Intelligent Factory Simulation

6068COMP Advanced Data Structures and Algorithms

Weighting: 35%

Joshua Collins (726681)

Liverpool John Moores University



Contents

[Figures 2](#_Toc511069243)

[Abstract 3](#_Toc511069244)

[1. Report on store room traversal 4](#_Toc511069245)

[1.1 State space enumeration 4](#_Toc511069246)

[1.2 Value function calculation 5](#_Toc511069247)

[1.3 Search algorithm discussion and choice with Big Oh Notation 7](#_Toc511069248)

[2. Report on factory robot agent design and AgentSheets simulation 12](#_Toc511069249)

[2.1 Introduction 12](#_Toc511069250)

[2.2 Knowledge Acquisition 12](#_Toc511069251)

[2.3 Reasoning Process 14](#_Toc511069252)

[2.4 Rules Implementation/AgentSheets Execution 21](#_Toc511069253)

[2.5 Conclusion 28](#_Toc511069254)

[References 29](#_Toc511069255)

# Figures

[Figure 1 Factory floor grid with four possible goal paths 4](#_Toc511140383)

[Figure 2 Tree representation of figure 1, showing four possible routes 5](#_Toc511140384)

[Figure 3 Maze traversal for factory floor 7](#_Toc511140385)

[Figure 4 A\* search example using the eight puzzle [1] 8](#_Toc511140386)

[Figure 5 A\* algorithm [11] 9](#_Toc511140387)

[Figure 6 BFS algorithm in pseudocode [5] 10](#_Toc511140388)

[Figure 7 Comparison summary of examined algorithms [3] [4] 10](#_Toc511140389)

[Figure 8 Robot agent 12](#_Toc511140390)

[Figure 9 Charger agent 13](#_Toc511140391)

[Figure 10 Import and Export vans agents 13](#_Toc511140392)

[Figure 11 Storage agents for start and end 13](#_Toc511140393)

[Figure 12 Machinery agents for cutter, welders and riveter 14](#_Toc511140394)

[Figure 13 if functions checking the robot energy levels before starting 15](#_Toc511140395)

[Figure 14 if function that manages interaction between the robot and cutter 16](#_Toc511140396)

[Figure 15 Even chance of welding a prestige skeleton car body 16](#_Toc511140397)

[Figure 16 Even chance of welding a standard skeleton car body 17](#_Toc511140398)

[Figure 17 Welder 2 can only weld standard skeleton car bodies 17](#_Toc511140399)

[Figure 18 Welder 1 simulation cycle behaviour 17](#_Toc511140400)

[Figure 19 Welder 2 simulation cycle behaviour 17](#_Toc511140401)

[Figure 20 Riveting rules that differentiate between car skeleton body qualities 18](#_Toc511140402)

[Figure 21 if function that manages interaction between the robot and store room 19](#_Toc511140403)

[Figure 22 Export van agent behaviour 21](#_Toc511140404)

[Figure 23 Starting position of factory worksheet 21](#_Toc511140405)

[Figure 24 Robot travelling laden with metal sheet 22](#_Toc511140406)

[Figure 25 Robot travelling laden with cut metal sheet 23](#_Toc511140407)

[Figure 26 Welder 1 producing a prestige skeleton car body 24](#_Toc511140408)

[Figure 27 Welder 1 producing a standard skeleton car body 24](#_Toc511140409)

[Figure 28 Welder 2 producing a standard skeleton car body 25](#_Toc511140410)

[Figure 29 Robot travelling laden with riveted standard car body 26](#_Toc511140411)

[Figure 30 import van delivering metal sheets to storestart 26](#_Toc511140412)

[Figure 31 Export van delivering completed cars to the facility 27](#_Toc511140413)

[Figure 32 Alert message for energy levels 27](#_Toc511140414)

[Figure 33 Robot alert depiction 27](#_Toc511140415)

[Figure 34 Robot currently charging 28](#_Toc511140416)

# Abstract

This technical report focuses on using algorithms and travels to ensure a factory robot functions efficiently and correctly on a factory floor. This section will analyse several algorithms, focusing on the advantages and disadvantages of implementing each algorithm into this factory scenario. Moreover, this report documents a factory robot simulation created in AgentSheets to follow specific requirements set by the client.

# 1. Report on store room traversal

## 1.1 State space enumeration

On the factory floor, sheets can be placed randomly in a storeroom, which is equipped with floor sensors. This factory floor can be presented as a grid shown in figure 1, where the robot is required to reach the goal (H10) from the starting position (A1) with the minimum number of moves possible where the robot can only move up, down, left or right. After analysing this grid searching for possible routes the robot can take with the minimum amount of steps. There is numerous possible solutions that all cost the minimum number of steps. The minimum number of steps the robot can take from A1 to H10 is 20 steps, where there are 63 possible solutions to do so. There are seven possible solutions between A1 and J6, and nine possible solutions between J6 and H10 that all cost the same number of steps.

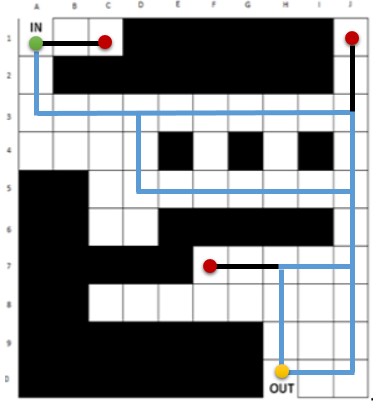


Figure 1 Factory floor grid with four possible goal paths

The possible solution shown in figure 1 can be presented in a tree, which includes other possible routes where different directions are taken at cells D3 and J7 but have the same steps taken to reach the goal. This can be seen in figure 2.

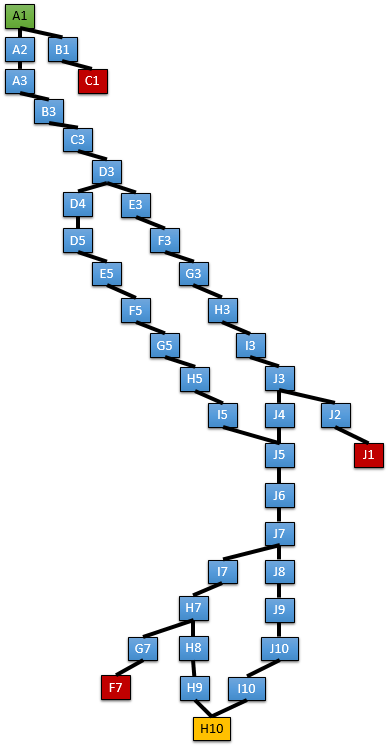
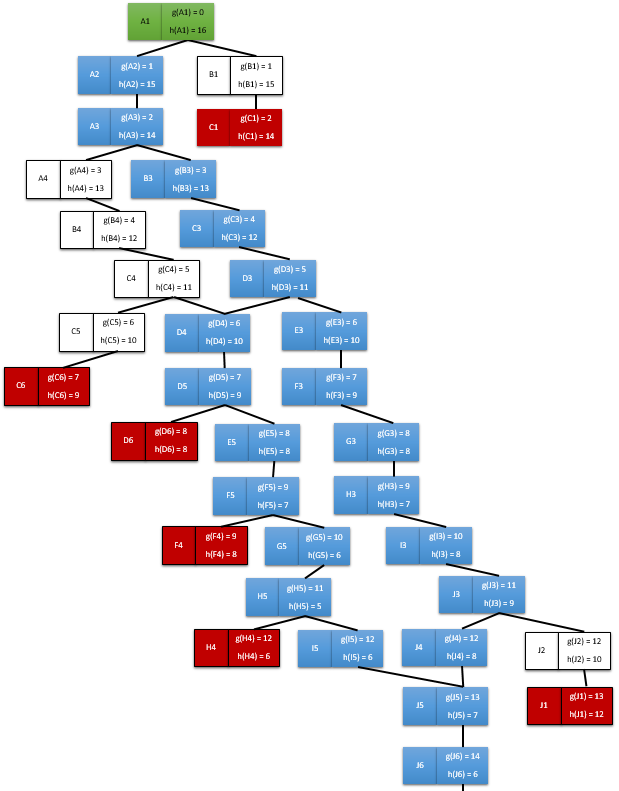


Figure 2 Tree representation of figure 1, showing four possible routes

## 1.2 Value function calculation

Using the grid displayed in figure 1, the Manhattan distance can be calculated to calculate the distance between the starting point to the end point in a grid-like path [2]. The Manhattan distance can be presented in a tree, a more detailed version than figure 2. The key benefit of using Manhattan distance is different locations in the grid, or factory can be compared upon their corresponding Manhattan distance, the distance from the goal. The figure below illustrates a more detailed tree of figure 2, where g() represents the exact cost of the path from the starting point to vertex n and h() represents the heuristic, Manhattan distance. Other trees can be created to illustrate other routes available in the factory, however there are many goal routes applicable in this factory therefore, this tree focuses on the routes previously identified.



