Requirement Analysis:

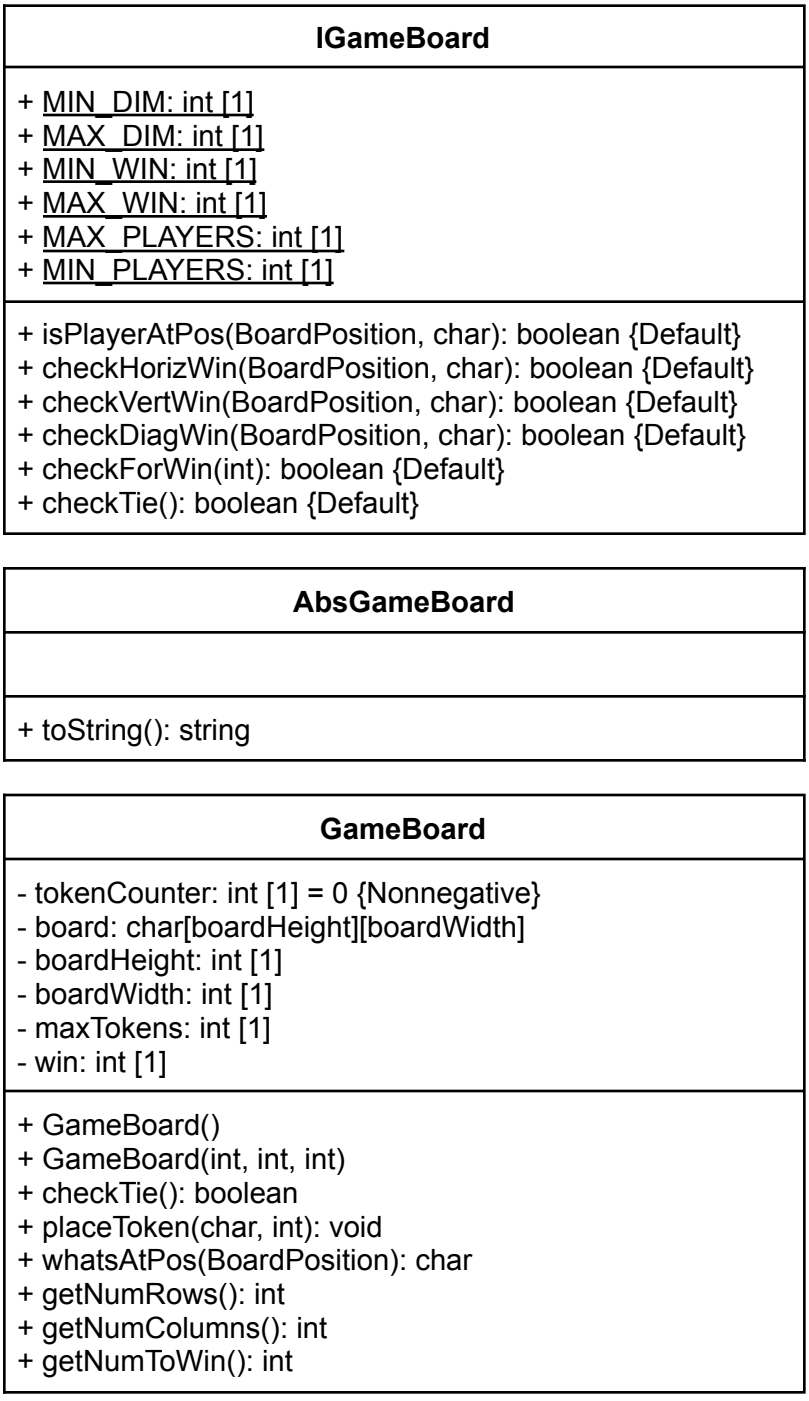
Functional Requirements:

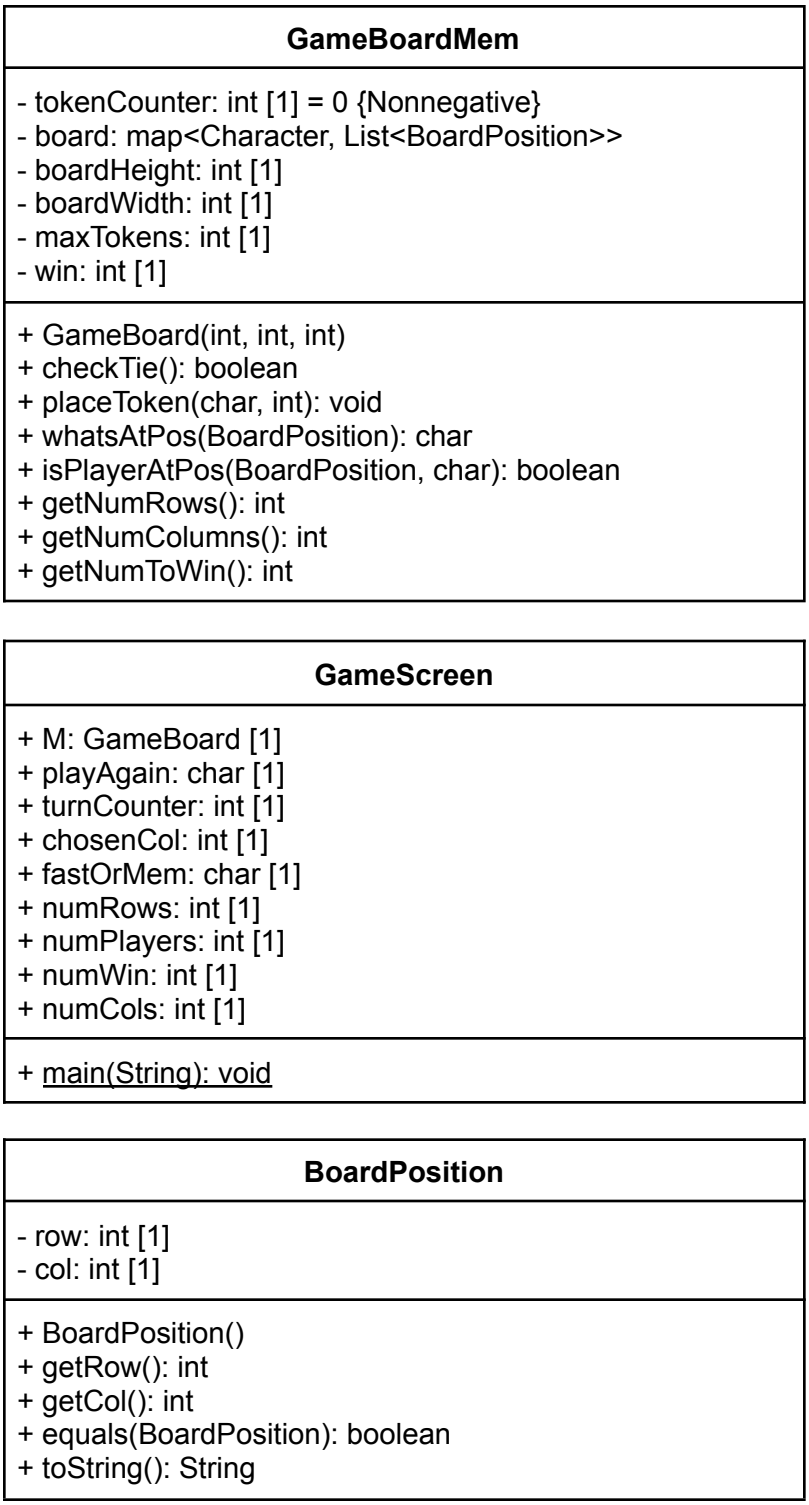
- As a player, I can choose a player that can be any character other than ‘ ‘ to determine which tokens I will use.
- As a player, I can place a token in a chosen column to try and connect 5 horizontally.
- As a player, I can place a token in a chosen column to try and connect 5 vertically.
- As a player, I can place a token in a chosen column to try and connect 5 diagonally.
- As a player, I can place enough tokens to reach the max count which will result in a tie condition.
- As a player, I can decide whether or not to play again at the end of a game by typing ‘y’ or ‘n’.
- As a player, I swap turns with another player, placing a token after the other player does so that the game is fair.
- As a player, I can select any column to place a token in, but if I am out of bounds when placing I will be prompted to place again, so that I do not waste a turn.
- As a player, I can choose the number of rows that the game board will have to add another dimension of variability to the game.
- As a player, I can choose the number of columns that the game board will have to add another dimension of variability to the game.
- As a player, I can choose the number of tokens needed to win the game to add another dimension of variability to the game.
- As a player, I can choose a faster version of the game or a memory efficient version of the game to play.
- As a player, I have access to seeing the game board to determine my next token placement.
- The game must accept column integer input from the user.
- The game must check to make sure that a column is not full when a player tries to place a token.
- The game must check to see if a player has won by connecting 5 tokens in a row horizontally, vertically, or diagonally.

