CPSC 2150 Project Report

Isaac Hine

Jerome Booth

Kevin Lin

Requirements Analysis

Functional Requirements:

1. As a player, I want to be prompted whether or not I'd like to play another game so I can continue playing or not.
2. As a player, I need to see the columns labeled in order so I know where to play my game piece.
3. As a player, I need to see a prompt so I know whenever it is my turn to play next.
4. As a player, I need to see the current board with all the pieces that have been played so I know what spaces are available to play.
5. As a player, I need to see text at the end of a game so I know if I have won.
6. As a player, I need to see text at the end of a game so I know if I have lost.
7. As a player, I need to see text at the end of a game so I know if I have tied.
8. As a player, I need to know which piece is mine and which is the opponent's so I know where to play.
9. As a player, I need to be able to pick which column to drop the token so I can play the game.
10. As a player, I need to be able to connect 5 in a row vertically in order to win.
11. As a player, I need to be able to connect 5 in a row horizontally in order to win.
12. As a player, I need to be able to connect 5 in a row diagonally in order to win.
13. As a player, I can redo my move if a column is full so I can still place my token.

Non-Functional Requirements

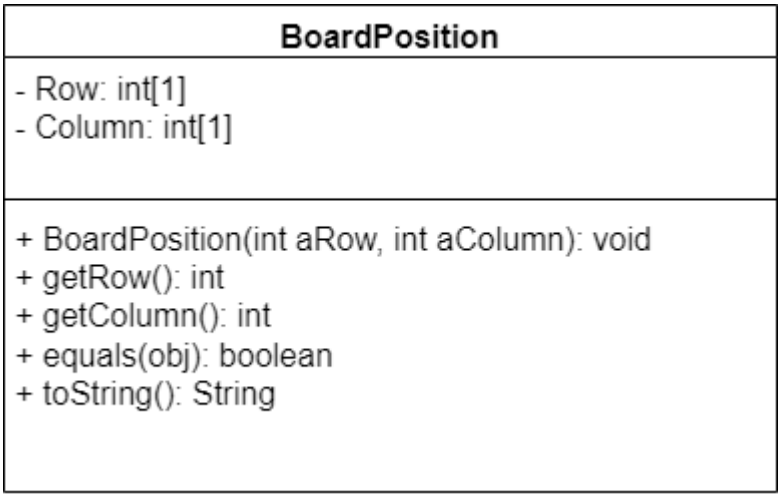
1. Runs on Unix/Must be a command-line application
2. Written in Java
3. X always goes first
4. (0,0) is at the bottom left of the board
5. The game must perform without failure for a prolonged period of time
6. The game must be able to be played on multiple platforms without little change
7. The game pieces must not go past the board size limit

8. The game must alternate turns after a move.
9. The game must show the updated version of the game after each move.
10. The gameboard size caps at 9x7.
11. The game is only for 2 players.
12. The game will detect connected wins.
13. The game will clear after a completed game.

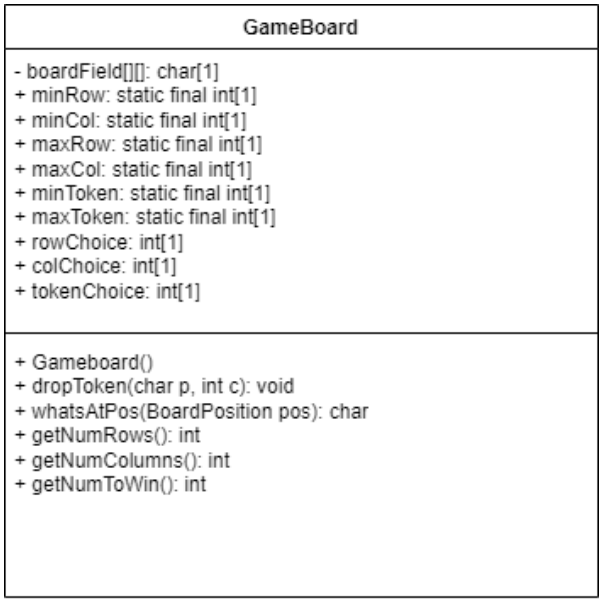
System Design

BoardPosition:

Class diagram

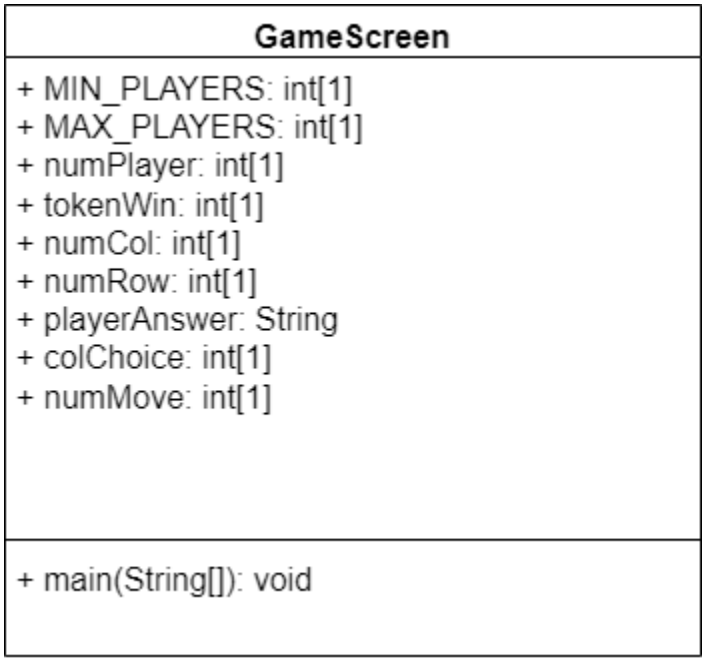


GameBoard:



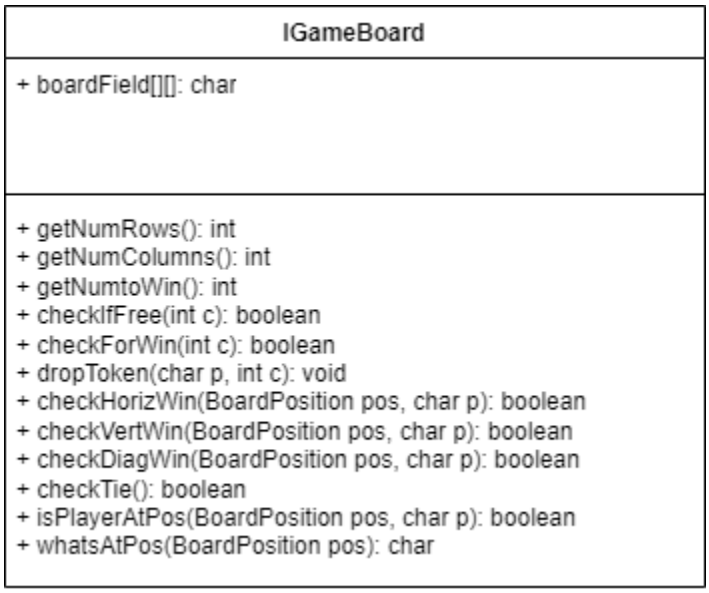
Class diagram

GameScreen:



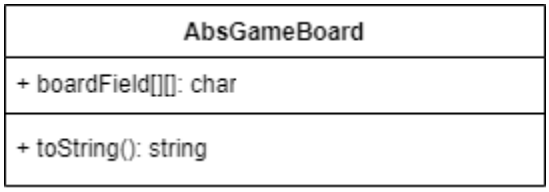
Class diagram

IGameBoard:



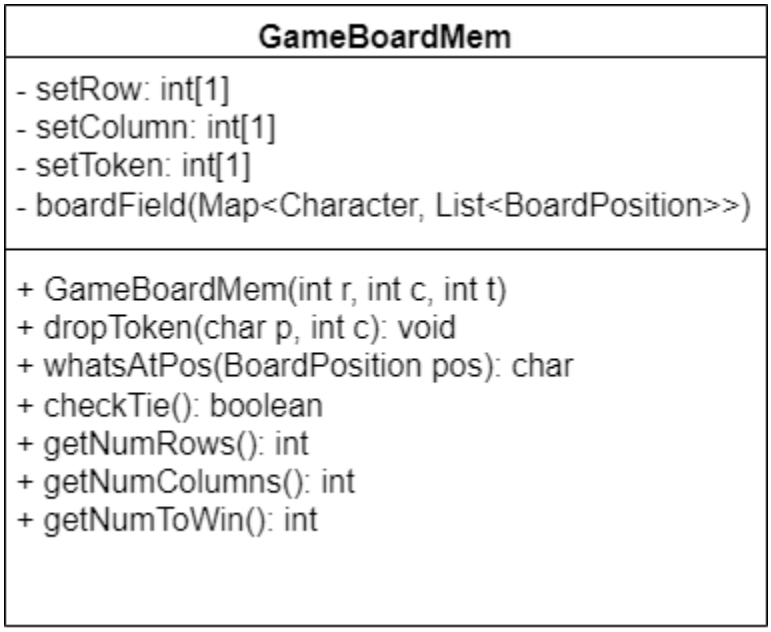
Class diagram

AbsGameBoard:



Class diagram

GameBoardMem:



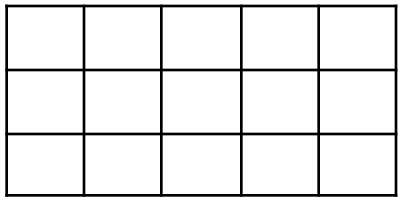
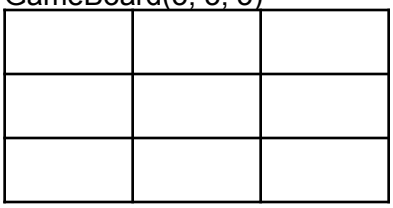
Class diagram

Test Cases

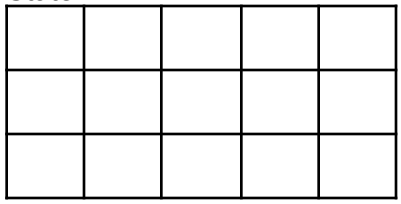
Test Cases:

Constructor: (`GameBoard(int r, int c, int t)`)

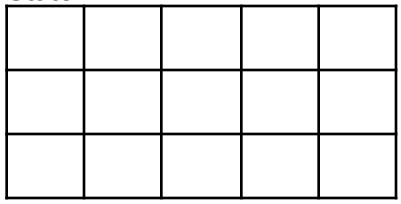
1.

Input: State:  r=3 c=3 t=3	**Output:** GameBoard(3, 3, 3) 	**Reason:** This test case is unique and distinct because the Gameboard initialization values are at the minimum. **Function Name:** testGameBoard_min_values_3_3_3

2.

Input: State:  r = 100 c = 100 t = 25	**Output:** GameBoard(100, 100, 25)	**Reason:** This test case is unique and distinct because the Gameboard initialization values are at the maximum. **Function Name:** testGameBoard_max_values_100_100_25

3.

Input: State:  r = 7 c =8 t =3	**Output:** GameBoard(7, 8, 3)	**Reason:** This test case is unique and distinct because the Gameboard is asymmetrical. **Function Name:** testGameBoard_varying_valesl_7_8_3

checkIfFree: (`boolean checkIfFree(int c)`)

1.

Input: State: X O X O O X O O X X O X X O c = 2	**Output:** checkIfFree = True State of the board is unchanged	**Reason:** This test case is unique and distinct because it is checking if the last available row in the column is available **Function Name:** testCheckifFree_true_last_space_2

2.

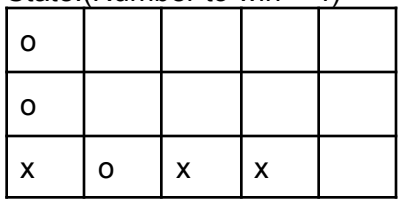
Input: State: X O O X X X X X O O O c = 1	**Output:** checkIfFree = False State of the board is unchanged	**Reason:** This test case is unique and distinct because it is checking if the only (full) column with tokens is free **Function Name:** testCheckifFree_false_full_col_1

3.

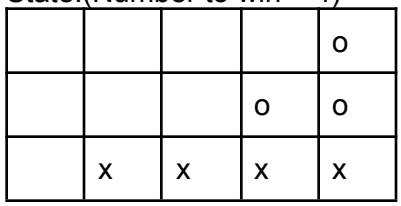
Input: State: c = 0	**Output:** checkIfFree = True State of the board is unchanged	**Reason:** This test case is unique and distinct because the entirety of the Gameboard has yet to be occupied. **Function Name:** testCheckifFree_true_empty_board_0

checkHorizWin: (`boolean checkHorizWin(BoardPosition pos, char p)`)

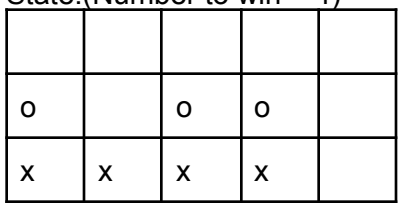
1.

