CPSC 2150 HW5 Writeup

Rex Oliver

April 24 2019

1 Requirements Analysis

Functional Requirements:

- As a user, I can choose how many rows should be on the board.
- As a user, I can choose how many columns should be on the board.
- As a user, I can choose how many in a row to win.
- As a user, I can choose what column to place my marker in.
- As a user, I can choose to play again or not.
- As a user, I can view the prompt that asks the user to make a move.
- As a user, I can view the prompt that asks the user to play again or not.
- As a user, I can view the prompt that asks the user how many rows should be on the board.
- As a user, I can view the prompt that asks the user how many column should be on the board.
- As a user, I can view the prompt that asks the user how many in a row to win?
- As a user, I can view the printed board with updated moves each turn.
- As a user, I can be notified of a win.
- As a user, I will not lose a turn for bad input.
- As a user, I can view the prompt that asks the user how many players to play the game.
- As a user, I can choose how many players to play in the game.

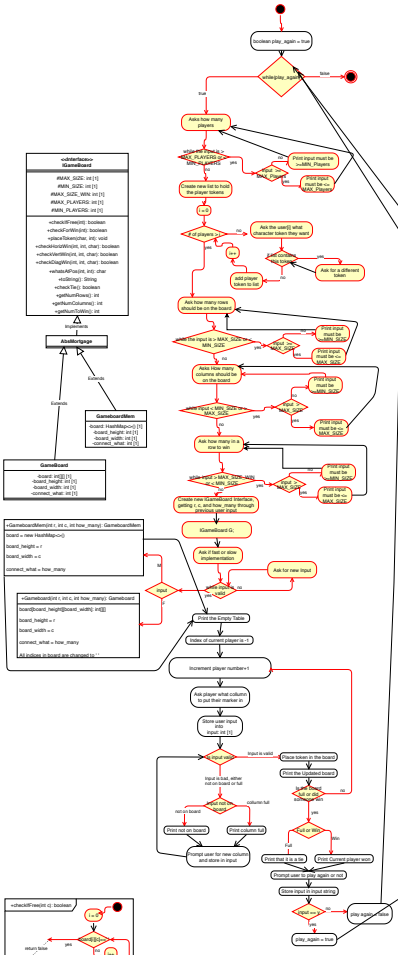
Non-functional Requirements

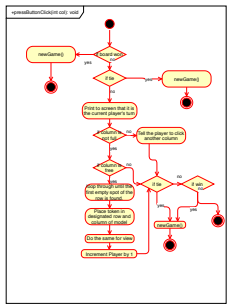
- The system must run on the School of Computing Linux Systems.
- The system must be written in Java.
- The system uses a GUI.

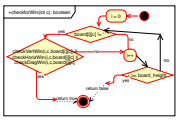
2 Design

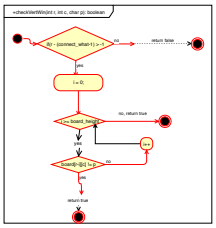
UML Diagram on next page, with the test cases listed right after.

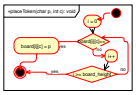
This beginning point is the documentation of the old main function in ConnectKGame

