CPSC 2150 Project Report

David Hayden Copeland

**Requirements Analysis**

**Functional Requirements:**

1. As a player I receive clear instructions so that I know how to play the game
2. As a Player I can Input my game-move so that I can play the game
3. As a Player I can clearly see the board after each turn so that I can make my next decision based on what moves are available
4. As a Player I can clearly see the board after each turn so that I can see the moves that the opponent made
5. As a player, I need the game to let me know if I win or lose so that I can decide if I want to play another game
6. As a player I cannot pick a position that is already played or a position so that I do not waste my turn
7. As a player I can choose to start again by clicking another space after a win or draw so I don’t need to restart the program to play again
8. As a player, I need the game to congratulate me so that there is added validation to playing well
9. As a player, I can enter the number of rows so that I can create different game scenarios to player
10. As a player, I can enter the number of columns so that I can create different game scenarios to player
11. As a player, I can enter the number of marks in a row to win so that I can create different game scenarios to player
12. As a player, I can select a row & column combination so that I can indicate which column I want to play on
13. As a player I can distinguish between my opponents input and my own, so I do not get confused who’s played in which spot on the board

**Non-Functional Requirements**

1. The program is written in Java
2. The game must function according to the user stories
3. System must run in a reasonable amount of time – little to no input/output delay
4. Output must correctly display the current gameboard with previous moves from both sides
5. The game implements Java GUI to make the game is more enjoyable to look at
6. The game must be adaptable and able to run on other systems
7. Code must be organized and readable so that it is easy to debug, read and change things as needed
8. The game is a looped event so that the game can be restarted after it is won, lost, or drawn.

**Deployment Instructions**Details in Projects 2-5.

“You do not need to provide anything for the deployment of this program. We will not be able to run this program from our SoC Unix command line since it uses a GUI, so you do not need to provide a makefile. See the Submission section below for more details.”

**System Design**

**TicTacToeController:**

**ClassDiagram**

**A picture containing chart

Description automatically generated**

**Activity diagram:**

**Diagram

Description automatically generated**

**BoardPosition:**

**Class diagram**

**Table

Description automatically generated**

**Activity diagrams**

**GameboardMem:**

**Class diagram:**

A picture containing table

Description automatically generated

**Activity Diagram:**

**Diagram

Description automatically generated**

**Gameboard:**

**Class diagram**

**A picture containing table

Description automatically generated**

**Activity diagrams**

**Diagram

Description automatically generated**

**Diagram

Description automatically generated**

**Diagram

Description automatically generated**

**Diagram

Description automatically generated**

**Diagram

Description automatically generated**

**IGameboard:**

**Class diagram**

**A picture containing table

Description automatically generated**

**Diagrams: Diagram

Description automatically generated**

**Diagram

Description automatically generated**

**Diagram

Description automatically generated**

**Diagram

Description automatically generated**

**A picture containing text, map, indoor

Description automatically generated**

**Diagram

Description automatically generated**

**AbsGameBoard:**

**Class diagram**

**Table

Description automatically generated**

**Activity Diagram:**

**Diagram

Description automatically generated**

**Test Cases**

Details in Project 4.

TestGameBoardMem

Test the following methods:

1. public GameBoard(int Rows,int Columns, int toWins)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  Rows = 3  Columns = 3  toWins= 3 | **Output:**   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  | | **Reason:**  This test case is unique and distinct because it creates a board that is 3X3 which is the minimum required  **Function Name:**  testConstMin |
| **Input:**  Rows = 1  Columns = 102  toWins= 26 | **Output:**   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  | | **Reason:**  This test case is unique and distinct because it checks to see if the loop will re prompt you if you input a value that is out of bounds  **Function Name:**  testConstDifRowCol |
| **Input:**  Rows = 11  Columns = 11  toWins= 11 | **Output:**   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | 0 |  |  |  |  |  |  |  |  |  |  |  | | 1 |  |  |  |  |  |  |  |  |  |  |  | | 2 |  |  |  |  |  |  |  |  |  |  |  | | 3 |  |  |  |  |  |  |  |  |  |  |  | | 4 |  |  |  |  |  |  |  |  |  |  |  | | 5 |  |  |  |  |  |  |  |  |  |  |  | | 6 |  |  |  |  |  |  |  |  |  |  |  | | 7 |  |  |  |  |  |  |  |  |  |  |  | | 8 |  |  |  |  |  |  |  |  |  |  |  | | 9 |  |  |  |  |  |  |  |  |  |  |  | | 10 |  |  |  |  |  |  |  |  |  |  |  | | **Reason:**  This test case is unique and distinct because it checks to see if the formatting is still correct even the size of the board include multi digit numbers  **Function Name:**  testConstMultiDig |

