Project 4 Testing Report
Noah Leibowitz

1. Testing the Main Method

The first part to test is whether the main method is properly launching the program and passing the command line arguments into the start() method. The user can set the dimensions of their board (rows and columns) and the amount needed to win. Most of these tests were done in the interactions pane by typing “java Gomoku 6 12 15” or whatever arguments were used.

- First click the “run” button and ensure that a 19x19 grid launches and that 5 in a row are needed to win (This is the default).
- For the next test, enter one argument into the interactions pane, this argument would tell the hasFiveInLine() method how many in a row are needed to win. Additionally, a default 19x19 grid size will be used.
- Next is to test when the single argument entered is a number less than 5. Since 5 is the minimum needed to win, the message “The length to win must be at least 5 in a row.” should print to the console.
- Test when two arguments are used. This sets the dimensions of the board, while the number to win is default to 5.
- Test when two arguments are used and the dimensions are less than 5x5. The message "The size of the board must be at least 5x5." should print to the console.
- Test when three arguments are used and the first argument is less than the dimensions of the board. This should initialize the board appropriately.
- Test when three arguments are used, but the first argument (the amount needed to win) is larger than the dimensions of the board. In this case, the message “Board size and length to win must be at least 5.” prints to the console.
- Test when three arguments are used, but the first argument is less than 5 (the minimum amount needed to win). The message “Board size and length to win must be at least 5.” prints to the console.
- Test when the user enters something other than ints. The program should catch a NumberFormatException and the message “Invalid input type used. Please enter valid integers or use the default board size and length to win." prints to the console.

To simulate the user input through the command line arguments I created a class called GomokuSample which contained four different constructors. One constructor takes three inputs (the amount needed to win, the number of rows, the number of columns), another constructor takes two inputs (the number of rows, the number of columns), another takes one input (the

amount needed to win) and another is default and takes no inputs. This enabled me to test the logic, independent of the GUI components, behind the Gomoku game.

2. Testing the Getter/Setter Methods

The next part to test is all of the getter/setter methods that are used by several helper methods to access private fields. The only set of getter/setter methods not in the JUnit class were getMessage() and setMessage() as they involve interacting with the GUI. For simplicity in testing, the helper methods were made public and fields remained private. The following tests were performed in the JUnit class GomokuTester:

- getRows()
- getCols()
- isBlackTurn()
- switchTurn()
- isGameOver()
- setGameOver()
- getLengthToWin()
- getBlackCountFourFour()
- getBlackCountThreeThree()
- getWhiteCountFourFour()
- getWhiteCountThreeThree()
- setBlackCountFourFour()
- setBlackCountThreeThree()
- setWhiteCountFourFour()
- setWhiteCountThreeThree()

3. Testing Logic and Helper Methods

The next part to test is all of the helper methods that don't involve any of the components. Since so many helper methods utilized GUI components, I created a GomokuSample() class that takes a two dimensional array of ints instead of Buttons and specifically implements the logic of the game. Additionally, the color black was set to the int 1 and the color white was set to the int 2. My violatesFourFour() and violatesThreeThree() methods seem to be implemented correctly in the GomokuSample class, but there are slight issues with their functionality when playing on the actual GUI. When playing the game on the GUI board, the four-four and three-three rules are checked and I used print statements to ensure that the logic is being checked. However, while a move isn't allowed if it violates these rules, if I then go to place the piece down again it works because the count variables are reset. Despite this,

I was still able to test these rules and the other logic elements of the game by playing the game itself and using the helper class GomokuSample. The following tests were implemented:

- Ensure a move is valid before the piece is placed.
- Testing where the length to win is 5 and exactly 5 pieces are placed in a row.
- Testing where the length to win is 7 and the user sets the board size to non default dimensions.
- Testing where the length to win is 8 and the default board size is used.
- Ensuring that if there are more elements in a row than the length to win that the game continues.
- Testing for wins in the vertical direction.
- Testing for wins in the diagonal direction.
- Testing when the color alternates when placing pieces in a row (in any direction).
- Testing the Four-Four rule logic of the game looking for two or more groups of 4 created by the placement of a piece.
- Testing the Three-Three rule logic of the game looking for two or more groups of 3 with empty ends.

4. Testing of GUI Components

In order to test the components of the GUI, I used the debugger in DrJava as well as tested playing the game myself. I ensured that the layout of the grid was appropriate, that the correct coloring was being used, and that the appropriate message was being displayed at the bottom. It is important to ensure that the correct grid size is being used based on the user inputs in the command line arguments. For example, if "java Gomoku 15 16" was entered into the interactions pane, a 15x16 grid should have appeared.

Furthermore, I ensured that every button is clickable and that when a player wins, all of the grid buttons become unclickable except for the play again button which forces the user to quit or start a new game. Additionally, a welcome message is displayed at the bottom of the board. When a player wins that message switches by informing the players who won. If they click the "Play Again!" button, that message is reset to the welcome message. Testing this by manually playing the game was necessary to make sure that the game board communicates to the user the current state of the game.

Through the testing of user input, field initialization with getter and setter methods, game logic with a GomokuSample class, and GUI components the Gomoku game was thoroughly examined to ensure its correctness, visual appeal, and clear communication with the user.