Project ANVIL - Spec 2.0

Aerodynamic Navigational Vehicles for Instantaneous Locomotion

Figure 1. Our plan to crush the competition...

ACME Corporation
October 31, 2020

Authors: CSCI 3081 Staff

Iteration 2: ANVIL Simulation

Note: This document is subject to change at any time depending on the business needs. We will send out a notification if there are any changes along with the documented revision.

Revision History:

- 2.1 - 11-13-2020 - CSCI 3081 Staff - Added details on Issue naming
- 2.0 - 10-31-2020 - CSCI 3081 Staff - ANVIL 2.0 specification release.

Drone Delivery Update

We are excited to see many promising prototypes from your teams! The work you have done so far has shown that our prototype phase has been a successful investment. It has even inspired new creative ideas from top management. We thank you for your continued efforts, and look forward to viewing your finished prototypes soon.

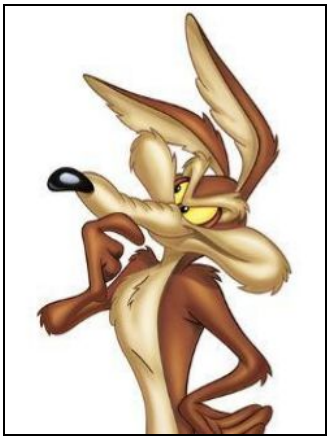
We are ready to start our next phase of development, which is far more practical. Ultimately we hope to profit from our investment in drone delivery. We now need a system that is both efficient and robust. Your next product needs to use real data to model realistic scenarios. You have shown that such a system could work, but now we need to develop such a system.

I want to update you on a new partnership with Planet X, one of our manufacturers. Through this relationship, we have made a strategic investment in a large fleet of diverse drones, acquired at stellar prices. Therefore, our simulation will need to support a wide range of specific drone types. Not all of these drones will deliver packages exactly the same way. Therefore, we need to start thinking about how to schedule drones based on location and ability.

We look forward to your solutions to this extremely difficult, but valuable, problem. Again, thank you and have fun changing the world!

Wile E. Coyote

Wile E. Coyote, CEO



1 Project Description and Overview

Due to increasing demand with online delivery and recent advances in drone technology, companies are excited to compete in logistics using the third dimension. As with anything, however, new technologies require overcoming significant challenges before implementation and deployment. These include but are not limited to physics, logistics, route planning, malfunctions, security, congestion, and cost.

Figure 2 - Drone delivery

Project ANVIL, Aerodynamic Navigational Vehicles for Instantaneous Locomotion, is our push to enable the state of the art **Drone Delivery System**. ANVIL itself is the proposed simulation of this system, which we must implement before deploying in the physical world. You and your team will prototype and simulate a real-world application, and optimize it for productivity in three separate iterations:

- ~~**Iteration 1 - Proof of Concept / Prototype**~~: The first goal is to develop a prototype of the system. Here we ask the question, can we get such a system to work at all?
- **Iteration 2 - Development**: In the development phase, we build the actual system to make it work and be useful.
- **Iteration 3 - Analysis and Optimization**: Here we enhance the system to handle real-world scenarios and more complicated situations. This involves data driven system analysis.

In each iteration, we will add more complexity to the project to continually approach the real physical world. This document describes **Iteration 2 - the ANVIL Simulation**. We start out with some updated background information about the state of the full drone solution. This is followed by the detailed specification for iteration 2. Then we provide evaluation criteria for what

we believe would satisfy the business needs of this project. A working simulation not only evaluates our assumptions, but helps us continue to improve our goals.

Project Timeline:

Below is a project timeline based on internal estimates:

Dates & Deadlines	**Item**	**Description**
11/2/2020 (Monday)	Initial Iteration 2 Specification	A release of the business requirements for iteration 2. At this point, design and development can begin.
11/9/2020 **(Due: Monday 11:55pm)** **Canvas**	First Deliverable: Peer Reviews	Provide feedback for your teammates on their iteration 1 design document.
11/17/2020 & 11/19/2020 (Tuesday & Thursday)	Automated Assessment	We will run our extended tests once on each codebase. This will help you fix any issues you might have with our grading scripts.
11/20/2020 **(Due: Friday 11:55pm)** **Canvas / GitHub**	Final Deliverable: Iteration 2	All remaining business requirements. This includes the design document.

