16_0629 CSCI 161 Group Project – Grocery Store Navigation App
How the Navigation Section of the project works

StorePlan

StorePlan is a Plan object of a particular store layout. It visually represents the floor plan of the store in a GUI or Applet. The StorePlan layout has two main parts – Bin objects, and Point objects.

A Bin object is a representation of a single horizontal segment of a shelf in a store aisle. A Bin object is visually represented in a GUI or Applet as a filled-in square, and the StorePlan groups Bin objects together to represent a floor plan. When created, each Bin is given a unique identification name, which will be used to place Items throughout the store, and navigate to them. Bin objects also contain data about the floor space in front of them, which will be needed later to match Bin objects to graph vertices for navigation.

A Point object is a representation of an x-y coordinate in two dimensions. Points are not visually represented themselves, but they form the basis of how the graph data structure is created for navigation. The StorePlan manually constructs a Point object to represent the exact location of each intersection in the paths around the store (where aisles cross), as well as the entrance, and the checkout. There is a Point object for each middle in a vertical aisle as well (for simplicity of graph construction).

In this FloorPlan, the store has a single shelf that wraps around its back (made of a single row of Bins), and multiple aisles for the rest (with Bins on both sides). Other parts of the visual floor plan are made of drawn shapes and outlines within the FloorPlan paint() method. The FloorPlan is designed that it can be scaled to fit any window, and the Bin and Point objects will still be correctly positioned. It can also be shifted anywhere in a window, and the Bin objects can be painted any given color.

The FloorPlan also has a method to visually indicate which Bin objects contain Items that the User has in his / her Cart.

GraphNavEngine

GraphNavEngine is a NavEngine built in the form of a graph data structure. For a given StorePlan object, GraphNavEngine constructs a graph consisting of vertices and edges. The graph has a vertex for every Point object in the StorePlan, as well as a given number of evenly spaced positions between these. The vertices are spaced to correspond to the dimension between the Bins, so that a single vertex lines up with at most two Bins (one on either side or one above and one below). Edges are then constructed between each vertex (see attached image), so that the graph is connected, and each edge is given a weight that is the distance between vertices.

When a GraphNavEngine object is constructed, it also constructs a matrix of distances from each vertex in the graph to every other vertex in the graph. It stores this information for use when navigating.

The graph and matrix provide a means of calculating the shortest route between a given number of unordered Bin objects, when starting at the Entrance and ending at the Checkout.

It first sets the Entrance, the Checkout, and any number of Bin objects as destinations. If the number of destinations to visit is small (less than 10), the class contains a means to find the shortest path connecting these destinations using brute force. It compares the path length of every single combination of Bin orders, and outputs the shortest path.

If the number of Bins to visit is larger, the class can also compute the shortest path, but does so in a two-fold way (brute force has an exponential running time, and above 10-12 destinations, the time for calculation is beyond desired wait time). First, it groups the Bin objects into local clusters, where each cluster has no more than 10 destinations to visit. It orders the clusters so that the cluster containing the Entrance is always first, and the cluster containing the Checkout is always last. The middle clusters are sorted right to left, so that the cluster containing the rightmost destinations is second on the list, and the cluster containing the leftmost destinations is second to last on the list.

Starting with the Entrance cluster, GraphNavEngine uses brute force to calculate the shortest path between the destinations in the cluster, always starting with the Entrance. It adds the result to a running list. Then it takes the last destination from that cluster and adds it to the next cluster.

With the next cluster, brute force is again used to calculate the shortest path between its destinations, and this time it starts with the last destination in from the previous cluster. The result is added to the same running list.