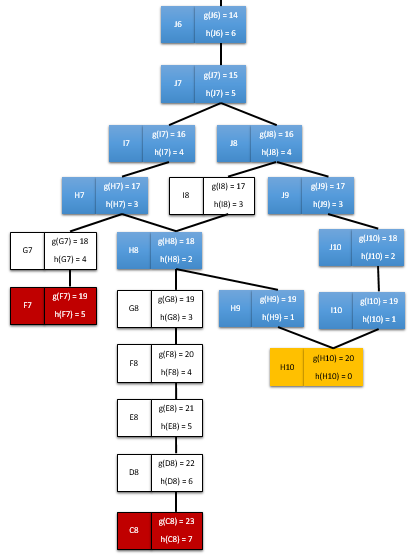


Figure 3 Maze traversal for factory floor

## 1.3 Search algorithm discussion and choice with Big Oh Notation

The purpose of algorithms used in maze traversals is to provide accurate searching whilst using the minimum cost to achieve the goal. Selecting the right algorithm to the right situation can be crucial in improving efficiency whilst maintain accuracy in a system. Various different search algorithms used for different situations and depends on what the goal and requirements of the problem to be solved. Some search algorithms used in industry include Best-First-Search, Depth-First-Search, Breadth-First-Search, A\* Search, Tabu Search and more.

The Best-First-Search algorithm, also known as BestFS uses an evaluation function that focuses on deciding which object is the most promising and then examines it [8]. A key strength of BestFS is the search space is evaluated according to the function set. What makes BestFS so unique compared to other search algorithms is it uses a Uniform Cost Search (UCS) where [4]. A major weakness of the Best-First-Search is it does not guarantee to find a shortest path, but does run faster than other algorithms. As this weakness could result in a major impact on the algorithm used on the factory, Best First Search would not be suitable for this situation.

Additionally, the A\* algorithm could be more suitable to the maze traversal in comparison to Best-First-Search. The A\* search algorithm can be used to find the shortest path in a maze, it operates similarly to the Breadth-First-Search algorithm, but a key difference is it uses a heuristic function to evaluate a search space [4] [8]. A good use for the A\* Algorithm is solving the eight puzzle, a simple but complicated game where you have 8 of 9 possible tiles and must reposition the jumbled up tiles in numerical order, an example is shown in figure 4. The key strengths of the A\* search is it uses open and closed lists to identify search limits and notes that have previously been visited, also it continually re-evaluates cost function as it re-visits nodes [4].

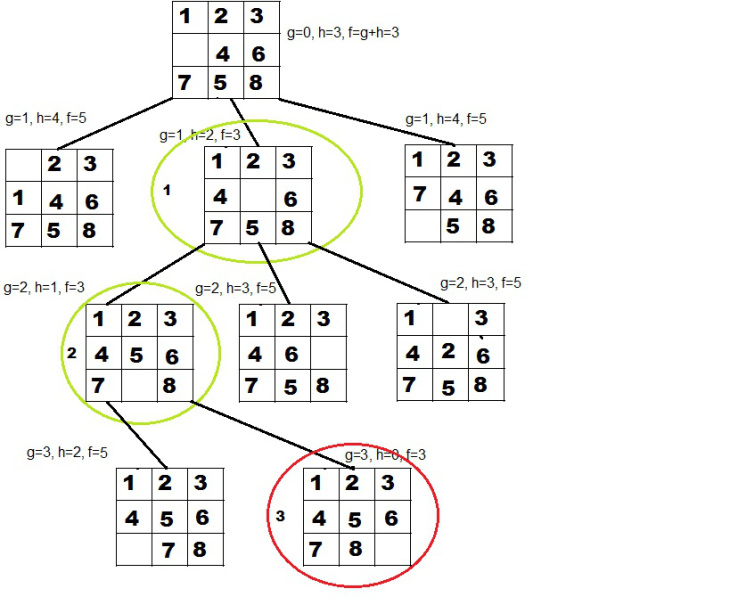


Figure 4 A\* search example using the eight puzzle [1]

Similar search algorithms Breadth-First-Search and Depth-First-Search can be used in the factory example previously described. However, although the logic behind each of the algorithms is similar in some ways, Depth-First-Search is not complete unless the state space is finite; therefore the algorithm is not optimal resulting in a large time complexity to execute. Breadth-First-Search focuses on searching the breadth before examining the depth of the data. A key strength of implementing Breadth-First-Search is unlike Depth-First-Search, it is complete. In addition, this algorithm can be optimal if all operates used have the same cost. The time complexity for Breadth-First-Search is [3]. To conclude, as Breadth-First-Search focuses on searching cells next in order of the distance from the start, then moving to the next row where this continues until the goal is found.

Figure 5 below shows an example of the A\* algorithm that may be used for solving a problem with 8-way directional movement, potentially an algorithm to solve the eight puzzle in figure 4. This algorithm looks rather complex in comparison to the simplicity of BFS and DFS with more functions, loops and ifs used throughout. Figure 6 shows an example of the Breadth-First-Search algorithm that would be used to search a tree. This algorithm looks much less complex in comparison to the A\* algorithm, with only one if statement being used. Breadth-First Search potentially a slower time complexity than A\* as this time to compute depends on the size of the tree that is being searched. If the tree is considerably large, as the algorithm is rather simple compared to the A\*, it would take longer to open and close each single node in a tree where BFS does not examine the node immediately too. Although A\* algorithm is more complex, it is more precise where notes are examined as they are searched so essentially nodes do not have to be re-visited.



Figure 5 A\* algorithm [11]

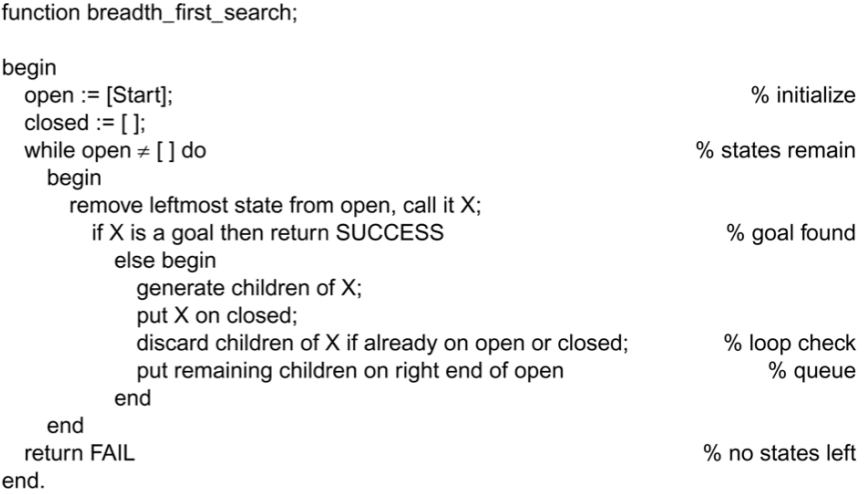


Figure 6 BFS algorithm in pseudocode [5]

