Testing

In-house:

I tested numerous times trying to make the program crash. Many times it crashed due to an array being out of bounds. This was often due to a mistake in operation signs (<,<=,=>,>,+,-).

Test 1: Board Settings check

I attempted to change the board settings to see which ones would work. This was a test of the settings part of the main method and also the displayboard method. It also had to deal with the static board array and board width/length variables. The program would crash if I assigned any dimensions that were apart by more than 3 or so (like 3 by 7). The program's column indicator and row indicator were out of place. This problem was caused by the fact that I set the board width as the column amount and board length as the row amount. This caused the settings to cause errors and mix up row/column indicators. After fixing this issue, board dimensions of 9 caused errors. This was later fixed by making the program go through the array 1 less than the amount of elements it had (a common mistake).

All errors related to the board settings were later fixed. The starting pieces will now always appear in the middle of the board or leaning towards the top left (if dimensions are odd numbers). Success on this part as well.

Test 2: Directions check

A major problem occurred while I was trying to make the program check for left and top directions. This problem happened in the validationcheck2 method; right and bottom checks worked and the program did the "connection" and converted all pieces in between the 2 friendly pieces to friendly pieces. However, the left and top checks did not do anything.

Later, I found that this problem was caused by the fact that I confused the row variable. Row actually goes down as you move up the board (I thought the opposite). Therefore I had to correct and change all signs relating to my row confusion.

Loops also caused errors when the program performed an action (in changepieces method) on the board. To fix this, instead of using i++ loops I used i- - loops as shown below.

Left (i- -loop): for(int i=column;i>column-checkfurtherL;i--){

Right(i- - loop): for(int i=column;i<column+checkfurtherR;i++){

Top also uses i- - loops and bottom uses i++ loops as for top you are moving down a row.

Diagonals were surprisingly not a problem for me I immediately got the idea of using another array (dynamicboard) to “mark” the areas that need to be changed in case a change can occur in a direction.

Test 3: Validation check

I attempted to test if the end conditions could be met and if the validations worked. Validation did have a problem. One could sometimes place a piece on a non “-“ (blank space) unit of the board. This was caused by the fact that validationcheck2 occurred even if the first one turned out false. Thus, validation still became true, omitting the first check. To fix this, I made so that the 2nd check would only occur if the first turned out true.

Boardavailability method (a method which helps the program determine if a turn has any moves or not) was another obstacle. The game will skip moves after the first turn. This was caused by the fact that boardavailability became true on the first turn and remained true. There no code that would reset its value to false. This was easily fixed.

Test 4: Tutorial check

Problems revolving around the fact that System.in.read(); did not work. To fix this, I just needed to add extra lines of it.

User Field Testing:

I don’t really understand what this type of testing means (should’ve asked sooner), but this is my best guess.

Many simulation tests were performed and I wasn’t the only one who tested the program in a *real-life* situation. The game was played by and between many people after its completion.

Test (after-completion)

Test attempted: Player vs. AI (tested by a friend in school environment)

Results: No problems were found. User may have been dissatisfied by pain of entering rows and columns instead of simply clicking the board.

Test (before-completion)

During program creation, user field testing simulations have also contributed to the fix of some of the factors described above at in-house testing.

This test was not mentioned in the in-house testing section. It is a small error I found while running user field simulation tests, documented here.

Test attempted: AI vs. AI (tested by me)

Results: One problem found. Program auto-skips on first turn (it should not) and crashes immediately after.

Fix: A value in the nomovecheck method was set to “y” when it was supposed to be set as “o” (for some reason, I inputted y instead of o due to the common conception of x and y being variables).