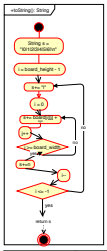


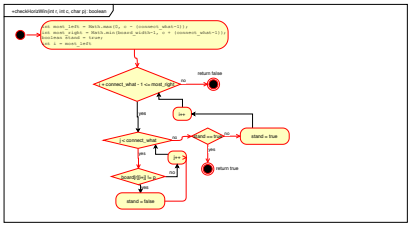


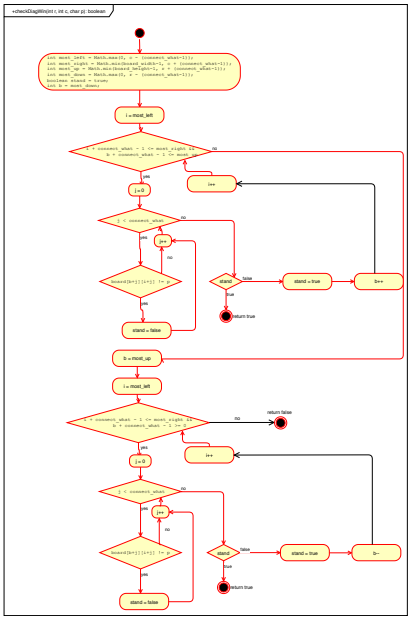


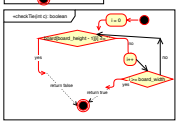


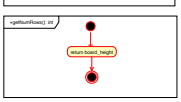


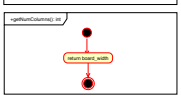


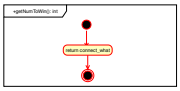




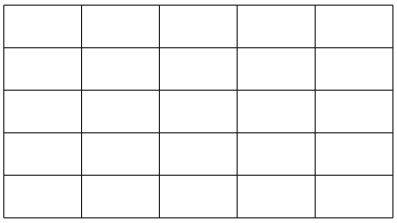




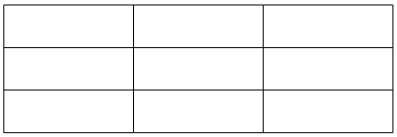




GameBoard(int r, int c, int to_win)

Input:	Output: GameBoard Object	Reason:
State: Gameboard not Initialized r: 5 c: 5 to_win: 3	State: GameBoard Initialized 	This test case is unique because it tests a value that is not a minimum or maximum size. It also checks the GameBoard object that is used for every other test case besides the ones that test the constructor Function: Constructor_NORM

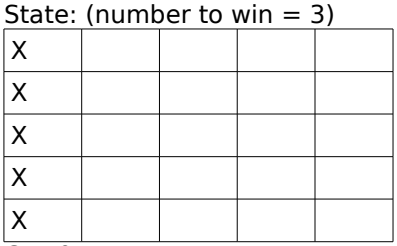
GameBoard(int r, int c, int to_win)

Input:	Output: GameBoard Object	Reason:
State: Gameboard not Initialized r: 3 c: 3 to_win: 3	State: GameBoard Initialized 	This test case is unique because it tests the minimum values the constructor can make a table with. Function: Constructor_MIN

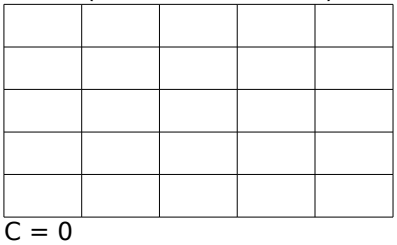
GameBoard(int r, int c, int to_win)

Input:	Output:	Reason:
State: Gameboard not Initialized r: 100 c: 100 to_win: 25	GameBoard Object State: GameBoard Initialized A GameBoard of 100x100 spaces unfortunately cannot be represented visually here, but that is what is created.	This test case is unique because it tests the maximum values the constructor can make a table with Function: Constructor_MAX

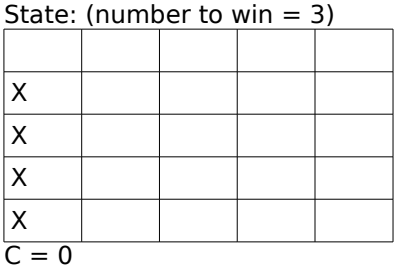
boolean checkIfFree(int c)

Input:	Output: false	Reason:
State: (number to win = 3)  C = 0	State: state of board is unchanged	This test case is unique because it tests checkIfFree on a full column and is designed to returned false Function: checkIfFree_column_full

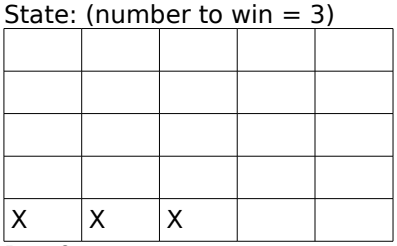
boolean checkIfFree(int c)

