Project: Auto-Shopper

Project Description and Scope

The project name is Auto-Shopper. A smart shopping cart that basically waits for items to be addressed to it. Once they are put in, it utilizes the self-driving capability to locate the items in the aisles. Once the user collects the item (/s), the cost is summed up and can be paid at the counter.

The scope of this project was to create a software blueprint for a self-checkout cart to help with issues like reducing queues in supermarkets, calculating the cost of products, increasing the customer's convenience during the time of shopping, and finally, introducing a more futuristic outlook on the traditional supermarket settings.

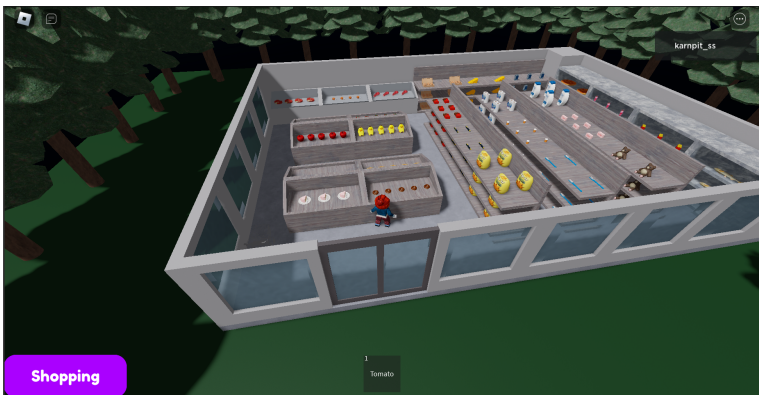
Solution

We tackled the above problems by adding auto-navigation in-store for the shopping cart to easily traverse paths A to B. We also added a calculating system which basically is registered for each and every product in the store. Its price is already pre-assigned and calculated once the user chooses. Since our project is based in the Virtual World, it's pretty similar to a Robot Simulator. Hence acting as a blueprint on the software side.

User Interface

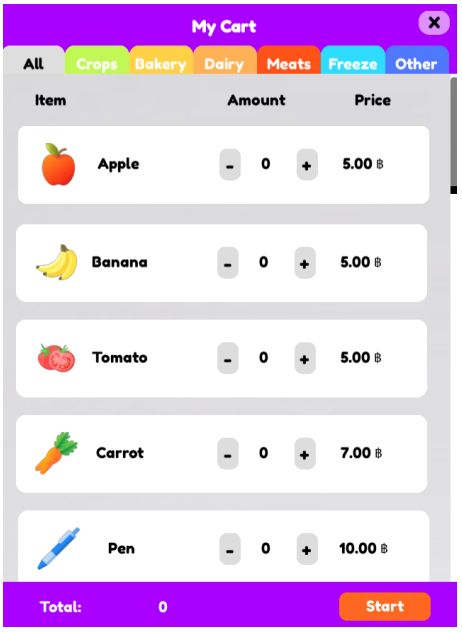
In the simulation, the user is able to select the items that they want to buy and see the total cost of it. After the user finishes selecting all items, then the cart is going to move according to the positions of the items. This can be performed by the following steps.

1. Click on the 'Shopping' button as shown in figure [a.1].



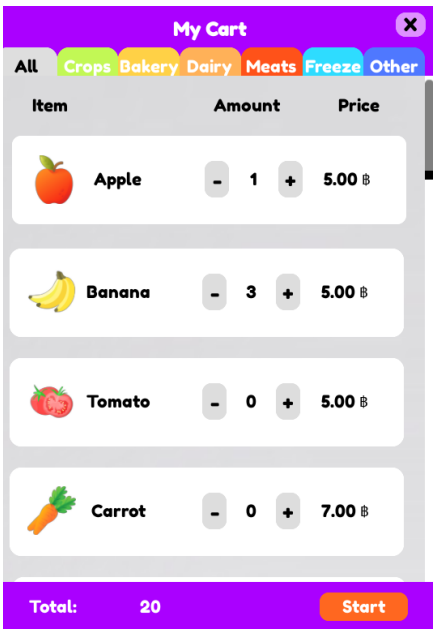
[a.1] Showing the start screen with the ‘Shopping’ button

2. The interface with a list of items pops up on the screen. There is an image, name, amount, and price of the item shown in each box. At the bottom, there is a box showing the total cost of the selected items as shown in the figure below [a.2]



