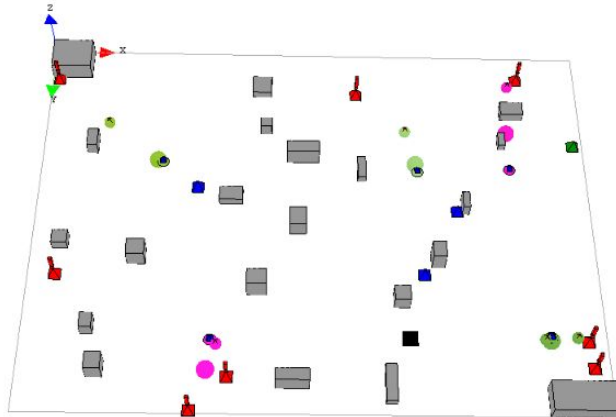


ID2209 – Distributed Artificial Intelligence and Intelligent Agents

Final project

Multi agent system for resource delegation between agents and stations in a dynamic 2D environment.



Course Coordinator:
Mihhail Matskin

Course Assistant:
Cosar Ghandeharioon
Shatha Jaradat

Group 14
Óttar Guðmundsson
Xin Ren
2017-12-23

Abstract

This project focuses on simulating resource delivery between agents in a dynamic 2D environment. Agents communicate with each other via FIPA protocol to meet at agreed coordinates that reduces total distance travelled for every agent in selected group. The method used for finding the coordinate is inverse factor point based on the agent's speed.

Keywords

Multi Agent System, FIPA, Resources, Camps, Supplies, Requesters, Delivery man

Introduction

In an area where demand for resources needs to be answered quickly on random occasions can be defined as high critical/high risk area. An idea of such an area can be refugee camps for people who have had to evacuate their city due to natural disasters. These camps might be scattered around so when a request for resources, like water or food would be made, someone from the camps would need to travel to the nearest supply station or a delivery truck would need to transport them over to each camp. The truck could map out the camps that needed resources before taking off and travel the shortest distance (traveling salesman problem) but that means it would have to choose which camps to go to first. Since supplies are vital the risk of letting some camps wait longer might lead to starvation of those resources which is an unwanted situation.

We could shorten the time of delivery if the requesters from the requesting camps would meet the truck midway instead of the delivery truck travelling to each location individually. But how would we decide on the best meet up coordinate when the requesters are different individuals? And what happens if a request is made after the delivery truck has left the supply center?

Our simulation solution addresses this problem by creating a multi-agent system for such resource delegation in a dynamic environment. In the simulation there are camps where resources are consumed on a different rate and also supply stations where stocks of resources are kept. When a camp is running out of resources, a requester from a camp contacts a supply station for resources. The project will mainly focus on cooperation and collaboration between agents to minimize the total delivery time of resources.

This project is a final project in the course Distributed Artificial Intelligence and Intelligent Agents (ID2209) at KTH. The main goal of this project is to try out methods studied in the course to incorporate them in a multi agent system in order to deliver a basic model that simulates these behaviours.

Platform used - GAMA 1.6:

GAMA is a Java based modeling language to develop simulation for agent environment. It's user friendly and high level language allows complex behaviours and interactions in a few lines of code. More information about the platform can be found [here](#).

Note: To run the file, please follow the instructions in the readme.txt file. The presentation accompanied this report can also be found under the name slides.pdf.

Release planning

After viewing the documentation for Gama to understand the syntax and the interface, a small release plan was created to keep track of what was needed to be done. This served as a base guide throughout the project.

Release 1.0	<i>In this very first release, the system supports multiple camps, one requester per camp, one supply station and multiple deliverymen.</i>
Camp behavior	<ol style="list-style-type: none">1. Created during system initiation at random location2. Displayed as sphere in red.
Supply station behavior	<ol style="list-style-type: none">1. Created during system initiation at random location.2. Displayed as pyramid in purple.3. Keep track of the current location and target location of all the deliverymen4. Listening to messages from requesters, upon receiving one message from a requester, it picks up a deliveryman whose target location is closest to the requester and replies with the deliveryman info.
Requester behavior	<ol style="list-style-type: none">1. Created during system initiation at host camp location2. Start sending request to supply station at random time.3. Listening to messages from supply station, upon receiving reply with deliveryman info, it sends a request to the deliveryman.4. Listening to messages from deliverymen, upon receiving a meeting point, move towards the point. After picking up the resources, move back to camp.
Deliveryman behavior	<ol style="list-style-type: none">1. Created during system initiation at random supply station location2. Listening to messages from requesters, upon receiving a request, calculate the best meeting point for all the requesters and send it to all the requesters.3. After delivering all the resources, move back to supply station.