Non-Functional Requirements:

- The game executable must run on Unix.
- The game must be between 3x3 and 100x100 board of characters.
- The game must allow player 1 to go first.
- The game must allow <0>,<0> to be the bottom left board position on the board.
- The game code must be able to compile in Java 11.
- The game code must be able to run in Java 11.

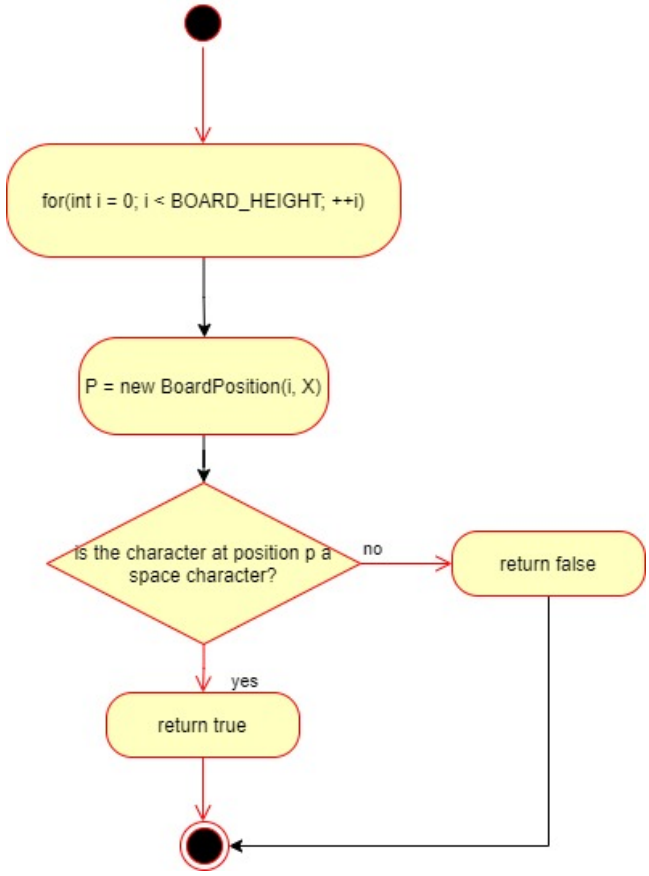
UML Class Diagrams:



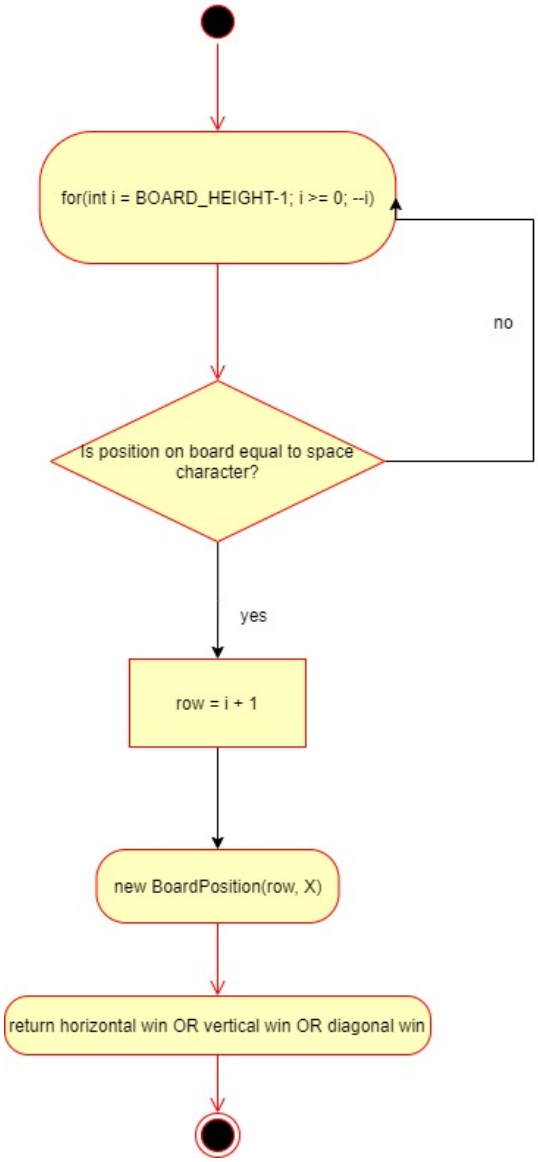

GameBoardMem
- tokenCounter: int [1] = 0 {Nonnegative}
- board: map<Character, List<BoardPosition>>
- boardHeight: int [1]
- boardWidth: int [1]
- maxTokens: int [1]
- win: int [1]
+ GameBoard(int, int, int)
+ checkTie(): boolean
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
GameScreen
+ M: GameBoard [1]
+ playAgain: char [1]
+ turnCounter: int [1]
+ chosenCol: int [1]
+ fastOrMem: char [1]
+ numRows: int [1]
+ numPlayers: int [1]
+ numWin: int [1]
+ numCols: int [1]
+ main(String): void
BoardPosition
- row: int [1]
- col: int [1]
+ BoardPosition()
+ getRow(): int
+ getCol(): int
+ equals(BoardPosition): boolean
+ toString(): String

UML Activity Diagrams:

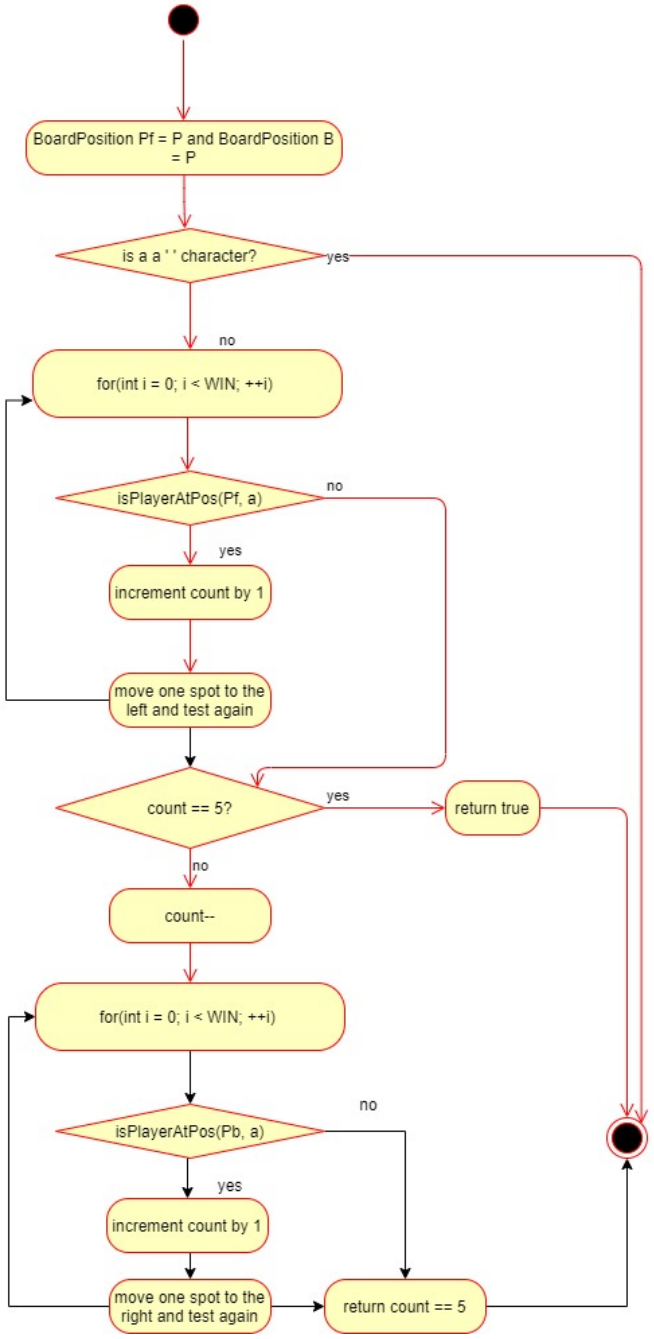
checkIfFree(int): boolean {default}



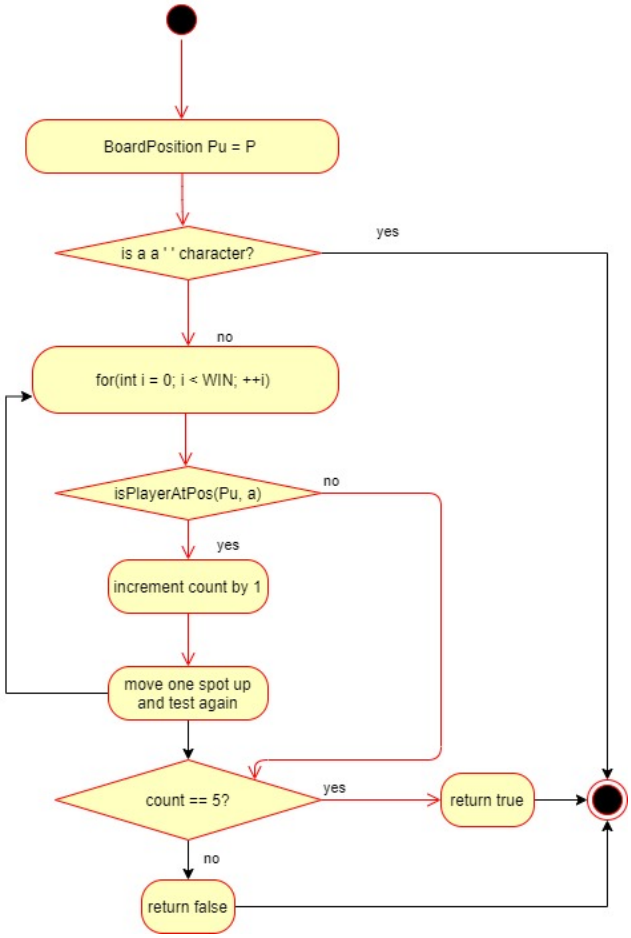
checkForWin(int): boolean {default}



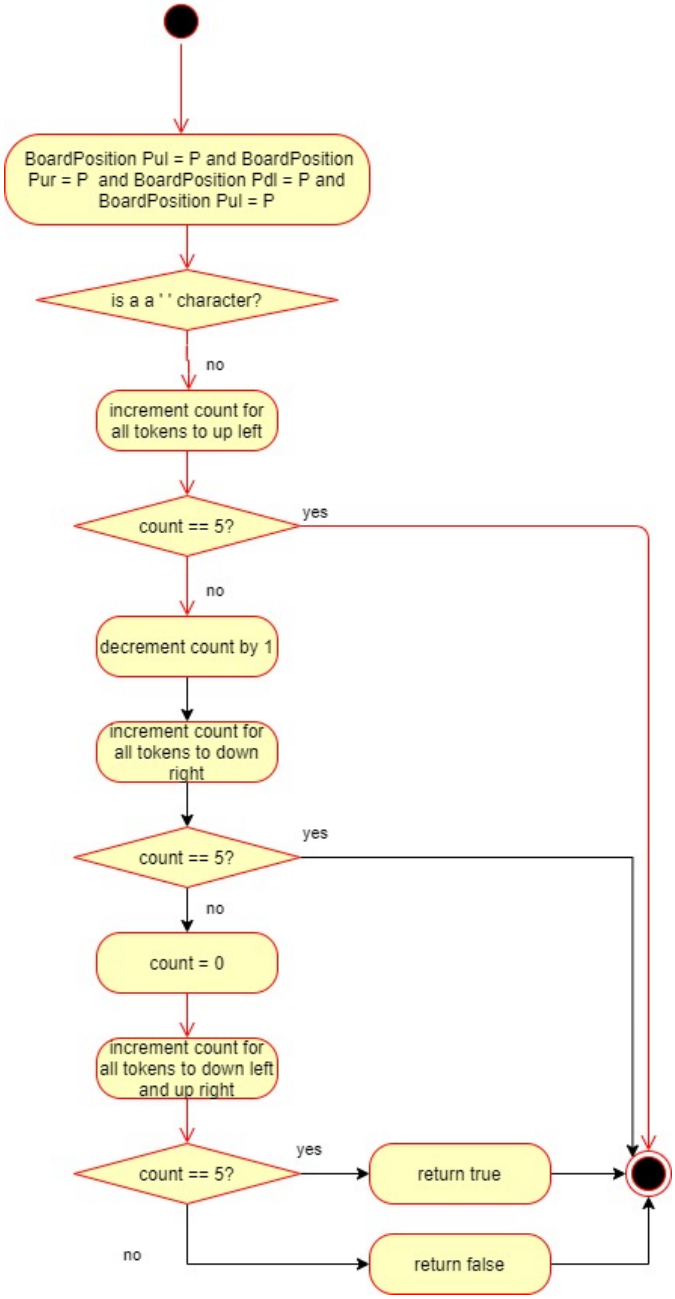
checkHorizWin(BoardPosition, char): boolean {default}



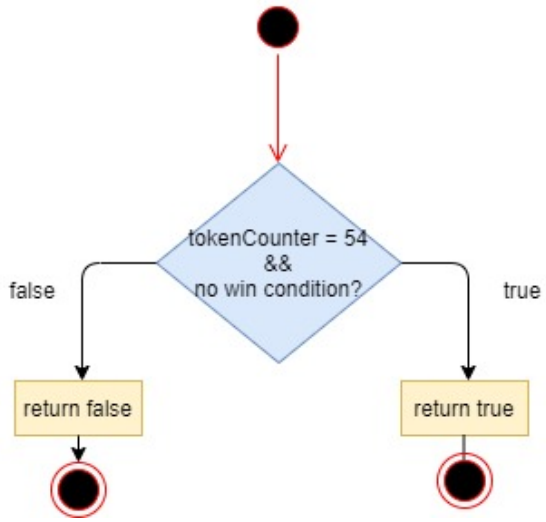
checkVertWin(BoardPosition, char): boolean {default}



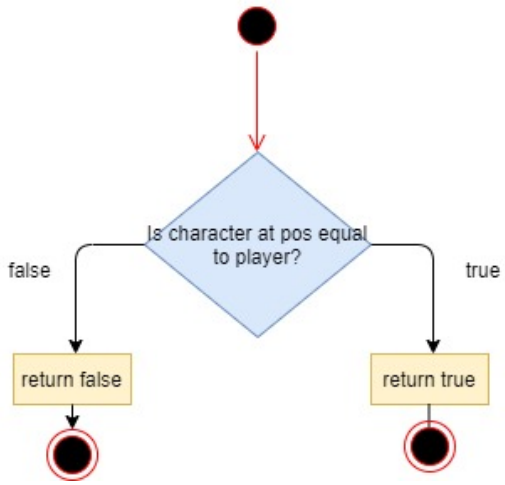
checkDiagWin(BoardPosition, char): boolean {default}



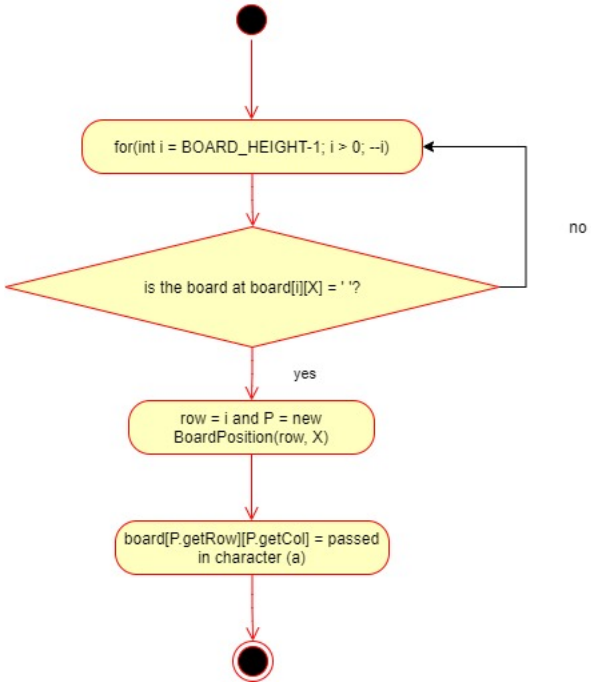
checkTie(): boolean (GameBoard and GameBoardMem)