Input: State:(Number to win = 4)  pos.getRow = 0 pos.getColumn = 3 p = ‘x’	**Output:** checkHorizWin = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the the would be ‘x’ win is broken in the middle by an ‘o’ token **Function name:** testCheckHorizWin_broken_by_o_end

2.

Input: State:(Number to win = 4)  pos.getRow = 0 pos.getColumn = 1 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 at the end of the string of 4 consecutive x’s, so the functions needs to count x’s on the left **Function name:** testCheckHorizWin_win_last_marker_left_end

3.

Input: State:(Number to win = 4)  pos.getRow = 0 pos.getColumn = 1 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 4 consecutive x’s so functions needs to count x’s on the left and right **Function name:** testCheckHorizWin_win_last_marker_middle

4.

Input: State:(Number to win = 4) x x x x o x o o x o o x o pos.getRow = 2 pos.getColumn = 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 at the end of the string of 4 consecutive x’s on the topmost row **Function name:** testCheckHorizWin_win_last_marker_left_end_top_row

checkVertWin: (`boolean checkVertWin(BoardPosition pos, char p)`)

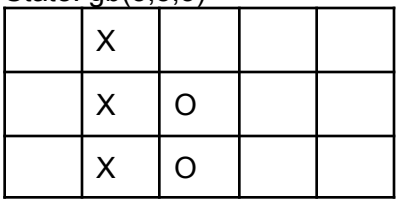
1.

Input: State: gb(3,5,3) X O X X O O X pos.getRow() = 2 pos.getCol() = 1 p = ‘X’	**Output:** checkVertWin = false State of the board is unchanged	**Reason:** This test case is unique and distinct because though there is a win, it is a diagonal, not vertical. **Function Name:** testcheckVertWin_false_diag_win_2_1_X

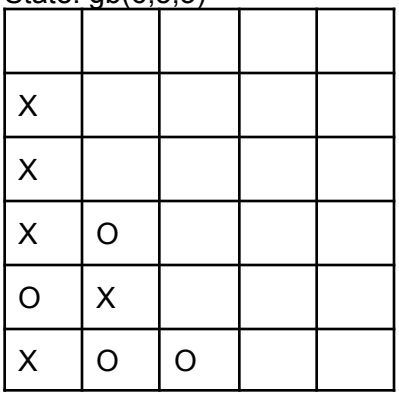
2.

Input: State: gb(3,5,3) X X O O X O X pos.getRow() = 2 pos.getCol() = 2 p = ‘X’	**Output:** checkVertWin = false State of the board is unchanged	**Reason:** This test case is unique and distinct because the vertical win is split in the middle (in the center column). **Function Name:** testCheckVertWin_false_split_vert_2_2_X

3.

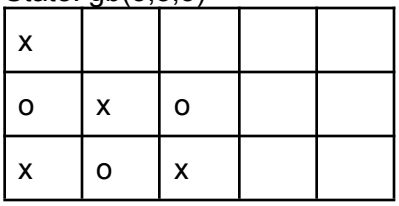
Input: State: gb(3,5,3)  pos.getRow() = 2 pos.getCol() = 1 p = 'X'	**Output:** checkVertWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because there is a vertical win for X **Function Name:** testCheckVertWin_false_O_win_2_1_X

4.

Input: State: gb(6,5,3)  pos.getRow() = 2 pos.getCol() = 0 p = 'X'	**Output:** checkVertWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because player X wins in the first column on top of O markers **Function Name:** testCheckVertWin_true_marker_X_first_column

checkDiagWIn: (`boolean checkDiagWin(BoardPosition pos, char p)`)

1.

Input: State: gb(3,5,3)  pos.getRow = 2 pos.getCol = 0	**Output:** checkDiagWin = True State of the board is unchanged	**Reason:** This test case is unique and distinct because x was placed at the end of a string of 3 consecutive x's at the top left so the function checks from the top left to bottom right **Function Name:** testCheckDiagWin_true_X_win_last_marker_top_left

2.

