Haisam Elkewidy

Computer Science E10B Term Project Proposal

Overall Idea of Term Project:

Aircraft departing to and from airports, hovering over wide miles across the country, and even flying over international waters, must always be in contact with some kind of air traffic controller to ensure their flight path is reasonably safe and without obstructions. This also alerts pilots to any incoming wind turbulence, any other aircraft in a particular airspace, among other complications. Recent innovations in aviation are not leaning towards bigger aircraft, but likely smaller ones. More specifically, jetpacks.

This project is a prototype for an air traffic controller dashboard designed specifically to operate locally and manage jetpacks flying around the city, or the locale even. It functions quite similarly to those monitoring large scale aircraft, or akin to those monitoring helicopters flying over a city airspace. The intention is to give individual free flyers all the data they need to know about their locale, including traffic, weather conditions, any external hazards, radar guidance, and so on. When an individual flyer provides a flight path to air traffic controller, guidance is expected to be returned back to them as they are en route to their destination, which may involve flying over buildings, or flying in between streets and other roads, be it roads, parks, or other areas of the locale. The locale itself would be displayed in some kind of grid system. The input for the flight path would be hypothetically done via Java Swing at a GUI. Once the flight path is entered, air traffic control handles the rest as the jetpack flies to its destination.

Why Bother Programming This Project?

In a hypothetical future where citizens own jetpacks the same way they own bikes, as in they are parked somewhere near their residence, likely registered with a serial number, and so on and so forth, the need will arise for some kind of air traffic control local to those citizens' cities, provinces, residential areas, and so on and so forth. Drones have recently been flying in local airspaces, and long before drones there have been helicopters, it is therefore only a matter of time before aerial vehicles fly directly over the same roads that cars drive on, and in some cases over entire neighborhoods and districts of cities. An opportunity like this calls for a new form of air traffic control to monitor it carefully, making sure local pilots are always safe in the skies.

Algorithms/Data Structures to be Used:

- ArrayLists: for storing jetpacks

- Arrays for select parking spaces. The idea being that parking spaces are rather limited, when a space is occupied it is inserted in that array. When otherwise empty it is null.
- Multidimensional Arrays/Maps - for the grid of the locale/city including denoting buildings, roads, parks, streets, highways, etc.
- Potentially implement Graphs using Vertex - closest thing to a Graph class or Graph-like implementation, allowing for determining the best routes from point A to point B
- Potentially Vectors for vector lines and coordinates.

Classes to be Invented:

- **DayTime class:** This class reveals the date and time on the air traffic controller, thus setting the UI to either day, night, sunset, dusk.
    - Known properties:
        - Map for different times of the day and their corresponding hours
    - Known methods:
        - setTime() - this sets the UI display to different time of the day based on the current datetime, which is then compared to the values in the different times of the day property
- **Jetpack class**: This class is for an individual jetpack in the current airspace. For simplicity's sake, the speed will be kept constant. Primarily the jetpack's starting position, it's current coordinates, and then it's final destination will be stored as properties. Current coordinates within the grid are meant to indicate where it is within the space, in the likely event of the need to detour or take another route due to unexpected circumstances. If it were to be expanded to multiple jetpacks, nomenclature that is similar to that of aircraft could also work (call signs, altitudes, flight speed, etc.)
    - **Known methods:** moveLeft(), moveRight(), ascend(), descend(), park()
        - moveLeft() - moves the jetpack to the left of the map
        - moveRight() - moves the jetpack to the right of the map
        - ascend() - causes the jetpack to ascend to higher altitude
        - descend() - causes the jetpack to lower to a certain altitude
        - park() - causes the jetpack to enter a parking space, or park at its destination
- **FlightPath class**: This class determines the flight path for a given jet pack. It is likely that each jetpack class will have it's own FlightPath object nested inside it, within the context of air traffic control. It is intended to be the starting point, the end point, and the hypothetical best route to get there assuming stable conditions. A boolean value exists for different hazards, such as inclement weather, a building collapses, an accident in the skies or in the area, police reported ahead, and so on and so forth.
    - **Known methods:** setFlightPath(), getFlightPath(), detour(), halt()
        - setFlightPath() - sets the flight path for a jetpack
        - getFlightPath() - gets the flight path for a jetpack
        - detour() - alters the flightpath by adding a detour in the event of a weather hazard
        - halt() - puts the flight path to a complete stop and causes the jetpack to do an emergency landing