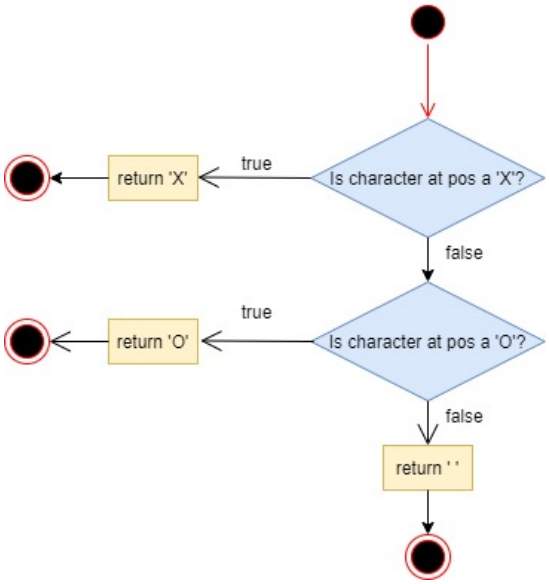
isPlayerAtPos(BoardPosition, char): boolean {default}



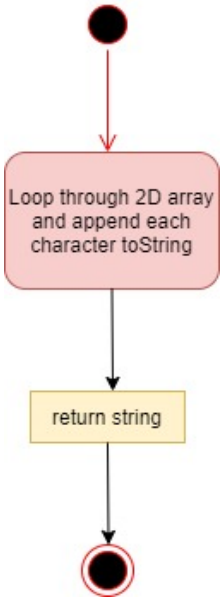
placeToken(char, int): void (GameBoard)



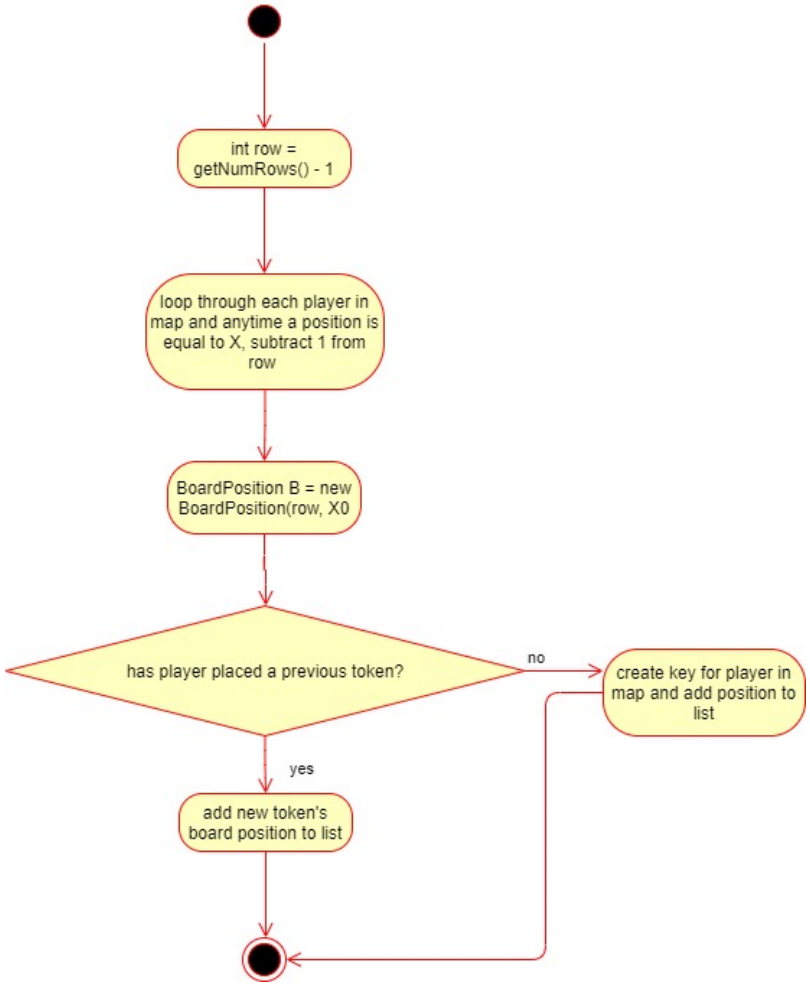
whatsAtPos(BoardPosition): char (GameBoard)



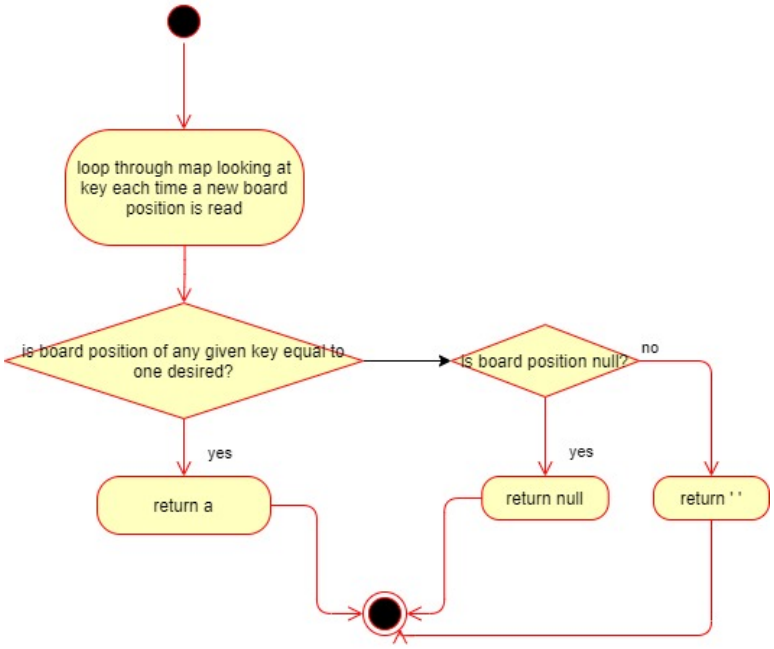
toString(): String



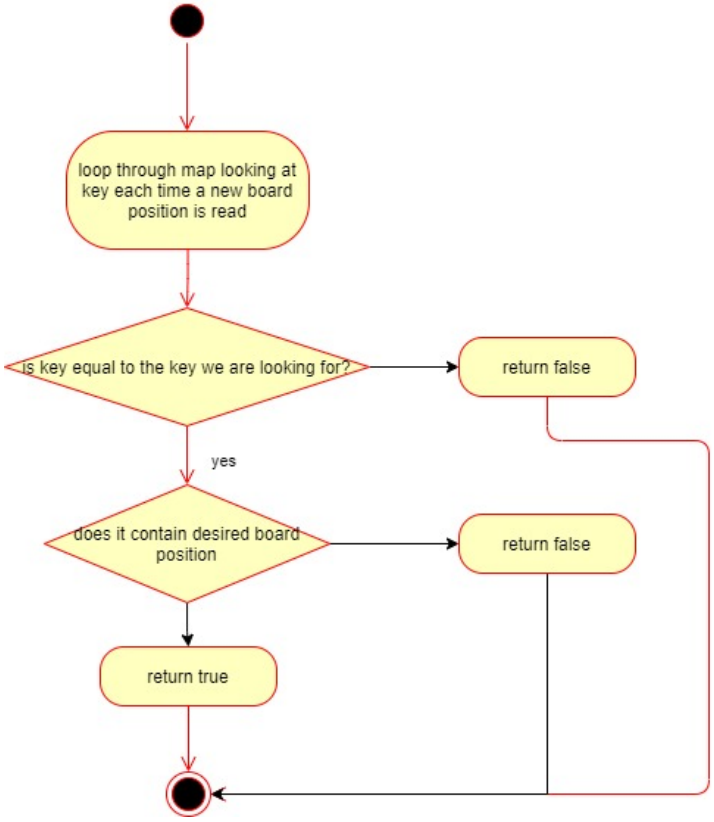
placeToken(Character, int): void (GameBoardMem)



