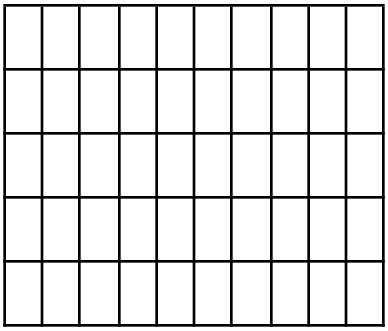
Project 4 Test Cases

Jacob Colson, Warren Wasden, Steven Cabezas

void GameBoard(int numRows, int numColumns, int numToWin)

Input:	**Output:**	**Reason:**
State: GameBoard has not yet been created	State: (numRows = 5, numColumns = 10, numToWin = 5)  numRows = 5 numColumns = 10 numToWin = 5	This test case is unique because we are creating a board of size 5x10, with the number to win set to 5. This verifies that the constructor function creates a blank board of specified size correctly. **Function Name:** test_5_10_5_GameBoard

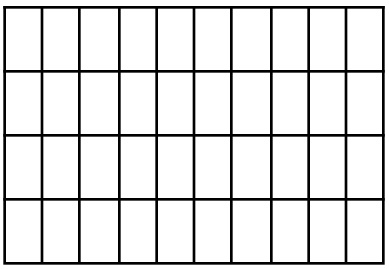
void GameBoard(int numRows, int numColumns, int numToWin)

Input:	**Output:**	**Reason:**
State: GameBoard has not yet been created	State: (numRows = 75, numColumns = 4, numToWin = 25) *Creates blank board with 75 rows, 4 columns and 25 to win* numRows = 75 numColumns = 4 numToWin = 25	This test case is unique because it makes sure that a game board of a large size can be created. Also verifies that one of non square size can be created, with odd number of rows and even number of columns. Also make sure that the max possible number to win also works. **Function Name:** test_75_4_25_GameBoard

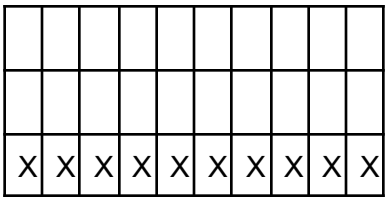
void GameBoard(int numRows, int numColumns, int numToWin)

Input:	**Output:**	**Reason:**
State: GameBoard has not yet been created	State: (numRows = 100, numColumns = 100, numToWin = 25) *Creates blank board of maxSize (100 x 100) with number to win max of 25* numRows = 100 numColumns = 100 numToWin = 25	This test case is unique because it verifies that a gameBoard can be created of maximum size (100x100). It also makes sure that the gameboard's number to win can also be a maximum of 25. **Function Name:** test_MaxSize_MaxNumToWin_GameBoard

boolean checkIfFree(int c)

Input:	**Output:**	**Reason:**
State: (number to win: 5)  c = 3	checkIfFree = true State of the board is unchanged	This test case is unique and distinct because it makes sure that checkIfFree returns true if the GameBoard is completely blank. **Function Name:** test_3_empty_CheckIfFree

boolean checkIfFree(int c)

Input:	**Output:**	**Reason:**
State: (number to win: 5)  	checkIfFree = true State of the board is unchanged	This test is unique because it verifies that checkIfFree does not return true if that column is not full. In this case each column is only half full. **Function Name:** test_0_half_CheckIfFree

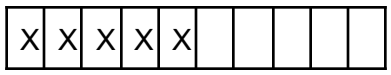
O O O O O O O O O O C = 0		

boolean checkIfFree(int c)

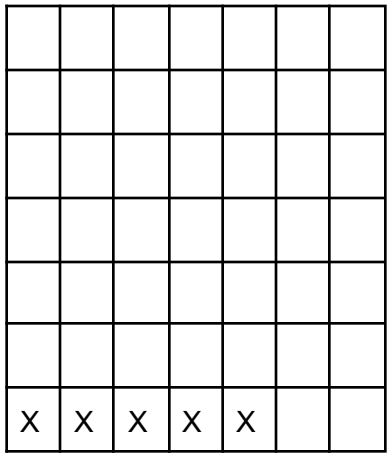
Input: State: (number to win: 5) X X X X X X X X X X O O O O O O O O O O X X X X X X X X X X O O O O O O O O O O C = 2	**Output:** checkIfFree = false State of the board is unchanged	**Reason:** This test is unique because it verifies that checkIfFree returns false if the column is full. In this case every column is full because the entire board is filled. **Function Name:** test_2_fullBoard_CheckIfFree

boolean checkHorizWin(BoardPosition pos, char p)

