Joshua Lin

CPSC 2150

Project 5

Kevin Plis

Testing Description for Each Test Case

Testing the Constructor

1. testConstructor\_row\_3\_col\_3\_numToWin\_3
   1. Input Values:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. initialize gameBoard by calling it’s constructor with values in parameters stated above
   2. Expected Output:
      1. gameBoard’s private variable numberOfRows will be set to 3
      2. gameBoard’s private variable numOfColumns will be set to 3
      3. gameBoard’s private variable numForWin will be set to 3
      4. gameBoard’s current state should be 3 rows with 3 columns set to all blank spaces
   3. Reason: I choose this test case to test whether setting rows, columns, and numForWin to the minimum, which is 3, would work correctly.
   4. Distinct: This test case is distinct because it tests the minimum, a boundary test, for rows, column, and numToWin.
2. testConstructor\_row\_100\_col\_100\_numToWin\_25
   1. Input Values:
      1. calls gameBoard constructor with parameter values of (100, 100, 25)
      2. initialize gameBoard by calling it’s constructor with values in parameters stated above
   2. Expected Output:
      1. gameBoard’s private variable numberOfRows will be set to 100
      2. gameBoard’s private variable numOfColumns will be set to 100
      3. gameBoard’s private variable numForWin will be set to 25
      4. gameBoard’s current state should be 100 rows with 100 columns set to all blank spaces
   3. Reason: I choose this test case to test whether setting rows, columns, and numForWin to the maximum, which is 100 for rows and columns and 25 for numForWin, would work correctly.
   4. Distinct: This test case is distinct because it tests the maximum, a boundary test, for rows, column, and numToWin.
3. testConstructor\_row\_25\_col\_10\_numToWin\_10\_routine
   1. Input Values:
      1. calls gameBoard constructor with parameter values of (25, 10, 10)
      2. initialize gameBoard by calling it’s constructor with values in parameters stated above
   2. Expected Output:
      1. gameBoard’s private variable numberOfRows will be set to 25
      2. gameBoard’s private variable numOfColumns will be set to 10
      3. gameBoard’s private variable numForWin will be set to 10
      4. gameBoard’s current state should be 25 rows with 10 columns set to all blank spaces
   3. Reason: I choose this test case to test whether setting gameBoard to have different rows and column numbers would work correctly.
   4. Distinct: This test case is distinct because it tests whether giving rows and columns different numbers would work. In addition, this would count as a routine test case since it does not test any boundary cases or any extreme cases.

Testing Check Space

1. testCheckSpace\_out\_of\_bounds
   1. Input Values:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns with all blank spaces
      4. checkSpace will have a BoardPosition object with row set to 5 and column set to 5 as a parameter
   2. Expected Output:
      1. checkSpace should return false
      2. gameBoard’s current state should not change and should still be 3 rows and columns with all spaces set to blanks
   3. Reason: I choose this test case to test whether checkSpace would correctly test an out of bounds position. Since that position does not exist within gameBoard, it is not an available space and checkSpace should return false.
   4. Distinct: This test case is distinct because it tests out of bounds, which is a boundary case.
2. testCheckSpace\_in\_bounds\_not\_blank\_space
   1. Input Values:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. gameBoard will have position 0,0 set to a ‘X’. All other spaces will be blank
      5. checkSpace will have a BoardPosition object with row set to 0 and column set to 0 as a parameter
   2. Expected Output:
      1. checkSpace should return false
      2. gameBoard’s current state should not change and should still be 3 rows and columns with 0,0 as ‘X’ and all other spaces blank
   3. Reason: I choose this test case to test whether checkSpace would correctly test an in-bounds position that already has a marker. Since that position already has a marker, it is not an available space and checkSpace should return false.
   4. Distinct: This test case is distinct because it tests a different path that checkSpace could go through. This path was not tested in the previous test case.
3. testCheckSpace\_in\_bounds\_blank\_space\_routine
   1. Input Values:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns with all blank spaces
      4. checkSpace will have a BoardPosition object with row set to 0 and column set to 0 as a parameter
   2. Expected Output:
      1. checkSpace should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with all spaces set to blanks
   3. Reason: I choose this test case to test whether checkSpace would correctly test an in-bounds position that was a blank space. Since that position exists within gameBoard and is a blank space, checkSpace should return true.
   4. Distinct: This test case is distinct because it tests another possible path of checkSpace. In addition, this is a routine (normal) test case.