Release 1.1	<i>Based on release v1.0, in this release, the system supports multiple supply stations, for which a supply control center agent is introduced.</i>
Camp behavior change	None
Supply control center behavior	<ol style="list-style-type: none"> 1. Created during system initiation at random location 2. Keep track of the location of all the supply stations. 3. Listening to messages from requesters, upon receiving a request from a requester, it forwards the request to the closest supply station to the requester.
Supply station behavior change	<ol style="list-style-type: none"> 1. Listening to messages from control center, upon receiving a request from the control center, it picks up a deliveryman whose target location is closest to the requester and sends deliveryman info to the requester. If All the deliverymen are further away from the requester than the station, let the requester come to the station and pick up.
Requester behavior change	<ol style="list-style-type: none"> 1. Start sending request to supply control center at random time 2. Pick up at station
Deliveryman behavior change	<ol style="list-style-type: none"> 1. Shared by all stations. When delivery done, go back to the closest station

Release 2.0	<i>Based on previous releases, in this release, the system introduces quantity of resources. We also introduce loading control in supply stations.</i>
Supply control center behavior change	None
Camp behavior change	<ol style="list-style-type: none"> 1. Created with a random quantity of resources, resource utility speed, resource threshold level. 2. Resource consuming. 3. Update the storage when requester comes back with resources.

Supply station behavior change	<ol style="list-style-type: none"> 1. When selecting the best deliveryman for new request, consider the remaining storage of the deliverymen. Select the one that is not loading at any station, whose target location is closest to the requester and has enough storage to satisfy the lower bound of the requester. If no deliveryman has enough storage, or the one selected is further from the requester than the station is from the requester, ask the requester come to station to pick up. 2. If a deliveryman is selected, make reservation in the deliveryman for the resources required by the requester, otherwise, make the reservation in the station. 3. Only one deliveryman can load resources at one time. Other deliverymen have to wait.
	<p><i>(Interiors)</i></p> <ol style="list-style-type: none"> 1. 3 staff inside the station to pick up resources and load the deliverymen. 2. Stop loading when the deliverymen storage reach the capacity.
Requester behavior change	<ol style="list-style-type: none"> 1. Monitor the storage of the camp, when it is lower than threshold, sends a request to control center with two numbers, initial storage of the camp and the percentage of the storage(lower bound) that he is willing to go out and pick up.
Deliveryman behavior change	<ol style="list-style-type: none"> 1. Delivery the reserved amount to a requester. 2. Start loading when no other deliveryman is loading in the station.

Release v2.1	<i>Based on previous releases, in this release, the system supports different kinds of resources with different priority, environment factors like weather, day or night and so on</i>
Day/night	Add a float variable that ranges from lower to upper bound by some factor. This variable will affect the camps consume rate and change visuals of the model.
Environment	<ol style="list-style-type: none"> 1. Create a horizon so the grid will not be floating mid air. 2. Set background color and make the model look more lively.
Airplanes	<ol style="list-style-type: none"> 1. Create an airplane that flies over the model.

	<ol style="list-style-type: none"> 2. Let it occasionally drop packages to supply stations. 3. Reset it when it has flown out of the screen
Random generated map	<p>Create an option that generates a random samples of buildings</p> <ol style="list-style-type: none"> 1. Set a random number of buildings with random width and height. 2. Mark all grid cells as obstacles
Multiple resources	<p>To support multiple resources, we propose following changes:</p> <ol style="list-style-type: none"> 1. In requester, it sends request as long as in the camp, one of the resources reaches threshold level and requests only the resources that reaches threshold level. 2. In supply station, deliveryman selection is based on utility for the requester, which is $\text{sum}(\text{reserved_amount} * \text{resource_prio}) - \text{distance} * \text{factor}$. The deliveryman with highest utility is selected unless the utility of the station is even higher. 3. In supply station, loading deliveryman selection is also based on utility which is $\text{sum}(\text{storage} * \text{resource_prio})$. The one with highest utility is selected from the loading queue. 4. In the interiors of the supply station, different resources are stored at different locations. Staff go to different locations to pick up and go to exit to load. 5. Draw the resources in the insideBuilding species and let the staff actually go to the ones that are needed. 6. In the deliveryman, when it goes back to a station, it puts itself into the loading queue.