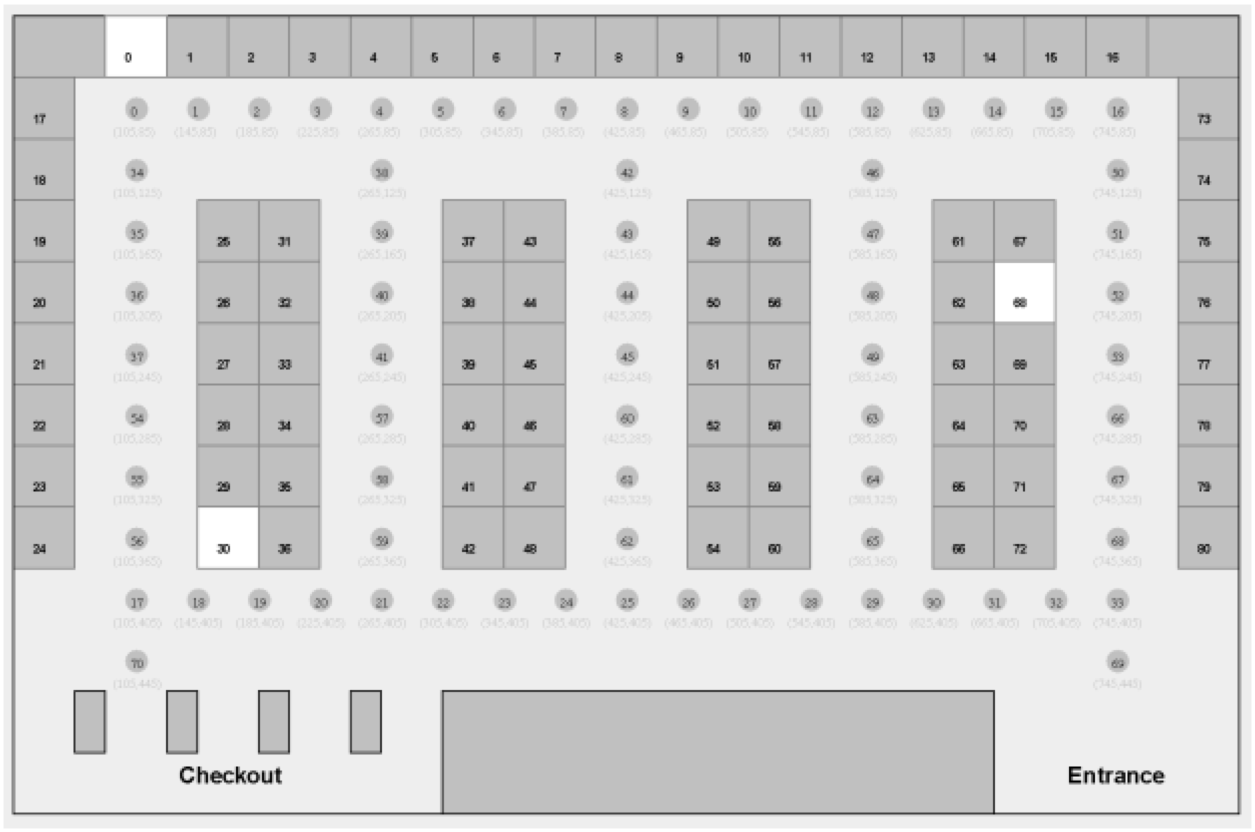
This continues until the Checkout cluster. Here, brute force is used, and the starting destination (which represents the last destination from the previous cluster) and the ending destination (the checkout) remain first and last, respectively. This result is added to the same running list, and now all destinations have been put in the list in proper order.

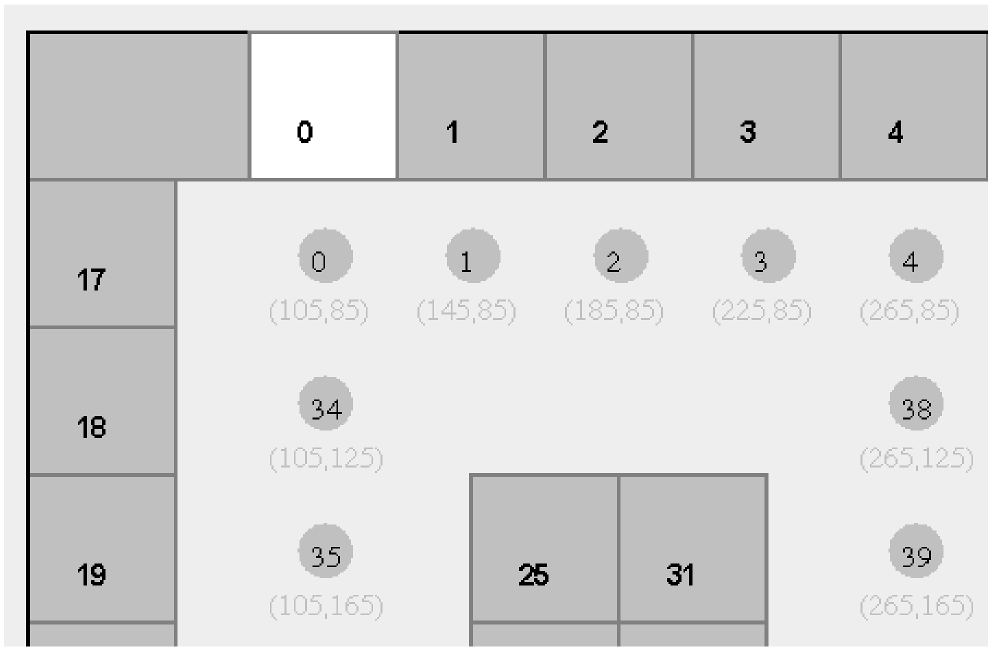
The NavGraphEngine then can take this ordered list of destinations, and translate it into an ordered shopping list. The shopping list is a set of directions. It specifies the order and location of each Item to get in the store in order to pick up all desired items in the most efficient route.

GraphNavEngine uses a number of data structures:

- Graph (as AdjacencyMapGraph)
- Map (for AdjacencyMapGraph)
- PositionalList (for AdjacencyMapGraph)
- AdaptablePriorityQueue / PriorityQueue (for AdjacencyMapGraph)
- Queue (for clustering)

Image of Graph visually represented over StorePlan
Vertices are represented by spheres. Each vertex has a unique identification numer, and a set of coordinates that correspond to the position of the vertex in the StorePlan space.


Checkout
Entrance


0
1
2
3
4
17
18
19
0
(105,85)
1
(145,85)
2
(185,85)
3
(225,85)
4
(265,85)
34
(105,125)
38
(265,125)
35
(105,165)
25
31
39
(265,165)