Informatics Large Practical — A Reflection on and Further Development of the PizzaDronz System

A Reflection of the Changes in the Requirements and the Implementation

Changes to the Specification

The requirements laid out prior to the implementation of this system have remained largely unchanged. The only change that has been made is the requirement for the system to ensure that the drone flies above the roofs of buildings where possible, while also avoiding any no-fly zones. Instead of that, the system now only ensures that the drone does not enter the no-fly zones, but does not require that the drone flies above the roofs of buildings. This change is to speed up the calculation of the flight path, as the system would otherwise need to access map data which would add significant overhead to the calculation of the flight path.

How the Requirements Have Been Implemented

The remaining requirements have been implemented as outlined in the project proposal. The system has a `RESTManager` class which is responsible for communicating with the REST API. This class takes advantage of the `jackson` library (*1*) to parse the JSON data returned by each endpoint of the REST API. Once the data has been parsed, the `OrderValidator` class (which implements the `OrderValidation` interface from the `IlpDataObjects` package (*2*)) is used to validate the orders. Validating an order consists of checking that the order has between one and four valid pizzas from the same restaurant, that restaurant is open, the total cost of the order is correct, and the credit card information is valid.

Once the system has validated all the orders for a given date, the system uses a `FlightPathGenerator` class which generates a flight path for the drone to follow. To generate the flight path, the system uses the a-star search algorithm (*3*) to find the shortest path between each restaurant and the drop-off point, then stitches these paths together to form a single flight path. To keep the total time of the calculation of the flight path to a minimum, the system does not calculate a new flight path for every order. Instead, the system caches the flight path for each restaurant, so they can be reused for multiple orders. Similarly, the system does not calculate a new flight path to get from the drop-off point to the restaurant, but simply reverses the path it has already calculated to get from the restaurant to the drop-off point. In order to avoid no-fly zones, the system simply disregards any nodes which are within a no-fly zone when running the a-star algorithm. The same technique ensures that the drone does not leave the central area once it has entered it, by considering anything outside the central area to be a no-fly zone, as soon as the drone has entered it.

The final requirements for the system were that the code should be maintainable, well documented, and data driven. To ensure that the code is maintainable, the code follows the standard Java coding conventions (*4*), and is well documented using standard Javadoc (*5*). To ensure that the system is data driven, there are very few hard-coded values in the system, instead retrieving as much data from the REST API as possible.

Difficulties During the Implementation

As expected, the most difficult part of the implementation was the calculation of the flight path. The initial plans for calculating the flight path involved using Dijkstra's algorithm (*6*) to find the shortest path between each restaurant and the drop-off point, then stitch these paths together to form a single flight path. The problem with Dijkstra's algorithm is that it will consider many nodes which will never be part of the shortest path, as seen in Figure 1.

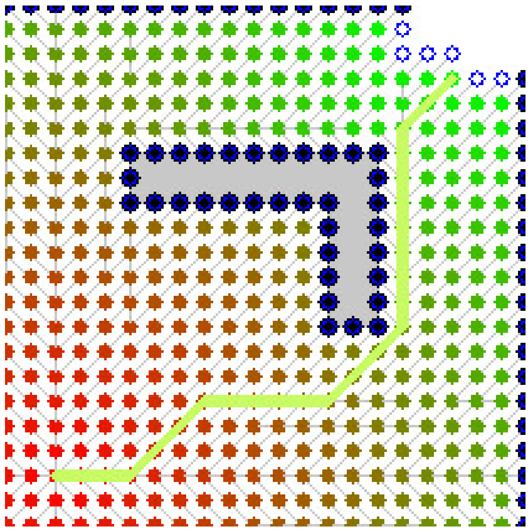

Figure 1: An illustration of Dijkstra's algorithm. Taken from Wikipedia (*6*).

To avoid this, the system instead uses the A* search algorithm (*3*) to find the shortest path between each restaurant and the drop-off point. The A* algorithm is a modified version of Dijkstra's algorithm which uses a heuristic to avoid considering nodes which are unlikely to be part of the shortest path. The heuristic used by the system is the Euclidean distance between the current node and the goal node. Figure 2 shows an example of the A* algorithm in action, while Figure 3 shows an example of the Weighted A* algorithm, which sacrifices optimality for speed. For the purposes of this system, the regular A* algorithm is fast enough, that the Weighted A* algorithm is not necessary.

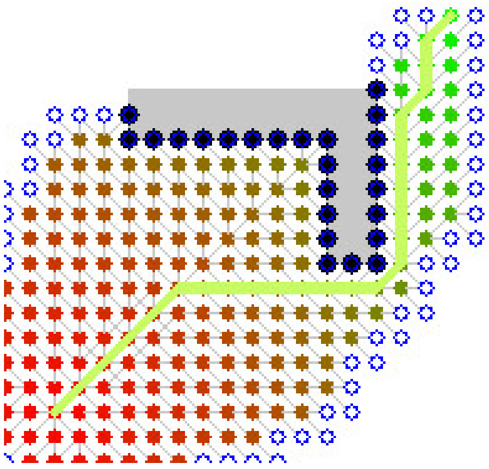

Figure 2: An illustration of the A* algorithm. Taken from Wikipedia (*3*).