Input: State: (Number to win = 3) <table><tr><td></td><td></td><td>x</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>o</td><td>o</td><td></td><td></td></tr></table> pos.getRow = 2 pos.getCol = 2	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because x was placed at the end of a string of 3 consecutive x's at the top right so the function checks from the top right to bottom left **Function Name:** testCheckDiagWin_true_X_win_last_marker_top_right

3.

Input: State: (Number to win = 3) <table><tr><td></td><td></td><td>o</td><td></td><td></td></tr><tr><td></td><td>o</td><td>o</td><td></td><td>x</td></tr><tr><td>o</td><td>x</td><td>x</td><td></td><td>x</td></tr></table> pos.getRow = 0 pos.getCol = 0	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the 'o' was placed at the end of a string of 3 consecutive o's at the bottom left so the function checks from the bottom left to top right **Function Name:** testCheckDiagWin_true_O_win_last_marker_bottom_left

4.

Input: State: (Number to win = 3) <table><tr><td>o</td><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td>o</td><td></td><td>x</td><td></td></tr><tr><td>x</td><td>x</td><td>o</td><td>x</td><td></td></tr></table>pos.getRow = 0 pos.getCol = 2	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the 'o' was placed at the end of a string of 3 consecutive o's at the bottom right so the function checks from the bottom right to top left **Function Name:** testCheckDiagWin_true_O_win_last_marker_bottom_right

5.

Input: State: (Number to win = 3) <table><tr><td></td><td></td><td>x</td><td></td><td></td></tr><tr><td>o</td><td>x</td><td>x</td><td></td><td></td></tr><tr><td>x</td><td>o</td><td>o</td><td></td><td></td></tr></table> pos.getRow = 1 pos.getCol = 1	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the 'x' was placed in the middle of a string of 3 consecutive x's and the function checks the bottom left and top right in this case **Function Name:** testCheckDiagWin_true_X_win_middle_bottomleft_topright

6.

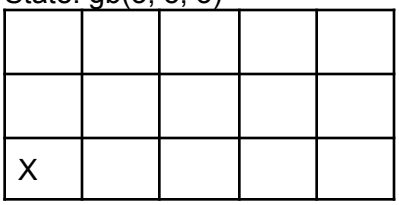
Input: State: (Number to win = 3) <table><tr><td>x</td><td></td><td></td><td></td><td></td></tr><tr><td>o</td><td>x</td><td>o</td><td></td><td></td></tr><tr><td>x</td><td>o</td><td>x</td><td></td><td></td></tr></table> pos.getRow = 1 pos.getCol = 1	**Output:** checkDiagWin = true State of the board is unchanged	**Reason:** This test case is unique and distinct because the 'x' was placed in the middle of a string of 3 consecutive o's and the function checks the bottom right and top left in this case **Function Name:** testCheckDiagWin_true_X_win_middle_bottomright_topleft

7.

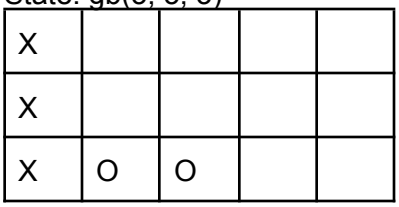
Input: State: (Number to win = 3) <table><tr><td></td><td></td><td>o</td><td></td><td></td></tr><tr><td>x</td><td>x</td><td>x</td><td></td><td></td></tr><tr><td>x</td><td>o</td><td>o</td><td></td><td></td></tr></table> pos.getRow = 1 pos.getCol = 1	**Output:** checkDiagWin = false State of the board is unchanged	**Reason:** This test case is unique and distinct because checkHorizWin = true when checkDiagWin = false **Function Name:** testCheckDiagWin_false_X_HorizWin_true

checkTie: (`boolean checkTie()`)

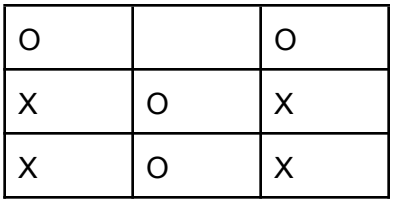
1.