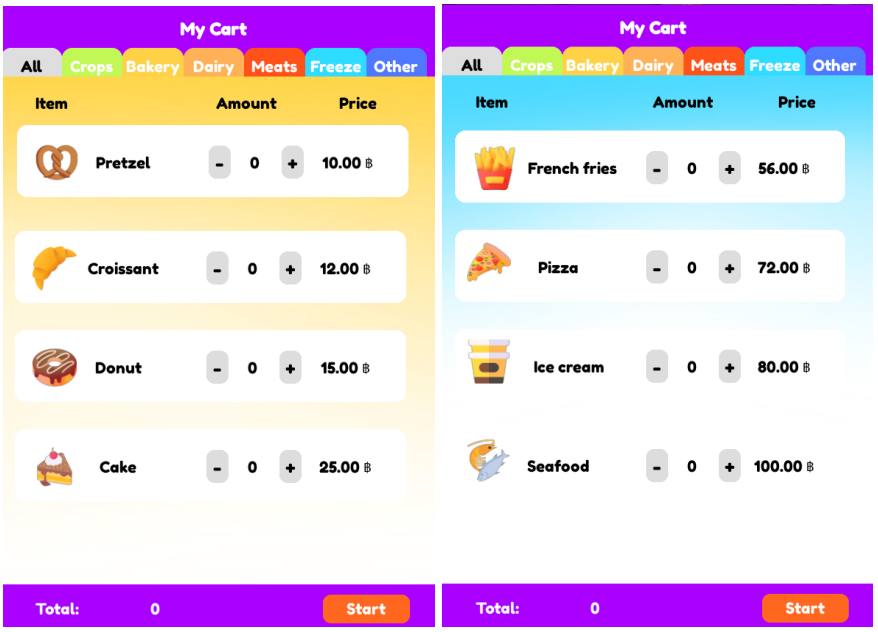
[a.2] The list of items that pop up

3. Clicking on a ‘+’ button in each box to add the item and on a ‘-’ button to remove the item. The number of items shown between the ‘+’ and ‘-’ buttons update according to the user’s interaction. The total cost is also updated As shown in [a.3]



[a.3] The updated list

4. If the user clicks on category boxes at the top, it will show only the item in the selected category as in the figure [a.4, a.5].

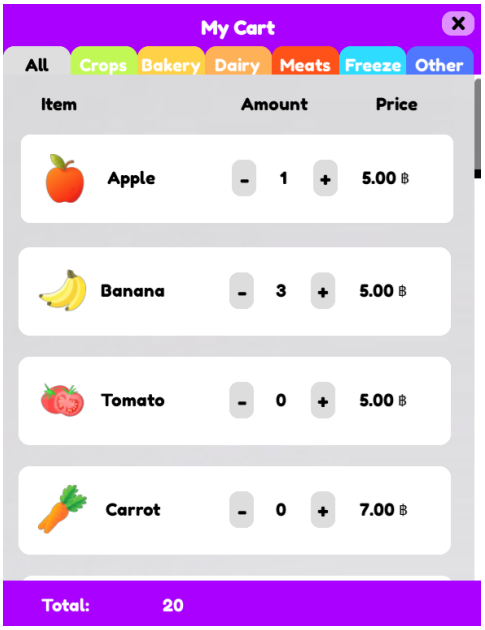


[a.4, a.5] The interface after selecting the category

5. If the user clicks the ‘Close’ button on the top right of the interface then it will go back to the ‘Shopping’ button.

6. After the user finishes selecting items, click the ‘Start’ button at the bottom right. It will show the ‘Shopping’ button and if the user clicks it then it will show all items with the amount of its

