|  |
| --- |
| Jeremy Holloway |
|  |

|  |  |
| --- | --- |
| 11/18/2018 | Technical Report |

|  |
| --- |
| Prepared for Kevin Plis  Prepared by Jeremy Holloway  CPSC-2150-001  Fall 2018 |

Table of Contents

[1. Requirement Analysis 2](#_Toc529983750)

[2. Design 4](#_Toc529983751)

[3. Testing 14](#_Toc529983752)

[4. Deployment 18](#_Toc529983753)

|  |
| --- |
|  |

# 1. Requirement Analysis

**User Story:**

* As a user, I can choose which column to place a token to win the game.
* As a user, I will be notified when it is my turn so I can place a marker.
* As a user, I will be notified of a win so I can rub it in my opponents face.
* As a user, I will be notified if I make an incorrect move so I may take another turn.
* As a user, I will be able to choose how many players can play the game.
* As a user, I will be able to select what character I want to play with.
* As a user, I will be notified if my selected token is already in use.
* As a user, I will be able to choose if I want to play a memory efficient game or a fast game.
* As a user, I will be able to run a functionality test to ensure I can play a game.

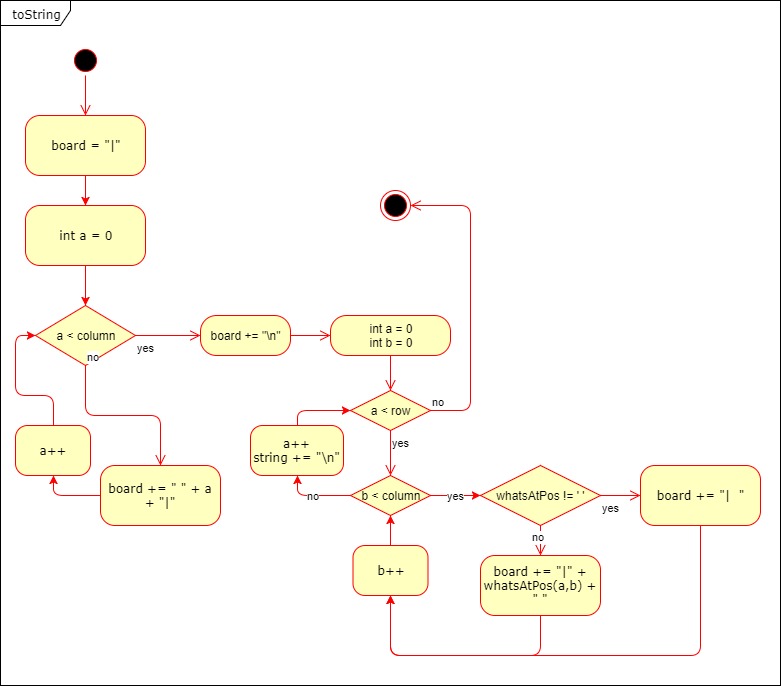
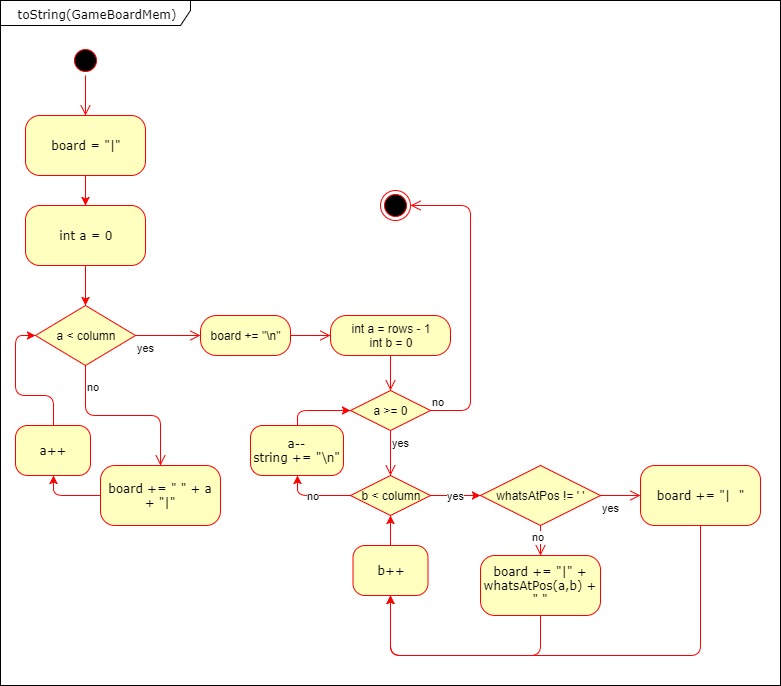
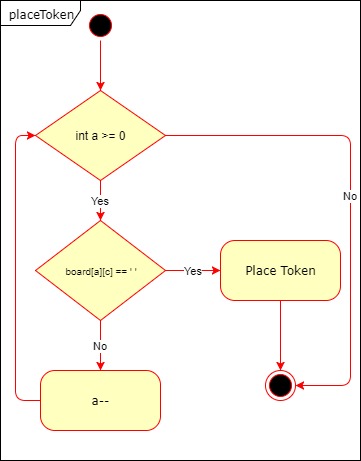
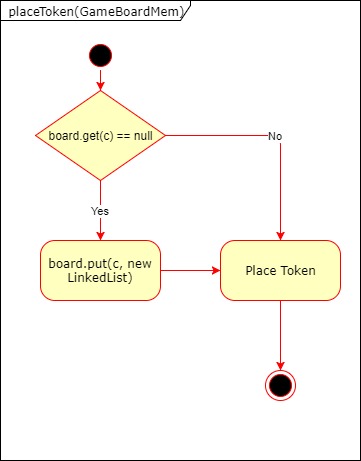
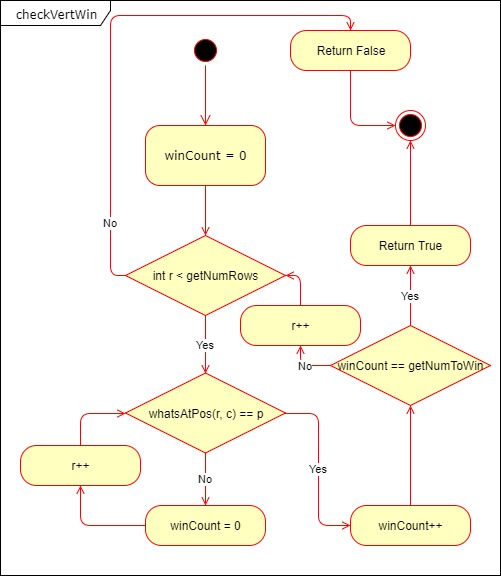
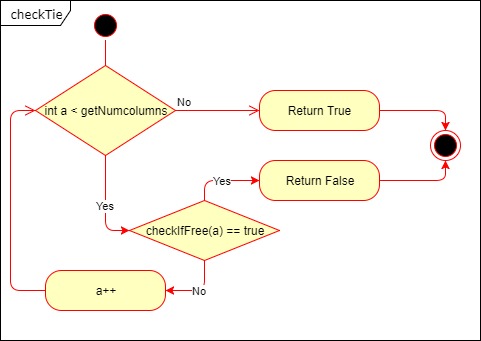
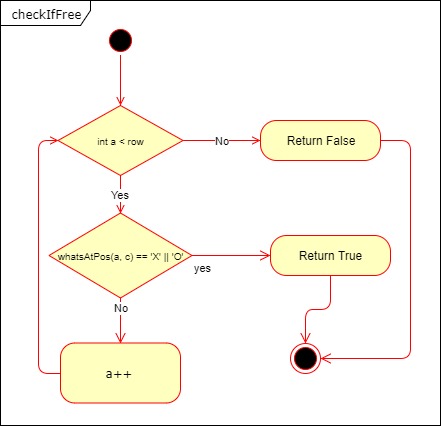
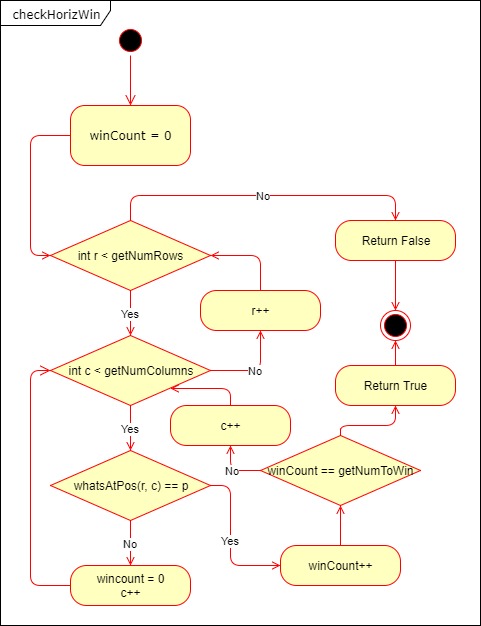
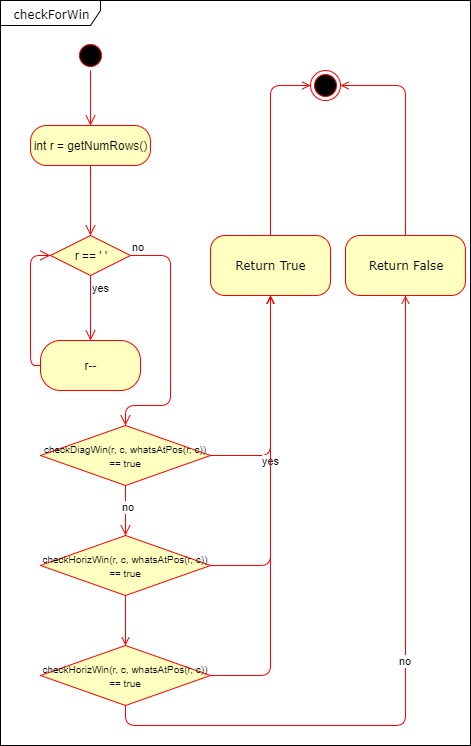
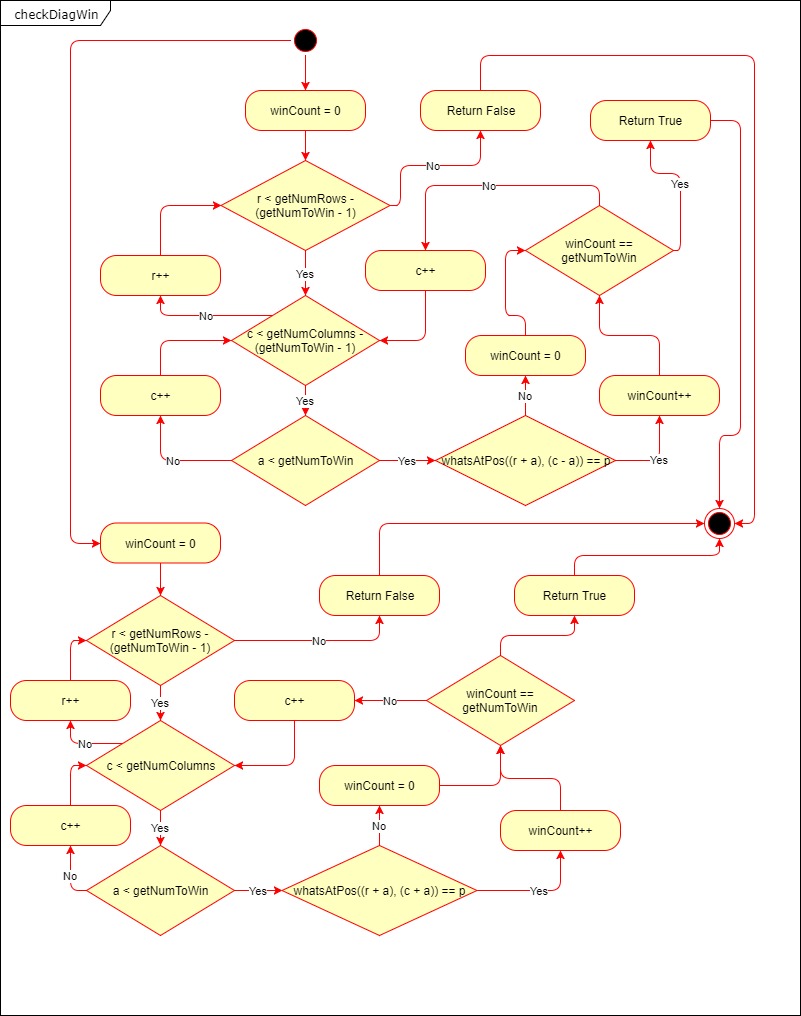
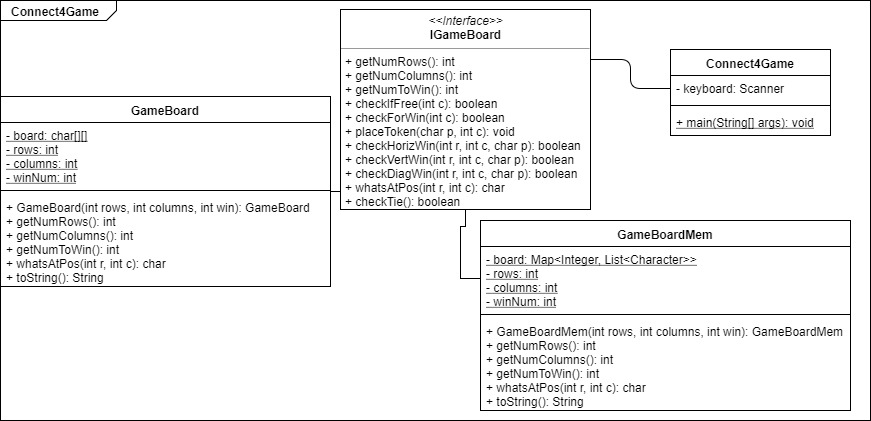
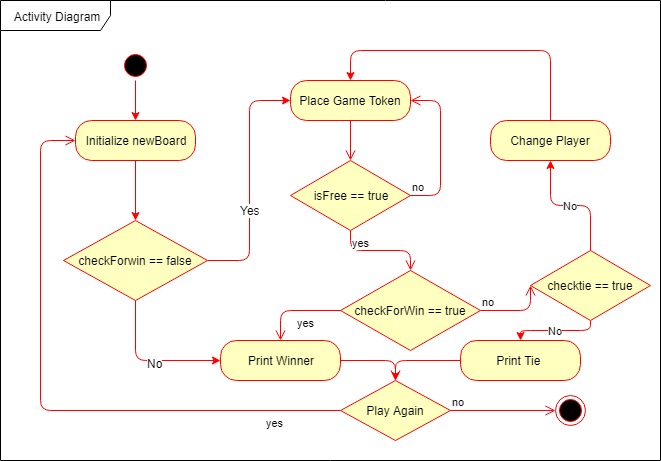
**Functional Requirements:**