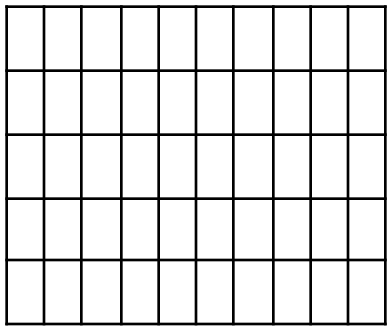
Input: State: (number to win: 5)	**Output:** checkHorizWin = true State of the board is unchanged	**Reason:** This test is unique because it verifies that a horizontal win occurs with the last placed token being placed at the right most position of 4 consecutive X's. In this case the last placed X was the 5th consecutive X placed leading to a horizontal win. **Function Name:** test_0_5_checkHorizWin

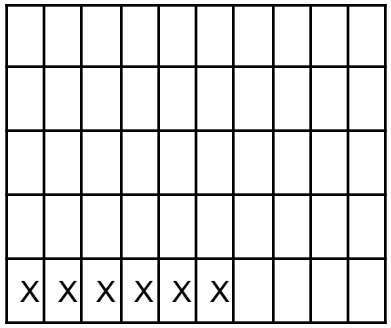
 pos.getRows = 0 pos.getColumn = 5 p = ‘X’		

boolean checkHorizWin(BoardPosition pos, char p)

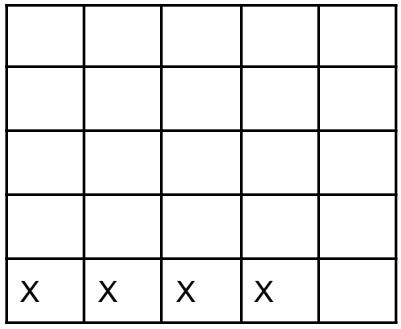
Input: State: (number to win: 5)  pos.getRow = 0 pos.getColumn = 0 p = ‘X’	**Output:** checkHorizWin = true State of the board is unchanged	**Reason:** This test is unique because it verifies that the last consecutive token placed in the left most position of the row resulted in a win. In this case, the 5th placed token was placed at the left most position of the 4 consecutive tokens resulting in a horizontal win. **Function Name:** test_5_0_checkHorizWin

boolean checkHorizWin(BoardPosition pos, char p)

Input: State (number to win: 6) 	**Output:** checkHorizWin = true State of the board is unchanged	**Reason:** This test is unique because the last consecutively placed token is in the middle of the row. This means that the checkHorizWin function needs to count to the left and right of the last placed token to verify if the last token was a win. **Function Name:** test_middle_placement_checkHorizWin

 pos.getRow = 0 pos.getColumn = 3 p = ‘X’		

boolean checkHorizWin(BoardPosition pos, char p)

Input: State: (number to win: 25) *This represents a Board of 50x50*  *Number of X’s placed is only 21* pos.getRow = 0 pos.getColumn = 1 p = ‘X’	**Output:** checkHorizWin = false State of the board is unchanged	**Reason:** This test is unique because it verifies that if the number of consecutive tokens placed is not equal to the number to win, then checkHorizWin returns false. In this case the number to win is set to the maximum possible of 25 and the number of consecutive tokens placed is only 21. **Function Name:** test_checkHorizWin_return_false

char whatsAtPos(BoardPosition pos)

Input: State: (number to win: 3)	**Output** whatsAtPos = ‘X’ State of the board is unchanged	**Reason:** This test is unique because it verifies that whatsAtPos can return the lowest element in the GameBoard. In this case

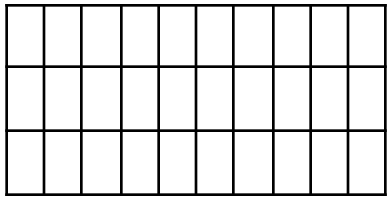
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table> pos.getRow = 0 pos.getColumn = 0		the position is 0,0 and the element at that position is X. **Function Name:** test_MinRow_MinCol_whatsAtPos

char whatsAtPos(BoardPosition pos)

Input: State: (number to win: 10) <table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table> pos.getRow = 0 pos.getColumn = 0	**Output** whatsAtPos = 'X' State of the board is unchanged	**Reason:** This test is unique because it verifies that whatsAtPos can return the lowest element in the GameBoard. In this case the position is 0,0 and the element at that position is X. **Function Name:** test_25_4_FULLBOARD_whatsAtPos

char whatsAtPos(BoardPosition pos)

Input: State: (number to win: 5) <table><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></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></table>	**Output:** whatsAtPos = ' ' State of the board is unchanged	**Reason:** This test is unique because it verifies that whatsAtPos is able to return a blank character when one is located on the board. Ih this case the whole board including position 7,9 is blank. **Function Name:** test_7_9_BLANK_whatsAtPos

 pos.getRow = 7 pos.getColumn = 9		

char whatsAtPos(BoardPosition pos)

