Group 10: Factory Simulation

Devin Shingadia | 170036317

Jacob Williams | 170050173

Mohammed Hamza Zaman | 170107477

Miraj Shah | 160050110

Christos Dolopikos | 170116343

Vivek Bhukhan | 159087932

Abstract:

This report will provide an in-depth view of the factory simulation that we have developed, highlighting, how the behaviour has been implemented, and the reaction of the simulation to various parameters, inputted through a ‘.SIM’ file. It will be structured as follows: there will be an outline of the various behaviours that have been implemented, followed by a conclusion which summarises the behaviour of the entities that we have created.

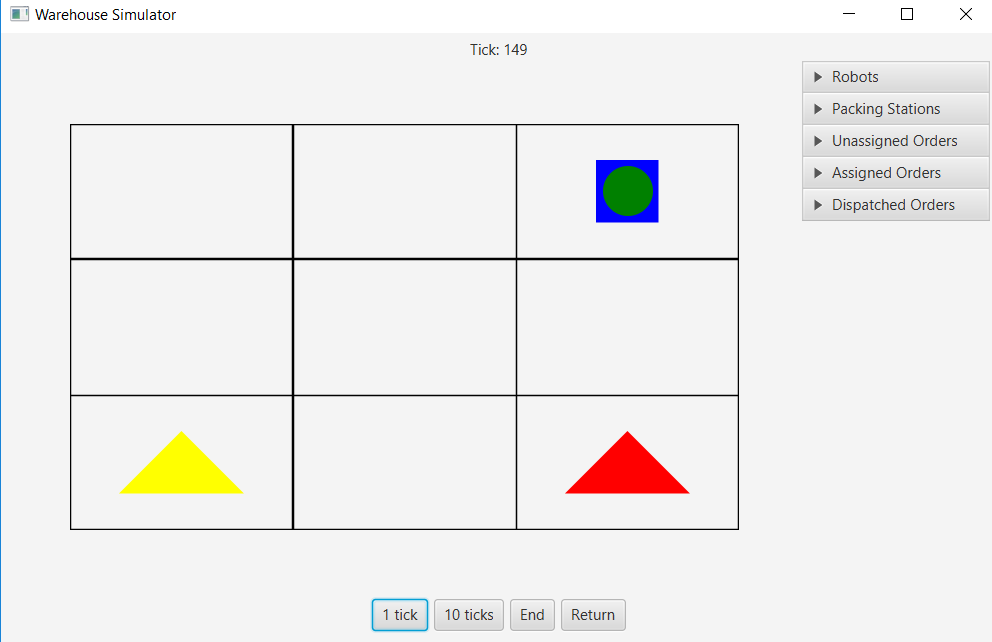
A report on what happens in the simulation as you change the different parameters. For instance, how do your robots react to configurations that have more overlap between their usual paths? How much of a safety margin do you need for the batteries in your robots?

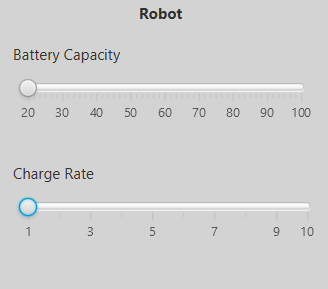
# Experiments

In this test I will be comparing the number of ticks required to carry out the orders that have been specified in the ‘OneOfEverything.SIM’ File. This file will be used to compare the effect of charge rate and battery capacity on the number of ticks that are required to carry out the order.

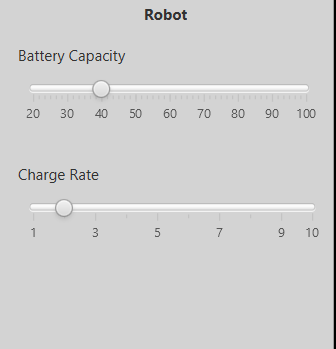
**Battery Capacity: 20**

**Charge Rate: 1**





.SIM file used: OneOfEverything.SIM

Result : With the lowest battery capacity and charge rate, the number of ticks required to carry out this activity was 149.

**Battery Capacity: 40**

**Charge rate: 2**

In theory, the robot should be able to carry out work in fewer ticks as it does not require to charge as often. Furthermore, whenever it does need to charge, the amount of time required to charge should be halved, as a result of doubling the charge rate.

# Discussion of the results

The results of the experiments show that most of our system is working as intended. Every feature that has been experimented works to some extent. Some features, such as the listView shown in the simulator view, for the robot information, does not work as intended. This is a result of lacking the knowledge to update the Coordinates of the robot in real time. The tests have shown us that there are some inadequacies in our code, as some parts do not perform the function that is intended, highlighting gaps in our knowledge.

