Universidade do Porto
FEUP Faculdade de Engenharia

Lift Management System

Agents and Distributed Artificial Intelligence

4th year 1st semester
2020/2021

Problem & Agent Description

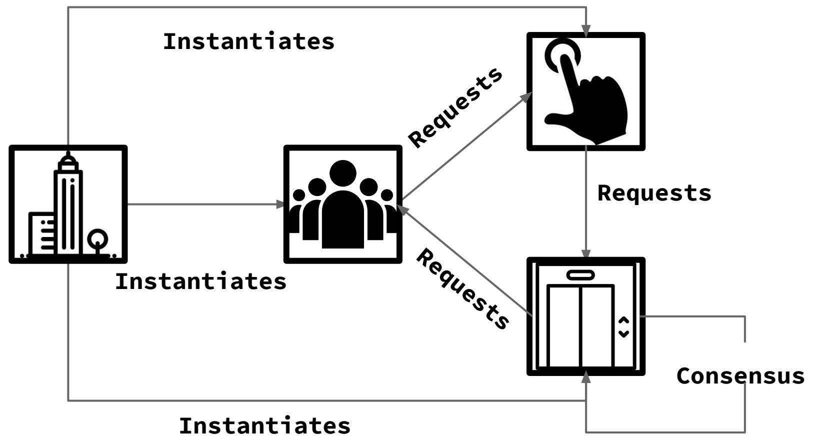
Managing the **Lift System** in a big skyscraper is a fairly complex task.

With many requests being made in a relatively short amount of time, optimizing elevator allocation for each call is essential to ensure everyone gets to their destination in a reasonable amount of time.

The objective of this project is to develop a **Multi-Agent** System for the management of a set of elevators in a building.

Building Agent - Skyscraper. Initiates environment.
Lift Agent - Building's lifts.
Floor Panel Agent - Floor's lift request button.
Request Agent - Simulates people in the building.

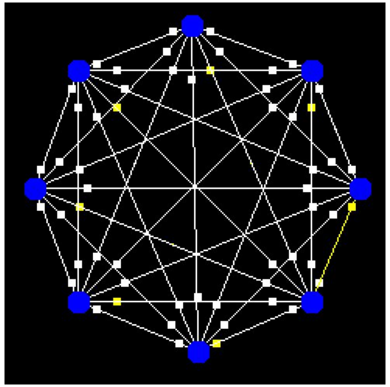
Building Agent   Lift Agent   Floor Panel Agent   Request Agent



Strategies & Decision Making

Consensus algorithm

Implemented a **Bully Election Algorithm Behavior** to decide which Lift will attend the Request.
When a request is received, a Lift will send its estimated time to all other Lift's and receive the other one's.
It will then compare its own proposed time (based on the optimal position for the request on its tasklist) with the rest of them. If it finds one that will get there quicker it will stop the comparison. Otherwise, if it is better than everyone else it will send an **HALT** message at the end. The other Lifts will then receive the **HALT** message and send their agreement to the elected Lift, which will add the request to its **taskList**.



Heuristic

The **optimal position** for a request on a lift's tasklist, used to calculate proposed time, is determined by the direction of movement.
While **going up** the lift attempts to get all "UP" and "End" Requests, and while **going down** all "Down" and "End" Requests, if the floor is on it's path. Additionally, **turning points** are also detected to ensure that, if need be, a lift can keep going in a particular direction to attend to a request, inserting the new task list entry on the turning point, when optimal.

Independent Variables

As it stands, the project has the ability to simulate several distinct configurations of characteristics of buildings, lifts, flow of people, simulation time and lift position update frequency:

Building Characteristics:

Number of Floors: Number of floors in the building.

Number of Lifts: The number of lifts in the building.

Lift Characteristics:

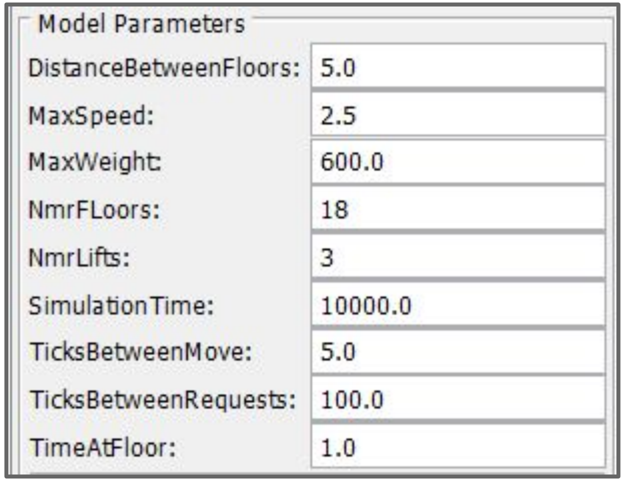
Maximum Weight: The maximum weight a Lift can carry. This can be translated into the occupation of a Lift since we consider a person to weigh 75kg.