Input: State: (number to win: 25) *This board represents a 100x100 board of max size* P J P J P J P J P J P J P J P J P J P J P J P J P J P J P J P J P J P J pos.getRow = 100 pos.getColumn = 100	**Output:** whatsAtPos = 'J' State of the board is unchanged	**Reason:** This test is unique because it verifies that whatsAtPos works at the maximum possible position, 100,100. In this case it returns a J because that is what is located at that position. **Function Name:** test_MaxCol_MaxROW_FULLBOARD_whatsAtPos

char whatsAtPos(BoardPosition pos)

Input: State: (number to win: 25) *This board represents a 100x100 board of max size* P J P J P J	**Output:** whatsAtPos = 'P' State of the board is unchanged	**Reason:** This test is unique because it verifies that whatsAtPos can return a character that is located somewhere in the middle of the board. In this case the position is at the exact center of the GameBoard. **Function Name:**

P J P J P J P J P J P J P J P J P J P J P J P J P J P J P J pos.getRow = 50 pos.getColumn = 50		test_ExactCenter_FULLBOARD_whatsAtPos

boolean isPlayerAtPos(BoardPosition pos, char player)

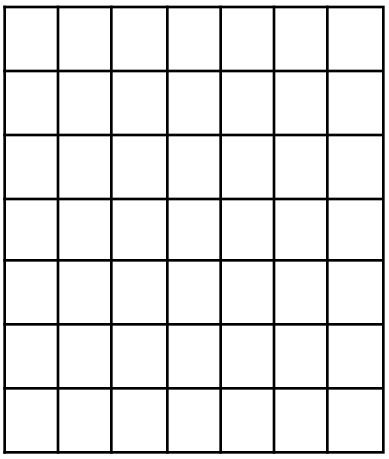
Input: State: (number to win: 5) O X O X O X O X O X O X O X O X O X O X pos.getRow = 0 pos.getColumn = 0 player = 'O'	**Output:** isPlayerAtPos = true State of the board is unchanged	**Reason:** This test is unique because it verifies that isPlayerAtPos returns true if the character at the minimum position on the board matches the desired character. **Function Name:** test_bottomLeft_FULLBOARD_isPlayerAtPos

boolean isPlayerAtPos(BoardPosition pos, char player)

Input: State: (number to win: 5) O X O X O X O X O X O X	**Output:** isPlayerAtPos = true State of the board is unchanged	**Reason:** This test is unique because it verifies that isPlayerAtPosition works at the bottom right position of the board and with a fully populated board. In this case it returns true because the character at the position is X and the expected character is X.

O X O X O X O X pos.getRow = 4 pos.getColumn = 3 player = ‘X’		**Function Name:** test_bottomRight_FULLBOARD_isPlayerAtPos

boolean isPlayerAtPos(BoardPosition pos, char player)

Input: State: (number to win: 5)  pos.getRow = 3 pos.getColumn = 3 player = ‘X’	**Output:** isPlayerAtPos = false State of the board is unchanged	**Reason:** This test is unique because it verifies that isPlayerAtPos correctly returns false when the expected character is not at the expected BoardPosition. In this case, the expected character was ‘X’ and at position 3,3 the actual character was a blank. **Function Name:** test_3_3_EMPTYBOARD_isPlayerAtPos

boolean isPlayerAtPos(BoardPosition pos, char player)

Input: State: (number to win: 5) U W U W U W U W U W U W U W U W	**Output:** isPlayerAtPos = true State of the board is unchanged	**Reason:** This test is unique because it checks to make sure isPlayerAtPos returns true if the expected character matches the actual character at the top rightmost position in the board. In this case, the expected character was U and actual character was a U. **Function Name:**

<table><tr><td>U</td><td>W</td><td>U</td><td>W</td></tr></table> pos.getRow = 4 pos.getColumn = 0 player = ‘U’		test_TopLeft_FULLBOARD_isPlayerAtPos

boolean isPlayerAtPos(BoardPosition pos, char player)

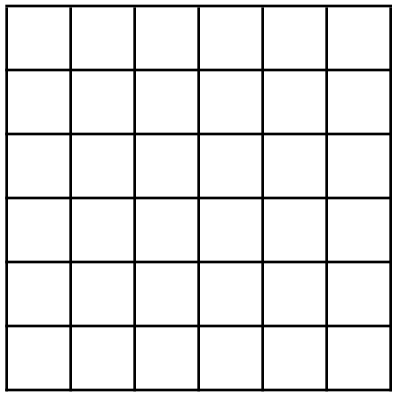
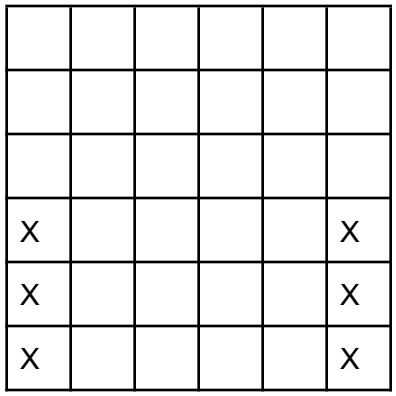
Input: State: (number to win: 5) *This board represents one with 75 rows and 4 columns* <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>O</td><td>X</td><td>O</td><td>X</td></tr></table> pos.getRow = 25 pos.getColumn = 2 player = ‘O’	**Output:** isPlayerAtPos = true State of the board is unchanged	**Reason:** This test is unique because it verifies that isPlayerAtPos correctly returns true if the expected character matches the actual character at a position somewhere in the middle of the board, and the board is only half full. **Function Name:** test_25_2_HalfFull_isPlayerAtPos

void dropToken(char p, int c)

