Project 4 Test Cases
(GameBoard & GameBoardMem)

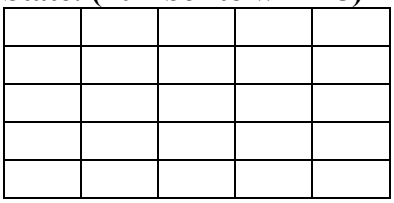
`Constructor(int inRow, int inColumn, int inNumToWin)`

Input:	Output:	Reason:
inRow = 3 inColumn = 3 inNumToWin = 3	**State: (number to win = 3)** **BOARD IS 3x3 and empty**	This test is unique because it tests that the constructor is able to correctly initialize an empty board to the minimum dimensions possible. **Function Name:** testConstructor_3_3_3_min_dimensions

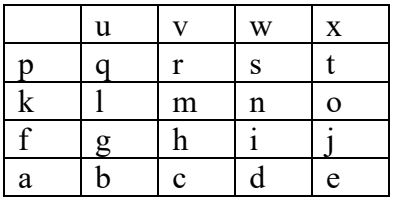
`Constructor(int inRow, int inColumn, int inNumToWin)`

Input:	Output:	Reason:
inRow = 100 inColumn = 100 inNumToWin = 25	**State: (number to win = 25)** **BOARD IS 100x100 and empty**	This test is unique because it tests that the constructor is able to correctly initialize an empty board to the maximum dimensions possible. **Function Name:** testConstructor_100_100_25_max_dimensions

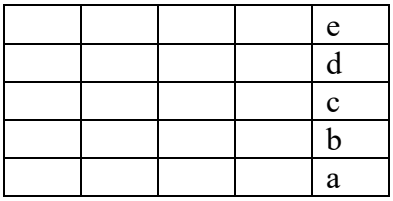
`Constructor(int inRow, int inColumn, int inNumToWin)`

Input:	Output:	Reason:
inRow = 9 inColumn = 7 inNumToWin = 5	**State: (number to win = 5)**  **BOARD IS 9x7 and empty**	This test is unique because it tests that the constructor can correctly initialize an empty board to dimensions containing different # of cols and rows **Function Name:** testConstructor_9_7_5_diff_dimensions

`boolean checkIfFree(int c)`

Input:	Output:	Reason:
State:  c = 0	checkIfFree = true state of the board is unchanged	This test is unique because the left most column is almost full and all other columns are full. **Function Name:** testCheckIFFree_board_full_except_left_column

`boolean checkIfFree(int c)`

Input:	Output:	Reason:
State:  c = 4	checkIfFree = false state of the board is unchanged	This test is unique because the column is full and no other columns are almost full. **Function Name:** testCheckIfFree_column_full_false

`boolean checkIfFree(int c)`

Input: State: u v w x p q r s t k l m n o f g h i j a b c d e c = 4	Output: checkIfFree = true state of the board is unchanged	Reason: This test is unique because the right most column is almost full and all other columns are full. Function Name: testCheckIfFree_board_full_except_right_column

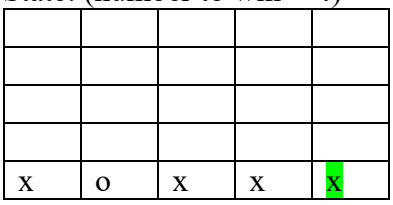
`boolean checkHorizWin(BoardPosition pos, char p)`

Input: State: (number to win = 4) x x x x m n o p i j k l e f g h a b c d pos.getRow = 4 pos.getCol = 0 p = 'x'	Output: checkHorizWin = true state of the board is unchanged	Reason: This test case is unique because the last 'x' is placed on far left of string of x's. The function must only count to the right. Function Name: testCheckHorizWin_last_placed_top_left_true