1. public boolean checkSpace(BoardPosition pos)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (number to win = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  | X |  | | 2 |  |  |  |   pos.getRow = 1  pos.getCol = 1 | **Output:**  checkSpace = true  state of the board is unchanged | **Reason:**  This check space Test is unique because it checks a space for player one on the smallest board  **Function Name:**  testCheckSpaceIsThere |
| **Input:**  State: (number to win = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O |  |  | | 1 |  | X |  | | 2 |  |  |  |   pos.getRow = 0  pos.getCol = 0 | **Output:**  checkSpace = true  state of the board is unchanged | **Reason:**  This check space Test is unique because it checks a space with a character other than player one  **Function Name:**  testCheckSpaceDifferentPlayerType |
| **Input:**   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | 0 |  |  |  |  |  |  |  |  |  |  |  | | 1 |  |  |  |  |  |  |  |  |  |  |  | | 2 |  |  |  |  |  |  |  |  |  |  |  | | 3 |  |  |  |  |  |  |  |  |  |  |  | | 4 |  |  |  |  |  |  |  |  |  |  |  | | 5 |  |  |  |  |  |  |  |  |  |  |  | | 6 |  |  |  |  |  |  |  |  |  |  |  | | 7 |  |  |  |  |  |  |  |  |  |  |  | | 8 |  |  |  |  |  |  |  |  |  |  |  | | 9 |  |  |  |  |  |  |  |  |  |  |  | | 10 |  |  |  |  |  |  |  |  |  |  | X |   pos.getRow = 10  pos.getCol = 10 | **Output:**  checkSpace = true  state of the board is unchanged | **Reason:**  This check space Test is unique because it checks a space on a board that is a bigger with double digits.  **Function Name:**  testCheckSpaceDifferentSizeBoard |

1. public boolean checkHorizontalWin(BoardPosition lastPos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X | X | X | | 1 |  |  |  | | 2 |  | O | O |   lastPos.getRow = 0  lastPos.getCol = 0  player = ‘X’ | **Output:**  checkHorizontalWin = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a win when there is a valid horizontal streak of 3 on the small board.  Function Name:  testCheckHorizontalWin\_smallBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  | X | | 1 | O | O | O |  |  | | 2 |  |  |  |  |  | | 3 | X | X | X | X | O | | 4 |  |  |  |  |  |   lastPos.getRow = 3  lastPos.getCol = 2  player = ‘X’ | **Output:** checkHorizontalWin = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a horizontal win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides  **Function Name:**  testCheckHorizontalWin\_bigBoard\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O | X | X | | 1 |  | X |  | | 2 |  |  | O |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output**: checkHorizontalWin = false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board when there is no valid winning streaks  **Function Name:**  testCheckHorizontalWin\_  notWin |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O | O | O | O | X | | 1 |  |  |  | X | X | | 2 |  |  |  |  | X | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 0  player = ‘O’ | **Output:** checkHorizontalWin = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks that a different player than player 1 has won the game  **Function Name:**  testCheckHorizontalWin\_  different\_player\_win |

