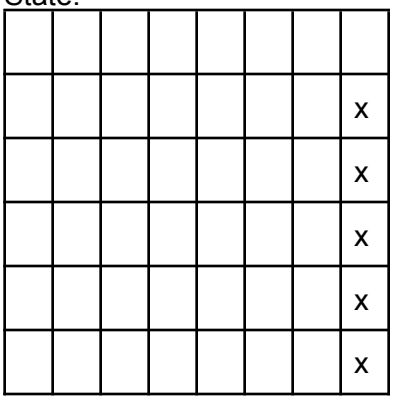
Input:	**Output:**	**Reason:** This test case is unique and distinct because the board may return differently if the width of the board is irregular for a connectX board. **Function Name:** `testConstructor_height_greater_than_width`
width = 3 height = 5 numToWin = 3	State: 	

Input:	**Output:**	**Reason:** This test case is unique and distinct because the board may return differently if the height of the board is irregular for a connectX board. **Function Name:** `testConstructor_height_less_than_width`
width = 5 height = 3 numToWin = 3	State: 	

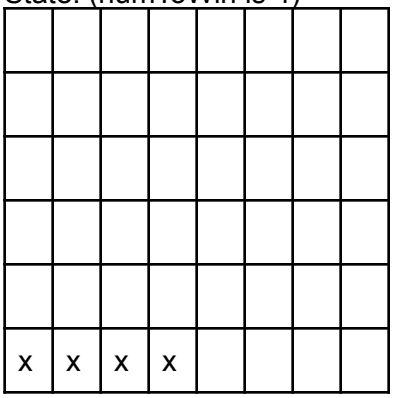
Boolean checkIfFree(int x)

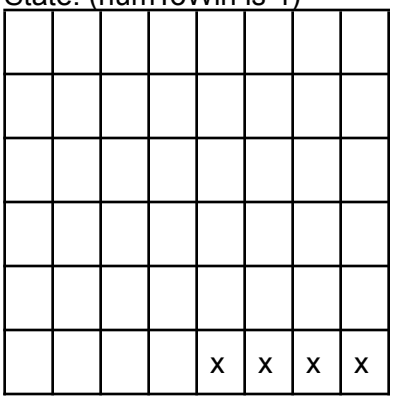
Input: State:  x=0	**Output:** checkIfFree = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to see whether or not it would see if a column is free if nothing was put in it yet. **Function Name:** `testCheckIfFree_empty_column`

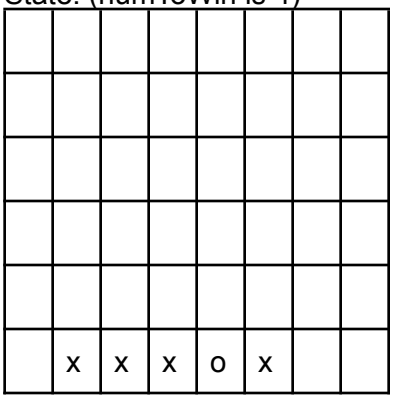
Input: State:  x=0	**Output:** checkIfFree = False State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to see whether or not the function would fail if the column was not free. **Function Name:** `testCheckIfFree_full_column`

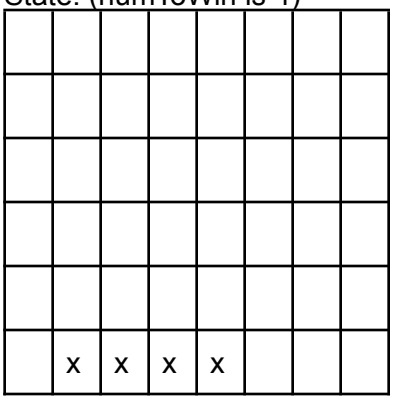
Input: State:  x=width-1	**Output:** checkIfFree = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to see whether or not it would see that I can place it in a partially filled column, especially one that would fill the column as a border case. **Function Name:** `testCheckIfFree_top_right_boundary`

boolean checkHorizWin(BoardPosition a, char c)

