Shoptimus Prime Final Report

Michael Kipper & Bryce Leung

University of Toronto

ECE 1778

Introduction

The goal of this project, Shoptimus Prime, is to create a mobile grocery shopping application targeting the Google Android platform. This application allows users to create their grocery list and obtain the location of the lowest price for that basket of goods. Data entry is primarily via barcode scan and manual price entry. This information will then be looked up either on our own database or third-party UPC databases available on the Internet to figure out what that item is, and then entered into our database along with a timestamp and location information. Over time, this will create an aggregated crowd-sourced snapshot of where grocery items can be purchased and for how much. Using the crowd-sourced grocery database, we can then find the lowest possible price of the total basket of goods and then advise the user where to go to buy everything.

While applications are already available that allow users to scan bar-codes of grocery items and then search online retailers' website for similar items, we believe this is the first implementation of a mobile grocery application that sources pricing information from people rather than a website. This means that this application will be useful to stores such as "Mom-and-Pop" grocers that aren't large enough to have an online shopping system.

Overall Design

Shoptimus Prime is made up of two separate yet equally important groups. The Activity classes which control the user experience, and the Manager classes which implement core functionality. These are their stories.

Activity Classes

The Activity classes each correspond to a single screen, or activity, and are shown below. For clarity, only classes which apply to the end-user are shown. Activities created to aid debugging or development are omitted.

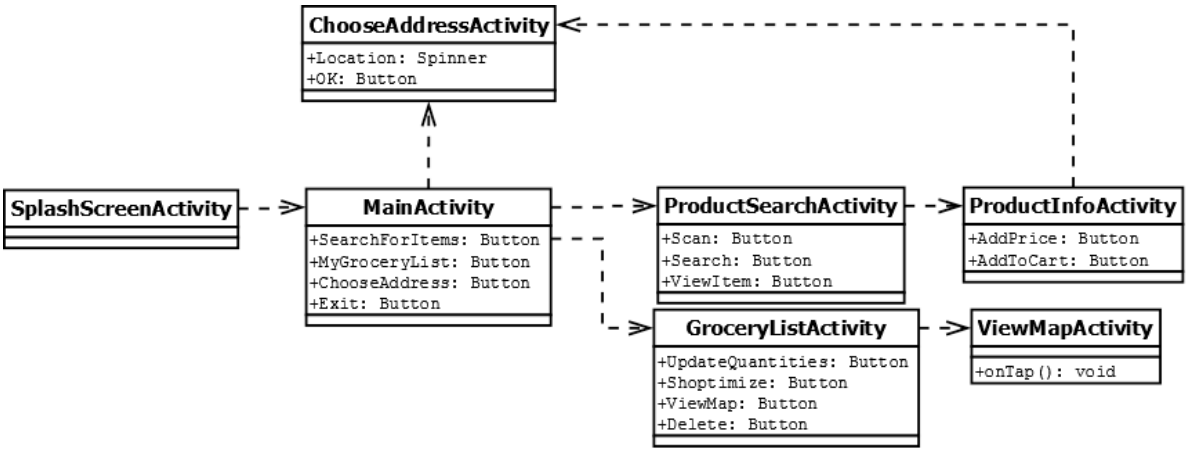

Figure 1. Activity Class Diagram.

SplashScreenActivity

This activity implements a simple splash screen displayed on start-up. It remains visible for a few seconds, and then continues on to the Main screen.

MainActivity

The MainActivity class is the home screen. The user is presented with a four buttons: Search For Items, My Grocery List, Set Location, and Exit. The first three buttons bring the user to ChooseAddressActivity, ProductSearchActivity, and GroceryListActivity, respectively. The Exit button quits the application.

ChooseAddressActivity

ChooseAddressActivity presents a spinner control to the user. Because a given longitude-latitude pair may correspond to multiple geo-coded locations (for example, a single location may be described as a street address, or a name of store), the user must select which description he or she feels describes their current location most accurately. This activity only really applies when the user is entering new pricing data for a particular item, and can be either done manually by pressing the Set Location button on MainActivity, or automatically when they press the Add Price button in ProductInfoActivity.

