

# Project ANVIL - Spec 1.2

Aerodynamic Navigational Vehicles for Instantaneous Locomotion



Our plan to crush the competition...

ACME Corporation  
September 21, 2020

Authors: CSCI 3081 Staff

## Iteration 1: ANVIL Simulation Prototype

**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:

- 1.2 - 10-15-2020 - CSCI 3081 Staff - Removed observer from required features.
  - Updated 2nd Deliverable Date to 10/23/20.
  - Moved observer to "Optional Feature - Package Observer" section.
  - Removed lab assignment on (10/23/20) since observer is no longer required.
- 1.1 - 10-08-2020 - CSCI 3081 Staff - Added teamwork documentation deliverable.
  - Added requirement 1.4 (Section 1. Design)
  - The Design Documents is now worth 20 points. (Evaluation Criteria)
  - Added Teamwork Documentation worth 5 points. (Evaluation Criteria)
- 1.0 - 09-21-2020 - CSCI 3081 Staff - Initial ANVIL specification release.

# Introduction and Welcome

Welcome to a team of elite developers! ACME means peak, zenith, prime, pinnacle, or simply “the best” ([ACME](#)). Therefore, our motto as a company has always been “Quality is our number one dream!” In fact, we even protect and defend our unique reputation as *the leader in creative mayhem*. Most of all, we are proud of our instantaneous delivery services that have made us famous for decades. No one delivers such an extensive and unique catalog as quickly as we do.

As the new CEO of the ACME Corporation, I am excited to announce our initiative to extend these delivery services to the third dimension. Project ANVIL, our proposed drone delivery system, will revolutionize the package delivery industry. This disruptive technology has the potential to change the fulfillment process and crush the competition (with an anvil). I want to encourage each of you to think creatively so we can solve these difficult problems together. As one who is very familiar with our product line, I encourage you to experiment, but also be careful with new technologies. We can all learn from our mistakes. I have made many mistakes (see my related work in the Background section of this document), but I am excited for the future.

This project is very organic as we will be continuously building on top of our best solutions. You will be assigned a team to work with. Over three project iterations, we will be noticing creative and optimal solutions across development teams. The top 5 teams, or the “High Five” will be recognized at the end each iteration. The High Five will be determined based on exceptional work and commitment to going above and beyond the specification. Remember that you are competing against other teams, so be sure not to share coding secrets during development.

Good luck and have fun!

*Wile E. Coyote*

Wile E. Coyote, CEO

# 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 1** - 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 1 - the ANVIL Simulation Prototype**. We start out with a brief background and motivation. This is followed by the detailed specification for iteration 1. Then we provide evaluation criteria for what we believe would be a successful prototype. This is followed by the proposed project timeline based on internal estimates. A

prototype not only evaluates our basic assumptions, but helps us continue to improve our analysis.

The aim is to use the best prototypes developed as a starting place for understanding Iteration 2. Therefore, we want to encourage you to explore beyond the requirements, so our final section includes extensions for achieving “High Five” (top 5 teams) status.

## Document Organization:

- Background
- Iteration 1: ANVIL Simulation Prototype Specification
- Evaluation Criteria
- Project Timeline
- High Five Extensions (Above and beyond to foster creativity, fun, and team competition)
- Optional Feature - Package Observer

## Background

The ACME Corporation was founded on three principles:

1. Deliver as fast as possible!
2. Deliver anything that a customer orders, no matter what it is.
3. Deliver anywhere we can.

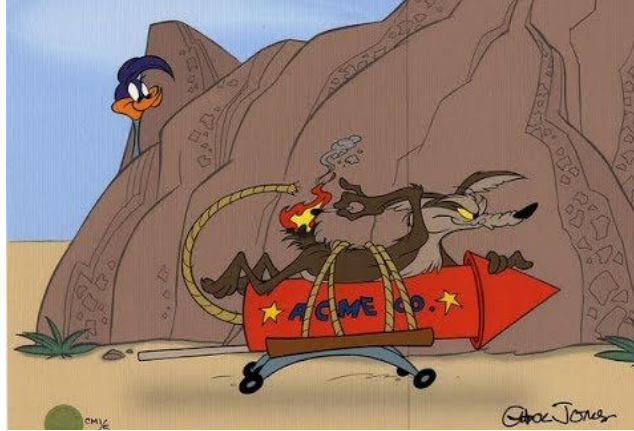
This usually means being creative and trying new things. We started in Fairfield, New Jersey as ACME Rocket Powered Products, Inc. selling rocket powered roller skates (see Figure 2). In fact, our CEO, Wile E. Coyote was one of the first beta testers (See Figure 4. Note: This is one of the primary reasons our company is focused on testing). As demand increased for new exciting products, we quickly expanded to include items from jet-propelled tennis shoes, tornado kits, costumes, and earthquake pills (see Figure 3).