The work

Finding best algorithm

The first step in our project was to find the best meeting point for agents to minimize the distance travelled for everyone. This sounded easy at first, but since agents had different kind of speed this needed a little revision. We tried out 4 different methods as demonstrated below.

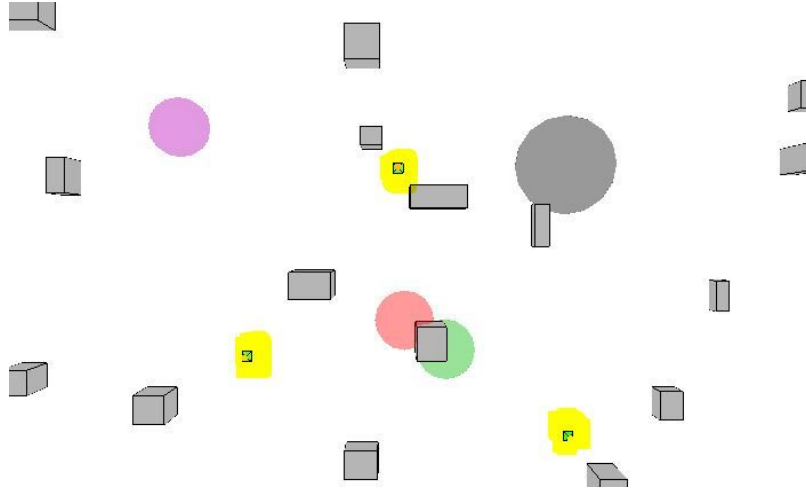


Image 1: All algorithms tested

Green - Average point

To start off somewhere, the average point for the agents was found by simply summing the coordinates and divide them by their numbers. This gave us a starting point but did not consider the agent's speed.

$$\bar{X} = \frac{\sum_{i=1}^n x_i}{n} , \quad \bar{Y} = \frac{\sum_{i=1}^n y_i}{n}$$

Purple - Normalized point

For each agent that was supposed to meet at a delivery point, the max and minimum speed of all agents was found. Their speed was then normalized and then multiplied with their coordinates before dividing by the total amount of agents. Thus, the coordinates of agents with the lowest speed had full value but higher speed had none. This did not work well, since the coordinates of the fastest agents were never considered.

$$X_n = \frac{(X - X_{min})}{(X_{max} - X_{min})}$$

Red - inverse factor weighting

To give the fastest agents some values, the speed was used to create an inverse weighted factor. This was done by summing the speed values of all agents, which was then divided by each speed to find out the speed factor. To find the weight of each coordinates was then found by dividing the total speed with the agent speed, which was then divided by the factor. This proved to give a good result, which can be seen in the result.

$$Z_P = \frac{\sum_{i=1}^n \left(\frac{z_i}{d_i} \right)}{\sum_{i=1}^n \left(\frac{1}{d_i} \right)}$$

Agent 1 at {1,1} speed 1	$(6/1) / 11 = 0.5454...$	* {1,1} -> {0.54 ; 0.54}
Agent 2 at {8,3} speed 2	$(6/2) / 11 = 0.2727...$	* {8,3} -> {2.16 ; 0.81}
Agent 3 at {5,8} speed 3	$(6/3) / 11 = 0.1818...$	* {5,8} -> {0.90 ; 1.44}
Total speed = 1 + 2 + 3 = 6	{0.54 ; 0.54} + {2.16 ; 0.81} + {0.90 ; 1.44}	
Total factors = 6/1 + 6/2 + 6/3 = 11	Meeting point is {3.60 ; 2.79}	



Image 2: Usage of the formula.

Gray - calculating total time for all agents

A bounding box that included all agents was found and the first point in that box was set as the selected meeting point. After doing so, the distance to that point for all agents was calculated and summed together. This was done for all points in the box and if a point with lower time was found, that point was selected as the best point. The results for this method was not successful since it was really computational heavy and the slow agents had way too much value.

Agents

Control Center

The control center is portrait as a black diamond. Each model only contains one control center that receives requests from Requesters and forwards the supply request to the Supply station that is closest to the Requester.