Choosing the right algorithm for a search is very important for this factory robot because the robot must be able to operate efficiently reducing energy consumed when operating but also ensuring the search is complete when finished. In this case, the time complexity of the selected algorithm will result in the operating time of the factory robot to move and store the metal sheets in the storeroom efficiently. The figure below shows the algorithms described in this section comparing the time complexity, completeness and if the algorithm is optimal. Looking at the key facts in this table, BestFS and DFS are ruled out of being used, as both are not optimal and DFS is not complete. Therefore, this results in the two most suitable search algorithms to choose from are A\* and BFS. An advantage of selecting A\* over BFS is it uses a heuristic function, where Manhattan distance can be calculated. However, a strong advantage of BFS is the time complexity is less than A\*, and has no derivative. For this scenario of the factory robot, the A\* search algorithm is most suitable as it enables the Manhattan distance to be calculated.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm | Time | Space | Optimal | Complete | Derivative |
| A\* |  |  | Yes | Yes | BestFS |
| BestFS |  |  | No | Yes | BFS/UCS |
| DFS |  |  | No | No |  |
| BFS |  |  | Yes | Yes |  |

b = branching factor, d = tree depth of the solution, m = tree depth

Figure 7 Comparison summary of examined algorithms [3] [4]

Big Oh Notation is an important measure for algorithms and is regularly used to measure the time complexity. This is beneficial to gain further understanding of the complexity of an algorithm and to understand how the algorithm will scale depending on the complexity and input size. For the factory robot example in figure 1, the time complexity of the algorithms would be relatively small, on a larger scale for example a warehouse, or geographic location time complexity of an algorithm is crucial to maximise to potential time efficiency in finding the goal. The most suitable algorithm for factory storeroom robot is A\*, key factors that decide this are:

* Optimal and complete, whereas BestFS and DFS are not
* Accurate time complexity of whereas the time complexity of BFS depends on the size of the branching factory
* Derivative of BestFS, where BestFS is a derivative of BFS

Should the client wish to expand the storeroom in the future resulting in a larger grid for the factory robot to manage, A\* should still be the chosen algorithm in this situation compared to other algorithms.

# 2. Report on factory robot agent design and AgentSheets simulation

## 2.1 Introduction

AgentSheets is a very useful piece of software that allows you to create games and simulations to be published. AgentSheets can be used to portray the functionality in a working environment such as a factory. This section of the report will document the AgentSheets simulation developed that portrays the functionality of a factory robot. More specifically, this factory robot follows a path where protocol must be followed throughout each completion of the circuit. The robot is required to collect a metal sheet from the storeroom, where this is then taken to the cutter where required shapes cut out of the sheet. The robot then takes the cut sheets to either of the two welders, which weld the cut sheets together to form a skeleton car body. The robot then takes the skeleton car body to a riveter, where the panels are riveted together forming a complete car body. The car body is finally carried to the storeroom where these can then be loaded onto vans to be delivered to another facility for further processing.

In addition, the two welders in this circuit have slight differences. Welder 1 and welder 2 can both produce standard skeleton car bodies at the cost of 4 cycles however; welder 1 is capable of producing prestige skeleton car bodies at the cost of 7 cycles. Furthermore, the robots can run out of charge during the cycle. When the robot is travelling form one station to another whilst laden, this costs 2 energy. When the robot is travelling un-laden from station to another, this costs 1 energy. Therefore, the robot can run out of energy whilst travelling the circuit, but checks are performed before starting the circuit to prevent this happening.

## 2.2 Knowledge Acquisition

This section will look closely at the agents that have been used in this factory simulation created in AgentSheets. Each agent can be evaluated across different platforms by understanding the following criteria: performance measure, environment, actuators and sensors.

The robot agent in this simulation uses different depiction that the robot can change to at any given point depending on the position and situation of the robot along the circuit. The robots performance is measured on the complete car bodies produced at the end of the circuit after visiting each tool along the circuit. The environment the robot operates in is following a set path with a wall on either sides, these acts as a set track for the robot with barriers on either side to prevent the robot from steering off. In addition, other environmental features include machinery along the circuit such as cutters, welders, riveters, storerooms and a charging point. Actuators on the robot agent include arms to allow the metal sheets after being cut, welded and riveted to be carried from one station to the next. Sensors of the robot agent include a standard sensor to detect what object is in each position of the robot, in all directions. These sensors allow the robot to know where to move, turn and stop along the circuit. Furthermore, the sensors essentially detect the environment surround the robot so the robot can change depiction to identify what form of car body the robot is carrying.



Figure 8 Robot agent

Some form of energy, in which the robot must be recharged every so often, powers the robot. The AgentSheets simulation contains one charging point for the robot where this re-charges at five units per simulation cycle. The performance measure of the charger is whether it recharges to robot to its full energy capacity when leaving the charging station and whether the robot only charges when the energy levels are almost ran out. The environment of the charger is stationary on the factory floor alongside the path, with walls either side of the charger. Actuators of the charger will include a port of some form to connect to the robot to begin charging. Sensors of the charger are a movement sensor to detect whether the robot is in front of the charger, where the robot performs checks whether it is required to be charged.

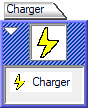


Figure 9 Charger agent

The factory robot simulation uses vans to import metal sheets to the storeroom at the start of the circuit, and export the completed car bodies from the end storeroom to the facility. The performance measure of the vans in this simulation are whether they travel to the facility to either retrieve metal sheets or deliver car bodies when there are either no metal sheets left in storage, or three completed car bodies in storage. The environment of the vans is a road which the storage and facility can be located at opposite ends, pedestrians, customers and other traffic. The actuators of the vans include; steering wheel, accelerator, brake, wheels, indicators, horn and more. Sensors of the vans vary on the make and model such as, engine sensors, odometer, and GPS to identify whether the vans have reached their destination.

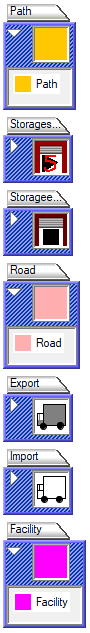
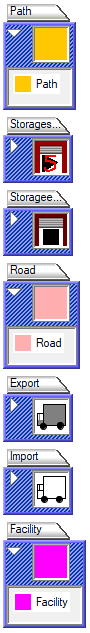


Figure 10 Import and Export vans agents

This robot factory simulation contains two storerooms that serve similar purposes, in a real life factory this would be the same room but is easier to represent the functions by keeping these separate. The performance measures of the storerooms are whether Storagestart provides the robot with metal sheets to begin the process, and whether Storageend keeps track of the complete car bodies that the robot delivers. The environment of both storage rooms is the same, in the factory and at the start/end of the circuit to improve the vehicle manufacturing process. The actuators on the storage rooms keep count on the number of metal sheets there are remaining in stock for Storagestart, and for Storageend counting the number of cars delivered by the robot and understanding which vehicle is either prestige of standard.

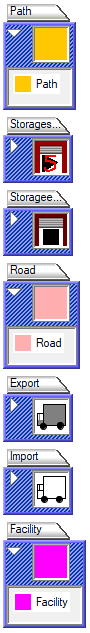
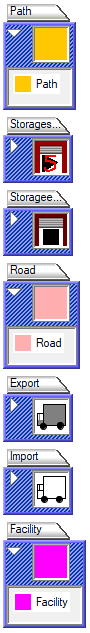


Figure 11 Storage agents for start and end

In addition, the factory robot simulation requires machinery to allow the whole process to happen. Machinery included along the circuit include a cutter, two welders and a riveter. The performance measures of this machinery are very simple, whether the cutter cuts out shapes from metal sheets the robot delivers. Welder one and two’s performance is measured on whether they can weld cut metal sheets into standard skeleton car bodies, welder one is also assessed whether it can also weld cut metal sheets into prestige skeleton car bodies. The performance measure of the riveter is whether it can weld either prestige or standard skeleton car bodies into complete car bodies. The environment for this machinery are the same, stationary and placed in order across the factory floor, alongside the path. Actuators of the machinery may include arms and hands if the machines are automated, or humans to operate on if the machinery is manual. There are no sensors at this machinery as the robot is intelligent and able to detect which machinery it is stopping at.

