Rafael Augusto Teixeira Dias (teixe029)

Andrew Tsai (tsai0126)

(unable to contact other two team members)

Video Link: <https://drive.google.com/file/d/1PPWldu2Zm3mrX6YqCrsjxnb1-vB3D_T6/view?usp=sharing>

Homework 4 Write Up

Team-010-45

This project is a Drone Simulation System, in which the user has the capability of scheduling trips, similarly to a ride app. In the simulation; however, the passengers are robots, and the cars are drones. The users instead of only having the ability of only scheduling trips for themselves, have the capability of creating as many robots, helicopters, and drones to perform as many trips needed. The decision of which drone will perform a trip is done by proximity. Lastly, a 3D virtual model of the University of Minnesota is the scene for the Simulation, which provides a considerably large model.

To initially start the project in a Linux terminal, one must change directory **(cd)** to the repository and build the project with **(make -j**). After the compilation, the following command will run the simulation

**=> ./build/bin/transit\_service 8081 apps/transit\_service/web/**

This command runs the executable transit\_service and using port 8081 communicates to the internet through the CSS and HTML files inside the apps folder.

The new feature implemented adds the possibility of accidents to happen during trips, which damage the drones, making them path to a repair station to be repaired or to stop working completely depending on the severity of the accident, if that is the case, we intend to make other drones assume the rides so they all can be completed. It is a significant extension since for any practical uses of a ride simulation (in a business for example), accidents should be accounted for as they might inhibit/delay rides, altering the logistics of the current rides. Additionally, if there is no possibility of repairing the fleet of drones, long term operation will be affected.

To implement the extension, the composite factory pattern (already developed to create the entities of the simulation) was extended to include a repair station entity that is in in the proximities of the Huntington Stadium. To track the possibilities of accidents, the decorator pattern was utilized to wrap the drone’s pathing strategy to include a 20% chance of a severe accident happening and a 20% chance of a minor accident to happen whenever a ride is scheduled, altering the strategy accordingly. This implementation was chosen, as instead of creating an additional creational design pattern when one is already implanted to create any object that inherits from the base form of entity of the simulation (IEntity). Additionally, in order to change the pathing of the drone, without significantly changing the core strategies that define paths, the decorator design pattern can just seemingly provide and add on to the pathing which covers only the necessary moving of the drone.

The probability of an accident cannot be altered through the running application, however, one can change the probability as they might see fit by accessing

=> **/libs/transit/src/ConditionDecorator.cc**

And altering the (**accidentProbability()**) function, and the if clauses in (**Move(IEntity\* entity, double dt)**).

The development process of this project consisted of one sprint, in which the team used the sashimi model to quickly perform the task due to the predictiveness of the model, while also having the flexibility to adapt earlier parts of the project as most phases of the project were done simultaneously. However, to increase the productivity of a next sprint, more adaptive models can be considered as debugging in general was one of the most prevalent problems.

The following is the UML diagram of the new classes, in which the UFO class stands for Unique Fixing Operation and works as the repair station:

Diagram

Description automatically generated