1. public boolean checkVerticalWin(BoardPosition lastPos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | X | O | O | | 2 | X |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output:** checkVerticalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a win when there is a valid vertical streak of 3 on the small board.  Function Name:  testCheckVerticalWin\_  win\_smallBoard\_win |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | O |  | X | | 1 |  |  |  |  | X | | 2 |  |  | O |  | X | | 3 |  |  | O |  | X | | 4 |  |  |  |  |  |   lastPos.getRow = 3  lastPos.getCol = 4  player = ‘X’ | **Output**:  checkVerticalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a vertical win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides  Function Name:  testCheckVerticalWin\_bigBoard\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | X | X |  | | 2 |  | O | O |   lastPos.getRow = 0  lastPos.getCol = 0  player = ‘X’ | **Output:** checkVerticalWin= false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for a vertical win streak when there are no valid winning streaks on the board  Function Name:  testCheckVerticalWin\_  no\_win |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | O |  | X | | 1 |  |  | O | X | X | | 2 |  |  | O |  | X | | 3 |  |  | O |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 2  lastPos.getCol = 2  player = ‘O’ | **Output:** checkVerticalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because because it checks that a different player than player 1 has won the game in a vertical streak  Function Name:  testCheckVerticalWin\_  different\_player\_win |

1. public boolean checkDiagonalWin(BoardPosition lastPos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 |  | X |  | | 2 | O | O | X |   lastPos.getRow = 2  lastPos.getCol = 2  player = ‘X’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a win when there is a valid diagonal streak of 3 on the small board going from top left corner to bottom right corner.  Function Name:  testCheckDiagonalWin\_  win\_smallBoard\_win |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  | X | | 1 |  |  |  | X |  | | 2 | O | O | X | O |  | | 3 |  | X |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 3  lastPos.getCol = 1  player = ‘X’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a diagonal win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides going from top right corner to bottom left corner.  Function Name:  testCheckDiagonalWin\_bigBoard\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  | X | | 1 |  | X | O | | 2 |  |  | O |   lastPos.getRow = 0  lastPos.getCol = 2  player = ‘X’ | **Output:** checkDiagonalWin= false  state of the board is unchanged | **Reason:**  This test case is unique and distinct it checks the board for a diagonal win streak when there are no valid winning streaks on the board  Function Name:  testCheckDiagonalWin\_  no\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 | O |  |  |  |  | | 3 | X | O |  |  |  | | 4 | X | X | O | X |  |   lastPos.getRow = 4  lastPos.getCol = 2  player = ‘O’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a diagonal win for a different player than player one and also on a big board with a smaller win streak  Function Name:  testCheckDiagonalWin\_different\_player\_win |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  | X |  | | 1 |  |  | X |  |  | | 2 | O | X |  |  |  | | 3 | X | O |  |  |  | | 4 |  |  | O |  |  |   lastPos.getRow = 0  lastPos.getCol = 3  player = ‘X’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a diagonal win for a different player than player one and also on a big board with a smaller win streak  Function Name:  testCheckDiagonalWin\_right\_up\_4\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | X | X | X |  |  | | 1 |  |  | O |  |  | | 2 |  | O |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output:** checkDiagonalWin= false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for a diagonal win when there is a different kind of win on the board  Function Name:  testCheckDiagonalWin\_different\_kind\_of\_win |
| **Input:**  State: (numtoWin = 5)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O |  |  |  |  | | 1 |  | O |  |  |  | | 2 |  | X | O |  |  | | 3 | X | X | X | O |  | | 4 |  | X |  |  | O |   lastPos.getRow = 4  lastPos.getCol = 4  player = ‘X’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks a win from the top left board all the way down to the bottom left. Also checks this for another player.  Function Name:  testCheckDiagonalWin\_left\_down\_full\_board\_win |