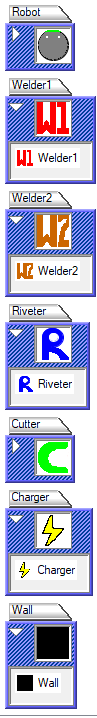
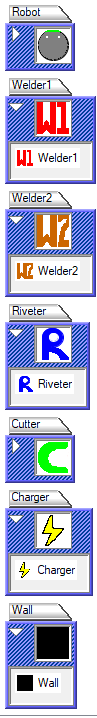
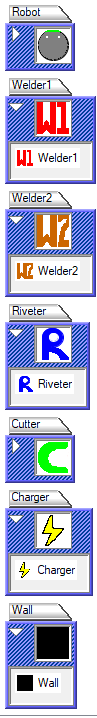
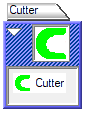


Figure 12 Machinery agents for cutter, welders and riveter

## 2.3 Reasoning Process

To produce an effective factory robot simulation for the given scenario, there are numerous rules that must be implemented to ensure each agent meets all core functionality requirements to make the system successful and perform to the standards required. On the course of the circuit the robot follows, energy is subtracted from the robot after each station. When the robot is travelling laden (carrying a metal sheet that is either plain, cut, welded or riveted), after reaching each station like this, 2 energy is subtracted. Therefore, to complete the factory circuit the robot must have a minimum of nine energy remaining to prevent the robot running out of power whilst live on the circuit.

A simple check is implemented into this simulation just as the robot begins the circuit. Where the robot’s position can see the path directly in front, Storestart to the left, wall to the right and its current state is un-laden, check the current energy level of the robot. If the robots current energy level is greater than or equal to nine, the robot has enough energy to go around the circuit without dying; therefore, the robot may pick up the metal sheet and begin the process. However, if the robot’s energy is less than nine, then the robot is unable to continue and must be re-charged, the state of the robot is changed to alert and a message is sent to the user informing them that there is not enough energy to complete the circuit.

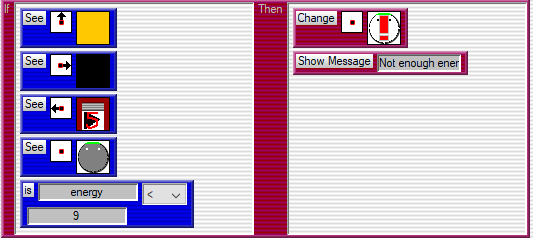
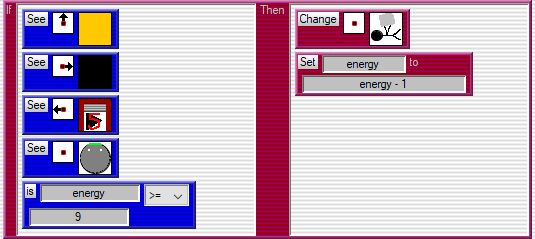


Figure 13 if functions checking the robot energy levels before starting

Functions are implemented across the circuit that the robot follows, often the rules are very similar that will check if the path is in front of the robot and checks that the wall runs along the path to continue moving. Some exceptions are encountered along the track where the wall is not on both sides of the circuit, but a tool can be found. For example, figure 14 illustrates a cutting tool should be below the robot, where the robot’s current state is holding an uncut metal sheet. Also, as according the factory requirements, when the robot is travelling laden this will cost the robot 2 energy. A rule implanted into this only allows the metal sheet to be cut if the robot has sufficient energy. In addition, if the robot has met the positional and energy requirements, the robot must stop shortly and change to the robot depiction that is now carrying a cut metal sheet. After this rule, another rule is added that will check whether the robot is carrying the metal sheet but in the current position which will begin to move the robot along the circuit.

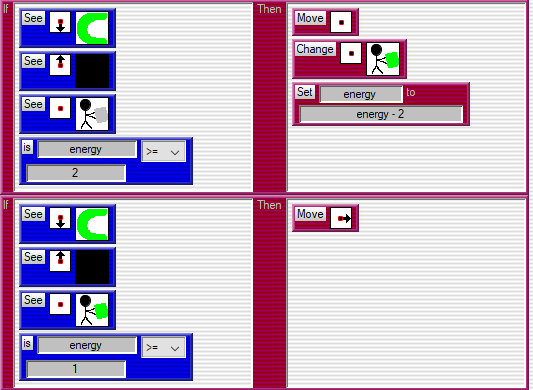


Figure 14 if function that manages interaction between the robot and cutter

Similar to figure 14, the robot is required to carry the required shapes cut by the cutter and transport to one of two welders in the factory. Similar positional situations have been implemented into the figures below that can identify a unique position of the robot for when the robot meets either welder 1 or welder 2. Welder 1 is required to operate differently to welder 2, as welder 1 can weld cut metal sheets into either prestige or standard skeleton car bodies. This is managed by adding a 50:50 chance of producing either quality skeleton car body. Like figure 14 above, using this machine costs the robot 2 energy to perform the transaction. The rule here checks if the robot has enough energy to carry out the action, if so then reduce the energy levels once the robot depiction has changed to carrying either prestige or standard. Additionally, the figure below this illustrates an identical rule that manages the welder two operations, where the difference here is welder 2 can only produce standard skeleton car bodies. In addition, the welders are required to take specific simulation cycles when welding the skeleton car bodies from cut metal sheets. When welder 1 is welding cut metal sheets, this takes seven simulation cycles but welder 2 takes four simulation cycles. This can be seen in figure 18 and 19.

