Evaluation

DC2300: Java Program Construction – Warehouse Coursework

The path finding algorithm and the algorithm of Robots calculating their decisions have evolved and been refined several times over the course of the project following evaluations on different parameters. We're going to briefly cover the way decisions were made and how this has affected those algorithms. We'll wrap up with the eventual results and discuss where improvements could have been made.

The earliest decision made was that path finding used in the simulation would be controlled through an interface, so that we could easily swap out different configurations. Research was carried out on a number of algorithms (Including A* and Cooperative pathfinding) and the eventual decision was we would produce a POC (Proof of concept) for a very basic algorithm we were comfortable with and then update it to a more advanced pathfinding algorithm at a later date. The POC we opted for was a BFS (Breath first search) algorithm as it was simple and would find the most efficient path although at a performance cost. However, we decided this trade off was worth it for the Robots performance.

Another early decision to be made was how we would test both the robot and pathfinding algorithms. We had unit tests for the individual methods, but this was not sufficient for analysing larger simulation runs and how our changes would affect any given parameters. For this reason, we opted to start with importing configurations for front end testing as it made it incredibly easy to rerun configurations that we were testing and view how they were affected by any changes to our algorithms. We also had several pre-provided configurations (See Appendix A) that aided with this process.

Once we were ready to begin consuming the pathfinding algorithm, we had to make another decision that would affect the robot's algorithm – should the robot dynamically calculate a new path each turn or calculate the path when the robot accepts the order and then check if they can make the next move each turn and wait if not.

The benefit of calculating it upfront was better code performance. It isn't looking for a new path each turn, just seeing if it can make its next move and making it. However, the downfall was it makes it very easy to encounter a deadlock scenario where two Robots want each other's space and never move.

We saw the benefit of dynamically calculating the path each turn as the robot being able to react to new obstacles and reroute if a space was no longer available, in order to keep the robot moving rather than waiting. We had also believed at this stage that it would negate the deadlock issue. However, this option came with the performance hit of recalculating each turn. We would also start running into scenarios where the robot may have rerouted in a way where they no longer had the battery to complete its path and needing to return to its charging pod.

After weighing up these options we opted for the dynamic calculation. We decided that the ability for the robots to keep reacting and pushing forward with their order would offer a better order delivery performance than them waiting and having to detect deadlock scenarios constantly.

Once this was agreed, an initial implementation was done for the robots' algorithms. At this initial stage there were two key algorithms:
Could the robot accept the order?
Could the robot continue the order?

Our algorithm for robots accepting the order can be seen in appendix B. This logic was solid and did not need altering on reviewing simulation run results.

The decision on if the robot could continue the order was the key one that was reviewed under simulation results.

Our first implementation of deciding if the robot could continue the order was looking at the current 'state' they were in (i.e. delivering an item, collecting an item…). From this point we then calculated their journey following much the same logic at the initial decision seen in appendix B, the key difference was that if a path could not be found we would not decide that they couldn't continue. Instead we would decide that their current path was just blocked and wait. This was because we had the knowledge that they had initially decided they could complete the order and the battery level hasn't changed, therefore by waiting they would likely be able to move next turn. If we decided they didn't have the battery to continue they would enter an emergency state and begin returning to their pod.

Once a robot had delivered to a packing station, we had them return to the charging pod after each order.

At this stage we ran the algorithms on all pre-provided configurations and found this worked. The results of this algorithm can be seen in the results table below:

Configuration	**Ticks to complete**
bottomStations.sim	1573
bottomStationsOldMac.sim	1573
bottomStationsWin.sim	1573
oneOfEverything.sim	1217
threeOfEverything.sim	1770
twoRobots.sim	1500
twoRobotsTwoShelves.sim	851
twoShelves-v2.sim	1484

After reviewing the spec, we acknowledged that there was no requirement for the robot to return to the charging pod after each order, and they could instead accept further requests. We believed if we implemented this change it could improve the speed of the simulation. The algorithm was updated so that the robot would consider an order the moment it no longer had a current assignment rather than waiting until it had returned and charged back up.