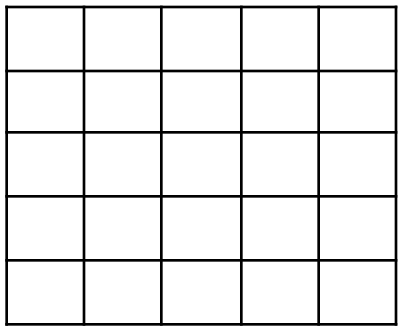
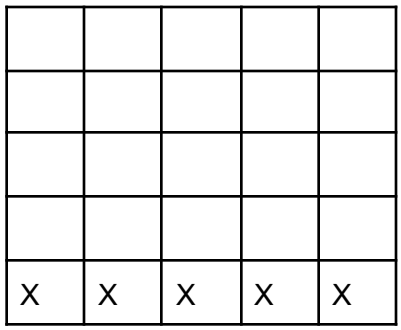
Input: State: <table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table> p = ‘X’	**Output:** State: <table><tr><td></td><td></td><td></td><td>X</td><td></td></tr><tr><td></td><td></td><td></td><td>X</td><td></td></tr><tr><td></td><td></td><td></td><td>X</td><td></td></tr><tr><td></td><td></td><td></td><td>X</td><td></td></tr><tr><td></td><td></td><td></td><td>X</td><td></td></tr></table>	**Reason:** This test is unique because it verifies that dropToken correctly drops tokens at the lowest position possible and as the column fills up, drop token places a token at the next available position in the column. **Function Name:** test_3_fullColumn_DropToken

c = 3		

void dropToken(char p, int c)

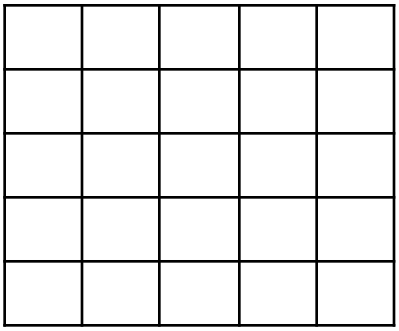
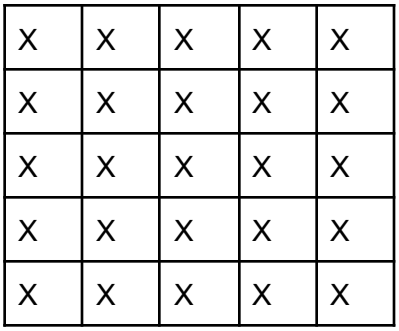
Input: State:  p = ‘X’ c = 0 c = 5	**Output:** State: 	**Reason:** This test is unique because it verifies that dropToken can correctly place tokens in two separate columns and is able to only fill the columns to a specified limit. In this case the columns are to only be half filled. **Function Name:** test_0_5_HalfFUll_DropToken

void dropToken(char p, int c)

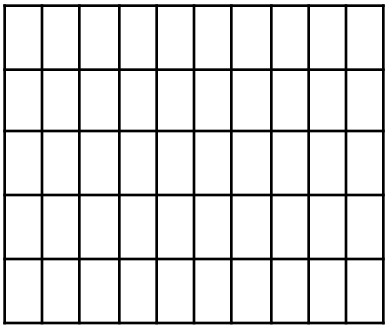
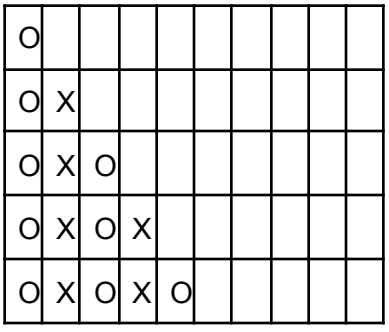
Input: State: *This board represents a board of size 25x25*  p = ‘X’ c = 0 c = 1 c = 3	**Output:** State: *This board represents a board of size 25x25* 	**Reason:** This test is unique because it checks that dropToken can correctly populate only 1 cell of a column and can correctly populate an entire row if available. In this case the bottom row is available to be populated. **Function Name:** test_fullBottomRow_DropToken

