Kellen Haas CPSC 2150

Project 5

12/1/2020

# Requirements Analysis

Functional Requirements

1. As a user, I can choose which row I want to place my marker in.
2. As a user, I can choose which column I want to place my marker in.
3. As a user, I can view the game board before my turn and after my turn with my updated marker that I just placed.
4. As a user, after the first user takes their turn, I can then choose my row.
5. As a user, after the first user takes their turn, I can choose my column.
6. As a user, I can expect that the system will notify me and my opponent if someone has won horizontally.
7. As a user, I can expect that the system will notify me and my opponent if someone has won vertically.
8. As a user, I can expect that the system will notify me and my opponent if someone has won diagonally.
9. As a user, I want to be notified by the system if there is a draw.
10. As a user, if I choose a position where a marker already has been placed, the system will tell me that I cannot place a marker there.
11. As a user, if I choose a position that is out of the bounds of the board, the system will tell me that it is not a valid position.
12. As a user, I want to be able to view both mine and my opponents placed markers after every turn.
13. As a user, I expect the top of the board to be the index 0, 0.
14. As a user, I expect to be prompted to press any button on the screen to play again.
15. As a user, if I choose to play again, then the program should start over from the beginning and clear the game board.
16. As a user, if I choose to play again, then I should be able to change the number of rows.
17. As a user, if I choose to play again, then I should be able to change the number of columns.
18. As a user, if I choose to play again, then I should be able to change the number of markers in a row to win the game.
19. As a user, if I choose to play again, then I should be able to change the number of players.
20. As a user, I expect that the game will prompt a player when it is their turn.

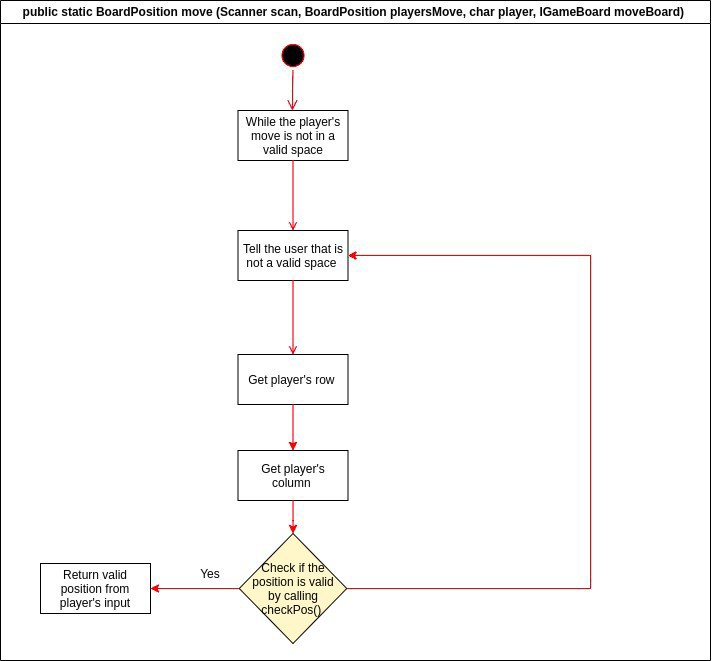
Non-Functional Requirements

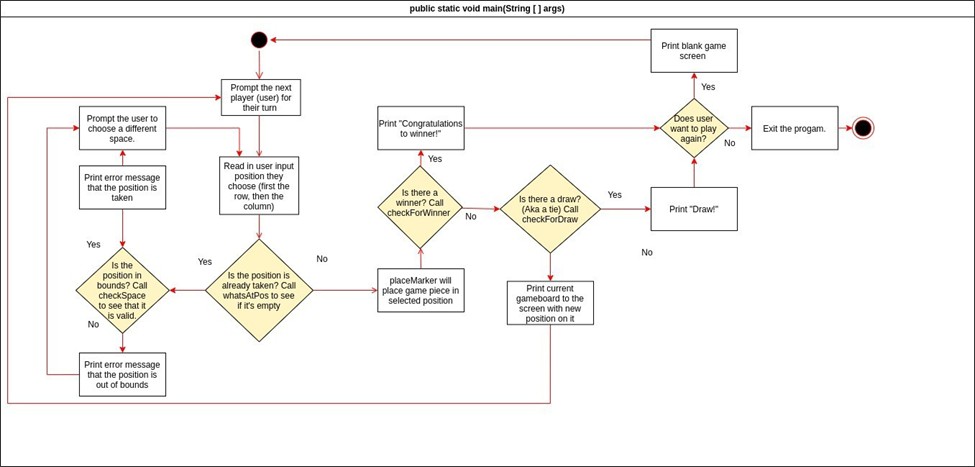
1. The system must be coded in Java programming language.
2. The system must be able to run on Unix/Linux.
3. Program must be able to compile and run quickly and efficiently.
4. The system must be written in IntelliJ IDE for debugging purposes in the future.
5. The top left corner of the game board must be 0, 0.
6. The game board is any size greater than 3 and less than 100.

# Design

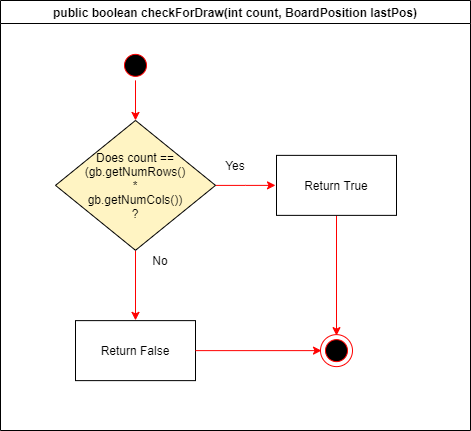
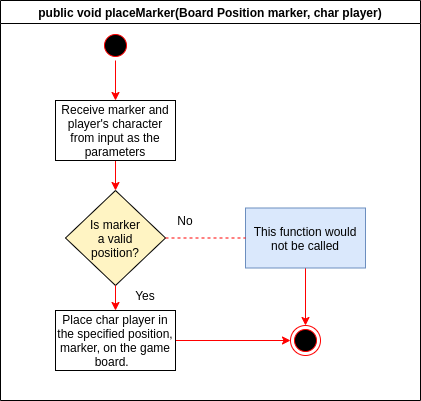
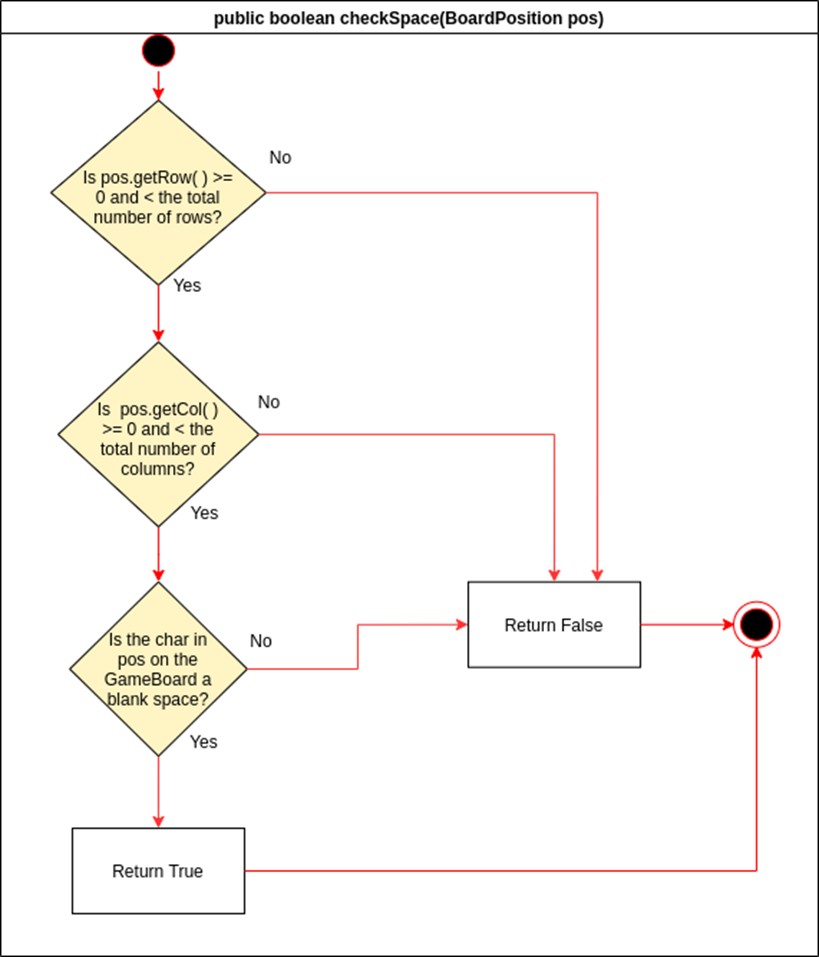
Activity Diagrams