Supply stations

Supply stations are displayed as blue pyramids. When a request is received from control center, a station selects the best deliveryman based on utility and make reservation for the original requester, then sends the deliveryman info to the requester. If itself has higher utility than the deliveryman, it sends its info to the requester and let the requester come and do self pick-up. When there is deliverymen in the loading queue, it selects the deliveryman with the highest utility to load.

Interiors of supply stations

Inside supply stations, there are staff that displayed as pyramid with a random color and separated locations for each resource displayed as sphere with different colors. When a deliveryman is selected to be loaded, the interior will first collect info from the deliveryman about for each resource, the quantity to load. Then the staff will start loading with following manners: if there is already enough staff going for a resource, he/she will go for next resource; if the amount left to load for a resource is lower than his/her capacity for this resource, he/she will just carry the amount left to load.

Camps

A camp is a green pyramid agent that is initialized with all types of resources. For each resource, it will have a different consumption rate and storage for that resource. When the storage value will be lower than the threshold, it will signal that it is out of stock and turn red. It will not consume more of that resource until it will receive a refill from its Requester.

Requester

A requestor starts in a camp and waits until one or more of it's resource will be emptied. When it notices that it needs more resources, it will send a request to the control center, then go and pick up the resources it needs. They can travel to meet a delivery man or go to a supply station by themselves, depending on what the closest supply station tells him to do. A requester has the same color as the deliveryman he/she is going to meet or the supply station in self pick-up case.

Delivery Man

When a deliveryman is selected by a station to deliver resources to a requester, the station updates the deliveryman the reservation info. When a deliveryman receives a request from a requester, he/she updates the meeting points with the new requester's location and sends the new meeting points to all the requesters going to meet he/she. When he/she meets a requester in the meeting point, he/she delivers the amount that reserved by the requester. When all the reserved resources have been delivered, the deliveryman travels back to the closest supply station and when it reaches the supply station, he/she registers in the loading queue.

Resource

A resource is a object that the model is based around. Each resource has a specific ID and priority and is used by each camp and carried around by requesters and delivery mans. The main storage of the resources are kept at supply stations.

Visuals

Interface

The interface of the model allows several customizations.

Use custom map feature loads a specific object file that has about 20 obstacles at given location. If not used, the loading will take a little longer but will create a random map. True by default.

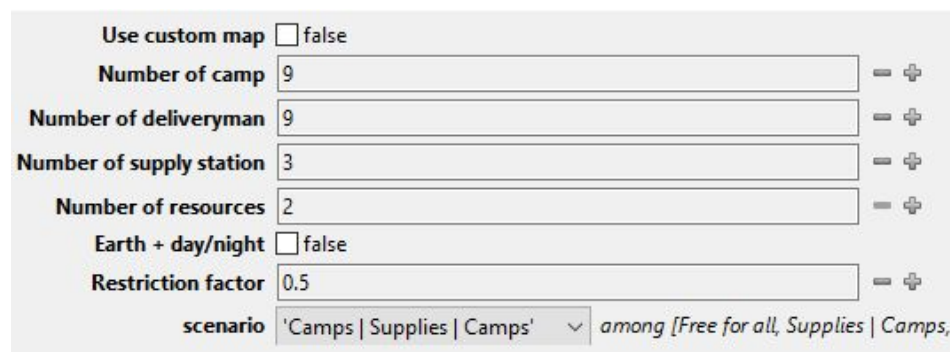
Number of camp sets how many camps will be loaded to the model. 9 by default.

Number of deliveryman sets how many delivery agents are in the model. Their starting location is any supply station in the model. 9 by default.

Number of resources sets how many different types of resources are being used in the model. 2 by default.

Earth + day/night affects the consume rate of the camps as well as providing a little bit more live to the model. False by default, since it slows down the simulation a bit.

Restriction factor / Scenario: See **Restriction factor** below.



The image shows a configuration interface with the following elements:

- Use custom map**: A checkbox labeled "false".
- Number of camp**: A text input field containing the value "9". To its right are minus and plus icons.
- Number of deliveryman**: A text input field containing the value "9". To its right are minus and plus icons.
- Number of supply station**: A text input field containing the value "3". To its right are minus and plus icons.
- Number of resources**: A text input field containing the value "2". To its right are minus and plus icons.
- Earth + day/night**: A checkbox labeled "false".
- Restriction factor**: A text input field containing the value "0.5". To its right are minus and plus icons.
- scenario**: A dropdown menu showing the selected option "'Camps | Supplies | Camps'" and a list of other options: "among [Free for all, Supplies | Camps,".

