**REQUIREMENT ANALYSIS**

Functional Requirements

1. As a player, I can enter my position so that I can make my move on the game board
2. As a player, I can view the game board, so I know the state of the game
3. As a player, I have the option to play the game again so I can play the game again
4. As a player, I need to know if the position I entered is available so that I can play that position or enter in a new one
5. As a player, I need to know if there’s a winner or draw so I can end the game or choose to play again
6. As a player, I get to view who won the game, so I know who won the game
7. As a player, I get to view who’s turn it is, so I don’t go when it is not my turn
8. As a player, I am informed to first enter the row and then the column, so I don’t mistakenly enter in the wrong position
9. As a player, I can pick again if I enter in an invalid (out of bounds) position so that I can play a valid position
10. As a player, I can win horizontally (five in a row horizontally) so I can win the game
11. As a player, I can win vertically so I can win the game
12. As a player, I can win diagonally so I can win the game
13. As a player, I play a turn and rotate turns with the other player, so the game is fair
14. As a player, I can pick the number of players that will play the game so the amount of people that will be playing the game can play the game
15. As a player, I can pick the number of rows for the game board so I can create my desired game board
16. As a player, I am informed If I have picked the number of rows that is too big or too small, so I can pick again
17. As a player, I can pick the number of columns for the game board so I can create my desired game board
18. As a player, I am informed if I have picked the number of columns that is too big or too small, so I can pick again
19. As a player, I can pick the number of tokens in a row to win so I know the number of tokens in a row to win
20. As a player, I am informed if the number of tokens in a row I have entered is too big or too small, so I can pick again
21. As a player, If I choose to play the game again, I am allowed to pick the number of rows, number of columns, number of tokens in a row to win, and type of game board again so I can design my game board however I want when I choose to play again

Non-Functional Requirements

1. The code runs on Unix (school of computing computers)
2. Use Java as the coding language
3. Gameboard at least will have 3 rows and 3 columns
4. Gameboard at most will have 20 rows and 20 columns
5. The number in a row to win will be at least 3 and at most 20
6. The number in a row to win cannot be greater than the number of rows and the number of columns of the game board
7. The number of players is at least 2 and at most 10
8. Players are capitalized characters
9. Player 1 always goes first
10. 0,0 is the top left of the board
11. Code is efficient
12. Cannot have players with same character
13. Code followers MVC design and uses observer pattern
14. Must be a GUI program