Document Organization:

1. Project Description and Overview
2. Background
3. Iteration 2: ANVIL Simulation Specification
4. Evaluation Criteria

2 Background

(For fun only! Move on to 2.1 for people who are just wanting to do work.)

On March 22, 2010, NASA's Mars rover, Spirit, sent it's last message after months of being trapped in sand at a location called "Troy" (Figure 3). Due to the ACME Company's close ties with government research, we had the privilege of knowing the cryptic final message: "Please do not block my view of Venus."

Not known to the public, 5 years later in 2015, NASA started receiving messages from Spirit. The first message said, "I have perfected your primitive technology and I will arrive soon to accomplish the mission - Commander of Flying Saucer X-2". The second message provided detailed specifications of flying drones written in a strange language.

Figure 3 - NASA Mars Exploration Patch

In early 2016, Marvin the Martian (apparently the commander of the Flying Saucer X-2), arrived at Area 51 to discuss what he called the Illudium Q-36 Explosive Space Modulator. He explained, "I am here to solve Earth's fuel problem." At this point the ACME Corporation became involved because of our reputation with creative explosives. Unfortunately, no scientist or engineer could understand his methods, which in Marvin's words made him "very angry, very angry indeed". Please refer to Exhibit A summarizing our interaction with Marvin the Martian.

Marvin appeared to be frustrated by the human intellect and left to start his own manufacturing company, Planet X. The company's motto is: "There is a growing tendency to think of man as a rational thinking being, which is absurd." Regardless, some of our best products come from this company (e.g. the ACME Ultimatum Dispatcher, the ACME Re-Integrator, and the ACME Time-Space Device - See Figure 4). Recently, we have discovered that Planet X is investing heavily in drone technology and looking for buyers. This appeared to be a match made in the heavens. Fortunately, due to our close historical ties with Marvin the Martian and strategic partnership with Planet X, we have been given the opportunity to purchase an initial fleet of drones at steller prices.

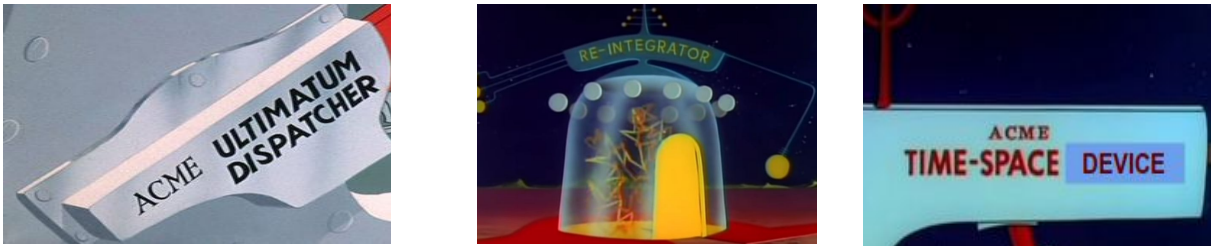


Figure 4 - ACME Products with ties to Planet X.

2.1 The Planet X Drone Fleet

Therefore, we are proud to announce our new Planet X Drone Fleet consisting of several different types of drones that need to be simulated (See section 2.1 for more details). We want to build the best delivery system based on their constraints, strengths, and weaknesses. In other words we are no longer working with simple drones.

Besides understanding how these drones work, it is important for us to start working with real street and topology data so that we can simulate existing environments. This is detailed below in section 2.2. Finally, we need sophisticated algorithms to accomplish efficient drone delivery based on real data.

Below are the existing specifications for the Planet X Drone Fleet. This list is meant to serve as a set of examples. They are not exhaustive as we should get new models as the hardware develops. The Planet-X specification for the models are stored in the following CSV file (keep in mind that these specifications may change):
`repo/project/data/planet-x.csv`

Model #	Mass (kg)	Max Speed (km/h)	Base Acceleration (m/s^2)	WeightCapacity (kg)	Base Battery Capacity (seconds)
Q-36-01	50	60	4	10	300
Q-36-02	200	55	4.5	20	10
Q-36-03	5	40	5	10	30
Q-36-04	120	30	6	19	90
Q-36-05	150	60	3	15	120

Table 1 - Example drone pool specifications.

2.2 Map Data

For ANVIL 2.0, we will be using real roadway data from the OpenStreetMap (OSM) application to build a 3D grid of publicly available routes a drone can legally traverse. Figure 5 shows a screenshot of the web based tool where users can export street data to an OSM file (XML file format).

