CPSC 2150 Project Report

Carson Tollison

**Requirements Analysis**

**Functional Requirements:**

1. As a player, I need be able to see the board so that I know which positions are available on the game board.
2. As a player, I need to know which marker I use so that I know who’s turn it is.
3. As a player, I need to be able to input my column number so that I can put my marker into that position.
4. As a player, I need to know how the columns are labeled so that I place my marker into the correct position.
5. As a player, I need to be able to choose if I want to play again or not so that I can quit or continue playing.
6. As a player, I can input my response again to play again if my input was invalid.
7. As a player, I can input my column again if I have invalid input so that I get my marker on the board.
8. As a player, I can input the number of columns I want to use on this game board.
9. As a player, I can input my number of columns again so that it is within the correct range.
10. As a player, I can input the number of rows I want to use on this game board.
11. As a player, I can input my number of rows again so that it is within the correct range.
12. As a player, I can input the number of tokens in a row it takes to win the game.
13. As a player, I can input my number of tokens in a row again so that it is within the correct range.
14. As a player, I need to know the game status so that I will know if it was a win, tie, or loss.
15. As a player, if I get the number of tokens in a row horizontally that I specified, I will win the game so that I can win the game
16. As a player, if I get the number of tokens in a row vertically that I specified, I will win the game so that I can win the game
17. As a player, if I get the number of tokens in a row diagonally that I specified, I will win the game so that I can win the game
18. As a player, I can pick again if I pick a column that does not exist, so I don't lose my turn
19. As a player, I can make a move after my opponent does (assuming they don't win), so I can always have my turn
20. As a player, I can end the game in a tie by taking the last space on the board without getting the number of tokens in a row that I specified, so the game can end
21. As a player, I can input the number of players that will play the game.
22. As a player, I can input the number of players again if the input was invalid.
23. As a player, I can input the token that each player will use.
24. As a player, I can input the token that a player would use again if I inputted a token that is taken.
25. As a player, I can choose if I want to play a fast game or a memory efficient game.
26. As a player, I can input my choice for a game type again if my input was invalid.

**Non-Functional Requirements**

1. The system must be programmed with java.
2. The system must run on Unix.
3. The system should be reliable.
4. The system should be fast if that game mode is chosen.
5. The system should be memory efficient if that game mode is chosen.
6. The board can be any size that the user specifies if it falls within range.

**Deployment Instructions**default: compiles code. Runs with the *make* command.   
run: runs code. Runs with the *make run* command.   
clean: removes compiled (.class) files. Runs with the *make clean* command.

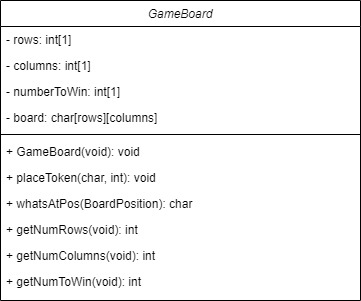
testGB: runs junit tests for the GameBoard class