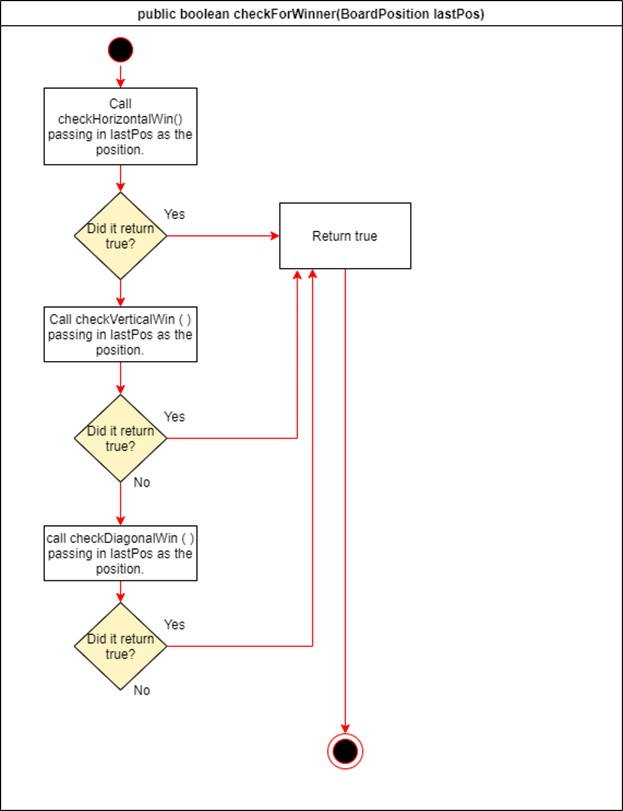
GameScreen.java

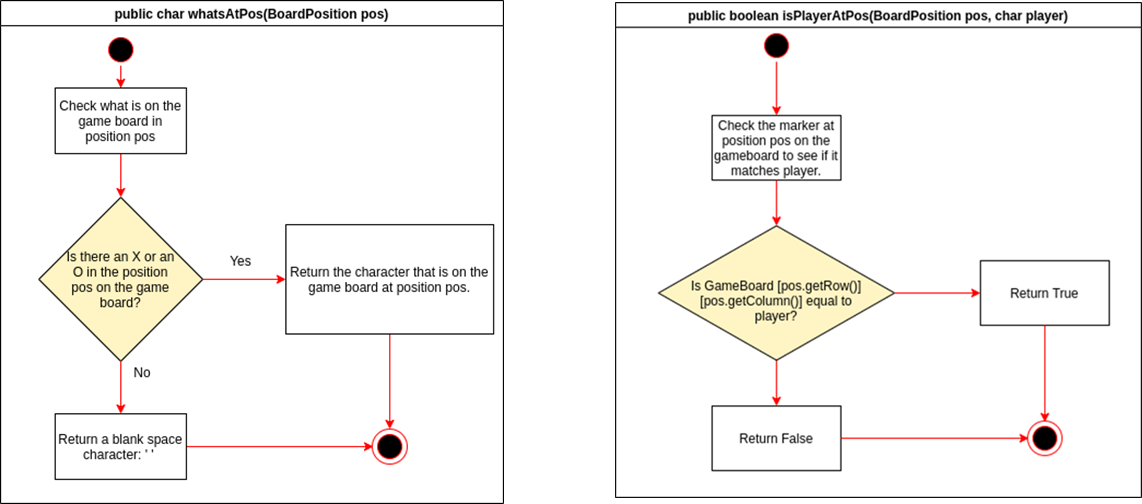


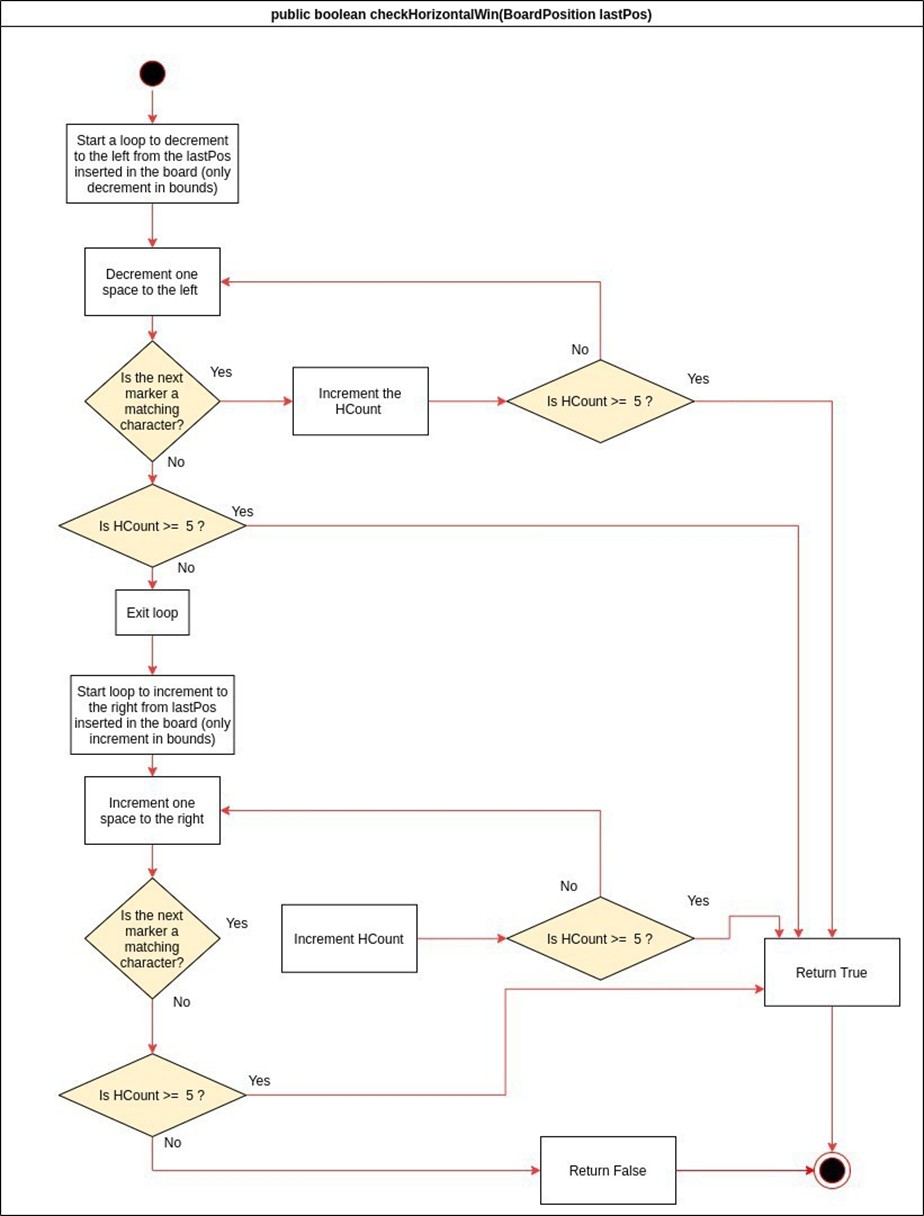