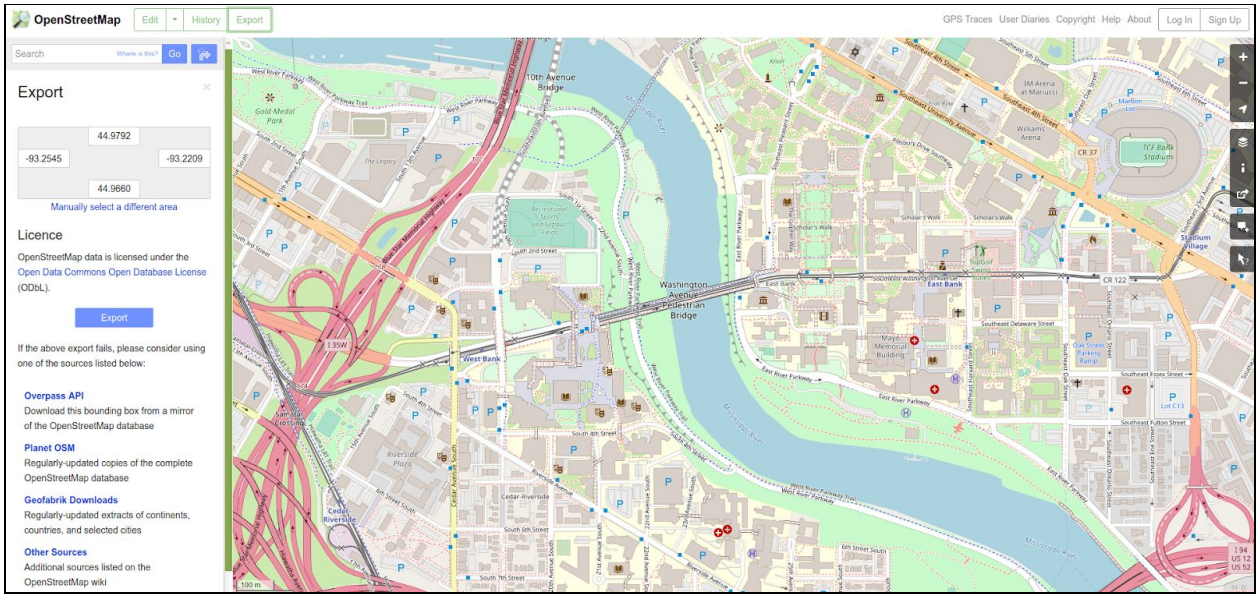


Figure 5 - The OpenStreetMap web based application.

We will also use the elevation data generated by retrieving a gtiff image from the OpenTopography API as the Shuttle Radar Topography Mission (SRTM) data. We then use a custom python script to smoothly interpolate between the height data points, converting the data into a grid of comma separated float values (CSV file).

Fortunately, the Rocket Visualization team has created a parser to help get the street graph information. This parser takes in an OSM file retrieved from the OpenStreetMap application and a height map file stored as a CSV (comma separated value) file. To convert the raw data files into a graph of nodes and edges, you can use the following code:

```
	entity_project::OSMGraphParser parser;
	const entity_project::IGraph* graph = parser.CreateGraph("data/umn.osm",
"data/umn-height.csv");
```

These files are needed for parsing the UMN data into a graph and are located at the following locations:

- OSM File:
  - **`repo/project/data/umn.osm`**
- Height File:
  - **`repo/project/data/umn-height.csv`**

2.3 Algorithms

In this iteration, there are several key algorithms that you may want to implement. In fact, the shortest path algorithm, Dijkstra, is required to efficiently plan routes in a large graph. Brute force approaches will not be feasible for a real time system.

Algorithm 2.3.1 - Dijkstra

Dijkstra's algorithm finds the shortest path between two points efficiently.

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

References

- https://www.youtube.com/watch?v=pVfj6mxhdMw
- https://www.youtube.com/watch?v=GazC3A4OQTE

3 Iteration 2: ANVIL Simulation Specification

The main goal of the ANVIL project ***Development*** phase, is for you and your teams to develop a working drone delivery simulation. We are looking for creative designs and implementations. This means that the project requirements are separated into two separate components, design (writing) and development (coding). The end goal of this iteration is as follows:

- **Design** - Clearly articulate your design decisions, so that developers and management can understand your design. Design is key to figuring out how to best move forward with iteration 3.
- **Development** - Implement the drone delivery system so that we have a working simulation to optimize in iteration 3.