Under normal circumstances, this only needs to be done once by the user when they update their first price. Their selection is then saved and used for subsequent updates, unless their location changes dramatically, in which case this activity will pop up again automatically when the user adds a new price.

ProductSearchActivity

This activity allows the user to search for an item, either by scanning its barcode or typing it into a text box. The Scan button calls out to a 3rd party Barcode Scanner API, which brings up a camera view and allows the user to scan barcodes. When a barcode gets scanned,

the application looks it up in the Shoptimus Prime database, and then, if that fails, an online UPC database to figure out what that item is. If the item is found on the UPC database, it is then automatically added to the application's own database. The Search button, as the name implies, searches for an item requested by the user.

Regardless of whether the item was found via barcode scan or searching, matching items are then displayed in a table to the user, with a button besides each labeled View, which then takes the user to ProductInfoActivity for that particular item.

ProductInfoActivity

ProductInfoActivity displays information about a single item, namely, its name, description, size, and previous prices for that item. The user can choose between two buttons: Add Price and Add To Cart. Add Price allows the user to enter in a new price for that item and upload it, and Add To Cart adds that item to their shopping cart.

GroceryListActivity

This activity displays the user's current shopping list. Each item the user has selected in ProductInfoActivity is displayed here, with its name, size, and the quantity of those items the user wants. The quantity is editable by the user, and can be updated by pressing the Update Quantity button. Individual items may also be deleted using the "Del" button beside each entry.

The Shoptimize button begins the price optimization algorithm, and upon its completion this activity is updated with the lowest price found for each item. A third button View Map also becomes enabled at this point, which brings the user to ViewMapActivity.

ViewMapActivity

The ViewMapActivity class shows a map using the Google Maps API telling the user where all the items in their shopping list are as well as a route to take. The map displays each item as a blue shopping cart, and also displays the user's current location as a red bulls-eye. All of these icons can be tapped by the user to show a description of which item corresponds to it.

Manager Classes

The core functionality of this application is implemented in Manager classes. These classes are implemented as a singleton design pattern, and are accessible from any activity via the "Managers" class, which simply stores a single pointer for each Manager class instantiation. Again, for clarity, only functionality meant for end-users is shown below.