Image 3: The starting interface when project is loaded

Environment

To make the system a little bit more visual, basic concepts of computer graphics were used such as background coloring, creating day and night system effect.

Day and night

The model does not take into account the location on the planet, so effects like summer and winter solstices are ignored. Instead, the day and night rotation follow a simple sinus function that ranges from -1 to 1 based on time passed. The value for this function is then used on several different places in the model. It controls the position of the sun's orbit which is a simple ellipse drawn in the background and also the ambient light and background color of the model. The biggest effect it has is the consumption rate of the camps, but at night the rate drops down to 0% while reaching 100% at midday.

Restriction area

When the model loads camps and supplies are placed on the grid where there are no obstacles. To make different types of behaviour the placement of supplies and camps can be restricted within certain areas. The latter two were made to create a more real world scenario. The former one, where the supplies are in the middle, a warehouse is delivering to other stores that are located around a town. The other one is a metaphor for a football game, where the fans might need beverages or food while watching the game.

Note that camps are never within 40 range of supplies.

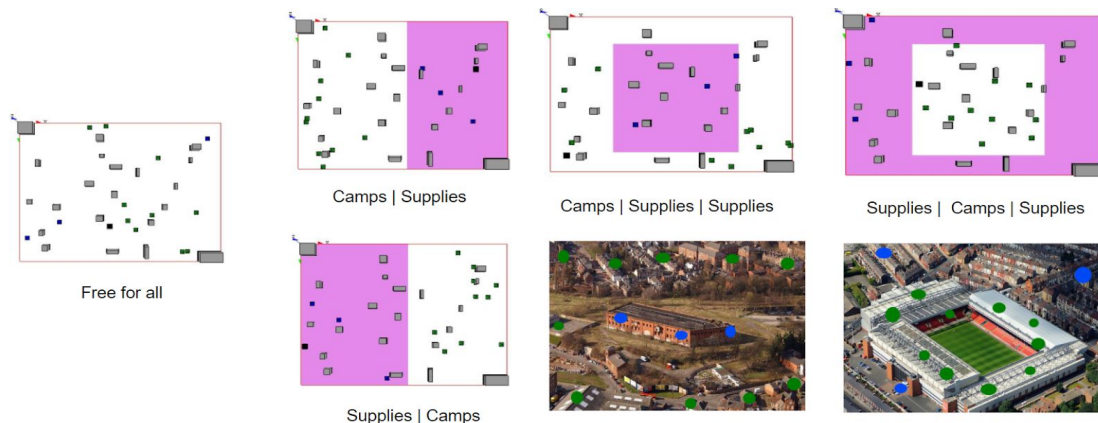


Image 4: Different types of scenarios

Result

To see if our methods were working correctly, a small static demo was created to measure time of resource returning to the camps. This demo had three camps and one supplier, this three requesters and one delivery man with these initial values

Agent	Location	Speed	Request time
Requester 1	{50, 20, 0}	1	2
Requester 2	{30, 70, 0}	2	5
Requester 3	{66, 140, 0}	3	8
Delivery man 1	{170, 70, 0}	2	

Table 1: Initial values for testing out method

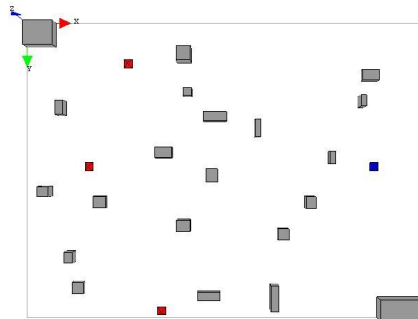
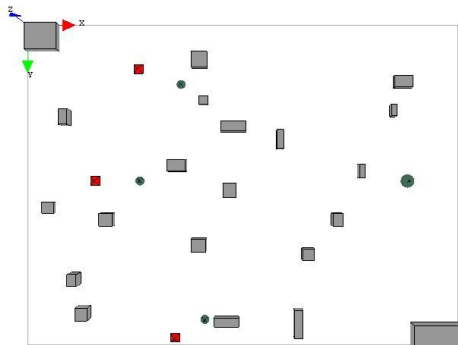


Image 5: Graphical environment from table 1

To test out if we are making any improvements we compare three different scenarios. These scenarios are when no method is being used, when finding just the average point and finally using inverse factor to determine the meeting point. Let's look at the results from those three different type of methods.

