CPSC 2150 Project 4 Report

Abby Thornton, Elle Hanckel, Jenna Grossmann

Requirements Analysis

Functional Requirements: As a <userRole> I <what/need/can> <goal> so that <reason>

1. As a player, I can choose what slot I want to place my marker
2. As a player, I can see the open slots, so I know where I can put my marker
3. As a player, I can see the taken slots, so I know where I cannot to put my marker
4. As a player, I can see where my opponent puts their marker, so I can use that information when deciding where to place my marker
5. As a player, I want to be informed when I have won so that I know when the game is over
6. As a player, I want to be given the option to play again after I complete a game, so that I can keep playing as long as I wish
7. As a player, I want to be given the option to exit the program after the game, so I can stop playing
8. As a player, I want to be able to decide to play in fast or memory efficient mode, so that it is either more efficient or runs quicker.
9. As a player, I want to be able to change the board, so I can play on different size boards
10. As a player, if I choose an already full column, I want to be informed so I can make a valid choice
11. As a player, if I choose a column that does not exist, I want to be informed so I can make a valid column choice
12. As a player, I want the game to automatically alternate between players, so we alternate turns correctly.
13. As a player, at the start of the game I want to see the blank board so I know where I can place my markers.
14. As a player, I want the different player's moves to be designated with the token markers they choose so I can see each player's moves.
15. As a player, I want to know what my token marker is throughout the game, so I dont forget what marker I am.
16. As a player, I should be informed the game ends in a tie if there are no other moves possible so that the game ends
17. As a player, I can win the game if I get the designated number of tokens in a row horizontally, so that the game ends
18. As a player, I can win the game if I get the designated number of tokens in a row vertically, so that the game ends
19. As a player, I can win the game if I get the designated number of tokens in a row diagonally, so that the game ends

Non-Functional Requirements

1. The system must be written in Java.
2. The system must be a command-line application.
3. Board has to be a size chosen by players.
4. (0,0) is at the bottom left of the board.
5. The system must validate input depending on the board size.
6. The system is accurate when placing a marker in the desired spot.

System Design

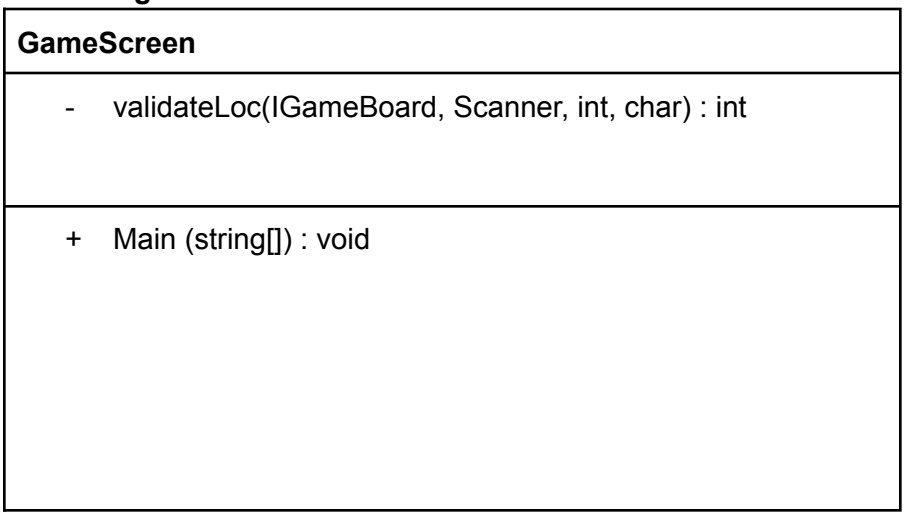
GameBoard

Class diagram

GameBoard
- gameBoard : char [][] + MAX_ROW : const int + MAX_COL : const int + TOKENS_TO_WIN: const int
+ gameBoard() : void + getNumColumns() : int + getNumRows() : int + getNumToWin() : int + dropToken(char, int) : void + whatsAtPos(BoardPosition) : char