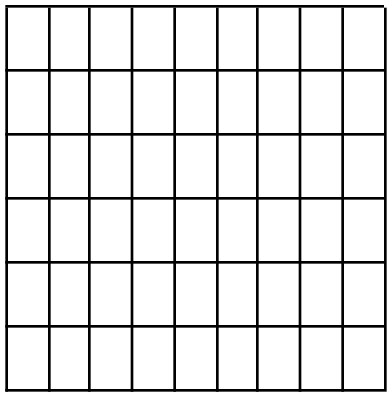
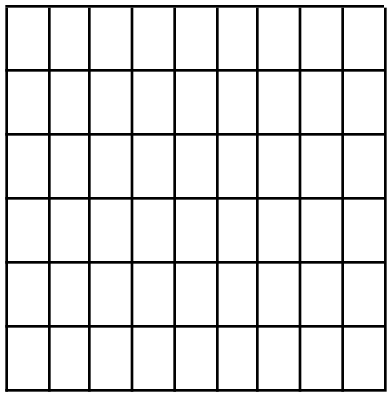
whatsAtPos(BoardPosition): Character (GameBoardMem)

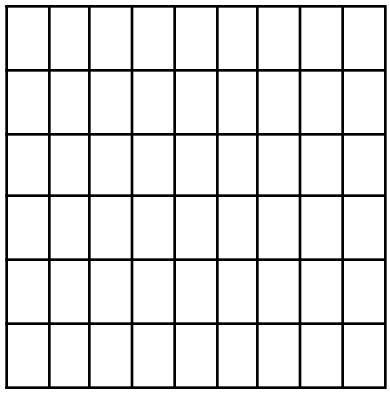
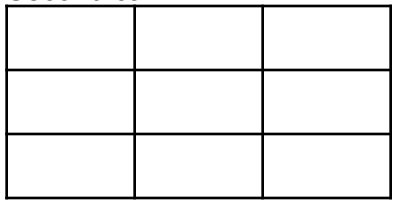
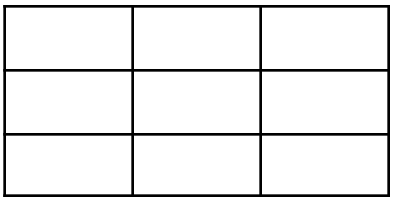


isPlayerAtPos(BoardPosition, char): boolean (GameBoardMem)

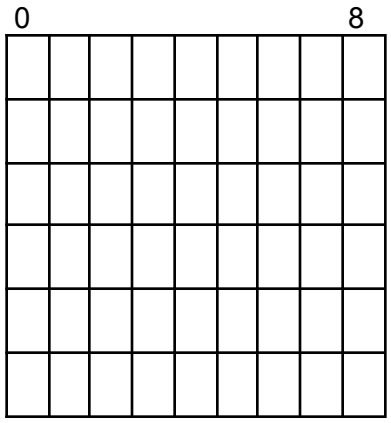


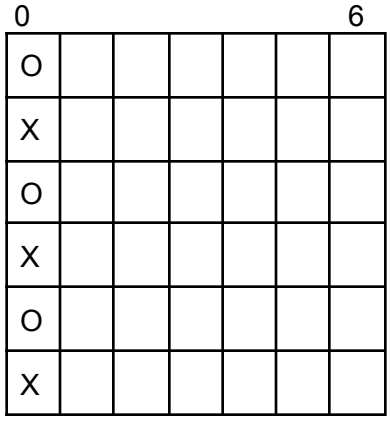
Test Cases:

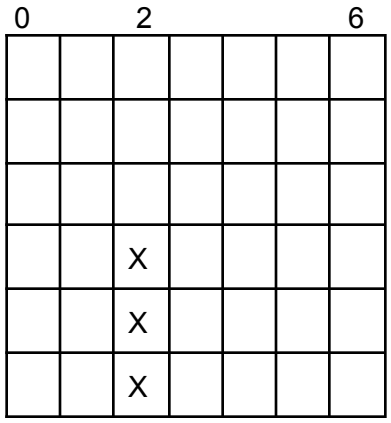
Input:	**Output:**	**Reason:**
State:  Board initialized to empty	 Expected string	This test case is unique because it is testing the constructor to initialize to a 6x9 board that is empty. **Function Name:** test_constructor_normal695

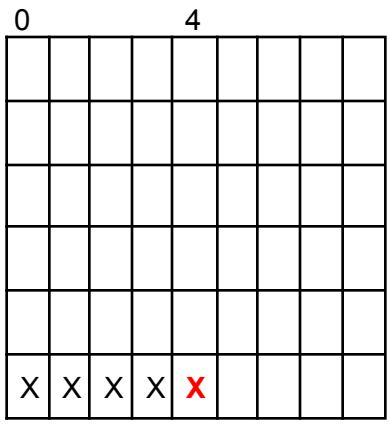
Input:	**Output:**	**Reason:**
State: First call:  Second call:  Board initialized to empty	 Expected string	This test case is unique because it is testing the resizability of the board. The board was first set to be an empty 6x9 board, but was then set to an empty 3x3 board and tested. It is also testing the minimum conditions of the game board. **Function Name:** test_constructor_resize

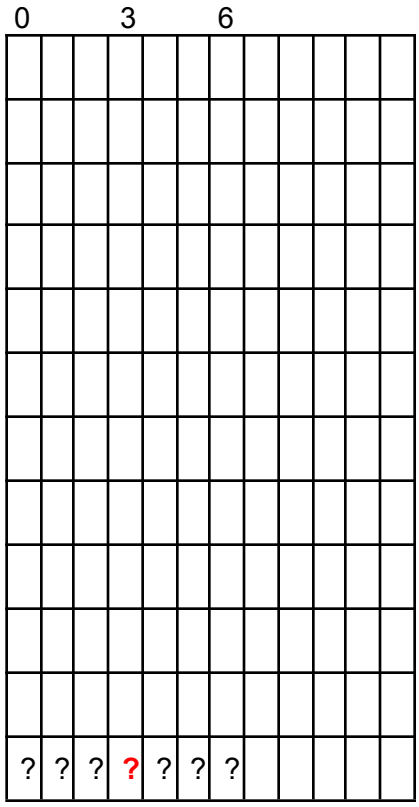
Input:	**Output:**	**Reason:**
State: 100x100 GameBoard object initialized (Too large to insert table) Board initialized to empty	100x100 table is expected Expected string	This test case is unique because it is testing the max conditions set for a GameBoard. Programs tend to crash with very large memory access, so testing to make sure the program runs well with max conditions is a unique case. **Function Name:** test_constructor_max10010025

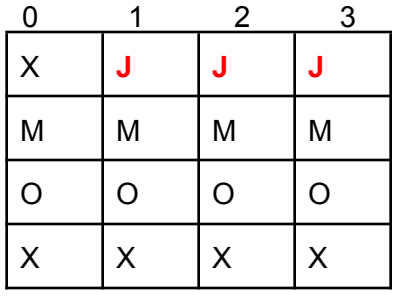
Input:	**Output:**	**Reason:**
State:  Board initialized to empty	checkIfFree = true State of board is unchanged	The test case is unique because it is an empty board with only space characters (fast version). No matter what column (0 < col < 8) checkIfFree is called it should return true. **Function Name:** test_checkIfFree_empty

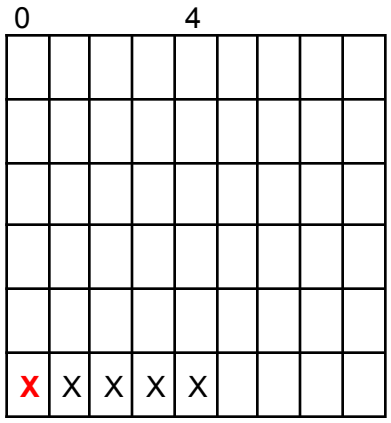
Input:	**Output:**	**Reason:**
State:  	checkIfFree = false State of board is unchanged	This test case is unique because it is testing the checkIfFree function on a column that is full of tokens. checkIfFree will return true if the column is greater than 0, but the column of 0 is filled, so checkIfFree will return false for that column. **Function Name:** test_checkIfFree_fullColumn