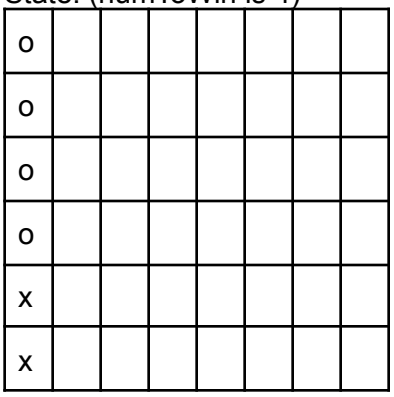
Input: State: (numToWin is 4)  a = 0,0 c = ‘x’	**Output:** checkHorizWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check the border case of if it saw horizontal wins on the left most position, as well as if it picked up wins where it only looked right from the position I checked. **Function Name:** `testHorizWin_leftmost_boundary`

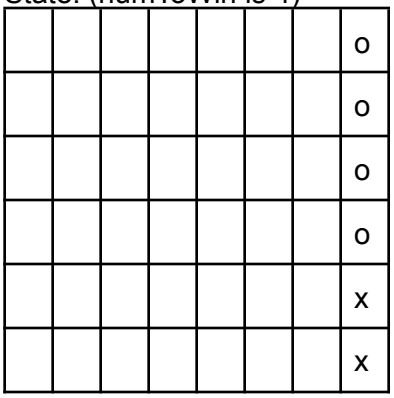
Input: State: (numToWin is 4)  a = width-1,0 c = ‘x’	**Output:** checkHorizWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check the border case of if it saw horizontal wins on the right most position, as well as if it picked up wins where it only looked left from the position I checked. **Function Name:** `testHorizWin_rightmost_boundary`

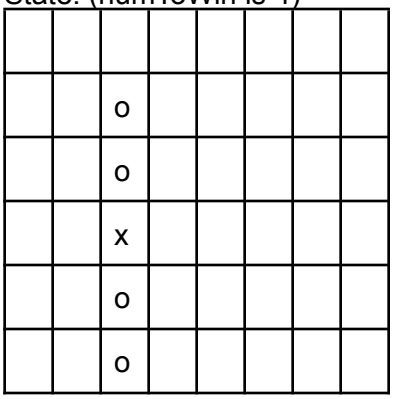
Input: State: (numToWin is 4)  a = 3,0 c = ‘x’	**Output:** checkHorizWin = False State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the function would stop when hit with other symbols or spaces, even though there are enough unconnected symbols in the row to win otherwise. **Function Name:** `testHorizWin_break_on_other_inputs`

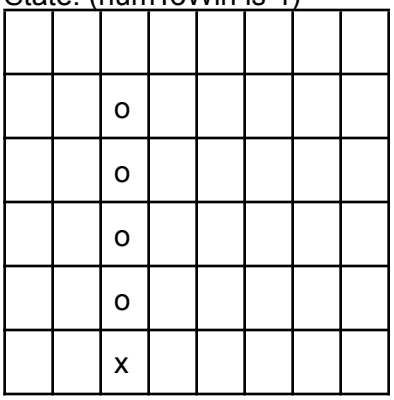
Input: State: (numToWin is 4)  a = 3,0 c = ‘x’	**Output:** checkHorizWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the function returned a win when the last position placed is in the middle of the line, in that it looks both left and right to find its win. **Function Name:** `testHorizWin_middle_win_placement`

boolean checkVertWin(BoardPosition a, char c)

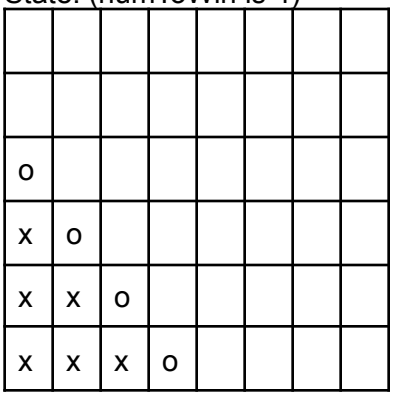
Input: State: (numToWin is 4)  a = 0,height-1 c = ‘o’	**Output:** checkVertWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check the border case of whether or not the function found a win on the leftmost position in a tricky spot at (0,height-1). **Function Name:** `testVertWin_leftmost_boundary`