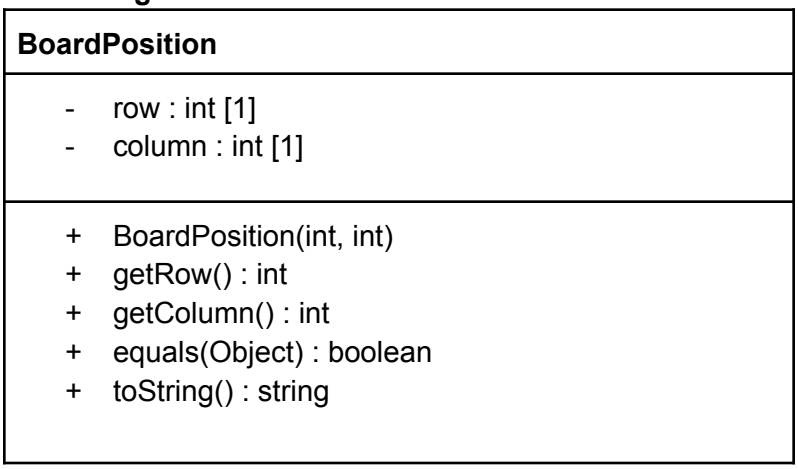
GameScreen

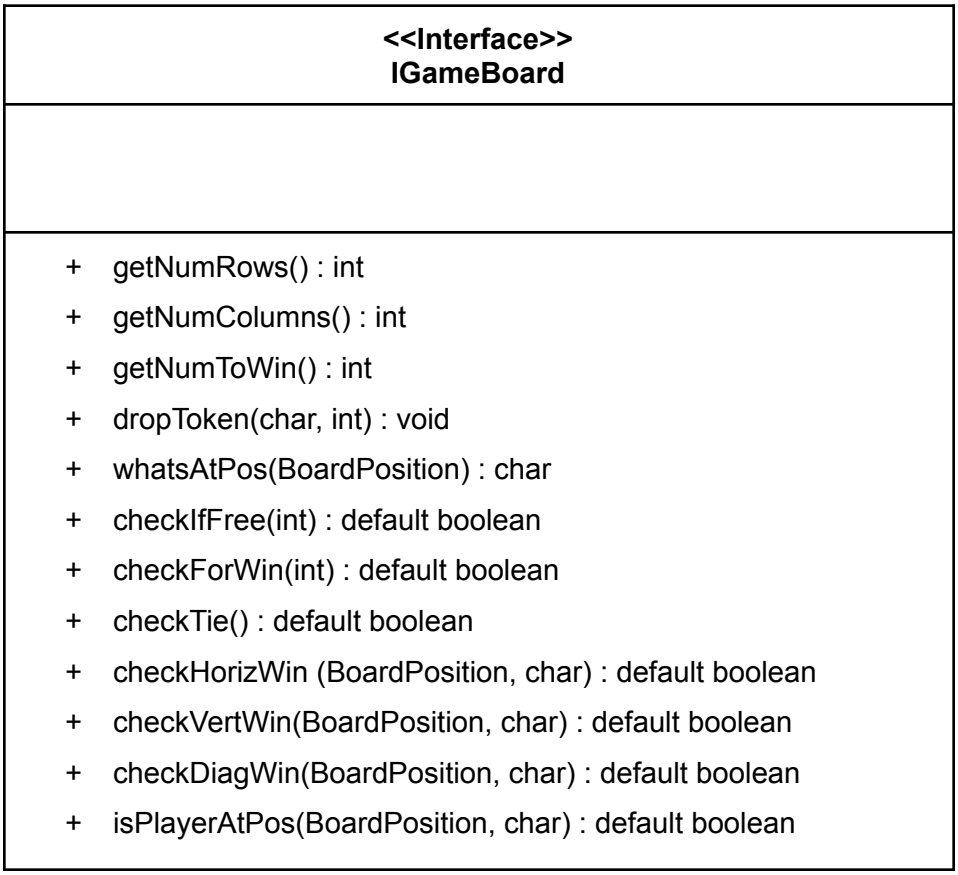
Class diagram

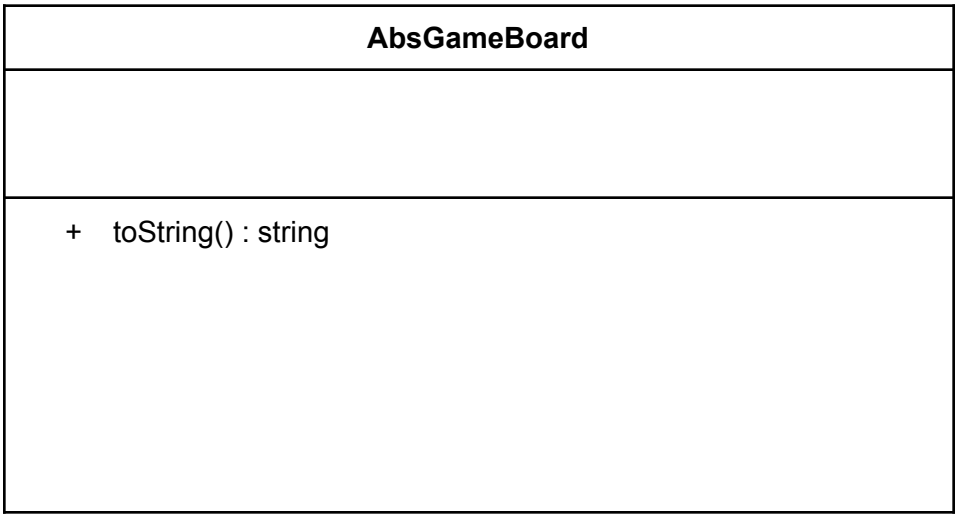


BoardPosition

Class diagram




<<Interface>>
IGameBoard
+ getNumRows() : int
+ getNumColumns() : int
+ getNumToWin() : int
+ dropToken(char, int) : void
+ whatsAtPos(BoardPosition) : char
+ checkIfFree(int) : default boolean
+ checkForWin(int) : default boolean
+ checkTie() : default boolean
+ checkHorizWin (BoardPosition, char) : default boolean
+ checkVertWin(BoardPosition, char) : default boolean
+ checkDiagWin(BoardPosition, char) : default boolean
+ isPlayerAtPos(BoardPosition, char) : default boolean


AbsGameBoard
+ toString() : string

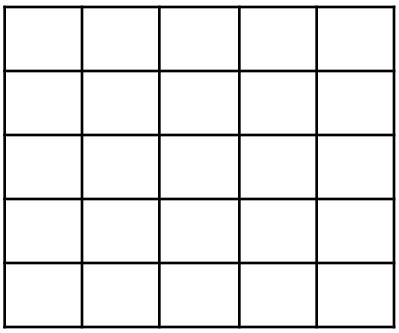
GameBoardMem
- map : HashMap<Character, List<BoardPosition>> + maxRow : int + maxCol : int + numToWin : int
+ GameBoardMem(int, int, int) + getNumRows() : int + getNumColumns() : int + getNumToWin() : int + dropToken(char, int) : void + whatsAtPos(BoardPosition) : char

Testing

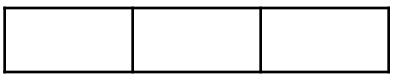
Tests for GameBoard

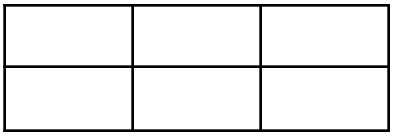
Constructor

`GameBoard(int userRow, int userCol, int token)`

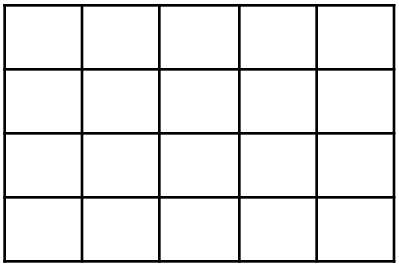
Input: userRow = 5 userCol = 5 Token = 3 State:	**Output:** State: 	**Reason:** This test case is unique and distinct because it tests that the gameboard is initialized to only empty blank space characters **Function Name:** testGameBoardConstructor_empty_array_initialized