The initial implementation of this change offered improved performance results on most configurations. However, it did introduce the issue of several robots now running out of battery part way through the simulation.

Configuration	Previous ticks to complete	Ticks to complete
bottomStations.sim	1573	Robot: r0 has run out of battery at 260 ticks
bottomStationsOldMac.sim	1573	Robot: r0 has run out of battery at 260 ticks
bottomStationsWin.sim	1573	Robot: r0 has run out of battery at 260 ticks
oneOfEverything.sim	1217	1217
threeOfEverything.sim	1770	Robot: r2 has run out of battery at 801 ticks
twoRobots.sim	1500	1488
twoRobotsTwoShelves.sim	851	781
twoShelves-v2.sim	1484	1150

After reviewing the battery failures we discovered that this was because if a robot entered an emergency state from rerouting during a journey, if they were carrying an item it made their journey back to the pod much more costly than if they had dropped off the item first. To resolve this, we added a check for if the robot could return to the pod in their algorithm. This check was as simple as just calculating if they could make the journey if they rerouted to the pod in their current condition. If they couldn't they would instead wait a turn until a path they could afford became available for delivering the item. (We knew this was possible because they'd already calculated they could on the previous turn)

The updates from this configuration were:

Configuration	Previous ticks to complete	Newest ticks to complete
bottomStations.sim	Robot: r0 has run out of battery at 260 ticks	Never completes
bottomStationsOldMac.sim	Robot: r0 has run out of battery at 260 ticks	Never completes
bottomStationsWin.sim	Robot: r0 has run out of battery at 260 ticks	Never completes
oneOfEverything.sim	1217	1217
threeOfEverything.sim	Robot: r2 has run out of battery at 801 ticks	Never completes
twoRobots.sim	1488	1488
twoRobotsTwoShelves.sim	781	781
twoShelves-v2.sim	1150	1157

As you can see – this altered the issue to the simulations never completing. On investigating this was caused by a deadlock issue with two robots being in each other's target location. To negate this, we introduced a deadlock detector that will attempt to move the robot on the next step towards its

ultimate destination after 5 idle ticks regardless of if their target destination is blocked – providing that the next move itself would not result in the robot crashing.

Configuration	Original ticks to complete	Previous ticks to complete	Ticks to complete
bottomStations.sim	1573	Never completes	1108
bottomStationsOldMac.sim	1573	Never completes	1108
bottomStationsWin.sim	1573	Never completes	1108
oneOfEverything.sim	1217	1217	1217
threeOfEverything.sim	1770	Never completes	1305
twoRobots.sim	1500	1488	1488
twoRobotsTwoShelves.sim	851	781	781
twoShelves-v2.sim	1484	1157	1157

As you can see this final change provided working simulations for all configurations and greatly improved the simulation time to the original algorithm. The greatest improvement can be seen in bottomStations.sim - an improvement of 465 ticks!

Now we have analysed the algorithm and how the different considerations have improved this we can see how the different configurations within this affect the simulation parameters. We get a general feel for how the different parameters affect the simulation from our completion ticks table. But to really understand how the different parameters affect the simulation we have taken threeOfEverything.sim as a base line and adjusted the configurations within it. (This was selected as there are an equal number of all actors in play so it's easy to see how decrementing or incrementing affects this)