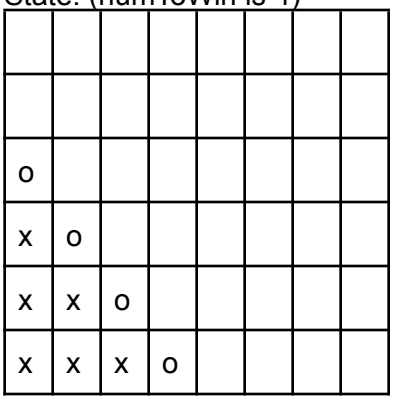
Input: State: (numToWin is 4)  a = width-1, height-1 c = ‘o’	**Output:** checkVertWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check the border case of whether or not the function found a win on the rightmost position in a tricky spot at (width-1,height-1). **Function Name:** `testVertWin_rightmost_boundary`

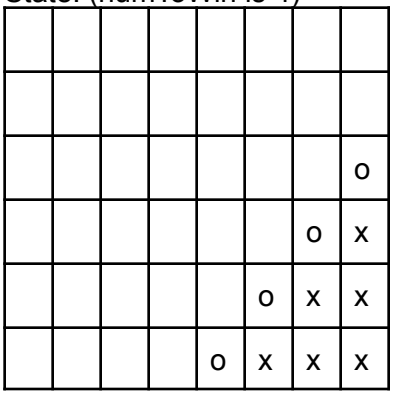
Input: State: (numToWin is 4)  a = 2, 4 c = ‘o’	**Output:** checkVertWin = False State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the function would stop when hit with other symbols or spaces, even though there are enough unconnected symbols in the column to win otherwise. **Function Name:** `testVertWin_break_on_other_inputs`

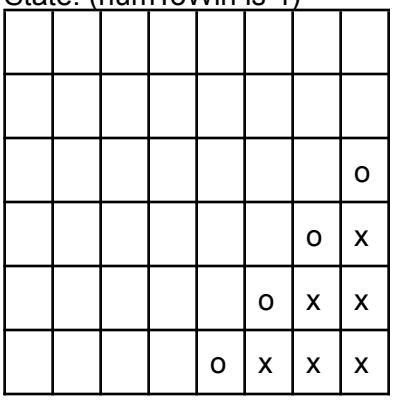
Input: State: (numToWin is 4)  a = 2,4 c = ‘o’	**Output:** checkVertWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the function found a win in the middle of the board, not hitting the top or the bottom of the graph. **Function Name:** `testVertWin_win_in_center_of_board`

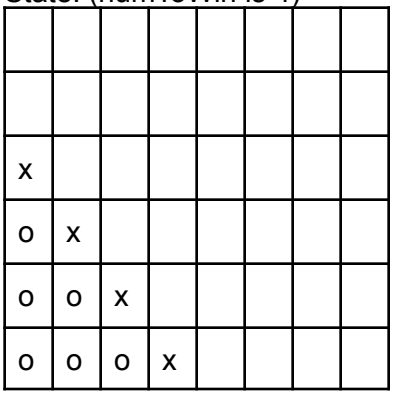
boolean checkDiagWin(BoardPosition a, char c)

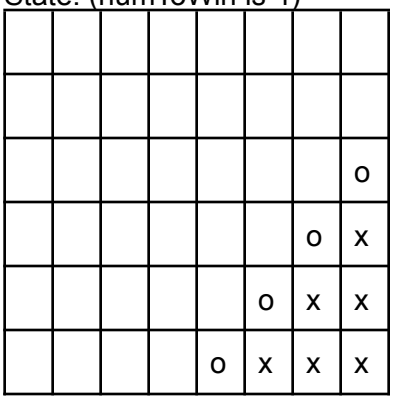
Input: State: (numToWin is 4)  a = 0,3 c = ‘o’	**Output:** checkDiagWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check whether the diagonal win would find a win in the border case of the leftmost position, as well as if the last position placed was on the left most position in the line, in that it needs to look only down and to the right. **Function Name:** `testDiagWin_leftmost_UL_to_DR`