`GameBoard(int userRow, int userCol, int token)`

Input: userRow = 3 userCol = 3 Token = 3	**Output:** State: 	**Reason:** This test case is unique and distinct because it tests the minimum amount of rows, columns, and number of tokens a gameBoard can

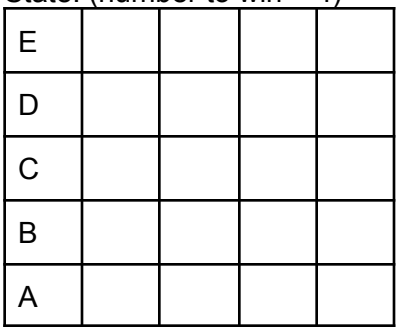
State:		have **Function Name:** testGameBoardConstructor_minimum_size

`GameBoard(int userRow, int userCol, int token)`

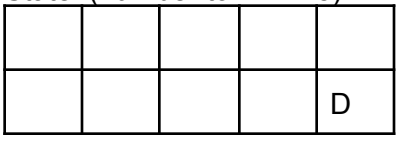
Input: userRow = 100 userCol = 100 Token = 25 State:	**Output:** State:  ** each box represents 5 boxes	**Reason:** This test case is unique and distinct because it tests that the gameboard array is initialized to the correct row and column size. **Function Name:** testGameBoardConstructor_maximum_size

<u>checkIfFree</u>

`checkIfFree(int c)`

Input: State: (number to win = 4)  c = 0	**Output:** checkIfFree = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the column 0 was full when checkIfFree was called, so the function should to go through the whole column to check **Function Name:** testCheckIfFree_full_far_left_column

`checkIfFree(int c)`

Input: State: (number to win = 5) 	**Output:** checkIfFree = true State of the board is unchanged	**Reason:** This test case is unique and distinct because column 4 was filled except one space when checkIfFree was called, so the function should find that the last space is available.

				C
				B
				A

c = 4

Function Name:
testCheckIfFree_almost_filled_far_right_column

`checkIfFree(int c)`

Input:

State: (number to win = 4)

X				

c = 2

Output:

State: checkIfFree = true

State of the board is unchanged

Reason:
This test case is unique and distinct because it checks if it can place a token in a completely empty column

Function Name:
testCheckIfFree_completely_empty_column

checkHorizWin

`Boolean checkHorizWin(BoardPosition pos, char p)`

Input:

State: (number to win = 5)

X	X	X	X	X
O	O	O	X	O

pos.getRow = 1
pos.getCol = 2
p = 'X'

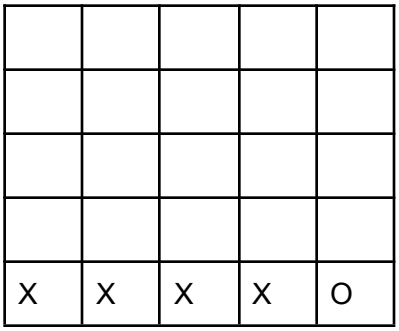
Output:
checkHorizWin = true

State of the board is unchanged

Reason:
This test case is unique and distinct because the last X was placed in the middle of the string of 5 consecutive X's as opposed to on the end, so the function needs to count X's on the right and left. The win is also in the middle of the board.

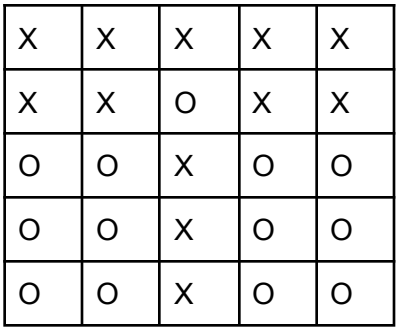
Function Name:
testCheckHorizWin_win_last_marker_in_between_tokens

`Boolean checkHorizWin(BoardPosition pos, char p)`

Input: State: (number to win = 4)  pos.getRow = 0 pos.getCol = 0 p = 'X'	**Output:** checkHorizWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X was placed in the left end of the string of 4 consecutive X's as opposed to in the middle, so the function needs to count X's on the right. The win is also on the bottom of the board **Function Name:** testCheckHorizWin_win_last_marker_left_bottom_row

X	X	X	X	O

`Boolean checkHorizWin(BoardPosition pos, char p)`

Input: State: (number to win = 5)  pos.getRow = 4 pos.getCol = 4 p = 'X'	**Output:** checkHorizWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X was placed in the right end of the string of 4 consecutive X's as opposed to in the middle or left end, so the function needs to count X's on the left. The win is also at the top of the board. **Function Name:** testCheckHorizWin_win_last_marker_right_top_row

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

`Boolean checkHorizWin(BoardPosition pos, char p)`

Input: State: (number to win = 3)	**Output:** checkHorizWin = false	**Reason:** This test case is unique and distinct because the last X

<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>X</td><td>X</td><td>O</td><td></td><td></td></tr><tr><td>X</td><td>X</td><td>O</td><td></td><td></td></tr></table> pos.getRow = 1 pos.getCol = 1 p = ‘X’	State of the board is unchanged	placed in the string did not result in a horizontal win. **Function Name:** testCheckHorizWin_no_win

checkVertWin

`Boolean checkVertWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) | | | | | | | | | | | | | | | | X | | | | | | X | | | | | O | X | | | | pos.getRow = 2 pos.getCol = 1 p = 'X'	checkVertWin = false State of the board is unchanged	This test case is unique and distinct because the last token placed did not result in a win. **Function Name:** testCheckVertWin_no_win

`Boolean checkVertWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 3) | | | | | | | | | | | | | | | X | | | | | | X | | | | | | X | | | | O | pos.getRow = 2 pos.getCol = 0 p = 'X'	checkVertWin = true State of the board is unchanged	This test case is unique and distinct because the last token placed was in the first column **Function Name:** testCheckVertWin_first_column_win

`Boolean checkVertWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) X X X X O pos.getRow = 4 pos.getCol = 4 p = ‘X’	checkVertWin = true State of the board is unchanged	This test case is unique and distinct because the last token placed was in the last row and last column of the board. **Function Name:** testCheckVertWin_last_row_last_column_win