Input: State: gb(3, 5, 3) 	**Output:** checkTie = False State of the board is unchanged	**Reason:** This test case is unique and distinct because it is checking for a tie after every move and is returning false as intended since there is only one token on the board **Function Name:** testcheckTie_Placement_token_tie_check

2.

Input: State: gb(3, 5, 3) 	**Output:** checkTie = False State of the board is unchanged	**Reason:** This test case is unique and distinct because the Gameboard is not full and assuming the amount of tokens to win is 3, player X would have won. **Function Name:** testcheckTie_false_checkVert_true

3.

Input: State: gb(3,3,3) 	**Output:** checkTie = False State of the board is unchanged	**Reason:** This test case is unique and distinct because the Gameboard is one token away from being filled and there is no win (assuming the amount of tokens to win is 3) **Function Name:** testcheckTie_false_one_before_win

4.

Input: State:State: gb(3,3,3)	**Output:**	**Reason:**
O X O X O X X O X	checkTie = True State of the board is unchanged	This test case is unique and distinct because the Gameboard is filled with no win in any direction **Function Name:** testcheckTie_true

whatsAtPos: (`char whatsAtPos(BoardPosition pos)`)

1.

Input: State:	**Output:**	**Reason:**
X O X O O X O X X O X O pos.getRow() = 0 pos.getCol() = 2	whatsAtPos = ‘ ‘ State of the board is unchanged	This test case is unique and distinct because the surrounding columns are completely filled with tokens while the selected BoardPosition column is empty **Function Name:** testwhatsAtPos_blank_empty_column_0_2

2.

Input: State:	**Output:**	**Reason:**
A C E D B E D C B A A B C D E pos.getRow() = 2 pos.getCol() = 1	whatsAtPos = ‘C‘ State of the board is unchanged	This test case is unique and distinct because there are multiple different tokens on the filled Gameboard **Function Name:** testwhatsAtPos_C_full_board_2_1

3.

Input: State: 					 pos.getRow() = 2 pos.getCol() = 0	**Output:** whatsAtPos = ‘ ‘ State of the board is unchanged	**Reason:** This test case is unique and distinct because while there are tokens on the board, the top-most row is empty. **Function Name:** testwhatsAtPos_blank_top_row_empty_2_0
O	X	O	X	O			
X	O	X	O	X			

4.

Input: State: 					 pos.getRow() = 0 pos.getCol() = 0	**Output:** whatsAtPos = ‘ ‘ State of the board is unchanged	**Reason:** This test case is unique and distinct because the Gameboard is empty, leaving all ‘ ‘ **Function Name:** testwhatsAtPos_blank_empty_board_0_0

5.

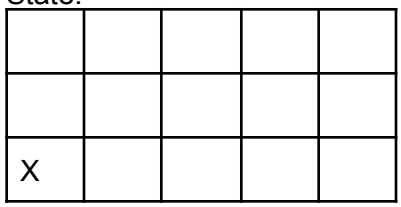
Input: State: 					 pos.getRow() = 0 pos.getCol() = 0	**Output:** whatsAtPos = ‘X’ State of the board is unchanged	**Reason:** This test case is unique and distinct because the Gameboard is only moderately filled. **Function Name:** testwhatsAtPos_X_slightly_filled_board_0_0
	X		O				
X	O	X	O	X			

isPlayerAtPos: (`boolean isPlayerAtPos(BoardPosition pos, char player)`)

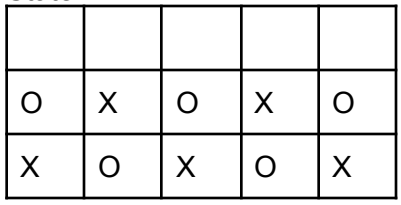
1.

Input: State: 					 pos.getRow() = 0 pos.getCol() = 0 player = ‘X’	**Output:** isPlayerAtPos = False State of the board is unchanged	**Reason:** This test case is unique and distinct because there are no player tokens on the GameBoard. **Function Name:** testisPlayerAtPos_false_empty_board_0_0_X

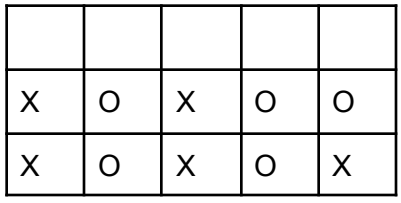
2.

