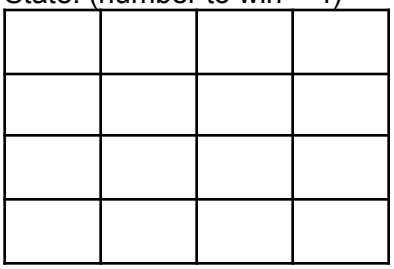
Test Cases for Project 4

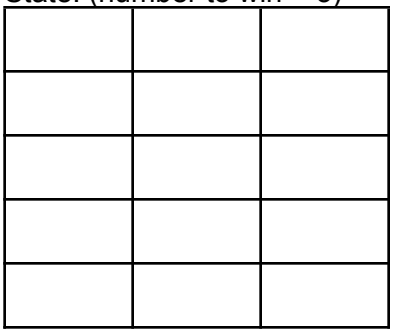
Evan Cox
Christian Herrman
Jonathan Maenche
Ben McDonnough

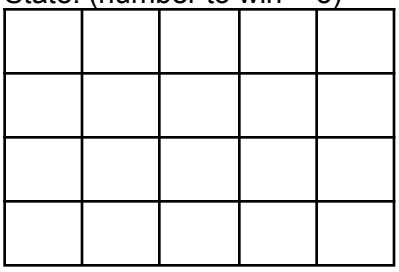
Test cases…

Constructor

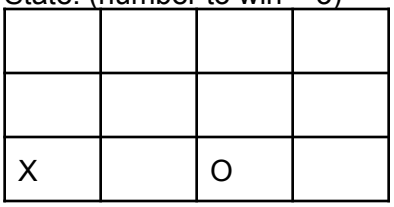
GameBoard(int row, int column, int numWin);

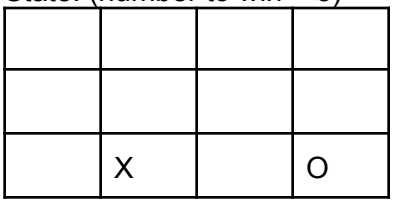
Input: State: (number to win = 4)  row = 4 column = 4 numWin = 4	**Output**: Returns GameBoard object with state of board initialized to be blank and number to win variable is set	**Reason**: This test case is unique because we are constructing a GameBoard object with the same row and column number AND because the number to win is equal to both the row and column number **Function name**: testConstructor_equal_row_column_numWin

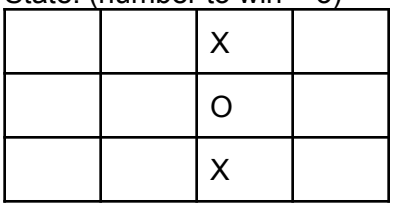
Input: State: (number to win = 3)  row = 5 column = 3 numWin = 3	**Output**: Returns GameBoard object with state of board initialized to be blank and number to win variable is set	**Reason**: This test case is unique because we are constructing a GameBoard object with the a row number that is larger than the column number AND because the number to win is equal to the column number **Function name**: testConstructor_larger_row_smaller_column

Input: State: (number to win = 3)  row = 4 column = 5 numWin = 3	**Output**: Returns GameBoard object with state of board initialized to be blank and number to win variable is set	**Reason**: This test case is unique because we are constructing a GameBoard object with the a row number that is smaller than the column number AND because the number to win is NOT equal to the column or the row number **Function name**: testConstructor_larger_column_ smaller_row_distinct_numWin

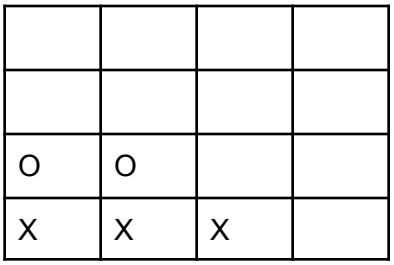
CheckIfFree
boolean checkIfFree(int c);

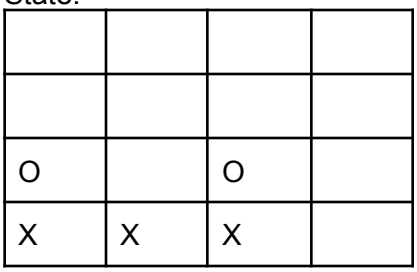
Input: State: (number to win = 3)  c = 1	**Output**: checkIfFree = true state of the board is unchanged	**Reason**: This test case is unique because we are calling checkIfFree on a column that is empty (it contains no tokens) **Function name**: testCheckIfFree_column_ empty_true