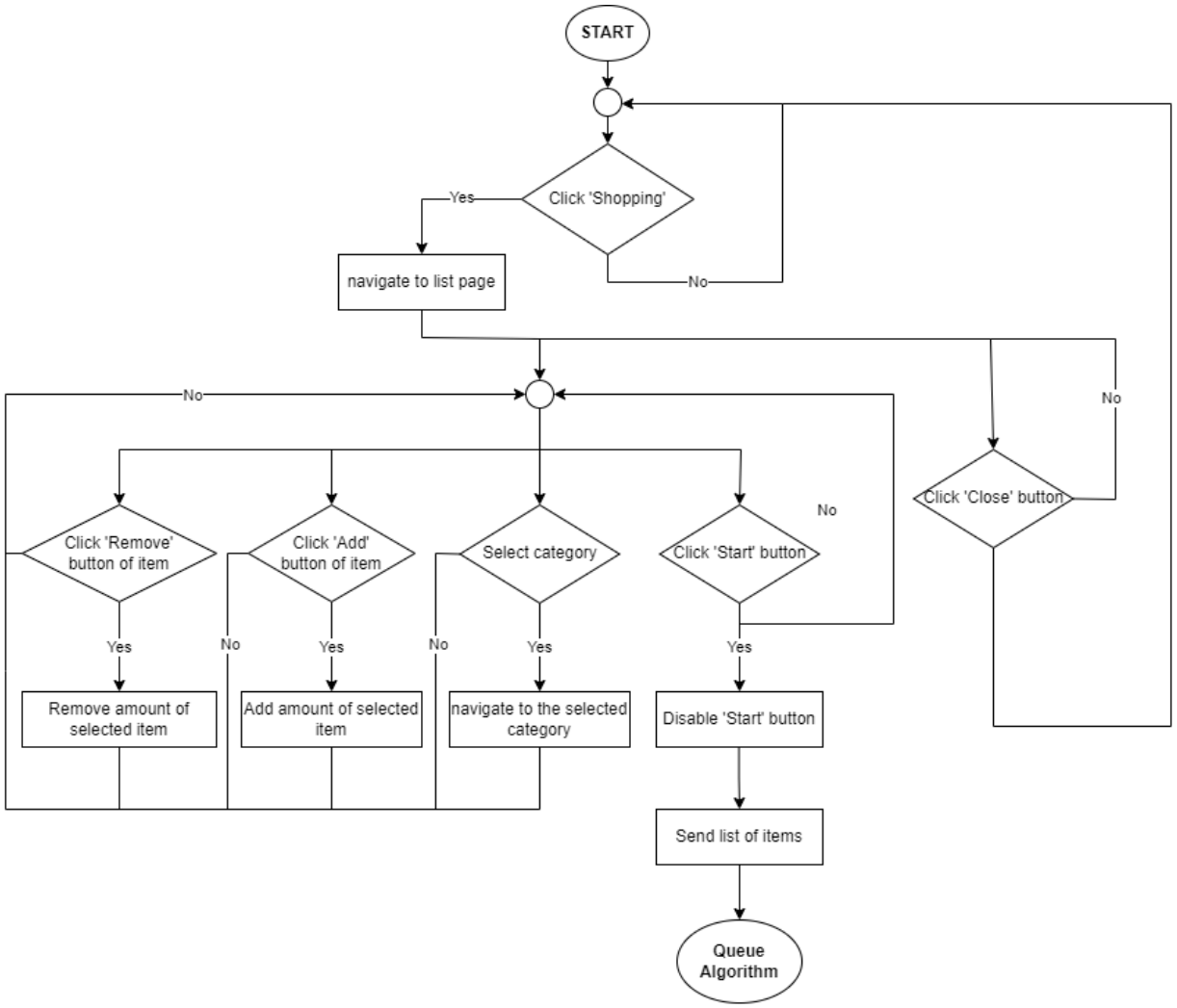
(including 0) and the ‘Start’ button will be disabled as shown in the figure below [a.6]. and the list of items is going to be sent for calculating the shortest path that is able to collect all selected items.



[a.6] The interface after clicking the ‘Start’ button

7. When the cart comes back to the start point, the interface is going to be refreshed. The amount of items is going to reset as 0 as well as the total cost. So, the user will be able to select items again.

The flowchart for User Interface



Shortest Path Calculating Algorithm

The purpose of this algorithm is to find the shortest path between every position in the supermarket and then store all data in a text file. Because these data are constant numbers, storing them and calling them later will help reduce running time when calculating the whole path.