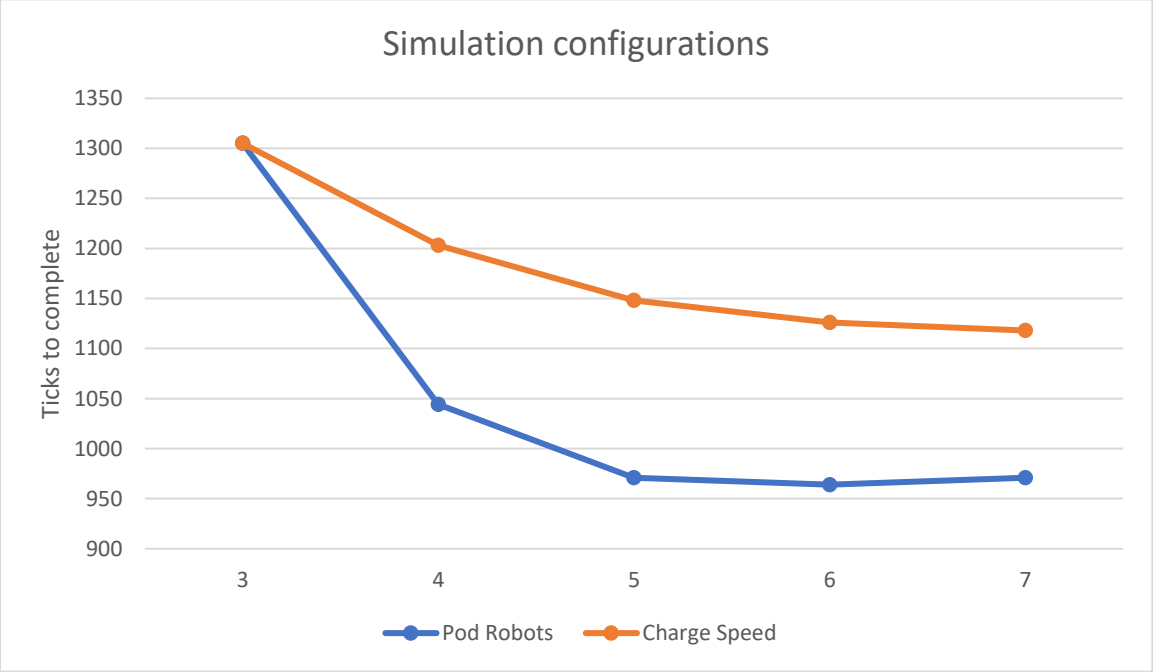
Configuration	Ticks to complete
ORIGINAL	1305
Capacity: 200	1206
Capacity: 50	1417
Charge Speed: 6	1126
Charge Speed: 2	1408
Pod robots: 6	964
Pod robots: 2	1832
Station: 6	1252
Station: 2	1359

As could be expected, reducing any of the values negatively affects the simulation run time. The greatest improvement comes from increasing the number of robots, followed by improving charge speed. If we look at these two configurations in more depth, we get the following outcomes:

Configuration	Ticks to complete
Pod Robots: 3 [ORIGINAL]	1305
Pod Robots: 4	1044
Pod Robots: 5	971
Pod Robots: 6	964
Pod Robots: 7	971

Configuration	Ticks to complete
Charge Speed: 3 [ORIGINAL]	1305
Charge Speed: 4	1203
Charge Speed: 5	1148
Charge Speed: 6	1126
Charge Speed: 7	1118

As you can see, once the ratio of robots to packing stations reaches 2:1, whereas increasing charging speed seems to always reduce the simulation time. The effect of the increase in robot numbers is dependent upon the number of packing stations in the simulation. 2 Robots to a packing station appears to be the optimal parameter, and then charging speed can continue to offer performance improvements (This improvement will logically stop when Robots can charge in a single tick). These results can also be visualised easier below:



It's also worth noting that increasing the number of robots further than 2:1 with packing stations results in a negative effect on the simulation. This appears to be a result of contention with too many robots attempting to navigate on the floor and having to avoid each other.