Input:	Output: true	Reason:
State: (number to win = 3)  C = 0	State: state of board is unchanged	This test case is unique because it tests checkifFree on an empty column, and therefore should return true Function: checkIfFree_column_empty

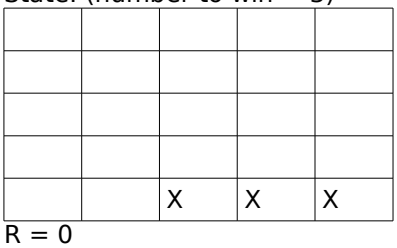
boolean checkIfFree(int c)

Input: State: (number to win = 3)  C = 0	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkIfFree on a partially full column rather than a full column or empty one Function: checkIfFree_column_partially_full

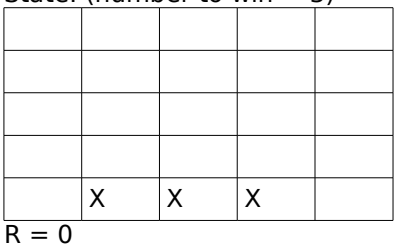
boolean checkHorizWin(int r, int c, char p)

Input: State: (number to win = 3)  R = 0 c = 0 p = ‘X	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkHorizWin from the beginning of a valid horizontal as opposed to the middle or end Function: checkHorizWin_row0_beginning

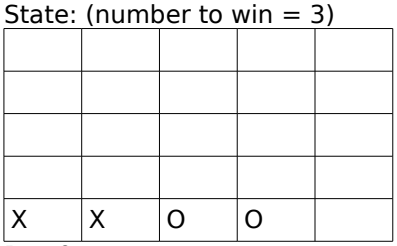
boolean checkHorizWin(int r, int c, char p)

Input: State: (number to win = 3)  R = 0 c = 4 p = ‘X’	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkHorizWin from the end of a valid horizontal as opposed to the middle or beginning Function: checkHorizWin_row0_end

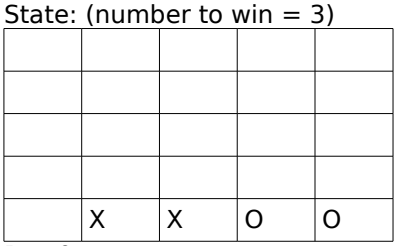
boolean checkHorizWin(int r, int c, char p)

Input: State: (number to win = 3)  R = 0 c = 2 p = ‘X’	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkHorizWin from the middle of a valid horizontal as opposed to the end or beginning Function: checkHorizWin_row0_middle

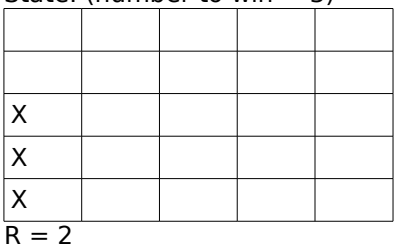
boolean checkHorizWin(int r, int c, char p)

Input: State: (number to win = 3)  R = 0 c = 0 p = X	Output: false State: state of board is unchanged	Reason: This test case is unique because it tests checkHorizWin from the beginning of an invalid horizontal as opposed to the middle or end Function: checkHorizWin_row0_false_beginning

boolean checkHorizWin(int r, int c, char p)

Input: State: (number to win = 3)  R = 0 c = 4 p = X	Output: false State: state of board is unchanged	Reason: This test case is unique because it tests checkHorizWin from the end of an invalid horizontal as opposed to the middle or beginning Function: checkHorizWin_row0_false_end

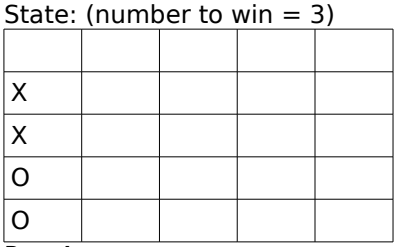
boolean checkVertWin(int r, int c, char p)

Input: State: (number to win = 3)  R = 2 c = 0 p = X	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkVertWin on a column with only the winning characters inside, as opposed to some below it Function: checkVertWin_top