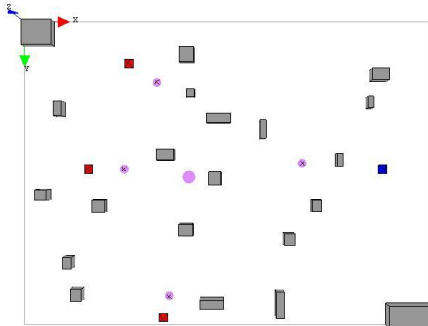
No method



No method	Start Cycle	Distance Travelled	End Cycle	Cycles delta
Requester 1	2	262,95	135	133
Requester 2	5	275,51	103	98
Requester 3	8	250,31	72	64
		788,77		295

Image 6: Total distance travelled when there is no delivery man

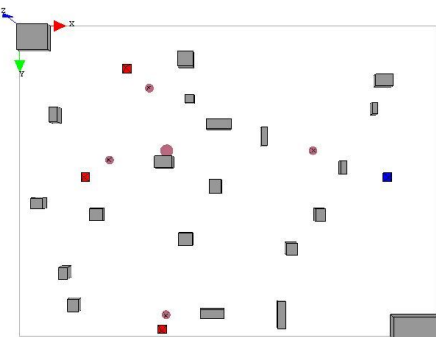
Average point



Average				
	Start Cycle	Distance Travelled	End Cycle	Cycles delta
Requester 1	2	126,28	81	79
Requester 2	5	92,79	44	39
Requester 3	8	132,07	46	38
DeliveryMan	-	98,48	43	
		449,62		156

Image 7: Total distance travelled when an average point is used

Inverse weighted factor



Inverse factor				
	Start Cycle	Distance Travelled	End Cycle	Cycles delta
Requester 1	2	86,93	58	56
Requester 2	5	76,62	43	38
Requester 3	8	169,58	58	50
DeliveryMan	-	104,16	33	33
		437,29		144

Image 8: Total distance travelled when the inverse weighted point is used

Comparing methods

As seen above, the total distance travelled for all agents drops down from 788.77 to 437.29 when comparing the inverse factor with no method and each camp receives their restock of supplies 62.37% faster. When comparing the average point to the inverse factor, the distance travelled only drop from 449.62 to 437.29 and camps receive their products only 8% faster than. It's still a big improvement, especially when the resources that are being sent are of a critical importance. Do note that the delivery man is also quicker delivering the products and can thus return back to the supply station to refill.

The rest of the model adds up to more behaviour and simulation with advanced features but is really hard to compare with the methods. Thus, after seeing these results, we were confident that the method did reduce distance travelled and time waited for resources. So by making the model more dynamic and adjust it more to the real world (like loading time, requesters going by themselves instead of waiting, interior loading etc) the method will still scale with the complexity.

Further improvements

This project can be extended to an more advanced simulation. There were a few nice to have features which were not implemented due to model complexity, scope and timing of the project. These ideas can be classified into two different categories.

Optimization

Agent walking path (Dijkstra, A*, weighted network)