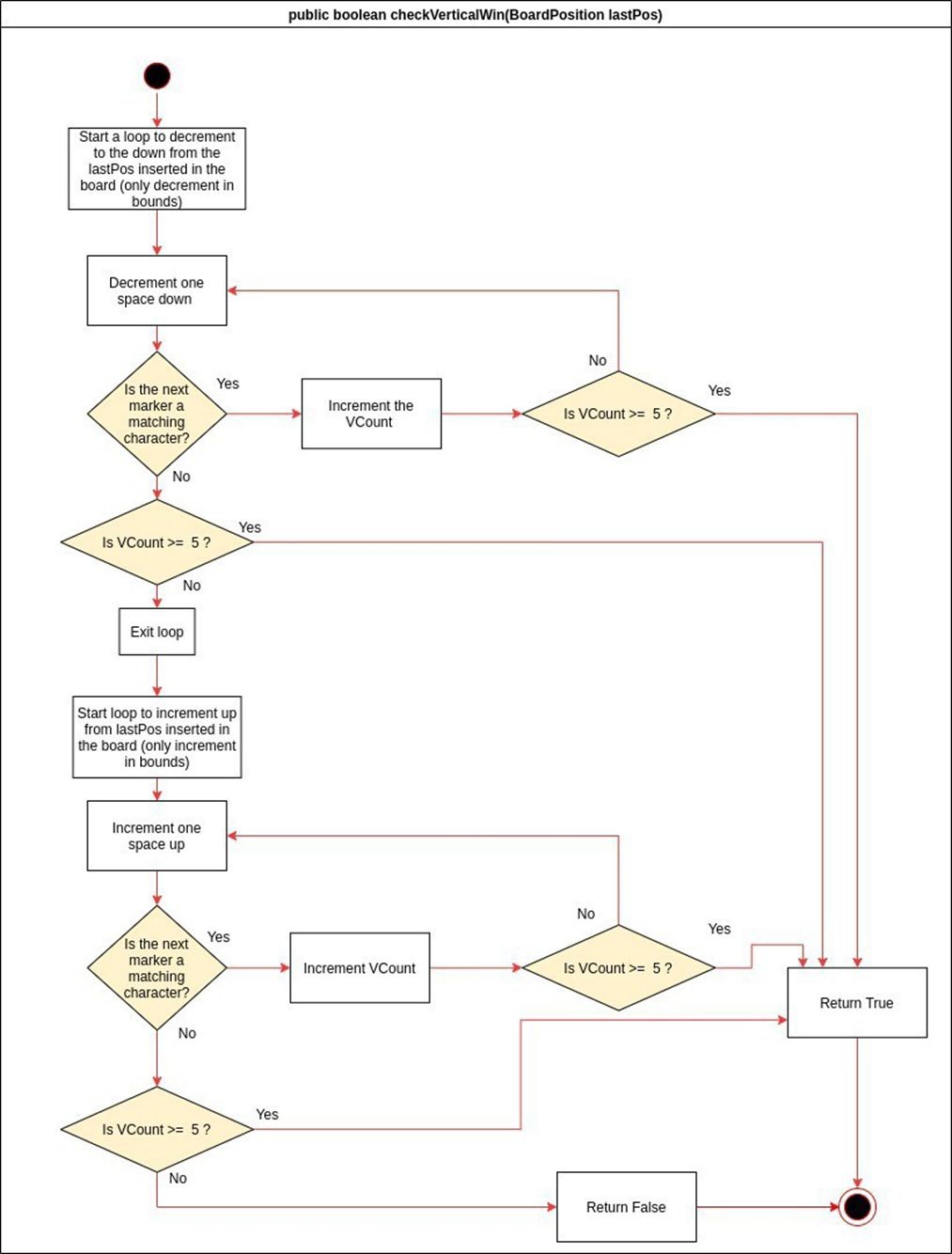
GameBoard.java

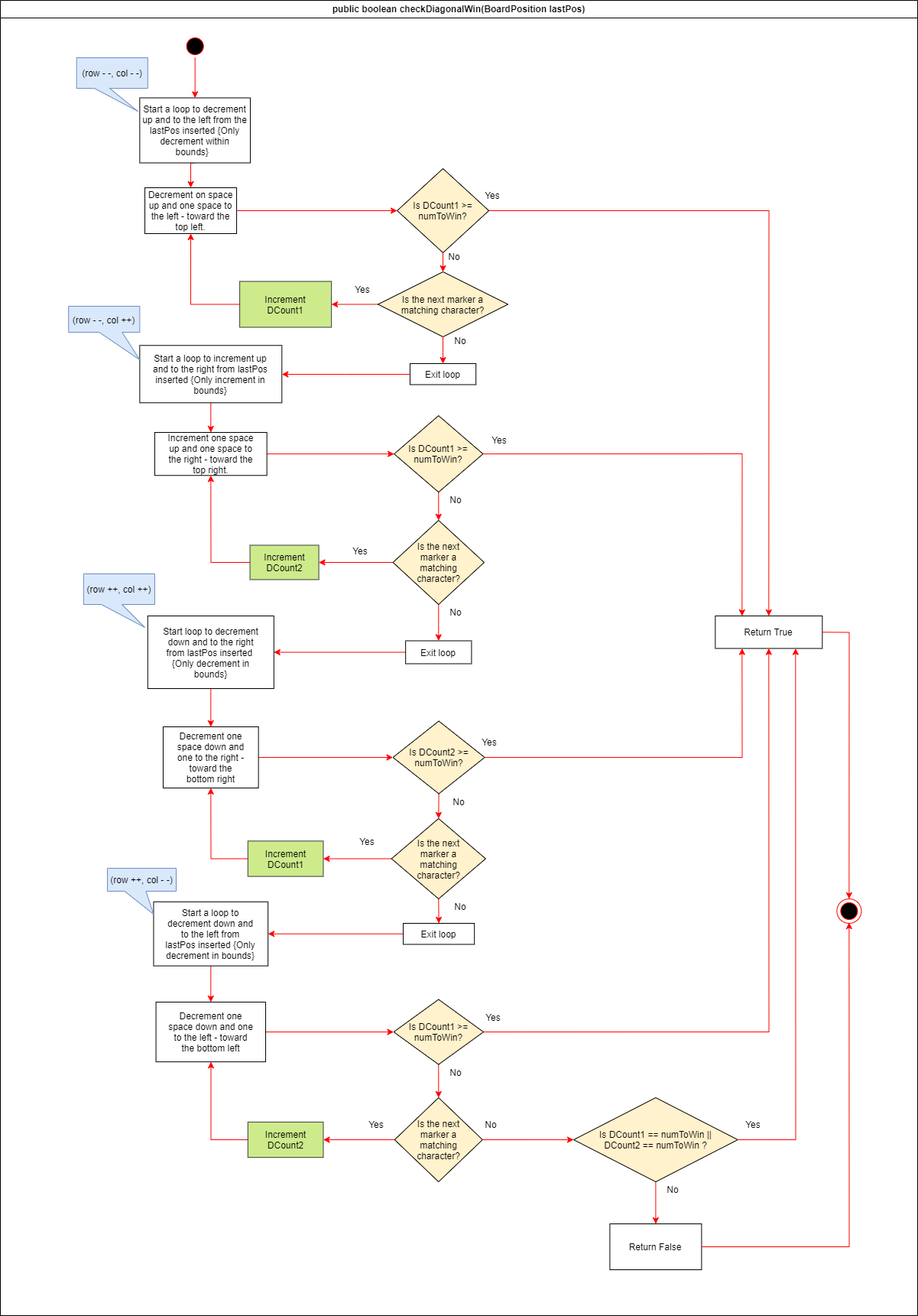