Figure 2 - Our first product



Figure 3- Unique product line



**Figure 4** - Our CEO started out as a “valued” customer and beta tester.

Now our company is ready to explore new technologies that link customers directly together. Delivering from customer to customer has now become an exciting option for package fulfillment. We have separate divisions working on three parallel projects related to the drone delivery system:

- **Project ANVIL** - Simulation
- **Project Rocket** - Visualization
- **Project Jet Powered** - Ecommerce

This document details the specification for **Project ANVIL**, but it may be necessary to interface with the other two applications as they are developed. At this point all you need to know about **Project Jet Powered** is that there is a web-based ecommerce application that keeps track of our product inventory and allows customers to order and schedule deliveries from one location to another. **Project Rocket**, is the visualization system that Project ANVIL relies on for real time analysis of the simulation. This visualization system is a work in progress and will improve as the project progresses. Eventually these systems will work together seamlessly.

## Project Rocket - Visualization

Project Rocket has been in development for about two months and has made progress towards a simple 3D drone visualization system. The test location for our initial system is at the University of Minnesota (see Figure 5). No rationale was given for the chosen location (Note: it is likely that Minnesota was chosen because it is too cold for roadrunner spies from Roadrunner Technologies, our number one competitor. In addition, MN is the opposite of NM - New Mexico, where roadrunners are often found). This is your test environment for working on the project simulation. Customers are currently represented as expressive robots (see Figure 6a). Drones (see Figure 6b) move packages (See Figure 6c) around as the simulation progresses.





**Figure 5** - 3D Visualization of the University of Minnesota



**Figure 6**

- Free models that were used in the visualization:
  - <https://github.com/mrdoob/three.js/tree/dev/examples/models/gltf/RobotExpressive>
  - [S9 Mini Drone - Download Free 3D model by AzurPoly \(@VapTor\) \[cf3ed83\]](#)
  - [Simple Low Poly Cardboard Box - Download Free 3D model by ShadowIgnition \(@ShadowIgnition\) \[1faa95a\]](#)

In order to take advantage of the latest visualization code (built on a docker image <https://hub.docker.com/r/csci3081f20/base>), it may be necessary to pull and build the latest docker base image with the following commands:

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

Any changes will be announced and will reference changes in this document. To run the visualization, which starts the simulation, you will run the following commands:

```
# Start the docker container (this will open up a bash inside)
./bin/run-env.sh
# Navigate to the project directory
cd project
# Build the simulation and the visualization
make
# Run the visualization, which starts the simulation
./bin/run.sh
# Navigate to the following link on your browser (e.g. Firefox)
http://127.0.0.1:8081
```

## Customers and Routes

The Rocket team has provided a simple interface that provides routes between 4 different customers defined below (see Figure 7):