`Boolean checkVertWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) O O O X O pos.getRow = 3 pos.getCol = 2 p = ‘O’	checkVertWin = true State of the board is unchanged	This test case is unique and distinct because the last token placed was in the middle of the board. **Function Name:** testCheckVertWin_middle_board_win

checkDiagWin

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 3) pos.getRow = 1 pos.getCol = 2 p = 'X'	checkDiagWin = false State of the board is unchanged	This test case is unique and distinct because the last X placed in the string did not result in a diagonal win. **Function Name:** testCheckDiagWin_no_win

O	X	X		
O	X	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 case is unique and distinct because the last X placed in the string was in the top right position of 5 diagonal X's as opposed to the top left position, so the function needs to count X's on a bottom left to top right win. It is also placed in the last column and last row. **Function Name:** testCheckDiagWin_top_pos_right_diag_win_last_row_last_col

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

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 3) | | | | | | | | | | | | | | | X | | | | | | X | X | | | | | O | X | X | | | pos.getRow = 0 pos.getCol = 3 p = 'X'	checkDiagWin = true State of the board is unchanged	This test case is unique and distinct because the last X placed in the string was in the bottom right position of 3 diagonal X's as opposed to the bottom left position, so the function needs to count X's on a bottom right to top left win **Function Name:** testCheckDiagWin_bottom_pos_left_diag_win

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 3) | | | | | | | | | | | X | | | | | X | X | | | | X | O | O | | | | X | O | X | | pos.getRow = 2 pos.getCol = 2 p = 'X'	checkDiagWin = true State of the board is unchanged	This test case is unique and distinct because the last X placed in the string was in the middle position of 3 diagonal X's as opposed to the bottom/top left or right position, so the function needs to count X's on a bottom left and top right diag to win **Function Name:** testCheckDiagWin_middle_pos_right_diag_win

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 3) pos.getRow = 3 pos.getCol = 1 p = ‘X’	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X placed in the string was in the top position of 3 diagonal X’s as opposed to the middle or bottom right position, so the function needs to count X’s on the bottom right. **Function Name:** testCheckDiagWin_top_pos_left_diag_win

	X			
	X	X		
	O	O	X	
	O	X	X	

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 3) pos.getRow = 0 pos.getCol = 1 p = ‘X’	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X placed in the string was in the bottom position in a diagonal of 3 X’s as opposed to the bottom or middle, so the function needs to count X’s on a bottom left and top right to win **Function Name:** testCheckDiagWin_bottom_pos_right_diag_win

			X	
		X	X	
	X	X	O	

`Boolean checkDiagWin(BoardPosition pos, char p)`

<table>
<tr>
<td>Input:

State: (number to win = 3)

<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td>O</td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td>X</td><td></td><td></td></tr>
</table>

pos.getRow = 2
pos.getCol = 1
p = ‘X’</td>
<td>Output:
checkDiagWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X placed in the string was in the middle position of 2 3-diagonal X’s as opposed to the 1 3-diagonal X’s, so the function needs to count X’s on a bottom left and top right or bottom right to top left to win

Function Name:
testCheckDiagWin_middle_pos_two_diags_win</td>
</tr>
</table>

checkTie

`Boolean checkTie()`

Input: State: (number to win = 5)	**Output:** checkTie = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the board is not full. **Function Name:** testCheckTie_board_not_full

				X
				X
				X
O	O			X

`Boolean checkTie()`

Input: State: (number to win = 5)	**Output:** checkTie = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the board is full and there is no win. **Function Name:** testCheckTie_board_full_no_win

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

`Boolean checkTie()`

Input: State: (number to win = 5) 					**Output:** checkTie = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the board is empty **Function Name:** testCheckTie_board_empty

`Boolean checkTie()`

Input: State: (number to win = 5) 					**Output:** checkTie = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the last token placed does not result in a tie but a win **Function Name:** testCheckTie_full_board_win_no_tie
O	O	O	O	O		
X	X	X	X	O		
X	X	X	X	O		
X	X	X	X	O		
X	X	X	X	O		

whatsAtPos

`char whatAtPos(BoardPosition pos)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) pos.getRow = 0 pos.getCol = 3	whatAtPos = ‘ ‘ State of the board is unchanged	This test case is unique and distinct because it checks what char is at a board position that has no char and returns that value. **Function Name:** testWhatsAtPos_empty_pos

				O
X	X	X		O

`char whatAtPos(BoardPosition pos)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) pos.getRow = 0 pos.getCol = 2	whatAtPos = ‘X’ State of the board is unchanged	This test case is unique and distinct because it returns correct player char in bottom row **Function Name:** testWhatsAtPos_bottom_row_pos

O				
O		X	X	X

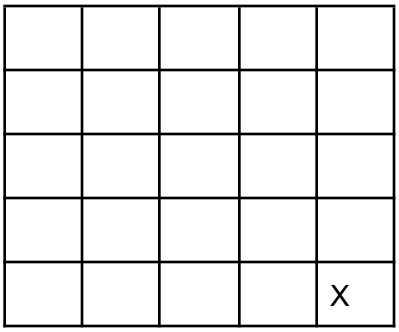
`char whatAtPos(BoardPosition pos)`

Input: State: (number to win = 4) E D C B A pos.getRow = 4 pos.getCol = 1	**Output:** whatAtPos = ‘E‘ State of the board is unchanged	**Reason:** This test case is unique and distinct because it checks what char is at the top row and returns the correct char value. **Function Name:** testWhatsAtPos_top_row_pos

`char whatAtPos(BoardPosition pos)`

Input: State: (number to win = 4) X pos.getRow = 0 pos.getCol = 0	**Output:** whatAtPos = ‘X‘ State of the board is unchanged	**Reason:** This test case is unique and distinct because it checks what char is in the first column and returns correct player char **Function Name:** testWhatsAtPos_first_column_pos

`char whatAtPos(BoardPosition pos)`

Input:	**Output:**	**Reason:**
State: (number to win = 4)  pos.getRow = 0 pos.getCol = 4	whatAtPos = ‘X‘ State of the board is unchanged	This test case is unique and distinct because it checks what char is in the last column and returns the correct player char. **Function Name:** testWhatsAtPos_last_column_pos

isPlayerAtPos

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input: State: (number to win = 4) 				
				O
X	X	X		O

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

Output:
isPlayerAtPos = false

State of the board is unchanged

Reason:
This test case is unique and distinct because it checks what char is at a board position in the middle right of the board in a row where the space is empty.