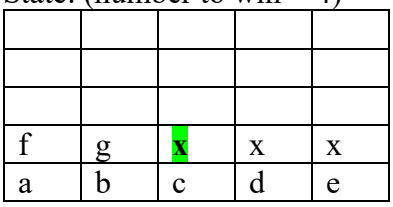
`boolean checkHorizWin(BoardPosition pos, char p)`

Input: State: (number to win = 4) x x x x e f g h a b c d pos.getRow = 2 pos.getCol = 2 p = 'x'	Output: checkHorizWin = true state of the board is unchanged	Reason: This test case is unique because the last x is placed in the middle of other x's, so the function must count on both sides of placed x. Function Name: testCheckHorizWin_last_placed_middle_true

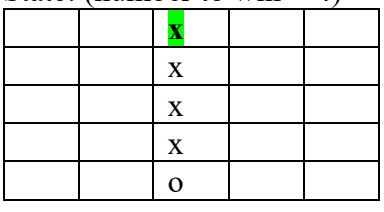
boolean checkHorizWin(BoardPosition pos, char p)

Input: State: (number to win = 4)  pos.getRow = 0 pos.getCol = 4 p = 'x'	Output: checkHorizWin = false state of the board is unchanged	Reason: This test is unique because the last x is placed on the right-hand side and function must count only to the left, until it runs into a 'o' character. **Function Name:** testCheckHorizWin_ last_placed_bottom _right_ false

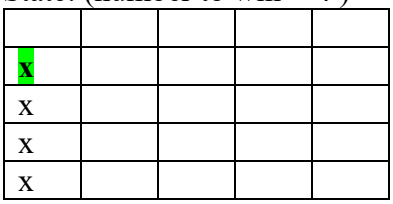
boolean checkHorizWin(BoardPosition pos, char p)

Input: State: (number to win = 4)  pos.getRow = 1 pos.getCol = 2 p = 'x'	Output: checkHorizWin = false state of the board is unchanged	Reason: This test is unique because the last 'x' is placed on left-hand side of string of x's and the function must count only to the right until it hits end of board. **Function Name:** testCheckHorizWin_last _placed_left_false

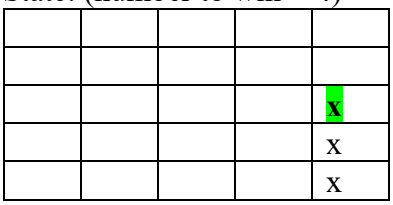
boolean checkVertWin(BoardPosition pos, char p)

Input: State: (number to win = 4)  pos.getRow = 4 pos.getCol = 2 p = 'x'	Output: checkVertWin = true state of the board is unchanged	Reason: This test is unique because there is an 'o' below string of x's and final 'x' is on top row. **Function Name:** testCheckVertWin_middle last_placed_in_top_row_ true

boolean checkVertWin(BoardPosition pos, char p)

Input:	Output:	Reason:
State: (number to win = 4)  pos.getRow = 3 pos.getCol = 0 p = 'x'	checkVertWin = true state of the board is unchanged	This test is unique because the last 'x' is not placed in top row of board and the string of x's are located at left-side of board. **Function Name:** testCheckVertWin_left_last_not_in_top_row_true

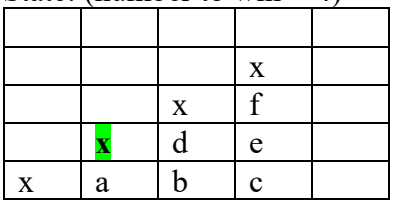
boolean checkVertWin(BoardPosition pos, char p)

Input:	Output:	Reason:
State: (number to win = 4)  pos.getRow = 2 pos.getCol = 4 p = 'x'	checkVertWin = true state of the board is unchanged	This test is unique because the string of x's are located at right-side of the board and the function must count down until reaching bottom of board. **Function Name:** testCheckVertWin_right_false