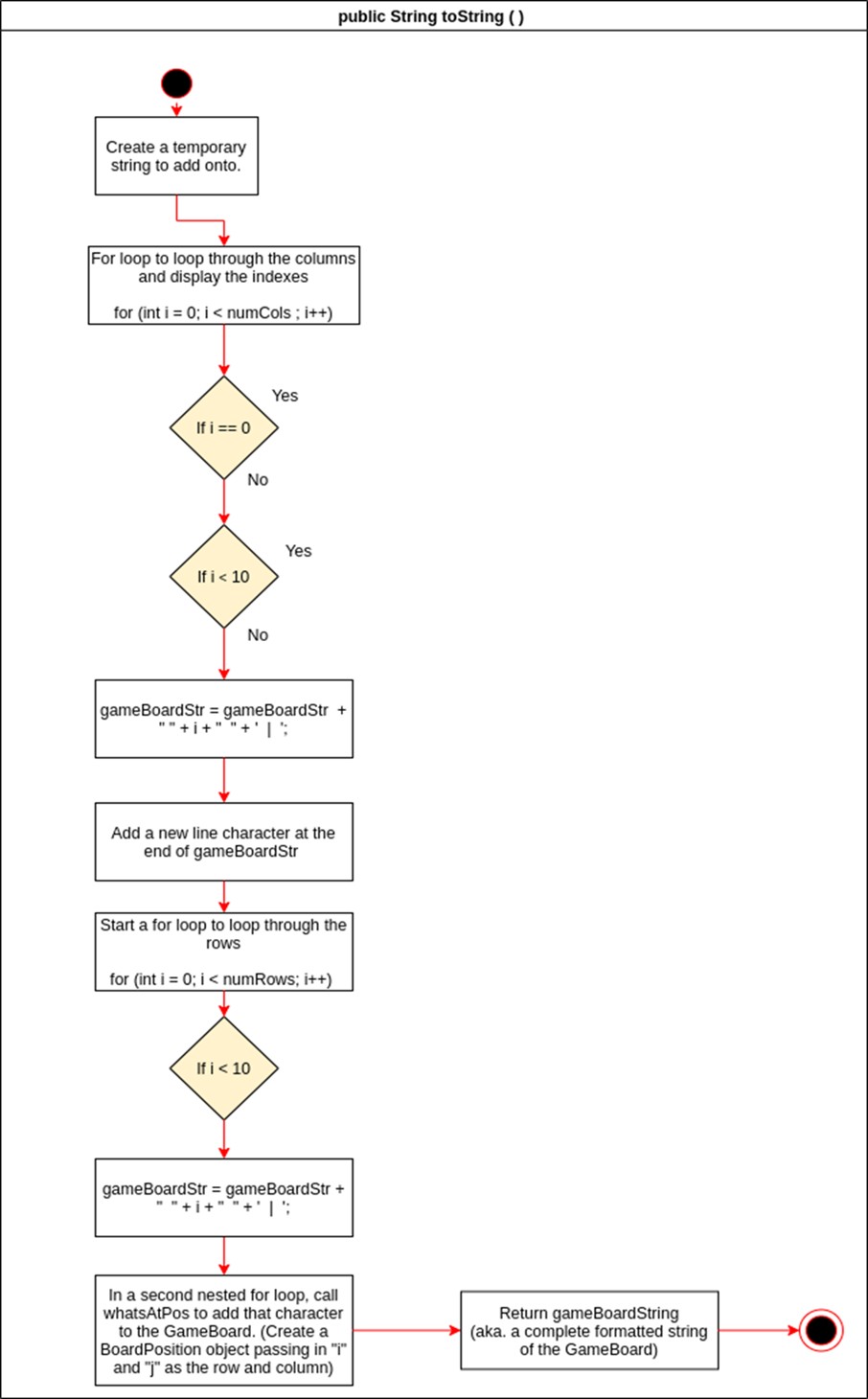




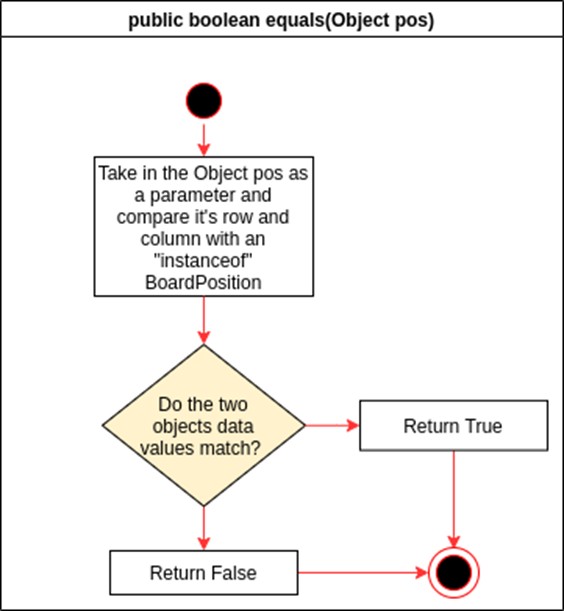
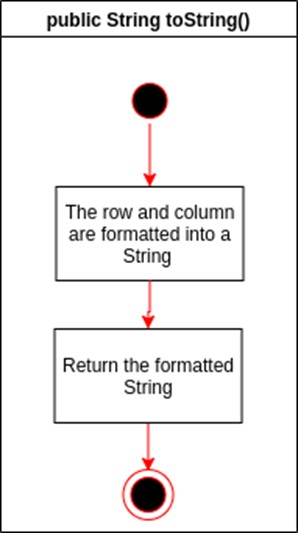