Function Name:
testIsPlayerAtPos_empty_position

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input: State: (number to win = 4) 				
A	B	C	D	E

pos.getRow = 0
pos.getCol = 1
player = 'B'

Output:
isPlayerAtPos = true

State of the board is unchanged

Reason:
This test case is unique and distinct because it checks what char is at a board position in the second column/ first row when the space has the correct player.

Function Name:
testIsPlayerAtPos_correct_player

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input: State: (number to win = 4) pos.getRow = 2 pos.getCol = 2 player = ‘O’	**Output:** isPlayerAtPos = false State of the board is unchanged	**Reason:** This test case is unique and distinct because it checks what char is at a board position in the last column/ first row when the space has the correct player. **Function Name:** testIsPlayerAtPos_incorrect_player

		X		
		O		
		X		

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input: State: (number to win = 4) pos.getRow = 0 pos.getCol = 0 player = ‘X’	**Output:** isPlayerAtPos = true State of the board is unchanged	**Reason:** This test case is unique and distinct because checks that the correct player is in the bottom row first column of the board **Function Name:** testIsPlayerAtPos_first_column_bottom_row

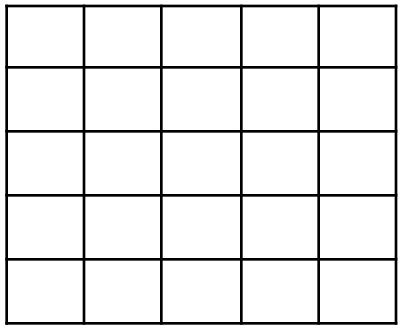
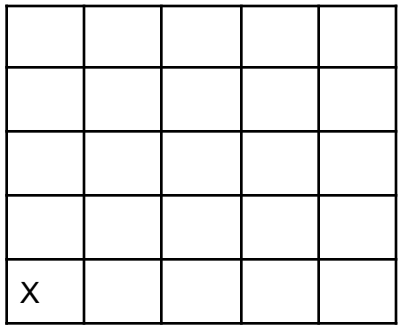
X	O	X	O	X

`boolean isPlayerAtPos(BoardPosition pos, char player)`

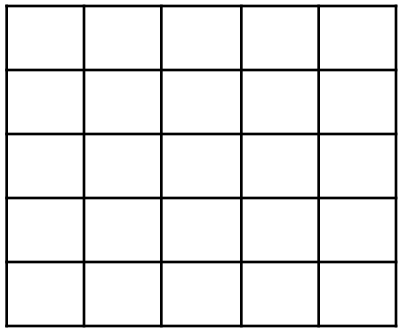
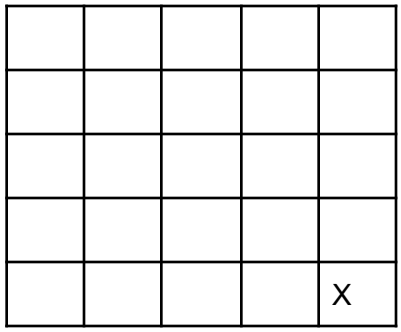
Input:	**Output:**	**Reason:**
State: (number to win = 4) |		|				X | |		|		|		O | |		|		|		X | |		|		|		O | |		|		|		X | pos.getRow = 4 pos.getCol = 4 player = ‘X’	isPlayerAtPos = true State of the board is unchanged	This test case is unique and distinct because it checks that the correct player is in the top row last column of the board. **Function Name:** testIsPlayerAtPos_top_row_last_column

dropToken

`void dropToken(char p, int c)`

Input:	**Output:**	**Reason:**
State: (number to win = 4)  c = 0 p = ‘X’	State: (number to win = 4) 	This test case is unique and distinct because it checks that the first column and first available row is empty and places the player token on board. **Function Name:** testDropToken_first_column

`void dropToken(char p, int c)`

Input:	**Output:**	**Reason:**
State: (number to win = 4)  c = 4 p = ‘X’	State: (number to win = 4) 	This test case is unique and distinct because it checks that the last column and first available row is empty and places the player token on board. **Function Name:** testDropToken_last_column

`void dropToken(char p, int c)`

Input: State: (number to win = 4) c = 2 p = ‘X’	**Output:** State: (number to win = 4)	**Reason:** This test case is unique and distinct because it checks if the position in the middle of the board is empty and places it there. **Function Name:** testDropToken_middle_pos

		X		O
		X		O

		X		
		X		O
		X		O

`void dropToken(char p, int c)`

Input: State: (number to win = 4) c = 2 p = ‘O	**Output:** State: (number to win = 4)	**Reason:** This test case is unique and distinct because it is a column that is not empty and does not place the token in the already taken spot. **Function Name:** testDropToken_full_column

		X		
		O		
		X		
		O		
		X		

		X		
		O		
		X		
		O		
		X		

`void dropToken(char p, int c)`

Input:

State: (number to win = 5)

Q	R	S	T	
M	N	O	P	X
I	J	K	L	W
E	F	G	H	V
A	B	C	D	U

c = 4
p = ‘Y’

Output:

State: (number to win = 5)

Q	R	S	T	Y
M	N	O	P	X
I	J	K	L	W
E	F	G	H	V
A	B	C	D	U

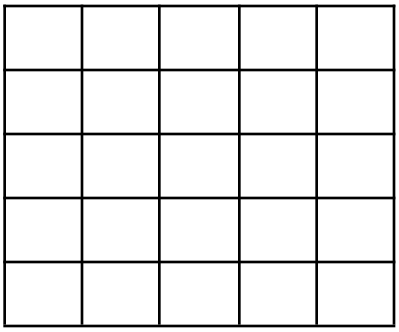
Reason:
This test case is unique and distinct because it checks the last available position is empty then places token in that position