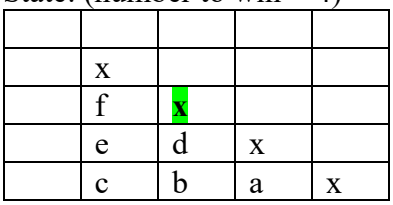
boolean checkVertWin(BoardPosition pos, char p)

Input:	Output:	Reason:
State: (number to win = 4)  pos.getRow = 4 pos.getCol = 0 p = 'x'	checkVertWin = false state of the board is unchanged	This test is unique because the function must count down from last 'x' placed until it hits an 'o' character and last x is placed in top row. **Function Name:** testCheckVertWin_left_last_placed_in_top_row_false

`boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 4)  pos.getRow = 1 pos.getCol = 1 p = ‘x‘	Output: checkDiagWin = true state of the board is unchanged	Reason: This test is unique because the last token is placed in the middle of the string of x’s so the function must count in both directions and it is testing positive diagonal. Function Name: testCheckDiagWin_last_placed_in_middle_pos_diag

`boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 4)  pos.getRow = 2 pos.getCol = 2 p = ‘x’	Output: checkDiagWin = true state of the board is unchanged	Reason: This test is unique because the last token is placed in the middle of the string of x’s so the function must count in both directions and it is testing negative diagonal. Function Name: testCheckDiagWin_last_placed_in_middle_neg_diag

boolean checkDiagWin(BoardPosition pos, char p)

Input:	Output:	Reason:
State: (number to win = 4) |		|				x | |			x | k | |		x | i | j | |	x | f | g | h | | a | b | c | d | e | pos.getRow = 4 pos.getCol = 4 p = 'x '	checkDiagWin = true state of the board is unchanged	This test is unique because the last token is placed in top right and is testing the positive diagonal. The function must count down and to the left. Function Name: testCheckDiagWin_last_last_placed_in_top_right

boolean checkDiagWin(BoardPosition pos, char p)

Input:	Output:	Reason:
State: (number to win = 4) | x |				| | k | x |		| | j | i | x |	| | h | g | f | x | | | e | d | c | b | a | pos.getRow = 4 pos.getCol = 0 p = 'x'	checkDiagWin = true state of the board is unchanged	This test is unique because the last token is placed in top left and it tests the negative diagonal. The function must count down and to the right Function Name: testCheckDiagWin_last_placed_in_top_left

boolean checkDiagWin(BoardPosition pos, char p)

Input:	Output:	Reason:
State: (number to win = 4) |				| |			x | | |		x | f | | |	x | d | e | | | x | a | b | c | | pos.getRow = 0 pos.getCol = 0 p = 'x'	checkDiagWin = true state of the board is unchanged	This test is unique because the last token is placed in bottom left, and it tests the positive diagonal. The function must count up and to the right. Function Name: testCheckDiagWin_last_placed_in_bottom_left

`boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 4) 				
	x			
	f	x		
	e	d	x	
	c	b	a	**x**

pos.getRow = 0
pos.getCol = 4
p = 'x'

Output: checkDiagWin = true state of the board is unchanged	**Reason:** This test is unique because the last token is placed in the bottom right and it is testing the negative diagonal. The function must count up and to the left. **Function Name:** testCheckDiagWin_last_placed_in_bottom_right

`boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 4) 				
x				
l	x			
k	j	x		
i	h	g	f	
e	d	c	b	a

pos.getRow = 4
pos.getCol = 0
p = 'x'

Output: checkDiagWin = false state of the board is unchanged	**Reason:** This test is unique because the last token is placed in top left and the function must count down and to the right until an 'o' is next. **Function Name:** testCheckDiagWin_last_placed_in_top_left_false

`boolean checkTie()`

Input: State: u v w x p q r s t k l m n o f g h i j a b c d e	Output: checkTie = false state of the board is unchanged	Reason: This test is unique because the entire board is full except for the top left spot. **Function Name:** testCheckTie_full_except_top_left_false