testGBmem: runs junit tests for the GameBoardmem class

**System Design**

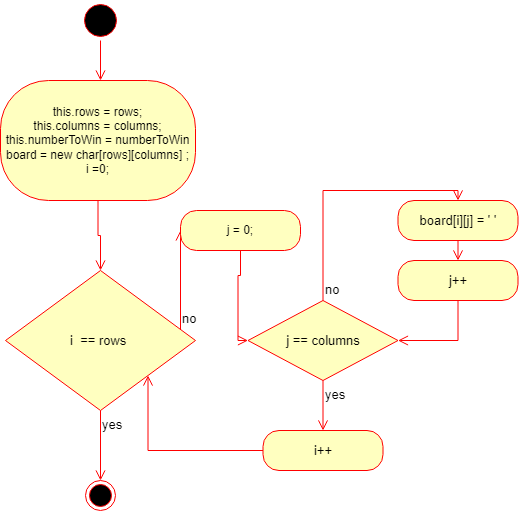
**Class 1:** GameBoard.java

**Class diagram**

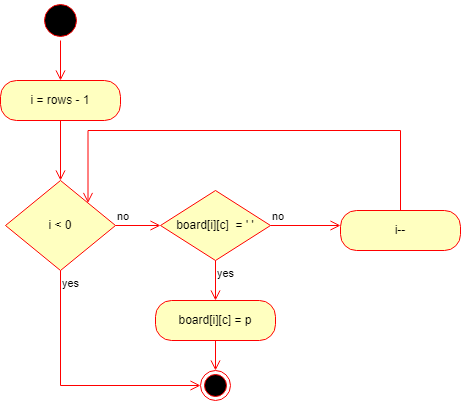
****

**Activity diagrams**

**GameBoard**

****

**placeToken**

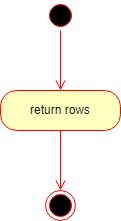
****

**whatsAtPos**

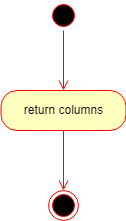
**Diagram

Description automatically generated**

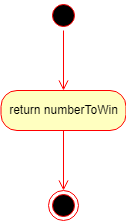
**getNumRows**

****

**getNumColumns**

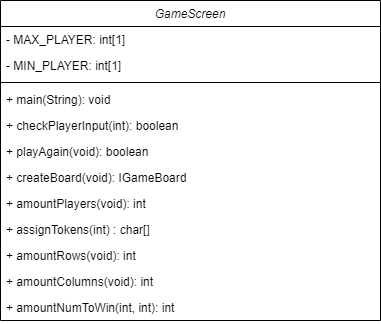
****

**getNumToWin**

****

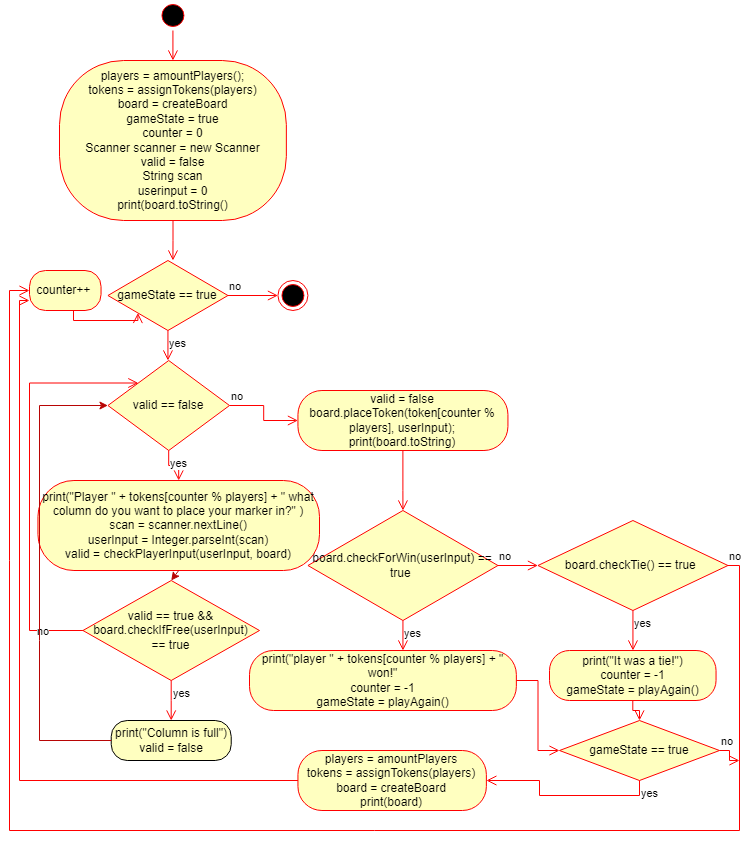
**Class 2:** GameScreen.java

**Class diagram**

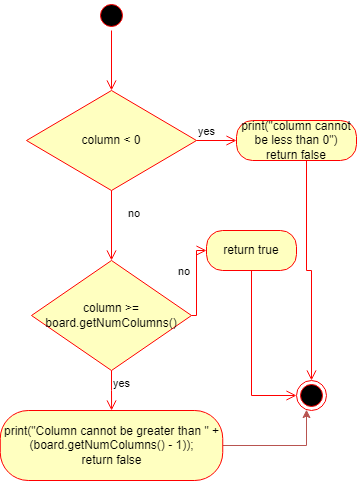
****

**Activity diagrams**

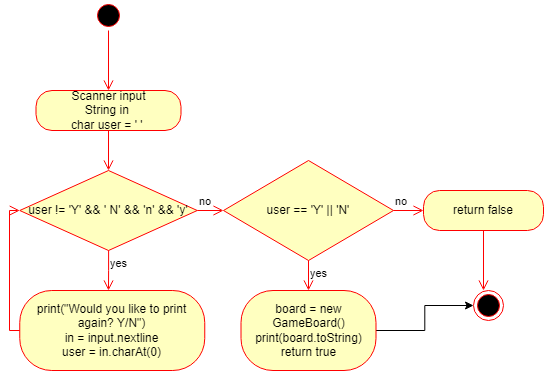
**main**

****

**checkPlayerInput**

****

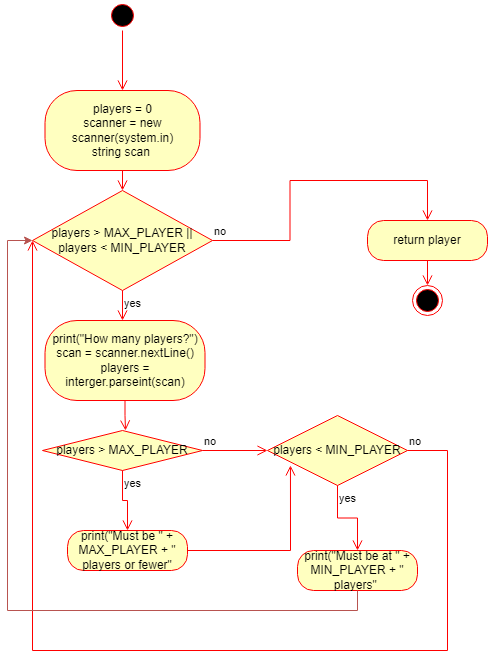
**playAgain**

****

**createBoardDiagram

Description automatically generated**

**amountPlayers**

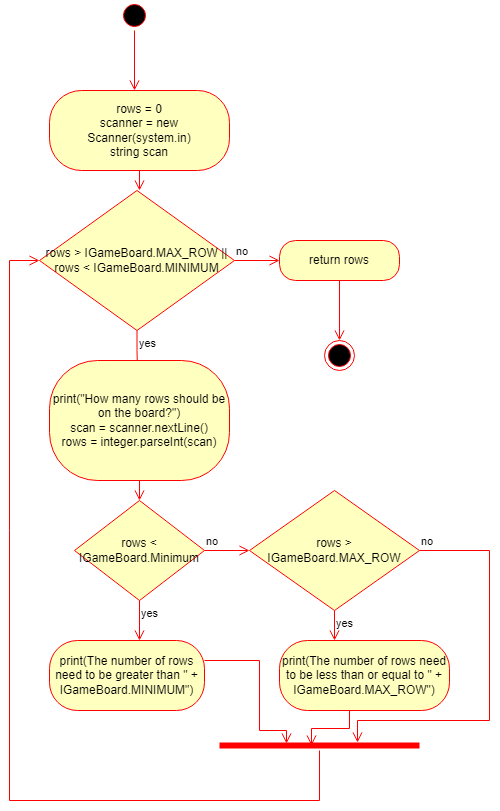
****

**assignTokens**

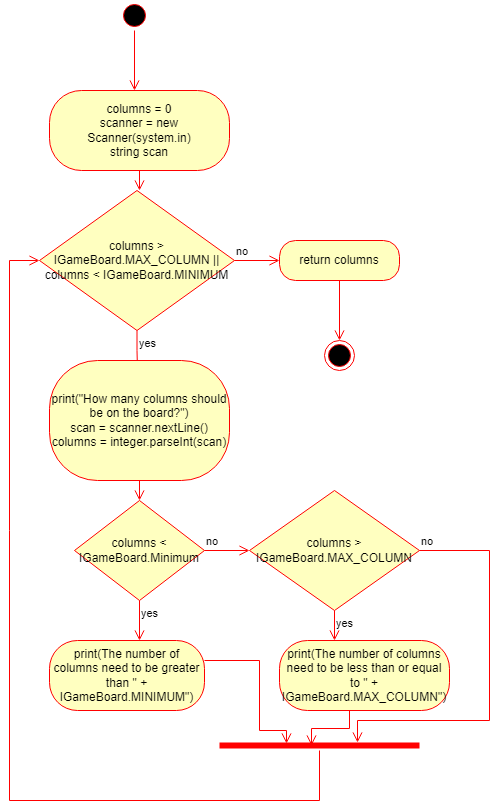
**Diagram

Description automatically generated**

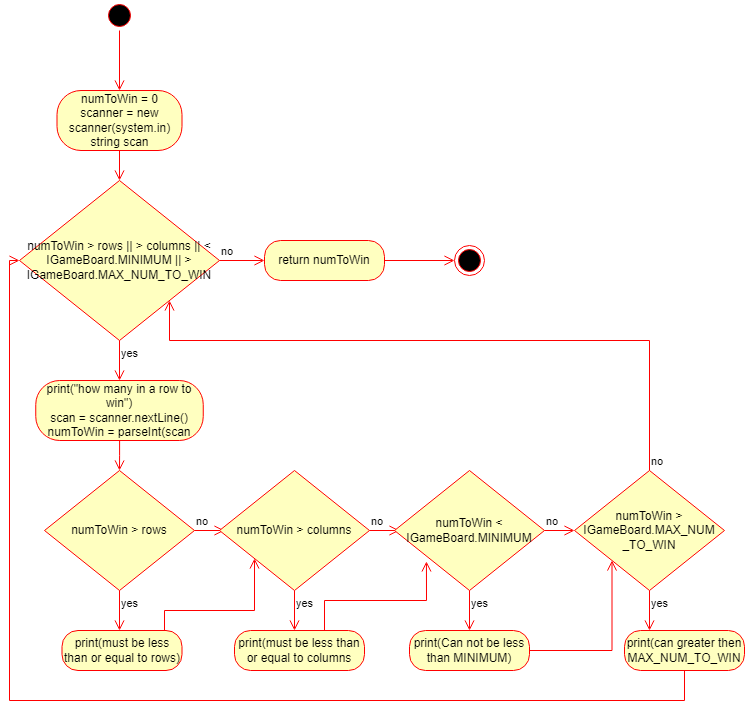
**amountRows**

****

**amountColumns**

****

**amountNumToWin**

****

**Class 3:** BoardPosition.java

**Class diagram**

**Table

Description automatically generated**

**Activity diagrams**

**BoardPosition**

**Diagram

Description automatically generated**

**getRow**

**Diagram

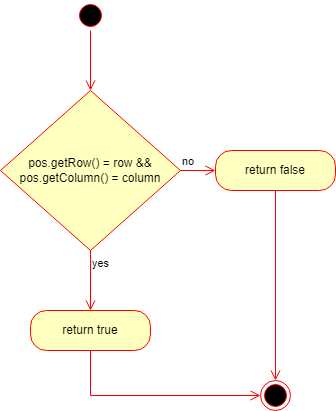
