**Design Proposal**

**Project Description**

This project is an “Air Traffic Control Simulator” that simulates the real-world air traffic control radars towers use to manage flights near airports. As a simulator it mimics the verbal commands controllers use to direct aircraft while having the game aspect of having objectives of directing arrivals and departures.

**Competitive Analysis**

There are instances of air traffic control games on the app store which are very basic “drawing games” where the user draws a path to the objective runway. This project differs to this game in that it tries to simulate real world procedures through a more complex command handling and execution model. Other instances of online projects include complex air traffic control simulators which lets users select an airport from a list of four or five. This project however uses intelligent airport and map generation to create different cases of the map each run. This helps create more variety and randomness to the game to help the user get better at “air traffic control” not the airport itself.

**Structural Plan**

**Files**

The project is divided into a main file (atc\_simulator.py), the classes file (objects.py), the generation files (flight\_generation.py, map\_generation.py, airport\_generation.py), the command handling and execution module (commands.py), and the data files which handle airline and aircraft data (airline\_data.py, aircraft\_data.py).

**Object Oriented Programming**

The project itself is highly dependent on objects that are set in the objects.py file. These objects include Flights (which contain subclasses of Departures and Arrivals), Airport, Runways, Airline, Aircraft, Weather. These objects help the generation features of the project to have dependencies on one another based on parameters of the object. For example, weather parameters affect airports and flights, airlines affect flight routes, airport size affects the number of runways or overall traffic.

**Functions**

Most of the app related functions are contained in the main file where it will call external functions and objects from other files. Also, many functions are object specific to simplify the structure where they are stored as methods for individual actions. With this structure it is easier to add features without them negatively affecting the game or other objects.

**Algorithmic Plan**

Two important features of this project are its command handling and execution as well as the intelligent map generation.

**Command Handling and Execution**

For the command handling, this is done with string searching. With a list of keywords that will recognize a certain action to be made, the game scans the command string inputted by the user for the keywords and then the parameters corresponding to the keyword. For example, in the command “Delta 123 turn right heading 220”, there are 2 major components of the command. The callsign, which is used to identify the flight, and the heading command where it directs to change its direction to a magnetic heading of 220 degrees. For most English sentence structures, it is normal for a command to have the numerical parameter ahead of the actual keyword. In this case the number 220 is ahead of the word heading. The program uses this linguistic feature to simplify finding the parameter.

Also, the command handling model is able to identify both verbal and technical expressions of the callsign. For example, “DAL123” or “DAL 123” (with spaces) which represents the flight number, will be treated the same as “Delta 123.” This is possible due to the projects overall structure. The project is built upon objects such as Flights, Airports, Airlines, Aircraft, Runways, etc. With airline data taken from real world data. The flight numbers are able to be converted into its airline name counterpart and vice versa.

**Map and Weather Generation**

**Airport and Runway Generation**

For map generation, the components of the map that need to be generated include the airport, which is simply a placeholder for where the runways will be, the runways, and the waypoints. Airports are generated by a random four-letter string generator which will give its airport code. With the airport position centered on the map, the runway positions are generated by creating a normal vector of random magnitude from the airport position and setting the runway as a normal line to that vector. This creates runways that mostly stay within airport range and airport heading constrictions.

**Flight Generation**

As mentioned above, this project utilizes a great number of different objects. Flights objects consist of Depa rtures and Arrivals which have different methods and parameters. To generate flights, random airlines, aircraft types are taken from existing data, as well as random numbers for flight numbers. The airline and type that was generated affects the flights overall generation process. The flights generate routes that from or to the game generated airport by calling an airport from a list of the airline’s most operated airports and will have size and fuel parameters based on the aircraft type. This allows for size, fuel constraints for airports and the flight itself. This is possible due to the airline and aircraft objects that are set within the game which contain individual parameters.

**Timeline Plan**

This project has been able to progress quickly with features set by the MVP and other features that overall increase the realism of the simulator. As of April 23, most of the command handling and execution module has been completed with tweaks to the typo correction under work. Intelligent generation of the airport, runways, flights, waypoints have been completed with preparation for future implementation of local weather systems. Constraints have also been modeled into the game though resulting game states are still a work in progress. With most of the game currently solid, I have started working on a weather generation algorithm that will change states of the airport, flights, and runways themselves.

**Version Control Plan -** [**https://github.com/ay0503/Air-Traffic-Control-Simulator**](https://github.com/ay0503/Air-Traffic-Control-Simulator)

Using a personal repository on **Github** I am able to make commits to changes I’ve made. Also, I have created a changelog that logs changes to the project everyday with the time commitment for the day as well. Also, the local history extension in VSCode automatically backs up changed code to a **local folder** as well as storing the entire project in a folder in **Box**.

**Module List**

This project currently does not use any external modules.