At first, for simplicity, we break the supermarket map into unit squares, so that each unit square counts as 1 unit distance away from the other. Then name each position that refers to each product. [b.1]

[b.1] Supermarket map in unit squares

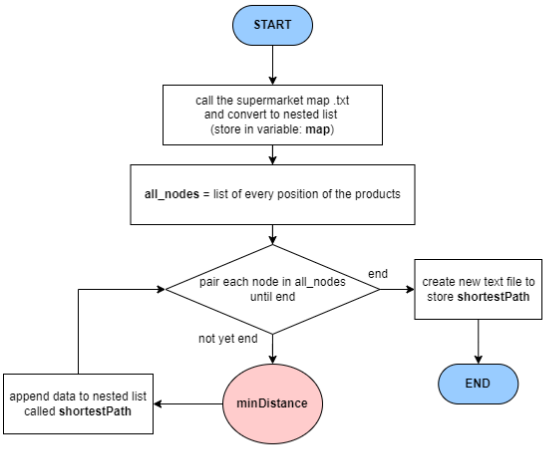
From the position names, we will call them nodes. There are a total of 27 nodes on the map. Then we store them in a text file to easily use to calculate.[b.2]

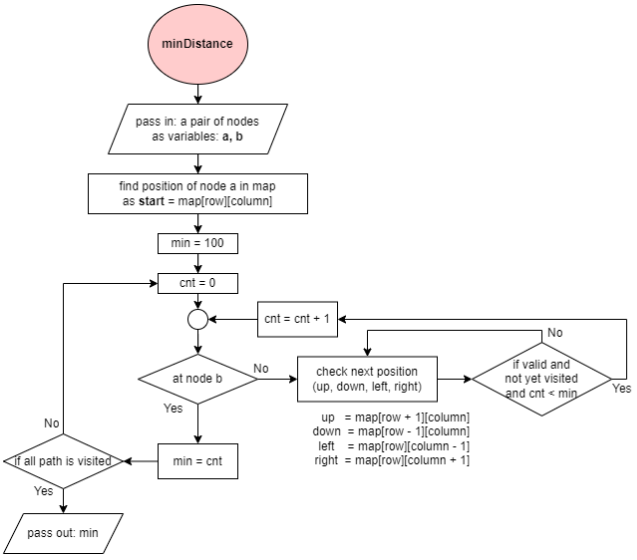
```
m1 m1 m2 m2 m3 m3 d1 d1 d2 d2 d3 d3 --
-- -- -- -- -- -- -- -- -- -- -- -- f1
-- v1 v1 v2 v2 -- t1 -- t4 -- t7 -- f1
-- v3 v3 v4 v4 -- t1 -- t4 -- t7 -- f2
-- -- -- -- -- -- t2 -- t5 -- t8 -- f2
-- b1 b1 b2 b2 -- t2 -- t5 -- t8 -- f3
-- b3 b3 b4 b4 -- t3 -- t6 -- t9 -- f3
-- -- -- -- -- -- t3 -- t6 -- t9 -- f4
-- -- -- -- -- -- -- -- -- -- -- -- f4
```

[b.2] Supermarket map to a text file

Then for the algorithm finding the shortest path between each node, we use python. By pairing each node in the map and finding paths that could go from one node to another. If it is the shortest or has the minimum distance, then store the value of distance in another text file called shorestPath.txt.

Flowchart of Algorithm:





When this algorithm is applied to every node in the map, we get the paths from 28 nodes to 28 nodes (including the starting point that is at the bottom left). And to easily see the picture we put it in excel.[b.3]

