Project requirements

Functional Requirements:

* 1. *The system should be able to take and store coordinate values*. This should be done by creating a class which has a constructor that can accept two double-precision numbers, where the numbers correspond to longitude and latitude measurements.

This requirement covers **functionality** and **correctness** of the system.

* 1. *The system should be able to check whether or not a given coordinate is within a given central area that is defined by other coordinates*.

This requirement covers **functionality** and **correctness** of the system.

* 1. *The system should be able to take two different points(coordinates) and return the Pythagorean distance between the two points*.

This requirement covers **functionality** and **correctness** of the system.

* 1. *The system should have a method which takes two points(coordinates) and checks if the points are less than 0.00015 degrees apart*.

This requirement covers **functionality** and **correctness** of the system.

* 1. *The system should be able to calculate the next position of the drone when a compass direction is given*.

This requirement covers **functionality** and **correctness** of the system.

* 1. *The system should be able to correctly retrieve, store and use coordinates from a specific remote server to check if the drone is within the central area*. These coordinates are what the drone uses to understand where it can and cannot fly, and also where Appleton Tower and all of the restaurants are located.

This requirement covers **functionality**, **correctness** and **privacy** of the system because if wrong coordinates are retrieved, the drone may fly to places where it is not allowed to fly, as well as **security** because if the system were to request data from the wrong address, the drone could potentially be used for malicious purposes.

* 1. *The system should return the drone to Appleton Tower before it runs out of battery*.

This requirement covers **functionality**, **reliability** and **safety** because if the drone were to run out of battery before its return, it dropping from the sky could injure people or cause damage to cars, buildings, etc.

* 1. *The system should not allow the drone to fly in no-fly zones*.

This requirement covers **functionality**, **reliability**, as well as **privacy** and **safety** because of civilians who are present in the no-fly zones.

* 1. *The system should always maximise the amount of orders to be completed for a day*. This requirement covers **functionality**, **efficiency** and **fairness** because it tries to fulfill as many orders as possible.
  2. *The drone should keep the pizzas insulated during flight*.

This requirement covers **functionality** and **pleasurability** since the quality of the pizza upon arrival is what determines how satisfied with the app the person is.

* 1. *The system should output a user-friendly message in the console if the console input is not valid*.

This requirement covers **functionality** and **liveness** since the response is nearly instant after providing an invalid input.

* 1. *The system should be able to generate a flight path and all other relevant files in no more than 60 seconds*.

This requirement covers **functionality** and **performance** since the program has a certain execution time limit.

* 1. *The project should have three classes named Restaurant, Menu and Order, where Restaurant and Menu represent the results from the data request made to the remote server*.

This requirement covers **functionality** and **maintainability**, making it easier to manage and modify existing code or add new code or remove old code.

Measurable attributes

* 1. **Efficiency**: measuring the time it takes for the pathfinding algorithm to finish calculations and generating necessary files.
  2. **Reliability**: can be measured by comparing the uptime and downtime of the app.
  3. **Privacy**: can be measured by looking at how well data being sent, received and stored is encrypted, where it is stored, for how long it is stored as well as how well-informed the people are about how their data is handled.
  4. **Accessibility**: The ease of use for the end users, which can be measured through how accessible the application and its contents are (e.g., on what devices and at what locations it works)
  5. **Performance**: the system's performance can be gauged by the daily average of pizza deliveries compared to the amount of orders placed
  6. **Functionality**: the system's functionality can be evaluated by assessing its compliance with all necessary requirements.
  7. **Maintainability**: the ease of adding new features and removing outdated ones determines how maintainable the system is.
  8. **Fairness**: the fairness of the system can be measured by how many students get their orders delivered compared to how many orders were placed.
  9. **Restorability**: the time it takes for the system to resume normal operation after a failure.
  10. **Interoperability**: measured by how well the app works on different systems and different versions of Java.