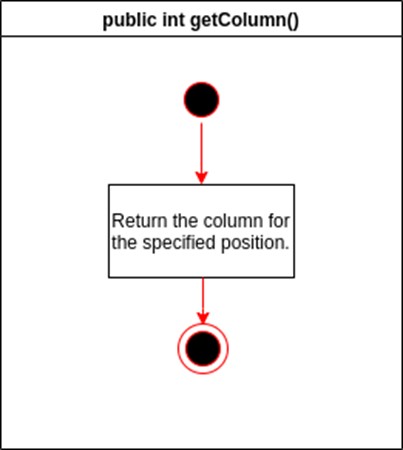
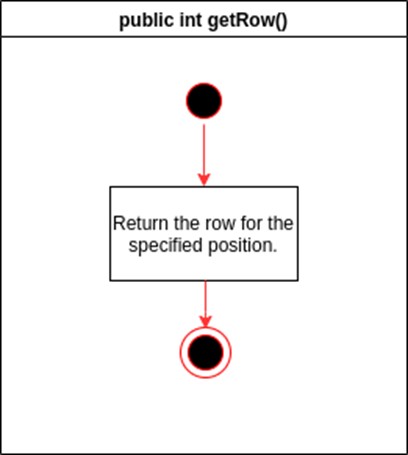






BoardPosition.java



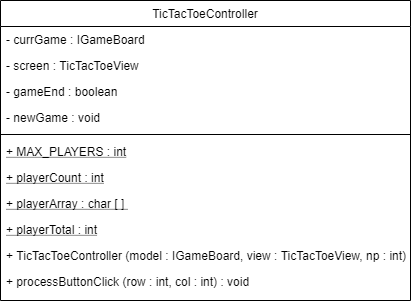


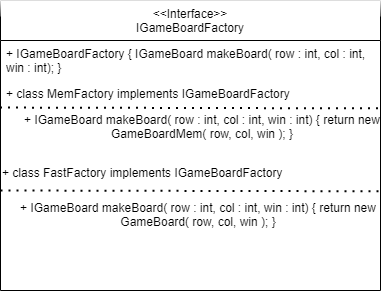
TicTacToeController.java

Diagram

Description automatically generated

# UML Class Diagrams





Testing

public void GameBoard()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:** | **Output:** | **Reason:** |
|  |  | This test case is unique and |
| State: (number to win = 3) | GameBoard constructor | distinct because it tests that the |
|  | getNumRows() = 3 | constructor successfully creates |
|  | getNumCols() = 3 | a game board using the |
|  | getNumToWin() = 3 | minimum values for row, |
|  |  | column, and number in a row to |
|  | State: unchanged | win. |
| IGameBoard gb = |  |  |
| GameBoardFactory(3, 3, 3) |  | **Function Name:**  TestGameBoard\_Minimums |

public void GameBoard()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:** | **Output:** | **Reason:** |
|  |  | This test case is unique and |
| State: (number to win = 25) | GameBoard constructor | distinct because it tests that the |
|  | getNumRows() = 100 | constructor successfully creates |
|  | getNumCols() = 100 | a game board using the |
|  | getNumToWin() = 25 | maximum values for row, |
|  |  | column, and number in a row to |
|  | State: unchanged | win. |
| IGameBoard gb = |  |  |
| GameBoardFactory(100, 100, 25) |  | **Function Name:**  TestGameBoard\_Maximums |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  | X |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

public void GameBoard()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:** | **Output:** | **Reason:** |
|  |  | This test case is unique and distinct because it |
| State: (number to win = 3) | GameBoard constructor | tests that the constructor successfully creates |
|  | getNumRows() = 4 | a game board using different values for row |
|  | getNumCols() = 6 | and column. |
|  | getNumToWin() = 3 |  |
|  | State: unchanged | **Function Name:**  TestGameBoard\_Diff\_RowsAndCols |
| IGameBoard gb = |  |  |
| GameBoardFactory(4, 6, 3) |  |  |

public void placeMarker(BoardPosition testPos, char player)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 whatsAtPos(testPos) = ‘X’ | **Reason:**  This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in the bottom right corner on the game board (2, 2).  **Function Name:**  Test\_PlaceMarker\_in\_Bottom\_Right\_Corner() |
|  | State: |  |
| IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ |  |  |
| testPos = new BoardPosition (2, 2) gb.placeMarker (testPos, player) |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

public void placeMarker(BoardPosition testPos, char player)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  | X |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 whatsAtPos(testPos) = ‘X’ | **Reason:**  This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in top right corner on the game board (0, 2). |
|  | State: | **Function Name:**  Test\_PlaceMarker\_in\_Top\_Right\_Corner() |
| IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ |  |  |
| testPos = new BoardPosition (0, 2) gb.placeMarker (testPos, player) |  |  |

public void placeMarker(BoardPosition testPos, char player)

**Output:**

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 whatsAtPos(testPos) = ‘X’

State:

IGameBoard gb =

GameBoardFactory(3, 3, 3) player = ‘X’

testPos = new BoardPosition (2, 0) gb.placeMarker (testPos, player)

**Input:**

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 | X |  |  |

**Reason:**

This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in the bottom left corner on the game board (2, 0).

**Function Name:**

Test\_PlaceMarker\_in\_Bottom\_Left\_Corner()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X | O | X |
| 1 | O | X | O |
| 2 | X | O | X |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

public void placeMarker(BoardPosition testPos, char player)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 whatsAtPos(testPos) = ‘X’ | **Reason:**  This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in top left corner on the game board (0, 0).  **Function Name:**  Test\_PlaceMarker\_in\_Top\_Left\_Corner() |
|  | State: |  |
| IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ |  |  |
| testPos = new BoardPosition (0, 0) gb.placeMarker (testPos, player) |  |  |

public void placeMarker(BoardPosition testPos, char player)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3)  IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3  use whatsAtPos(testPos) for comparing the results of a cell in the board after a marker is placed.  State: | **Reason:**  This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in the last available position on the game board (1, 1).  **Function Name:**  Test\_PlaceMarker\_Last\_Marker\_on\_Board() |
| testPos = new BoardPosition (1, 1) gb.placeMarker (testPos, player) |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

public char whatsAtPos(BoardPosition testPos)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 whatsAtPos(testPos) = ‘ ’ | **Reason:**  This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the specified position.  **Function Name:**  Test\_WhatsAtPos\_Blank\_Board() |
|  | State: |  |
| IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ |  |  |
| testPos = new BoardPosition (2, 2) |  |  |

public char whatsAtPos(BoardPosition testPos)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 whatsAtPos(testPos) = ‘X’ | **Reason:**  This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the specified position even if it is the only marker on the board. |
|  | State: | **Function Name:**  Test\_WhatsAtPos\_One\_Marker\_Placed() |
| IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ |  |  |
| testPos = new BoardPosition (0, 0) |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  | X |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

public char whatsAtPos(BoardPosition testPos)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  | X |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 whatsAtPos(testPos) = ‘X’ | **Reason:**  This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the bottom right corner of the game board even if it is the only marker on the board. |
|  | State: | **Function Name:**  Test\_WhatsAtPos\_Bottom\_Right\_Corner() |
| IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ |  |  |
| testPos = new BoardPosition (2, 2) |  |  |

public char whatsAtPos(BoardPosition testPos)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 whatsAtPos(testPos) = ‘X’ | **Reason:**  This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the top right corner of the game board even if it is the only marker on the board. |
|  | State: | **Function Name:**  Test\_WhatsAtPos\_Top\_Right\_Corner() |
| IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ |  |  |
| testPos = new BoardPosition (0, 2) |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