* A user will be able to choose which column to place their token.
* A user will be able to choose to play again after winning, losing, or drawing.
* A user will be able to choose the number of players.
* A user will be able to choose which version of the game to play.
* A user will be able to choose to run a functionality test.

**Non-Functional Requirements:**

* The program will automatically determine if a winning move is made after each turn or if a draw has occurred.
* The program will be implemented using a 2D character array or a hash map to represent the gameboard.
* The program will have an instant response time.
* The program is modular. If the client desires to remove certain functions, they may be commented out. These methods will work on any system with JDK8 installed.
* The program is written in Java. The system must have JDK8 installed.
* The program was designed for a Unix based environment.
* This program will use a makefile with the commands; make, make run, make test, make runtest, and make clean.
* All interfaces between the user and the app will be displayed via the terminal window.
* The program will be able to run a functionality test on either game configuration.

# 2. Design



# 3. Testing

The following test cases can be found in TestGameBoard.java

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  State of the board is unchanged | Reason:  This case is unique because it creates the smallest possible size board.  Function:  test\_min\_board |
| Input:  rows = 100  columns = 100  number to win = 25   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   \*each square represents 20 squares\* | Output:  100x100 game board  State of the board is unchanged | Reason:  This case is unique because it creates the largest possible size board.  Function:  test\_max\_board |
| Input:  rows = 45  columns = 54  number to win = 6     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  |   \*each square represents 9 squares\* | Output:  45x54 game board  State of the board is unchanged | Reason:  This case is unique because it creates an irregular size board.  Function:  test\_nonsquare\_board |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkIfFree == true  State of the board is unchanged | Reason:  This case is unique because it verifies checkIfFree will return true on columns with free space.  Function:  test\_free\_empty |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkIfFree == false   |  |  |  | | --- | --- | --- | |  | X |  | |  | X |  | |  | X |  | | Reason:  This case is unique because it verifies checkIfFree will return false on a column with no free space.  Function:  test\_full\_column |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkIfFree == false   |  |  |  | | --- | --- | --- | | X | X | X | | X | X | X | | X | X | X | | Reason:  This case is unique because it verifies checkIfFree will return false on all columns with no free space.  Function:  test\_full\_board |
| Input:  rows = 4  columns = 4  number to win = 3   |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | | Output:  4x4 game board  checkHorizWin == true   |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | | X | X | X |  | | Reason:  This case is unique because it checks for a horizontal win starting at the left most edge of the board.  Function:  test\_leftH\_win |
| Input:  rows = 4  columns = 4  number to win = 3   |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | | Output:  4x4 game board  checkHorizWin == true   |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | X | X | X | | Reason:  This case is unique because it checks for a horizontal win starting at the right most edge of the board.  Function:  test\_rightH\_win |
| Input:  rows = 5  columns = 5  number to win = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkHorizWin == true   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  | X | X | X |  | | Reason:  This case is unique because it checks for a horizontal win starting at the middle most position of the board.  Function:  test\_midH\_win |
| Input:  rows = 5  columns = 5  number to win = 4   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkHorizWin == true   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | O |  |  |  | O | | X | X | X | X | O | | X | X | O | X | O | | Reason:  This case is unique because it checks for a win under a normal use situation with two players.  Function:  test\_normH\_win |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkHorizWin == false   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | | X |  | X | | Reason:  This case is unique because it checks for a false return if there is not the specified amount in a row.  Function:  test\_noH\_win |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkVertWin == true   |  |  |  | | --- | --- | --- | | X |  |  | | X |  |  | | X |  |  | | Reason:  This case is unique because it verifies checkVertWin will return true on the left most column.  Function:  test\_leftV |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkVertWin == true   |  |  |  | | --- | --- | --- | |  |  | X | |  |  | X | |  |  | X | | Reason:  This case is unique because it verifies checkVertWin will return true on the right most column.  