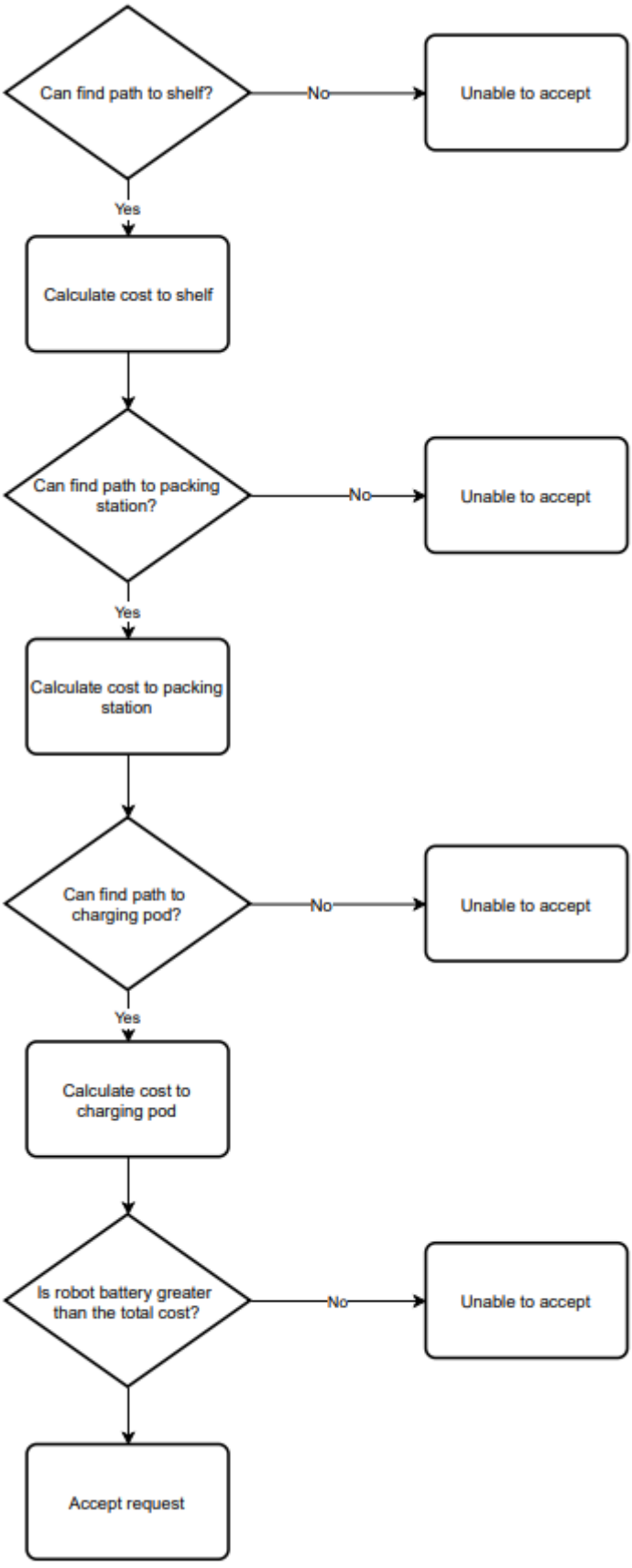
Following the results of this analysis, it is worth taking into consideration that we have not investigated how the layout of the floors affects the results, only the actual configuration numbers. In addition to this, we would have also like to have seen the improvements offered by other algorithms, primarily the cooperative path finding which would have seen the robots consider each other's paths (Resolving the need for deadlock checking).

Appendix

Appendix A)

Pre-provided configurations:
bottomStations.sim
bottomStationsOldMac.sim
bottomStationsWin.sim
oneOfEverything.sim
threeOfEverything.sim
twoRobots.sim
twoRobotsTwoShelves.sim
twoShelves-bad.sim
twoShelves-v2.sim

Appendix B)


Can find path to shelf?
No
Unable to accept
Yes
Calculate cost to shelf
Can find path to packing station?
No
Unable to accept
Yes
Calculate cost to packing station
Can find path to charging pod?
No
Unable to accept
Yes
Calculate cost to charging pod
Is robot battery greater than the total cost?
No
Unable to accept
Accept request