Homework 4

CSCI 3081w

Due: 5.1.22

Group Members: chen6751, diaz0300, joh20191, ronde021

**Write Up**

**What does our feature do?**

Our feature uses the singleton design pattern to collect data from drone trips. The point of our feature is to compare data using the various pathing methods such as Beeline, Dijkstra, DFS, and A\*. We specifically collect the instantiation time of a pathing method object, as well as the total time for the drone to complete the trip. With these values, we are able to identify key characteristics that make these different algorithms unique.

**Which design pattern did we implement and why?**

We implemented the singleton design pattern for our feature. The singleton class provides key advantages that aid in our goal; instance control, flexibility, and memory efficiency. In data collection, we need to ensure that we keep our control variables static while testing. We also need the flexibility to switch between the various search algorithms to complete an extensive test suite. Finally, we do not want our testing to be a major drain on memory, a singleton class will make our process memory efficient.

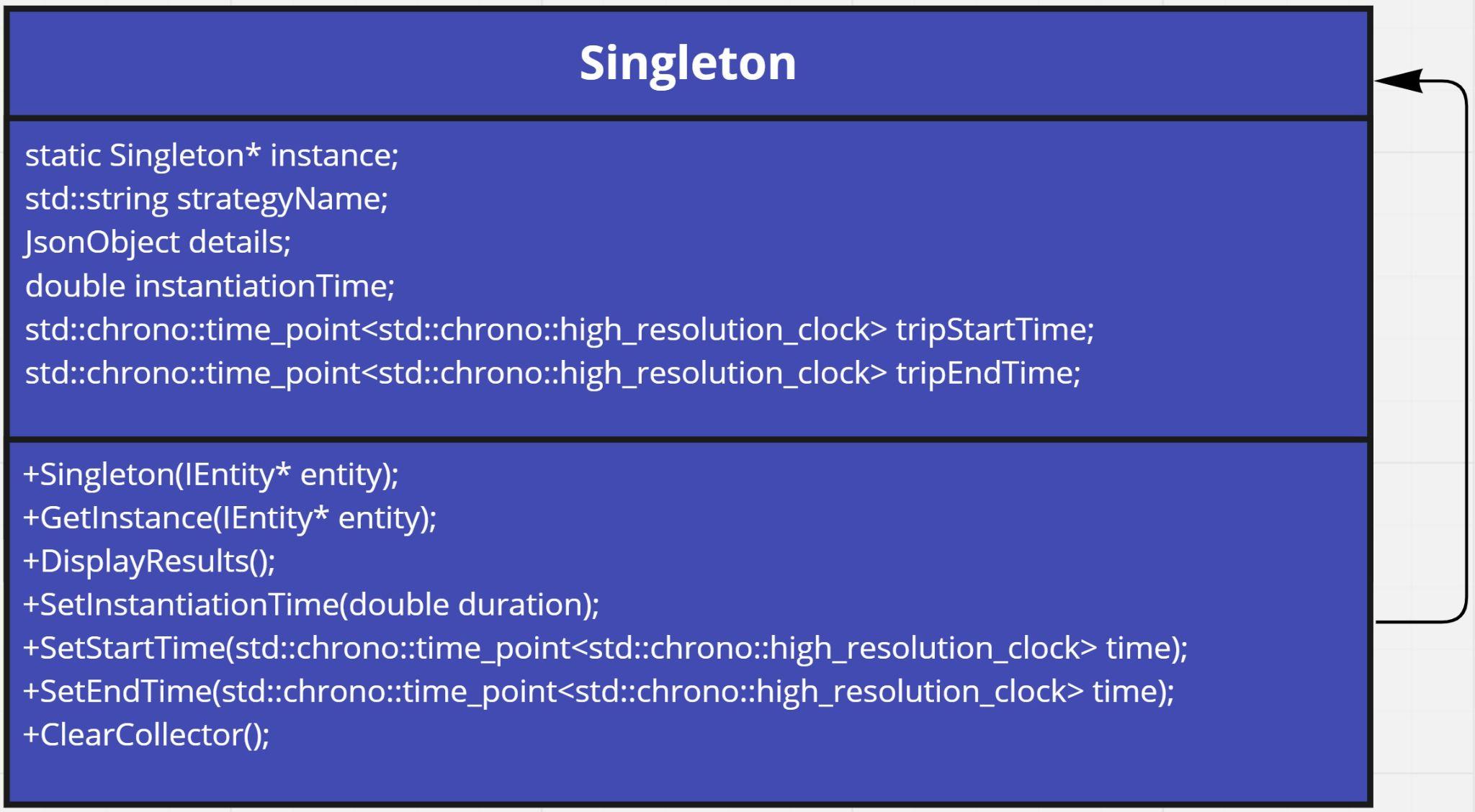
**How does this feature add to the existing work?**