**DESIGN**

GameScreen Class

Text

Description automatically generated

Main

Diagram

Description automatically generated

getNumRowsFromUser

Diagram

Description automatically generated

getNumColumnsFromUser

Diagram

Description automatically generated

getNumToWinFromUser

Diagram

Description automatically generated

getGameBoardInfo

Diagram

Description automatically generated

createGameBoard

Diagram

Description automatically generated

getNumPlayerfromUser

Diagram

Description automatically generated

storePlayers

Diagram

Description automatically generated

getPlayerPos

Diagram

Description automatically generated

userInputPlayAgain

Diagram

Description automatically generated

playGame

Diagram

Description automatically generated

IGameBoard Interface

**Text

Description automatically generated**

**checkSpace**

**Diagram

Description automatically generated**

**checkForWinner**

**Diagram

Description automatically generated**

**checkForDraw**

**Diagram

Description automatically generated**

**checkHorizontalWin**

**Diagram

Description automatically generated**

**checkVerticalWin**

**Diagram

Description automatically generated**

**checkDiagonalWin**

**Diagram

Description automatically generated**

**isPlayerAtPos**

**Diagram

Description automatically generated**

GameBoard Class

**Diagram

Description automatically generated**

**getNumRows**

**Diagram

Description automatically generated**

**getNumColumns**

**Diagram

Description automatically generated**

**getNumToWin**

**Diagram

Description automatically generated**

**placeMarker**

**Diagram

Description automatically generated**

**whatsAtPos**

**A picture containing timeline

Description automatically generated**

**GameBoard (constructor)**

**Diagram

Description automatically generated**

**checkForDraw**

**Diagram

Description automatically generated**

GameBoardMem Class

**Diagram

Description automatically generated**

**GameBoardMem (constructor)**

**Diagram

Description automatically generated**

**getNumRows**

**Diagram

Description automatically generated**

**getNumColumns**

**Diagram

Description automatically generated**

**getNumToWin**

**Diagram

Description automatically generated**

**isPlayerAtPos**

**Diagram

Description automatically generated**

**placeMarker**

**Diagram

Description automatically generated**

**whatsAtPos**

**Diagram

Description automatically generated**

**checkForDraw**

**Diagram

Description automatically generated**

BoardPosition Class

**Table

Description automatically generated**

**BoardPosition (constructor)**

**Diagram

Description automatically generated**

**getRow**

**Diagram, schematic

Description automatically generated**

**getColumn**

**Diagram, schematic

Description automatically generated**

**Equals**

**Diagram

Description automatically generated**

**toString**

**Diagram

Description automatically generated**

AbsGameBoard Class

**Diagram

Description automatically generated**

**toString**

**Diagram

Description automatically generated**

TicTacToeController Class

Text

Description automatically generated

**processButtonClick**

Diagram

Description automatically generated

**TESTING**

constructor(int numRows, int numColumns, int numToWin)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  numRows = 5  numColumns = 8  numToWin = 5 | **Output:**  State:  numRows = 5  numColumns = 8  numToWin = 5   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 0 |  |  |  |  |  |  |  |  | | 1 |  |  |  |  |  |  |  |  | | 2 |  |  |  |  |  |  |  |  | | 3 |  |  |  |  |  |  |  |  | | 4 |  |  |  |  |  |  |  |  | | **Reason:**  This is a routine test case to see if the constructor can generate an empty 5x8 grid  Function Name:  testConstructor\_routine |
| Input:  numRows = 3  numColumns = 3  numToWin = 3 | Output:  State:  numRows = 3  numColumns = 3  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  | | Reason:  This test case is unique and distinct because it tests to see if the constructor can generate the right grid with number of rows, number of columns, and numToWin all set to the minimum  Function Name:  testConstructor\_boundary\_min |
| Input:  numRows = 100  numColumns = 100  numToWin = 25 | Output:  State:  numRows = 100  numColumns = 100  numToWin = 25  game board = [100 x 100 grid of empty spaces]   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | … | 100 | | 0 |  |  |  |  | | 1 |  |  |  |  | | … |  |  |  |  | | 100 |  |  |  |  | | Reason:  This test case is unique and distinct because it tests to see if the constructor can generate the right grid with number of rows, number of columns, and numToWin all set to the maximum  Function Name:  testConstructor\_boundary\_max |

boolean checkSpace(BoardPosition pos)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  | X |  | | 1 | O |  |  | | 2 |  |  |  |   pos.getRow = 1  pos.getColumn = 1 | Output:  checkSpace = true  state of the board is unchanged | Reason:  This is a routine test case. Test on a valid (not out of bounds) and the space is not taken (blank space character).  Function Name:  testCheckSpace\_  validBounds\_spaceNotTaken |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  |   pos.getRow = 3  pos.getColumn = 3 | Output:  checkSpace = false  state of the board is unchanged | Reason:  This test case is unique and distinct because it tests for out of bounds when the row and column are greater than the numRows and numColumns of the grid  Function Name:  testCheckSpace\_outOfBounds |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  | X |  | | 2 |  |  |  |   pos.getRow = 1  pos.getColumn = 1 | Output:  checkSpace = false  state of the board is unchanged | Reason:  This test case is unique and distinct because I am testing for valid bounds, but space is already taken  Function Name:  testCheckSpace\_validBounds\_  spaceTaken |