Description automatically generated**

**getColumn**

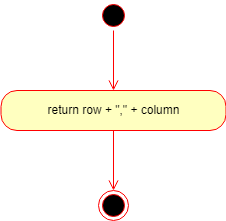
**Diagram

Description automatically generated**

**equals**

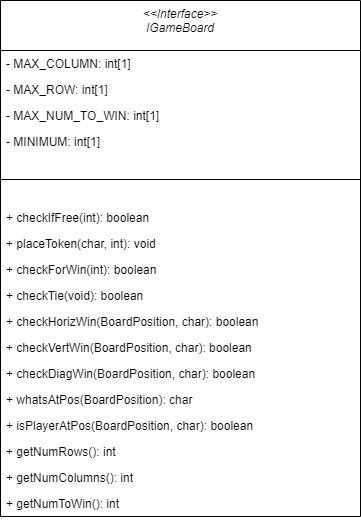
****

**toString**

****

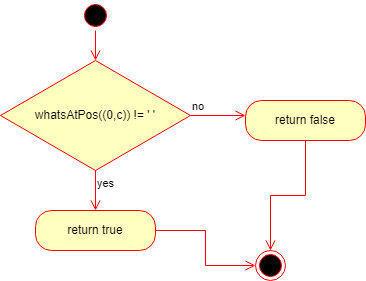
**Class 4:** IGameBoard.java

**Class diagram**

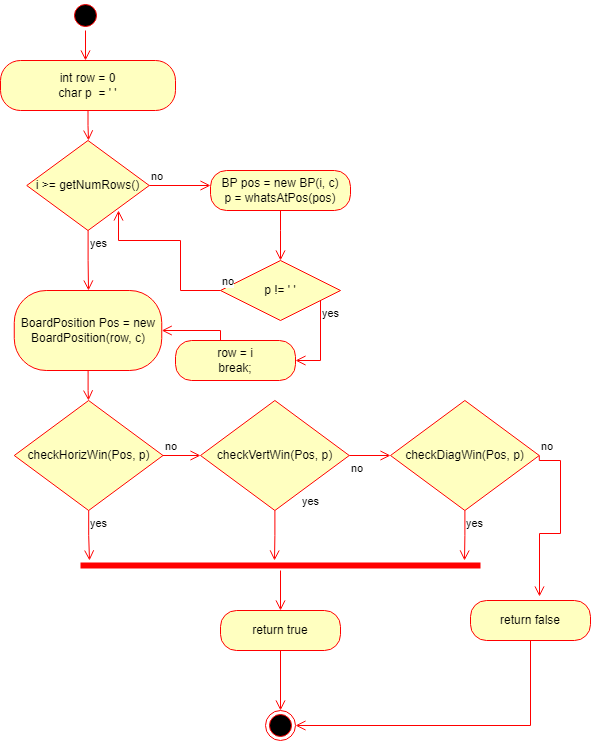
****

**Activity diagrams**

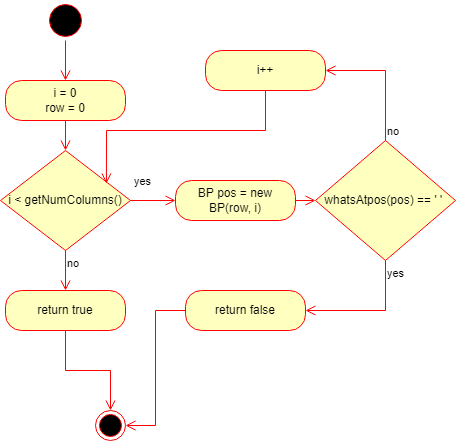
**checkIfFree**

****

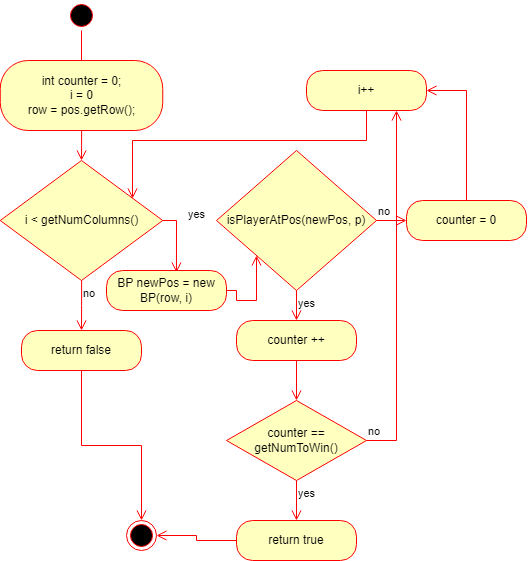
**checkForWin**

****

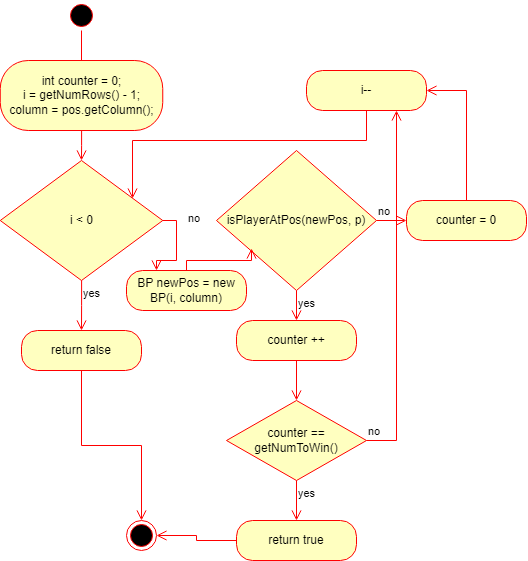
**checkTie**

****

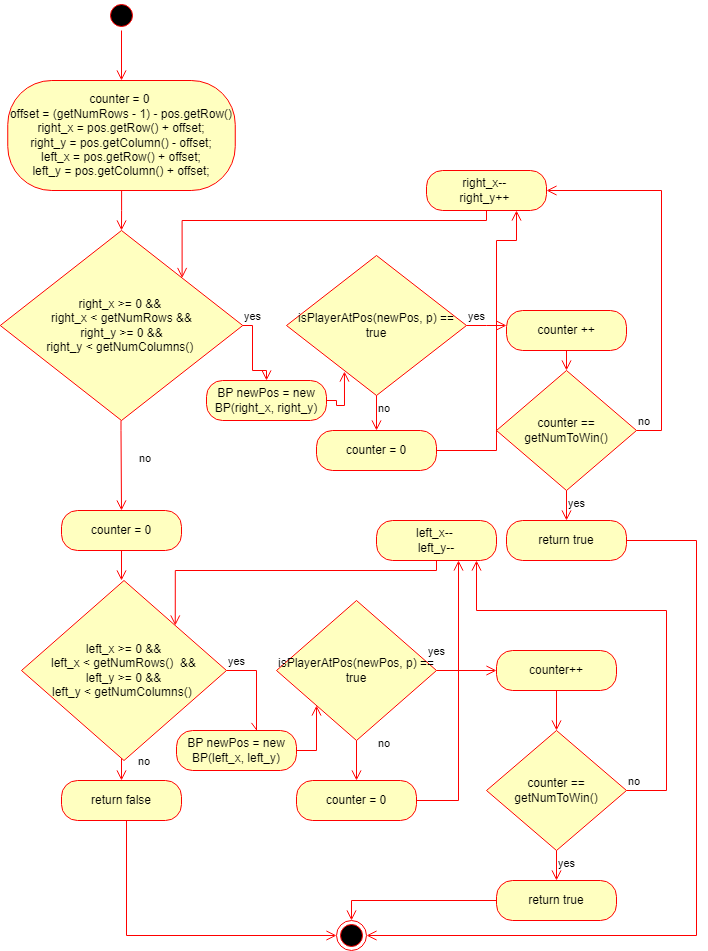
**checkHorizWin**

****

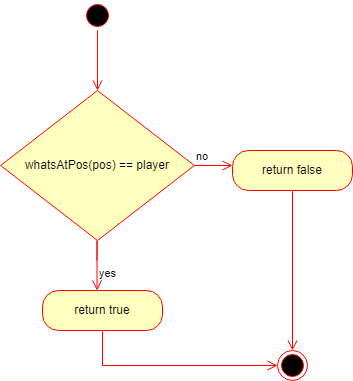
**checkVertWin**

****

**checkDiagWin**

****

**isPlayerAtPos**

****

**Class 5:** AbsGameBoard.java

**Class diagram**

**Table

Description automatically generated**

**Activity diagrams**

**toString**

**Diagram

Description automatically generated**

**Class 6:** GameBoardMem.java

**Class diagram**

**Table

Description automatically generated with medium confidence**

**Activity diagrams**

**GameBoardMem**

**Diagram

Description automatically generated**

**placeToken**

**Diagram

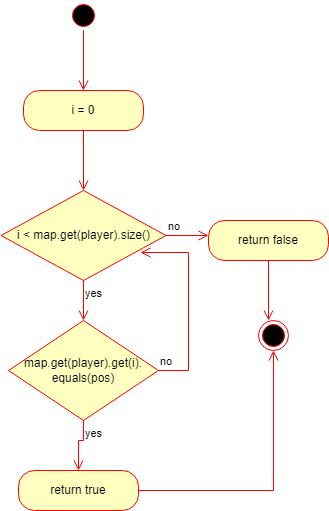
Description automatically generated**

**whatsAtPos**

**Diagram

Description automatically generated**

**isPlayerAtPos**

****

**getNumRows**

**Diagram

Description automatically generated**

**getNumColumns**

**Diagram

Description automatically generated**