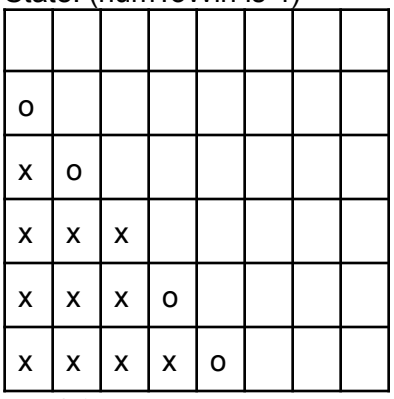
Input: State: (numToWin is 4)  a = 3,0 c = ‘o’	**Output:** checkDiagWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check whether the diagonal win would find a win if the last position placed was on the right most position in the line, in that it needs to look only up and to the left. **Function Name:** `testDiagWin_rightmost_UL_to_DR`

Input: State: (numToWin is 4)  a = width-4,0 c = ‘o’	**Output:** checkDiagWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check whether the diagonal win would find a win if the last position placed was on the left most position in the line, in that it needs to look only up and to the right. **Function Name:** `testDiagWin_leftmost_UR_to_DL`

Input: State: (numToWin is 4)  a = width-1,3 c = ‘o’	**Output:** checkDiagWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check whether the diagonal win would find a win in the border case of the rightmost position, as well as if the last position placed was on the right most position in the line, in that it needs to look only down and to the left. **Function Name:** `testDiagWin_rightmost_UR_to_DL`

Input: State: (numToWin is 4)  a = 2,1 c = ‘o’	**Output:** checkDiagWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check whether the diagonal win would find a win if the last placed token is in the center of the line in that it needs to look both up left and down right to find a win. **Function Name:** `testDiagWin_middle_win_UL_to_DR`

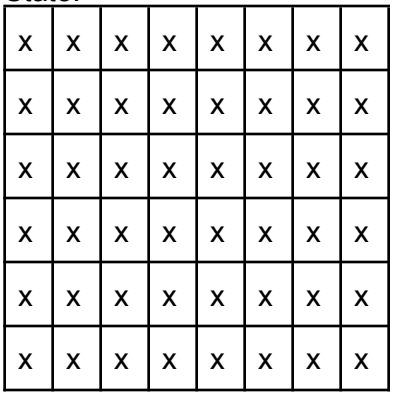
Input: State: (numToWin is 4)  a = width-2,2 c = ‘o’	**Output:** checkDiagWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check whether the diagonal win would find a win if the last placed token is in the center of the line in that it needs to look both up right and down left to find a win. **Function Name:** `testDiagWin_middle_win_UR_to_DL`

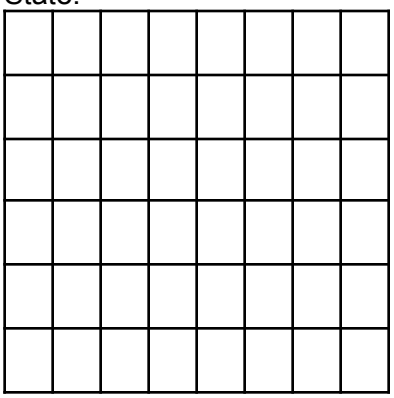
Input: State: (numToWin is 4)  a = 1,3 c = ‘o’	**Output:** checkDiagWin = False State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the function would stop when hit with other symbols or spaces, even though there are enough unconnected symbols in the diagonal line to win otherwise. **Function Name:** `testDiagWin_break_on_other_inputs`

boolean checkTie()

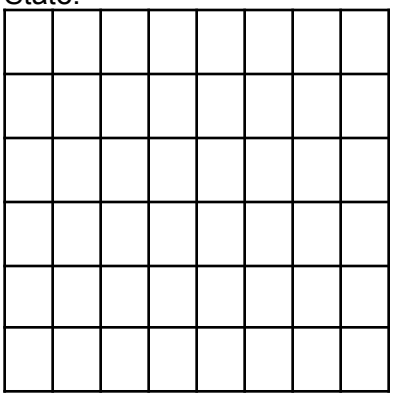
Input: State: x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	**Output:** checkTieWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the checkTie function returned a tie on the border case where the last placed token is in the top right position. **Function Name:** `testCheckTie_final_place_top_right`