	ss	m1	m2	m3	d1	d2	d3	v1	v2	v3	v4	b1	b2	b3	b4	t1	t2	t3	t4	t5	t6	t7	t8	t9	f1	f2	f3	f4
ss	0	7	9	11	13	15	17	6	10	5	7	3	7	2	4	10	8	6	12	10	8	14	12	10	17	15	13	11
m1	7	0	1	3	5	7	9	0	2	2	6	4	6	5	9	5	7	9	7	9	11	9	11	13	10	12	14	16
m2	9	1	0	1	3	5	7	0	0	4	4	6	6	7	7	3	5	7	5	7	9	7	9	11	8	10	12	14
m3	11	3	1	0	1	3	5	2	0	6	2	6	4	9	5	1	3	5	3	5	7	5	7	9	6	8	10	12
d1	13	5	3	1	0	1	3	4	2	7	3	7	5	10	6	2	4	6	1	3	5	3	5	7	4	6	8	10
d2	15	7	5	3	1	0	1	6	4	9	5	9	7	12	8	2	4	6	2	4	6	1	3	5	2	4	6	8
d3	17	9	7	5	3	1	0	8	6	11	7	11	9	14	10	4	6	8	2	4	6	2	4	6	0	2	4	6
v1	6	0	0	2	4	6	8	0	1	1	5	3	5	4	8	4	6	8	6	8	10	8	10	12	9	11	13	15
v2	10	2	0	0	2	4	6	1	0	5	1	5	3	8	4	0	2	4	4	6	8	6	8	10	7	9	11	12
v3	5	2	4	6	7	9	11	1	5	0	1	0	1	3	5	4	3	5	9	11	10	11	13	12	12	14	15	13
v4	7	6	4	2	3	5	7	5	1	1	0	1	0	5	3	0	1	3	5	7	8	7	9	10	8	10	12	11
b1	3	4	6	6	7	9	11	3	5	0	1	0	1	1	5	4	3	5	9	11	10	11	13	12	12	14	15	13
b2	7	6	6	4	5	7	9	5	3	1	0	1	0	5	1	2	0	1	7	8	6	9	10	8	10	12	11	9
b3	2	5	7	9	10	12	14	4	8	3	5	1	5	0	1	7	5	3	11	9	7	13	11	9	15	14	12	10
b4	4	9	7	5	6	8	10	8	4	5	3	5	1	1	0	3	1	0	8	7	5	10	9	7	11	12	10	8
t1	10	5	3	1	2	2	4	4	0	4	0	4	2	7	3	0	1	3	4	6	8	6	8	10	7	9	11	11
t2	8	7	5	3	4	4	6	6	2	3	1	3	0	5	1	1	0	1	6	8	6	8	10	8	9	11	11	9
t3	6	9	7	5	6	6	8	8	4	5	3	5	1	3	0	3	1	0	8	6	4	10	8	6	11	11	9	7
t4	12	7	5	3	1	2	2	6	4	9	5	9	7	11	8	4	6	8	0	1	3	4	6	8	5	7	9	9
t5	10	9	7	5	3	4	4	8	6	11	7	11	8	9	7	6	8	6	1	0	1	6	8	6	7	9	9	7
t6	8	11	9	7	5	6	6	10	8	10	8	10	6	7	5	8	6	4	3	1	0	8	6	4	9	9	7	5
t7	14	9	7	5	3	1	2	8	6	11	7	11	9	13	10	6	8	10	4	6	8	0	1	3	3	5	7	7
t8	12	11	9	7	5	3	4	10	8	13	9	13	10	11	9	8	10	8	6	8	6	1	0	1	5	7	7	5
t9	10	13	11	9	7	5	6	12	10	12	10	12	8	9	7	10	8	6	8	6	4	3	1	0	7	7	5	3
f1	17	10	8	6	4	2	0	9	7	12	8	12	10	15	11	7	9	11	5	7	9	3	5	7	0	1	3	5
f2	15	12	10	8	6	4	2	11	9	14	10	14	12	14	12	9	11	11	7	9	9	5	7	7	1	0	1	3
f3	13	14	12	10	8	6	4	13	11	15	12	15	11	12	10	11	11	9	9	9	7	7	7	5	3	1	0	1
f4	11	16	14	12	10	8	6	15	12	13	11	13	9	10	8	11	9	7	9	7	5	7	5	3	5	3	1	0