Function:  test\_rightV |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkVertWin == false   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Reason:  This case is unique because it verifies checkVertWin will return false on empty column.  Function:  test\_noWinV |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkVertWin == false   |  |  |  | | --- | --- | --- | |  |  | X | |  |  | O | |  |  | X | | Reason:  This case is unique because it verifies checkVertWin will return false on an incorrect sequence of characters in a column.  Function:  test\_noWin\_XandO |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  rows = 5  columns = 5  number to win = 4   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkHorizWin == true   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | | O |  |  |  | O | | X | X | X | X | O | | X | X | O | X | O | | Reason:  This case is unique because it checks for a win under a normal use situation with two players.  Function:  test\_normH\_win |
| Input:  rows = 5  columns = 5  number to win = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkDiagWin == true   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  | |  | X | X |  |  | | X | X | X |  |  | | Reason:  This case is unique because it checks for a win on the bottom left edge of the gameboard.  Function:  test\_diag\_down\_left |
| Input:  rows = 5  columns = 5  number to win = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkDiagWin == false   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  | |  | X | X |  |  | | O | X | X |  |  | | Reason:  This case is unique because it checks for a correct sequence on the bottom left edge of the gameboard.  Function:  test\_noWin\_diag\_down\_left |
| Input:  rows = 5  columns = 5  number to win = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkDiagWin == true   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  | |  |  | X | X |  | |  |  | X | X | X | | Reason:  This case is unique because it checks for a win on the bottom right edge of the gameboard.  Function:  test\_diag\_down\_right |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  rows = 5  columns = 5  number to win = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkDiagWin == false   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  | X |  |  | |  |  | X | X |  | |  |  | X | X | O | | Reason:  This case is unique because it checks for a correct sequence on the bottom right edge of the gameboard.  Function:  test\_noWin\_diag\_down\_right |
| Input:  rows = 5  columns = 5  number to win = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkDiagWin == true   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  | X | |  |  |  | X | X | |  |  | X | X | X | |  |  | X | X | X | |  |  | X | X | X | | Reason:  This case is unique because it checks for a win on the top right edge of the gameboard.  Function:  test\_diag\_up\_right |
| Input:  rows = 5  columns = 5  number to win = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkDiagWin == false   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  | X | |  |  |  | X | X | |  |  | O | X | X | |  |  | O | X | X | |  |  | O | X | X | | Reason:  This case is unique because it checks for a correct sequence on the upper right edge of the gameboard.  Function:  test\_noWin\_diag\_up\_right |
| Input:  rows = 5  columns = 5  number to win = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkDiagWin == true   |  |  |  |  |  | | --- | --- | --- | --- | --- | | X |  |  |  |  | | X | X |  |  |  | | X | X | X |  |  | | X | X | X |  |  | | X | X | X |  |  | | Reason:  This case is unique because it checks for a win on the top left edge of the gameboard.  Function:  test\_diag\_up\_left |
| Input:  rows = 5  columns = 5  number to win = 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | Output:  5x5 game board  checkDiagWin == false   |  |  |  |  |  | | --- | --- | --- | --- | --- | | X |  |  |  |  | | X | X |  |  |  | | X | X | O |  |  | | X | X | O |  |  | | X | X | O |  |  | | Reason:  This case is unique because it checks for a correct sequence on the top left edge of the gameboard.  Function:  test\_noWin\_diag\_up\_left |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkTie == false   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Reason:  This case is unique because it verifies there are empty positions on the board.  Function:  test\_tie\_empty |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkTie == false   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | | X | X | X | | Reason:  This case is unique because it returns false is someone has won the game with a horizontal.  Function:  test\_tie\_horizW |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkTie == false   |  |  |  | | --- | --- | --- | | X |  |  | | X |  |  | | X |  |  | | Reason:  This case is unique because it returns false is someone has won the game with a vertical.  Function:  test\_tie\_vertW |
| Input:  rows = 3  columns = 3  number to win = 3   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | Output:  3x3 game board  checkTie == true   |  |  |  | | --- | --- | --- | | X | O | X | | O | R | O | | X | O | X | | Reason:  This case is unique because it returns true when no one has won the game and there are no moves left.  Function:  test\_tie |