1. public boolean checkForDraw()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O | O | X | | 1 | X | X | O | | 2 | O | X | O |   lastPos.getRow = 2  lastPos.getCol = 2  player = ‘X’ | **Output:** checkForDraw = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for when there is a proper draw on a small board  Function Name:  testCheckForDraw\_smallBoard\_draw |
| **Input:**  State: (numtoWin = 5)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O | X | X | O | O | | 1 | X | X | O | X | O | | 2 | X | X | X | X | O | | 3 | X | X | O | O | X | | 4 | O | O | X | O | O |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output:** checkForDraw = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for when there is a proper draw on a big board  Function Name:  testCheckForDraw\_bigBoard |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  | O | | 1 |  | X |  | | 2 |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output**: checkForDraw = false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a draw when there is not a valid draw on the board  Function Name:  testCheckForDraw  no\_draw |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output**: checkForDraw = false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a draw when the board is completely empty  Function Name:  testCheckForDraw\_EmptyBoard |

1. whatsAtPos - Create 5 distinct test cases

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | X |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output:** returns “X”  state of the board remains unchanged | this test case is unique because it checks to see if there's a player at 0,0  Function Name:  test\_whatsAtPos\_smallBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  | X |   lastPos.getRow = 4  lastPos.getCol = 4  player = ‘X’ | **Output**: returns ‘X’  state of the board is unchanged | this test case is unique because it checks for a player on a bigger board outside the bounds of a board minimum  Function Name:  test\_whatsAtPos\_bigBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘ ’ | **Output**: returns “ “  state of the board is unchanged | this test case is unique because it checks an empty space  test\_whatsAtPos\_emptySpace |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  | O |  |  |  | | 1 | X |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘O’ | **Output**: returns O  state of the board is unchanged | this test case is unique because it checks for a board position that contains a player other than player one  test\_whatsAtPos\_differentPlayer |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output**: null  state of the board is unchanged | this test case is unique because it checks to make sure that the spots outside the bounds of the board are empty  test\_whatsAtPos\_outside\_Bounds |

1. public boolean isPlayerAtPos(BoardPosition pos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | X | O | O | | 2 | X |  |  |   lastPos.getRow = 1  lastPos.getCol = 0  player = ‘X’ | **Output**: isPlayerAtPos = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks to see if a player at a position when it is  Function Name:  testIsPlayerAtPos\_smallBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | X |  |  | | 1 | O |  | X |  |  | | 2 |  | O | X |  |  | | 3 |  |  | X |  |  | | 4 |  |  |  | O |  |   lastPos.getRow = 4  lastPos.getCol = 3  player = ‘X’ | **Output:** isPlayerAtPos = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks a different position and different board size to see if a player is there  Function Name:  testIsPlayerAtPos\_bigBoard |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O | X | X | | 1 |  |  | X | | 2 | O |  | O |   lastPos.getRow = 2  lastPos.getCol = 1  player = ‘X’ | **Output**: isPlayerAtPos = false  **Output:** isPlayerAtPos = false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks to see if a player is at a position when it isn’t  Function Name:  testIsPlayerAtPos\_no\_player |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O |  |  |  | X | | 1 | O |  | X | X | X | | 2 | O |  |  |  |  | | 3 | O |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 0  player = ‘X’ | **Output:** isPlayerAtPos = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for a positon when the player is a player other than player 1  Function Name:  testIsPlayerAtPos\_different\_player |
| **Input:**   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  |   lastPos.getRow = 2  lastPos.getCol = 1  player = ‘X’ | Output: isPlayerAtPos = false  state of the board is unchanged | This test case is unique and distinct because it checks to see if a player is at a position when a different player is already there  testIsPlayerAtPos\_different\_player\_is\_in\_Position |