1. Walter Library
2. Carlson
3. Alumni Center
4. Mariucci Arena



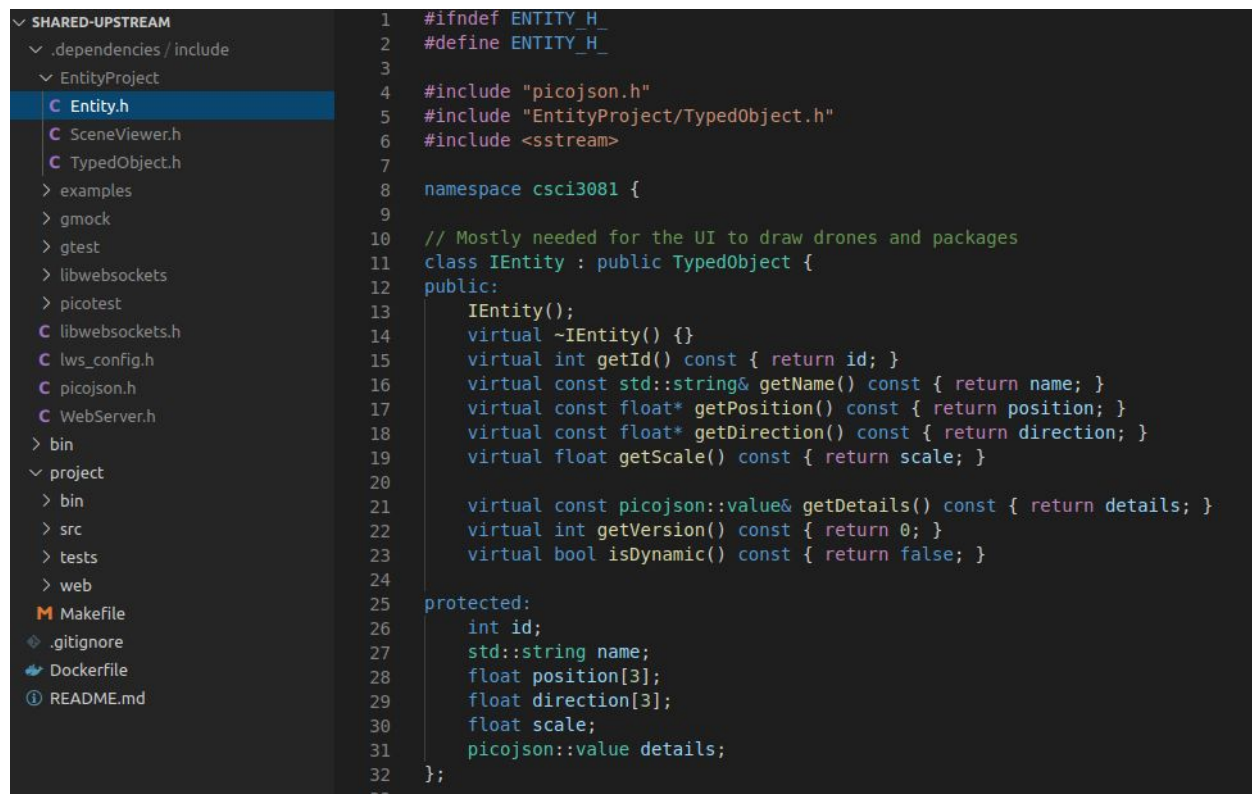
**Figure 7** - Initial customers

We can expect that more customers will be provided in future iterations, so keep this in mind. In order to get routes between customers that follow the streets or paths (so drones do not run into buildings), the Rocket team has also built a simple interface into the docker image you can use. Here is code that will give you routes between different customers:

```
#include "EntityProject/simple_UMN_route_manager.h"

...
SimpleUMNRouteManager manager;
std::vector<SimpleUMNRoutePoint> points = manager.getRoute(src, dest);
...
```

Project Rocket is calling their more generic system **EntityProject** and you can find all the relevant header files in your repository's `.dependencies/include` directory after running `./bin/build-env.sh`. For example when using VSCode, your project should look like this with all the dependencies in the `.dependencies/include`. Notice the visualization uses web-based technologies like web sockets, web server, and json libraries. You should not need to touch web related code, but you will need to become familiar with the **EntityProject** and **picojson** (<https://github.com/kazuho/picojson>) so that you can interact with the visualization.



```
1  #ifndef ENTITY_H_
2  #define ENTITY_H_
3
4  #include "picojson.h"
5  #include "EntityProject/TypedObject.h"
6  #include <sstream>
7
8  namespace csci3081 {
9
10 // Mostly needed for the UI to draw drones and packages
11 class IEntity : public TypedObject {
12 public:
13     IEntity();
14     virtual ~IEntity() {}
15     virtual int getId() const { return id; }
16     virtual const std::string& getName() const { return name; }
17     virtual const float* getPosition() const { return position; }
18     virtual const float* getDirection() const { return direction; }
19     virtual float getScale() const { return scale; }
20
21     virtual const picojson::value& getDetails() const { return details; }
22     virtual int getVersion() const { return 0; }
23     virtual bool isDynamic() const { return false; }
24
25 protected:
26     int id;
27     std::string name;
28     float position[3];
29     float direction[3];
30     float scale;
31     picojson::value details;
32 };
33
```