Testing checkHorizontalWin

1. testHorizontalWin\_check\_left\_of\_last\_position\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 0,1 and 0,2 should all have the marker ‘X’
      5. checkHorizontalWin will have a position of 0,2 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkHorizontalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 0,1 and 0,2 having the marker ‘X’ and all other positions blank
   3. Reason: checkHorizontalWin in gameBoard should check to the left and right of the last position to see whether there would be enough markers in a row to make a win. I choose this test case to test whether checkHorizontalWin would correctly check a horizontal win with all the other positions to the left of the last position.
   4. Distinct: This test case is distinct because it tests a possible path of checkHorizontalWin, which is the checking of the markers left of the last position. It is a routine test case.
2. testHorizontalWin\_check\_right\_of\_last\_position\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 0,1 and 0,2 should all have the marker ‘X’
      5. checkHorizontalWin will have a position of 0,0 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkHorizontalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 0,1 and 0,2 having the marker ‘X’ and all other positions blank
   3. Reason: checkHorizontalWin in gameBoard should check to the left and right of the last position to see whether there would be enough markers in a row to make a win. I choose this test case to test whether checkHorizontalWin would correctly check a horizontal win with all the other positions to the right of the last position.
   4. Distinct: This test case is distinct because it tests a possible path of checkHorizontalWin, which is the checking of the markers right of the last position. It is a routine test case.
3. testHorizontalWin\_check\_left\_of\_last\_position\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 0,1 and 0,2 should all have the marker ‘X’
      5. checkHorizontalWin will have a position of 0,1 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkHorizontalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 0,1 and 0,2 having the marker ‘X’ and all other positions blank
   3. Reason: checkHorizontalWin in gameBoard should check to the left and right of the last position to see whether there would be enough markers in a row to make a win. I choose this test case to test whether checkHorizontalWin would correctly check a horizontal win with the other positions to the left and right of the last position. This test case tests whether the left and right checking of checkHorizontalWin could work correctly together.
   4. Distinct: This test case is distinct because it tests whether the two different checking done in checkHorizontalWin would work correctly together, which is the checking of the markers to the left and right of the last position. This test case seems to just combine the two test cases previously but it is distinct because the two test cases may work separately but not together. It is a routine test case.
4. testHorizontalWin\_no\_win\_routine
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 1,1 and 2,2 should all have the marker ‘X’
      5. checkHorizontalWin will have a position of 2,2 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkHorizontalWin should return false
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,1 and 2,2 having the marker ‘X’ and all other positions blank
   3. Reason: I choose this test case to test whether checkHorizontalWin would correctly check a horizontal win and return false if gameBoard did not have the correct number of markers in a row horizontally to win.
   4. Distinct: This test case is distinct because it is a routine test for checkHorizontalWin, which is the checking of the correct number of markers in a row horizontally for a win. In addition, this test case tested a no win scenario, which is different from the previous test cases.