1. public void placeMarker(BoardPosition marker, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | O |  | O | | 2 | X |  |  |   marker.getRow = 2  marker.getCol = 2  player = ‘X’ | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | O | X | O | | 2 | X |  |  | | **Reason:**  This test case is unique and distinct because is places a marker on a small board  Function Name:  testPlaceMarker\_smallBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | X |  |  | | 1 | O |  | X |  |  | | 2 |  | O | X |  |  | | 3 |  |  | X |  |  | | 4 |  |  |  | O |  |   marker.getRow = 0  marker.getCol = 1  player = ‘X’ | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | X |  |  | | 1 | O |  | X |  |  | | 2 |  | O | X |  |  | | 3 |  |  | X |  |  | | 4 |  |  |  | O |  | | **Reason:**  This test case is unique and distinct because is places a marker on a bigger board  Function Name:  test\_placeMarker\_ZeroZero |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O |  | X | | 1 |  |  |  | | 2 | X |  | O |   marker.getRow = 4  marker.getCol = 4  player = ‘X’ | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O |  | X | | 1 |  |  |  | | 2 | X |  | O | | **Reason:**  This test case is unique and distinct because it tries to place a marker on a position that is not on the board  Function Name:  test\_placeMarker\_bigBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 | X |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   marker.getRow = 1  marker.getCol = 1  player = ‘O’ | State of board:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 | X | O |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  | | **Reason:**  This test case is unique and distinct because it tries to place a marker of a player other than player one  Function Name:  test\_placeMarker\_ThreePlayers |
| **Input:**   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 |  |  |  | | 2 |  |  |  |   marker.getRow = 0  marker.getCol = 0  player = ‘O’ | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 |  |  |  | | 2 |  |  |  | | **Reason:**  This test case is unique and distinct because it places a marker on a position that is already occupied by another player  Function Name:  test\_placeMarker\_filledBoard |

**Test Cases**

Details in Project 4.

TestGameBoard

Test the following methods:

1. public GameBoard(int Rows,int Columns, int toWins)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  Rows = 3  Columns = 3  toWins= 3 | **Output:**   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  | | **Reason:**  This test case is unique and distinct because it creates a board that is 3X3 which is the minimum required  **Function Name:**  testConstMin |
| **Input:**  Rows = 1  Columns = 102  toWins= 26 | **Output:**   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  | | **Reason:**  This test case is unique and distinct because it checks to see if the loop will re prompt you if you input a value that is out of bounds  **Function Name:**  testConstDifRowCol |
| **Input:**  Rows = 11  Columns = 11  toWins= 11 | **Output:**   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | 0 |  |  |  |  |  |  |  |  |  |  |  | | 1 |  |  |  |  |  |  |  |  |  |  |  | | 2 |  |  |  |  |  |  |  |  |  |  |  | | 3 |  |  |  |  |  |  |  |  |  |  |  | | 4 |  |  |  |  |  |  |  |  |  |  |  | | 5 |  |  |  |  |  |  |  |  |  |  |  | | 6 |  |  |  |  |  |  |  |  |  |  |  | | 7 |  |  |  |  |  |  |  |  |  |  |  | | 8 |  |  |  |  |  |  |  |  |  |  |  | | 9 |  |  |  |  |  |  |  |  |  |  |  | | 10 |  |  |  |  |  |  |  |  |  |  |  | | **Reason:**  This test case is unique and distinct because it checks to see if the formatting is still correct even the size of the board include multi digit numbers  **Function Name:**  testConstMultiDig |

1. public boolean checkSpace(BoardPosition pos)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (number to win = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  | X |  | | 2 |  |  |  |   pos.getRow = 1  pos.getCol = 1 | **Output:**  checkSpace = true  state of the board is unchanged | **Reason:**  This check space Test is unique because it checks a space for player one on the smallest board  **Function Name:**  testCheckSpaceIsThere |
| **Input:**  State: (number to win = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O |  |  | | 1 |  | X |  | | 2 |  |  |  |   pos.getRow = 0  pos.getCol = 0 | **Output:**  checkSpace = true  state of the board is unchanged | **Reason:**  This check space Test is unique because it checks a space with a character other than player one  **Function Name:**  testCheckSpaceDifferentPlayerType |
| **Input:**   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | 0 |  |  |  |  |  |  |  |  |  |  |  | | 1 |  |  |  |  |  |  |  |  |  |  |  | | 2 |  |  |  |  |  |  |  |  |  |  |  | | 3 |  |  |  |  |  |  |  |  |  |  |  | | 4 |  |  |  |  |  |  |  |  |  |  |  | | 5 |  |  |  |  |  |  |  |  |  |  |  | | 6 |  |  |  |  |  |  |  |  |  |  |  | | 7 |  |  |  |  |  |  |  |  |  |  |  | | 8 |  |  |  |  |  |  |  |  |  |  |  | | 9 |  |  |  |  |  |  |  |  |  |  |  | | 10 |  |  |  |  |  |  |  |  |  |  | X |   pos.getRow = 10  pos.getCol = 10 | **Output:**  checkSpace = true  state of the board is unchanged | **Reason:**  This check space Test is unique because it checks a space on a board that is a bigger with double digits.  **Function Name:**  testCheckSpaceDifferentSizeBoard |