`boolean checkTie()`

Input: State: u v w x p q r s t k l m n o f g h i j a b c d e	Output: checkTie = false state of the board is unchanged	Reason: This test is unique because the entire board is full except for the top right spot. **Function Name:** testCheckTie_full_except_top_right_false

`boolean checkTie()`

Input: State: f g h i j a b c d e	Output: checkTie = false state of the board is unchanged	Reason: This test is unique because no columns in the board are full and the board is not almost full. **Function Name:** testCheckTie_no_full_columns_false

`boolean checkTie()`

Input: State: u v w x y p q r s t k l m n o f g h i j a b c d e	Output: checkTie = true state of the board is unchanged	Reason: This test is unique because all columns are full. Function Name: testCheckTie_full_board_true

`char whatsAtPos(BoardPosition pos)`

Input: State: u v w x y p q r s t k l m n o f g h i j **a** b c d e pos.getRow = 0 pos.getCol = 0	Output: whatsAtPos = 'a' state of the board is unchanged	Reason: This test is unique because it is testing that whatsAtPos can properly return character in border case where char is at bottom left corner of board. Function Name: testWhatsAtPos_char_at_bottom_left_corner

`char whatsAtPos(BoardPosition pos)`

Input: State: **u** v w x y p q r s t k l m n o f g h i j a b c d e pos.getRow = 4 pos.getCol = 0	Output: whatsAtPos = 'u' state of the board is unchanged	Reason: This test is unique because it is testing that whatsAtPos can properly return character in border case where char is at top left corner of board. Function Name: testWhatsAtPos_char_at_top_left_corner

`char whatsAtPos(BoardPosition pos)`

Input: State: u v w x y p q r s t k l m n o f g h i j a b c d e pos.getRow = 0 pos.getCol = 4	Output: whatsAtPos = 'e' state of the board is unchanged	Reason: This test is unique because it is testing that whatsAtPos can properly return character in border case where char is at bottom right corner of board. Function Name: testWhatsAtPos_char_at_bottom_right_corner

`char whatsAtPos(BoardPosition pos)`

Input: State: u v w x y p q r s t k l m n o f g h i j a b c d e pos.getRow = 4 pos.getCol = 4	Output: whatsAtPos = 'y' state of the board is unchanged	Reason: This test is unique because it is testing that whatsAtPos can properly return character in border case where char is at top right corner of board. Function Name: testWhatsAtPos_char_at_top_right_corner

`char whatsAtPos(BoardPosition pos)`

Input: State: k l m n f g h i j a b c d e pos.getRow = 2 pos.getCol = 2	Output: whatsAtPos = ' ' state of the board is unchanged	Reason: This test is unique because it is testing to make sure whatsAtPos properly returns a space character when board space is empty. Function Name: testWhatsAtPos_return_space

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input: State: u v w x y p q r s t k l m n o f g h i j **a** b c d e pos.getRow = 0 pos.getCol = 0 player = 'a'	Output: isPlayerAtPos = true state of the board is unchanged	Reason: This test is unique because it is testing that isPlayerAtPos can correctly identify a char in the border case that the char is in bottom left corner of board. Function Name: testIsPlayerAtPos_char_at_bottom_left_corner

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input: State: **u** v w x y p q r s t k l m n o f g h i j a b c d e pos.getRow = 4 pos.getCol = 0 player = 'u'	Output: isPlayerAtPos = true state of the board is unchanged	Reason: This test is unique because it is testing that isPlayerAtPos can correctly identify a char in the border case that the char is in top left corner of board. Function Name: testIsPlayerAtPos_char_at_top_left_corner

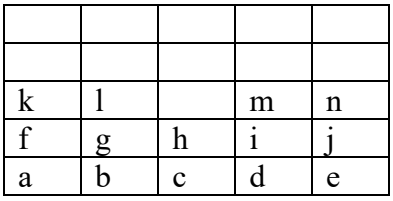
boolean isPlayerAtPos(BoardPosition pos, char player)