[b.3] Shortest path between each node

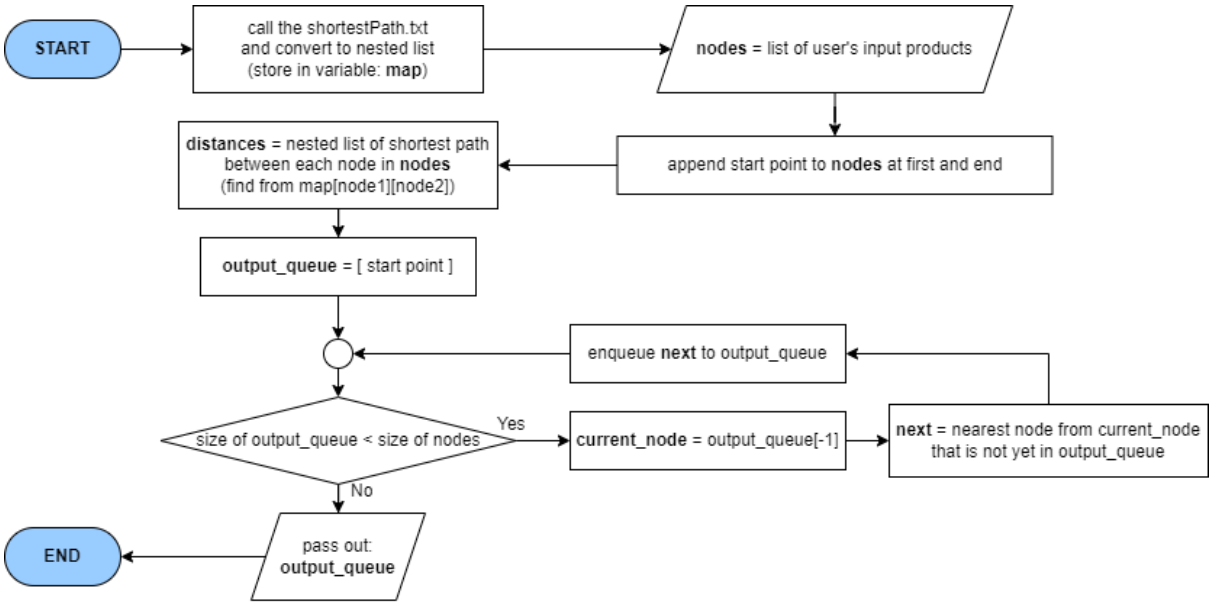
Queue Order of Products Algorithm

After the user has inputted the list of products through the UI, this algorithm will queue the products into order from starting point to the next product to pick up and next until it reaches all products in the list, then comes back to the starting point to end the process.

For this algorithm, we choose Dijkstra's Algorithm to find the shortest path of all nodes in the shopping list. And we choose to use the Lua language

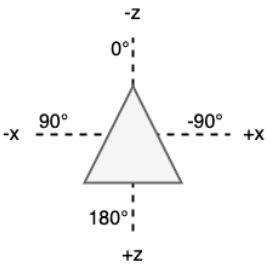
Dijkstra's Algorithm is to find the shortest paths between nodes in graphs, by choosing the minimum weight, until it covers the path from the start to the endpoint. And from that, we also add to get all the required nodes in the list. After finding the next minimum node then enqueue the node into our queue until reaches all nodes in the list. Then pass the queue to the cart moving algorithm.

Flowchart of Algorithm:



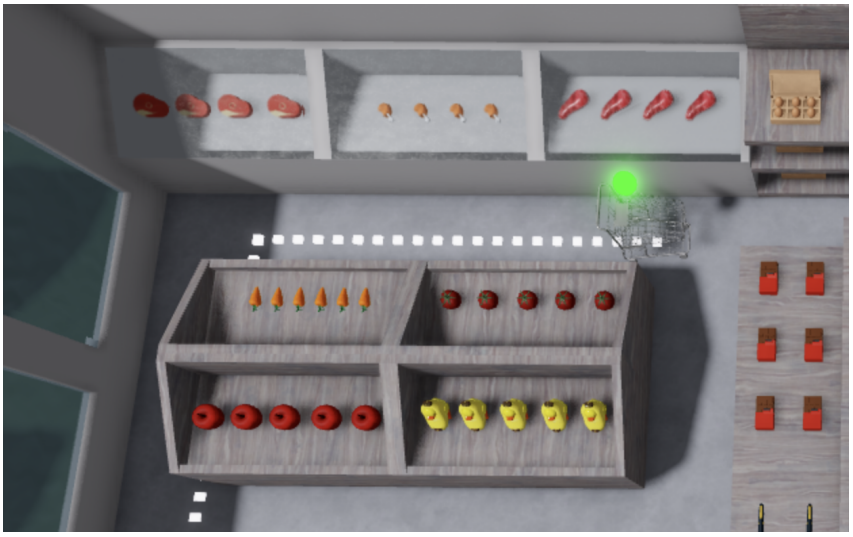
Cart Movement Algorithm

In this project, there are two Roblox services that are used: PathfindingService and TweenService. PathfindingService is used to find the minimum path from one position to another considering the obstacle in the environment. TweenService is used to create an animation for an object from its position, orientation, size, color, or transparency difference from its current value.



[c.1] Cart angle diagram (upper view)

Another thing to keep in mind is the orientation value of the cart with 0 degrees at the front, 90 degrees on the left side, -90 degrees on the right side, and 180 degrees at the rear – the figure is shown in [c.1].



