Team: 001-18
Members: Enkhjin Boldbaatar (boldb002), Sol Kim (kim01540), Cole Bye (bye00036), James Kallagunta (kalla104)

About the project:
This project is a transportation simulation system like an Uber. You can schedule trips by setting the pickup location and the destination of the robot. The drone will come and pick up the robot toward its destination.

How to run the simulation:
1. Navigate to the project directory by “cd /path/to/repo/project” in the terminal.
2. To run the simulation, do "make -j" in the terminal, then "./build/bin/transit_service 8081 apps/transit_service/web/".
3. Now you can schedule trips of the robots with the robot’s name and its pickup location and destination in this page “http://127.0.0.1:8081/schedule.html”
4. You can watch the simulation on this page “http://127.0.0.1:8081” and change your camera view into locking on to entities using the buttons in the top right corner.

What does the simulation do:
On the schedule page, you can add helicopters, add repair stations, and schedule trips.
- A helicopter can be added by simply clicking a button named “Add Helicopter.
- A repair station can be added after selecting its location on the map and clicking the “Add Repair Station” button.
- To schedule a trip, you will define the robot’s name, select a search strategy that will be used to move toward the destination. You can select the pickup location and the final destination on the map. Then, you will be able to schedule a trip by clicking the “Schedule Trip” button.

On the 3D visualization page, you will see helicopters, drones, robots, and repair stations.
- Robots are at their pickup location and waiting for drones to pick them up.
- Drone will pick up a robot toward its destination according to the schedule.

New feature
Q. What does it do?
: The new feature has the drones take damage as they travel. When the drones take a significant amount of damage they travel to a repair station where their health is fixed back to the maximum. Users can also choose to create more repair stations.

Q. Why is it significantly interesting?
: This is interesting because in the real world drones would naturally wear down and take damage during operation. The added maintenance feature makes the simulation more realistic

and applicable to a real system. A real-world example is the people that go around to the electric scooters on campus to maintain them and charge the batteries. Since drones can fly on their own we can make it so they bring themselves to the nearest station.

Q. How does it add to the existing work?
: This adds to the existing work because we needed to add a new entity for the repair stations as well as a new system to keep track of the drone's health. The logic of the drone had to be updated to support the new repair feature. Our group also worked on the front end to allow for the creation of more repair stations.

Q. Which design pattern did you choose to implement it and why?
: The repair station entities were implemented using the abstract factory pattern. They were inherited from the IEntityFactory and aggregated in the CompositeFactory. This made the most sense because we were creating objects in our simulation and we already had an abstract factory for the drones and robots. Adding onto this with the repair station was the most logical and fit in with the O of solid. We also used a decorator pattern to implement the health/durability feature. We had a base decorator inheriting from IEntity with aggregation and a durability decorator inheriting from the base decorator. We used decorator because it is helpful for adding additional functionality to an existing object. In the future, if we wanted to add more features to the drone, we could easily do it without changing anything in the drone or durability decorator.

Q. Instruction to use this new feature
: This feature runs automatically with the simulation. For additional functionality, we also added a button to the bottom of the scheduler to add repair stations similar to how the labs added a helicopter but you can pinpoint where you want the repair station on the map. We also added a "clear points" button to make adding repair station and trip scheduling easier. If the user messes up the location of a point on the map they can click that button to try again.

Sprint retrospective

What went well

- We divided our tasks which helped us stay organized
- We were responsible in our duties and finished our tasks on time
- We asked for help from the TAs when we needed help which helped smoothen the coding process