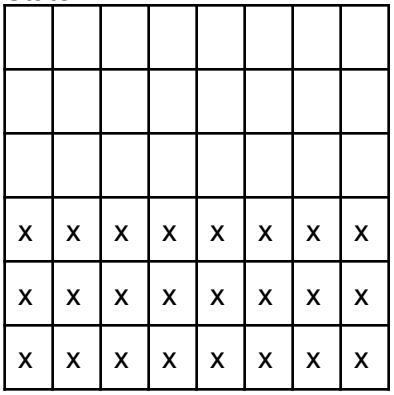
Input: State: x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	**Output:** checkTieWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the checkTie function returned a tie on the border case where the last placed token is in the top left position. **Function Name:** `testCheckTie_final_place_top_left`

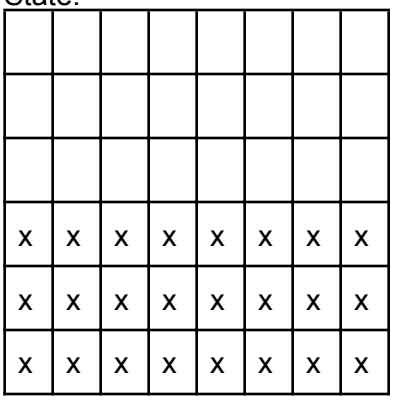
Input: State: 	**Output:** checkTieWin = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the checkTie function returned a tie in the case where the last placed token is in a neutral position, between the two extremes of 0 and width-1. **Function Name:** `testCheckTie_final_place_in_between`

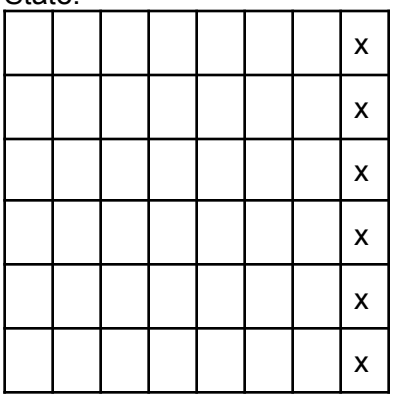
Input: State: 	**Output:** checkTieWin = False State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the checkTie function returned a false in the case that the game has not yet tied. **Function Name:** `testCheckTie_empty_board`

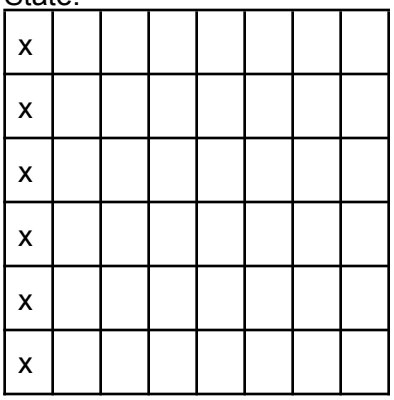
char whatsAtPos(BoardPosition a)

Input: State:  a = 0,0	**Output:** whatsAtPos = ‘ ‘ State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the whatsAtPos function returned the correct output on an empty position before anything has been filled in. **Function Name:** `testWhatsAtPos_empty_space_empty_board`

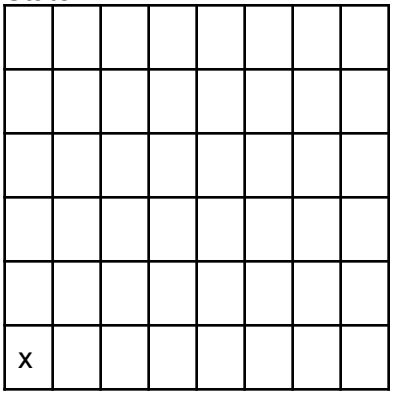
Input: State:  a = 0,4	**Output:** whatsAtPos = ‘ ‘ State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the whatsAtPos function returned the correct output on an empty position after some things have been filled in. **Function Name:** `testWhatsAtPos_empty_space_partly_filled_board`