**getNumToWin**

**Diagram

Description automatically generated**

**Test Cases**

GameBoard(int row, int column, int numberToWin)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:  Board is not Initialized  Row = 5  Column = 5  Number to Win = 3 | Output:  Board is created  Board.row = 5  Board.column = 5  Board.numToWin = 3  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Reason:  This test case is unique and distinct because the board is initialized to a certain size. So, the functions need to initialize the board to the size specified.  Function Name:  testConstructor\_row5\_  column5\_numberToWin3 |

|  |  |  |
| --- | --- | --- |
| Input:  State:  Board is not Initialized  Row = 100  Column = 100  Number to Win = 25 | Output:  Board is created  Board.row = 100  Board.column = 100  Board.numToWin = 25  State:  (This board is 100 by 100 so I won’t be including the diagram so it fits the screen) | Reason:  This test case is unique and distinct because the board is initialized the maximum size it that can be created to. Also, number to win is set to its maximum value.  Function Name:  testConstructor\_rowMAX\_  columnMAX\_numberToWinMAX |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:  Board is not Initialized  Row = 3  Column = 3  Number to Win = 3 | Output:  Board is created  Board.row = 3  Board.column = 3  Board.numToWin = 3  State:   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Reason:  This test case is unique and distinct because the board is initialized the minimum size it that can be created to. Also, number to win is set to its minimum value.  Function Name:  testConstructor\_rowMIN\_  columnMIN\_numberToWinMIN |