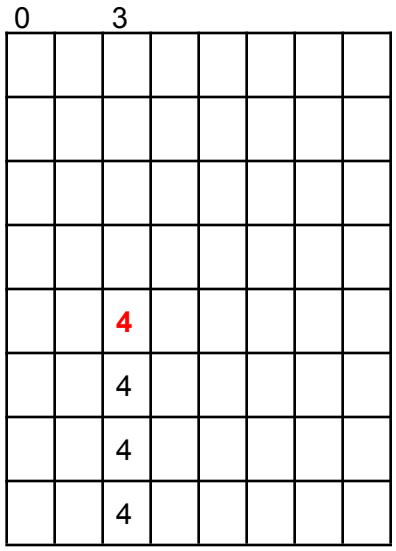
Input:	**Output:**	**Reason:**
State:  	checkIfFree = true State of board is unchanged	This test case is unique because it tests the checkIfFree function on a column that has tokens placed in it, but is not full. Since the column is not full checkIfFree on this column will return true. **Function Name:** test_checkIfFree_halfColumn

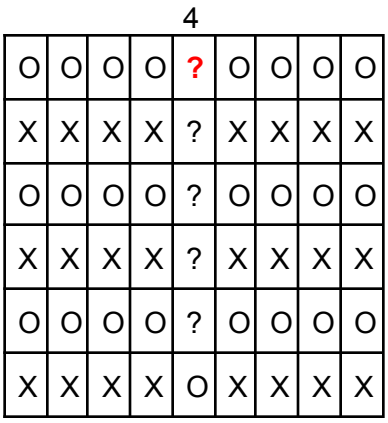
Input:	**Output:**	**Reason:**
State: (win = 5)  	checkHorizWin = true State of board is unchanged	This test case is unique because it tests checkHorizWin when the final token placed in the segment is on the right side of the horizontal. Counting will only take place on tokens to the left. **Function Name:** test_horizWin_rightSide

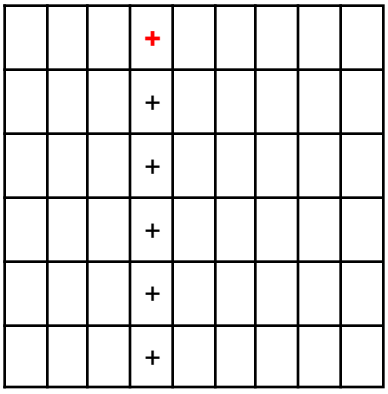
Input:	**Output:**	**Reason:**
State: (win = 7)  	checkHorizWin = true State of board is unchanged	This test case is unique because it tests checkHorizWin when the final token is placed in the middle of the segment. Counting will take place on the left and right sides of the final token placement. **Function Name:** test_horizWin_middle

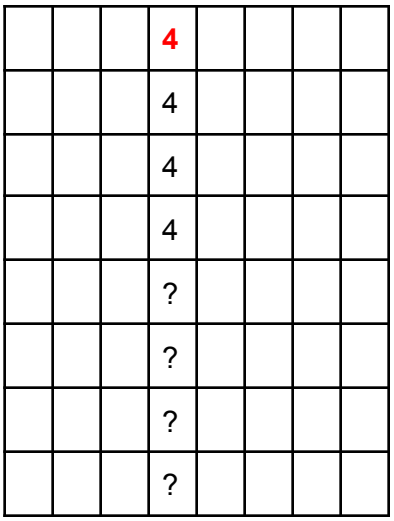
Input:	**Output:**	**Reason:**
State: (win = 3)  	checkHorizWin = true State of board is unchanged	This test case is unique because it is testing checkHorizWin whenever there is a full board **Function Name:** test_horizWin_fullBoard

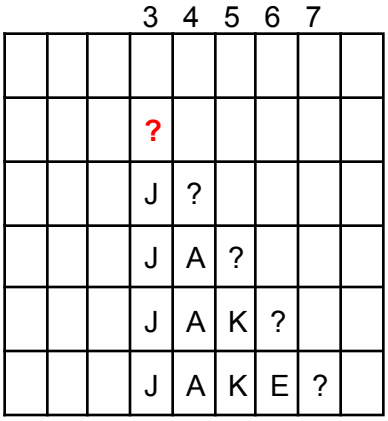
Input:	**Output:**	**Reason:**
State: (win = 5)  	checkHorizWin = true State of board is unchanged	This test case is unique because it is testing the checkHorizWin on the left side of the segment. In this case, counting will only occur to the right since the final token was placed on the left side of the horizontal. **Function Name:** test_horizWin_leftSide

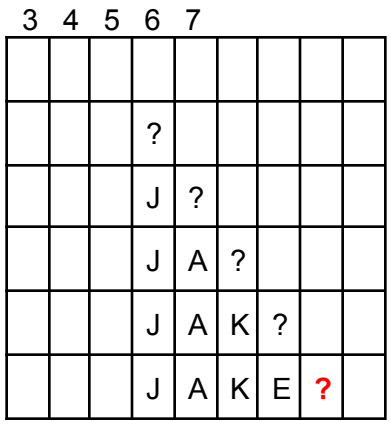
Input:	**Output:**	**Reason:**
State: (win = 4)  	checkVertWin = true State of board is unchanged	This test case is unique because it is testing the checkVertWin condition on the top of the vertical line. The token placed is on the top, so the counting will occur on tokens below the top token. **Function Name:** test_vertWin_bottomBoard

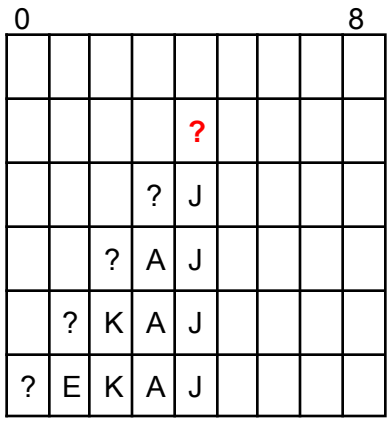
Input:	**Output:**	**Reason:**
State: (win = 5) 	checkVertWin = true State of board is unchanged	This test case is unique because it is testing the checkVertWin on a board that is maxed out on tokens. This is distinct because it could fail if checkTie were to execute before checkVertWin did. **Function Name:** test_checkVertWin_fullBoard

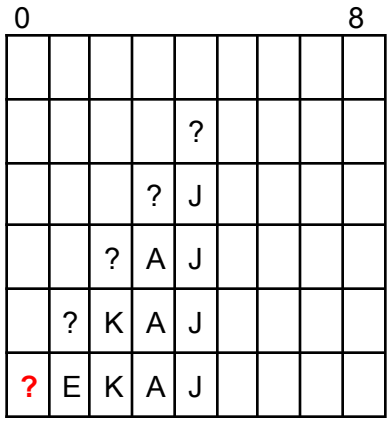
Input:	**Output:**	**Reason:**
State: (win = 5) 	checkVertWin = true State of board is unchanged	This test case is unique because it is testing the checkVertWin on a board where the number of connected vertical tokens is greater than the win condition. This should still return true. **Function Name:** test_vertWin_moreTokens_thanW

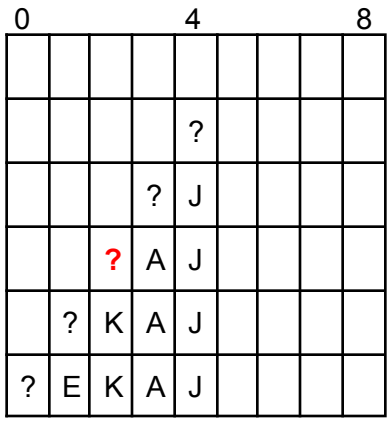
Input:	**Output:**	**Reason:**
State:  	checkVertWin = true State of board is unchanged	This test case is unique because it is testing checkVertWin on a board where the top token is placed at the very top of the board. If there is nothing wrong with the borders of the board then this will return true. **Function Name:** test_vertWin_topBoard

