Nathan Nguyen

12/8/19

Harold Connamacher

Programming Project 5 - Testing Document

1. Method 1 - numberInLine
   1. Testing general integers in line for all directions
      1. For this method, as I was constructing my project and implementing the various algorithms into my program, I found it necessary to ensure that all of my directions were working properly. **There were 8 different conditionals that needed to be tested within this method.** As such, I created 8 different general unit tests that ensured all of the conditionals for my directions were being executed and counting the correct number of equal integers in a straight line.
   2. Testing 0, 1, and many
      1. For this method, **I also found it necessary to test for 0, 1, and many integers that followed a specified element in a straight line in a particular direction**. As such, I tested these cases using the *“up”* direction and the integer *3* to ensure I could expect and would obtain the proper number of equal integers in a straight line as I changed my tests from 0, 1, and many equal integers following the specified integer while minimizing room for any error or interference.
   3. Testing first, middle, and last
      1. For this method, **I also found it necessary to test for an integer that breaks the line at the first, middle, and last position of the straight line.** As such, I again tested these cases using the *“up”* direction and the integer *3* to ensure that I could expect and would obtain the correct number of equal integers in a straight line had I inputted boards given the above scenarios.
2. Method 2 - isOpen
   1. Testing general is open for all directions
      1. For this method, as I was constructing my project and implementing the various algorithms into my program, I found it necessary to ensure that all of my directions were working properly. **There were 8 conditionals within this method that needed to be tested for both cases, giving a total of 16 unit tests.** As such, I provided scenarios in which the method would and would not find an empty space in each of the 8 different directions that could be inputted into the method.
   2. Testing 0 and 1
      1. For this method, **I also found it necessary to test for 0 and 1 empty spaces in a particular direction of a straight line.** As such, I tested these cases using the “up” direction and the integer *3* to ensure that I could expect and would obtain the correct boolean value given 0 empty spaces and 1 empty space in a straight line of equal integers. **Note, it was not necessary for me to test for many in this context because that test is already covered from my test 1, and I would receive the same boolean regardless.**
   3. Testing first, middle, and last
      1. For this method, **I also found it necessary to test for an empty space that occurs in the first, middle, and last position of a straight line of equal integers in a particular direction.** As such, I tested these cases using the “up” direction and the integer *3* to ensure that I could expect and would obtain the correct boolean values if I inserted the non-equal integer in the first, middle, and last positions of a straight line of equal integers.
3. (Helper) Method 3 - threeThree algorithm
   1. General verifying all 8 directions
      1. For this method, it was necessary to ensure that all 8 directions causing three in a row would properly count towards the algorithm of executing a threeThree scenario. **There were 8 different conditionals that needed to be tested within this method.** As such, I created 8 different unit tests that would allow me to ensure that a threeThree scenario would execute given a particular direction with 3 equal integers in a straight line with ends of the group of three being empty. I conducted this test by first ensuring that the *“up”* direction would count towards the algorithm of executing the threeThree scenario, and I mixed this with the 7 remaining directions to ensure that I could expect and would obtain the correct boolean value of the executed method given using all 8 of the different scenarios.
   2. Verifying different possible threeThree scenarios by testing 0, 1, 2, and more than 2
      1. For this method, **I found it necessary to test for 0, 1, 2, and more than 2 directions that would have three straight equal integers with ends of the groups of three being empty.** For this test, the primary element that I used to test for a straight line was the integer *4*. As such, I tested the scenarios in which there were no directions that caused a three in a row scenario, a scenario in which there were three integers in a row in the *“up”* direction, a scenario in which there were three integers in a row scenario in both the *“up”* and the *“right”* directions (causes threeThree scenario), and a scenario in which there were three integers in a row in the *“down”*, *“left”*, and *“down”* direction (also causes threeThree scenario).
4. (Helper) Method 4 - fourFour algorithm
   1. General verifying all 8 directions
      1. For this method, it was necessary to ensure that all 8 directions causing four in a row would properly count towards the algorithm of executing a fourFour scenario. **There were 8 different conditionals that needed to be tested within this method.** As such, I created 8 different unit tests that would allow me to ensure that a fourFour scenario would execute given a particular direction with 4 equal integers in a straight line. I conducted this test by first ensuring that the *“up”* direction would count towards the algorithm of executing the fourFour scenario, and I mixed this with the 7 remaining directions to ensure that I could expect and would obtain the correct boolean value of the executed method given using all 8 of the different scenarios.
   2. Verifying different possible threeThree scenarios by testing 0, 1, 2, and more than 2
      1. For this method, **I found it necessary to test for 0, 1, 2, and more than 2 directions that would have four straight equal integers.** For this test, the primary element that I used to test for a straight line was the integer *4*. As such, I tested the scenarios in which there were no directions that caused a four in a row scenario, a scenario in which there were four integers in a row in the *“up”* direction, a scenario in which there were three integers in a row scenario in both the *“up”* and the *“left”* directions (causes fourFour scenario), and a scenario in which there were three integers in a row in the *“up”*, *“left down”*, and *“right down”* direction (also causes fourFour scenario).
5. (Helper) Method 5 - fiveFive algorithm (gameInPlay)
   1. For this method, I found it necessary to check for the situations that a winner could be determined. **There were 3 different conditionals that needed to be tested within this method.** First of all, I tested a board that contained a scenario in which neither player has won yet (no equal integers in a straight line of size 5). Second of all, I tested a board that contained a scenario in which the white player has won. Finally, I tested a board that contained a scenario in which the black player has won. I observed that this test worked by referencing my interactions pane to confirm that the proper messages were printed for a white player’s win and a black player’s win.