public char whatsAtPos(BoardPosition testPos)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 | X |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 whatsAtPos(testPos) = ‘X’ | **Reason:**  This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the bottom left corner of the game board even if it is the only marker on the board. |
|  | State: | **Function Name:**  Test\_WhatsAtPos\_Bottom\_Left\_Corner() |
| IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ |  |  |
| testPos = new BoardPosition (2, 0) |  |  |

public char whatsAtPos(BoardPosition testPos)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 whatsAtPos(testPos) = ‘X’ | **Reason:**  This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the top left corner of the game board even if it is the only marker on the board. |
|  | State: | **Function Name:**  Test\_WhatsAtPos\_Top\_Left\_Corner() |
| IGameBoard gb = GameBoardFactory(3, 3, 3) player = ‘X’ |  |  |
| testPos = new BoardPosition (0, 0) |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  | X |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

default boolean checkSpace(BoardPosition pos)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 gb.checkSpace(pos) = true | **Reason:**  This test case is unique and distinct because it tests that the checkSpace function can successfully determine whether the position the user wants to place their marker is empty.  **Function Name:**  Test\_CheckSpace\_Empty\_Space() |
|  | State is unchanged |  |
| IGameBoard gb = GameBoardFactory(3, 3, 3) |  |  |
| testPos = new BoardPosition (0, 0) |  |  |

default boolean checkSpace(BoardPosition pos)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 gb.checkSpace(testPos) = false | **Reason:**  This test case is unique and distinct because it tests that the checkSpace function can successfully determine whether the position the user wants to place their marker is already taken (aka not empty).  **Function Name:**  Test\_CheckSpace\_Unavailable\_Space() |
| IGameBoard gb = GameBoardFactory(3, 3, 3) | State: |  |
| testPos = new BoardPosition (2, 2) |  |  |
| gb.placeMarker(testPos) |  |  |

default boolean checkSpace(BoardPosition pos)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 gb.checkSpace(pos) = false | **Reason:**  This test case is unique and distinct because it tests that the checkSpace function can successfully determine whether the position is out of the bounds of the array.  **Function Name:**  Test\_CheckSpace\_Out\_of\_Bounds() |
|  | State is unchanged |  |
| IGameBoard gb = GameBoardFactory(3, 3, 3) |  |  |
| testPos = new BoardPosition (3, 3) |  |  |

public boolean isPlayerAtPos(BoardPosition pos, char player)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Input:**  State: (number to win = 3) | **Output:**  getNumRows() = 3  getNumCols() = 3  getNumToWin() = 3 gb.isPlayerAtPos(testPos, player) = false | **Reason:**  This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player’s char passed into the function matches the char in position pos. |
| IGameBoard gb = GameBoardFactory(3, 3, 3) | State is unchanged | **Function Name:**  Test\_IsPlayerAtPos\_Empty\_Board() |
| testPos = new BoardPosition (1, 1) |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

public boolean isPlayerAtPos(BoardPosition pos, char player)

**Output:**

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.isPlayerAtPos(testPos, player) = true

State:

IGameBoard gb =

GameBoardFactory(3, 3, 3)

//Assume player = ‘X’

testPos = new BoardPosition (1, 1) gb.placeMarker(testPos, player)

**Input:**

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  | X |  |
| 2 |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player’s char passed into the function matches the char in position pos. In this case, they should match.

**Function Name:** Test\_IsPlayerAtPos\_with\_Player\_i n\_Pos()

public boolean isPlayerAtPos(BoardPosition pos, char player)

**Output:**

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.isPlayerAtPos(testPos, player) = false

State:

IGameBoard gb =

GameBoardFactory(3, 3, 3)

//Assume player = ‘X’

gb.placeMarker((0, 0), ‘A’)

gb.placeMarker((0, 1), ‘B’)

testPos = new BoardPosition (0, 2) gb.placeMarker(testPos, ‘C’)

**Input:**

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | A |  |  |
| 1 | B |  |  |
| 2 | C |  |  |

**Reason:**

This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player’s char passed into the function matches the char in position pos. In this case, they should not match.

**Function Name:** Test\_IsPlayerAtPos\_with\_Wrong\_Pl ayer\_in\_Pos()

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

public boolean isPlayerAtPos(BoardPosition pos, char player)

**Output:**

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.isPlayerAtPos(testPos, player) = true

State:

IGameBoard gb =

GameBoardFactory(3, 3, 3)

//Assume player = ‘X’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((0, 1), ‘X’)

testPos = new BoardPosition (0, 2) gb.placeMarker(testPos, player)

**Input:**

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X |  |  |
| 1 | X |  |  |
| 2 | X |  |  |

**Reason:**

This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player’s char passed into the function matches the char in position pos, even in the case where the entire row is marked by the same char.

**Function Name:**

Test\_IsPlayerAtPos\_True\_Row()

public boolean isPlayerAtPos(BoardPosition pos, char player)

**Output:**

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.isPlayerAtPos(testPos, player) = true

State:

IGameBoard gb =

GameBoardFactory(3, 3, 3)

//Assume player = ‘X’

testPos = new BoardPosition (0, 2) gb.placeMarker(testPos, player)

**Input:**

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  | X |  |
| 2 |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player’s char passed into the function matches the char in position pos, even in the case where the char to match is in the middle of the board.

**Function Name:**

Test\_IsPlayerAtPos\_Middle()

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

public boolean checkForDraw(BoardPosition lastPos)

**Output:**

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.checkForDraw(testPos) = false

State:

IGameBoard gb =

GameBoardFactory(3, 3, 3)

//Assume player = ‘O’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((0, 1), ‘O’)

gb.placeMarker((0, 2), ‘X’)

gb.placeMarker((2, 0), ‘O’)

gb.placeMarker((2, 1), ‘X’)

testPos = new BoardPosition (2, 2) gb.placeMarker(testPos, player)

**Input:**

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X | O | X |
| 1 |  |  |  |
| 2 | O | X | O |

**Reason:**

This test case is unique and distinct because it tests that the checkForDraw function can successfully determine if the board is full. In this case, checkForDraw should be false because there is an empty row.

**Function Name:**

Test\_CheckForDraw\_One\_Empty\_Row()

public boolean checkForDraw(BoardPosition lastPos)

**Output:**

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.checkForDraw(testPos) = false

State:

IGameBoard gb =

GameBoardFactory(3, 3, 3)

//Assume player = ‘O’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((1, 0), ‘O’)

gb.placeMarker((2, 0), ‘X’)

gb.placeMarker((0, 1), ‘O’)

gb.placeMarker((1, 1), ‘X’)

testPos = new BoardPosition (2, 1) gb.placeMarker(testPos, player)

**Input:**

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X | O |  |
| 1 | O | X |  |
| 2 | X | O |  |

**Reason:**

This test case is unique and distinct because it tests that the checkForDraw function can successfully determine if the board is full. In this case, checkForDraw should be false because there is an empty column.