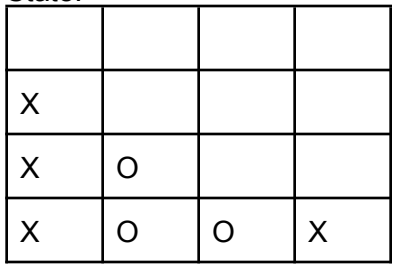
Input: State: (number to win = 3)  c = 1	**Output**: checkIfFree = true state of the board is unchanged	**Reason**: This test case is unique because we are calling checkIfFree on a column that is NOT empty but also is NOT full (it contains 1 token) **Function name**: testCheckIfFree_column_ has_1_token_true

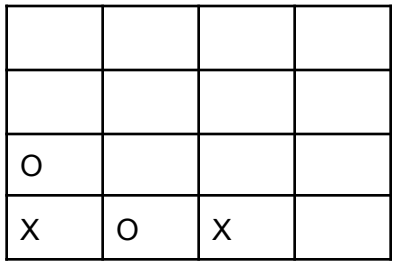
Input: State: (number to win = 3)  c = 2	**Output**: checkIfFree = false state of the board is unchanged	**Reason**: This test case is unique because we are calling checkIfFree on a column that is full (it contains 3 tokens) **Function name**: testCheckIfFree_column_full_false

CheckHorizWin
boolean checkHorizWin(BoardPosition pos, char p);

Input: State:  pos.getRow = 0 pos.getCol = 2 player = ‘X’	**Output**: checkHorizWin = true state of the board is unchanged	**Reason**: This test case is unique because we are calling checkHorizWin after X has placed their token at the right end of a string of 3 tokens, which is the number needed to win, meaning checkHorizWin needs to return true after checking the two tokens to the left. **Function name**: testCheckHorizWin_last_token_end_true

Input: State:  pos.getRow = 0 pos.getCol = 1 player = ‘X’	**Output**: checkHorizWin = true state of the board is unchanged	**Reason**: This test case is unique because we are calling checkHorizWin after X has placed their token at the in the middle of a string of 3 tokens, which is the number needed to win, meaning checkHorizWin needs to return true after checking the tokens to both the left and right. **Function name**: testCheckHorizWin_last_token_middle_true

Input: State:  pos.getRow = 2 pos.getCol = 0 player = ‘X’	**Output**: checkHorizWin = false state of the board is unchanged	**Reason**: This test case is unique because we are calling checkHorizWin after both X and O have placed various tokens, but neither have completed a horizontal win, so checkHorizWin should return false, even if a player has won in a direction other than horizontal. **Function name**: testCheckHorizWin_no_ horizontal_win_false

Input: State:  pos.getRow = 0 pos.getCol = 1 player = ‘O’	**Output**: checkHorizWin = false state of the board is unchanged	**Reason**: This test case is unique because we are calling checkHorizWin after player O has placed their token between two X tokens, preventing player X from creating a string of 3, which is the number needed to win the game, and this ensures that checkHorizWin works correctly with differentiating between player’s tokens. **Function name**: testCheckHorizWin_block_win_ false

CheckVertWin
boolean checkVertWin(BoardPosition pos, char p);

Input:

State:

X			
X	O		
X	O		

pos.getRow = 2
pos.getCol = 0
p = 'X'

Output:
checkVertWin = true

state of the board is unchanged

Reason: This test case is unique because we are calling checkVertWin after player X has placed enough tokens in a row to get a vertical win to ensure that the function checkVertWin can correctly identify a win.

