CSCI 3081W Project README
Team 10
Jimmy Xiao, Harrison Gauerke, Sean Beaulieu
xia00261, gauer045, beaul116

Docker Link:
https://hub.docker.com/repository/docker/lolvscf/team-010-39-homework4

Video Demo Link:
https://drive.google.com/file/d/1ZSumjvoWq2Ifp-PreGV2hcjzFj6Xl19V/view?usp=sharing

Project Overview:

This project is about a transportation simulation similar to ridesharing apps such as Uber. This project is written in C++. Development of this simulation system incorporates many aspects of software development that were taught in class - including but not limited to: software design patterns, software development methodologies, Git, code documentation, and code styling. In doing this project, many aspects of the development process were practiced.

How to run the simulation:

The simulation can be ran by these steps:

- Navigate to the projects base directory on terminal
- Run “make -j”
- Start the project to a local host by running “./build/bin/transit_service (port) apps/transit_service/web/
- Navigating to a local web browser and typing in the local host and port
- localhost:port.schedule.html in order to schedule the trips

What does the simulation do?

The simulation begins by idling - there are several moving objects on the map. Among these are a car, a drone, a helicopter, and a duck. Transportation can be scheduled by navigating to the schedule page. From there, users can input a trip name, a pickup spot, and a destination on the map, as well as a preferred navigation strategy (being Depth First Search, dijkstra, or AStar). Once that trip is scheduled, as soon as the drone is done with the other trips first in line, it will travel to the pickup destination and pick up the target. It will then use the given navigation strategy to deliver the customer to their destination. Once there, it will idle until given another trip to carry out.

Additional Functionality;

The scope of the additional functionality added on to this project include adding a battery feature to the drones, fixed recharge stations to replenish the battery, and an emergency pickup in case of the drone running out of power. The battery feature functions similarly to how drones and vehicles in real life work - cars need gas and drones need electricity. Drones have an additional variable to them - charge. If the drone runs out of charge, it will be unable to move. To remedy this, recharge stations have been added. Additional logic has been added to the navigation of the drones by determining whether a drone can finish a trip and also make the distance to the recharge station. In the event that a drone gets stuck without power, there is an emergency pickup drone that will travel to the drone using the beeline strategy and return the drone to the nearest recharge station in order for the charge to be refilled[1].

One other extension that has been added is the mobile recharge station. Using a pathing strategy to perpetually loop around campus, the mobile recharge station is a station that drones can recharge at if it is the closest station to them. While the mobile recharge station does not run on charge, it still provides utility to the drones and increased interaction with the map. Functionally, the mobile recharge station is no different than a fixed recharge station with the exception of it's pathing around the map.

All of these extensions feature no user interactability. While the user can see the mobile recharge station as well as the fixed recharge station, they cannot influence the

charging patterns of the drone, nor can they control the path of the mobile recharge station.

The battery feature is implemented using a decorator design pattern. This seemed logical to use given that the battery function could easily be 'wrapped' around the existing drone object, while maintaining the structure and functionality of the drone. This is relevant to the single-responsibility principle, by adding new functionality to the Drone object at runtime while avoiding making another Drone class that includes the battery extension.

The recharge and mobile recharge stations are implemented using the factory design pattern. The factory design pattern allows for multiple different types of entities to be made from the same function in different factory classes. By creating the recharge stations via a factory method, the responsibility of instantiation can be given to subclasses instead of the same class for every entity.