**Function Name:**

Test\_CheckForDraw\_One\_Empty\_Col()

public boolean checkForDraw(BoardPosition lastPos)

## Input:

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

IGameBoard gb = GameBoardFactory(3, 3, 3)

//Assume player = ‘O’ gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((0, 1), ‘O’)

gb.placeMarker((0, 2), ‘X’)

gb.placeMarker((1, 0), ‘O’)

gb.placeMarker((1, 1), ‘X’)

gb.placeMarker((1, 2), ‘O’)

gb.placeMarker((2, 0), ‘X’)

testPos = new BoardPosition (2, 1) gb.placeMarker(testPos, player)

## Output:

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.checkForDraw(testPos) = false

State:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X | O |  |
| 1 | O | X |  |
| 2 | X | O |  |

## Reason:

This test case is unique and distinct because it tests that the checkForDraw function can successfully determine if the board is full. In this case, checkForDraw should be false because there is an empty space left.

## Function Name:

Test\_CheckForDraw\_One\_Empty\_Space()

public boolean checkForDraw(BoardPosition lastPos)

## Input:

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

IGameBoard gb = GameBoardFactory(3, 3, 3)

//Assume player = ‘O’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((0, 1), ‘X’)

gb.placeMarker((0, 2), ‘O’)

gb.placeMarker((1, 0), ‘O’)

gb.placeMarker((1, 1), ‘O’)

gb.placeMarker((1, 2), ‘X’)

gb.placeMarker((2, 0), ‘X’)

gb.placeMarker((2, 1), ‘X’)

testPos = new BoardPosition (2, 2) gb.placeMarker(testPos, player)

## Output:

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.checkForDraw(testPos) = true

State:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X | X | O |
| 1 | O | O | X |
| 2 | X | X | O |

## Reason:

This test case is unique and distinct because it tests that the checkForDraw function can successfully determine if the board is full. In this case, checkForDraw should be true because every space on the board is full and there was no winner.

## Function Name:

Test\_CheckForDraw\_Full\_Board()

default boolean checkDiagonalWin(BoardPosition lastPos)

## Input:

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

IGameBoard gb = GameBoardFactory(3, 3, 3)

//Assume player = ‘X’ gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((0, 1), ‘O’)

gb.placeMarker((0, 2), ‘X’)

gb.placeMarker((1, 0), ‘O’)

gb.placeMarker((1, 1), ‘X’)

gb.placeMarker((1, 2), ‘O’)

gb.placeMarker((2, 0), ‘X’)

gb.placeMarker((2, 1), ‘O’)

testPos = new BoardPosition (2, 2) gb.placeMarker(testPos, player)

## Output:

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.checkDiagonalWin(testPos) = true

State:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X | O | X |
| 1 | O | X | O |
| 2 | X | O | X |

## Reason:

This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from bottom right corner up to the top left corner.

## Function Name:

Test\_Diagonal\_Win\_Start

\_at\_Bottom\_Right()

default boolean checkDiagonalWin(BoardPosition lastPos)

## Input:

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

IGameBoard gb = GameBoardFactory(3, 3, 3)

//Assume player = ‘X’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((0, 1), ‘O’)

gb.placeMarker((0, 2), ‘X’)

gb.placeMarker((1, 0), ‘O’)

gb.placeMarker((1, 1), ‘X’)

gb.placeMarker((1, 2), ‘O’)

gb.placeMarker((2, 0), ‘X’)

gb.placeMarker((2, 1), ‘O’)

testPos = new BoardPosition (2, 2) gb.placeMarker(testPos, player)

## Output:

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.checkDiagonalWin(testPos) = true

State:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  | X |
| 1 |  | X |  |
| 2 | X |  |  |

## Reason:

This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top right corner down to bottom left corner.

## Function Name:

Test\_Diagonal\_Win\_Start

\_at\_Top\_Right()

default boolean checkDiagonalWin(BoardPosition lastPos)

## Input:

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

IGameBoard gb = GameBoardFactory(3, 3, 3)

//Assume player = ‘X’ gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((0, 1), ‘O’)

gb.placeMarker((0, 2), ‘X’)

gb.placeMarker((1, 0), ‘O’)

gb.placeMarker((1, 1), ‘X’)

gb.placeMarker((1, 2), ‘O’)

gb.placeMarker((2, 0), ‘X’)

gb.placeMarker((2, 1), ‘O’)

testPos = new BoardPosition (2, 2) gb.placeMarker(testPos, player)

## Output:

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.checkDiagonalWin(testPos) = true

State:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  | X |
| 1 |  | X |  |
| 2 | X |  |  |

## Reason:

This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from bottom left corner up to top right corner.

## Function Name:

Test\_Diagonal\_Win\_Start

\_at\_Bottom\_Left()

default boolean checkDiagonalWin(BoardPosition lastPos)

## Input:

State: (number to win = 3)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |

IGameBoard gb = GameBoardFactory(3, 3, 3)

//Assume player = ‘X’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((0, 1), ‘O’)

gb.placeMarker((0, 2), ‘X’)

gb.placeMarker((1, 0), ‘O’)

gb.placeMarker((1, 1), ‘X’)

gb.placeMarker((1, 2), ‘O’)

gb.placeMarker((2, 0), ‘X’)

gb.placeMarker((2, 1), ‘O’)

testPos = new BoardPosition (2, 2) gb.placeMarker(testPos, player)

## Output:

getNumRows() = 3

getNumCols() = 3

getNumToWin() = 3 gb.checkDiagonalWin(testPos) = true

State:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | X |  |  |
| 1 |  | X |  |
| 2 |  |  | X |

## Reason:

This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top left corner down to bottom right corner.

## Function Name:

Test\_Diagonal\_Win\_Start

\_at\_Top\_Left()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  |  |
| 1 |  |  | X |  |  |  |  |  |
| 2 |  |  |  | X |  |  |  |  |
| 3 |  |  |  |  | X |  |  |  |
| 4 |  |  |  |  |  | X |  |  |
| 5 |  |  |  |  |  |  | X |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

default boolean checkDiagonalWin(BoardPosition lastPos)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| **Input:** | **Output:** |
| State: (number to win = 5) | getNumRows() = 8  getNumCols() = 8  getNumToWin() = 5 gb.checkDiagonalWin(testPos) = true |
|  | State: |
| IGameBoard gb = GameBoardFactory(8, 8, 3)  //Assume player = ‘X’ |  |
| gb.placeMarker((1, 2), ‘X’)  gb.placeMarker((2, 3), ‘X’)  gb.placeMarker((3, 4), ‘X’)  gb.placeMarker((4, 5), ‘X’)  testPos = new BoardPosition (5, 6) gb.placeMarker(testPos, player) |  |

**Reason:**

This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top left corner down to bottom right corner.

**Function Name:**

Test\_Diagonal\_Win\_Middle\_of\_Larger\_Board()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkDiagonalWin(testPos) = false

State:

default boolean checkDiagonalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘X’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((1, 1), ‘X’)

gb.placeMarker((2, 2), ‘X’)

gb.placeMarker((3. 3), ‘O’)

testPos = new BoardPosition (4, 4) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | X |  |  |  |  |  |  |  |  |  |
| 1 |  | X |  |  |  |  |  |  |  |  |
| 2 |  |  | X |  |  |  |  |  |  |  |
| 3 |  |  |  | O |  |  |  |  |  |  |
| 4 |  |  |  |  | X |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top left corner down to bottom right corner but is cut short because it encounters a different character before reaching 5 characters that matched ‘X”.