- **AirTrafficController class:** This class represents the overall AirTrafficController tower for the locale. For now, this would be for just one given airspace but can always be expanded to accommodate any location in the United States, or even worldwide. It is also likely that this will be implemented via Java Swing GUI.
- **Grid class:** This class represents the grid for the airspace the jetpacks are flying in. It includes numerous types of buildings, streets, parks, residential areas, commercial areas, and the sort. Within this grid, the jetpacks are flying around trying to get to their destination on time.
- **Radio class:** The radio class is designed to offer instructions for takeoff and landing from one point to another within the locale. The radio class does so by transmitting messages to the jetpack pilot, which the pilot should receive.
    - giveNewCoordinates() - communicates to a jetpack new coordinates to take to ensure safe flight
    - giveNewAltitudes() - communicates to a jetpack new changes to their altitude depending on situation
- **Radar class:** The radar class is meant for the AirTrafficController to have positional awareness of every jetpack within the locale's airspace. Not quite sure how this would be implemented yet.
    - **Known methods:**
        - getJetPackPositions() - gets the positions of every jetpack on the radar map and monitors them
        - updateJetPackPositions() - frequently update jetpack's positions on the radar, intended to help them detour to a certain direction and/or altitude
        - identifyAircraft() - used to fetch jetpack's identifiers and display them on the map
- **AccidentAlert class:** This class alerts jetpack pilots of any accidents on the road, be it at intersections or highways. Pr even of collisions between other jetpacks.
    - alertJetpacksOfAccident() - alerts nearby jetpacks of an accident in a certain section of the city
    - removesAlert() - removes the alert for the accident if it has been addressed already
- **Weather class:** This class represents different kinds of weather along with varying degrees of severity. The different options for weather and severity can be described by the following tables:
    - Known properties:
        - A Map or Dictionary for Different types of weathers and the category of their severity. The possible values for this are as follows:
            - Clear/Sunny, 1
            - Partly Cloudy/Overcast, 1
            - Light Rain/Drizzle, 1
            - Light Snow/Flurries, 1
            - Fog/Mist, 2
            - Steady Rain, 2
            - Showers/ Thunder Showers, 2

        - Windy Conditions, 2
        - The values 1 and 2 correspnond to “everyday conditions, minimal risk” and “notcieable impact, manageable disruptions” on the global weather severity scale. This can of course.be expanded to 5 categories and beyond but for now the implementation is simple.
    - Known methods:
        - sendWeatherBroadcast(): sets the weather and global severity on the GUI for the Air Traffic Controller
        - changeWeather(): changes the weather announcement to a different type depending on the situation

Forms Of Input:

The expected inputs into the program include, but are not limited to, as follows:

- Registering jetpack information (callsign, brand, model, makeYear)
- Each jetpack’s flight start point and final destination (route will be advised to them by the program)

Forms Of Output:

- Air traffic controller radar dashboard.
- The grid for the city or locale in which the jetpacks are flying.
- Parking spaces denoted in each section of the grid, including how many spaces are available.
- The current time on the far right corner, and what point of the day it is at (Sunrise, Dawn, Sunset, Dusk, Nighttime, etc.)
- Weather alerts, identifying a type of weather and certain degree of severity.
- Guidance on which jetpack should detour from its current flight path, and which should simply stop flying and descend immediately to the ground.
- Each jetpack displayed on the grid, as well as its metadata and its coordinates.