**Figure 8** - The base code structure for the visualization / simulation.



# Iteration 1: ANVIL Simulation Prototype Specification

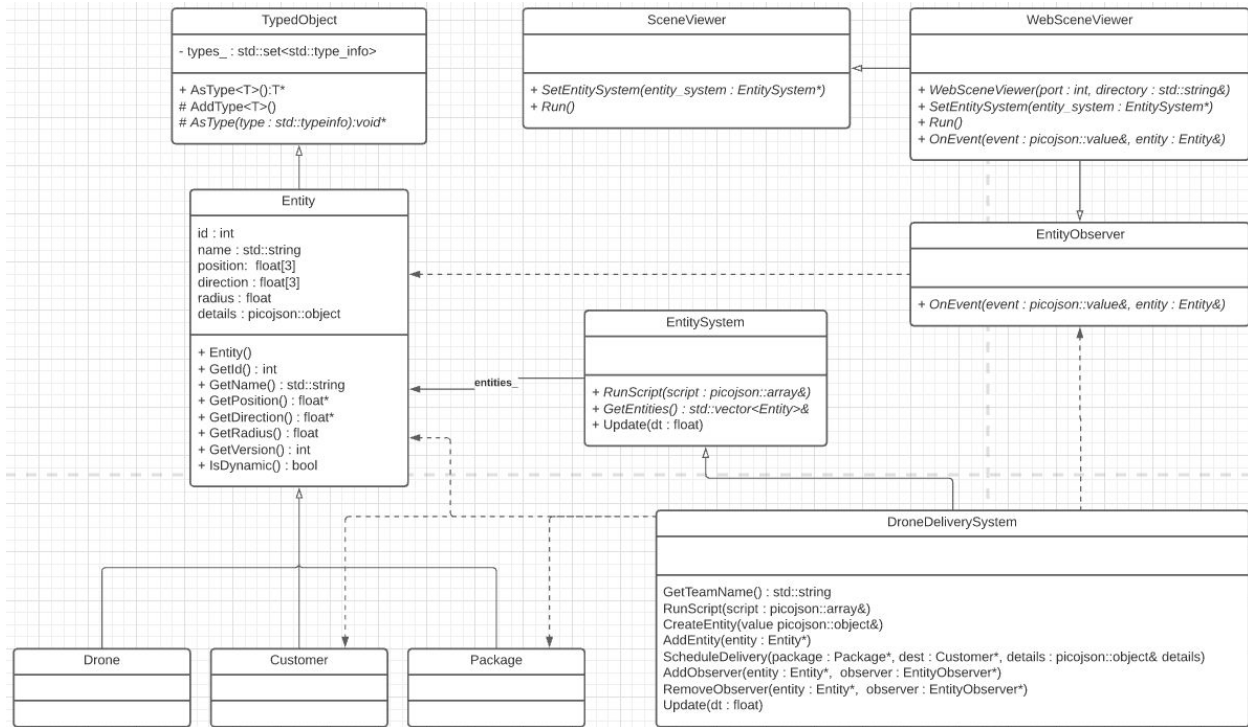
As developers of the Project ANVIL prototype phase, our main goal is for each of you individually to prototype the system to see if such a simulation is even possible. What we are looking for is creative designs and implementations. This means that the project requirements are separated into two separate 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. We realize that with experimentation, these are key in figuring out how to best move forward with iteration 2.
- **Development** - Implement something that works, so we can prove that our design decisions are sufficient for moving forward to iteration 2.

All work on this iteration can be done in cooperation with your assigned team. Remember you are competing against other teams, so don't share design decisions with anyone else on other teams. Also, since we are looking for different designs and implementations to optimize our use of prototype time, individuals within a team should not simply copy from each other. Rather, you should use your teammates' design ideas as inspiration for your own design choices. The goal in this iteration is to come up with as many different solutions as possible so we have many to pick from in iteration 2. Therefore, the best designs will rise to the top, rather than being forced into one design early on. That being said, it is understandable if teams share general design decisions and help each other fix issues. Code itself should not be shared and everyone should do their own work in this iteration. Another reason this is true, is so that everyone has a basic understanding of how to code the simulation so they will be more helpful in future iterations.