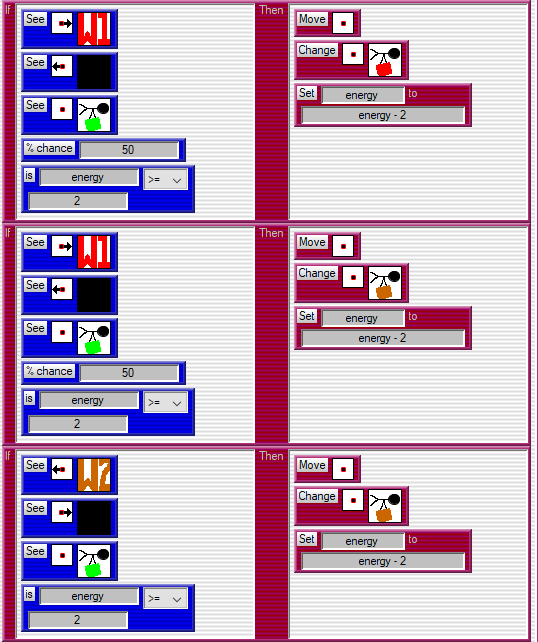


Figure 15 Even chance of welding a prestige skeleton car body

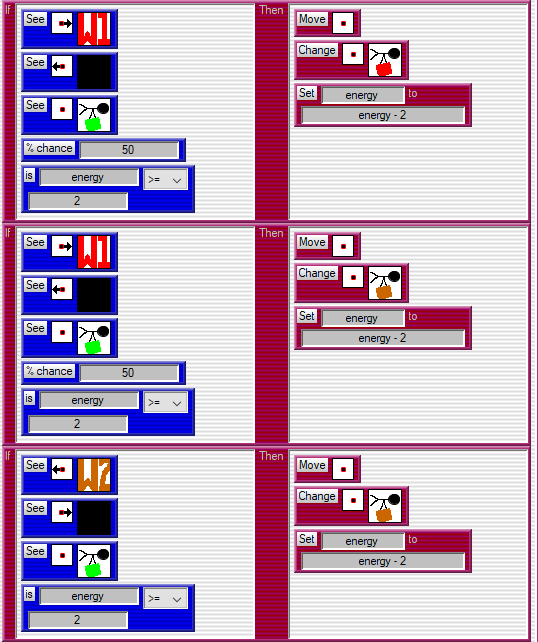


Figure 16 Even chance of welding a standard skeleton car body

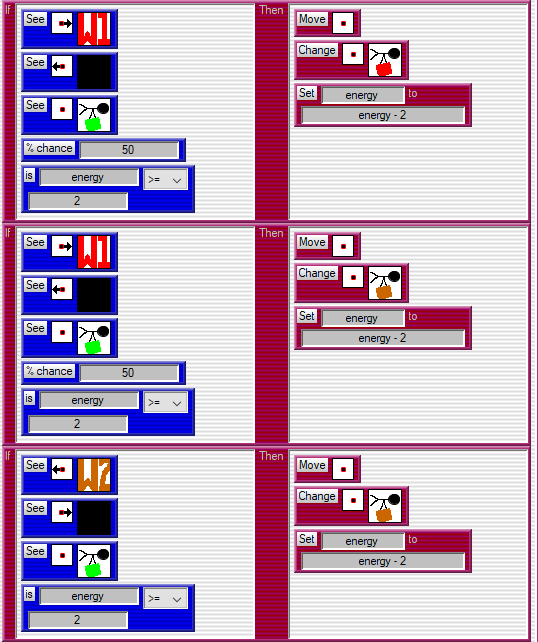


Figure 17 Welder 2 can only weld standard skeleton car bodies

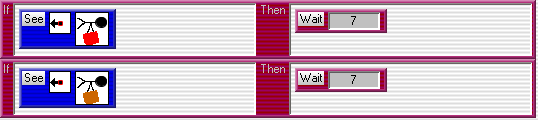


Figure 18 Welder 1 simulation cycle behaviour



Figure 19 Welder 2 simulation cycle behaviour

Furthermore, the skeleton car bodies must be riveted after stopping at welder 1 and welder 2. The riveting machine operates identically to the cutting machine described above. A key focus point here is the riveter must differentiate between when the robot is travelling laden with either a prestige or standard skeleton car body. The riveter contains two different rules to manage this to ensure the depiction changes accordingly so there is no confusion on the factory floor. Like with every machine the robot visits when travelling laden, this costs the robot 2 energy but also is required to check whether the robot has sufficient energy to perform the action first. Another two rules are added after the two below to then send the robot on its way to proceed with the factory circuit.

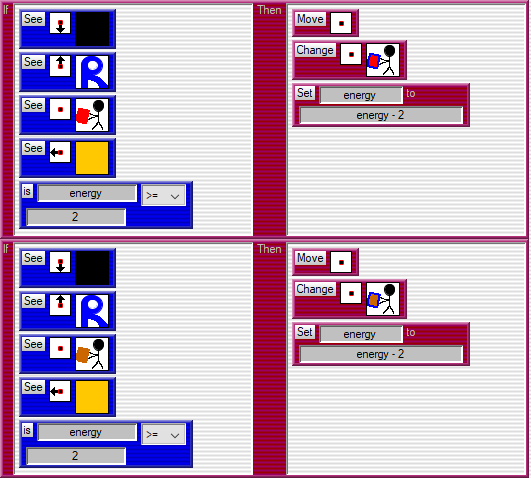


Figure 20 Riveting rules that differentiate between car skeleton body qualities

In addition, on completing the circuit the robot must deliver the riveted car body to the storeroom. This also costs the robot 2 energy as the robot is travelling laden up to this point. Unique positional identifies to find what the robot can see in several directions are used to find this. Once the robot has visited the storeroom, the robot depiction changes to show the robot is no longer travelling laden. Also, similar rules are implemented here to that allow the store room to also change depiction that counts the number of car bodies that have been delivered, as the robot on arrival may be carrying a prestige or standard skeleton car body, the store room is able to see this. The store room contains several rules to calculate this, which when changing depiction, a delivery van is able to identify when there are 3 cars stored and take them away to the facility. When the van has exported the complete cars, this then returns, and the storeroom depiction is reset but continues to count all cars produced.

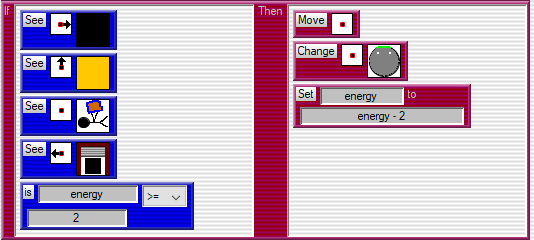


Figure 21 if function that manages interaction between the robot and store room

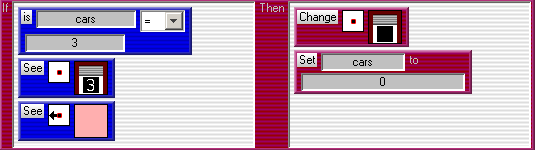
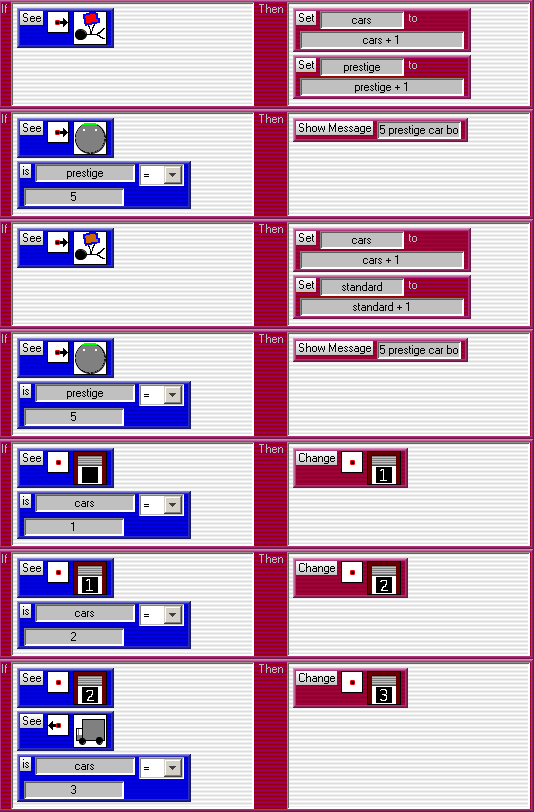


Figure 22 Export van agent behaviour

## 2.4 Rules Implementation/AgentSheets Execution

The robot factory simulation produced uses many different rules to manage all situations faced around the factory circuit and execute tasks correctly to meet the requirements set. Figure 23 below shows the worksheet for this simulation in the starting positions. Here it is evident that following the path, the robot must follow a sequence along the factory: collect a metal sheet, go to the cutter, then either welder, riveter then drop the car off at the store. With the stores in this figure, the first store visited holds the metal sheets where a stock level is five before the import van must deliver more. The final store has a capacity of three vehicles before the export van must deliver these to the facility.