Testing checkVerticalWin

1. testVerticalWin\_check\_upward\_of\_last\_position\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 1,0 and 2,0 should all have the marker ‘X’
      5. checkVerticalWin will have a position of 2,0 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkVerticalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,0 and 2,0 having the marker ‘X’ and all other positions blank
   3. Reason: checkVerticalWin in gameBoard should check upward and downward of the last position to see whether there would be enough markers in a vertical row to make a win. I choose this test case to test whether checkVerticalWin would correctly check a vertical win with all the other positions upward of the last position.
   4. Distinct: This test case is distinct because it tests a possible path of checkHorizontalWin, which is the checking of the markers upward of the last position. It is a routine test case.
2. testVerticalWin\_check\_downward\_of\_last\_position\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 1,0 and 2,0 should all have the marker ‘X’
      5. checkVerticalWin will have a position of 0,0 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkVerticalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,0 and 2,0 having the marker ‘X’ and all other positions blank
   3. Reason: checkVerticalWin in gameBoard should check upward and downward of the last position to see whether there would be enough markers in a vertical row to make a win. I choose this test case to test whether checkVerticalWin would correctly check a vertical win with all the other positions downward of the last position.
   4. Distinct: This test case is distinct because it tests a different possible path of checkVerticalWin, which is the checking of the markers downward of the last position. It is a routine test case.
3. testVerticalWin\_last\_pos\_in\_middle\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 1,0 and 2,0 should all have the marker ‘X’
      5. checkVerticalWin will have a position of 2,0 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkVerticalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,0 and 2,0 having the marker ‘X’ and all other positions blank
   3. Reason: checkVerticalWin in gameBoard should check upward and downward of the last position to see whether there would be enough markers in a vertical row to make a win. I choose this test case to test whether checkVerticalWin would correctly check a vertical win with all the other positions downward and upward of the last position. This test case tests whether checkVerticalWin could have the checking of upward and downward from the last position work together correctly.
   4. Distinct: This test case is distinct because it tests a different possible path of checkVerticalWin, which is the checking of the markers downward and upward of the last position. This essentially tests whether the checking of upward and downward would work correctly together. It is a routine test case.
4. testVerticalWin\_no\_win\_routine
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 1,1 and 2,2 should all have the marker ‘X’
      5. checkVerticalWin will have a position of 2,2 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkVerticalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,1 and 2,2 having the marker ‘X’ and all other positions blank
   3. Reason: I choose this test case to test whether checkVerticalWin would correctly check a vertical win and return false if gameBoard did not have the correct number of markers in a row vertically to win.
   4. Distinct: This test case is distinct because it is a routine test for checkVerticalWin, which is the checking of the correct number of markers in a row horizontally for a win. In addition, this test case tested a no win scenario, which is different from the previous test cases.