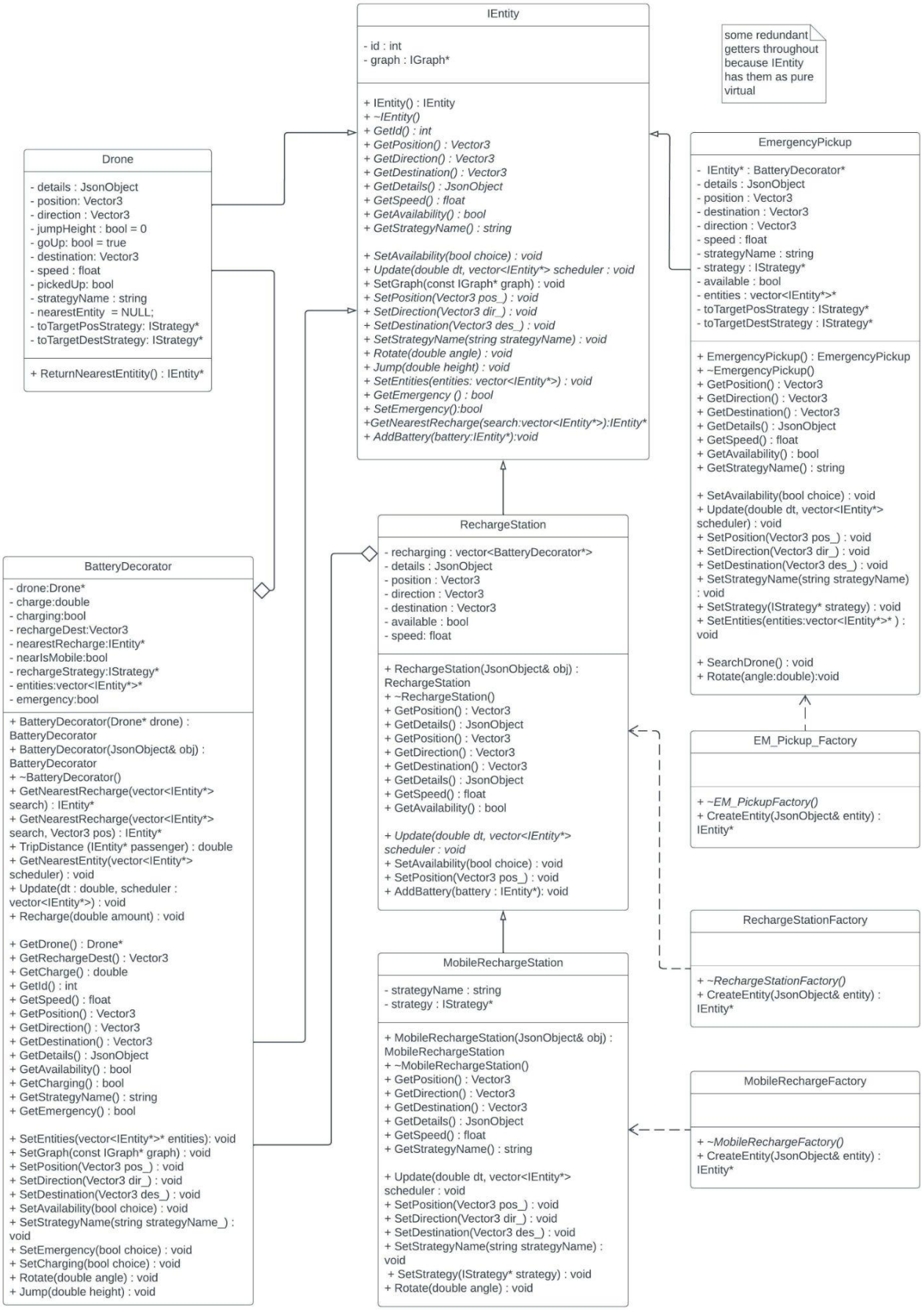
The extensions are interesting because having automated drones taking into account their range and distance to the nearest point of recharge mirror what the delivery and transport systems in real life likely do. Drone delivery services are one such example. More and more delivery services are expected to use drones in the future to deliver goods and products. Globally, the drone delivery market size is forecasted to reach around $10 billion USD by 2027. This is a large market, and there will no doubt be technological advancements in both the routing capabilities of delivery drones as well as their power management. The company that provides the best code and hardware for the fastest, longest-lasting drone will likely secure a wealth of revenue from delivery providers.

[1] Error Note: if the drone fails/dies after it has assigned itself a trip, despite wanting to do so we were unable to "cancel" the trip (make the robot available again and do the same for the drone when its emergency state is settled). We believe this is an issue with the multithreading of the program simultaneously reading/writing the availability state and were unable to directly fix it in time. However, this situation never arises so long as at least a FULL battery is able to complete a worst-case trip (corner to opposite corner furthest from recharge), so we adjusted the maximum charge accordingly in our final iteration.

Agile Sprint Retrospective:

Development for the latter end of the project featured the incorporation of the software development methodology Agile. By using the tool Jira, made by Atlassian, the group was able to better prioritize the tasks that the project required. Initially, the sprint planning meeting was useful in figuring out which group members would commence work on specific parts of the project. One challenge that was encountered was the lack of daily scrum standup meetings. When these meetings are done daily or frequently, they allow the project team to communicate about the work being done efficiently. The problem was remedied in part due to communication over text and Discord - these channels allowed the group to stay informed when live meetings weren't an option.

UML Class Diagram (below):