All work on this iteration should be done in cooperation with your assigned team. You will make all changes in a shared repository on github labeled **repo-iter2-<class section>-<team number>**. All work should be done in this repository.

Automated feedback tests will be based on your `devel` branch , and automated assessments will be based on your `release` branch. **Your final score will be based on your `release` branch.** If your code does not compile on the release branch, you will get 0 points for the development functionality.

Success in Iteration 2

You will be successful in iteration 2 if your drones deliver packages from any point on the map to a customer anywhere on the map. The drones should follow the existing road and path system. In other words, drones should not go through buildings. Multiple drones should be delivering different sized packages at different speeds. Finally, the visual and other observers should be notified of package delivery events. This is the foundation of a useful drone delivery system.

Priorities

In order to succeed, we have here put together three priorities or “Core Goals” that should be part of this system. These core goals are necessary for Iteration 3, so they are of utmost importance. The priorities are:

- **Priority 1: Observer** - Clients can be notified of package delivery.
- **Priority 2: Dynamic Routing** - Use the shortest path on a graph for package delivery.
- **Priority 3: Drone Pool** - Use a diverse set of drones to deliver packages.

Once these three features are implemented, we will have the foundation we need for optimizing drone delivery in iteration 3. In Section 3.2 we detail the technical features that will enable each of these three goals. A finished product in iteration 2 will implement all three priorities and

include one or two additional features. It should also include several types of tests (Regression, Unit, and Integration. Requirements for these are listed in the Evaluation section) as well as bug fixes.

3.1 Design

In this iteration we are focusing on one design pattern that will be necessary for building a flexible drone delivery system: **Observer**. We are also implementing algorithms using graph theory to efficiently navigate a connected grid (**data structures and algorithms)**.

3.1.1 Overview of Changes

In order to achieve these design goals, there have been key changes to the support code. Figure 6 shows the updated UML diagram. Below the figure is a detailed list of changes.

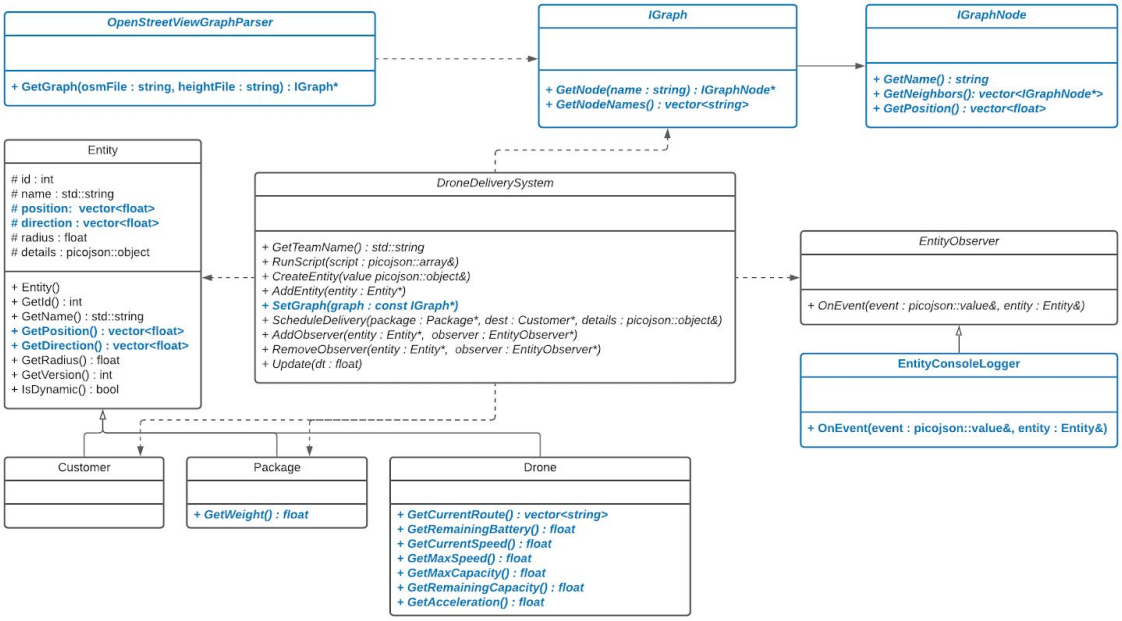


Figure 6 - Updated UML of the support code. New classes and methods are highlighted in **blue**.