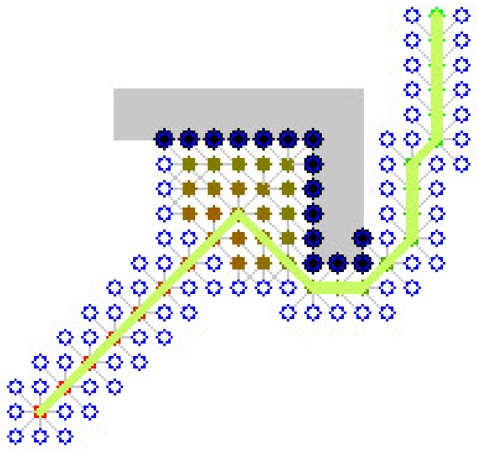

Figure 3: An illustration of the Weighted A* algorithm. Taken from Wikipedia (*3*).

Implementing a Mobile Interface for the Client-Facing Part of the System

Introduction

Having implemented the backend of the system, the next step is to implement the frontend which allows users to place orders. This frontend will be implemented as a mobile application, as this is often the most accessible way for users to interact with the system. Creating a mobile app for ordering pizzas is not a new idea, so there are many existing apps which can be used as inspiration for the design of the PizzaDronz application. This essay focuses on the user interface and user experience of the PizzaDronz app, and will draw inspiration from some of the most popular pizza ordering services such as Domino's (*7*), Pizza Hut (*8*), and Papa John's (*9*).

Mobile User Interfaces and User Experiences

The two most common mobile operating systems are iOS and Android, and each has their own substantial library of apps (*10*) available via the App Store (*11*) and the Google Play Store (*12*) respectively. While these apps often look and feel quite different, they all follow the same basic UI and UX principles, which are outlined in the Android and iOS design guidelines (*13*, *14*). It is important for mobile apps to have a consistent user interface which follows proper guidelines, as this directly contributes to a good user experience, by allowing users to quickly learn how to use the app. These guidelines also help to ensure that the app is accessible for as many users as possible, including those with disabilities such as colour-blindness, or those who struggle to use touch screen devices.

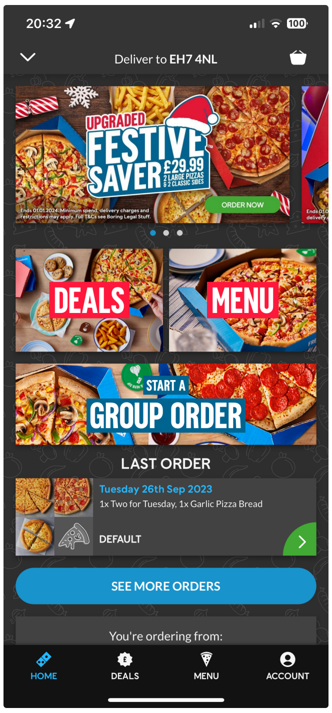

Figure 4: The home screen of the Domino's Pizza app on iOS.

One of the most consistent features across many mobile apps is the use of a tab bar at the bottom of the screen, as seen in Figure 4. The tab bar allows users to quickly navigate between the main sections of the app, and often contains just a few items, including the home screen and the user's account. Keeping the tab bar at the bottom of the screen allows users to easily reach the buttons with their thumbs, which is important for one-handed use of the app, as shown in Figure 5.

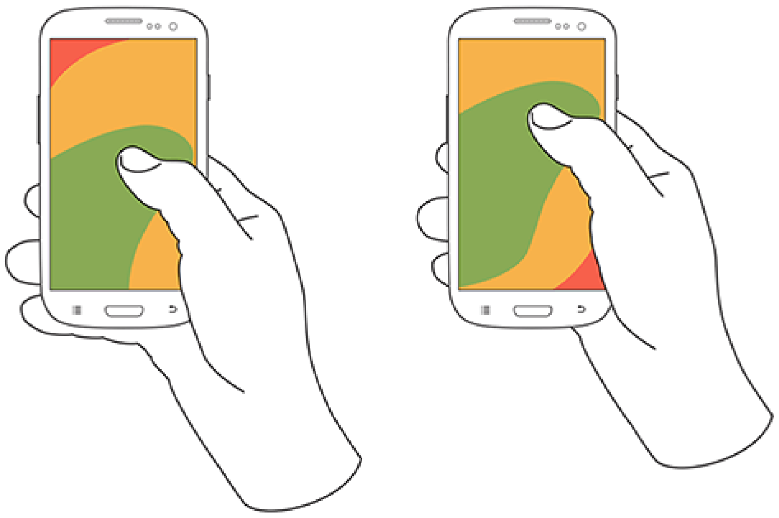

Figure 5: Representation of the comfort of a person's one-handed reach on a smartphone. Green: area a user can reach easily. Yellow: area that requires a stretch. Red: area that requires users to shift the way they're holding a device. Source: uxmatters.