1. Case: test\_pos\_empty
   1. Input: rows = 3, columns = 3, number to win = 3
   2. Output: blank 3x3 board
      1. This test will ensure whatsAtPos will return a blank if nothing has been placed.
2. Case: test\_pos\_left
   1. Input: rows = 3, columns = 3, number to win = 3
   2. Output: 3x3 board with one char in the left most column
      1. This test will ensure whatsAtPos will return a char in a left edge case.
3. Case: test\_pos\_right
   1. Input: rows = 3, columns = 3, number to win = 3
   2. Output: 3x3 board with one char in the right most column
      1. This test will ensure whatsAtPos will return a char in a right edge case.
4. Case: test\_pos\_full
   1. Input: rows = 5, columns = 5, number to win = 3
   2. Output: 5x5 board with every position filled with the same char
      1. This test will ensure whatsAtPos will return a char in every board position.
5. Case: test\_pos\_place\_all
   1. Input: rows = 5, columns = 5, number to win = 3
   2. Output: 5x5 board filled with a different char in every position
      1. This test will ensure whatsAtPos will return a variety of player tokens.
6. Case: test\_pos\_left\_uneven
   1. Input: rows = 3, columns = 5, number to win = 3
   2. Output: 3x5 board with one char in the left most column
      1. This test will ensure whatsAtPos will return a left edge case on a non-square board.
7. Case: test\_pos\_right\_uneven
   1. Input: rows = 3, columns = 5, number to win = 3
   2. Output: 3x5 board with one char in the right most column
      1. This test will ensure whatsAtPos will return a right edge case on a non-square board.
8. Case: test\_token\_left
   1. Input: rows = 3, columns = 3, number to win = 3
   2. Output: 3x3 board with one char in the left most column
      1. This test ensures placeToken is placing the chars correctly in left edge cases.
9. Case: test\_token\_right
   1. Input: rows = 3, columns = 3, number to win = 3
   2. Output: 3x3 board with one char in the right most column
      1. This test ensures placeToken is placing the chars correctly in right edge cases.
10. Case: test\_token\_middle
    1. Input: rows = 3, columns = 3, number to win = 3
    2. Output: 3x3 board with one char in the left most column
       1. This test ensures placeToken is placing the chars correctly in left edge cases.
11. Case: test\_token\_two
    1. Input: rows = 3, columns = 3, number to win = 3
    2. Output: 3x3 board with two chars in the middle most column
       1. This test ensures placeToken is placing the chars correctly in the same row.
12. Case: test\_token\_top\_left
    1. Input: rows = 3, columns = 3, number to win = 3
    2. Output: 3x3 board with the left most column full of chars
       1. This test ensures placeToken is placing the chars correctly in top edge cases.

# 4. Deployment

# 

This program will include a make file which can be used for three operations make, make run, make clean. The makefile must be in the same folder as cpsc2150 and not inside of it or any other subfolder.

**make:**

Expected output:

javac cpsc2150/connectX/Connect4Game.java

**make run:**

Expected output:

java cpsc2150.connectX.Connect4Game

\*\*start of game\*\*

**make test:**

Expected output:

javac -cp .:/usr/share/java/junit4.jar cpsc2150/connect/TestIGameBoard.java

**make runtest:**

Expected output:

Java -cp .:/usr/share/java/junit4.jar org.junit.runner.JUnitCore cpsc2150.connectX.TestIGameBoard

**make clean:**

Expected output:

rm -f \*.class