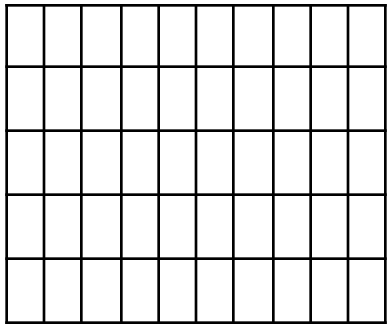
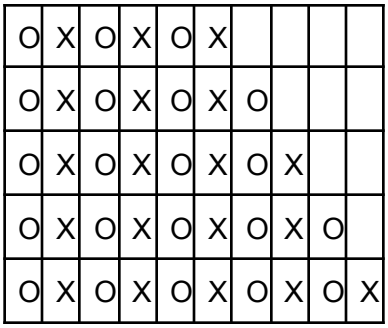
……. c = 25		

void dropToken(char p, int c)

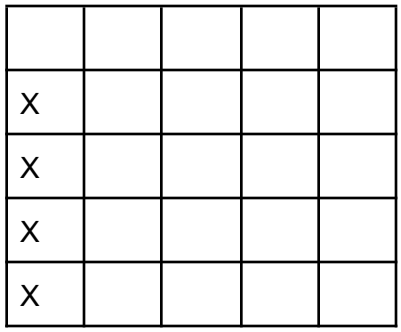
Input: State: *This board represents a board of size 25x25*  p = ‘X’ p = ‘X’ c = 0 c = 1 c = 3 ……. c = 25	**Output:** State: *This board represents a board of size 25x25* 	**Reason:** This test is unique because it checks that dropToken can correctly fill a board going column by column. In this case all 25 columns are being filled by calling dropToken to the corresponding number of rows and columns. **Function Name:** test_FullBoard_DropToken

void dropToken(char p, int c)

Input: State: *This board represents a board of size 25x25* 	**Output:** State: *This board represents a board of size 25x25* 	**Reason:** This test is unique because it checks that dropToken can correctly fill a board going diagonally. In this case the board is being filled diagonally by calling dropToken to the corresponding number of rows and columns. **Function Name:** test_DropDiagonally_DropToken

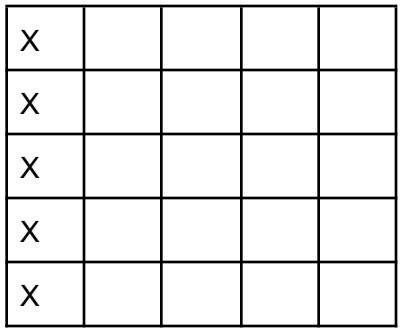
 p = ‘X’ p = ‘O’ c = 0 c = 1 c = 3 ……. c = 25		

boolean checkVertWin(BoardPosition pos, char p)

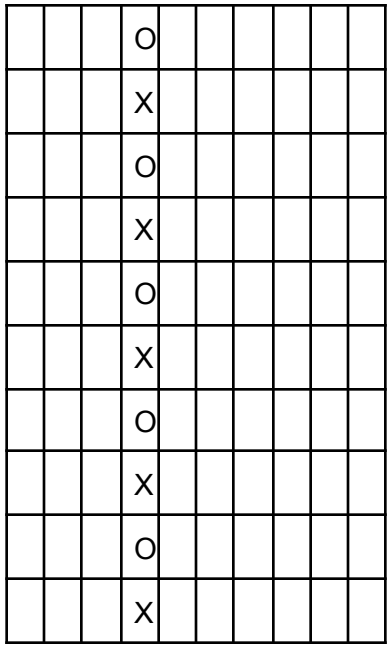
Input: State: (number to win: 5) *This board represents a board of size 20x20*  pos.getRow = 3 pos.getCol = 0 p = ‘X’	**Output:** checkVertWin = false State of the board is unchanged	**Reason:** This test is unique because it verifies that if the number of consecutive tokens placed is not equal to the number to win, then checkVertWin returns false. In this case the number to win is set to 5 and the number of consecutive tokens placed is 4. **Function Name:** test_checkVertWin_return_false

boolean checkVertWin(BoardPosition pos, char p)

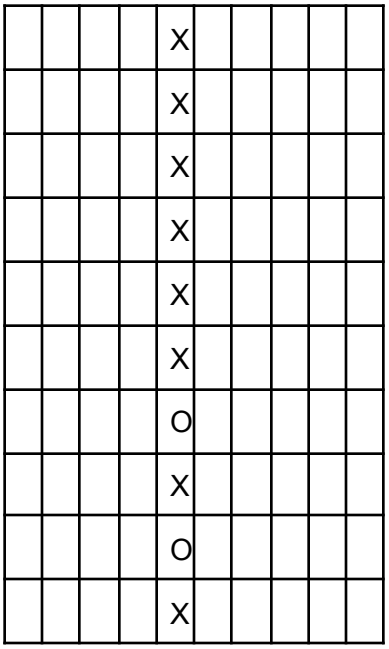
Input: State: (number to win: 5)	**Output:** checkVertWin = true State of the board is	**Reason:** This test is unique because it verifies that a vertical win occurs with the last placed