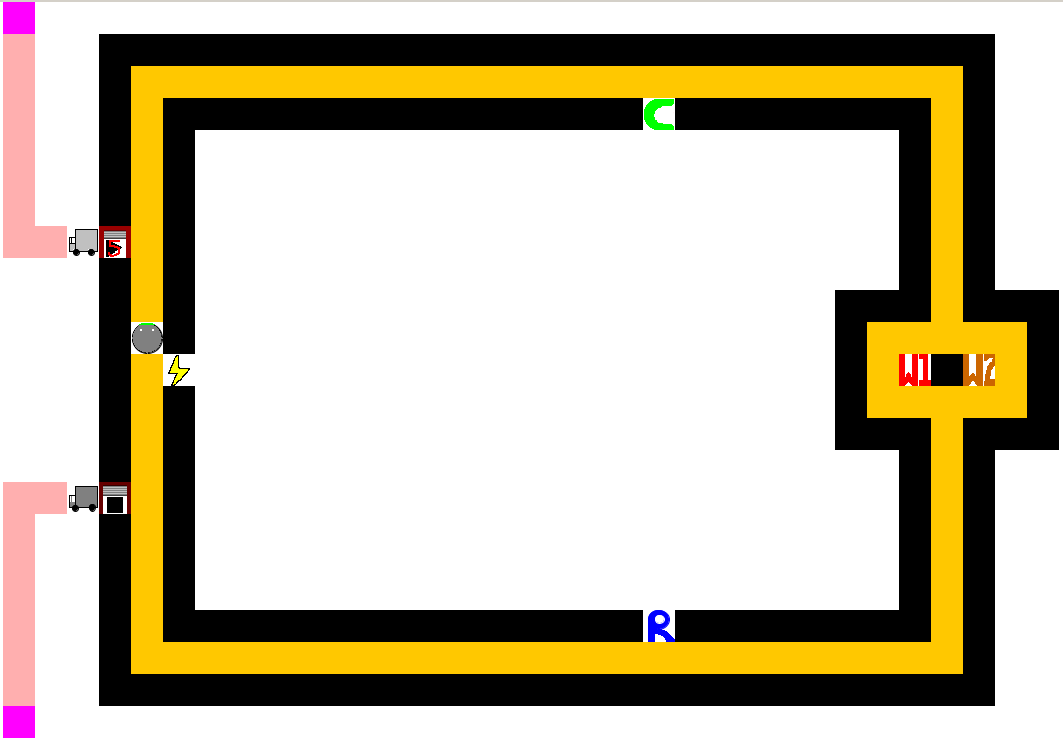


Figure 23 Starting position of factory worksheet

Once the robot has collect a metal sheet from the storestart agent. The robot’s depiction changes to illustrate the status of the robot, which direction it is facing and what object it is carrying. Figure 24 below shows this as the robot is making its way to the cutter. In addition, the storestart depiction has updated that now signifies there are four metal sheets remaining before a delivery is required.

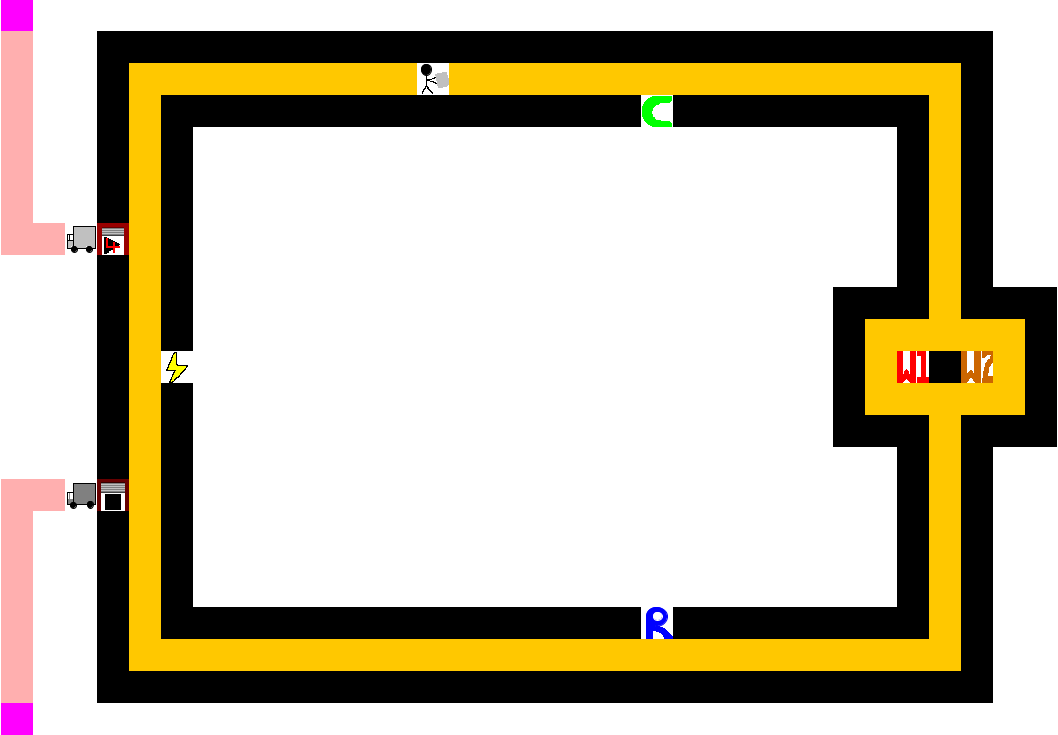


Figure 24 Robot travelling laden with metal sheet

The next stage of the simulation is the metal sheets are delivered to cutters who press the required shapes out of the sheet metal. Figure 25 below shows the robot on route to delivering the cut metal sheet to one of the two available welders. Notice also that the depiction has updated to identify the robot is carrying the cut metal sheet, where colour co-ordination is used to identify what the robot is carrying.

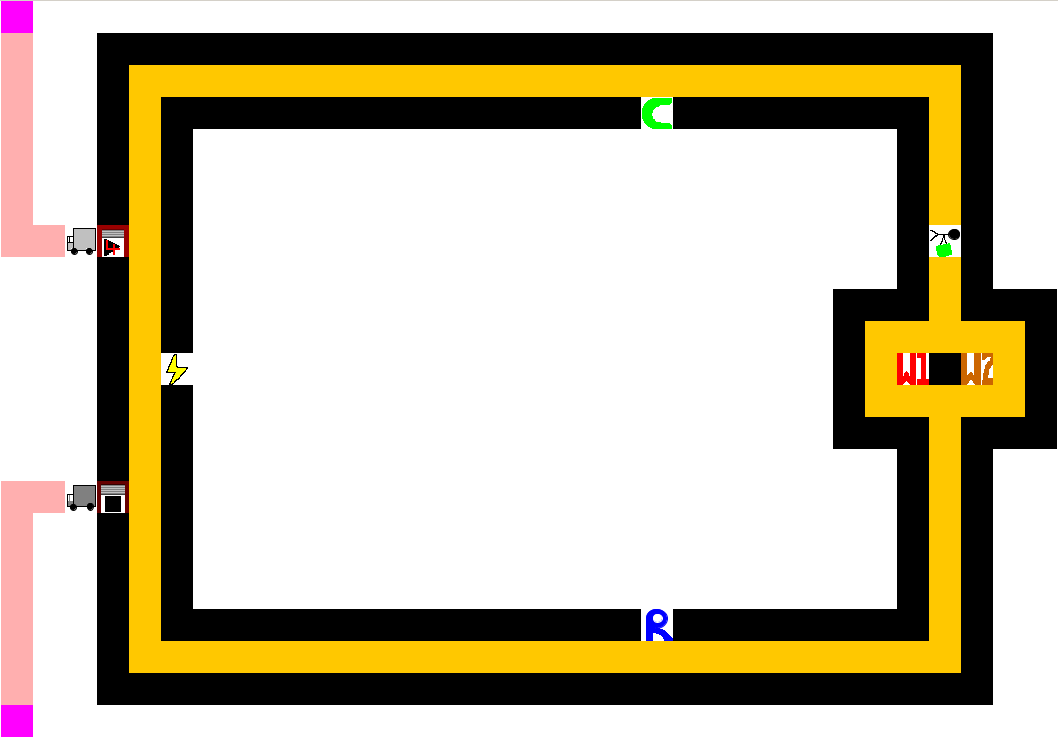


Figure 25 Robot travelling laden with cut metal sheet

Moving on from above, the robot is required to visit either welder 1 or welder 2 with a 50:50 chance of turning left or right at the junction. This function is described in 2.3, allowing an even chance with no priority implemented of which welder to visit. The next three figures, 24-26 illustrate the robot visiting welder 1 and welder 2 during the simulation. Colour co-ordination is used between the welder, quality and depiction of the robot. Where brown signifies welder 2 and a standard skeleton car body as only this quality can be produced and welder 2. Red signifies welder 1 and a prestige skeleton car body. However, welder 1 is able to produce both prestige and standard qualities therefore; figures 26 and 27 two figures illustrate the robot carrying either of these qualities after visiting welder 1.