3D environment and paths

Time taken to deliver to requester

Add priority to inverse weighted point

Supply stations run out of resources

Supply station only allow X many delivery man

Supplies can weigh, so they slow agents down

Simulation enhancement

Special entrances at camps/supplies

Weather effect that slows down everyone

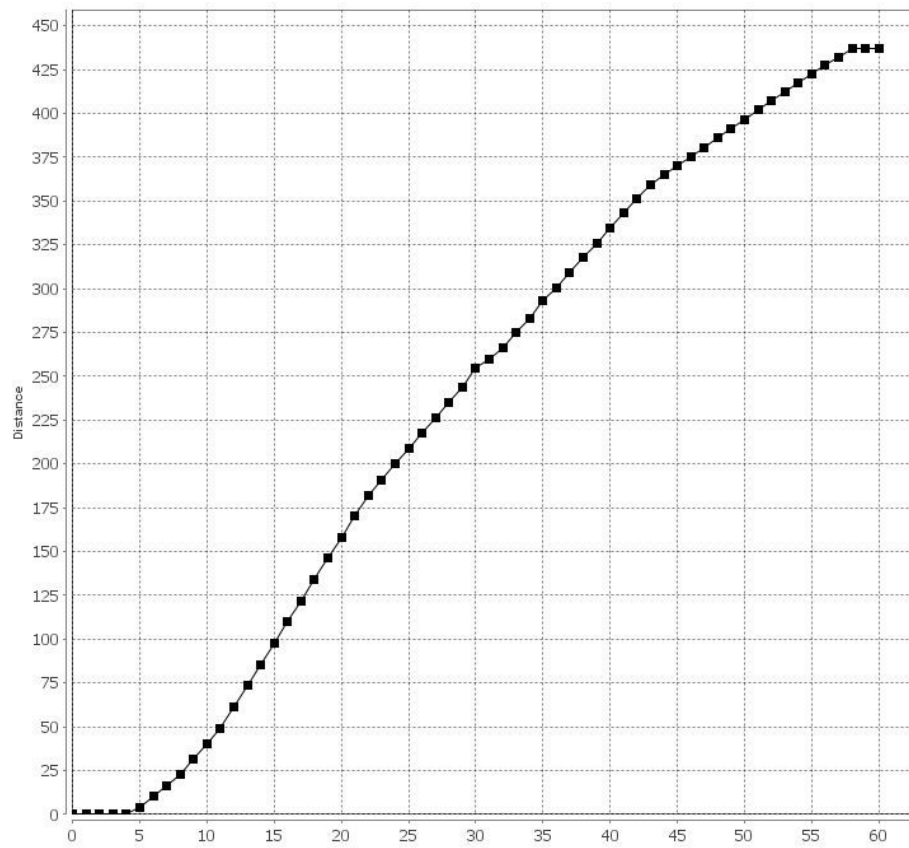
At night, people might get robbed

Conclusions

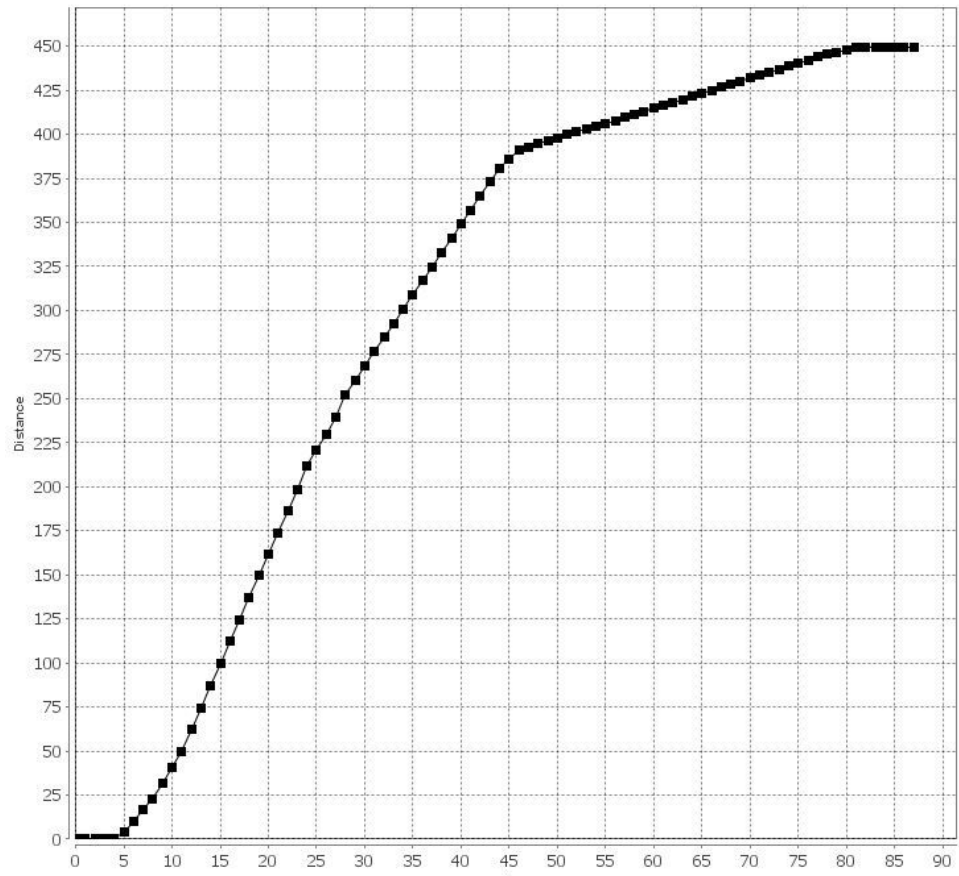
Originally, the project didn't have any clear goals or a theory that we hoped to prove. Since the teacher assistant simply said that we had no specific requirements, only that she was sure that we would deliver something *awesome*, we were excited and motivated to go beyond the "minimal requirements" of passing.