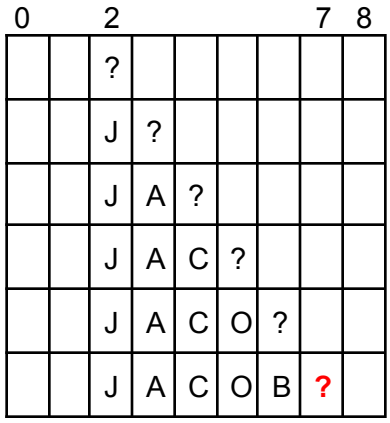
Input:	**Output:**	**Reason:**
State: (win = 5)  	checkDiagWin = true State of board is unchanged	This test case is unique because it is testing checkDiagWin to the downright starting at the top. Meaning that only the tokens to the down right will be counted and it will return true. **Function Name:** test_diagWin_downRight

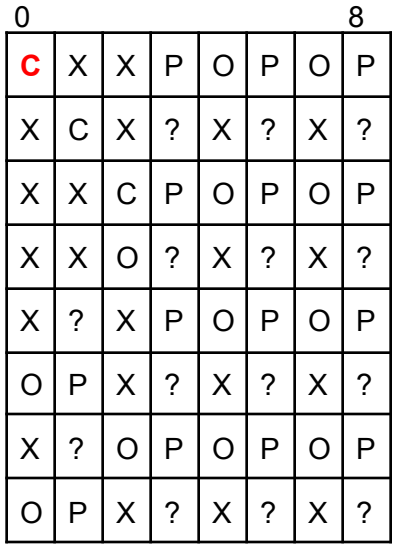
Input:	**Output:**	**Reason:**
State: (win = 5)  	checkDiagWin = true State of board is unchanged	This test case is unique because it is testing checkDiagWin to the up left starting at the bottom. Meaning that only the tokens to the up left will be counted and it will return true. **Function Name:** test_diagWin_upLeft

Input:	**Output:**	**Reason:**
State: (win = 5)  	checkDiagWin = true State of board is unchanged	This test case is unique because it is testing checkDiagWin to the down left starting at the top. Meaning that only the tokens to the down left will be counted and it will return true. **Function Name:** test_diagWin_downLeft

Input:	**Output:**	**Reason:**
State: (win = 5)  	checkDiagWin = true State of board is unchanged	This test case is unique because it is testing checkDiagWin to the upright starting at the bottom. Meaning that only the tokens to the upright will be counted and it will return true. **Function Name:** test_diagWin_upRight

Input:	**Output:**	**Reason:**
State: (win = 5)  	checkDiagWin = true State of board is unchanged	This test case is unique because it is testing checkDiagWin to the upright and down left starting at the middle. Meaning that the tokens to the upright and down left will be counted and it will return true. **Function Name:** test_diagWin_middle

Input:	**Output:**	**Reason:**
State: (win = 5)  	checkDiagWin = true State of board is unchanged	The test case is unique because it tests checkDiagWin whenever there are more tokens than necessary to create a diagonal win condition. This should still return true. **Function Name:** test_diagWin_moreTokens_thanW

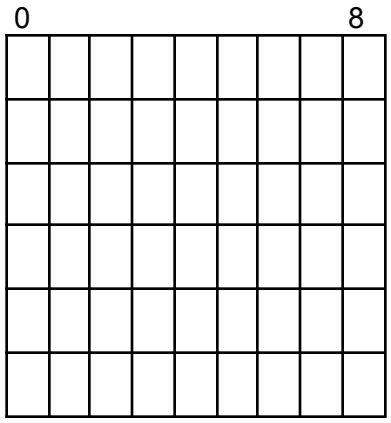
Input:	**Output:**	**Reason:**
State: (win = 3)  	checkDiagWin = true State of board is unchanged	The test case is unique because it tests checkDiagWin whenever the board is full. **Function Name:** test_diagWin_fullBoard

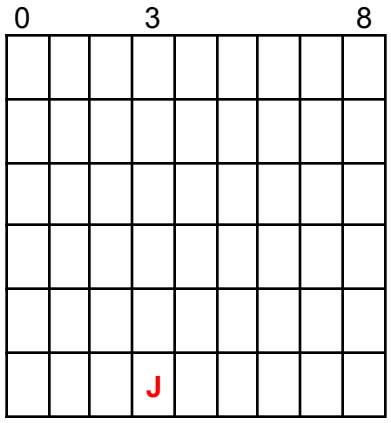
Input:	**Output:**	**Reason:**
State: 0 8 X X X X ? X X X X ? ? ? ? X ? ? ? ? X X X X ? X X X X ? ? ? ? X ? ? ? ? X X X X ? X X X X ? ? ? ? X ? ? ? ?	checkTie = true State of board is unchanged	The test case is unique because it is a filled board with no win condition. This board is standard 6x9 and does not violate pre conditions for win. **Function Name:** test_checkTie_fullBoard

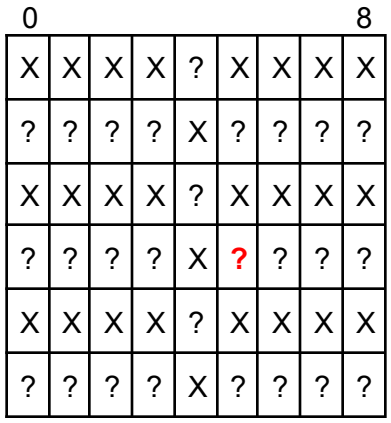
Input:	**Output:**	**Reason:**
State: 0 1 2 X O X O X O X O X	checkTie = true State of board is unchanged	The test case is unique because it is a minimum capacity board that does not contain a win condition. **Function Name:** test_checkTie_finalTokenPlaced_minBoard

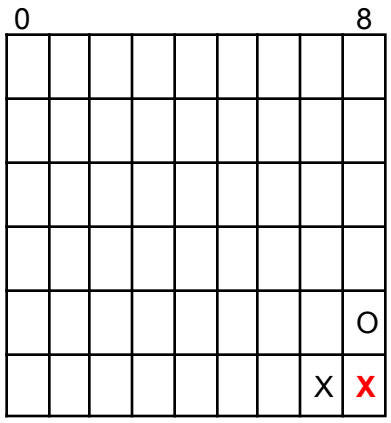
Input:	**Output:**	**Reason:**
State: 100x100 board with tokens in every slot (Too large to display here)	checkTie = true State of board is unchanged	The test case is unique because it is a maximum board size filled with tokens meaning that there is a lot of memory being taken up. This test is to ensure that the program does not crash whenever conditions are met that could potentially cause a crash. **Function Name:** test_checkTie_maxBoard

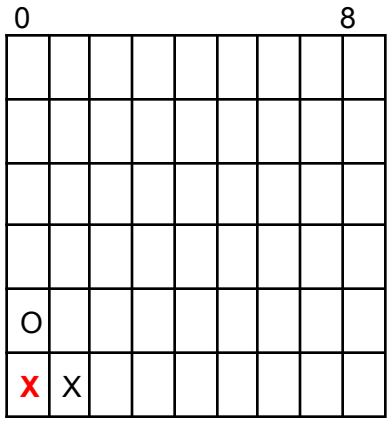
Input:	**Output:**	**Reason:**
State: 25x25 empty board (Too large to display here)	checkTie = false State of board is unchanged	The test case is unique because it tests checkTie whenever a tie condition should not be met. The board is empty and a tie condition can only occur when the board is full. **Function Name:** test_checkTie_emptyBoard

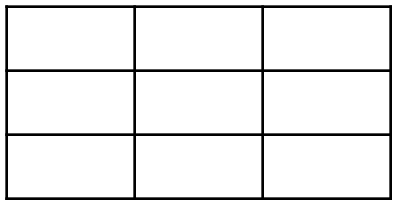
Input:	**Output:**	**Reason:**
State:  Board initialized to empty	Expected character = ‘ ‘ Result character = ‘ ‘ State of board is unchanged	The test case is unique because it tests whatsAtPos on an empty board. The character that should be returned is a ‘ ‘. **Function Name:** test_whatsAtPos_space

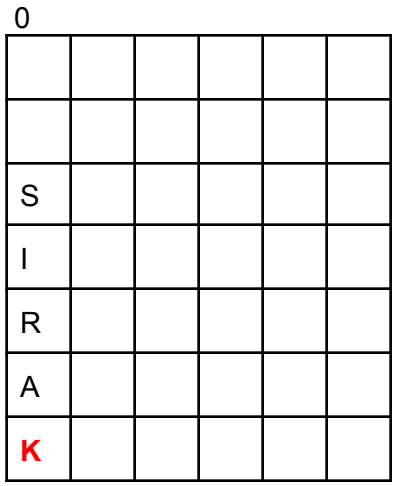
Input:	**Output:**	**Reason:**
State:  	Expected character = ‘J‘ Result character = ‘J‘ State of board is unchanged	The test case is unique because it tests whatsAtPos on a board with one character. The character that should be returned is a ‘J‘. **Function Name:** test_whatsAtPos_character