Function Name:
testDropToken_last_pos

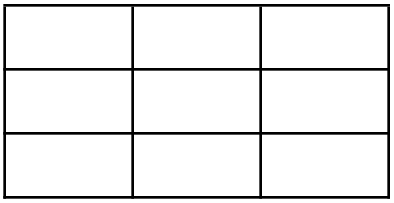
Tests for GameBoardMem

Constructor

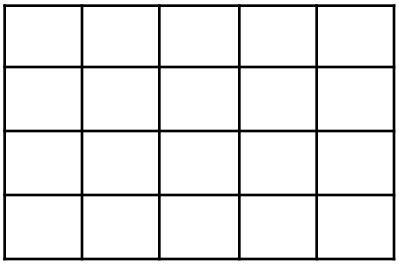
`GameBoardMem(int userRow, int userCol, int token)`

Input:	**Output:**	**Reason:**
userRow = 5 userCol = 5 Token = 3 State:	State: 	This test case is unique and distinct because it tests that the gameboard is initialized to only empty blank space characters **Function Name:** testGameBoardMem_Constructor_empty_array_initialized

`GameBoardMem(int userRow, int userCol, int token)`

Input:	**Output:**	**Reason:**
userRow = 3 userCol = 3 Token = 3 State:		This test case is unique and distinct because it tests the minimum amount of rows, columns, and number of tokens a gameBoard can have **Function Name:** testGameBoardMem_Constructor_minimum_size

`GameBoardMem(int userRow, int userCol, int token)`

Input:	**Output:**	**Reason:**
userRow = 100 userCol = 100 Token = 25 State:	State:  ** each box represents 5 boxes	This test case is unique and distinct because it tests that the gameboard array is initialized to the correct maximum row and column size. **Function Name:** testGameBoardMem_Constructor_maximum_size

checkIfFree

`checkIfFree(int c)`

Input: State: (number to win = 4) c = 0	**Output:** checkIfFree = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the column 0 was full when checkIfFree was called, so the function should to go through the whole column to check **Function Name:** testCheckIfFree_full_far_left_column

E				
D				
C				
B				
A				

`checkIfFree(int c)`

Input: State: (number to win = 5) c = 4	**Output:** checkIfFree = true State of the board is unchanged	**Reason:** This test case is unique and distinct because column 4 was filled except one space when checkIfFree was called, so the function should find that the last space is available. **Function Name:** testCheckIfFree_almost_filled_far_right_column

				D
				C
				B
				A

`checkIfFree(int c)`

Input: State: (number to win = 4) c = 2	**Output:** State: checkIfFree = true State of the board is unchanged	**Reason:** This test case is unique and distinct because it checks if it can place a token in a completely empty column **Function Name:** testCheckIfFree_completely_empty_column

X				

checkHorizWin

`Boolean checkHorizWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 5) |		|		|		| |---|---|---|---|---| | | | | | | | | | | | | | | | | | | | X | X | X | X | X | | O | O | O | X | O | pos.getRow = 1 pos.getCol = 2 p = 'X'	**Output:** checkHorizWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X was placed in the middle of the string of 5 consecutive X's as opposed to on the end, so the function needs to count X's on the right and left. The win is also in the middle of the board. **Function Name:** testCheckHorizWin_win_last_marker_in_between_tokens

`Boolean checkHorizWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) |		|		|		| |---|---|---|---|---| | | | | | | | | | | | | | | | | | | | | | | | | | X | X | X | X | O | pos.getRow = 0 pos.getCol = 0 p = 'X'	**Output:** checkHorizWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X was placed in the left end of the string of 4 consecutive X's as opposed to in the middle, so the function needs to count X's on the right. The win is also on the bottom of the board **Function Name:** testCheckHorizWin_win_last_marker_left_bottom_row

`Boolean checkHorizWin(BoardPosition pos, char p)`

Input: State: (number to win = 5) X X X X X X X O X X O O X O O O O X O O O O X O O pos.getRow = 4 pos.getCol = 4 p = 'X'	**Output:** checkHorizWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X was placed in the right end of the string of 4 consecutive X's as opposed to in the middle or left end, so the function needs to count X's on the left. The win is also at the top of the board. **Function Name:** testCheckHorizWin_win_last_marker_right_top_row

`Boolean checkHorizWin(BoardPosition pos, char p)`

Input: State: (number to win = 3) X X O X X O pos.getRow = 1 pos.getCol = 1 p = 'X'	**Output:** checkHorizWin = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X placed in the string did not result in a horizontal win. **Function Name:** testCheckHorizWin_no_win

checkVertWin

`Boolean checkVertWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) pos.getRow = 2 pos.getCol = 1 p = 'X'	checkVertWin = false State of the board is unchanged	This test case is unique and distinct because the last token placed did not result in a win. **Function Name:** testCheckVertWin_no_win

	X			
	X			
O	X			

`Boolean checkVertWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 3) pos.getRow = 2 pos.getCol = 0 p = 'X'	checkVertWin = true State of the board is unchanged	This test case is unique and distinct because the last token placed was in the first column **Function Name:** testCheckVertWin_first_ column_win

X				
X				
X				O

`Boolean checkVertWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) X X X X O pos.getRow = 4 pos.getCol = 4 p = ‘X’	checkVertWin = true State of the board is unchanged	This test case is unique and distinct because the last token placed was in the last row and last column of the board. **Function Name:** testCheckVertWin_last_row_last_column_win

`Boolean checkVertWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) O O O X O pos.getRow = 3 pos.getCol = 2 p = ‘O’	checkVertWin = true State of the board is unchanged	This test case is unique and distinct because the last token placed was in the middle of the board. **Function Name:** testCheckVertWin_middle_board_win

checkDiagWin

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input:	**Output:**	**Reason:**
State: (number to win = 3) pos.getRow = 1 pos.getCol = 2 p = 'X'	checkDiagWin = false State of the board is unchanged	This test case is unique and distinct because the last X placed in the string did not result in a diagonal win. **Function Name:** testCheckDiagWin_no_win