1. public boolean checkHorizontalWin(BoardPosition lastPos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X | X | X | | 1 |  |  |  | | 2 |  | O | O |   lastPos.getRow = 0  lastPos.getCol = 0  player = ‘X’ | **Output:**  checkHorizontalWin = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a win when there is a valid horizontal streak of 3 on the small board.  Function Name:  testCheckHorizontalWin\_smallBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  | X | | 1 | O | O | O |  |  | | 2 |  |  |  |  |  | | 3 | X | X | X | X | O | | 4 |  |  |  |  |  |   lastPos.getRow = 3  lastPos.getCol = 2  player = ‘X’ | **Output:** checkHorizontalWin = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a horizontal win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides  **Function Name:**  testCheckHorizontalWin\_bigBoard\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O | X | X | | 1 |  | X |  | | 2 |  |  | O |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output**: checkHorizontalWin = false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board when there is no valid winning streaks  **Function Name:**  testCheckHorizontalWin\_  notWin |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O | O | O | O | X | | 1 |  |  |  | X | X | | 2 |  |  |  |  | X | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 0  player = ‘O’ | **Output:** checkHorizontalWin = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks that a different player than player 1 has won the game  **Function Name:**  testCheckHorizontalWin\_  different\_player\_win |

1. public boolean checkVerticalWin(BoardPosition lastPos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | X | O | O | | 2 | X |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output:** checkVerticalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a win when there is a valid vertical streak of 3 on the small board.  Function Name:  testCheckVerticalWin\_  win\_smallBoard\_win |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | O |  | X | | 1 |  |  |  |  | X | | 2 |  |  | O |  | X | | 3 |  |  | O |  | X | | 4 |  |  |  |  |  |   lastPos.getRow = 3  lastPos.getCol = 4  player = ‘X’ | **Output**:  checkVerticalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a vertical win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides  Function Name:  testCheckVerticalWin\_bigBoard\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | X | X |  | | 2 |  | O | O |   lastPos.getRow = 0  lastPos.getCol = 0  player = ‘X’ | **Output:** checkVerticalWin= false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for a vertical win streak when there are no valid winning streaks on the board  Function Name:  testCheckVerticalWin\_  no\_win |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | O |  | X | | 1 |  |  | O | X | X | | 2 |  |  | O |  | X | | 3 |  |  | O |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 2  lastPos.getCol = 2  player = ‘O’ | **Output:** checkVerticalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because because it checks that a different player than player 1 has won the game in a vertical streak  Function Name:  testCheckVerticalWin\_  different\_player\_win |