This board represents a board of size 20x20  pos.getRow = 4 pos.getCol = 0 p = 'X'	unchanged	token being placed on top of 4 consecutive X's. In this case the X was the 5th consecutive X placed leading to a vertical win. **Function Name:** test_column_0_checkVertWin

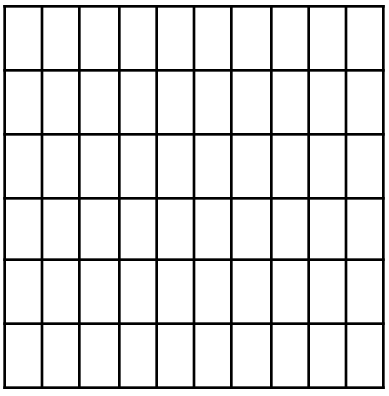
boolean checkVertWin(BoardPosition pos, char p)

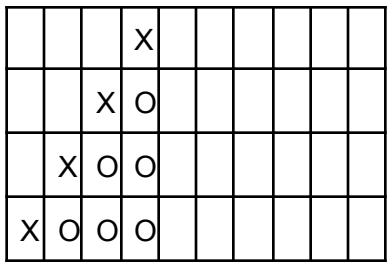
Input: State: (number to win: 5)  pos.getRow = 3 p = 'X'	**Output:** checkVertWin = false State of the board is unchanged	**Reason:** This test is unique because it verifies that if the number of consecutive tokens placed is not equal to the number to win, then checkVertWin returns false. In this case the characters are alternating and there are no consecutive characters of 5 to return true so the test is false.. **Function Name:** test_col3_alteratingCharacters_checkVertWin

boolean checkVertWin(BoardPosition pos, char p)

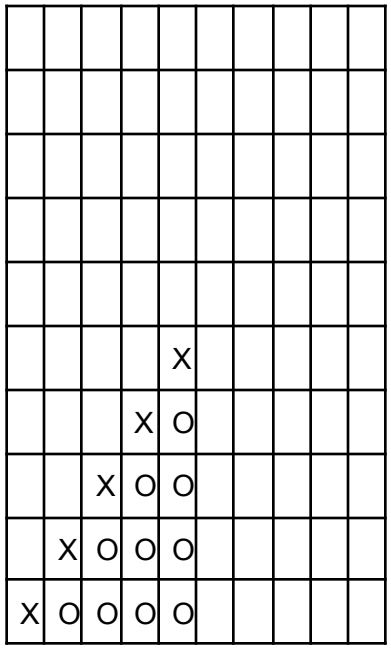
Input: State: (number to win: 5)  pos.getRow = 4 p = ‘X’	**Output:** checkVertWin = true State of the board is unchanged	**Reason:** This test is unique because it verifies that if the number of consecutive tokens placed is equal to the number to win, then checkVertWin returns true. In this case the characters are alternating until they reach row 4 and have 6 consecutive characters which means the test returns true. **Function Name:** test_verticalWinNotOnBottomRow_col4_checkVertWin

boolean checkDiagWin(BoardPosition pos, char p)

Input: State: (number to win: 5) 	**Output:** checkDiagWin = false State of the board is unchanged	**Reason:** This test is unique because it verifies that if the number of consecutive tokens placed is not equal to the number to win, then checkDiagWin returns false. In this case the number to win was set to 5 and the number of consecutive tokens placed is only 4. **Function Name:** test_checkDiagWin_return_false

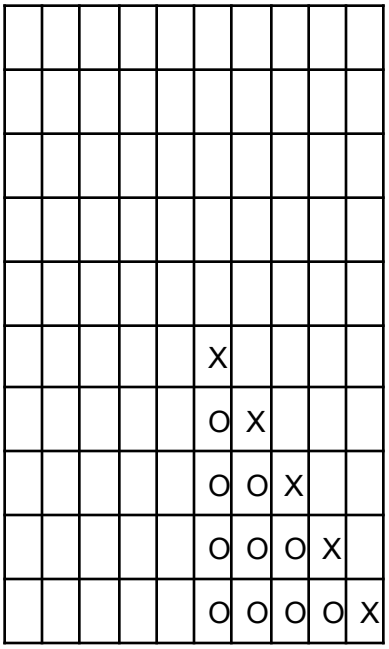
 pos.getRow = 4 pos.getCol = 4 p = ‘X’		

boolean checkDiagWin(BoardPosition pos, char p)