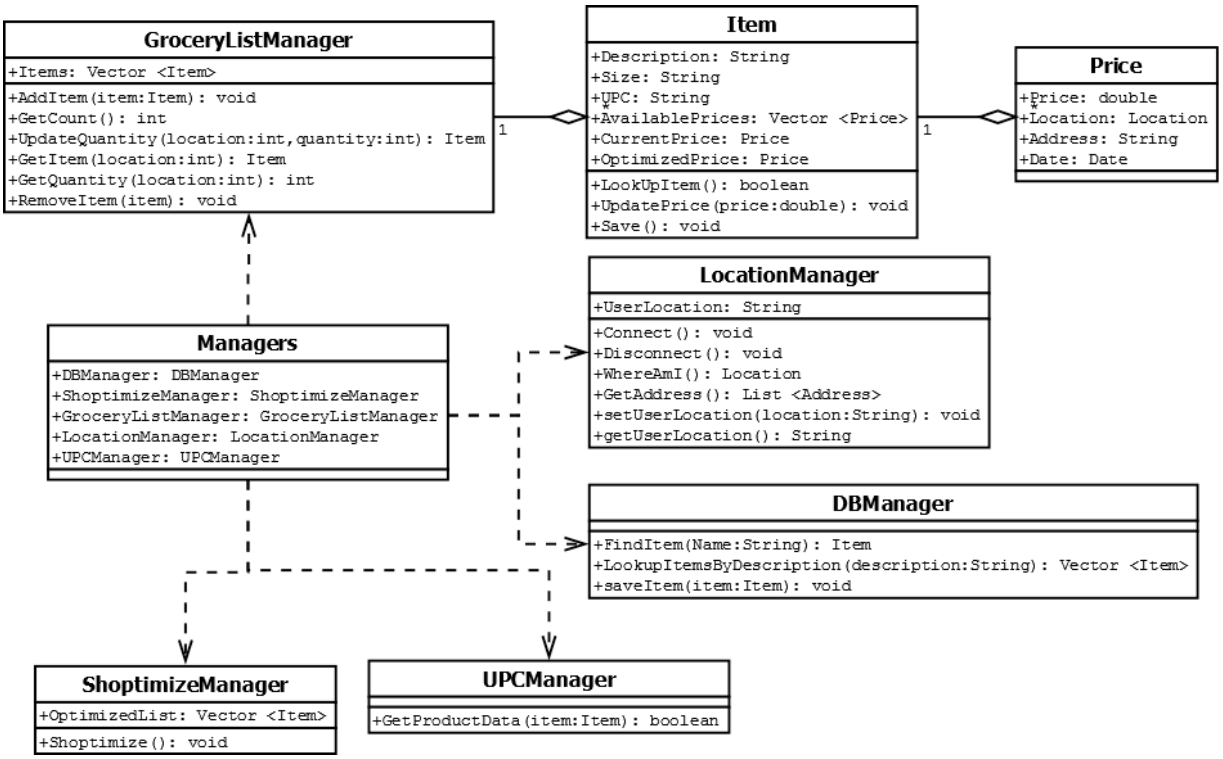

Figure 2. Manager Class Diagram.

GroceryListManager

The GroceryListManager class maintains the user's current grocery list as a vector of Item objects and provides a set of methods allowing activities to access this list. This class is used mainly by ViewItemActivity to add new items to the list, and GroceryListActivity to display/manipulate it.

Item

The Item class encapsulates all the information about a single item, including description, size, and UPC. Pricing information is stored in three fields: AvailablePrices which is a list of all prices that have been previously entered, CurrentPrice which is what the user inputs, and OptimizedPrice, which is the lowest possible price that's found by the optimization algorithm.

This class also has the methods LookUpItem, which queries the database for information about itself, UpdatePrice, which adds new pricing data, and Save, which saves the item to the database.

Price

The price class is a simple struct-style class that represents a single tuple of price, location, geo-coded address, and date.

LocationManager

The LocationManager class is responsible for all location-related activities. This includes managing communication with the GPS API, retrieving the current location of the user, and translating locations into geo-coded addresses.

DBManager

The DBManager class is responsible for all communication to/from the database. The database is implemented using Google's SQLite API, but this class is built so that it can be dynamically switched to communicate with any database that understands SQL commands.

UPCManager

The UPCManager class is responsible for all communication to the Internet in order to look up UPC of items that aren't currently stored in our application's database. It currently queries the free website www.upcdatabase.com.

ShoptimizeManager

The ShoptimizeManager class implements the optimization algorithm. It interfaces with the GroceryListManager and stores its results by updating each Item object's OptimizedPrice field. It also stores its own copy of the grocery list that's order determines the order in which the user should visit each location. ViewMapActivity uses this when it displays the suggested route to the user.

Statement of Functionality

The original core goals of this project, mainly to implement an app that allows people to scan barcodes, enter pricing data, and then use that pricing data to optimize a shopping list, have all been met. Scanning barcodes was fairly simple as Google already provided an API to do this.

The other goals were accomplished by building a database system to store pricing information and building the APIs to allow our application to understand it. The following figures show the screens used to search for items, update prices, and manage a user's shopping cart, respectively.

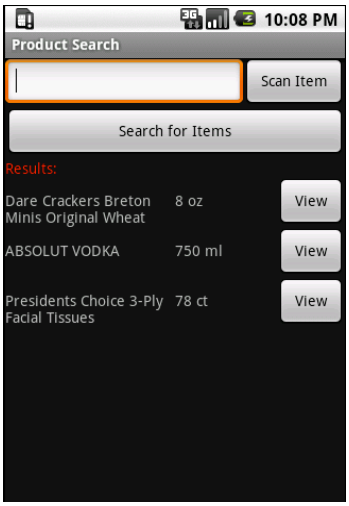

Figure 3. Search Screen.