Input: State: u v w x y p q r s t k l m n o f g h i j a b c d **e** pos.getRow = 0 pos.getCol = 4 player = 'e'	Output: isPlayerAtPos = true state of the board is unchanged	Reason: This test is unique because it is testing that isPlayerAtPos can correctly identify a char in the border case that the char is in the bottom right corner of board. Function Name: testIsPlayerAtPos_char_at_bottom_right_corner

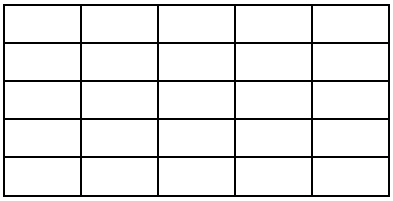
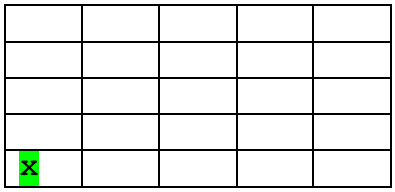
boolean isPlayerAtPos(BoardPosition pos, char player)

Input: State: u v w x **y** p q r s t k l m n o f g h i j a b c d e pos.getRow = 4 pos.getCol = 4 player = 'y'	Output: isPlayerAtPos = true state of the board is unchanged	Reason: This test is unique because it is testing that isPlayerAtPos can correctly identify a char in the border case that the char is in the top right corner of the board. Function Name: testIsPlayerAtPos_char_at_top_right_corner

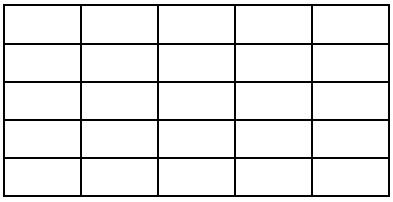
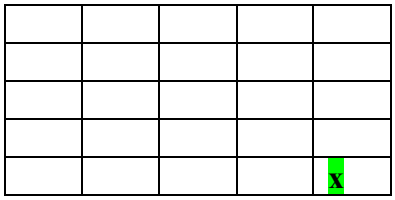
`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input: State:  pos.getRow = 2 pos.getCol = 2 player = 'x'	Output: isPlayerAtPos = false state of the board is unchanged	Reason: This test is unique because it is testing that isPlayerAtPos can correctly identify if a char is not at the specified location. Function Name: testIsPlayerAtPos_ middle_of_board_false

`void dropToken(char p, int c)`

Input: State:  p = 'x' c = 0	Output: State: 	Reason: This test is unique because it is testing the border case of dropping a token into bottom left corner of board. Function Name: testDropToken_empty_ board_bottom_left

`void dropToken(char p, int c)`

Input: State:  p = 'x' c = 4	Output: State: 	Reason: This test is unique because it is testing the border case of dropping a token into bottom right corner of board. Function Name: testDropToken_empty_ board_bottom_right

`void dropToken(char p, int c)`

Input:

State:

d				
c				
b				
a				

p = 'e'
c = 0

Output:

State:

e				
d				
c				
b				
a				

Reason:
This test is unique because it is testing the border case of dropping a token into top left corner of the board.

Function Name:
testDropToken_top_left_corner

`void dropToken(char p, int c)`

Input:

State:

				d
				c
				b
				a

p = 'e'
c = 4

Output:

State:

				e
				d
				c
				b
				a

Reason:
This test is unique because it is testing the border case of dropping a token into the top right corner of the board.

Function Name:
testDropToken_top_right_corner

`void dropToken(char p, int c)`

Input:

State:

a	b	c	d	e

p = 'f'
c = 2

Output:

State:

		f		
a	b	c	d	e

Reason:
This test is unique because it is testing if dropToken properly places new token on top of already existing tokens.

Function Name:
testDropToken_middle column_not_empty