boolean checkHorizontalWin(BoardPosition lastPos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 | X | X | X | X |  | | 3 | O | O | O | X | O | | 4 |  |  |  |  |  |   lastPos.getRow = 2  lastPos.getColumn = 2  player = ‘X’ | Output:  checkHorizontalWin = 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 as opposed to on the end, so the function needs to count x’s on the right and left  Function Name:  testCheckHorizontalWin\_  win\_lastMarkerMiddle |
| Input:  State:  numToWin = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 |  | O | O |  | | 1 | X | X | X |  | | 2 |  |  |  |  | | 3 |  |  |  |  |   lastPos.getRow = 1  lastPos.getColumn = 2  player = ‘X’ | Output:  checkHorizontalWin = true  state of the board is unchanged | Reason:  This test case is unique and distinct because the last X was placed at the right most end, so the function needs to count x’s just to the left of it  Function Name:  testCheckHorizontalWin\_  win\_lastMarkerRightEnd |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  | X |  | | 2 |  |  |  |   lastPos.getRow = 1  lastPos.getColumn = 1  player = ‘X’ | Output:  checkHorizontalWin = false  state of the board is unchanged | Reason:  This test case is unique and distinct because I am testing to see if the function will output false after not seeing any consecutive X’s to the left or right  Function Name:  testCheckHorizontalWin\_noWin  \_noLeft\_noRight |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 | O | X | X | X | O | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 2  lastPos.getColumn = 1  player = ‘X’ | Ouput:  checkHorizontalWin = false  state of the board is unchanged | Reason:  This test case is unique and distinct because I am testing to see if the function counts X’s just to the right of it and the count would not be enough to result in a horizontal win  Function Name:  testCheckHorizontalWin\_noWin  \_noLeft\_rightNotEnough |

boolean checkVerticalWin(BoardPosition lastPos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | X | O |  | | 1 |  |  | X | O |  | | 2 |  |  | X | O |  | | 3 |  |  | X | X |  | | 4 |  |  |  | O |  |   lastPos.getRow = 2  lastPos.getColumn = 2  player = ‘X’ | Output:  checkVerticalWin = 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 as opposed to on the end, so the function needs to count x’s up and down  Function Name:  testCheckVerticalWin\_  win\_lastMarkerMiddle |
| Input:  State:  numToWin = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 |  | X |  |  | | 1 |  | X | O |  | | 2 |  | X | O |  | | 3 |  |  |  |  |   lastPos.getRow = 2  lastPos.getColumn = 1  player = ‘X’ | Output:  checkVerticalWin = true  state of the board is unchanged | Reason:  This test case is unique and distinct because the last X was placed at the bottom most end, so the function needs to count x’s just to the top of it  Function Name:  testCheckVerticalWin\_  win\_lastMarkerBottomEnd |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  | X |  | | 2 |  |  |  |   lastPos.getRow = 1  lastPos.getColumn = 1  player = ‘X’ | Output:  checkVerticallWin = false  state of the board is unchanged | Reason:  This test case is unique and distinct because I am testing to see if the function will output false after not seeing any consecutive X’s up or down  Function Name:  testCheckVerticalWin\_noWin  \_noUp\_noDown |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | 0 |  |  | | 1 |  |  | X |  |  | | 2 |  |  | X |  |  | | 3 |  |  | X |  |  | | 4 |  |  | O |  |  |   lastPos.getRow = 1  lastPos.getColumn = 2  player = ‘X’ | Ouput:  checkVerticalWin = false  state of the board is unchanged | Reason:  This test case is unique and distinct because I am testing to see if the function counts X’s just below it and the count would not be enough to result in a vertical win  Function Name:  testCheckVerticalWin\_noWin  \_noUp\_downNotEnough |