**Function Name:** Test\_Diagonal\_Win\_Three\_ Out\_of\_Five\_In\_a\_Row()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkDiagonalWin(testPos) = false

State:

default boolean checkDiagonalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘O’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((1, 1), ‘X’)

gb.placeMarker((2, 2), ‘X’)

gb.placeMarker((3. 3), ‘X’)

testPos = new BoardPosition (4, 4) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | X |  |  |  |  |  |  |  |  |  |
| 1 |  | X |  |  |  |  |  |  |  |  |
| 2 |  |  | X |  |  |  |  |  |  |  |
| 3 |  |  |  | X |  |  |  |  |  |  |
| 4 |  |  |  |  | O |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top left corner down to bottom right corner but is cut short because it encounters a different character where the 5th matching character would have to be located.

**Function Name:** Test\_Diagonal\_Win\_Four\_O ut\_of\_Five\_In\_a\_Row()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkVerticalWin(testPos) = false

State:

default boolean checkVerticalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘O’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((1, 0), ‘X’)

gb.placeMarker((2, 0), ‘X’)

gb.placeMarker((3. 0), ‘X’)

testPos = new BoardPosition (4, 0) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | X |  |  |  |  |  |  |  |  |  |
| 1 | X |  |  |  |  |  |  |  |  |  |
| 2 | X |  |  |  |  |  |  |  |  |  |
| 3 | X |  |  |  |  |  |  |  |  |  |
| 4 | O |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkVerticalWin function is accurately reading the first column on the board since it is a bound.

**Function Name:** Test\_CheckVerticalWin\_Far\_ Left\_Column\_One\_Short()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkVerticalWin(testPos) = false

State:

default boolean checkVerticalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘O’

gb.placeMarker((0, 9), ‘X’)

gb.placeMarker((1, 9), ‘X’)

gb.placeMarker((2, 9), ‘X’)

gb.placeMarker((3. 9), ‘X’)

testPos = new BoardPosition (4, 9) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  | X |
| 1 |  |  |  |  |  |  |  |  |  | X |
| 2 |  |  |  |  |  |  |  |  |  | X |
| 3 |  |  |  |  |  |  |  |  |  | X |
| 4 |  |  |  |  |  |  |  |  |  | O |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkVerticalWin function is accurately reading the last column on the board since it is a bound.

**Function Name:** Test\_CheckVerticalWin\_Far\_ Right\_Column\_One\_Short()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkVerticalWin(testPos) = true

State:

default boolean checkVerticalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘X’

gb.placeMarker((0, 5), ‘X’)

gb.placeMarker((0, 0), ‘O’)

gb.placeMarker((1, 5), ‘X’)

gb.placeMarker((1, 1), ‘O’)

gb.placeMarker((2, 5), ‘X’)

gb.placeMarker((3, 4), ‘O’)

gb.placeMarker((3, 5), ‘X’)

gb.placeMarker((2, 7), ‘O’)

testPos = new BoardPosition (4, 5) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | O |  |  |  |  | X |  |  |  |  |
| 1 |  | O |  |  |  | X |  |  |  |  |
| 2 |  |  |  |  |  | X |  | O |  |  |
| 3 |  |  |  |  | O | X |  |  |  |  |
| 4 |  |  |  |  |  | X |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkVerticalWin function is accurately detecting five characters in a row.

**Function Name:**

Test\_CheckVerticalWin\_Middle

\_Column\_Five\_In\_a\_Row()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkVerticalWin(testPos) = false

State:

default boolean checkVerticalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘X’

gb.placeMarker((0, 8), ‘X’)

gb.placeMarker((1, 8), ‘X’)

gb.placeMarker((2, 8), ‘O’)

gb.placeMarker((3, 8), ‘X’)

testPos = new BoardPosition (4, 8) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  | X |  |
| 1 |  |  |  |  |  |  |  |  | X |  |
| 2 |  |  |  |  |  |  |  |  | O |  |
| 3 |  |  |  |  |  |  |  |  | X |  |
| 4 |  |  |  |  |  |  |  |  | X |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkVerticalWin function stops scanning for matching characters when it encounters one that does not match.

**Function Name:**

Test\_CheckVerticalWin\_Two\_In

\_a\_Row\_Separated\_by\_One()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkHorizontalWin(testPos) = false

State:

default boolean checkHorizontalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘O’

gb.placeMarker((0, 0), ‘X’)

gb.placeMarker((2, 2), ‘O’)

gb.placeMarker((0, 1), ‘X’)

gb.placeMarker((2, 4), ‘O’)

gb.placeMarker((0, 2), ‘X’)

gb.placeMarker((7, 7), ‘O’)

gb.placeMarker((0, 3), ‘X’)

gb.placeMarker((9, 5), ‘O’)

testPos = new BoardPosition (0, 4) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | X | X | X | X | O |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  | O |  | O |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  | O |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  | O |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkHorizontalWin function is correctly detecting characters in the first row.

**Function Name:** Test\_CheckHorizontalWin\_Top\_ Row\_One\_Short()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkHorizontalWin(testPos) = false

State:

default boolean checkHorizontalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘O’

gb.placeMarker((9, 0), ‘X’)

gb.placeMarker((2, 2), ‘O’)

gb.placeMarker((9, 1), ‘X’)

gb.placeMarker((2, 4), ‘O’)

gb.placeMarker((9, 2), ‘X’)

gb.placeMarker((7, 7), ‘O’)

gb.placeMarker((9, 3), ‘X’)

testPos = new BoardPosition (9, 4) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  | O |  | O |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  | O |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 | X | X | X | X | O |  |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkHorizontalWin function is correctly detecting characters in the last row.

**Function Name:** Test\_CheckHorizontalWin\_Bott om\_Row\_One\_Short()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkHorizontalWin(testPos) = true

State:

default boolean checkHorizontalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘X’

gb.placeMarker((4, 3), X’)

gb.placeMarker((4, 2), ‘O’)

gb.placeMarker((4, 4), ‘X’)

gb.placeMarker((5, 7), ‘O’)

gb.placeMarker((4. 5), ‘X’)

gb.placeMarker((7, 7), ‘O’)

gb.placeMarker((4, 6), ‘X’)

gb.placeMarker((9, 5), ‘O’)

testPos = new BoardPosition (4, 7) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  | O | X | X | X | X | X |  |  |
| 5 |  |  |  |  |  |  |  | O |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  | O |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  | O |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkHorizontalWin function is correctly detecting characters in a row in the middle of the board.

**Function Name:** Test\_CheckHorizontalWin\_Midd le\_5\_in\_a\_Row()

**Output:**

getNumRows() = 10

getNumCols() = 10

getNumToWin() = 5 gb.checkHorizontalWin(testPos) = false

State:

default boolean checkHorizontalWin(BoardPosition lastPos)

**Input:**

State: (number to win = 5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

IGameBoard gb = GameBoardFactory(10, 10, 5)

//Assume player = ‘X’

gb.placeMarker((5, 0), X’)

gb.placeMarker((5, 1), ‘X’)

gb.placeMarker((5, 2), ‘O’)

gb.placeMarker((5, 3), ‘X’)

testPos = new BoardPosition (5, 4) gb.placeMarker(testPos, player)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 | X | X | O | X | X |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |

**Reason:**

This test case is unique and distinct because it tests that the checkHorizontalWin function stops scanning for matching characters when it encounters one that does not match.

**Function Name:** Test\_CheckHorizontalWin\_Two\_ In\_a\_Row\_Separated\_by\_One()