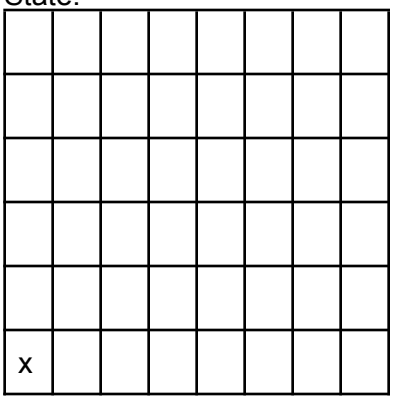
Input: State:  a = 0,2	**Output:** whatsAtPos = ‘x‘ State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the whatsAtPos function returned the correct output on a filled position after some things have been filled in. **Function Name:** `testWhatsAtPos_filled_space_partly_filled_board`

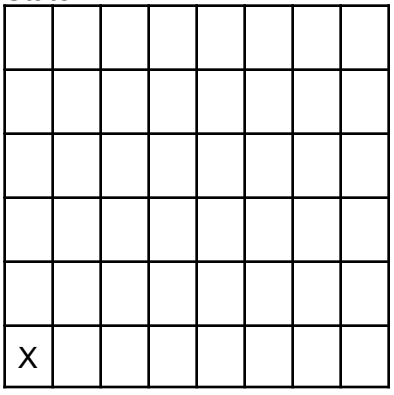
Input: State:  a = width-1,height-1	**Output:** whatsAtPos = ‘x‘ State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the whatsAtPos function returned the correct output on a border case in the tricky position (width-1,height-1). **Function Name:** `testWhatsAtPos_filled_space_top_right_boundary`

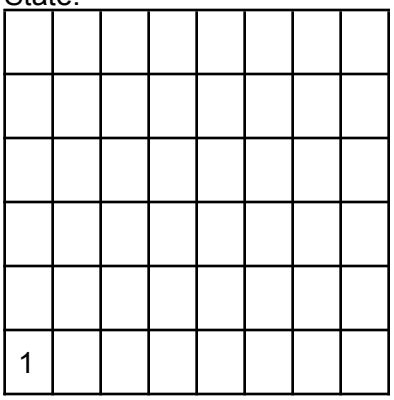
Input: State:  a = 0,height-1	**Output:** whatsAtPos = ‘x‘ State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the whatsAtPos function returned the correct output on a border case in the tricky position (0,height-1). **Function Name:** `testWhatsAtPos_filled_space_top_left_boundary`

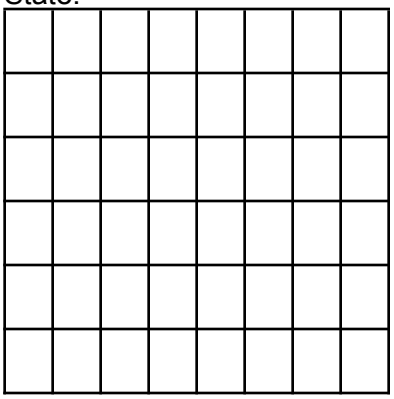
boolean isPlayerAtPos(BoardPosition a, char c)

Input: State:  a = 0,0 c = ‘o’	**Output:** isPlayerAtPos = False State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the isPlayerAtPos function returned if the expected output is different from the actual token on the table. **Function Name:** `testIsPlayerAtPos_return_false_for_nonmatching`

