Enabling Robot Flight Using Drone-Delivered Jet Packs

Khalid Barakzai, Daniel Kong, Alexander Zhu

University of Minnesota - Twin Cities
CSCI3081W — Program Development

May 1, 2022

1 Introduction

"What does it do?"

Robots in their current state are unable to move and hopelessly bound to the influence of gravity. To cure this weakness, this feature introduces jet packs to the project! Jet packs are delivered by drones to robots. With them, robots can fly using the beeline pathing strategy, high above the buildings. This also allows drones to be freed up to do other tasks instead of carrying around robots all day!

This feature also originally intended to include a jet pack model that would be rendered with both drones and robots. However, there were some issues exporting from Blender, a problem which was decided to be outside the scope of this project.

These new files are added to the code base:

```
project/libs/transit/include/RobotWithJetpack.h
project/libs/transit/src/RobotWithJetpack.cc
project/libs/transit/include/RobotWithJetpackFactory.h
project/libs/transit/src/RobotWithJetpackFactory.cc
project/libs/transit/include/NoMove.h
project/libs/transit/src/NoMove.cc
project/apps/transit_service/web/assets/model/jetpack.glb [NOT IMPLEMENTED]
```

The following existing files are modified:

```
project/libs/transit/src/simulation_model.cc
project/libs/transit/src/Drone.cc
project/apps/transit_service/web/schedule.html
```

A repository to the modified codebase for this project can be found here:

https://github.umn.edu/umn-csci-3081-s22/3081W-Project-Repository

2 Interest & Motivation

"Why is this work significantly interesting?"

As mentioned in the introduction, jet packs allow robots to fly – freeing up drones for other tasks. There are also a variety of jet pack speeds, which allows them to be more modular and gives greater ability for the user to define certain aspects of entity behavior.

Essentially, this work extends the functionality of the existing robot and drone classes. It also serves as practice in implementing new factories. Additionally, it required solving some unique problems, like creating a new method for adding and removing entities in a specific position from an existing environment and a new movement strategy pattern for post-delivery behavior of the drone.

3 Implementation

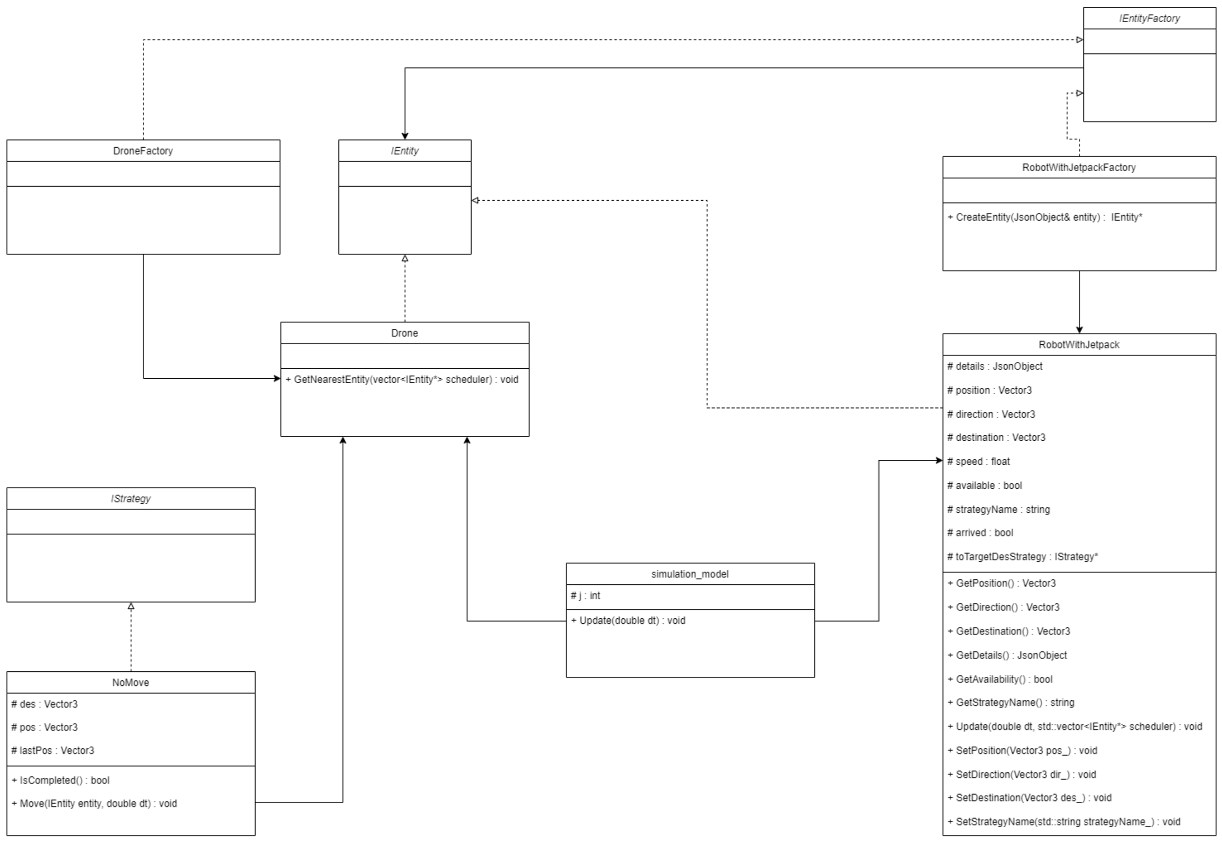
"How does it add to the existing work? Which design pattern did you choose to implement and why?"

Jet packs are implemented with robots as a new entity defined in `RobotWithJetpack.cc` that has its own factory and behavior (as outlined in the introduction). Since, these jet packs can vary in speed, which necessitated the implementation of a new factory pattern `RobotWithJetpackFactory.cc`. This new entity replaces existing robot entities (which are deleted from the environment, using some new methods in the `update()` function of `simulation_model.cc`) upon the delivery of a jet pack.

Drones track the robot's position and stop in place upon reaching the robot, at which point the robot will be equipped with a jet pack. The new `NoMove.cc` strategy stops the drone from continuing to move and frees up its availability, as though it had finished delivering a robot. `Drone.cc` is modified to include this new strategy. Alsoall of the existing strategy patterns are still available for robot delivery missions!

There are also slight modifications to `schedule.html`, where the drop-down menu has been modified to reflect new features.

4 UML Diagram



IEntityFactory
DroneFactory
IEntity
RobotWithJetpackFactory
+ CreateEntity(JsonObject& entity) : IEntity*
Drone
+ GetNearestEntity(vector<IEntity*> scheduler) : void
RobotWithJetpack
# details : JsonObject
# position : Vector3
# direction : Vector3
# destination : Vector3
# speed : float
# available : bool
# strategyName : string
# arrived : bool
# toTargetDesStrategy : IStrategy*
+ GetPosition() : Vector3
+ GetDirection() : Vector3
+ GetDestination() : Vector3
+ GetDetails() : JsonObject
+ GetAvailability() : bool
+ GetStrategyName() : string
+ Update(double dt, std::vector<IEntity*> scheduler) : void
+ SetPosition(Vector3 pos_) : void
+ SetDirection(Vector3 dir_) : void
+ SetDestination(Vector3 des_) : void
+ SetStrategyName(std::string strategyName_) : void
IStrategy
simulation_model
# j : int
+ Update(double dt) : void
NoMove
# des : Vector3
# pos : Vector3
# lastPos : Vector3
+ IsCompleted() : bool
+ Move(IEntity entity, double dt) : void