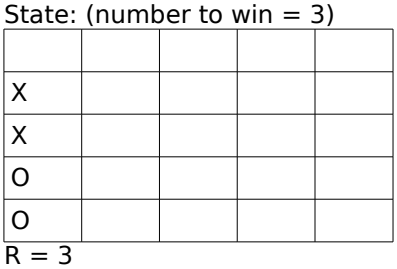
boolean checkVertWin(int r, int c, char p)

Input: State: (number to win = 3)  R = 2 c = 0 p = X	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkVertWin on a winning column with both winning and nonwinning characters inside, as opposed to exclusively winning chars Function: checkVertWin_top_spots_below

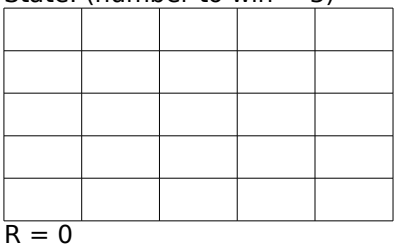
boolean checkVertWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3)  R = 4 c = 0 p = X	Output: false State: state of board is unchanged	This test case is unique because it tests checkVertWin on a non winning column with different characters inside from an empty spot above the highest char in the column, as opposed to exclusively one type of char Function: checkVertWin_false_empty_spots_below

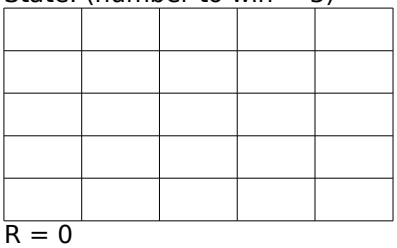
boolean checkVertWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3)  R = 3 c = 0 p = X	Output: false State: state of board is unchanged	This test case is unique because it tests checkVertWin on a non winning column with different characters inside, as opposed to exclusively one type of char Function: checkVertWin_false_top_spots_below

boolean checkVertWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3)  R = 0 c = 0 p = X	Output: false State: state of board is unchanged	This test case is unique because it tests checkVertWin on an empty board, and is trying to make sure it does not check indices that do not exist, causing null ptr exception Function: checkVertWin_false_empty_board

boolean checkDiagWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3)  R = 0 c = 0 p = X	Output: false State: state of board is unchanged	This test case is unique because it tests checkDiagWin on an empty board, and is trying to make sure it does not check indices that do not exist, causing null ptr exception Function: checkDiagWin_false_empty_board

boolean checkDiagWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3) R = 2 c = 2 p = X	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkDiagWin on a diagonal where the left is the bottom and the right side is the top, as opposed to the opposite way of right bottom left top Function: checkDiagWin_left_bottom_right_top

		X		
	X	X		
X	X	X		

boolean checkDiagWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3) R = 2 c = 0 p = X	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkDiagWin on a diagonal where the left is the top and the right side is the bottom, as opposed to the opposite way of right top left bottom Function: checkDiagWin_left_top_right_bottom

X				
X	X			
X	X	X		

boolean checkDiagWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3) R = 4 c = 4 p = X	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkDiagWin on a diagonal where the winning characters have a pattern of left is the bottom and the right side is the top with non winning characters below, as opposed to the opposite way of right bottom left top Function: checkDiagWin_left_bottom_right_top_spots_under

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

boolean checkDiagWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3) R = 4 c = 0 p = X	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkDiagWin on a diagonal where the winning characters have a pattern of left is the top and the right side is the bottom with non winning characters below, as opposed to the opposite way of right bottom left top Function: checkDiagWin_left_top_right_bottom_spots_under

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

boolean checkDiagWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3) R = 0 c = 0 p = X	Output: false State: state of board is unchanged	This test case is unique because it tests checkDiagWin on a non winning diagonal where the left is the bottom and the right side is the top, as opposed to the opposite way of right bottom left top Function: checkDiagWin_left_bottom_right_top_insufficient_chars