IEntity
- id : int
- graph : IGraph*
+ IEntity() : IEntity
+ ~IEntity()
+ GetId() : int
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetSpeed() : float
+ GetAvailability() : bool
+ GetStrategyName() : string
+ SetAvailability(bool choice) : void
+ Update(double dt, vector<IEntity*> scheduler : void
+ SetGraph(const IGraph* graph) : void
+ SetPosition(Vector3 pos_) : void
+ SetDirection(Vector3 dir_) : void
+ SetDestination(Vector3 des_) : void
+ SetStrategyName(string strategyName) : void
+ Rotate(double angle) : void
+ Jump(double height) : void
+ SetEntities(entities: vector<IEntity*>) : void
+ GetEmergency () : bool
+ SetEmergency():bool
+GetNearestRecharge(search:vector<IEntity*>):IEntity*
+ AddBattery(battery:IEntity*):void
some redundant getters throughout because IEntity has them as pure virtual
Drone
- details : JsonObject
- position: Vector3
- direction : Vector3
- jumpHeight : bool = 0
- goUp: bool = true
- destination: Vector3
- speed : float
- pickedUp: bool
- strategyName : string
- nearestEntity = NULL;
- toTargetPosStrategy: IStrategy*
- toTargetDestStrategy: IStrategy*
+ ReturnNearestEntitity() : IEntity*
EmergencyPickup
- IEntity* : BatteryDecorator*
- details : JsonObject
- position : Vector3
- destination : Vector3
- direction : Vector3
- speed : float
- strategyName : string
- strategy : IStrategy*
- available : bool
- entities : vector<IEntity*>*
- toTargetPosStrategy : IStrategy*
- toTargetDestStrategy : IStrategy*
+ EmergencyPickup() : EmergencyPickup
+ ~EmergencyPickup()
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetSpeed() : float
+ GetAvailability() : bool
+ GetStrategyName() : string
+ SetAvailability(bool choice) : void
+ Update(double dt, vector<IEntity*> scheduler) : void
+ SetPosition(Vector3 pos_) : void
+ SetDirection(Vector3 dir_) : void
+ SetDestination(Vector3 des_) : void
+ SetStrategyName(string strategyName) : void
+ SetStrategy(IStrategy* strategy) : void
+ SetEntities(entities:vector<IEntity*>*) : void
+ SearchDrone() : void
+ Rotate(angle:double):void
RechargeStation
- recharging : vector<BatteryDecorator*>
- details : JsonObject
- position : Vector3
- direction : Vector3
- destination : Vector3
- available : bool
- speed: float
+ RechargeStation(JsonObject& obj) : RechargeStation
+ ~RechargeStation()
+ GetPosition() : Vector3
+ GetDetails() : JsonObject
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetSpeed() : float
+ GetAvailability() : bool
+ Update(double dt, vector<IEntity*> scheduler : void
+ SetAvailability(bool choice) : void
+ SetPosition(Vector3 pos_) : void
+ AddBattery(battery : IEntity*): void
BatteryDecorator
- drone:Drone*
- charge:double
- charging:bool
- rechargeDest:Vector3
- nearestRecharge:IEntity*
- nearIsMobile:bool
- rechargeStrategy:IStrategy*
- entities:vector<IEntity*>*
- emergency:bool
+ BatteryDecorator(Drone* drone) : BatteryDecorator
+ BatteryDecorator(JsonObject& obj) : BatteryDecorator
+ ~BatteryDecorator()
+ GetNearestRecharge(vector<IEntity*> search) : IEntity*
+ GetNearestRecharge(vector<IEntity*> search, Vector3 pos) : IEntity*
+ TripDistance (IEntity* passenger) : double
+ GetNearestEntity(vector<IEntity*> scheduler) : void
+ Update(dt : double, scheduler : vector<IEntity*>) : void
+ Recharge(double amount) : void
+ GetDrone() : Drone*
+ GetRechargeDest() : Vector3
+ GetCharge() : double
+ GetId() : int
+ GetSpeed() : float
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetAvailability() : bool
+ GetCharging() : bool
+ GetStrategyName() : string
+ GetEmergency() : bool
+ SetEntities(vector<IEntity*>* entities): void
+ SetGraph(const IGraph* graph) : void
+ SetPosition(Vector3 pos_) : void
+ SetDirection(Vector3 dir_) : void
+ SetDestination(Vector3 des_) : void
+ SetAvailability(bool choice) : void
+ SetStrategyName(string strategyName_) : void
+ SetEmergency(bool choice) : void
+ SetCharging(bool choice) : void
+ Rotate(double angle) : void
+ Jump(double height) : void
EM_Pickup_Factory
+ ~EM_PickupFactory()
+ CreateEntity(JsonObject& entity) : IEntity*
RechargeStationFactory
+ ~RechargeStationFactory()
+ CreateEntity(JsonObject& entity) : IEntity*
MobileRechargeStation
- strategyName : string
- strategy : IStrategy*
+ MobileRechargeStation(JsonObject& obj) : MobileRechargeStation
+ ~MobileRechargeStation()
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetSpeed() : float
+ GetStrategyName() : string
+ Update(double dt, vector<IEntity*> scheduler : void
+ SetPosition(Vector3 pos_) : void
+ SetDirection(Vector3 dir_) : void
+ SetDestination(Vector3 des_) : void
+ SetStrategyName(string strategyName) : void
+ SetStrategy(IStrategy* strategy) : void
+ Rotate(double angle) : void
MobileRechargeFactory
+ ~MobileRechargeFactory()
+ CreateEntity(JsonObject& entity) : IEntity*