Function name:

testCheckVertWin_min_to_win_X_true

Input:

State:

X			
X			
O			
X	O		

pos.getRow = 3
pos.getCol = 0
p = 'X'

Output:
checkVertWin = false

state of the board is unchanged

Reason: This test case is unique because we are calling checkVertWin after player O has blocked player X from winning which ensures that checkVertWin is checking the token that was dropped against the tokens around it when looking for a win

Function name:

testCheckVertWin_blocked_win_false

Input:
State:

O	O	O	
X	X	X	

pos.getRow = 0
pos.getCol = 3
p = 'X'

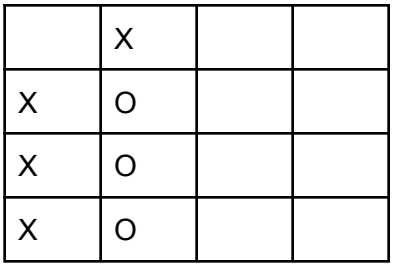
Output:
checkVertWin = false

state of the board is unchanged

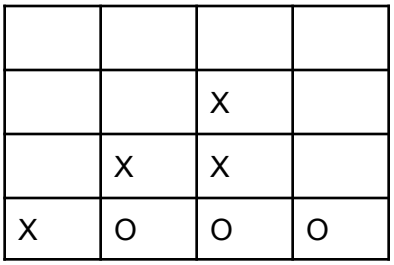
Reason: This test case is unique because we are calling checkVertWin after both X and O have placed various tokens, but neither have completed a vertical win, so checkVertWin should return false, even if a player has won in a direction other than vertical.

Function name:

testCheckVertWin_no_vert_win_false

Input: State:  pos.getRow = 1 pos.getCol = 3 p = ‘X’	**Output**: checkVertWin = false state of the board is unchanged	**Reason**: This test case is unique because we are calling checkVertWin after both X and O have both technically won, but the last token was not a win. This ensures that checkVertWin is actually checking the LAST token for a win instead of any win on the board. **Function name**: testCheckVertWin_check_last_token_false

CheckDiagWin
boolean checkDiagWin(BoardPosition pos, char p);

Input: State:  pos.getRow = 2 pos.getCol = 2 p = ‘X’	**Output**: checkDiagWin = true state of the board is unchanged	**Reason**: This test case is unique because we are calling checkDiagWin after X has won diagonally up-and-right with the last token being placed at the top right of the string. **Function name**: testCheckDiagWin_check_last_token_up_right_true

Input: State: <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td>X</td><td>X</td><td></td></tr><tr><td>O</td><td>O</td><td>O</td><td>X</td></tr></table> pos.getRow = 2 pos.getCol = 1 p = ‘X’	**Output**: checkDiagWin = true state of the board is unchanged	**Reason**: This test case is unique because we are calling checkDiagWin after X has won diagonally up-and-left with the last token being placed at the top left of the string. **Function name**: testCheckDiagWin_check_last_token_up_left_true
Input: State: <table><tr><td></td><td></td><td>O</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></td><td>X</td><td>O</td><td>X</td></tr></table> pos.getRow = 1 pos.getCol = 1 p = ‘X’	**Output**: checkDiagWin = false state of the board is unchanged	**Reason**: This test case is unique because we are calling checkDiagWin after X has placed more than 1 token in the up-and-right diagonal, but does not have a long enough string to win. **Function name**: testCheckDiagWin_not_up_and_right_false
Input: State: <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td>O</td><td>X</td><td></td></tr><tr><td></td><td>X</td><td>O</td><td></td></tr></table> pos.getRow = 2 pos.getCol = 1 p = ‘X’	**Output**: checkDiagWin = false state of the board is unchanged	**Reason**: This test case is unique because we are calling checkDiagWin after X has placed more than 1 token in the up-and-left diagonal, but does not have a long enough string to win. **Function name**: testCheckDiagWin_not_up_and_left_false

Input:

State:

		X	
	X	O	
X	O	O	X

pos.getRow = 1
pos.getCol = 1
p = ‘X’

Output:
checkDiagWin = true

state of the board is unchanged

Reason: This test case is unique because we are calling checkDiagWin after X has won diagonally up-and-right with the last token being placed in the middle of the winning string.

Function name:

testCheckDiagWin_last_token_middle_up_and_right_true

Input:

State:

	X		
	O	X	
X	O	O	X

pos.getRow = 1
pos.getCol = 2
p = ‘X’

Output:
checkDiagWin = true

state of the board is unchanged

Reason: This test case is unique because we are calling checkDiagWin after X has won diagonally up-and-left with the last token being placed in the middle of the winning string.