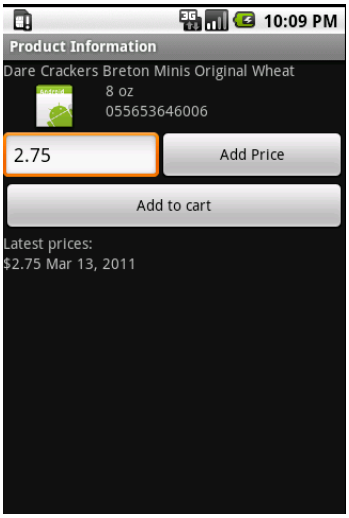

Figure 4. View Item Screen.

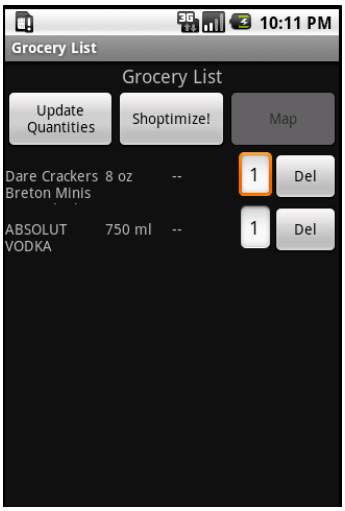

Figure 5. Grocery List Screen.

Two of the additional "nice-to-have" goals we had in mind when we started this project, geo-coded location awareness and plotted routes, were also successfully implemented using Google's provided geo-coding and GPS API and Maps API, respectively.

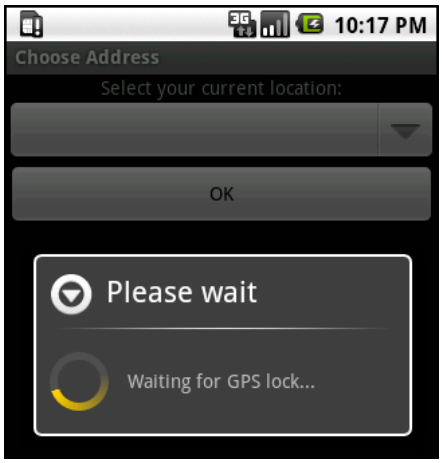

Figure 6. Location Screen Shot.

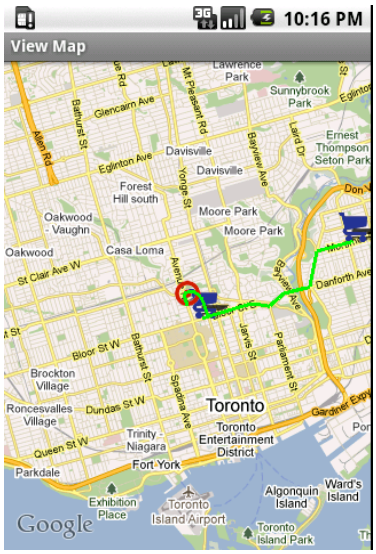

Figure 7. Map View Screen Shot.

Early on in development, we recognized that the problem of minimizing cost for a basket of goods is trivial if we assume the user doesn't want to go to more than one store, and we considered multi-store optimization a "nice-to-have" feature. In the final product, we actually did attempt to solve this problem and it seems that our solution does work fairly well in practice.

As both team members have a background in FPGA's, we instinctively imagined each grocery item as a "circuit element", the possible locations as the "FPGA fabric", and the routes between them as "wires", and realized that this problem is actually fairly similar to FPGA place-and-route. Thus, we split up the problem into two sub-problems: Placement and Routing.