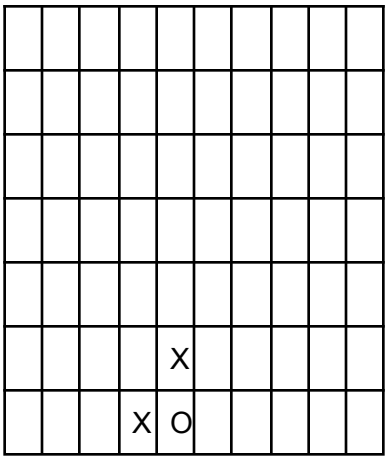
Input:	**Output:**	**Reason:**
State: (number to win: 5)  pos.getRow = 4 pos.getCol = 4 p = ‘X’	checkDiagWin = true State of the board is unchanged	This test is unique because it tests checkDiagWin going from the bottom left to the top right to make sure there are 5 consecutive tokens placed. In this case the last token was placed in the top right and the test verified that 5 consecutive tokens were placed and returned true. **Function Name:** test_topRight_to_bottomLeft_checkDiagWin

boolean checkDiagWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**

State: (number to win: 5)  pos.getRow = 0 pos.getCol = 9 p = 'X'	checkDiagWin = true State of the board is unchanged	This test is unique because it tests checkDiagWin going from the bottom right to the top left to make sure there are 5 consecutive tokens placed. In this case the last token was placed in the bottom right and the test verified that 5 consecutive tokens were placed and returned true. **Function Name:** test_bottomRight_to_topLeft_checkDiagWin

boolean checkDiagWin(BoardPosition pos, char p)

Input: State: (number to win: 5) 	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test is unique because it tests checkDiagWin going from the bottom left to the top right to make sure there are 5 consecutive tokens placed. In this case the last token was placed in the bottom left and the test verified that 5 consecutive tokens were placed and returned true. **Function Name:** test_bottomleft_to_topRight_checkDiagWin

		X	O	O					
	X	O	O	O					
X	O	O	O	O					

pos.getRow = 0
pos.getCol = 0
p = ‘X’

boolean checkDiagWin(BoardPosition pos, char p)

Input:

State: (number to win: 5)

					X				
					O	X			
					O	O	X		
					O	O	O	X	
					O	O	O	O	X

pos.getRow = 0
pos.getCol = 9
p = ‘X’

Output:
checkDiagWin = true

State of the board is unchanged

Reason:
This test is unique because it tests checkDiagWin going from the top left to the bottom right to make sure there are 5 consecutive tokens placed. In this case the last token was placed in the bottom right and the test verified that 5 consecutive tokens were placed and returned true.

Function Name:
test_topLeft_to_bottomRight_checkDiagWin

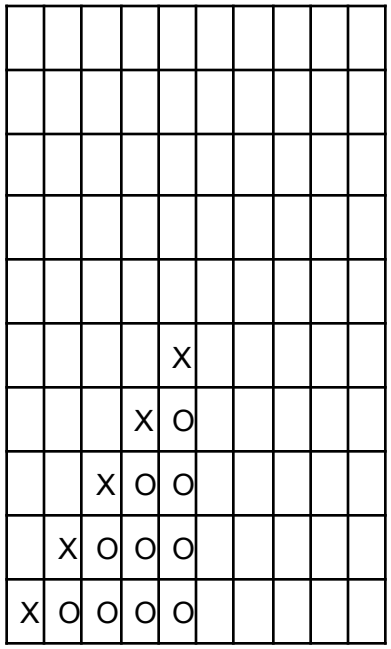
boolean checkDiagWin(BoardPosition pos, char p)

Input:

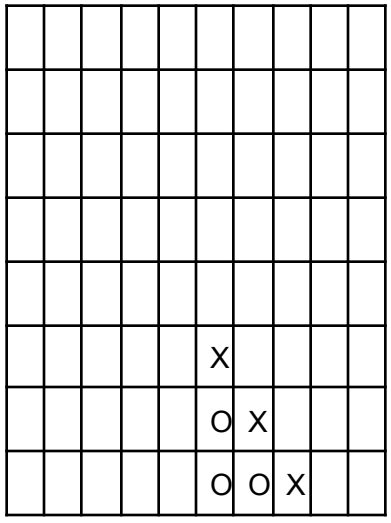
State: (number to win: 5)

Output:
checkDiagWin = true

Reason:
This test is unique because it tests checkDiagWin going

 pos.getRow = 2 pos.getCol = 2 p = 'X'	State of the board is unchanged	from the top right to the bottom left to make sure there are 5 consecutive tokens placed. In this case the last token was placed in the middle of the diagonal and the test verified that 5 consecutive tokens were placed and returned true. **Function Name:** test_bottomLeft_to_topRight_and_topRight_to_bottomLeft_middlePlacement_checkDiagWin

boolean checkDiagWin(BoardPosition pos, char p)