This feature adds to the existing work since it will help users understand and differentiate between the various pathing methods. We are able to find which algorithms return optimal paths that give us the shortest travel time possible. Also, we can differentiate between the optimal path algorithms by seeing how much computational energy they require to return their results. Additionally, it is very extendable for other uses since its sole purpose is to collect and present data. Thus, if future changes were to be made, this class is very easy to modify in order to obtain new data. Our implementation also avoids putting more strain on the user, as the data collection process does not require any additional input.

**Why is this feature interesting?**

Our feature is interesting because it will allow the analysis of different data between the different pathing methods. This will show whether a certain pathing strategy is more effective than the other such as the pathing methods mentioned above. We can also identify which algorithms offer optimal results. Obviously, we found that the fastest and most efficient pathing method is the Beeline method since there is barely any logic involved in how the drone has to move and it doesn’t actually follow the roads. Thus, this method had the fastest instantiation and trip duration by far. From the data we collected, we found that A\* and Dijkstra are both optimal algorithms, as they both find the shortest path from the starting point to the ending point. Thus, the drone always takes the same path between two points when using either algorithm. The difference we found, however, is that the Dijkstra algorithm was just barely faster when instantiating in comparison to the A\* algorithm. Finally, we found that the DFS algorithm is not optimal as it would typically take a path that differed from the A\* and Dijkstra algorithms. This trend became way more apparent when testing with a longer distance, as the drone would go way off the shortest path. This resulted in the trip duration being longer than all the other algorithms, and would typically differ more the longer the distance. We also found that the DFS had relatively the same instantiation time when compared to the A\* and Dijkstra methods. We came to an assumption that the DFS algorithm would likely instantiate faster than A\* and Dijkstra for distances that are outside the scope of this project since it doesn’t try to find the shortest path. However, we could not actually collect data of this happening since the bounds of the map are just too small to test this theory.

**UML Diagram**



**Test Results**

**Short Trip**

| Algorithm | Instantiation Time (ms) | Trip Duration (s) |
| --- | --- | --- |
| Beeline | 0.000826 | 25.4369 |
| DFS | 13.0168 | 31.3295 |
| A\* | 11.6303 | 31.2009 |
| Dijkstra | 11.5271 | 31.1350 |

**Long Trip**

| Algorithm | Instantiation Time (ms) | Trip Duration (s) |
| --- | --- | --- |
| Beeline | 0.000899 | 37.3136 |
| DFS | 21.0269 | 48.4731 |
| A\* | 21.5407 | 46.9321 |
| Dijkstra | 17.4372 | 46.9461 |

**Notes**

* Beeline is our control algorithm for these tests.
* A\* and Dijkstra found the same path for the short and long trips. (Optimal path)
* DFS, A\*, and Dijkstra took around the same time for the short trip
* Dijkstra instantiates a little faster than A\* and DFS
* DFS does not find the shortest path, as it took a different path for the long trip
* DFS took a long time to instantiate for both trips
* There is a systematic error to account for in both instantiation time and trip duration, as there currently isn’t a method to consistently choose the exact same starting and ending points
* There is also random error as the machine doesn’t consistently take the exact same amount of time to run