As each item can only be placed in a relatively small number of locations, we realized that the placement problem is similar, though orders of magnitude simpler, than in FPGA's. Thus, we chose an approach known as exhaustive enumeration, which is simply to enumerate all possible solutions and then pick the best one. In this case, a "solution" was a legal placement of all grocery items to some location, and the solution space was simply the Cartesian product of all possible placements for each individual item. To each solution, we applied a cost function, which was simply:

$$Cost = \sum_{i=0}^{N} C_i + D * C_D$$

Where C_i is the cost of each individual item at that location, D is the total distance to travel from the user's current location to each location, and C_D is the cost of covering that distance. The nice thing about this algorithm is that it's simple, and is guaranteed to find the

globally optimal solution. It also allows room for future expansion, as changing the behavior of the algorithm could be accomplished fairly easily by messing around with the cost function. For example, C_D is currently just the current cost of gas. If a user really didn't like driving, we could tune the algorithm by simply increasing C_D, which would make the algorithm favor less travelling.

The other part of the optimization algorithm is routing, but because the problem is to simply minimize the total distance, the solution is actually fairly trivial, and is implemented as a hill-climbing or greedy router, where each successive hop is chosen based on shortest straight-line distance to the next location. Not only is this algorithm simple and fast, it also guarantees shortest total travel time.

After we implemented this, we found that the algorithm performed as expected, and was in fact successful at finding optimal solutions within a reasonable time.

What We Learned

While there were the normal challenges encountered in learning the Google Android framework, there weren't a lot of technical obstacles we couldn't overcome, and this was a bit expected as an all-programmer team. Also as expected, we both learned a tremendous amount about programming for mobile devices. However, what surprised both of us was how difficult we found making an attractive, easy-to-use UI. Even in its final form, Shoptimus Prime is functional, but not especially pretty, and when compared to what we saw in other groups' demos, we realized that our application definitely leaves something to be desired in the aesthetics category. If we were to do this project again, we would more aggressively seek out an Apper, especially one with visual design experience to provide a visual design skill set that we clearly don't have.

Group Member Contributions

Our group is composed of two members with programming backgrounds: Michael Kipper and Bryce Leung. We had a bit of a weird design flow in this project, as only Michael owned an Android phone, so Bryce had to do all development on the emulator. As such, tasks which required access to an actual phone were done by Michael. Specifically, he wrote all the code related to accessing the phone's GPS, camera, anything to do with scanning, geo-coded locations, and communicating with the online UPC database. Bryce was responsible mainly for the database back-end, the Shoptimization algorithm, the maps and route plotting code. The GUI was created by both team members on an as-needed basis, but the majority of it was written by Michael as he seemed to have infinitely more patience for figuring out how to make the buttons go where he wants them to than Bryce did.

Future Direction

Working on this app was a surprisingly enjoyable experience on top of being educational. The biggest thing that needs to be improved in the future for this app is ergonomics. People will only use this app if it's dead simple to use, so a lot of thought will need to be put into the user experience and how to make that as seamless as possible. In particular, Optical Character Recognition (OCR) would greatly enhance ease-of-use, especially for people scanning prices, as being able to wave the phone in front of a price tag would make it considerably easier to scan multiple items in succession. Also, the search functionality is extremely rudimentary, and is currently little more than substring matching. To continue the FPGA analogy, this can be likened to technology mapping where a user's entry of "Rice" can be mapped to all brands of rice. Finally, we would have to do something to deal with items that inherently have no bar-codes, such as fruit and vegetables. Those items have been completely ignored in this prototype.

Once those problems have been solved, this product does have significant commercial value. As mentioned before, this product does offer something that no other shopping app offers, which is that it works with stores even if they don't sell their products online. The problem is that, as with any crowd-sourced project, there's a challenge reaching the "critical mass" threshold where there'd be enough information in the database to make the app useful, and before that happens it would be difficult to sell, so there's a "chicken-and-the-egg" problem that needs to be addressed.

Word Count: 2435