Testing checkDiagonalWin

1. testDiagonalWin\_last\_pos\_bottom\_right\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 1,1 and 2,2 should all have the marker ‘X’
      5. checkDiagonalWin will have a position of 2,2 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkDiagonalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,1 and 2,2 having the marker ‘X’ and all other positions blank
   3. Reason: I choose this test case to test whether checkDiagonalWin would correctly check a diagonal win with all the other positions to the top left of the last position and return true.
   4. Distinct: This test case is distinct because it tests a possible path of checkDiagonalWin, which is the checking of the markers to the top left of the last position. It is a routine test case.
2. testDiagonalWin\_last\_pos\_top\_left\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 1,1 and 2,2 should all have the marker ‘X’
      5. checkDiagonalWin will have a position of 0,0 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkDiagonalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,1 and 2,2 having the marker ‘X’ and all other positions blank
   3. Reason: I choose this test case to test whether checkDiagonalWin would correctly check a diagonal win with all the other positions to the bottom right of the last position and return true.
   4. Distinct: This test case is distinct because it tests a possible path of checkDiagonalWin, which is the checking of the markers to the bottom right of the last position. It is a routine test case.
3. testDiagonalWin\_last\_pos\_bottom\_left\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,2 and 1,1 and 2,0 should all have the marker ‘X’
      5. checkDiagonalWin will have a position of 2,0 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkDiagonalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,2 and 1,1 and 2,0 having the marker ‘X’ and all other positions blank
   3. Reason: I choose this test case to test whether checkDiagonalWin would correctly check a diagonal win with all the other positions to the top right of the last position and return true.
   4. Distinct: This test case is distinct because it tests a possible path of checkDiagonalWin, which is the checking of the markers to the top right of the last position. It is a routine test case.
4. testDiagonalWin\_last\_pos\_top\_right\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,2 and 1,1 and 2,0 should all have the marker ‘X’
      5. checkDiagonalWin will have a position of 0,2 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkDiagonalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,2 and 1,1 and 2,0 having the marker ‘X’ and all other positions blank
   3. Reason: I choose this test case to test whether checkDiagonalWin would correctly check a diagonal win with all the other positions to the bottom left of the last position and return true.
   4. Distinct: This test case is distinct because it tests a possible path of checkDiagonalWin, which is the checking of the markers to the bottom left of the last position. It is a routine test case.
5. testDiagonalWin\_top\_left\_to\_bottom\_right\_last\_pos\_in\_middle\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 1,1 and 2,2 should all have the marker ‘X’
      5. checkDiagonalWin will have a position of 1,1 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkDiagonalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,1 and 2,2 having the marker ‘X’ and all other positions blank
   3. Reason: I choose this test case to test whether checkDiagonalWin would correctly check a diagonal win with all the other positions to the top left and bottom right of the last position and return true.
   4. Distinct: This test case is distinct because it tests a possible path of checkDiagonalWin, which is the checking of the markers to the top left and bottom right of the last position. It is a routine test case.
6. testDiagonalWin\_top\_right\_to\_bottom\_left\_last\_pos\_in\_middle\_win
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 2,0 and 1,1 and 0,2 should all have the marker ‘X’
      5. checkDiagonalWin will have a position of 1,1 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkDiagonalWin should return true
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,1 and 2,2 having the marker ‘X’ and all other positions blank
   3. Reason: I choose this test case to test whether checkDiagonalWin would correctly check a diagonal win with all the other positions to the top right and bottom left of the last position and return true.
   4. Distinct: This test case is distinct because it tests a possible path of checkDiagonalWin, which is the checking of the markers to the top right and bottom left of the last position. It is a routine test case.
7. testDiagonalWin\_no\_win\_routine
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 and 1,1 and 0,2 should all have the marker ‘X’
      5. checkDiagonalWin will have a position of 0,2 and a marker of ‘X’ as the parameters
   2. Expected Output:
      1. checkDiagonalWin should return false
      2. gameBoard’s current state should not change and should still be 3 rows and columns with position 0,0 and 1,1 and 0,2 having the marker ‘X’ and all other positions blank
   3. Reason: I choose this test case to test whether checkDiagonalWin would correctly check a diagonal win and return false when there is no diagonal win.
   4. Distinct: This test case is distinct because it tests a possible path of checkDiagonalWin, which is having no diagonal win and returning false.

Testing checkForDraw

1. testCheckForDraw\_no\_draw\_sqaure\_board
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions should be blank spaces
   2. Expected Output:
      1. checkForDraw should return false
      2. gameBoard’s current state should not change and should still be all positions blank
   3. Reason: I choose this test case to test whether checkForDraw would correctly return false for a no draw scenario when the gameBoard was a square (row and column having the same number).
   4. Distinct: This test case is distinct because it tests a no draw scenario specifically when gameBoard has the same number of rows and columns (square)
2. testCheckForDraw\_rectangle\_board\_no\_draw
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (5, 3, 3)
      2. sets the rows to 5, and columns and numToWin to 3 for gameBoard
      3. gameBoard should have 5 rows and 3 columns
      4. all positions should be blank spaces
   2. Expected Output:
      1. checkForDraw should return false
      2. gameBoard’s current state should not change and should still be all positions blank
   3. Reason: I choose this test case to test whether checkForDraw would correctly return false for a no draw scenario when the gameBoard was a rectangle (row and column having the same number).
   4. Distinct: This test case is distinct because it tests a no draw scenario specifically when gameBoard has a different number of rows and columns (rectangle)
3. testCheckForDraw\_sqaure\_board\_draw
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns, and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions should be filled with the marker ‘X’
   2. Expected Output:
      1. checkForDraw should return true
      2. gameBoard’s current state should not change and should still be all positions with marker ‘X’
   3. Reason: I choose this test case to test whether checkForDraw would correctly return true for a draw scenario when the gameBoard was a square (row and column having the same number).
   4. Distinct: This test case is distinct because it tests a draw scenario specifically when gameBoard has the same number of rows and columns (square)
4. testCheckForDraw\_rectangle\_board\_draw
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (5, 3, 3)
      2. sets the rows to 5, and columns and numToWin to 3 for gameBoard
      3. gameBoard should have 5 rows and 3 columns
      4. all positions should be filled with the marker ‘X’
   2. Expected Output:
      1. checkForDraw should return true
      2. gameBoard’s current state should not change and should still be all positions filled with the marker ‘X’
   3. Reason: I choose this test case to test whether checkForDraw would correctly return true for a draw scenario when the gameBoard was a rectangle (row and column having the same number).
   4. Distinct: This test case is distinct because it tests a draw scenario specifically when gameBoard has a different number of rows and columns (rectangle)