## 1. Design

In this iteration we are focusing on three different design patterns that we think will be helpful for building a flexible drone delivery system: **Facade**, **Factory**, and **Observer**. Here is a UML diagram of the base code structure:



**Figure 9 - Base framework structure UML**

The DroneDeliverySystem is a Facade interface that allows us to evaluate your prototype in a consistent way. It is also helpful for enabling the visualization (Please see <https://www.dofactory.com/net/facade-design-pattern> for a description of the Facade pattern and examples). Behind the facade we don't care how you implement the system to make it work, but we are interested in your solutions (your design decisions). We think that a Factory behind the createEntity(...) method may help, but we are also interested in whether you think it is useful or not and your thoughts here. Finally, we think an observer is necessary for the package so that we can know when it is scheduled, when it is in route, and when it arrives. The following writing portions of this iteration are detailed below:

1.1 UML - The first deliverable is a UML document depicting the design of the system. This should be implemented using [Lucidchart](https://www.lucidchart.com) and submitted at the end of the first week of the iteration (3 weeks long). Turning it in early will help us determine whether or not to redirect development.

1.2. Doxygen - The second deliverable will be a fully documented interface.

- A. All classes and methods should be documented and built using Doxygen.
- B. Developer documentation including descriptions on how the code works and how to get started.

1.3 Design Document - This is a document that describes all design decisions in the system including how you implemented the delivery system behind the facade. You should include an updated UML based on changes since the 1.1 deliverable. Be sure to include a discussion of alternative designs especially related to the Facade, Factory, and **Observer**. We are especially interested in these (Please see the UMN Center for Writing for additional writing resources: <http://writing.umn.edu/>).

1.4 Teamwork Documentation - This document describes how a team worked together with respect to assigned team roles. Management would like to understand how teams work together. This document should be easy to prepare and can be written as a team (e.g. using a shared Google Document with screenshots, conversations, and raw data - meeting times, etc...). Here is the ultimate question we are hoping to answer: Is there enough team documentation to prove that you have actively and effectively executed your assigned role over the course of the iteration. Here are recommended practices:

- The **scheduler** role should initiate the team discussion at the start of each iteration.
- You may change role assignments if you like (but include the changes in the document)
- Meet as a team at least once a week.
- Use the Lab time as a time to meet as a group and with TAs.
- Setup a project space to discuss development and tasks. (We recommend creating a Slack project and using Discord or Zoom for meetings).
- We recommend using Google Documents for shared documentation.
- Make a development plan early on and shoot to meet deadlines.
- Document your contributions to the team (which may include helping others).

Here are the roles that are assigned when you get your Iteration One teams:

<b><u>Role</u></b>	<b><u>Description</u></b>
Development Lead	Makes all the major decisions for design and implementation. Not everything needs to be the development lead's idea, but this person should decide if there are competing design ideas. For Iteration One (since it is an individual project), the Development Lead role is to help provide a shared direction.
<b>Scheduler</b>	In charge of scheduling team meetings. (The When2Meet application is great for this: <a href="https://www.when2meet.com/">https://www.when2meet.com/</a> ). The scheduler is also in charge of setting up shared discussion environments like (Slack, Discord, or Google Documents).
Reporter	In charge of group work and discussion documentation (turned in with Iteration 1). This person should divide up the group documentation work (meeting minutes, design discussions, project calendar / schedule information, etc...).
Project Manager	Keeps team members on track so that group and individual deadlines are being met. Manages the project timeline and plan (to be submitted with the documentation).

## 2. Development

Implementation in the prototype phase is important for developing a proof of concept. Here is an overview of what we are proposing:

1. When the application is run, the visualization loads and starts the simulation. This should be available with the base code.
2. Entities are created based on the `CreateEntity(...)` method within the facade. This includes all drones, packages, and customers.
3. The created entities are added to the system within the `AddEntity(...)` function within the facade.
4. Once entities are added, they are updated with each time delta `dt` inside the `Update(float dt)` function within the facade. This means drones and packages move within this function.
5. Packages are scheduled for delivery within the `ScheduleDelivery(...)` function within the facade. Once scheduled, drones can pick up packages and deliver them from the source customer to the destination customer. The package moves with the drone.
6. Finally, we need to know when the package is originally scheduled, in route, and when it arrives at its destination. This is handled by the `AddObserver(...)` function. The `EntityObserver` class is defined in `EntityProject/entity.h`.