O	X	X		
O	X	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 case is unique and distinct because the last X placed in the string was in the top right position of 5 diagonal X's as opposed to the top left position, so the function needs to count X's on a bottom left to top right win. It is also placed in the last column and last row. **Function Name:** testCheckDiagWin_top_pos_right_diag_win_last_row_last_col

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

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 3) pos.getRow = 0 pos.getCol = 3 p = 'X'	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X placed in the string was in the bottom right position of 3 diagonal X's as opposed to the bottom left position, so the function needs to count X's on a bottom right to top left win **Function Name:** testCheckDiagWin_bottom_pos_left_diag_win

X				
X	X			
O	X	X		

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 3) pos.getRow = 2 pos.getCol = 2 p = 'X'	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X placed in the string was in the middle position of 3 diagonal X's as opposed to the bottom/top left or right position, so the function needs to count X's on a bottom left and top right diag to win **Function Name:** testCheckDiagWin_middle_pos_right_diag_win

			X	
		X	X	
	X	O	O	
	X	O	X	

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 3) pos.getRow = 3 pos.getCol = 1 p = ‘X’	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X placed in the string was in the top position of 3 diagonal X’s as opposed to the middle or bottom right position, so the function needs to count X’s on the bottom right. **Function Name:** testCheckDiagWin_top_pos_left_diag_win

	X			
	X	X		
	O	O	X	
	O	X	X	

`Boolean checkDiagWin(BoardPosition pos, char p)`

Input: State: (number to win = 3) pos.getRow = 0 pos.getCol = 1 p = ‘X’	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the last X placed in the string was in the bottom position in a diagonal of 3 X’s as opposed to the bottom or middle, so the function needs to count X’s on a bottom left and top right to win **Function Name:** testCheckDiagWin_bottom_pos_right_diag_win

			X	
		X	X	
	X	X	O	

`Boolean checkDiagWin(BoardPosition pos, char p)`

<table>
<tr>
<td>Input:

State: (number to win = 3)

<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td>O</td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td>X</td><td></td><td></td></tr>
</table>

pos.getRow = 2
pos.getCol = 1
p = ‘X’</td>
<td>Output:
checkDiagWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because the last X placed in the string was in the middle position of 2 3-diagonal X’s as opposed to the 1 3-diagonal X’s, so the function needs to count X’s on a bottom left and top right or bottom right to top left to win

Function Name:
testCheckDiagWin_middle_pos_two_diags_win</td>
</tr>
</table>

checkTie

`Boolean checkTie()`

Input: State: (number to win = 5)	**Output:** checkTie = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the board is not full. **Function Name:** testCheckTie_board_not_full

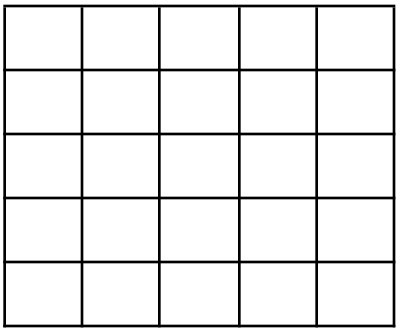
				X
				X
				X
O	O			X

`Boolean checkTie()`

Input: State: (number to win = 5)	**Output:** checkTie = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the board is full and there is no win. **Function Name:** testCheckTie_board_full_no_win

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

`Boolean checkTie()`

Input: State: (number to win = 5) 	**Output:** checkTie = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the board is empty **Function Name:** testCheckTie_board_empty

`Boolean checkTie()`

Input: State: (number to win = 5) O O O O O X X X X O X X X X O X X X X O X X X X O	**Output:** checkTie = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the last token placed does not result in a tie but a win **Function Name:** testCheckTie_full_board_win_no_tie

whatsAtPos

`char whatAtPos(BoardPosition pos)`

Input: State: (number to win = 4) pos.getRow = 0 pos.getCol = 3	**Output:** whatAtPos = ‘ ‘ State of the board is unchanged	**Reason:** This test case is unique and distinct because it checks what char is at a board position that has no char and returns that value. **Function Name:** testWhatsAtPos_empty_pos

				O
X	X	X		O

`char whatAtPos(BoardPosition pos)`

Input: State: (number to win = 4) pos.getRow = 0 pos.getCol = 2	**Output:** whatAtPos = ‘X’ State of the board is unchanged	**Reason:** This test case is unique and distinct because it returns correct player char in bottom row **Function Name:** testWhatsAtPos_bottom_row_pos

O				
O		X	X	X

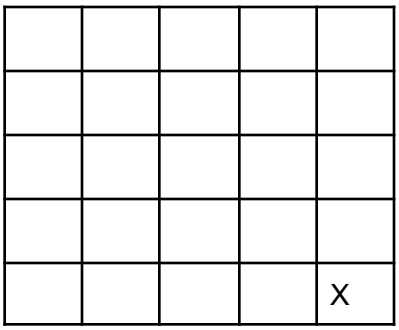
`char whatAtPos(BoardPosition pos)`

Input: State: (number to win = 5) 					**Output:** whatAtPos = ‘E‘ State of the board is unchanged	**Reason:** This test case is unique and distinct because it checks what char is at the top row and returns the correct char value. **Function Name:** testWhatsAtPos_top_row_pos
	E					
	D					
	C					
	B					
	A					
pos.getRow = 4 pos.getCol = 1						

`char whatAtPos(BoardPosition pos)`

Input: State: (number to win = 4) 					**Output:** whatAtPos = ‘X‘ State of the board is unchanged	**Reason:** This test case is unique and distinct because it checks what char is in the first column and returns correct player char **Function Name:** testWhatsAtPos_first_column_pos
X						
pos.getRow = 0 pos.getCol = 0						