boolean checkDiagonalWin(BoardPosition lastPos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:    State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | X |  |  | O | O | | 1 |  | X |  |  | O | | 2 |  |  | X |  |  | | 3 |  |  |  | X |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getColumn = 0  player = ‘X’ | Output:  checkDiagonalWin = true  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing backslash diagonal with row and column increasing (going down) where my last position placed is at the top left end so I only check in one direction (row and column increasing).  Function Name:  testCheckDiagonalWin\_win\_  backslash\_  lastMarkerTopLeftEnd |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | X |  |  | O | O | | 1 |  | X |  |  | O | | 2 |  |  | X |  |  | | 3 |  |  |  | X |  | | 4 |  |  |  |  |  |   lastPos.getRow = 3  lastPos.getColumn = 3  player = ‘X’ | Output:  checkDiagonalWin = true  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing backslash diagonal. My last position placed is at the bottom right end so I will only be checking in one direction (row and column decreasing)  Function Name:  testCheckDiagonalWin\_win\_  backslash\_lastMarker  BottomLeftEnd |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | X |  |  | O | O | | 1 |  | X |  |  | O | | 2 |  |  | X |  |  | | 3 |  |  |  | X |  | | 4 |  |  |  |  |  |   lastPos.getRow = 1  lastPos.getColumn = 1  player = ‘X’ | Output:  checkDiagonalWin = true  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing backslash diagonal where I will be testing for both directions. My last position placed is in the middle so I will have to count X’s in both direction  Function Name:  testCheckDiagonalWin\_win\_  backslash\_lastMarkerMiddle |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O | O |  |  |  | | 1 | O |  |  | X |  | | 2 |  |  | X |  |  | | 3 |  | X |  |  |  | | 4 | X |  |  |  |  |   lastPos.getRow = 4  lastPos.getColumn = 0  player = ‘X’ | Output:  checkDiagonalWin = true  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing forward slash diagonal where my last position placed is at the bottom left end, and I am only checking in one direction where my row is decreasing and my column is increasing (going towards top-right)  Function Name:  testCheckDiagonalWin\_win\_  forwardSlash\_  lastMarkerBottomLeftEnd |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O | O |  |  |  | | 1 | O |  |  | X |  | | 2 |  |  | X |  |  | | 3 |  | X |  |  |  | | 4 | X |  |  |  |  |   lastPos.getRow = 1  lastPos.getColumn = 3  player = ‘X’ | Output:  checkDiagonalWin = true  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing forward slash diagonal where my last position placed is at the top right end, and I am only checking in one direction where my row is increasing and my column is decreasing (going towards bottom-left)  Function Name:  testCheckDiagonalWin\_win\_  forwardSlash\_  lastMarkerTopRightEnd |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O | O |  |  |  | | 1 | O |  |  | X |  | | 2 |  |  | X |  |  | | 3 |  | X |  |  |  | | 4 | X |  |  |  |  |   lastPos.getRow = 2  lastPos.getColumn = 2  player = ‘X’ | Output:  checkDiagonalWin = true  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing forward slash diagonal where I will be testing for both directions. My last position placed is in the middle so I will have to count X’s in both direction  Function Name:  testCheckDiagonalWin\_win\_  forwardSlash\_lastMarkerMiddle |
| Input:  State:  numToWin = 4   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | X |  |  |  | X | | 1 |  | X |  | X |  | | 2 |  |  | X |  |  | | 3 |  |  |  | O | O | | 4 |  |  |  | O | O |   lastPos.getRow = 2  lastPos.getColumn = 2  player = ‘X’ | Output:  checkDiagonalWin = false  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing backslash diagonal but it fails, and then attempting forward slash diagonal and that fails too. So I’m checking both diagonals  Function Name:  testCheckDiagonalWin\_noWin\_  check\_both\_diagonals |

boolean checkForDraw()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X | O | O | | 1 | O | X | X | | 2 | X | X | O | | Output:  checkForDraw = true  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing to see if a full grid would result in a draw  Function Name:  testCheckForDraw\_draw\_  fullGrid |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  | X |  | | 2 |  |  |  | | Output:  checkForDraw = false  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing for boundary cases where there is only one marker on grid  Function Name:  testCheckForDraw\_noDraw\_  boundary\_oneMarker |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O | X | O | | 1 | O | X | X | | 2 |  | O | X | | Output:  checkForDraw = false  state of board is unchanged | Reason:  This test case is unique and distinct because I am testing for the boundary case where the grid is almost full. Only one position empty.  Function Name:  testCheckForDraw\_noDraw\_  \_boundary\_almostFull |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X | O |  | | 1 | O |  |  | | 2 |  | X | X | | Output:  checkForDraw = false  state of board is unchanged | Reason:  This test case is unique and distinct because it is a routine test case where it is in the middle of almost full and almost empty.  Function Name:  testCheckForDraw\_noDraw\_  routine |

char whatsAtPos(BoardPosition pos)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 | X |  |  | | 2 |  |  |  |   pos.getRow = 1  pos.getColumn = 0 | Output:  whatsAtPos = ‘X’  state of board is unchanged | Reason:  This test case is unique and distinct because it checks boundary case where column is 0 (minimum)  Function Name:  testWhatsAtPos\_boundary\_  column\_min |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  | X |  | | 1 |  |  |  | | 2 |  |  |  |   pos.getRow = 0  pos.getColumn = 1 | Output:  whatsAtPos = ‘X’  state of board is unchanged | Reason:  This test case is unique and distinct because it checks boundary case where row is 0 (minimum)  Function Name:  testWhatsAtPos\_boundary\_  row\_min |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  | X | | 2 |  |  |  |   pos.getRow = 1  pos.getColumn = 2 | Output:  whatsAtPos = ‘X’  state of board is unchanged | Reason:  This test case is unique and distinct because it checks boundary case where column is at the max  Function Name:  testWhatsAtPos\_boundary\_  column\_max |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  | X |  |   pos.getRow = 1  pos.getColumn = 2 | Output:  whatsAtPos = ‘X’  state of board is unchanged | Reason:  This test case is unique and distinct because it checks boundary case where the row is at the max  Function Name:WE  testWhatsAtPos\_boundary\_  row\_max |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  | X |  | | 2 |  |  |  |   pos.getRow = 1  pos.getColumn = 1 | Output:  whatsAtPos = ‘X’  state of board is unchanged | Reason:  This test case is unique and distinct because this is a routine test case where it is not in any of the boundaries  Function Name:  testWhatsAtPos\_routine |