This above is the basic flow of the system. Below are the business requirements for the actual drone simulation:

- Note: Entity inherits from `TypedObject` which has a really great method called `AsType<T>()`. This allows you to down cast in a more efficient way than using `dynamic_cast<>()`. `dynamic_cast` can be slow, but the `AsType<T>()` uses static casting instead to avoid any reflection. This is necessary for real-time simulation.
- In order to change any entity into your own types, you will need to do the following:
  - Call `AddType<MyDrone>()` inside the constructor. (see `IDrone`).
  - Then anywhere you can call `MyDrone* drone = entity.AsType<MyDrone>()`. If it is not the type you want, it will return `NULL`.

2.1 - The facade interface **`drone_delivery_system.h`** **should not be changed**. This is used by several testing systems including the visualization system. Changes to this file could be problematic and error prone. That being said, you may create any number of classes behind the facade and use them however you need. You have complete freedom in how you actually implement the system. To do this, you will need to inherit from the `DroneDeliverySystem` and override the virtual methods. Add your facade class to `drone_delivery_system.cc` in the

`GetDroneDeliverySystem(...)` method. Return your team name in `GetTeamName()` method inside the facade.

2.2 - Drones, Packages, and Customers should be created inside the `CreateEntity(...)` method. Implement this method using a factory pattern. Drone, Package, and Customer (defined in `drone_delivery_system.h`) should be the base classes for your own drones, packages, and customers. This allows other systems (e.g. the visualization system) to know what type of object is being created without knowing about your objects (Please see <https://www.dofactory.com/net/abstract-factory-design-pattern> for a description and example of a the abstract factory pattern, though there are other types of factories).

- The JSON structure for drones, packages, and customers are illustrated in the examples below (for more information about JSON please review the JSON tutorial here: <https://www.tutorialspoint.com/json/index.htm>. Remember also we are using the picojson library for the C++ implementation documented here: <https://github.com/kazuho/picojson>):
  - Drone:
    - { "type": "drone", "name": "drone", "position": [0,0,0], "direction": [1,0,0], "speed": 1.0, "radius": 1.0 }
  - Package:
    - { "type": "package", "name": "package", "position": [0,0,0], "direction": [1,0,0], "radius": 1.0 }
  - Customer:
    - { "type": "customer", "name": "customer", "position": [0,0,0], "direction": [1,0,0], "radius": 1.0 }
- Be sure to set the details variable in Entity so that the visualization can use it for rendering. The values should be the same as passed in from the `CreateEntity(...)` method.

2.2 - Drones should be able to move from point A to point B at a constant speed. You can use the following formula for calculating movement for the prototype:

- Pseudocode: `position = position + direction*speed*dt`
- Note: position and direction are stored in the `IEntity` class interface in `Entity.h`.

2.3 - When a package is scheduled for delivery, a drone should navigate to the package, pick up the package, and bring it to its destination. The drone should follow the route from the source to the destination based on the path defined in `SimpleUMNRouteManager`. The drone should never exceed the constant speed for this iteration.

2.4 - When a drone's radius is within the radius of the package it will pick it up. When a drone's radius is within the radius of a destination customer, it will drop off the package.



~~2.5--(Moved to: Optional Feature - Package Observer) When an observer observes a package, the system...~~

~~2.6--(Moved to: Optional Feature - Package Observer) Packages can have many observers.~~

~~2.7--(Moved to" Optional Feature - Package Observer) You should be able to remove any observers with the removeObserver(...) method.~~

2.8 - Key classes should have unit tests that test every method (especially the facade, factory, and observer related classes).

2.9 - Code should be well documented and follow style guidelines (e.g. doxygen and cpplint).