Function name:

testCheckDiagWin_last_token_middle_up_and_left_true

Input:

State:

X		X	O
O	X	O	X

pos.getRow = 0
pos.getCol = 1
p = ‘X’

Output:
checkDiagWin = false

state of the board is unchanged

Reason: This test case is unique because we are calling checkDiagWin after X has placed tokens in both the up-and-left and up-and-right diagonals totaling enough to win if they were in a single diagonal, but not enough in either single diagonal alone.

Function name:

testCheckDiagWin_different_diagonals_false

CheckTie
boolean checkTie();

Input: State: O O X X X X O O O O X X X X O O	**Output**: checkTie = true state of the board is unchanged	**Reason**: This test case is unique because it's a full board where neither player actually wins. **Function name**: testCheckTie_full_board_true

Input: State: O O X X X X O O	**Output**: checkTie = false state of the board is unchanged	**Reason**: This test case is unique because it's a non-full board where none of the columns are full **Function name**: testCheckTie_half_full_board_false

Input: State: O X O X	**Output**: checkTie = false state of the board is unchanged	**Reason**: This test case is unique because it's a non-full board where only one of the columns is full **Function name**: testCheckTie_single_full_column_false

Input: State: O O X X X O O O O X X X X O O	**Output**: checkTie = false state of the board is unchanged	**Reason**: This test case is unique because it's a non-full board where all but one of the columns is full **Function name**: testCheckTie_all_columns_but_one_full_false

WhatsAtPos
char whatsAtPos(BoardPosition pos);

Input: State: X pos.getRow = 0 pos.getCol = 0	**Output**: whatsAtPos = 'X' state of the board is unchanged	**Reason**: This test case is unique because it tests the function with the bottom left corner, which is a boundary case. **Function name**: testWhatsAtPos_bottom_left_corner

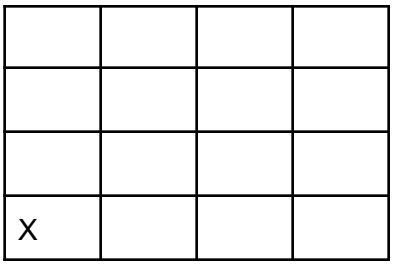
Input: State: X pos.getRow = 0 pos.getCol = 3	**Output**: whatsAtPos = 'X' state of the board is unchanged	**Reason**: This test case is unique because it tests the function with the bottom right corner, which is a boundary case. **Function name**: testWhatsAtPos_bottom_right_corner

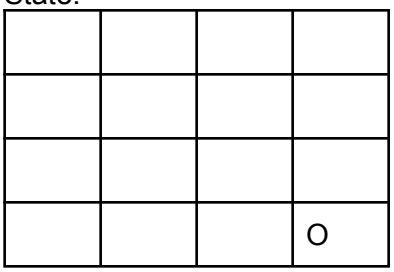
Input: State: O | | | | X | | | | O | | | | X | | | | pos.getRow = 3 pos.getCol = 0	**Output**: whatsAtPos = 'O' state of the board is unchanged	**Reason**: This test case is unique because it tests the function with the top left corner, which is a boundary case. **Function name**: testWhatsAtPos_top_left_corner

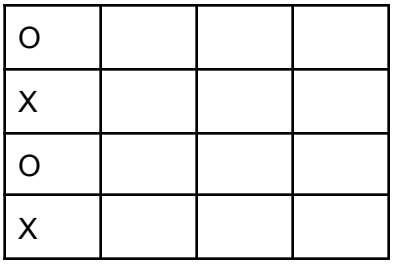
Input: State: | | | | O | | | | X | | | | O | | | | X pos.getRow = 3 pos.getCol = 3	**Output**: whatsAtPos = 'O' state of the board is unchanged	**Reason**: This test case is unique because it tests the function with the top right corner, which is a boundary case. **Function name**: testWhatsAtPos_top_right_corner

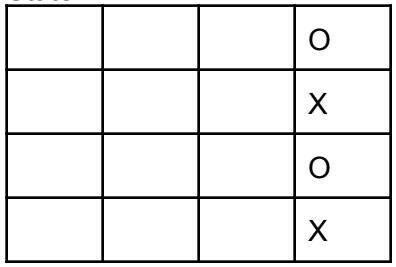
Input: State: | | | | | | | | | | | | O | | | | X | | pos.getRow = 1 pos.getCol = 2	**Output**: whatsAtPos = 'O' state of the board is unchanged	**Reason**: This test case is unique because it tests the function with one of the inside cells on a board. **Function name**: testWhatsAtPos_position_on_inside_of_board