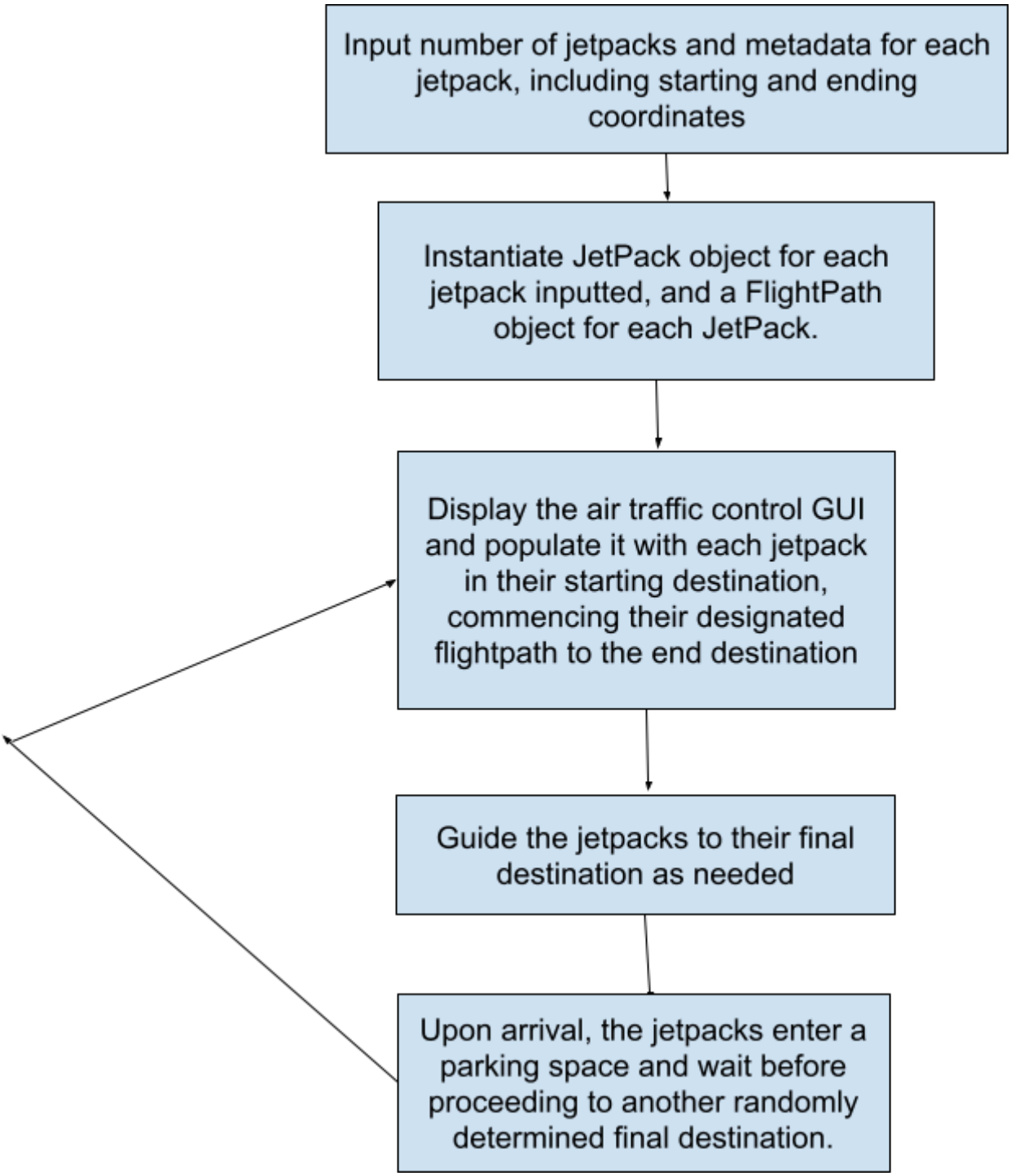
Expected State/Functionality of the Program:

Expected modus operandi is the following steps:

1. Input the number of jetpacks to display on the radar, and information for each one. An object will then be instantiated for each jetpack based on the metadata provided. An example dashboard or GUI is shown below with an example of inputting 3 jetpacks into the air.

2. Each jetpack will then have a flightpath created for it by first inputting its starting point and final destination on the map. The map will be displayed beforehand and coordinates are numerical values on the grid.
3. Once all the information has been inputted, the grid is displayed in the GUI, along with all the jetpack’s starting positions. Information about the day and time is displayed on the far right corner, and the radar guides each jetpack to their destination in real-time.

4. Whilst in operation, weather and accident alerts are generated randomly and may not necessarily generate during each instance the application is run.
5. Once the jetpacks arrive at their final destination, they will park at the corresponding parking space and then randomly be assigned a new destination to travel to. They will then have a new flight path and travel to that route. Rinse and repeat.



Air Traffic Control Screen example. Source: https://prd-sc102-cdn.rtx.com/-/media/ca/product-assets/marketing/a/air-traffic-getty-1351088941.jpg?rev=cbbc080bbe804ad4baeb6b87dbe011a1