	X			
X	X			

boolean checkDiagWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3) R = 1 c = 1 p = X	Output: false State: state of board is unchanged	This test case is unique because it tests checkDiagWin on a non winning diagonal where the left is the top and the right side is the bottom, as opposed to the opposite way of right top left bottom Function: checkDiagWin_false_left_top_right_bottom_insufficient_chars

	X			
	X	X		

boolean checkDiagWin(int r, int c, char p)

Input:	Output:	Reason:
State: (number to win = 3) R = 2 c = 2 p = O	Output: false State: state of board is unchanged	This test case is unique because it tests checkDiagWin on a board that is alternating, allowing it to test all diagonals negatively, making sure that there are no false positives for these cases Function: checkDiagWin_full_alternating_board

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

boolean checkTie()

Input:	Output:	Reason:
State: (number to win = 3)	Output: true State: state of board is unchanged	This test case is unique because it tests checkTie on a board that is full as opposed to empty or partially full Function: checkTie_full_board

X	X	X	X	X
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X

boolean checkTie()

Input: State: (number to win = 3)	Output: false State: state of board is unchanged	Reason: This test case is unique because it tests checkTie on an empty board as opposed to full or partially full Function: checkTie_empty_board

boolean checkTie()

Input: State: (number to win = 3)	Output: false State: state of board is unchanged	Reason: This test case is unique because it tests checkTie on a board where only some of the columns are full as opposed to some of the rows full or entire board empty or full Function: checkTie_some_columns_full

boolean checkTie()

Input: State: (number to win = 3)	Output: true State: state of board is unchanged	Reason: This test case is unique because it tests checkTie on a full, alternating character board as opposed to a non-alternating full board, partially full, or empty board Function: checkTie_full_alternating_board

char whatsAtPos(int r, int c)

Input: State: (number to win = 3) R = 0 c = 0	Output: ‘ ‘ State: state of board is unchanged	Reason: This test case is unique because it tests WhatsAtPos on an empty space on an empty board as opposed to full or partially full Function: WhatsAtPos_empty_space_empty_board

char whatsAtPos(int r, int c)

Input: State: (number to win = 3) R = 1 c = 0	Output: ‘ ‘ State: state of board is unchanged	Reason: This test case is unique because it tests WhatsAtPos on an empty space on an partially full board as opposed to full or empty one Function: WhatsAtPos_empty_space_part_full_board

char whatsAtPos(int r, int c)

Input: State: (number to win = 3) X X X X X X X X X X X X X X X X X X X X X X X X R = 4 c = 4	Output: ‘ ‘ State: state of board is unchanged	Reason: This test case is unique because it tests WhatsAtPos on the only empty space on an nearly filled board as opposed to full or empty board Function: WhatsAtPos_empty_space_almost_full_board

char whatsAtPos(int r, int c)

Input: State: (number to win = 3) X R = 0 c = 3	Output: ‘X‘ State: state of board is unchanged	Reason: This test case is unique because it tests WhatsAtPos on the only non-empty space on the board as opposed to a completely full or empty board Function: WhatsAtPos_only_spot_on_board

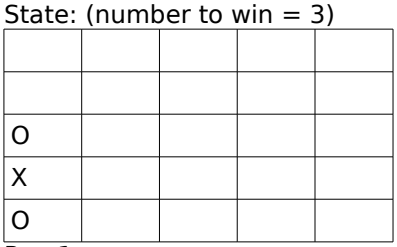
char whatsAtPos(int r, int c)

Input: State: (number to win = 3) O O O X O R = 0 c = 3	Output: ‘X‘ State: state of board is unchanged	Reason: This test case is unique because it tests WhatsAtPos on a different char from the rest of its row as opposed to the same one as the rest, making sure it is picking the right one on the row Function: WhatsAtPos_diff_char_than_rest_of_row

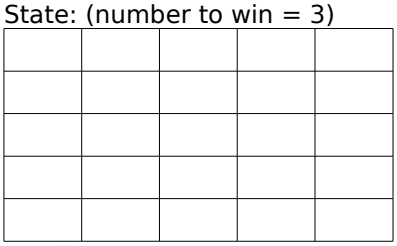
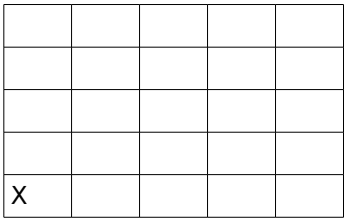
char whatsAtPos(int r, int c)