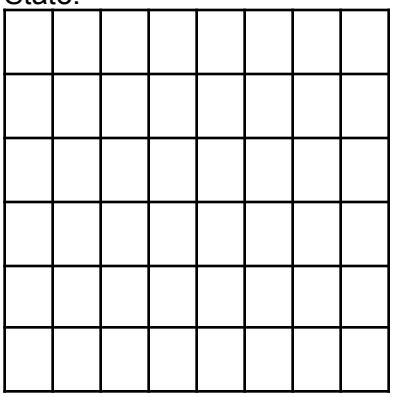
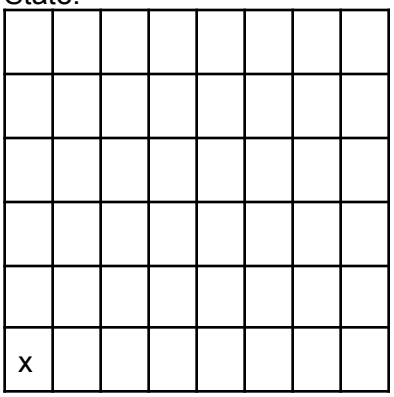
Input: State:  a = 0,0 c = ‘x’	**Output:** isPlayerAtPos = true State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the isPlayerAtPos function returned if the expected output is the same as the actual token on the table. **Function Name:** `testIsPlayerAtPos_return_true_for_matching`

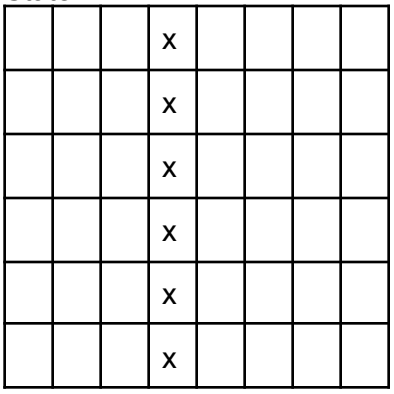
Input: State:  a = 0,0 c = ‘x’	**Output:** isPlayerAtPos = False State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the isPlayerAtPos function returned if the expected output checked whether capital letters are different from lowercase letters. **Function Name:** `testIsPlayerAtPos_check_capitalization`

Input: State:  a = 0,0 c = ‘1’	**Output:** isPlayerAtPos = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the isPlayerAtPos function correctly returned if the expected output was something other than a letter. **Function Name:** `testIsPlayerAtPos_match_with_nonletter_character`

Input: State:  a = 0,0 c = ‘ ’	**Output:** isPlayerAtPos = True State of the board is unchanged	**Reason:**This test case is unique and distinct because I needed to check if the isPlayerAtPos function correctly returned if the expected output was an empty space. **Function Name:** `testIsPlayerAtPos_check_empty_space`

void placeToken(char c, int x)

Input:	**Output:**	**Reason:** This test case is unique and distinct because I needed to check if you could place in an empty column, especially in the corner of the table at (0,0).
State:  c = ‘x’ x = 0	State: 	**Function Name:** `testPlaceToken_empty_column`

Input:	**Output:**	**Reason:** This test case is unique and distinct because I needed to check if you could place it in a filled column.
State:  c = ‘x’ x = 3	State of the board is unchanged	**Function Name:** `testPlaceToken_full_column`

Input:

State:

x							
x							
x							
x							
x							

c = ‘x’
x = 0

Output:

State:

x							
x							
x							
x							
x							
x							

Reason: This test case is unique and distinct because I needed to check if you could place in a nearly full column, especially in the corner of the table at (0,height-1) where the placement would fill the column.

Function Name:
`testPlaceToken_top_left_boundary`

Input:

State:

							x
							x
							x
							x
							x

c = ‘x’
x = width-1

Output:

State:

							x
							x
							x
							x
							x
							x

Reason: This test case is unique and distinct because I needed to check if you could place in a nearly full column, especially in the corner of the table at (width-1,height-1) where the placement would fill the column.

Function Name:
`testPlaceToken_top_right_boundary`

Input:

State:

o	o	o	o	o	o	o	
x	x	x	x	x	x	x	x
o	o	o	o	o	o	o	o
x	x	x	x	x	x	x	x

c = ‘o’
x = width-1

Output:

State:

o	o	o	o	o	o	o	o
x	x	x	x	x	x	x	x
o	o	o	o	o	o	o	o
x	x	x	x	x	x	x	x

Reason: This test case is unique and distinct because I needed to check if you could place in a partially filled table in which I was not looking at a border case, but somewhere in between.

Function Name:
`testPlaceToken_partially_filled_board`