boolean checkIfFree (int c)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | X |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  |   C = 0 | Output:  checkIfFree = true  state of the board is unchanged | Reason:  This test case is unique and distinct because the column we are trying to see if free is full so it should return that it is not free  Function Name:  testCheckIfFree\_Column\_Full |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   C = 0 | Output:  checkIfFree = false  state of the board is unchanged | Reason:  This test case is unique and distinct because the column we are trying to see if free is completely empty so it should return that it is free  Function Name:  testCheckIfFree\_  Column\_Empty |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | O |  |  |  |  |   C = 0 | Output:  checkIfFree = false  state of the board is unchanged | Reason:  This test case is unique and distinct because the column we are trying to see if free is has tokens in it but is not full so it should return that it is free  Function Name:  testCheckIfFree\_  Column\_Partial\_Filled |

boolean checkHorizWin ( BoardPosition pos, char p)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   Pos.getRow = 0  Pos.getColumn = 0  P = ‘X’ | Output:  checkHorizWin = false  state of the board is unchanged | Reason:  This test case is unique and distinct because the board is empty and ‘X’ does not exist on the board, so the function will not find a win on the board  Function Name:  testHorizWin\_no\_Win\_  Condition\_empty\_Board |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | X | X | X |  |  |   Pos.getRow = 0  Pos.getColumn = 2  P = ‘X’ | Output:  checkHorizWin = true  state of the board is unchanged | Reason:  This test case is unique and distinct because when the token is placed at the right end of the row of X’s it needs to check across the board on the left for side for a win  Function Name:  testHorizWin\_bottom\_Row  \_Win\_right\_Marker\_End |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | X | X | X |  |  | | X | X | O | X | X | | X | X | O | X | X |   Pos.getRow = 2  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 token is placed into the middle and the function needs to check the left and right of it for a win  Function Name:  testHorizWin\_last  \_Marker\_Middle |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | O | O | X |  |  |   Pos.getRow = 0  Pos.getColumn = 2  P = ‘X’ | Output:  checkHorizWin = false  state of the board is unchanged | Reason:  This function is unique because the last token placed created enough tokens in a row to win. However, they are not the same tokens so it should return false.  Function Name:  testHorizWin\_last\_  Marker\_Not\_Win |