[c.2] Picture of cart with path marking (white dot) and halt signal (green ball)

- Movement

  The Roblox PathfindingService returns a list of waypoints, where they are the positions of the path considering the obstacle in the environment. Using the TweenService to create an animation that moves the cart from one point to another point in the waypoints with a new position and orientation parameter.

- Angle Calculation

  The angle is calculated from the arctan of the x-axis difference over the z-axis difference “arctan(Δx / Δz)”, where the difference is found from the difference between the new position and the current position. To integrate the calculated angle with the game scenario, the angle has to be adjusted according to the orientation of the cart – the figure is shown in [c.1]. Without angle calculation, the cart movement would look unrealistic.

- Path Marking

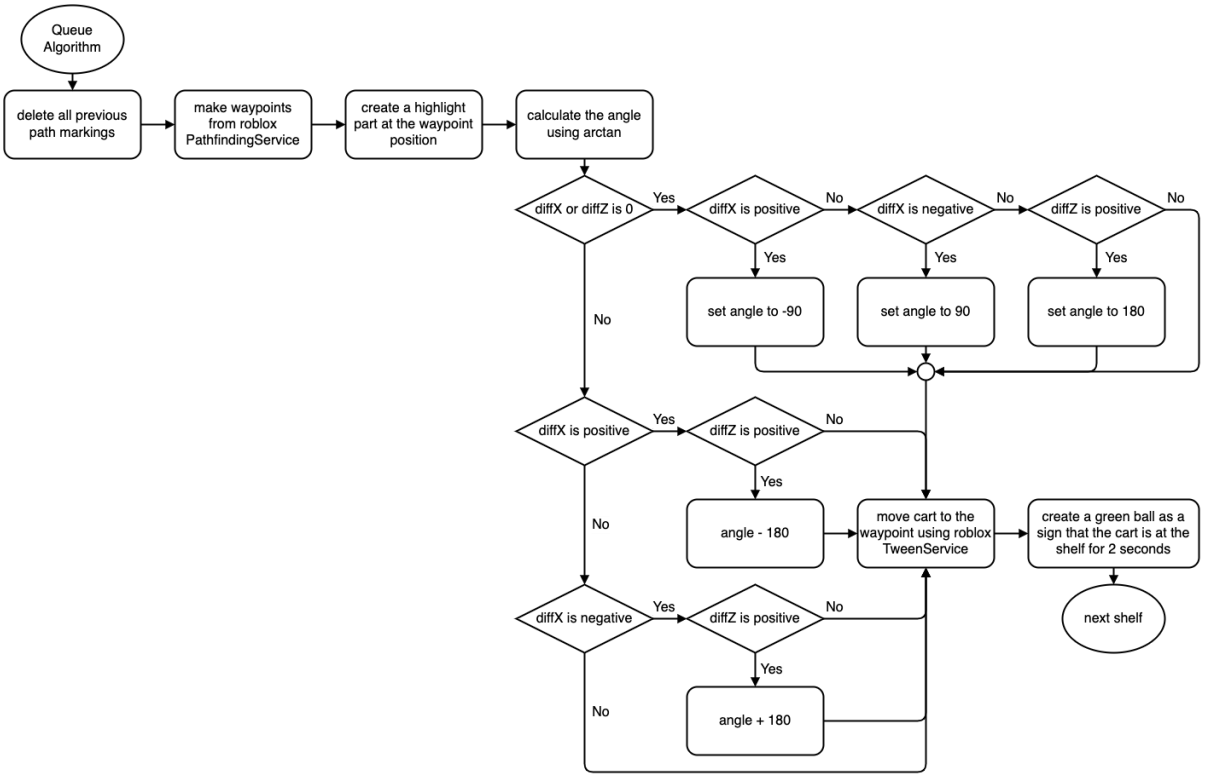
  While the cart moves from one point to another, a white dot is marked at that position – the figure is shown in [c.2]. In the end, the user

can track the path using this marking. However, when starting a new path from the new shopping list, the previous path will be deleted.