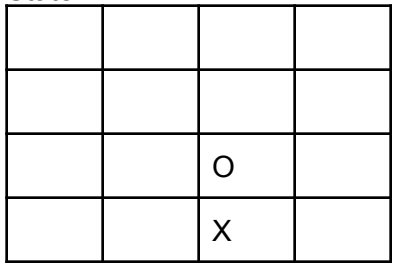
isPlayerAtPos
boolean isPlayerAtPos(BoardPosition pos, char player);

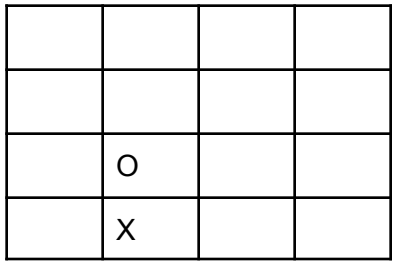
Input: State:  pos.getRow = 0 pos.getCol = 0 player = 'X'	**Output**: isPlayerAtPos = true state of the board is unchanged	**Reason**: This test case is unique because it tests the function with the bottom left corner, which is a boundary case. It is also unique because it returns true. **Function name**: testIsPlayerAtPos_bottom_left_corner_true

Input: State:  pos.getRow = 0 pos.getCol = 3 player = 'X'	**Output**: isPlayerAtPos = false state of the board is unchanged	**Reason**: This test case is unique because it tests the function with the bottom right corner, which is a boundary case. It is also unique because it returns false. **Function name**: testIsPlayerAtPos_bottom_right_corner_false

Input: State:  pos.getRow = 3 pos.getCol = 0 player = 'X'	**Output**: isPlayerAtPos = false state of the board is unchanged	**Reason**: This test case is unique because it tests the function with the top left corner, which is a boundary case. It is also unique because it returns false. **Function name**: testIsPlayerAtPos_top_left_corner_false

Input: State:  pos.getRow = 3 pos.getCol = 3 player = ‘O’	**Output**: isPlayerAtPos = true state of the board is unchanged	**Reason**: This test case is unique because it tests the function with the top right corner, which is a boundary case. It is also unique because it returns true. **Function name**: testIsPlayerAtPos_top_right_corner_true

Input: State:  pos.getRow = 1 pos.getCol = 2 player = ‘O’	**Output**: isPlayerAtPos = true state of the board is unchanged	**Reason**: This test case is unique because it tests the function with one of the inside cells on a board. It is also unique because it returns true. **Function name**: testIsPlayerAtPos_position_on_inside_of_board_true

Input: State:  pos.getRow = 1 pos.getCol = 1 player = ‘X’	**Output**: isPlayerAtPos = false state of the board is unchanged	**Reason**: This test case is unique because it tests the function with one of the inside cells on a board. It is also unique because it returns false. **Function name**: testIsPlayerAtPos_position_on_inside_of_board_false

DropToken
void dropToken(char player, int column);

Input:

State:

player = 'X'
column = 0

Output:

State:

X			

Reason: This test case is unique because it starts with an empty board

Function name:

testDropToken_empty_board

Input:

State:

		X	
X	O	X	

player = 'X'
column = 0

Output:

State:

X		X	
X	O	X	

Reason: This test case is unique because it drops a token on top of another token (from the same player)

Function name:

testDropToken_drop_onto_another_token_same_player

Input:

State:

		X	
X	O	X	

player = 'X'
column = 1

Output:

State:

	X	X	
X	O	X	

Reason: This test case is unique because it drops a token on top of another token (from a different player)

Function name:

testDropToken_drop_onto_another_token_diff_player

Input:

State:

O			
X			
X			

player = ‘O’
column = 0

Output:

State:

O			
O			
X			
X			

Reason: This test case is unique because it fills up a column

Function name:

testDropToken_fill_column

Input:

State:

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

player = ‘X’
column = 3

Output:

State:

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

Reason: This test case is unique because it fills up the entire board

Function name:

testDropToken_fill_board