Input: State: (number to win = 3) O O O O O O O O O O O O X O O O O O O O O O O O O R = 2 c = 2	Output: ‘X‘ State: state of board is unchanged	Reason: This test case is unique because it tests WhatsAtPos on a different char from the rest of the entire board as opposed to the same one as the rest, making sure it is picking the right one on the board Function: WhatsAtPos_diff_char_than_rest_of_board

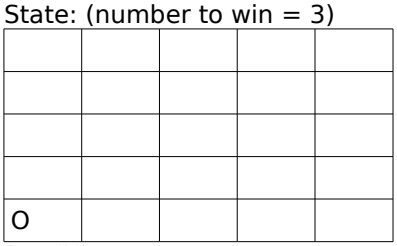
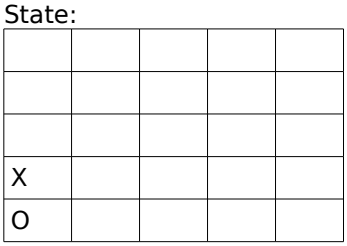
char whatsAtPos(int r, int c)

Input:	Output:	Reason:
State: (number to win = 3)  R = 1 c = 0	Output: 'X' State: state of board is unchanged	This test case is unique because it tests WhatsAtPos when there are different characters above and below it, making sure the function is picking the right character Function: WhatsAtPos_spots_taken_above_and_below

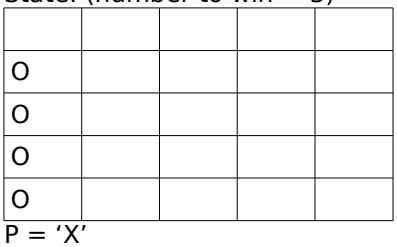
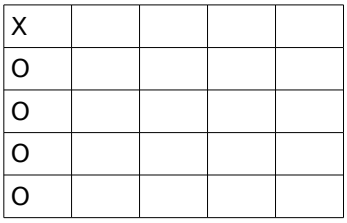
void placeToken(char p, int c)

Input:	Output:	Reason:
State: (number to win = 3)  p = X c = 0	Output: State: 	This test case is unique because it tests if placeToken can put a character on an empty board as opposed to a partially full one Function: placeToken_empty_board

void placeToken(char p, int c)

Input:	Output:	Reason:
State: (number to win = 3)  P = 'X' c = 0	Output: State: 	This test case is unique because it tests if placeToken can put a char on a column that is partially full, as opposed to empty or 1 away from full Function: placeToken_column_part_full

void placeToken(char p, int c)

Input:	Output:	Reason:
State: (number to win = 3)  P = 'X' c = 0	Output: State: 	This test case is unique because it tests if placeToken can put a char on a column that would completely fill the column as opposed to a column with nothing or only 1 or 2 chars inside Function: placeToken_to_fill_column

void placeToken(char p, int c)

Input:
State: (number to win = 3)

O	O	O	O	

P = ‘X’
c = 4

Output:
State:

O	O	O	O	X

Reason:
This test case is unique because it tests if placeToken can put a char on a column that would completely fill the row as opposed to a row with nothing or only 1 or 2 chars inside

Function: placeToken_to_fill_row

void placeToken(char p, int c)

Input:
State: (number to win = 3)

O	O	O	O	
O	O	O	O	O
O	O	O	O	O
O	O	O	O	O
O	O	O	O	O

P = ‘X’
c = 4

Output:
State:

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

Reason:
This test case is unique because it tests if placeToken can put a char on a column that would completely fill the board as opposed to a board with nothing or only 1 or 2 chars inside

Function: placeToken_to_fill_board

3 Deployment

Untar the provided HW5.tar.gz
Run the ConnectXApp.java