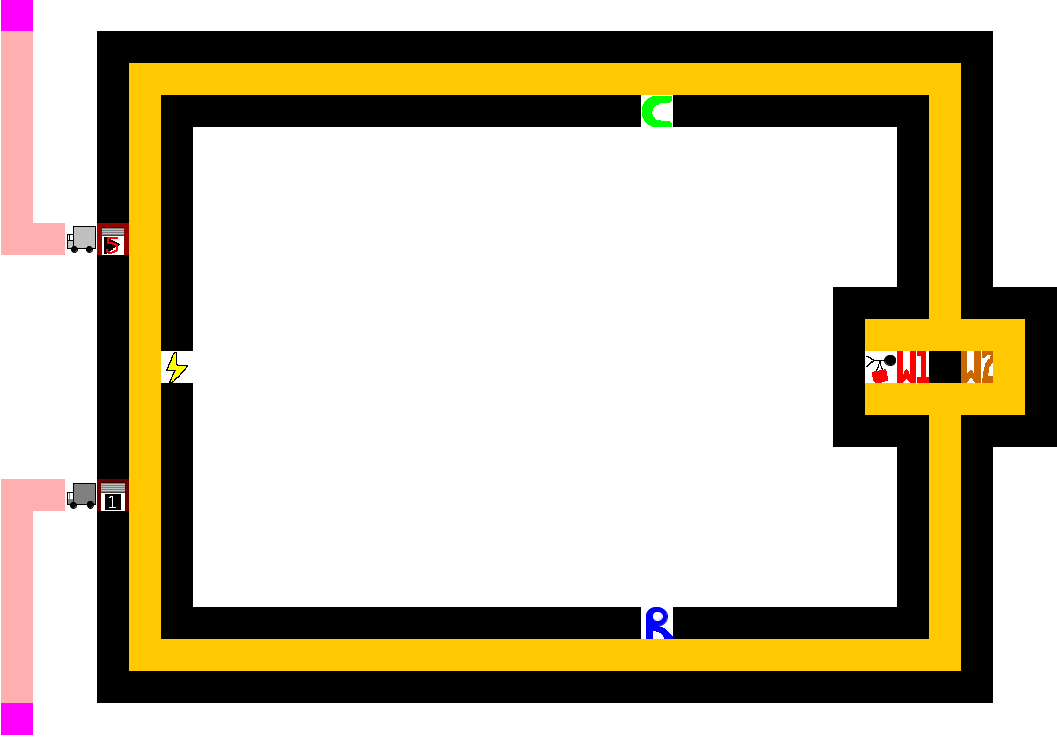


Figure 26 Welder 1 producing a prestige skeleton car body

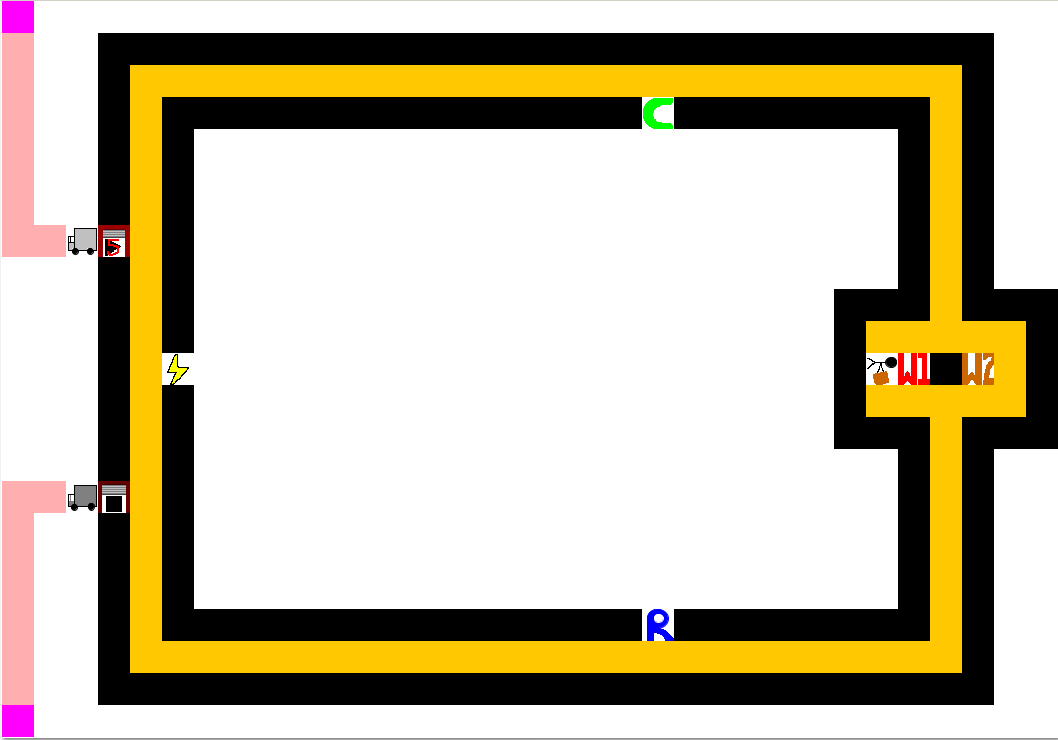


Figure 27 Welder 1 producing a standard skeleton car body

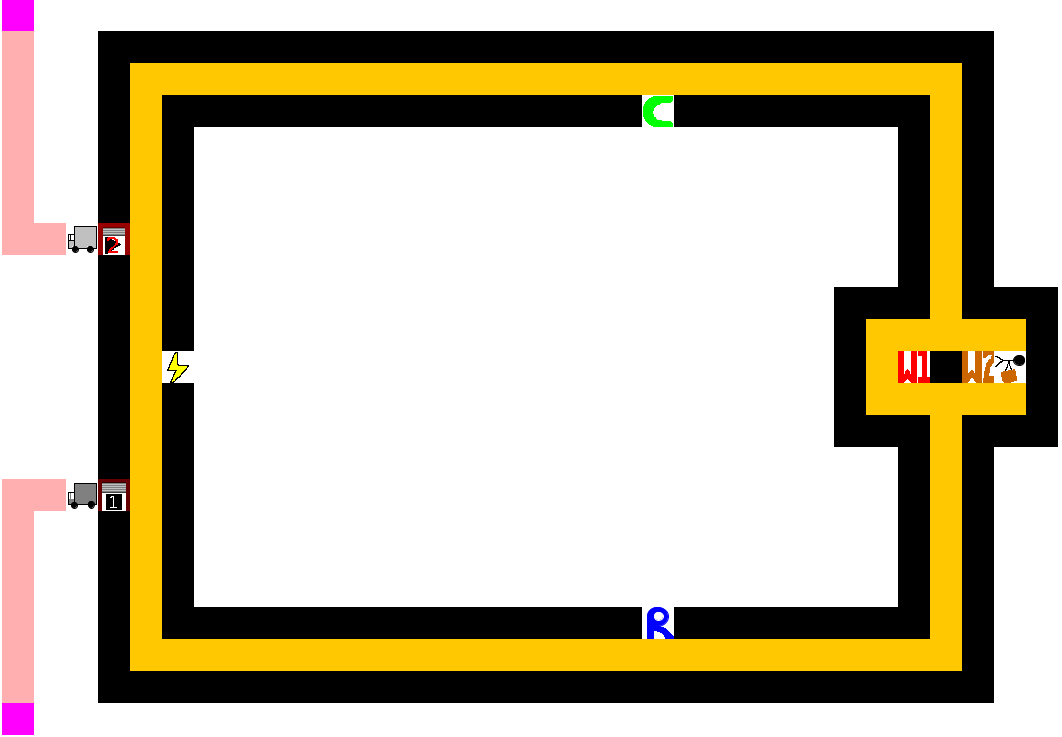


Figure 28 Welder 2 producing a standard skeleton car body

After visiting either welder available on the factory floor, the robot delivers the skeleton car body to the riveter, where the panels are riveted together to complete the car manufacturing process. Like with the other figures displayed above, the depiction changes to identify what car body the robot is carrying, using the same coordination described above after visiting the welders. Figure 29 shows the robot is travelling with a standard car body, and about to deliver this to storeend.