boolean checkVertWin (BoardPosition pos, char p)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   Pos.getRow = 0  Pos.getColumn = 0  P = ‘X’ | Output:  checkVertWin = false  state of the board is unchanged | Reason:  This test case is unique and distinct because the board is empty and ‘X’ does not exist on the board, so the function will not find a win on the board  Function Name:  testVertWin\_no\_Win  \_Condition\_empty\_Board |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  |   Pos.getRow = 2  Pos.getColumn = 0  P = ‘X’ | Output:  checkVertWin = true  state of the board is unchanged | Reason:  This test is unique because the last marker placed creates a win condition in the first column. The function needs to count from the bottom to check.    Function Name:  testVertWin\_win  \_last\_marker\_placed |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | O |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  |   Pos.getRow = 2  Pos.getColumn = 0  P = ‘O’ | Output:  checkVertWin = false  state of the board is unchanged | Reason:  This test is unique because an O token is placed on top of two X tokens which is enough for the win. The function needs to decipher that these are separate tokens and return not a win.  Function Name:  testVertWin\_not\_win  \_last\_marker\_placed |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | | O |  |  |  |  | | O |  |  |  |  | | O |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  |   Pos.getRow = 4  Pos.getColumn = 0  P = ‘O’ | Output:  checkVertWin = true  state of the board is unchanged | Reason:  This test is unique because an O token is placed on the top of the column that has a win. It needs to skip the other character in this check and return a win for O.  Function Name:  testVertWin\_win\_last\_  placed\_top\_column |

boolean checkDiagWin (BoardPosition pos, char p)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   Pos.getRow = 0  Pos.getColumn = 0  P = ‘X’ | Output:  checkDiagWin = false  state of the board is unchanged | Reason:  This test case is unique and distinct because the board is empty and ‘X’ does not exist on the board, so the function will not find a win on the board  Function Name:  testDiagWin\_no\_Win\_  Condition\_empty\_Board |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  | |  | X | O |  |  | | X | O | O |  |  |   Pos.getRow = 2  Pos.getColumn = 2  P = ‘X’ | Output:  checkDiagWin = true  state of the board is unchanged | Reason:  This test case is unique because it needs to read the tokens from left to right after the token was placed on the right end.  Function Name:  testDiagWin\_win\_  condition\_met\_ltoR\_right |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | X |  |  |  |  | | O | X |  |  |  | | O | O | X |  |  |   Pos.getRow = 0  Pos.getColumn = 2  P = ‘X’ | Output:  checkDiagWin = true  state of the board is unchanged | Reason:  This test is unique because it needs to read the tokens from right to left after the token was placed on the right end  Function Name:  testDiagWin\_win\_  condition\_met\_RtoL\_right |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  | |  | X | O |  |  | | X | O | O |  |  |   Pos.getRow = 0  Pos.getColumn = 0  P = ‘X’ | Output:  checkDiagWin = true  state of the board is unchanged | Reason:  This test case is unique because it needs to read the tokens from left to right after the token was placed on the left end.  Function Name:  testDiagWin\_win\_  condition\_met\_ltoR\_left |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | X |  |  |  |  | | O | X |  |  |  | | O | O | X |  |  |   Pos.getRow = 2  Pos.getColumn = 0  P = ‘X’ | Output:  checkDiagWin = true  state of the board is unchanged | Reason:  This test is unique because it needs to read the tokens from right to left after the token was placed on the left end  Function Name:  testDiagWin\_win\_  condition\_met\_RtoL\_left |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  | |  | X | O |  |  | | X | O | O |  |  |   Pos.getRow = 1  Pos.getColumn = 1  P = ‘X’ | Output:  checkDiagWin = true  state of the board is unchanged | Reason:  This test is unique because it needs to read the tokens from the left to right side after the token was placed in the middle to create a win  Function Name:  testDiagWin\_win\_  condition\_met\_ltoR\_middle |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | X |  |  |  |  | | O | X |  |  |  | | O | O | X |  |  |   Pos.getRow = 1  Pos.getColumn = 1  P = ‘X’ | Output:  checkDiagWin = true  state of the board is unchanged | Reason:  This test is unique because it needs to read the tokens from the right to left side after the token was placed in the middle to create a win  Function Name:  testDiagWin\_win\_  condition\_met\_RtoL\_middle |

boolean checkTie (void)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 3)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  checkTie = false  State of board is unchanged | Reason:  This test is unique because it checks for a tie on an empty board where there are no characters so it should return false  Function Name:  testcheckTie\_empty\_Board |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 4)   |  |  |  |  |  | | --- | --- | --- | --- | --- | | O | X | O | X | O | | O | X | O | X | O | | X | O | X | O | X | | O | X | O | X | O | | O | X | O | X | O | | Output:  checkTie = true  state of board is unchanged | Reason:  This test is unique because the board is full and has no win conditions, so it needs to detect there is a tie  Function Name:  testcheckTie\_full\_board\_tie |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 4)   |  |  |  |  |  | | --- | --- | --- | --- | --- | | O | X | O | X |  | | O | X | O | X | O | | X | O | X | O | X | | O | X | O | X | O | | O | X | O | X | O | | Output:  checkTie = false  state of board is unchanged | Reason:  This test is unique because the board is almost full but has one spot left on the end, so it needs to detect that open spot  Function Name:  testcheckTie\_space\_free\_no\_tie |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State: (num to win = 4)   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | X | O | X | O | X | | O | X | O | X | O | | O | X | O | X | O | | Output:  checkTie = false  state of board is unchanged | Reason:  This test is unique because the board has tokens in it but there is still a lot of open places without a win  Function Name:  testcheckTie\_half\_board\_no\_tie |

