For testing my project, I took a structural testing approach as I believed that the internal structure of the software, the functions, classes, algorithms and data structures used were the most important things to test.

Tests conducted:

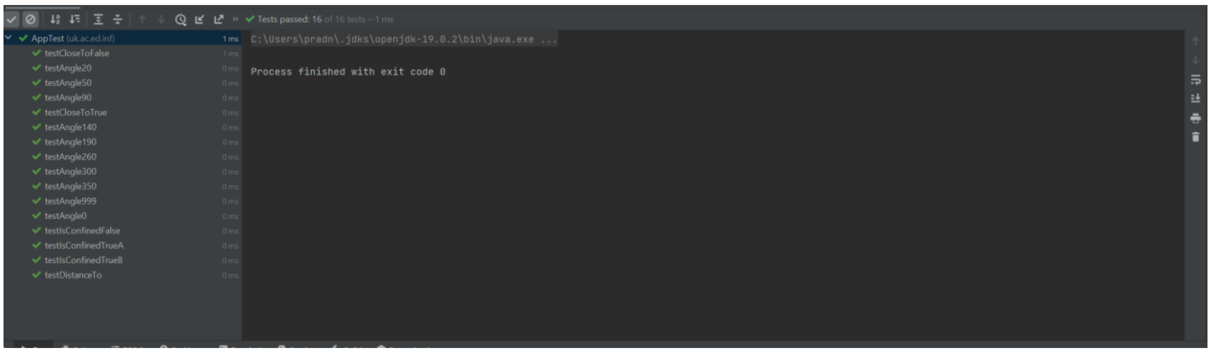
- Unit testing of the methods
    - Flightpath of drone
    - Movement into no fly zones
    - Movement out of confined area
    - Angles of movement
    - Number of orders placed by a person
    - Number of items in an order
    - Number of restaurants an order consists of
    - Connection to the web server and database
- Efficiency and performance of different heuristics and travelling salesman approaches
- Average amount of money earned per day
- Number of orders failed per day
- Performance logs
- Number of moves the drone makes based on different TSP algorithms

Implementation of the tests:

1. The unit tests were made for each of the methods that played a crucial role in the internal structure of the project. Unit tests and test cases were written in the AppTest.java file and ran to get the results.
2. The efficiency and performance of the different heuristics and tsp algorithms was tested by calculating the average money earned and the number of failed deliveries based on some synthetic data. Random, greedy and nearest neighbour approach was used as heuristics and djikstras algorithm was used for the tsp algorithm. Christofedes algorithm was also meant to be implemented but there was not enough time. The same test with the same synthetic data was run for each heuristic and was then compared. The synthetic data was created to simulate a years worth of orders.
3. I implemented a new method called getPercentageMonetaryValue in the Drone.Java file. This method would return the percentage of money earned for orders that were successful compared to failed ones and return it. This method allowed me to compared the different heuristics and also test the algorithm if it fulfilled the requirement of maximizing the money earned.
4. I added a log function for the number of orders failed per day
5. I added performance logs throughout the project such as the average time for a delivery per day, the average money earned in a month, the time taken to compute the flightpath, a connection was established to the server or the database.
6. I logged the number of moves the drone made when it completed all order before the battery runs out for 100 days and plotted a scatter plot that helped visualize the data.

Results:

1. Only 50% of the unit tests passed when tested at first but after rectifying those issues all unit tests passed in the end.



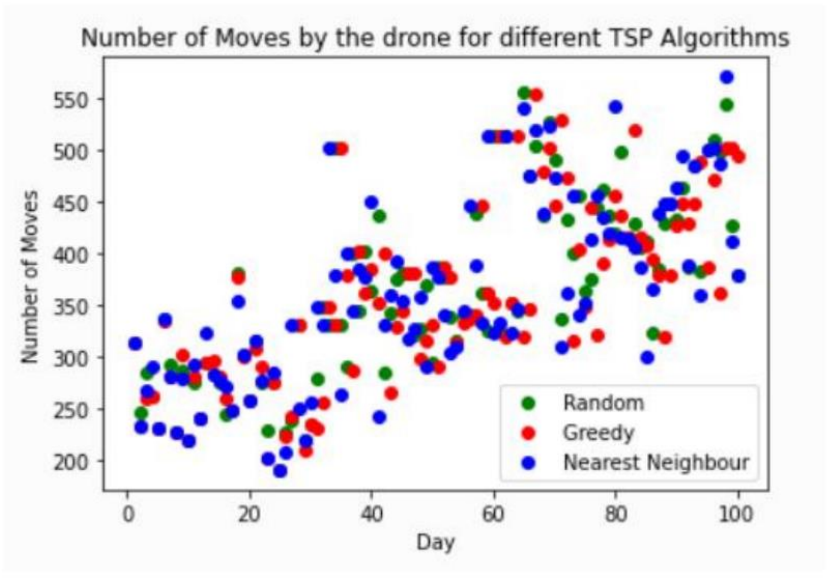
```
Tests passed: 16 of 16 tests – 1 ms

AppTest (uk.ac.ed.inf)        1 ms
    testCloseToFalse          1 ms
    testAngle20               0 ms
    testAngle50               0 ms
    testAngle90               0 ms
    testCloseToTrue           0 ms
    testAngle140              0 ms
    testAngle190              0 ms
    testAngle260              0 ms
    testAngle300              0 ms
    testAngle350              0 ms
    testAngle999              0 ms
    testAngle0                0 ms
    testIsConfinedFalse       0 ms
    testIsConfinedTrueA       0 ms
    testIsConfinedTrueB       0 ms
    testDistanceTo            0 ms

C:\Users\pradn\.jdks\openjdk-19.0.2\bin\java.exe ...

Process finished with exit code 0
```

2. Random approach: on average it completed only 60% of the orders and earned only 56% of the money
   This approach worked fine when the number of orders per day was low like 5 or 6 but got worse as the numbers increased.
   Nearest neighbours : on average it completed 65% of the orders and earned 63% of the money
   This order worked well in most cases, but it prioritised getting the lowest number of moves over getting the most money hence on days when it was impossible to complete all orders it made less money than the greedy approach
   Greedy: on average it completed 88% of the orders and earned 90% of the money
   The greedy approach seemed to be the best among the three as it prioritised the money earned which is the main objective of the system. Even on the day with the greatest number of orders which is 27/12/2023 the percentage of monetary value is 84% which is higher than the other algorithms.
3. As shown in number 2, it was used to compare the different algorithms
4. Shown in number 2
5. When the results is different from what was expected, it returns a different log which helped me track the progress of the program



6.

Evaluation of the results:

I ran the built-in code coverage generator in the intellij editor on all of my unit tests and test cases. This also ran line, method and branch coverage on all classes in the project. These results can be found in the code coverage folder of the repository. The results show that the majority of the branches were executed during testing . The same can be said about the line coverage. However there are still a lot of classes with low line coverage such as the Menus and LongLat class. As for those with a value of 0% these classes represent objects. The results from the code coverage show that the a low amount of untested code in the project but since some of them are still low, this may pose an issue in the future.

The defect detection rate was at an acceptable rate of 3%. This shows that it is not perfect and still has some defects that can be improved. The test case efficiency was 0.12 test per millisecond which is an acceptable time hence it can be safe to say that based on the evaluation criteria I chose, the testing process and techniques I used were rather sufficient and adequate but could be improved.