- **Updated the Entity class** - Uses a `vector<float>` instead of a `float*` for position and direction to prevent the buffer overflow problem.
- **Updated the DroneDeliverySystem abstract class** - Added a `SetGraph(...)` method for graph traversal.
- **Added methods to the Package and Drone classes** - These methods allow us to get additional details about drone delivery (movement, capacity, etc…)

- **Created an IGraph interface (Includes IGraphNode)** - These are used to describe the road map for the system.
- **Created an OpenStreetViewGraphParser class** - This class allows us to parse the raw osm and height data into an IGraph class, which can then be added to DroneDeliverySystem.
- **EntityConsoleLogger** - This is a new observer that prints the JSON values to the console screen when the `OnEvent(...)` method is called from a subject.

3.1.2 Requirements

Below are the design requirements for Iteration 2.

Design Requirements	
ID	Description
Design-1	**Team: Updated Doxygen Documentation & UML** ● As a team, build Doxygen documentation, similar to iteration 1 for the team project. ● Include an updated UML diagram detailing the solution.
Design-2	**Individual: Peer Review** **(Due 11/9/2020 @ 11:59pm - Canvas Submission)** ● Review the design documents (from iteration 1) for exactly 2 of your Iteration Two team members. ● Share your design documents as a team (everyone should have access to each other's design docs). ● This deliverable includes 2 separate reviews describing your recommendations for improving the writing along with additional comments. ● Make sure everyone in your group gets two reviews (divide up the reviews fairly). ● Each review should be clear and concise with good feedback to help your team members improve their writing. ● Use complete sentences as a well written argument. ● Feedback should be helpful describing how the reviewer thinks the design document would need to change in order to do well on the iteration 1 Design Document rubric on Canvas. ● Ultimate question for this review: "How would you grade this assignment based on your understanding of the Iteration 1 design document rubric?" What feedback would you give? ● Upload to canvas and email the review to each team member.

Design-3	**Individual: Design Document** Part of this design document is an iteration 1 grade (1st bullet point) and part is part of your iteration 2 grade (2nd bullet point). ● Iteration 1 - Update the design document to include revisions based on the peer reviews from your teammates. (Note: We will grade your revisions for iteration 1). ● Iteration 2 - Add an additional Iteration 2 design section. Make it obvious via section headings or bolding the first sentence so we can find it. Make it at least 3 paragraphs and answer the following. ○ Add a short description describing one priority or feature that **you** implemented. ○ Add a short description describing one priority or feature that **a teammate** implemented. ○ Quickly discuss design trade offs.

Table 2 - Design Requirements

3.2 Development

ANVIL 2.0 is the simulation of a drone delivery system. Below we explain the development process as well as the functional requirements for the development part of iteration 2.

3.2.1 Using the Iteration 2 Support Code

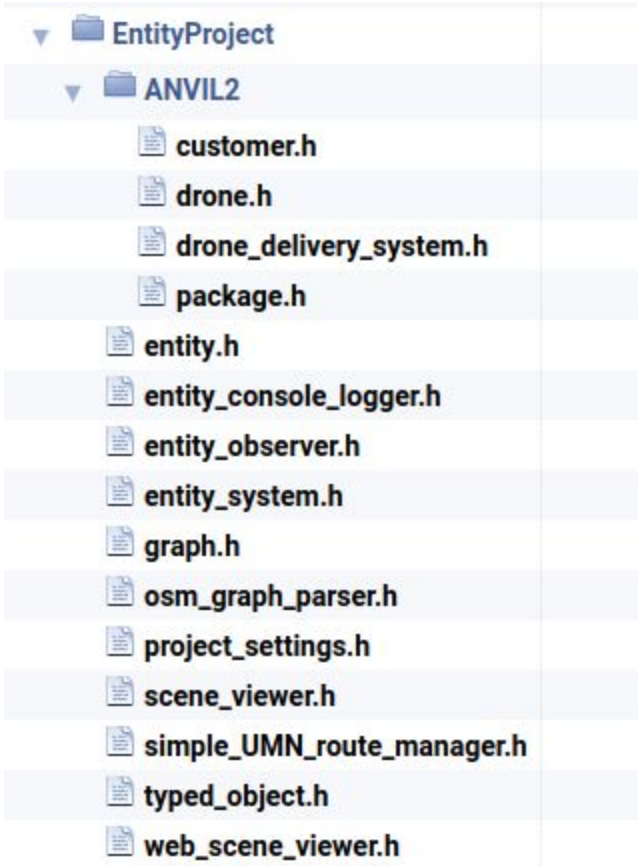
Here are a few links to help get you oriented to the new support code:

- Updated Support Code Documentation
- Support Code Header Files
- Project Portal WIKI

You will need to pull the latest from the `shared-upstream/support-code` repository/branch in order to start development. You will also need to build your docker environment to get the latest changes:

```
# Build the development environment (docker image)
./bin/build-env.sh
```

In order to maintain backward compatibility with Iteration 1, the new header files are located in the **`/project/grades/Fall-2020/csci3081/dependencies/include/EntityProject/ANVIL2`**
Here is a screenshot of the updated files:



3.2.2 GitHub Development Process

It is important to divide up the development work efficiently. Teams are required to implement the 3 priorities, however, individuals are required to do a total of 20 points of work (described in section 4 Evaluation). Since each priority and feature is worth 10 points, it is to the team's advantage to implement more features. Fortunately, each bug or test that is fixed is worth 1 point, so it is possible to not implement the additional features if one or two teammates want to focus on testing and fixing bugs.

In order to make this work, we will track individual work using GitHub issues to let us know what priority / feature / bug / test each person has implemented. **For specifically the priorities and features listed below, use the ID as the title of the GitHub Issue!** Refer to Lab 13 regarding the procedure on how to record issues, general naming conventions, and comments. Lab 13 uses the word *core* (under the note on individual grades) instead priority, but they refer to the same thing. The three Priority features for iteration 2 are:

- **Priority 1: Observer** - Clients can be notified of package delivery.
- **Priority 2: Dynamic Routing** - Use the shortest path on a graph for package delivery.
- **Priority 3: Drone Pool** - Use a diverse set of drones to deliver packages

3.2.3 Requirements

Below are the features that need to be developed for Iteration 2. Please do each of the priorities. Also, it is important to realize that not all features need to be implemented to succeed in this project. These are helpful if a developer has not yet achieved his or her individual 20 points.

<table>
<tr><th colspan="2">Priority 1: Observer</th></tr>
<tr><td>ID</td><td>Description</td></tr>
<tr><td>Priority-1-Obs</td><td>Observe Packages

We need to know when the package is originally scheduled, in route, and when it arrives at its destination. This is handled by the <code>AddObserver(...)</code> function.

We have already made observers for you: the WebSceneViewer and the EntityConsoleLogger. Your job is to implement the subject part of the Observer pattern.

When an observer observes a package, the subject should report an event to the observer by calling the <code>OnEvent(...)</code> method of the observer in the following situations (for an explanation of the observer pattern and examples, please review https://www.dofactory.com/net/observer-design-pattern):
● Package scheduled:
○ <code>observer->OnEvent({“type”: “notify”, “value”: “scheduled” }, package)</code>
● Package picked up by drone:
○ <code>observer->OnEvent({“type”: “notify”, “value”: “en route” }, package)</code>
● Package dropped off by drone:
○ <code>observer->OnEvent({“type”: “notify”, “value”: “delivered” }, package)</code>
● This notification will be displayed as an overlay on top of the visualization.
● Packages can have many observers.
● You should be able to remove any observers with the <code>removeObserver(...)</code> method.</td></tr>
<tr><th colspan="2">Priority 2: Dynamic Routing</th></tr>
</table>

ID	Description
Priority-2-Route	**Shortest Path Algorithm** Using a given IGraph, routes need to be calculated using the shortest path algorithm to efficiently move over a public road system. ● Instead of using the SimpleUMNRouteManager for route calculation, use your own implementation using the graph passed in through the `DroneDeliverySystem::SetGraph(...)` function. ● Routes are calculated using Dijkstra's shortest path algorithm between two points. ● Include the source and destination points as the first and last points on the route. ● The weights between each edge is the Euclidean Distance between the nodes. ● See references above concerning Dijkstra's search descriptions and videos.
Priority 3: Drone Pool & Selection Strategy	
ID	Description
Priority-3-Pool	**Drone Pool** Allow for many drones in your simulation and each drone can have different properties (weight, capacity, max speed, acceleration, etc…). See Table 1 in section 2.1. ● We will add a model number for each drone when creating it. An example of JSON passed into create entity is as follows: ○ `{ "type": "drone", "name": "T-800", "model":"Q36-01", "position": [0,0,0], "direction": [1,0,0] }` ● Set all the appropriate variables for your drone (notice that speed has been replaced with max speed and acceleration). ● Be sure to look up the specific model's traits based on the CSV file described in section 2.1. We encourage you to cache the results in a data structure so that you can look up the model attributes by model number when you create drones.
Additional Features	
ID	Description