# 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:

## Design

- 10 points - Initial UML Design
  - Quality Design
  - Correct use of UML
- 15 points - Doxygen Documentation
  - All classes and methods are documented. Quality naming for classes, methods, and variables.
  - Quality developer documentation to orient developers to the project and design. This includes adding additional pages.
    - It is okay to copy or include the design document, but it is important to include code specific details (getting started, details about key classes, how to build and run, etc...).
    - An updated UML Design that shows the final design that was used.
- 20 points - Design Document (Uploaded on Canvas)
  - Effective introduction discussing the project and purpose of the document. This includes references to related work.
  - Well written and clearly articulated overview of prototype design.
  - Detailed description of the **three two** design patterns and how they were used in this prototype. What design decisions were made and why?
  - A discussion of alternative designs for the **three two** patterns, comparing and contrasting the trade offs. Include lessons learned and potential improvements to the chosen design.
  - A conclusion that quickly summarizes results and provides insights into future development, especially how this document relates to iteration 2.
  - Use UML to show the design decisions that were made throughout the document.
- 5 points - Teamwork Documentation (Uploaded on Canvas)
  - There is enough documented evidence to show that the team member was an active and effective participant in their role.
  - Describes role assignments and team organization - including platforms (slack, discord, email, Google Docs, etc...)
  - Provides overview of team activities (screenshots of meetings with timestamp and attendance / important group conversations, etc...).

## Development

- 10 points - Testing
  - Tests for the drone delivery system facade.
  - Unit Tests for 5 key classes.
- 25 points - Functional Requirements
  - Program builds and runs without crashing for the basic scenarios.
  - Drones, customers, and packages are created correctly.
  - A drone can deliver a package to a customer following the correct route in a reasonable time.
  - ~~○ Observers can be added and removed from packages.~~
  - ~~○ Observers are notified of package events as defined in the specification.~~
- 10 points - Robust Implementation
  - Good management. No memory leaks, dangling pointers, segfaults.
  - Program handles incorrect input and undocumented scenarios.
- 5 points - Code quality
  - Follows Google Style Guide (cpplint)

# Project Timeline

Dates & Deadlines	Item	Description
9/22/2020 (Tuesday)	Initial Iteration 1 Specification	A release of the business requirements for iteration 1. At this point, design can begin.
10/1/2020 (Thursday)	Support code released	At this point, development can begin.
<b>10/9/2020</b> <b>(Due: Friday 11:55pm)</b>	First Deliverable: Initial UML Design	Business requirement 1.1. An initial design of the UML.
<b>10/23/2020</b> <del>10/16/2020</del> <b>(Due: Friday 11:55pm)</b>	Second Deliverable: Doxygen Documentation of Interface	Business requirement 1.2.A. Documentation with all classes and methods. Code does not have to work. This includes all functionality (facade, factory, and schedule / delivery) <del>except the observer</del> (Observer is no longer required)-
<del>10/23/2020</del> <del>(Friday)</del>	<del>Project Related Lab Assignment</del> <del>(Observer pattern)</del>	<del>This lab will help with the project, and will be turned in with the final deliverable of iteration 1.</del>
10/27/2020 (Tuesday)	Extended automated feedback	We will run our extended tests once on each codebase. This will help you fix any issues you might have with our grading scripts.
10/29/2020 (Thursday)	Extended automated feedback	We will run our extended tests once on each codebase. This will help you fix any issues you might have with our grading scripts.
<b>10/30/2020</b> <b>(Due: Friday 11:55pm)</b>	Final Deliverable: Iteration 1	All remaining business requirements. This includes the design document.

# High Five Extensions

**Note:** the High Five Extensions are **optional**, but the **top 5 teams will receive extra credit**. 1st place will receive **10 extra credit points**. 2nd-5th will receive **5 extra credit points**. It is just fun to build exciting systems and compete with others.



**Figure 11** - The High Five represent the top team scores for the iteration. This is like old arcade games or the Ready Player One movie.

During the project, we will be continuously evaluating the success of each team based on the progress of individuals' collective points (sum of all scores based on a group of 4. If you are in a group of three, the fourth person will be an average). In order to reach the High Five, you will need to compete with other teams to both fulfill the basic requirements described in the specification, but also fulfill other goals.

The score is calculated every time a team member checks into github.

You can check the current status of the High Five at any time at the following url:

<https://github.umn.edu/umn-csci-3081-f20/project-portal>.