`char whatAtPos(BoardPosition pos)`

Input: State: (number to win = 4)  pos.getRow = 0 pos.getCol = 4	**Output:** whatAtPos = ‘X‘ State of the board is unchanged	**Reason:** This test case is unique and distinct because it checks what char is in the last column and returns the correct player char. **Function Name:** testWhatsAtPos_last_column_pos

isPlayerAtPos

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) pos.getRow = 0 pos.getCol = 3 player = 'X'	isPlayerAtPos = false State of the board is unchanged	This test case is unique and distinct because it checks what char is at a board position in the middle right of the board in a row where the space is empty. **Function Name:** testIsPlayerAtPos_empty_position

				O
X	X	X		O

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input:	**Output:**	**Reason:**
State: (number to win = 4) pos.getRow = 0 pos.getCol = 1 player = 'B'	isPlayerAtPos = true State of the board is unchanged	This test case is unique and distinct because it checks what char is at a board position in the second column/ first row when the space has the correct player. **Function Name:** testIsPlayerAtPos_correct_player

A	B	C	D	E

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input: State: (number to win = 4) pos.getRow = 2 pos.getCol = 2 player = ‘O’	**Output:** isPlayerAtPos = false State of the board is unchanged	**Reason:** This test case is unique and distinct because it checks what char is at a board position in the last column/ first row when the space has the correct player. **Function Name:** testIsPlayerAtPos_incorrect_player

		X		
		O		
		X		

`boolean isPlayerAtPos(BoardPosition pos, char player)`

Input: State: (number to win = 4) pos.getRow = 0 pos.getCol = 0 player = ‘X’	**Output:** isPlayerAtPos = true State of the board is unchanged	**Reason:** This test case is unique and distinct because checks that the correct player is in the bottom row first column of the board **Function Name:** testIsPlayerAtPos_first_column_bottom_row

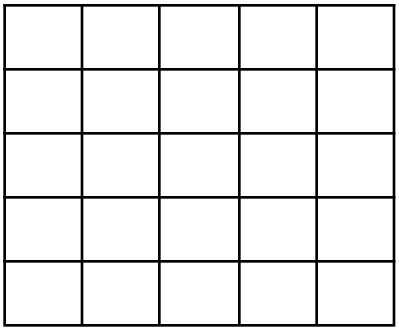
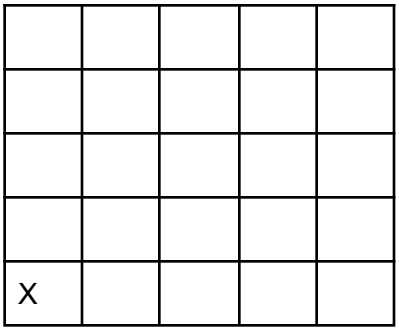
X	O	X	O	X

`boolean isPlayerAtPos(BoardPosition pos, char player)`

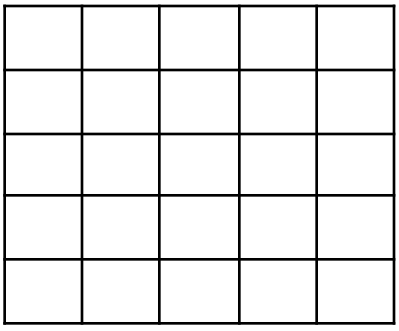
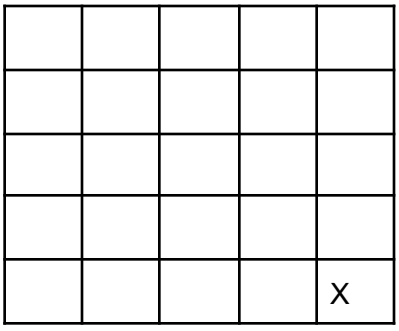
Input:	**Output:**	**Reason:**
State: (number to win = 4) |		|				X | |		|		|		O | |		|		|		X | |		|		|		O | |		|		|		X | pos.getRow = 4 pos.getCol = 4 player = ‘X’	isPlayerAtPos = true State of the board is unchanged	This test case is unique and distinct because it checks that the correct player is in the top row last column of the board. **Function Name:** testIsPlayerAtPos_top_row_last_column

dropToken

`void dropToken(char p, int c)`

Input:	Output:	Reason:
State: (number to win = 4)  c = 0 p = 'X'	State: (number to win = 4) 	This test case is unique and distinct because it checks that the first column and first available row is empty and places the player token on board. **Function Name:** testDropToken_first_column

`void dropToken(char p, int c)`

Input:	Output:	Reason:
State: (number to win = 4)  c = 4 p = 'X'	State: (number to win = 4) 	This test case is unique and distinct because it checks that the last column and first available row is empty and places the player token on board. **Function Name:** testDropToken_last_column

`void dropToken(char p, int c)`

Input: State: (number to win = 4) c = 2 p = 'X'	**Output:** State: (number to win = 4)	**Reason:** This test case is unique and distinct because it checks if the position in the middle of the board is empty and places it there. **Function Name:** testDropToken_middle_pos

O		X		
O		X		

		X		
O		X		
O		X		

`void dropToken(char p, int c)`

Input: State: (number to win = 4) c = 2 p = 'X'	**Output:** State: (number to win = 4)	**Reason:** This test case is unique and distinct because it is a column that is not empty and does not place the token in the already taken spot. **Function Name:** testDropToken_full_column

		X		
		O		
		X		
		O		
		X		

		X		
		O		
		X		
		O		
		X		

`void dropToken(char p, int c)`

Input:

State: (number to win = 5)

Q	R	S	T	
M	N	O	P	X
I	J	K	L	W
E	F	G	H	V
A	B	C	D	U

c = 4
p = ‘y’

Output:

State: (number to win = 5)

Q	R	S	T	Y
M	N	O	P	X
I	J	K	L	W
E	F	G	H	V
A	B	C	D	U

Reason:
This test case is unique and distinct because it checks the last available position is empty then places token in that position

Function Name:
testDropToken_last_pos

Deployment: Makefile Instructions

To compile the program, use the "make" command. To run the program, use the "make run" command. To compile the Testing files, use the "make test" command. To run the test cases for GameBoard, use the "make TestGB" command. To run the test cases for GameBoardMem, use the "make TestGBmem" command. To remove the compiled files, use the "make clean" command.