Feature-1-Drone-Observer	**Drone Observer** We need to be notified of all drone path changes so we can analyze and keep track of the drones. When an observer observes a drone, the subject should report an event to the observer by calling the `OnEvent(...)` method of the observer in the following situations: ● Drone enters idle state (not moving): ○ `observer->OnEvent({"type": "notify", "value": "idle" }, drone)` ● Drone enters moving state: ○ `observer->OnEvent({"type": "notify", "value": "moving" "path":[[x0,y0,z0],[x1,y1,z1],...,[xn,yn,zn]]}, drone)` ● Drones can have many observers. ● You should be able to remove any observers with the `removeObserver(...)` method.
Feature-2-Drone-Functionality	**Drone Delivery Functionality** Drones now have the following realistic limitations: ● Battery life (seconds) ● Carrying capacity (kg) ● Maximum speed (meters/sec) Drones now behave differently based on the following rules: ● When a drone is moving, the battery life starts to drain. Once the battery life is gone, the drone switches to idle mode, and can no longer move. Packages are dropped when this happens and should be rescheduled. Rescheduling reports to the observer that the package has been scheduled again (see Core-Obs-1). ● Drones can carry multiple packages, but the sum of the package weights cannot exceed the drone's maximum carrying capacity. ● Drones cannot exceed their maximum speed. ● Drones enter idle mode if they are not moving. In idle mode, there is no battery drain because the drone is not moving. ● If a package is dropped, a new drone should be allocated to complete the delivery.
Feature-3-Schedule-Functionality	**Schedule Delivery Functionality** Packages can be scheduled at any time, can have priority (high, medium,

	low), and a minimum delivery time (seconds). ● `DroneDeliverySystem::ScheduleDelivery(...)` can be called any time during the simulation. ● Multiple packages can be scheduled. ● A package can only be scheduled for one customer at any given time. Ignore packages that are already scheduled. ● Once a package is delivered it is deleted and removed from the system. ● We will add details to `ScheduleDelivery(...)` to account for the additional features. An example of JSON passed into the method is as follows: ○ `ScheduleDelivery(package, customer, details = { "priority":"medium", "minDeliveryTime":60 })` ● In this iteration we will not use the priority or minimum delivery time, but you should store this value for future iterations.
Feature-4-Drone -Physics	**Drone Physics** The simulation should handle three separate physics models (velocity, acceleration, and force). ● **Note:** For these it is highly recommended that you use something like a Vector3D class instead of array manipulation. ● The physics model for drones is specified in the `CreateEntity(...)` method of the DroneDeliverySystem. ○ Example JSON: `{"type": "drone", "physics-model":"velocity"}` <u>Velocity</u> ● Use the max velocity as the speed for Euler integration of the position (just like iteration 1). ○ Position = Position + Direction*MaxSpeed*dt <u>Acceleration</u> ● Use Euler integration to calculate the new velocity: ○ Velocity = Direction*CurrentSpeed ○ Velocity = Velocity + Direction*Acceleration*dt ○ CurrentSpeed = |Velocity| ● Use Euler integration to calculate the new position: ○ Position = Position + Velocity*dt <u>Force</u> ● Definitions ○ F_drone = the amount of force the mode can use.

	○ M_drone = the mass of the drone without load. ○ M_total is the total mass of the drone and it's packages ○ BaseAcceleration = acceleration of drone without load ● First calculate the force used to move the drone at it's base acceleration. We can do this using F=ma (force = mass * acceleration). ○ F_drone = M_drone * BaseAcceleration ● Next calculate the acceleration using the total mass (with load) ○ Acceleration = F_drone / M_total ● Use Euler integration to calculate the new velocity: ○ Velocity = Direction*CurrentSpeed ○ Velocity = Velocity + Direction*Acceleration*dt ○ CurrentSpeed = |Velocity| ● Use Euler integration to calculate the new position: ○ Position = Position + Velocity*dt

Table 3 - Functional Requirements

4 Evaluation Criteria

In order to evaluate the success of the project we will score each prototype with the following criteria. 50% of the score will be design related and 50% will be development related.