- Halt Signal

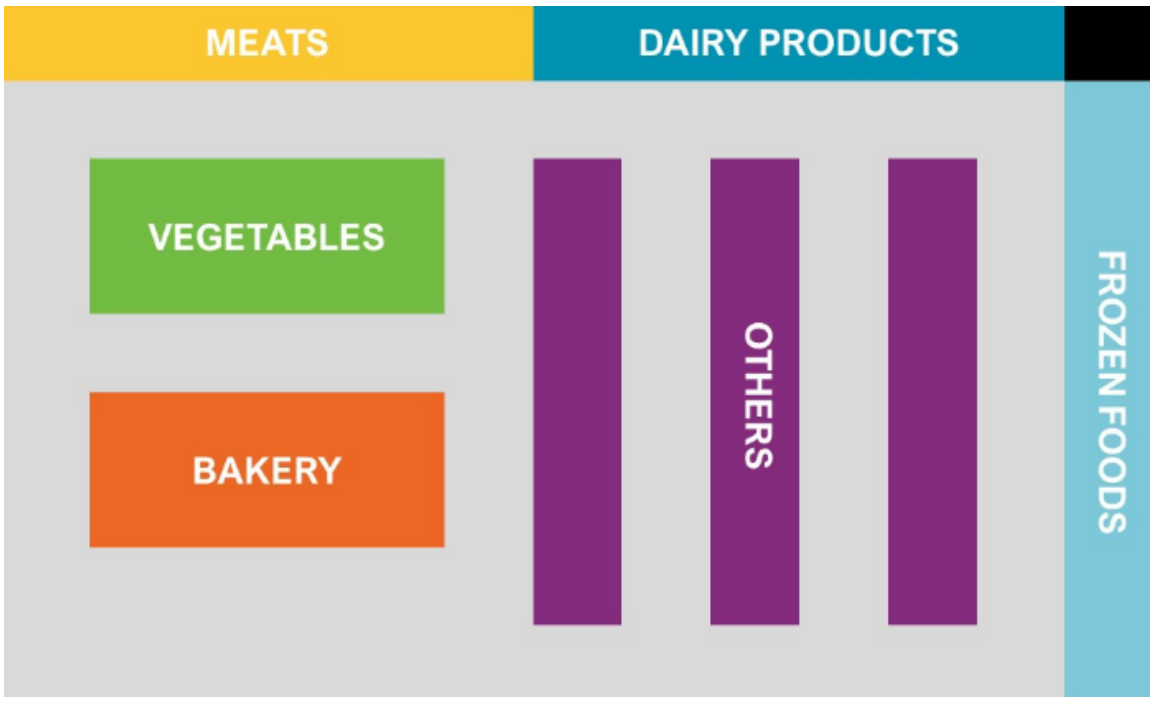
  To make a sign that the cart is already at the shelf position, a glowing green ball will be created above the cart and deleted after 2 seconds when the cart starts moving to another point – the figure is shown in [c.2].

Flowchart of Algorithm:



Results

The simulation is successfully built in Roblox studio. This program lets you create 3D objects and transform 3D objects freely. Moreover, the codes edit or written to create algorithms and programmes objects. An interactable shop is successfully constructed in figure [d.1], with the design of different alleys for up to 28 different types of products. The space between each block or shelf is a fit for one cart. The entrance of the shop is on the bottom left side of the map below the bakery zone. Figure [d.2] below, displays the demonstration of the shop in the simulation.



[d.1] The map design of the shop.

[d.2] The shop in the simulation.

The auto-shopper or the cart is placed by the shop entrance, recognising that position as the starting position. The shopping button on the bottom left of the screen, allows the user to view and select a list of wanted products, which the total cost will be calculated and displayed automatically. The list of products will then be processed to move the cart to the closest product in the list

received, using the list of positions, that has been calculated to be the shortest route. After the cart arrived at the position of the product's shelf. The movement will become stationary, followed by a green circle light that will appear for 3 seconds above the cart. Sending a signal to the user to pick up the product. Moments after, the cart will automatically move to the next closest product on the list and loop through the process until all the items on the list are visited.

After all items on the lists are visited, the cart will automatically be returned to its starting point, by the entrance door. After that, the user can interact with the auto-shopper and select another list of products again.

Work distribution

User Interface: 64011423 Karnpitcha Kasensirinavin

Path Finding: 64011467 Natasha Greenough

Cart Movement: 64011664 Thanakin Chakanjsilp

Products UI and Map Design: 64011727 Chalita Thongborisut

Designing Environment: 64011748 Ahmad Sohail