We wanted to create something far more impressive and outstanding than the initial assignment, something that addresses real world problems and is thus more relevant. After working on it for some time, we saw more possibilities to improve our model and realized what the platform was capable of doing. By being Agile and sticking roughly to our plan, we allowed our minds to wander a bit and adjusted new ideas once they came up. We hoped to surpass the basic requirement scope and we are pretty confident we did so. We think that our results and implementation of the model was rather good and demonstrated our skill, creativity and enthusiasm for the course. Hopefully, this project will set an example of what students can create after taking this course and inspire others to try out something on their own by using methods and concepts learned in class.

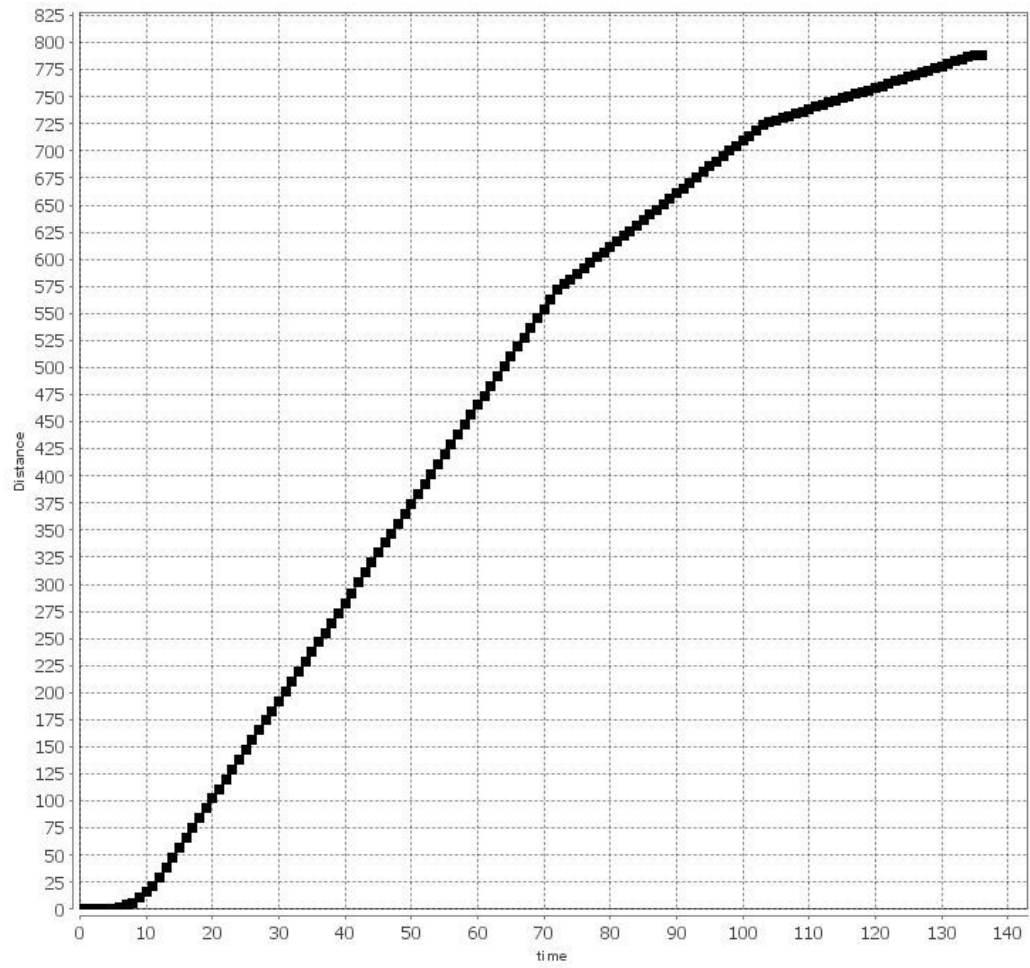
Appendix - Data



Graph 1: Inverse weighted point on total distance travelled at cycle



Graph 2: Average point on total distance travelled at cycle



Graph 3: No method on total distance travelled at cycle