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 and distinct because X is the only player token placed and the sole token on the Gameboard. **Function Name:** testisPlayerAtPos_true_one_token_0_0_X

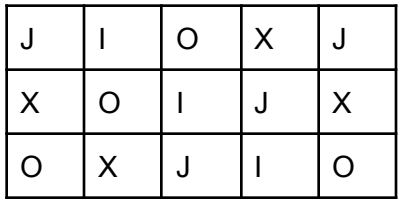
3.

Input: State:  pos.getRow() = 1 pos.getCol() = 2 player = ‘X’	**Output:** isPlayerAtPos = False State of the GameBoard is unchanged	**Reason:** This test case is unique and distinct because ‘X’ is on the board numerous times, but not in the BoardPosition being called upon. **Function Name:** testisPlayerAtPos_false_not_in_that_postion_1_2_X

4.

Input: State:  pos.getRow() = 0 pos.getCol() = 4 player = ‘X’	**Output:** isPlayerAtPos = True State of the Gameboard is unchanged	**Reason:** This test case is unique and distinct because the board is moderately filled with two player tokens. **Function Name:** testisPlayerAtPos_true_near_full_board_0_4_X

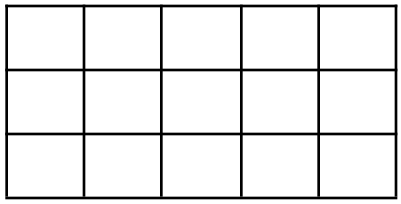
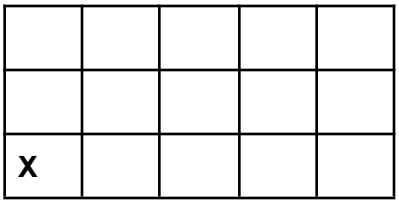
5.

Input: State:  pos.getRow() = 0 pos.getCol() = 0	**Output:** isPlayerAtPos = True State of the Gameboard is unchanged	**Reason:** This test case is unique and distinct because there are multiple different tokens on the Gameboard. **Function Name:** testisPlayerAtPos_true_full_board_many_tokens_0_0_O

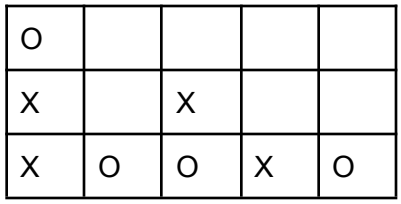
player = ‘O’		

dropToken: (`void dropToken(char p, int c)`)

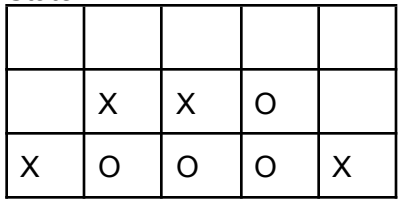
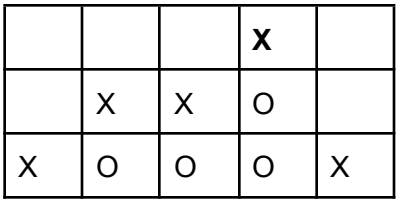
1.

Input: State:  p = ‘X’ c = 0	**Output:** State: 	**Reason:** This test case is unique and distinct because I am placing my token in a column of an empty GameBoard. **Function Name:** testdropToken_empty_board_X_0

2.

Input: State:  p = ‘X’ c = 0	**Output:** State of the board is unchanged	**Reason:** This test case is unique and distinct because I am attempting to place my token in the only full column. **Function Name:** testdropToken_full_col_X_0

3.

Input: State:  p = ‘X’ c = 3	**Output:** 	**Reason:** This test case is unique and distinct because I am attempting to place my token in a nearly full column. **Function Name:** testdropToken_last_col_space_3

4.

Input:
State:

	O	X	O	
X	O	X	O	X

p = ‘X’
c = 2

Output:

		X		
	O	X	O	
X	O	X	O	X

Reason:

This test case is unique because I am placing my token in a moderately filled column.

Function Name:
testdropToken_moderately_full_col_X_2

5.

Input:
State:

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

p = ‘X’
c = 0

Output:
State:

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

Reason:

This test case is unique and distinct because I am placing my token in the last available row of the nearly full column.

Function Name:
testdropToken_almost_full_board_X_0