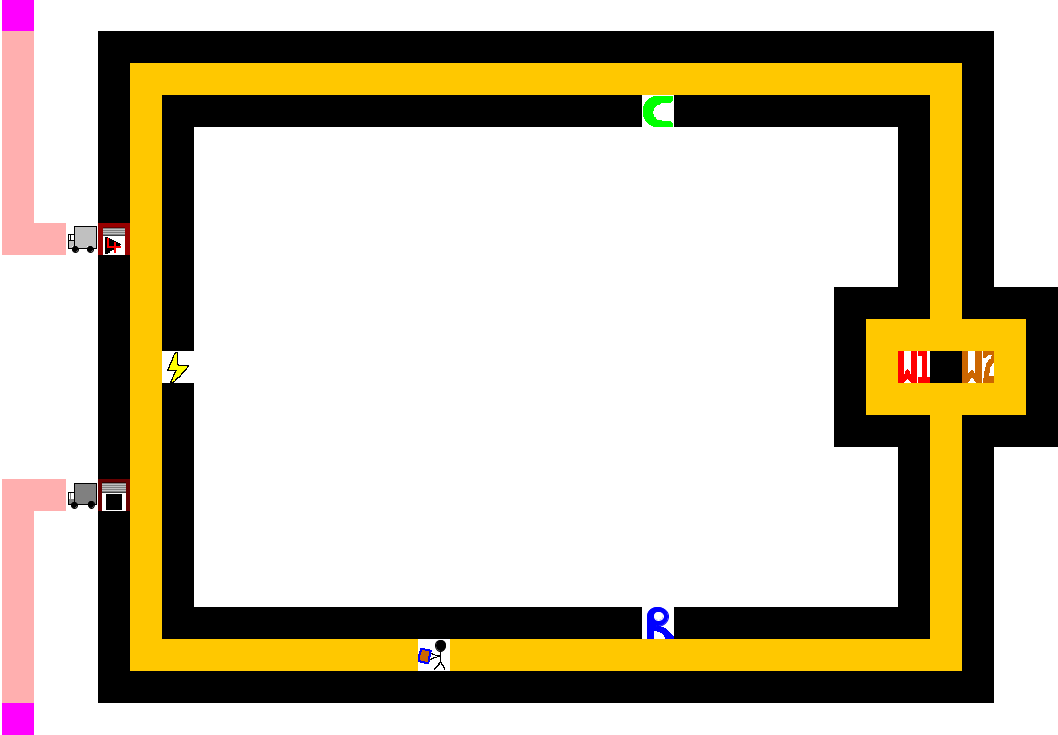


Figure 29 Robot travelling laden with riveted standard car body

Additional functionality has been implemented into the factory simulation that allows imports and exports to be made to either of the storerooms. There are two different storerooms used in this simulation that serve similar purposes, the storestart begins with holding five metal sheets in stock for the robot to collect when beginning the process. After each visit, the depiction of the storestart changes, decreasing each time. When running out of metal sheets, changing the depiction again signals the import van to visit the facility (Pink Square) and return updating the metal sheets stock to five again.

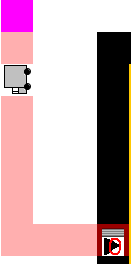


Figure 30 import van delivering metal sheets to storestart

A similar method has been implemented to the storeend, where the number of cars delivered increases when the robot passes at the end of the circuit. The storeend has a capacity of three, when reaching this quantity; the export van departs delivering the completed cars to the facility. On returning, the storeend depiction is reset to zero and begins counting again.

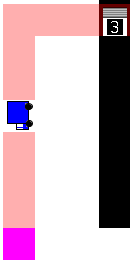


Figure 31 Export van delivering completed cars to the facility

The robot operating on the factory floor uses energy to operate as required. When travelling laden and visiting a checkpoint, this costs two energy. Travelling from the two store rooms un-laden costs one energy. Rules are implemented to perform checks on the robot that only allow the robot to proceed with sufficient energy. Another check is performed at the start of the circuit that detects whether the robot has enough energy to complete the circuit. The minimum energy cost of the circuit is nine. Therefore, if the robot’s energy levels is lower than nine when collecting a metal sheet, an alert is raised to prevent the robot breaking down along the track. Figures 32 and 33 show the alert message and the robot alert depiction changed if this situation arises. Additionally, as the robot energy levels are being reduced when functioning, the robot must re-charge when levels are too low to complete a circuit. When visiting a charging point at low energy levels, the robot depiction updates to the charging status where this recharges at five units per simulation.

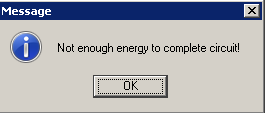


Figure 32 Alert message for energy levels



Figure 33 Robot alert depiction



Figure 34 Robot currently charging

## 2.5 Conclusion

Completing the factory robot simulation required taking a simple approach by breaking all the requirements down in to small steps. Firstly, analysing the requirements from the client to understand what objects are to be used, and deciding what agents and depictions will be used throughout. The requirement analysis provided understanding that the robot required several depictions and there would be four pieces of machinery along the circuit, with a storeroom and charging point.

When implementing the functionality to the robot, it was important to draw the circuit first and aim to get the robot moving along the track accordingly, with depictions changing at corners to signal which way the robot is facing. After this, the machinery points were drawn onto the circuit, which allowed the robot to again change depiction after visiting each station, cutter, welders and riveter. Once the robot is changing accordingly and moving appropriately around the circuit, the energy requirements and simulations cycles could be implemented into the project. This was then followed up with additional functionality to the import and export vans when the state of the storerooms change.

To conclude, a requirements analysis was beneficial to producing the factory robot simulation as the simulation produced is essentially numerous simple methods used to produce the functionality, where the simulation as a whole is rather complex. To summarise, the factory robot simulation produced is a simple complex system.

# References

1. algorithmsinsight. (2018, February). *Implementing A-Star(A\*) to solve N-Puzzle.* Retrieved April 2018, from Wordpress: https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/implementing-a-star-to-solve-n-puzzle/
2. Improved Outcomes Software. (2004). *Manhattan*. Retrieved April 2018, from Improved Outcomes Software: http://www.improvedoutcomes.com/docs/WebSiteDocs/Clustering/Clustering\_Parameters/Manhattan\_Distance\_Metric.htm
3. Kolivand, D. H. (2018, January 11). *Lecture2-4*. Retrieved April 2018, from Canvas: https://canvas.ljmu.ac.uk/courses/13604/files/601579?module\_item\_id=289217
4. Kolivand, D. H. (2018, January 25). *Lecture3-1*. Retrieved April 2018, from Canvas: https://canvas.ljmu.ac.uk/courses/13604/files/612862?module\_item\_id=293733
5. Patel, A. (2018, March 2). *Heuristics*. Retrieved April 2018, from Stanford University: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
6. Patel, A. (2018, March 2). *Introduction to A\**. Retrieved April 2018, from Stanford University: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
7. Pitts, R. I. (2000). *Recursion: Solving a Maze*. Retrieved April 2018, from Boston University: https://www.cs.bu.edu/teaching/alg/maze/
8. University of Washington. (2017). *Graph Traversals for Maze Search*. Retrieved April 2018, from University of Washington: https://courses.cs.washington.edu/courses/cse326/09sp/projects/proj2/t2.html
9. Wikipedia. (2018, January 15). *Maze Solving Algorithm*. Retrieved April 2018, from Wikipedia: https://en.wikipedia.org/wiki/Maze\_solving\_algorithm
10. Imms, D. (2012, June 3). A\* pathfinding algorithm. Retrieved April 2018, from Growing with the Web: http://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html