The PizzaDronz App

As with any user interface design, it is important that as soon as the user opens the app, they are immediately presented with access to the most crucial information. In the PizzaDronz app, the first thing the user should see is a list of participating restaurants, including the name of the restaurant, its distance from Appleton Tower, and the estimated delivery time. The list should be sorted by the estimated delivery time, so that the restaurant with the shortest estimated delivery time is at the top of the list. As the participating restaurants are not open every day, any restaurants which are closed should be greyed out, and placed at the bottom of the list. Upon clicking on a restaurant the user should be taken to a new page, which should show its menu, a list of pizzas, with the name of the pizza, its price, and a picture. Each pizza should have a button which allows the user to add it to their order.

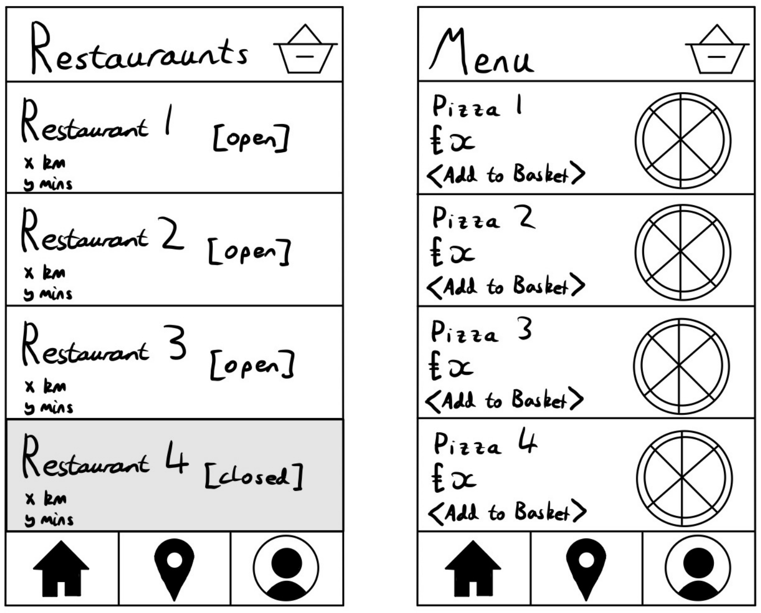

Figure 6: A mock-up of the PizzaDronz app.

A basic mock-up of the PizzaDronz app can be seen in Figure 6, depicting both the home page and the menu page. In this mock-up, the tab bar at the bottom of the screen has just three items: home, map and account, and it should be noted that these menus use symbols rather than text, as this is more accessible for a wider range of users. The map page should show a map of the city, with the locations of each restaurant as well as the location of Appleton Tower marked on the map. The account page should allow the user to view their order history, including any current orders which they may wish to track or cancel.

The app should ensure that the user cannot place an invalid order, meaning that each order must contain between one and four pizzas from the same restaurant. Once a user has selected the pizzas they want to order, they should be able to go to their basket, where they can see the total cost of their order, and proceed to the checkout. As with any modern checkout page, the user should be presented with a choice of payment methods, including credit/debit card, Apple Pay, and Google Pay.

Once an order has been placed and sent to the database, the user should be presented with a confirmation page, showing the estimated delivery time. This page should also include a map marking the location of the restaurant, the location of Appleton Tower, and the flight path the drone will take. Once the drone has arrived at Appleton Tower, the user should be notified that their order has arrived, so they can collect it from the drone before it flies away.

Conclusion

In conclusion, to ensure a good user experience, the PizzaDronz app should follow the standard user interface and user experience guidelines for mobile applications, and should have as simple and intuitive a design as possible. The app should allow users to quickly and easily place orders, while preventing them from placing invalid orders. The app should also allow users to track or cancel their orders, and should notify them when their order has arrived.

References

1. FasterXML, *Jackson* (`https://github.com/FasterXML/jackson`).
2. M. Glinecke, *IlpDataObjects* (`https://github.com/mglienecke/IlpDataObjects`).
3. Wikipedia, *A* Search Algorithm* (`https://en.wikipedia.org/wiki/A*_search_algorithm`).
4. Oracle, *Code Conventions for the Java™ Programming Language* (`https://www.oracle.com/java/technologies/javase/codeconventions-contents.html`).
5. Oracle, *Javadoc* (`https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html`).
6. Wikipedia, *Dijkstra's Algorithm* (`https://en.wikipedia.org/wiki/Dijkstras_algorithm`).
7. *Dominos* (`https://www.dominos.co.uk`).
8. *Pizza Hut* (`https://www.pizzahut.co.uk`).
9. *Papa John's* (`https://www.papajohns.co.uk`).
10. L. Ceci, *Number of apps available in leading app stores as of 3rd quarter 2022* (`https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores`).
11. Apple, *Apple App Store* (`https://www.apple.com/uk/app-store/`).
12. Google, *Google Play Store* (`https://play.google.com/store/games?hl=en&gl=GB`).
13. Google, *Android Developers Design and Plan* (`https://developer.android.com/design`).
14. Apple, *iOS Human Interface Guidelines* (`https://developer.apple.com/design/human-interface-guidelines`).