1. public boolean checkDiagonalWin(BoardPosition lastPos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 |  | X |  | | 2 | O | O | X |   lastPos.getRow = 2  lastPos.getCol = 2  player = ‘X’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a win when there is a valid diagonal streak of 3 on the small board going from top left corner to bottom right corner.  Function Name:  testCheckDiagonalWin\_  win\_smallBoard\_win |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  | X | | 1 |  |  |  | X |  | | 2 | O | O | X | O |  | | 3 |  | X |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 3  lastPos.getCol = 1  player = ‘X’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a diagonal win in a different row from the middle of the streak, meaning it has to utilize checking the left and right sides going from top right corner to bottom left corner.  Function Name:  testCheckDiagonalWin\_bigBoard\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  | X | | 1 |  | X | O | | 2 |  |  | O |   lastPos.getRow = 0  lastPos.getCol = 2  player = ‘X’ | **Output:** checkDiagonalWin= false  state of the board is unchanged | **Reason:**  This test case is unique and distinct it checks the board for a diagonal win streak when there are no valid winning streaks on the board  Function Name:  testCheckDiagonalWin\_  no\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 | O |  |  |  |  | | 3 | X | O |  |  |  | | 4 | X | X | O | X |  |   lastPos.getRow = 4  lastPos.getCol = 2  player = ‘O’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a diagonal win for a different player than player one and also on a big board with a smaller win streak  Function Name:  testCheckDiagonalWin\_different\_player\_win |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  | X |  | | 1 |  |  | X |  |  | | 2 | O | X |  |  |  | | 3 | X | O |  |  |  | | 4 |  |  | O |  |  |   lastPos.getRow = 0  lastPos.getCol = 3  player = ‘X’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a diagonal win for a different player than player one and also on a big board with a smaller win streak  Function Name:  testCheckDiagonalWin\_right\_up\_4\_win |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | X | X | X |  |  | | 1 |  |  | O |  |  | | 2 |  | O |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output:** checkDiagonalWin= false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for a diagonal win when there is a different kind of win on the board  Function Name:  testCheckDiagonalWin\_different\_kind\_of\_win |
| **Input:**  State: (numtoWin = 5)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O |  |  |  |  | | 1 |  | O |  |  |  | | 2 |  | X | O |  |  | | 3 | X | X | X | O |  | | 4 |  | X |  |  | O |   lastPos.getRow = 4  lastPos.getCol = 4  player = ‘X’ | **Output:** checkDiagonalWin= true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks a win from the top left board all the way down to the bottom left. Also checks this for another player.  Function Name:  testCheckDiagonalWin\_left\_down\_full\_board\_win |

1. public boolean checkForDraw()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O | O | X | | 1 | X | X | O | | 2 | O | X | O |   lastPos.getRow = 2  lastPos.getCol = 2  player = ‘X’ | **Output:** checkForDraw = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for when there is a proper draw on a small board  Function Name:  testCheckForDraw\_smallBoard\_draw |
| **Input:**  State: (numtoWin = 5)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O | X | X | O | O | | 1 | X | X | O | X | O | | 2 | X | X | X | X | O | | 3 | X | X | O | O | X | | 4 | O | O | X | O | O |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output:** checkForDraw = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for when there is a proper draw on a big board  Function Name:  testCheckForDraw\_bigBoard |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  | O | | 1 |  | X |  | | 2 |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output**: checkForDraw = false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a draw when there is not a valid draw on the board  Function Name:  testCheckForDraw  no\_draw |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ | **Output**: checkForDraw = false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks for a draw when the board is completely empty  Function Name:  testCheckForDraw\_EmptyBoard |

1. whatsAtPos - Create 5 distinct test cases

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ |  | this test case is unique because it checks to see if there's a player at 0,0  Function Name:  test\_whatsAtPos\_smallBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ |  | this test case is unique because it checks for a player on a bigger board outside the bounds of a board minimum  Function Name:  test\_whatsAtPos\_bigBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ |  | this test case is unique because it checks an empty space  test\_whatsAtPos\_emptySpace |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ |  | this test case is unique because it checks for a board position that contains a player other than player one  test\_whatsAtPos\_differentPlayer |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 |  |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 1  player = ‘X’ |  | this test case is unique because it checks to make sure that the spots outside the bounds of the board are empty  test\_whatsAtPos\_outside\_Bounds |