char whatsAtPos (BoardPosition pos)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   Pos.getRow = 0  Pos.getColumn = 0 | Output:  whatsAtPos = ‘ ‘  The state of the board is unchanged | Reason:  This test checks to see what is at a certain position on an empty board. It should be ‘ ‘ since no characters are placed on the board  Function Name:  testwhatsAtPos\_empty\_Board |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  | O |  |  | |  |  | O |  |  | |  |  | O |  |  | |  |  | O |  |  | |  |  | X |  |  |   Pos.getRow = 0  Pos.getColumn = 2 | Output:  whatsAtPos = ‘ ‘  The state of the board is unchanged | Reason:  This test is unique because it checks to make sure that the token was not replaced as tokens were placed in the column  Function Name:  testwhatsAtPos\_first\_in\_column |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | X |  |  |  |  |   Pos.getRow = 0  Pos.getColumn = 0 | Output:  whatsAtPos = ‘X’  The state of the board is unchanged | Reason:  This test is unique because it checks to make sure that the function can return the correct token placed on the left side  Function Name:  testwhatsAtPos\_left\_Place |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  |   Pos.getRow = 0  Pos.getColumn = 2 | Output:  whatsAtPos = ‘X’  The state of the board is unchanged | Reason:  This test is unique because it checks to make sure that the function can return the correct token placed in the middle  Function Name:  testwhatsAtPos\_middle\_place |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  | X |   Pos.getRow = 0  Pos.getColumn = column - 1 | Output:  whatsAtPos = ‘X’  The state of the board is unchanged | Reason:  This test is unique because it checks to make sure that the function can return the correct token placed on the right side  Function Name:  testwhatsAtPos\_right\_place |

Boolean isPlayerAtPos(BoardPosition pos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   Pos.getRow = 0  Pos.getColumn = 0  Player = ‘ ’ | Output:  isPlayerAtPos = true  State of board is unchanged | Reason:  This test checks to see what is at a certain position on an empty board. It should be ‘ ‘ since no characters are placed on the board  Function Name:  testIsPlayerAtPos\_emptyBoard |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  | O |  |  | |  |  | O |  |  | |  |  | O |  |  | |  |  | O |  |  | |  |  | X |  |  |   Pos.getRow = 0  Pos.getColumn = 2  Player = ‘X’ | Output:  isPlayerAtPos = true  State of board is unchanged | Reason:  This test is unique because it checks to make sure that the token at the bottom position has not been replaced  Function Name:  testIsPlayerAtPos\_first\_  in\_column |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | X |  |  |  |  |   Pos.getRow = 0  Pos.getColumn = 0  Player = ‘X’ | Output:  isPlayerAtPos = true  State of board is unchanged | Reason:  This test is unique because it checks to make sure that the function can find the character on the left side  Function Name:  testIsPlayerAtPos\_left\_Place |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  |   Pos.getRow = 0  Pos.getColumn = 2  Player = ‘X’ | Output:  isPlayerAtPos = true  State of board is unchanged | Reason:  This test is unique because it checks to make sure that the function can find the character in the middle  Function Name:  testIsPlayerAtPos\_middle\_place |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  | X |   Pos.getRow = 0  Pos.getColumn = column - 1  Player = ‘X’ | Output:  isPlayerAtPos = true  State of board is unchanged | Reason:  This test is unique because it checks to make sure that the function can find the character on the right side  Function Name:  testIsPlayerAtPos\_right\_place |

void placeToken (char p, int c)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   P = ‘X’  C= 0 | Output:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | X |  |  |  |  | | Reason:  This test case is unique because it is placing a token in an empty board and is on the left side of the board  Function Name:  testplaceToken\_  left\_corner\_place |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | X | O | X | O |  | | O | X | O | X | O | | O | X | O | X | O |   P = ‘X’  C= column - 1 | Output:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | X | O | X | O | X | | O | X | O | X | O | | O | X | O | X | O | | Reason:  This test case is unique because it is placing a token in column that is not empty. This column is over half full.  Function Name:  testplaceToken\_board\_filled |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   P = ‘X’  C= column - 1 | Output:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  | X | | Reason:  This test case is unique because it is placing a token in an empty board and is on the right side of the board  Function Name:  testplaceToken\_  right\_corner\_place |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  |   P = ‘X’  C= 0 | Output:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | X |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  | | X |  |  |  |  | | Reason:  This test is unique because it is placing a token into a column that Is about to be full after it is placed in the top row  Function Name:  testplaceToken\_top\_column |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  |   P = ‘X’  C= 2 | Output:  State:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  | |  |  | X |  |  | | Reason:  This test is unique because it is placing a token into a column in the middle of the board. This column is also not empty.  Function Name:  testplaceToken\_middle\_column |