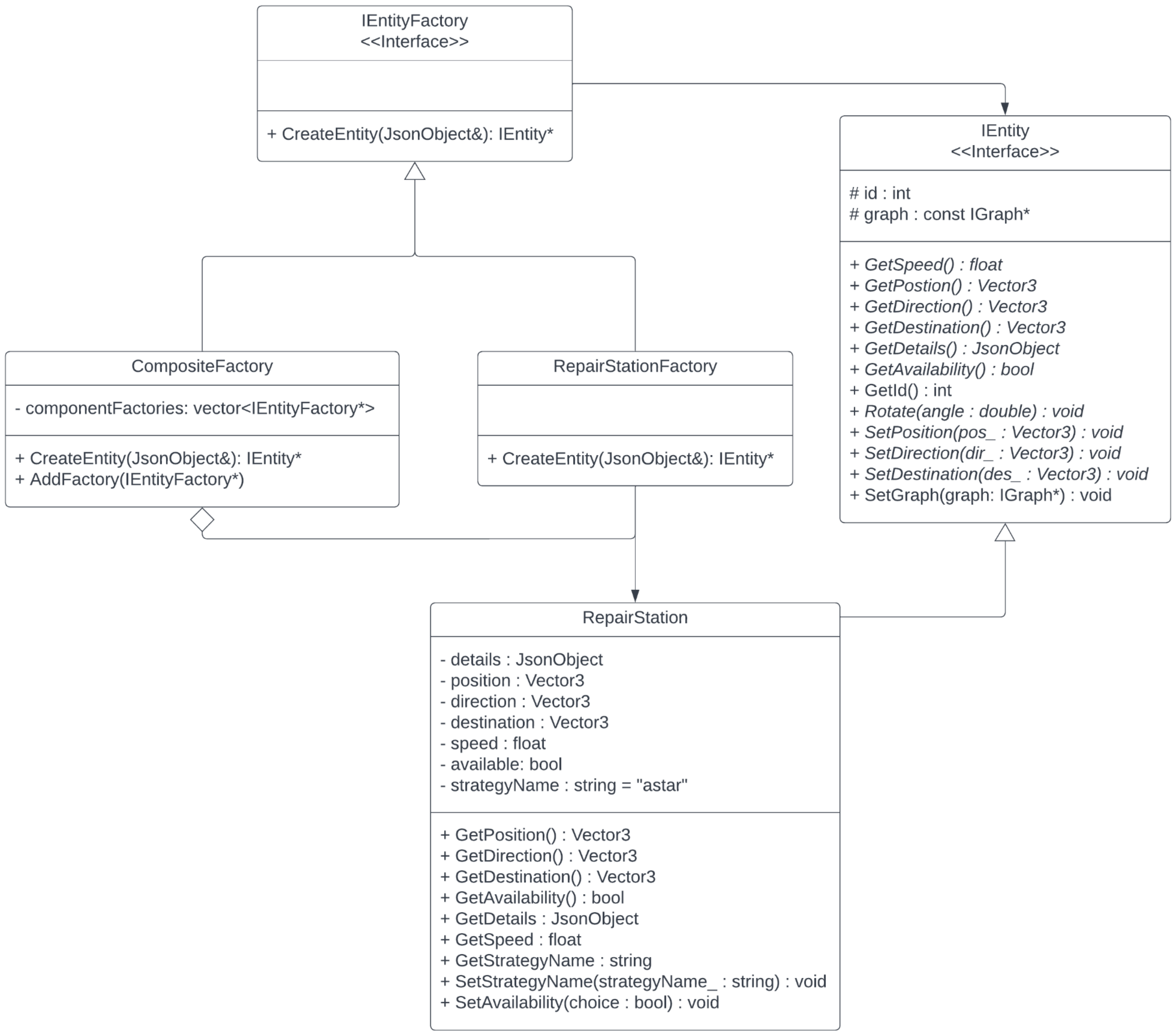
What did not go well

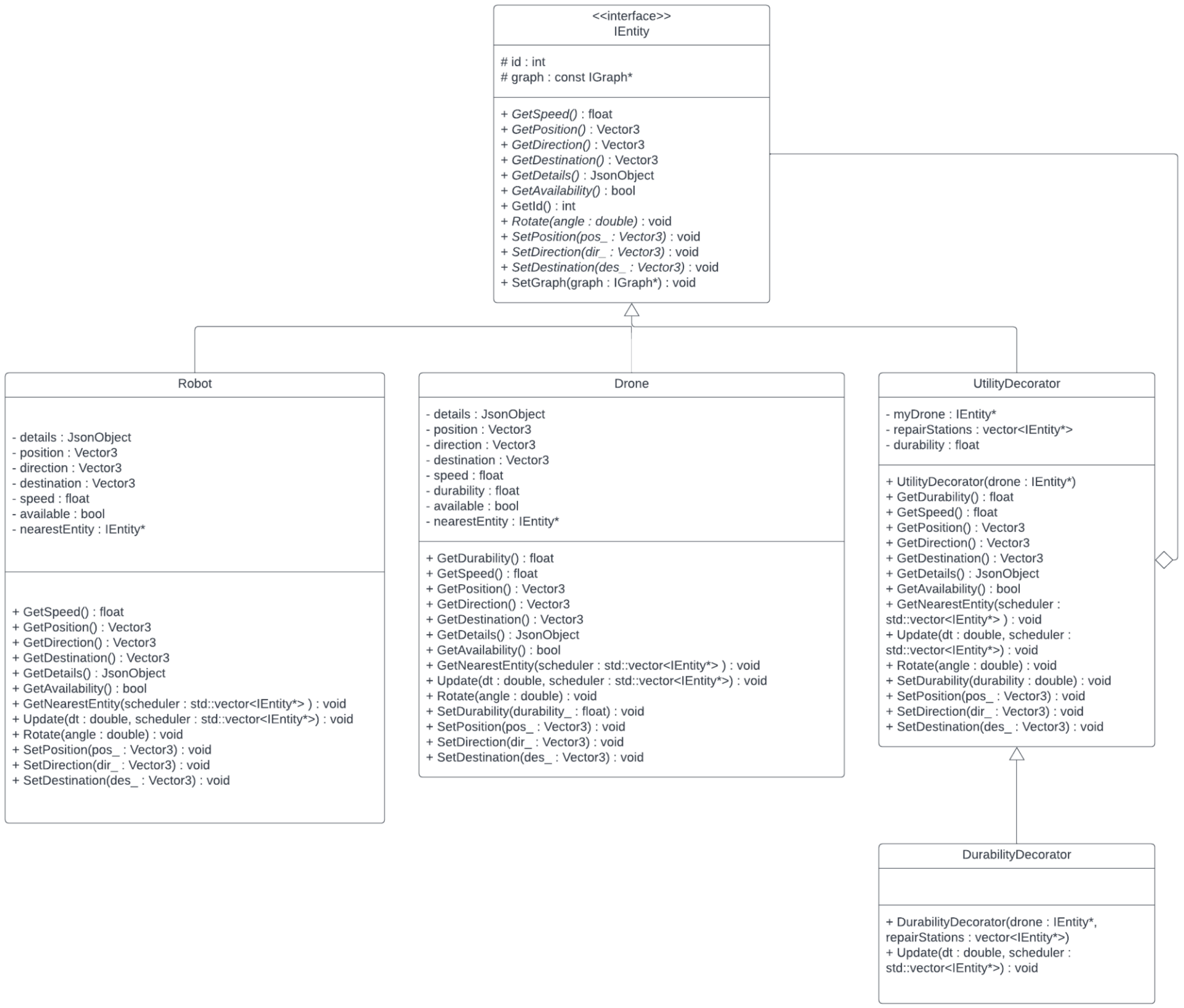
- Duration was too short, I feel like we could've done more if we had more time
- We only had one sprint