Testing whatsAtPos

1. testWhatsAtPos\_blank\_space\_square\_board
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions should be blank
      5. whatsAtPos will have a boardPosition (0,0) as a parameter
   2. Expected Output:
      1. whatsAtPos should return a blank space (‘ ’)
      2. gameBoard’s current state should not change and should still be all positions with blank space
   3. Reason: I choose this test case to test whether whatsAtPos would correctly return a blank space from a position while having a square board.
   4. Distinct: This test case is distinct because it tests whether whatsAtPos would correctly return a blank space specifically when gameBoard has the same numbers of rows and columns (square) while also testing a boundary of top left corner (0,0)
2. testWhatsAtPos\_blank\_space\_rectangular\_board
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (5, 3, 3)
      2. sets the rows to 5, and columns and numToWin to 3 for gameBoard
      3. gameBoard should have 5 rows and 3 columns
      4. all positions should be blank
      5. whatsAtPos will have a boardPosition (0,0) as a parameter
   2. Expected Output:
      1. whatsAtPos should return a blank space (‘ ’)
      2. gameBoard’s current state should not change and should still be all positions with blank space
   3. Reason: I choose this test case to test whether whatsAtPos would correctly return a blank space from a position while having a rectangle board.
   4. Distinct: This test case is distinct because it tests whether whatsAtPos would correctly return a blank space specifically when gameBoard has different numbers of rows and columns (rectangle) while also testing a boundary of top left corner (0,0)
3. testWhatsAtPos\_not\_blank\_square\_board
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position 0,0 will have the marker ‘Y’ while all other positions are empty spaces
      5. whatsAtPos will have a boardPosition (0,0) as a parameter
   2. Expected Output:
      1. whatsAtPos should return a character (‘Y’)
      2. gameBoard’s current state should not change and should still be all positions with blank space
   3. Reason: I choose this test case to test whether whatsAtPos would correctly return a blank space from a position while having a square board.
   4. Distinct: This test case is distinct because it tests whether whatsAtPos would correctly return a marker at a position specifically when gameBoard has the same numbers of rows and columns (square) while also testing a boundary of top left corner (0,0)
4. testWhatsAtPos\_not\_blank\_rectangular\_board
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (5, 3, 3)
      2. sets the rows to 5, and columns and numToWin to 3 for gameBoard
      3. gameBoard should have 5 rows and 3 columns
      4. position 0,0 will have the marker ‘Y’ while all other positions are empty spaces
      5. whatsAtPos will have a boardPosition (0,0) as a parameter
   2. Expected Output:
      1. whatsAtPos should return a blank space (‘ ’)
      2. gameBoard’s current state should not change and should still be all positions with blank space
   3. Reason: I choose this test case to test whether whatsAtPos would correctly return a blank space from a position while having a rectangle board.
   4. Distinct: This test case is distinct because it tests whether whatsAtPos would correctly return a marker at a position specifically when gameBoard has different number of rows and columns (square) while also testing a boundary of top left corner (0,0)