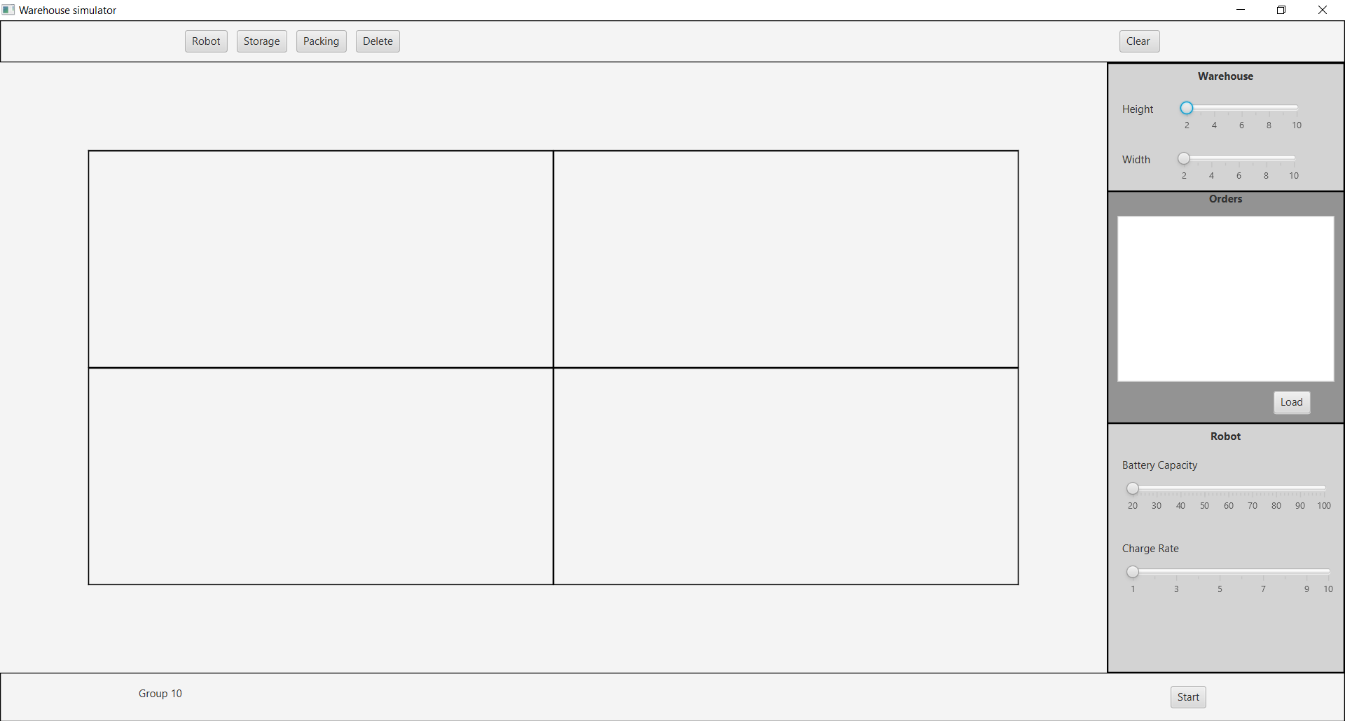
# The behaviour of the simulation

**The main view**

Add/Delete entities to the grid. Adding these entities will also create distinct objects.

Clear the grid

Adjust height and width of grid



‘.SIM’ File displayed here.

Load the ‘.SIM’ FIle

Set the battery capacity of the robot.

The grid is where entities can be added and removed. It can also show a preview of the simulation before it is run.

Open the simulator view

Set the charge rate of the robot

This is the first user interface that the user will see when starting the program. From here, the user can interact with the grid to create their own simulation, with their own defined, custom parameters, or they can load from a ‘.SIM’ File which predefines the parameters used in the simulation. After the user is satisfied with their simulation settings, they can press the ‘Start’ button and then view the simulation in action.

**The simulator view**

Listviews representing the behaviour and entities.



Gridview showing the entities in the simulation. These entities will become animated as the user presses ‘1 Tick’, ’10 Ticks’

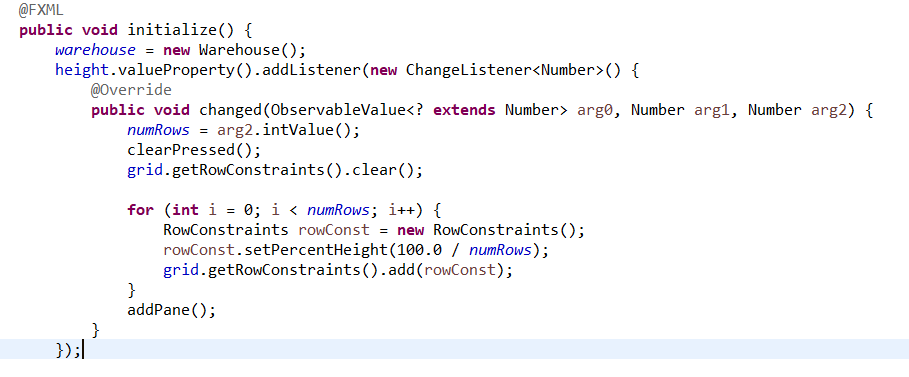
End the simulation or return to the main view.

Advance through the simulation; 1 tick at a time or 10 ticks at time

**The slider behaviour**



The slider object, named height.



Call ValueProperty() This returns the current value that is set by the user for the grid height.

Call a listener on the valueProperty(), which is a changeListener**.** The changelistener detects whether the valueProperty of the height slider has been changed. When it does, the field called numRowswill be changed.

Furthermore, when the slide is adjusted, the clearPressed() method is called, which will remove all entities from the grid.

Call the getRowConstraints method, which clears the X and Y coordinates held in the grid, so that it can be redrawn.