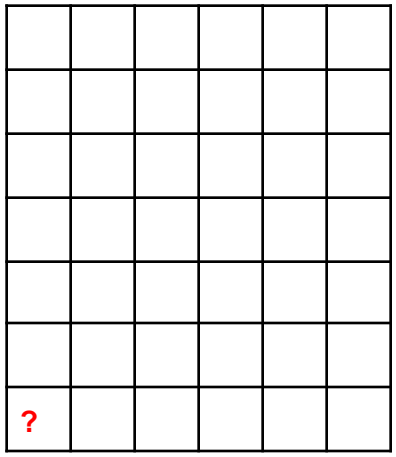
Input: State:  	**Output:** Expected character = ‘?‘ Result character = ‘?‘ State of board is unchanged	**Reason:** The test case is unique because it tests whatsAtPos on a board that is full. The character that should be returned is a ‘?‘. **Function Name:** test_whatsAtPos_fullBoard

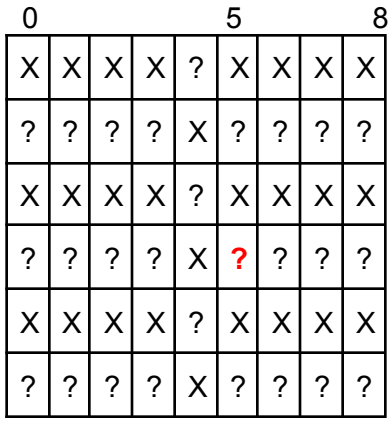
Input: State:  	**Output:** Expected character = ‘X‘ Result character = ‘X‘ State of board is unchanged	**Reason:** The test case is unique because it tests whatsAtPos on a board with multiple characters and the desired character placed in the bottom right corner. **Function Name:** test_whatsAtPos_bottomRight

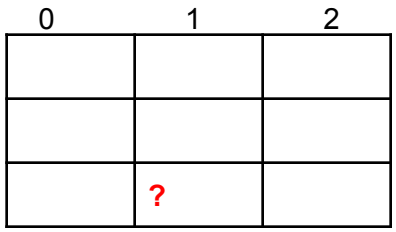
Input:	**Output:**	**Reason:**
State:  	Expected character = ‘X‘ Result character = ‘X‘ State of board is unchanged	The test case is unique because it tests whatsAtPos on a board with multiple characters and the desired character placed in the bottom left corner. **Function Name:** test_whatsAtPos_bottomLeft

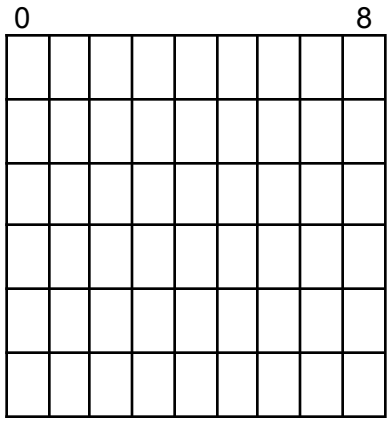
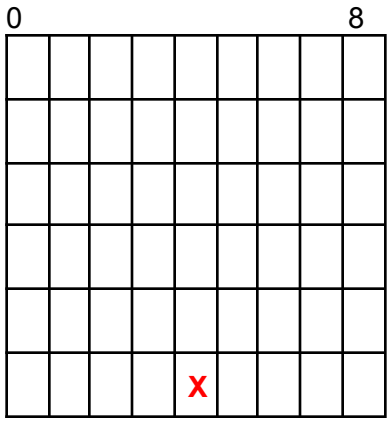
Input:	**Output:**	**Reason:**
State:  Board initialized to empty	isPlayerAtPos = false Expected character = ‘?’ Result character = ‘ ‘ State of board is unchanged	The test case is unique because it is testing isPlayerAtPos on an empty board. The character looked for is a ‘?’ character while the only characters on the board are spaces. **Function Name:** test_isPlayerAtPos_emptyBoard

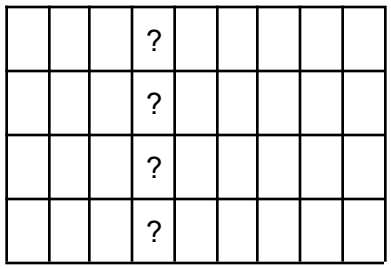
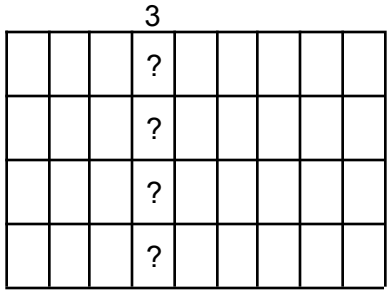
Input:	**Output:**	**Reason:**
State:  	isPlayerAtPos = true Expected character = 'K' Result character = 'K' State of board is unchanged	The test case is unique because it tests isPlayerAtPos on a board with some characters in it. The board position checked did indeed have the character desired so it returns true. **Function Name:** test_isPlayerAtPos_correctChar

Input:	**Output:**	**Reason:**
State:  	isPlayerAtPos = false Expected character = '!' Result character = '?' State of board is unchanged	The test case is unique because it tests isPlayerAtPos on a board with some characters in it. This time the method checks for an exclamation point but the character at the position is a question mark. In this case the method returns false. **Function Name:** test_isPlayerAtPos_wrongChar

Input:	**Output:**	**Reason:**
State:  	isPlayerAtPos = true Expected character = '?' Result character = '?' State of board is unchanged	The test case is unique because it tests isPlayerAtPos on a board that is full. The character that should be returned is a '?'. In this case it does find that character and returns true. **Function Name:** test_isPlayerAtPos_fullBoard

Input:	**Output:**	**Reason:**
State:  	isPlayerAtPos = true Expected character = '?' Result character = '?' State of board is unchanged	The test case is unique because it is testing isPlayerAtPos on a board that only contains one character. The rest of the spaces are empty. **Function Name:** test_isPlayerAtPos_onlyChar

Input:	Output:	Reason:
State:  P = 'X' C = 4 Board initialized to empty	 	The test case is unique because it tests placeToken on an empty board. A token is placed in the middle column. **Function Name:** test_placeToken_emptyBoard

Input:	Output:	Reason:
State:  P = '?' C = 3	 	The test case is unique because it tests placeToken on a column that is already full. A loop runs over 50 times trying to place tokens in column 3 but it is full. **Function Name:** test_placeToken_fullColumn

Input:

State:

0								8
X	X	X	X	?	X	X	X	X
?	?	?	?	X	?	?	?	?
X	X	X	X	?	X	X	X	X
?	?	?	?	X	?	?	?	?
X	X	X	X	?	X	X	X	X
?	?	?	?	X	?	?	?	?

P = ‘A’
C = 0-8

Output:

0								8
X	X	X	X	?	X	X	X	X
?	?	?	?	X	?	?	?	?
X	X	X	X	?	X	X	X	X
?	?	?	?	X	?	?	?	?
X	X	X	X	?	X	X	X	X
?	?	?	?	X	?	?	?	?

Reason:

The test case is unique because it tests placeToken on a full board. A loop runs over every column attempting to place another token, but all are full so a token is not placed in the board.

Function Name:
test_placeToken_fullBoard

Input:

State:

					5				
					J				
					!				
					?				
					J				
					!				
					?				

P = ‘A’
C = 5

Output:

					5				
					A				
					J				
					!				
					?				
					J				
					!				
					?				

Reason:

The test case is unique because it tests placeToken on a column that is one token away from being full. This test ensures that the rows for placing tokens are working properly.

Function Name:
test_placeToken_lastToken_inColumn

Input:

State:

P = ‘A’ and ‘B’
C = 0 and 8

Output:

B					B
A					A
B					B
A					A
B					B
A					A

Reason:

The test case is unique because it tests placeToken on each corner position. This ensures that the corner boundaries are working properly for placeToken.

Function Name:
test_placeToken_4corners