1. testWhatsAtPos\_square\_board\_bottom\_right\_position\_blank\_space
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions should be blank
      5. whatsAtPos will have a boardPosition (2,2) as a parameter
   2. Expected Output:
      1. whatsAtPos should return a blank space (‘ ’)
      2. gameBoard’s current state should not change and should still be all positions with blank space
   3. Reason: I choose this test case to test whether whatsAtPos would correctly return a blank space from a position while having a square board.
   4. Distinct: This test case is distinct because it tests a different boundary than the previous test cases (2,2). It also tests whether or not whatsAtPos would correctly return a blank space for a square board.

Testing isPlayerAtPos

1. testIsPlayerAtPos\_square\_board\_blank\_space
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions should be blank
      5. isPlayerAtPos will have a boardPosition (0,0) and a marker ‘X’ as a parameter
   2. Expected Output:
      1. isPlayerAtPos should return false
      2. gameBoard’s current state should not change and should still be all positions having blank space
   3. Reason: I choose this test case to test whether isPlayerAtPos would correctly return false when comparing a blank space, from the position given in the parameter, with the marker ‘X’ while having a square board.
   4. Distinct: This test case is distinct because it tests whether isPlayerAtPos would correctly return false when comparing the blank space from position from the parameter with a marker ‘X’ specifically when gameBoard has the same numbers of rows and columns (square). It also tests a boundary of top left corner (0,0)
2. testIsPlayerAtPos\_rectangular\_board\_blank\_space
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (5, 3, 3)
      2. sets the rows to 5, and columns and numToWin to 3 for gameBoard
      3. gameBoard should have 5 rows and 3 columns
      4. all positions should be blank
      5. isPlayerAtPos will have a boardPosition (0,2) and a marker ‘Y’ as a parameter
   2. Expected Output:
      1. isPlayerAtPos should return false
      2. gameBoard’s current state should not change and should still be all positions having blank space
   3. Reason: I choose this test case to test whether isPlayerAtPos would correctly return false when comparing a blank space, from the position given in the parameter, with the marker ‘Y’ while having a square board.
   4. Distinct: This test case is distinct because it tests whether isPlayerAtPos would correctly return false when comparing the blank space from position from the parameter with a marker ‘Y’ specifically when gameBoard has different numbers of rows and columns (rectangle)
3. testIsPlayerAtPos\_square\_board\_not\_blank\_space
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position (2,0) should have the marker ‘X’ and all other positions should be blank
      5. isPlayerAtPos will have a boardPosition (2,0) and a marker ‘X’ as the parameter
   2. Expected Output:
      1. isPlayerAtPos should return true
      2. gameBoard’s current state should not change
   3. Reason: I choose this test case to test whether isPlayerAtPos would correctly return true when comparing ‘X’, from the position given in the parameter, with the marker ‘X’ (from the parameter) while having a square board.
   4. Distinct: This test case is distinct because it tests whether isPlayerAtPos would correctly return true when comparing ‘X’ from the position from the parameter with a marker ‘X’ specifically when gameBoard has the same numbers of rows and columns (square). In addition, it also tests a boundary of the bottom left corner.
4. testIsPlayerAtPos\_rectangular\_board\_not\_blank\_space
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (5, 3, 3)
      2. sets the rows to 5, and columns and numToWin to 3 for gameBoard
      3. gameBoard should have 5 rows and 3 columns
      4. position (2,2) should have the marker ‘Y’ and all other positions should be blank
      5. isPlayerAtPos will have a boardPosition (2,2) and a marker ‘Y’ as a parameter
   2. Expected Output:
      1. isPlayerAtPos should return true
      2. gameBoard’s current state should not change
   3. Reason: I choose this test case to test whether isPlayerAtPos would correctly return true when comparing ‘Y’, from the position given in the parameter, with the marker ‘Y’ while having a square board.
   4. Distinct: This test case is distinct because it tests whether isPlayerAtPos would correctly return true when comparing ‘Y’ from the position from the parameter with a marker ‘Y’ (from the parameter) specifically when gameBoard has different number of rows and columns (rectangle).
5. testIsPlayerAtPos\_square\_board\_marker\_not\_part\_of\_game
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. position (0,0) will contain ‘X’
      5. position (1,0) will contain ‘Z’
      6. all other positions will contain blank spaces
      7. isPlayerAtPos will have a boardPosition (1,1) and a marker ‘Y’ as a parameter
   2. Expected Output:
      1. isPlayerAtPos should return false
      2. gameBoard’s current state should not change
   3. Reason: I choose this test case to test whether isPlayerAtPos would correctly return false when comparing a marker that is not being used in the game (‘Y’) with a blank space while having a square board.
   4. Distinct: This test case is distinct because it tests whether isPlayerAtPos would work correctly when comparing with a marker not being used in the game. This is different than the previous test cases where they tested markers that were being used by players. The board is also square so it can test that too.