Input: State: (number to win: 5) 	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test is unique because it tests checkDiagWin going from the top left to the bottom right to make sure there are 5 consecutive tokens placed. In this case the last token was placed in the middle of the diagonal and the test verified that 5 consecutive tokens were placed and returned true. **Function Name:** test_topLeft_to_bottomRight_and_bottomRight_to_topLeft_middlePlacement_checkDiagWin

					O	O	O	X	
					O	O	O	O	X

pos.getRow = 2
pos.getCol = 7
p = ‘X’

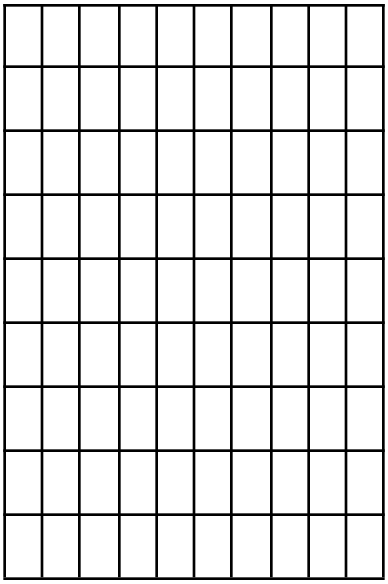
boolean checkTie()

Input:	**Output:**	**Reason:**
State: (number to win: 5)	checkTie = true State of the board is unchanged	This test is unique because it tests checkTie to see if there is a tie. In this case, there are no available spots for a token to be placed. Because there is nowhere to place a token the test returns true because there is a tie. **Function Name:** test_when_there_is_tie_checkTie

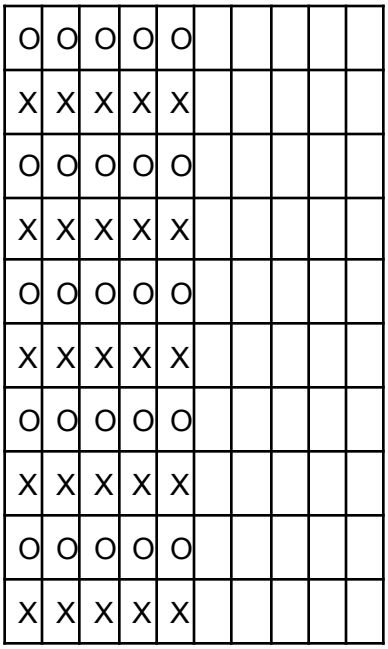
O	O	O	O	O	O	O	O	O	O
X	X	X	X	X	X	X	X	X	X
O	O	O	O	O	O	O	O	O	O
X	X	X	X	X	X	X	X	X	X
O	O	O	O	O	O	O	O	O	O
X	X	X	X	X	X	X	X	X	X
O	O	O	O	O	O	O	O	O	O
X	X	X	X	X	X	X	X	X	X
O	O	O	O	O	O	O	O	O	O
X	X	X	X	X	X	X	X	X	X

boolean checkTie()

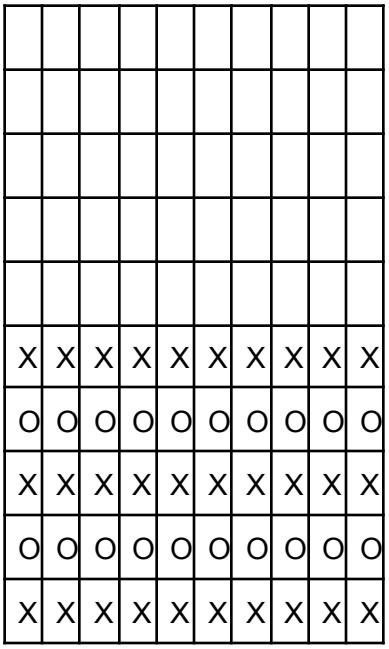
Input:	**Output:**	**Reason:**
State: (number to win: 5)	checkTie = false State of the board is unchanged	This test is unique because it tests checkTie to see if there is a tie. In this case, there are available spots for a token to be placed. Because you can place a token anywhere the test returns false because

 pos.getRow = 0 pos.getCol = 9 p = 'X'		there is no tie. **Function Name:** test_empty_board_checkTie

boolean checkTie()

Input: State: (number to win: 5) 	**Output:** checkTie = false State of the board is unchanged	**Reason:** This test is unique because it tests checkTie to see if there is a tie. In this case, only half of the columns are taken up so there are available spots for a token to be placed. Because you can place a token on the right side of the board the test returns false because there is no tie. **Function Name:** test_half_full_columns_check Tie

boolean checkTie()

Input: State: (number to win: 5) 	**Output:** checkTie = false State of the board is unchanged	**Reason:** This test is unique because it tests checkTie to see if there is a tie. In this case, only half of the rows are taken up so there are available spots for a token to be placed. Because you can place a token on the top side of the board the test returns false because there is no tie. **Function Name:** test_half_full_rows_checkTie