boolean isPlayerAtPos(BoardPosition pos, char Player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  | X |  | | 2 |  |  |  |   Player = ‘O’  pos.getRow = 1  pos.getColumn = 1 | Output:  isPlayerAtPos = false  state of the board is unchanged | Reason:  This test case is unique and distinct because it is routine and tests to see if the function would return false when the player is not at pos  Function Name:  testIsPlayerAtPos\_false\_routine |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 | X |  |  | | 2 |  |  |  |   Player = ‘X’  pos.getRow = 1  pos.getColumn = 0 | Output:  isPlayerAtPos = true  state of the board is unchanged | Reason:  This test case is unique and distinct because it tests a boundary case where column is at min (o)  Function Name:  testIsPlayerAtPos\_true\_  boundary\_column\_min |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  | X |  | | 1 |  |  |  | | 2 |  |  |  |   Player = ‘X’  pos.getRow = 0  pos.getColumn = 1 | Output:  isPlayerAtPos = true  state of the board is unchanged | Reason:  This test case is unique and distinct because it tests a boundary case where row is at min (0)  Function Name:  testIsPlayerAtPos\_true\_  boundary\_row\_min |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  | X | | 2 |  |  |  |   Player = ‘X’  pos.getRow = 1  pos.getColumn = 2 | Output:  isPlayerAtPos = true  state of the board is unchanged | Reason:  This test case is unique and distinct because it tests a boundary case where column is at max  Function Name:  testIsPlayerAtPos\_true\_  boundary\_column\_max |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  | X |  |   Player = ‘X’  pos.getRow = 1  pos.getColumn = 2 | Output:  isPlayerAtPos = true  state of the board is unchanged | Reason:  This test case is unique and distinct because it tests a boundary case where row is at max  Function Name:  testIsPlayerAtPos\_true\_  boundary\_row\_max |

void placeMarker(BoardPosition marker, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  | X |  | | 1 | O |  |  | | 2 |  |  |  |   marker.getRow = 1  marker.getColumn = 1  player = ‘A’ | Output:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  | X |  | | 1 | O | A |  | | 2 |  |  |  | | Reason:  This test case is unique and distinct because I am placing a marker representing a player who has not been placed on this board before  Function Name:  testPlaceMarker\_routine\_new  \_player |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  |   marker.getRow = 0  marker.getColumn = 1  player = ‘X’ | Output:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  | X |  | | 1 |  |  |  | | 2 |  |  |  | | Reason:  This test case is unique and distinct because I am placing a marker at a boundary case where row is at min  Function Name:  testPlaceMarker\_boundary\_  row\_min |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  |   marker.getRow = 1  marker.getColumn = 2  player = ‘X’ | Output:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  | X | | 2 |  |  |  | | Reason:  This test case is unique and distinct because I am placing a marker at a boundary case where column is at max  Function name:  testPlaceMarker\_boundary\_  column\_max |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  |   marker.getRow = 2  marker.getColumn = 1  player = ‘X’ | Output:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  | X |  | | Reason:  This test case is unique and distinct because I am placing a marker at a boundary case where row is at max  Function name:  testPlaceMarker\_boundary\_  row\_max |
| Input:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  |   marker.getRow = 1  marker.getColumn = 0  player = ‘X’ | Output:  State:  numToWin = 3   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 | X |  |  | | 2 |  |  |  | | Reason:  This test case is unique and distinct because I am placing a marker at a boundary case where column is at min  Function name:  testPlaceMarker\_boundary\_  column\_min |