Ideas for hypothetical future sprints

- Do multiple sprints
- Start working early

UML


IEntityFactory
<<Interface>>
+ CreateEntity(JsonObject&): IEntity*
IEntity
<<Interface>>
# id : int
# graph : const IGraph*
+ *GetSpeed() : float*
+ *GetPostion() : Vector3*
+ *GetDirection() : Vector3*
+ *GetDestination() : Vector3*
+ *GetDetails() : JsonObject*
+ *GetAvailability() : bool*
+ GetId() : int
+ *Rotate(angle : double) : void*
+ *SetPosition(pos_ : Vector3) : void*
+ *SetDirection(dir_ : Vector3) : void*
+ *SetDestination(des_ : Vector3) : void*
+ SetGraph(graph: IGraph*) : void
CompositeFactory
- componentFactories: vector<IEntityFactory*>
+ CreateEntity(JsonObject&): IEntity*
+ AddFactory(IEntityFactory*)
RepairStationFactory
+ CreateEntity(JsonObject&): IEntity*
RepairStation
- details : JsonObject
- position : Vector3
- direction : Vector3
- destination : Vector3
- speed : float
- available: bool
- strategyName : string = "astar"
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetAvailability() : bool
+ GetDetails : JsonObject
+ GetSpeed : float
+ GetStrategyName : string
+ SetStrategyName(strategyName_ : string) : void
+ SetAvailability(choice : bool) : void


<<interface>>
IEntity
# id : int
# graph : const IGraph*
+ GetSpeed() : float
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetAvailability() : bool
+ GetId() : int
+ Rotate(angle : double) : void
+ SetPosition(pos_ : Vector3) : void
+ SetDirection(dir_ : Vector3) : void
+ SetDestination(des_ : Vector3) : void
+ SetGraph(graph : IGraph*) : void
Robot
- details : JsonObject
- position : Vector3
- direction : Vector3
- destination : Vector3
- speed : float
- available : bool
- nearestEntity : IEntity*
+ GetSpeed() : float
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetAvailability() : bool
+ GetNearestEntity(scheduler : std::vector<IEntity*>) : void
+ Update(dt : double, scheduler : std::vector<IEntity*>) : void
+ Rotate(angle : double) : void
+ SetPosition(pos_ : Vector3) : void
+ SetDirection(dir_ : Vector3) : void
+ SetDestination(des_ : Vector3) : void
Drone
- details : JsonObject
- position : Vector3
- direction : Vector3
- destination : Vector3
- speed : float
- durability : float
- available : bool
- nearestEntity : IEntity*
+ GetDurability() : float
+ GetSpeed() : float
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetAvailability() : bool
+ GetNearestEntity(scheduler : std::vector<IEntity*>) : void
+ Update(dt : double, scheduler : std::vector<IEntity*>) : void
+ Rotate(angle : double) : void
+ SetDurability(durability_ : float) : void
+ SetPosition(pos_ : Vector3) : void
+ SetDirection(dir_ : Vector3) : void
+ SetDestination(des_ : Vector3) : void
UtilityDecorator
- myDrone : IEntity*
- repairStations : vector<IEntity*>
- durability : float
+ UtilityDecorator(drone : IEntity*)
+ GetDurability() : float
+ GetSpeed() : float
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetAvailability() : bool
+ GetNearestEntity(scheduler : std::vector<IEntity*>) : void
+ Update(dt : double, scheduler : std::vector<IEntity*>) : void
+ Rotate(angle : double) : void
+ SetDurability(durability : double) : void
+ SetPosition(pos_ : Vector3) : void
+ SetDirection(dir_ : Vector3) : void
+ SetDestination(des_ : Vector3) : void
DurabilityDecorator
+ DurabilityDecorator(drone : IEntity*, repairStations : vector<IEntity*>)
+ Update(dt : double, scheduler : std::vector<IEntity*>) : void

Video Presentation Link: https://youtu.be/89a8dgU8DBY

Docker Link: https://hub.docker.com/repository/docker/jam4352/homework4-extension