Informatics Large Practical — Project Proposal

Introduction

A pizza drone delivery service is a large system, consisting of many moving parts. This proposal outlines the requirements, challenges, and plan for implementing such a system. The full system will require two parts [1, § 0.1]: a mobile app or web interface for users to place orders, and a backend system to handle order validation, delivery route planning, and drone delivery. The frontend system, which will communicate with the backend via a REST API [1, §§ 0.6-0.7] to place orders, is not handled by this project. This project will focus on retrieving data from the REST API, validating orders, and planning delivery routes.

Requirements

There are several requirements to be considered for this system to be successful, which are outlined below. I have divided these requirements into a few categories: the ordering system, the drone delivery system and scalability. The ordering system will handle fetching orders from the REST API, validating them [1, § 0.1], and passing them to the drone delivery system. The drone delivery system will handle planning delivery routes and delivering pizzas to customers.

The Ordering System

As stated above, this project does not handle the frontend section of the system. However, the system must be able to retrieve orders from the REST API, and then validate these orders before making the delivery. As the REST API returns data in the form of JSON [1, § 0.6], the system must be able to parse this data and extract the relevant information. Some checks involved in validating an order include checking that the order only contains pizzas from one restaurant, that the restaurant in question is open, and that the payment details of the order are correct [1, § 0.7.3].

The Drone Delivery System

The drone delivery system is the main focus of this project. The system must be able to plan delivery routes for the drone, ensuring that the drone avoids populated areas and flies over the roofs of buildings wherever possible [1, § 0.1]. These routes should be as direct as possible, while still avoiding any populated areas, and flying over the roofs of buildings where possible, so a balance should be found between staying over roofs and going as the crow flies. Calculating the drone's route should take no more than 60 seconds, so as not to delay the delivery process unnecessarily [1, § 0.7.7]. The drone must be able to deliver pizzas to the designated drop off point, and must only carry one delivery at a time.

Scalability and Maintainability

As the system will be maintained by future students [1, § 0.1], it is important that the codebase is well-documented and easy to understand. The system should also be designed to be scalable, allowing for additional drones, restaurants, and drop-off points to be added to the system without requiring major changes to the codebase [1, § 0.1]. For this reason, the system should be data-driven, such that changes in participating restaurants, menus, and delivery locations within the REST API do not require any changes in the codebase. Another factor which much be considered for scalability is the performance of the system, as it must be able to efficiently calculate the fastest route for the drone to take, and must be able to handle a large volume of orders.

Challenges

Implementation Challenges

The main challenge of implementing this system is the route planning. As stated in the requirements above, the system must be able to calculate the fastest route for the drone to take, while avoiding populated areas and flying over the roofs of buildings where possible. The challenge is to calculate this route in a reasonable amount of time, as the system must be able to calculate the route in under 60 seconds [1, § 0.7.7]. The next section outlines one viable solution for calculating the shortest path.

Another challenge which may arise when implementing this system is the reliability of the REST API. As the system will be reliant on the REST API for fetching orders, it is important that the REST API has a reliable uptime. In the event that the REST API does go down for a period of time, the system will not be able to fetch any new orders.

External Challenges

Aside from the challenges which could arise while implementing the system, there are also some external challenges which could affect the system. One such challenge is the weather, as the system will be reliant on the drone being able to fly to the drop-off point. If the weather is too poor for the drone to fly, the system will not be able to deliver any pizzas. Other factors which may impact the flight of the drone include birds in the area, which could cause the drone to crash.

Project Plan

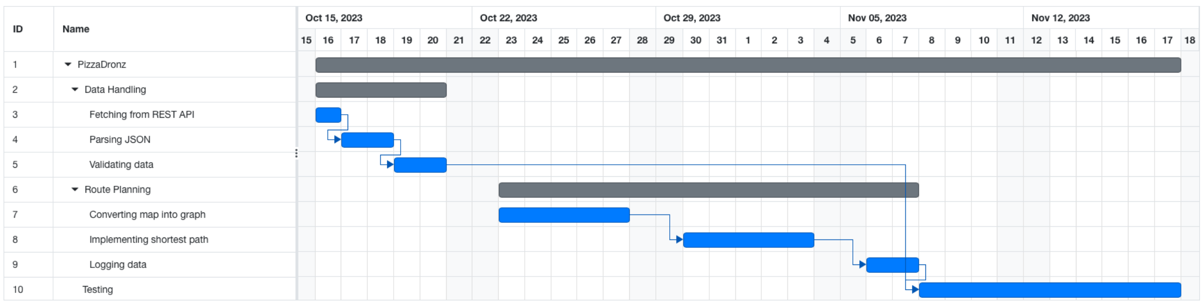
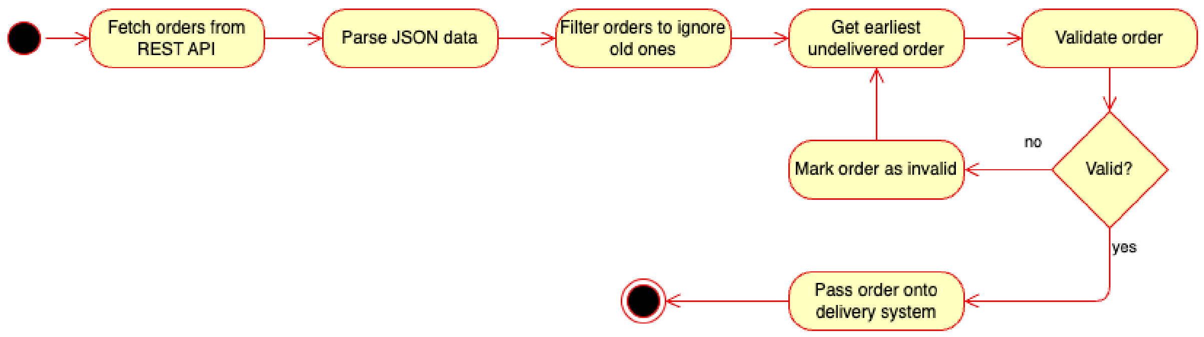


Figure 1: A Gantt diagram outlining roughly how much will be spent on each part of the project.