1. public boolean isPlayerAtPos(BoardPosition pos, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | X | O | O | | 2 | X |  |  |   lastPos.getRow = 1  lastPos.getCol = 0  player = ‘X’ | **Output**: isPlayerAtPos = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks to see if a player at a position when it is  Function Name:  testIsPlayerAtPos\_smallBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | X |  |  | | 1 | O |  | X |  |  | | 2 |  | O | X |  |  | | 3 |  |  | X |  |  | | 4 |  |  |  | O |  |   lastPos.getRow = 4  lastPos.getCol = 3  player = ‘X’ | **Output:** isPlayerAtPos = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks a different position and different board size to see if a player is there  Function Name:  testIsPlayerAtPos\_bigBoard |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O | X | X | | 1 |  |  | X | | 2 | O |  | O |   lastPos.getRow = 2  lastPos.getCol = 1  player = ‘X’ | **Output**: isPlayerAtPos = false  **Output:** isPlayerAtPos = false  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks to see if a player is at a position when it isn’t  Function Name:  testIsPlayerAtPos\_no\_player |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 | O |  |  |  | X | | 1 | O |  | X | X | X | | 2 | O |  |  |  |  | | 3 | O |  |  |  |  | | 4 |  |  |  |  |  |   lastPos.getRow = 0  lastPos.getCol = 0  player = ‘X’ | **Output:** isPlayerAtPos = true  state of the board is unchanged | **Reason:**  This test case is unique and distinct because it checks the board for a positon when the player is a player other than player 1  Function Name:  testIsPlayerAtPos\_different\_player |
| **Input:**   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | 2 |  |  |  |   lastPos.getRow = 2  lastPos.getCol = 1  player = ‘X’ | Output: isPlayerAtPos = false  state of the board is unchanged | This test case is unique and distinct because it checks to see if a player is at a position when a different player is already there  testIsPlayerAtPos\_different\_player\_is\_in\_Position |

1. public void placeMarker(BoardPosition marker, char player)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | O |  | O | | 2 | X |  |  |   marker.getRow = 2  marker.getCol = 2  player = ‘X’ | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 | O | X | O | | 2 | X |  |  | | **Reason:**  This test case is unique and distinct because is places a marker on a small board  Function Name:  testPlaceMarker\_smallBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | X |  |  | | 1 | O |  | X |  |  | | 2 |  | O | X |  |  | | 3 |  |  | X |  |  | | 4 |  |  |  | O |  |   marker.getRow = 0  marker.getCol = 1  player = ‘X’ | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  | X |  |  | | 1 | O |  | X |  |  | | 2 |  | O | X |  |  | | 3 |  |  | X |  |  | | 4 |  |  |  | O |  | | **Reason:**  This test case is unique and distinct because is places a marker on a bigger board  Function Name:  test\_placeMarker\_ZeroZero |
| **Input:**  State: (numtoWin = 3)   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O |  | X | | 1 |  |  |  | | 2 | X |  | O |   marker.getRow = 4  marker.getCol = 4  player = ‘X’ | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | O |  | X | | 1 |  |  |  | | 2 | X |  | O | | **Reason:**  This test case is unique and distinct because it tries to place a marker on a position that is not on the board  Function Name:  test\_placeMarker\_bigBoard |
| **Input:**  State: (numtoWin = 4)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 | X |  |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  |   marker.getRow = 1  marker.getCol = 1  player = ‘O’ | State of board:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | | 0 |  |  |  |  |  | | 1 | X | O |  |  |  | | 2 |  |  |  |  |  | | 3 |  |  |  |  |  | | 4 |  |  |  |  |  | | **Reason:**  This test case is unique and distinct because it tries to place a marker of a player other than player one  Function Name:  test\_placeMarker\_ThreePlayers |
| **Input:**   |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 |  |  |  | | 2 |  |  |  |   marker.getRow = 0  marker.getCol = 0  player = ‘O’ | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | X |  |  | | 1 |  |  |  | | 2 |  |  |  | | **Reason:**  This test case is unique and distinct because it places a marker on a position that is already occupied by another player  Function Name:  test\_placeMarker\_filledBoard |