We are evaluating the development part of this assignment based on both team and individual work. As a team, **you must implement all 3 priorities robustly (this is required for this iteration for the group grade)** and implement the three types of tests specified below. Additional implemented features will be graded if specified as part of the iteration by an individual team member.

The individual part looks at **who is assigned to each Priority or Feature in the GitHub issues**. It also looks at bugs and tests. Depending on the features developed, it will assign a point value, which should add up to 20 points max. This means individuals could get points any number of ways. An example is given below the rubric.

Design

- **Team (20%)**
  - 20 points - Updated Doxygen Documentation & UML
- **Individual (30%)**
  - 15 points - Peer Review: Iteration 1 Design Document (Monday, Nov 9)
    - See design feature “Design-2” in section 3.1.2 for more details.
  - 15 points - Design Document
    - Revise your design document based on peer reviews. (This revision will be applied to your iteration 1 grade).
    - Add an Iteration 2 design section discussing two features (yours and a teammate’s). See design feature “Design-3” in section 3.1.2 for more details.

Development

- **Team (30%)**
  - 15 points - Functional Requirements (80% of tests pass for full credit)
    - Program builds and runs without crashing for the basic scenarios.
    - Priority 1: Observers can be added and removed from packages.
    - Priority 2: Drones follow the shortest path on a graph.
    - Priority 3: A diverse set of drones are created.
    - Implemented Features: Tests will run based on what is implemented. These additional features are described in Section 3.2.
  - 10 points - Testing (At least 3 of each - we will spot check these! So make sure they are relevant tests to the project and cover different feature areas).
    - Regression Tests - Make sure your new changes do not break iteration 1.
    - Unit Tests - Test additional classes that are created for iteration 2.

        - Integration Tests - Test multiple classes interacting together.
    - 3 points - Robustness
        - Good management. No memory leaks, dangling pointers, segfaults.
        - Program handles incorrect input and undocumented scenarios (this means passing the robustness tests we run in your feedback and assessment).
    - 2 points - Code Quality
        - Follows Google Style Guide (Cpplint)
- **Individual (20%)**
    - 20 points max - Core Requirements / Strategies / Issues / Tests
        - 10 points each - Priority Requirements (e.g. Issue ID: Priority-1-Obs)
        - 10 points each - Features (e.g. Issue ID: Feature-1)
        - 1 point each (up to 5 points) - Additional Issues (e.g. Issue ID: Bug or Enhancement)
        - 1 point each - Test (up to 10 points) (e.g. Issue ID: Test)

Development Team Member Issues Example (This is just an example):

- Team Member 1
    - **10 points - Priority-1-Obs**
    - 10 points - Feature-1
- Team Member 2
    - **10 points - Priority-3-Pool**
    - 2 points - Two Bugs
    - 4 points - Four Unit Tests
    - 3 points - Three Integration Tests
    - 1 point - One Regression Test
- Team Member 3
    - 10 points - Feature-2
    - 10 points - Feature-3
- Team Member 4
    - **10 points - Priority-2-Route**
    - 5 points - Five Bug**s**
    - 2 points - Two Regression Tests
    - 3 points - Three Integration Test

Issue Naming Details:

- For core features (i.e. Priority-1 or Feature-1), just start the issue name with the prefix ID of the core feature. Make sure to **assign yourself to the issue to get points**.
    - Examples:
        a. Priority-1-<any description>
        b. Priority-1-Obs
        c. Priority-3-This is another description
        d. Feature-1-Drone-Observer
        e. Feature-1-another description
        f. ….etc
- For Tests, the issue must be of the form `Test-<testName>-<typeOfTest>` where `Test` is the literal word, `<testName>` is the name of your google test and finally `<typeOfTest>` is either integration, unit or regression. Make sure to **assign yourself to the issue to get points**
    - Examples:
        - Test-PackageMovesCorrectly-unit
        - Test-DroneDeliversPackage-Integration
        - Test-AssignPackageToDrone-REGRESSION
        - Mainly your tests are in the form Test-ExampleTestName-[unit/integration/regression]
        - Etc…
- For Bugs or Enhancements, the issue name must contain the “bug” or “enhancement” or labeled as “bug” or “enhancement”. Make sure to **assign yourself to the issue to get points**