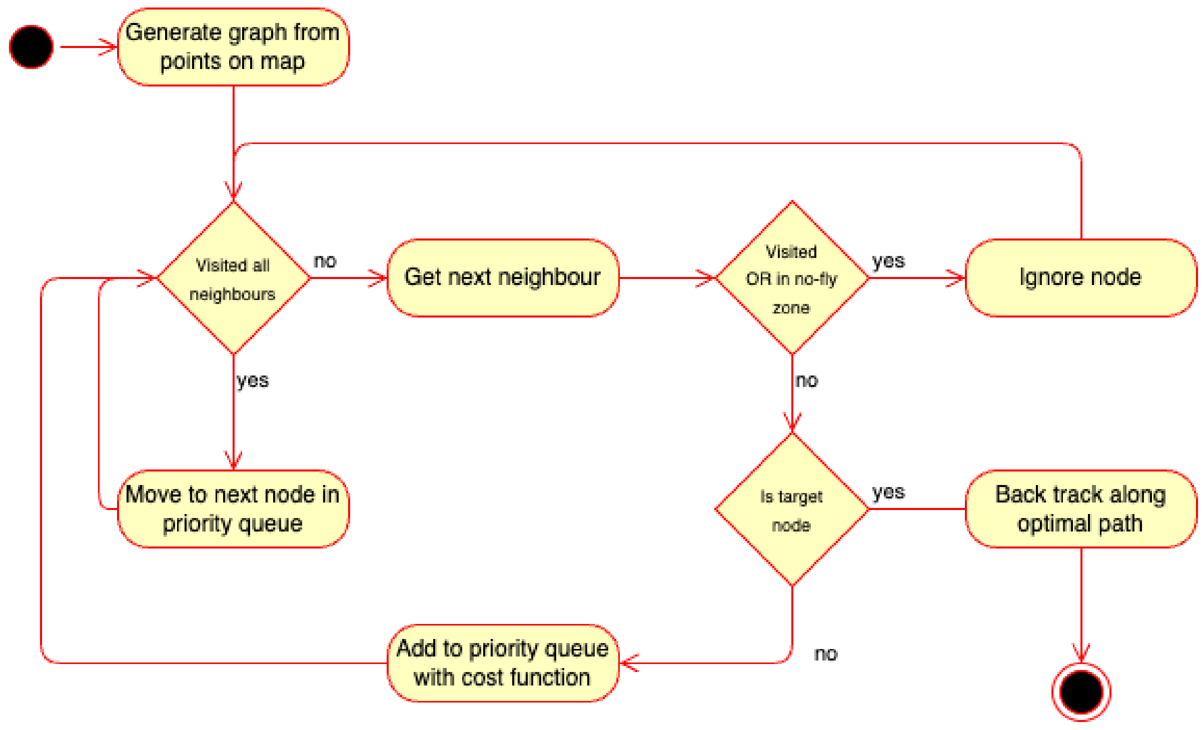
The bulk of the time spent on this project, as shown in figure 1, will be focused on the route planning system. However, before implementing this, I will implement the fetching of orders from the REST API, validating them, and passing them onto the route planning system. As this is a critical feature of the system, I have created an activity diagram to outline the process of retrieving the next order valid order, as depicted in figure 2.



Fetch orders from REST API
Parse JSON data
Filter orders to ignore old ones
Get earliest undelivered order
Validate order
Valid?
no
Mark order as invalid
yes
Pass order onto delivery system

Figure 2: An activity diagram depicting the process of fetching the next order from the REST API, validating it, and passing it onto the delivery system.

The route planning system can be implemented using the weighted A* algorithm [2], which is a version of Dijkstra's algorithm [3] that is optimised for finding a single location. By considering positions on the map as nodes on a graph, we can use the A* algorithm to find the shortest path between two points on the map. Nodes which appear within no-fly zones can be removed from the map, so the drone will not be able to visit them. A very high level outline of the A* algorithm can be seen in figure 3.



Generate graph from points on map
Visited all neighbours
no
Get next neighbour
Visited OR in no-fly zone
yes
Ignore node
yes
no
Move to next node in priority queue
Is target node
yes
Back track along optimal path
Add to priority queue with cost function
no

Figure 3: An activity diagram depicting a high level overview of the A* algorithm.

References

[1] Michael Glienecke - School of Informatics - University of Edinburgh. *ILP Course Specification*. 2023.

[2] Wikipedia. *A* Search Algorithm*. URL: `https://en.wikipedia.org/wiki/A*_search_algorithm`.

[3] Wikipedia. *Dijkstra's Algorithm*. URL: `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`.