Below are things we hope to implement in future project iterations. Remember, some of these may help you in future iterations, so it is good to prototype them now:

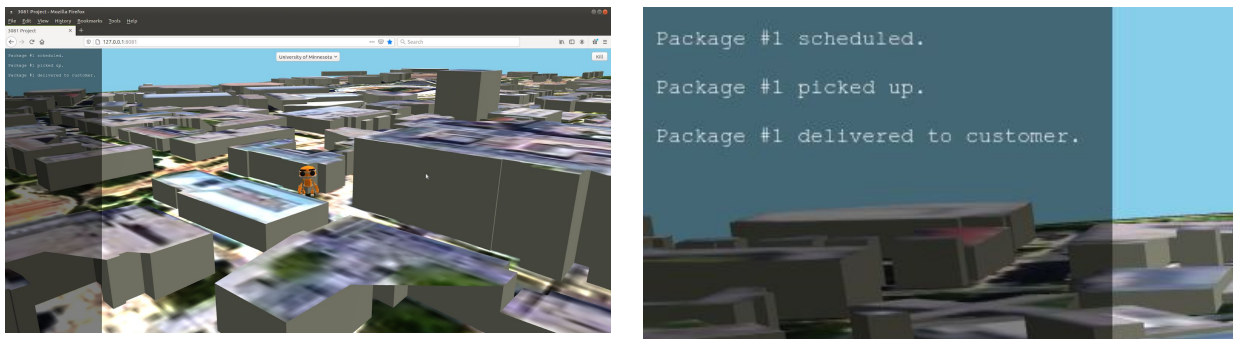
- One drone delivers multiple packages.
- If multiple drones exist and one package, only one drone delivers the package.
- Multiple drones simultaneously deliver multiple packages.
- Implement collision avoidance so that drones do not collide with each other. Have one increase or decrease it's height to avoid collision.
- Write a `EntityEventOutput` that implements the `EntityObserver` and outputs events to the command line when they happen.
- Write a `EntityEventFileWriter` that implements the `EntityObserver` and outputs events to a file specified through the constructor.
- Write a `EntityEventLogger` that implements the `EntityObserver`. It contains a method named `AddObserver(Observer* observer)`. It should log any events to all observers that are added.
- Allow observers to be added to drones and customers that report the following situations:
  - Drone moved:
    - `observer->OnEvent({"type": "movement" }, drone)`
  - Drone en route to package:
    - `observer->OnEvent({"type": "notify", "value": "en route" }, drone)`
  - Drone status set to delivering package:
    - `observer->OnEvent({"type": "notify", "value": "delivering" }, drone)`
  - Drone completed delivery of package:
    - `observer->OnEvent({"type": "notify", "value": "delivered" }, drone)`
  - Customer waiting for package:
    - `observer->OnEvent({"type": "notify", "value": "waiting" }, package)`
  - Customer received package:
    - `observer->OnEvent({"type": "notify", "value": "received" }, package)`
- Change the mesh of the drone, customer, or package by editing the "mesh" property in the JSON (The load script is located at `web/scenes/umn.json`):
  - `{ "type": "drone", "position": [0,0,0], "direction": [1,0,0], "speed": 1.0, "mesh": "models/myModel.obj" }`
- Add delayed scheduling by adding a delay within the JSON for scheduling a package:
  - For example:
    - `ScheduleDelievery(package, dest, { "delay": "30.0" })`
  - This means that the package is not available until 30 seconds into the simulation.
- **More High Five options are possible...**

- If you want to propose additional High Five extensions, please submit to the following form and we can try to add them for this iteration or for future iterations:
- High Five Extension Form: <https://forms.gle/KZKg9aEQkeiV8efj6>

## Optional Feature - Package Observer

2.5 - When an observer observes a package, the system will report an event to the observer by calling the `OnEvent(...)` 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:
  - `observer->OnEvent({"type": "notify", "value": "scheduled"}, package)`
- Package picked up by drone:
  - `observer->OnEvent({"type": "notify", "value": "en route"}, package)`
- Package dropped off by drone:
  - `observer->OnEvent({"type": "notify", "value": "delivered"}, package)`
- This will notification will be displayed as an overlay on top of the visualization.



**Figure 10** - Overlay for observer showing package delivery information.

2.6 - Packages can have many observers.

2.7 - You should be able to remove any observers with the `removeObserver(...)` method.