Testing placeMarker

1. testPlaceMarker\_top\_left\_corner
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions in gameBoard should be blank
      5. placeMarker will have a boardPosition (0,0) and a marker ‘X’ as a parameter
   2. Expected Output:
      1. Position (0,0) in the gameBoard should have the marker ‘X’ but the other positions in gameBoard should not be changed
   3. Reason: I choose this test case to test whether placeMarker would correctly place the marker in the top left corner of gameBoard.
   4. Distinct: This test case is distinct because it is a boundary test case, which tests the top left corner of placeMarker
2. testPlaceMarker\_top\_right\_corner
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions in gameBoard should be blank
      5. placeMarker will have a boardPosition (0,2) and a marker ‘X’ as a parameter
   2. Expected Output:
      1. Position (0,2) in the gameBoard should have the marker ‘X’ but the other positions in gameBoard should not be changed
   3. Reason: I choose this test case to test whether placeMarker would correctly place the marker in the top right corner of gameBoard.
   4. Distinct: This test case is distinct because it is a boundary test case, which tests the top right corner of placeMarker
3. testPlaceMarker\_bottom\_left\_corner
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions in gameBoard should be blank
      5. placeMarker will have a boardPosition (2,0) and a marker ‘X’ as a parameter
   2. Expected Output:
      1. Position (2,0) in the gameBoard should have the marker ‘X’ but the other positions in gameBoard should not be changed
   3. Reason: I choose this test case to test whether placeMarker would correctly place the marker in the bottom left corner of gameBoard.
   4. Distinct: This test case is distinct because it is a boundary test case, which tests the bottom left corner of placeMarker
4. testPlaceMarker\_bottom\_right\_corner
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions in gameBoard should be blank
      5. placeMarker will have a boardPosition (2,2) and a marker ‘X’ as a parameter
   2. Expected Output:
      1. Position (2,2) in the gameBoard should have the marker ‘X’ but the other positions in gameBoard should not be changed
   3. Reason: I choose this test case to test whether placeMarker would correctly place the marker in the bottom right corner of gameBoard.
   4. Distinct: This test case is distinct because it is a boundary test case, which tests the bottom right corner of placeMarker
5. testPlaceMarker\_middle\_pos\_special\_character\_routine
   1. InputValues:
      1. calls gameBoard constructor with parameter values of (3, 3, 3)
      2. sets the rows, columns and numToWin to 3 for gameBoard
      3. gameBoard should have 3 rows and columns
      4. all positions in gameBoard should be blank
      5. placeMarker will have a boardPosition (1,1) and a marker ‘;’ as a parameter
   2. Expected Output:
      1. Position (1,1) in the gameBoard should have the marker ‘;’ but the other positions in gameBoard should not be changed
   3. Reason: I choose this test case to test whether placeMarker would correctly place the special character marker in the middle position of gameBoard.
   4. Distinct: This test case is distinct because it is a routine test case, which is different from the previous test cases. In addition, it also tests a special character, which is different from the normal alphabet letters, to see if it works normally.