MaxSpeed: The speed of the Lift. It is the distance travelled per tick between move.

Ticks Between Move: Number of ticks between a Lift movement.

Ticks Between Request: Number of ticks between each request is sent.

Simulation Time: Number of ticks between each request is sent.



Model Parameters

DistanceBetweenFloors:	5.0
MaxSpeed:	2.5
MaxWeight:	600.0
NmrFLoors:	18
NmrLifts:	3
SimulationTime:	10000.0
TicksBetweenMove:	5.0
TicksBetweenRequests:	100.0
TimeAtFloor:	1.0

Dependent Variables

Lift’s Current Floor: Each Lift’s current floor changes as time passes. This information is important for us to understand their trajectories.

Lift’s Current Weight: Each Lift’s occupation can be calculated through its current weight.

Average Waiting time: This tells us the average time each Lift takes to attend a request. It is important for us to understand if we need more Lifts in the building.

Average Lift’s occupation: This is important to understand which Lifts are being over or under used.

People Entering and Exiting Floor: Tells us how many people are entering or leaving on each Floor. This helps us see which Floors are the most requested.

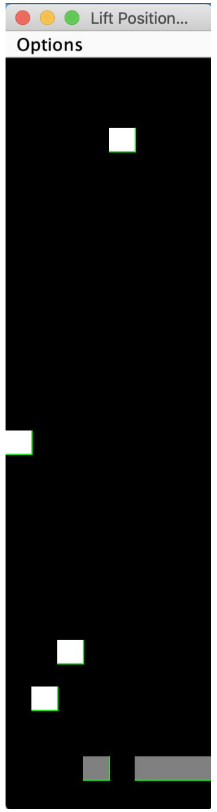
Visualization of the Simulation execution

To be able to visualize how our dependent variables vary over the simulation time, we use the following Repast features:

2D Grid space
Histograms
OpenSequenceGraph
Network space

2D Grid space

This helps us to clearly see the Lift's movement and their trajectory.Each little block represents a Lift(ordered by their ID increasing from left to right). When the block's color is gray it means that it has no requests to fulfill.



Visualization of the Simulation execution

OpenSequenceGraph

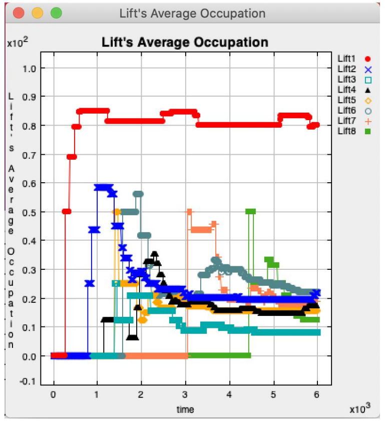
This type of plots allows us to analyze how certain dependent variables decrease or increase over time, allowing us to understand some behaviours in run time.

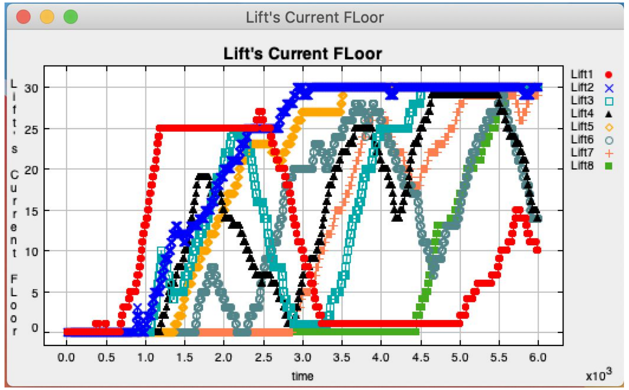
The dependent variables that we visualise through OpenSequenceGraphs in the context of this project are:

Current Lift position - allowing us to view the movement patterns of lifts

Average lift occupation - allowing us to determine how full lifts are

Flow of people at each floor (in/out) - allowing us to see when people enter and exit lifts at each floor

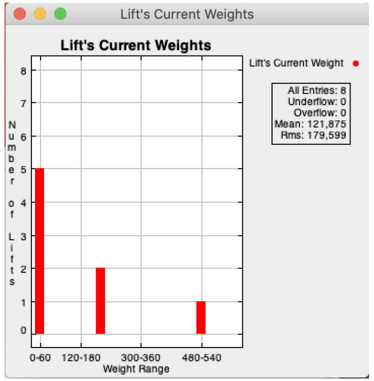


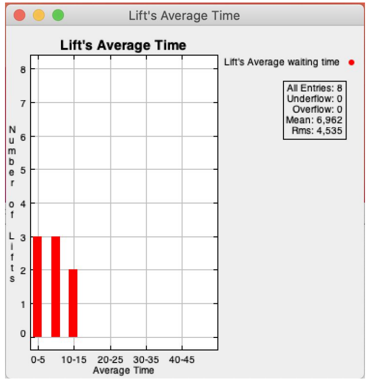


Visualization of the Simulation execution

Histograms

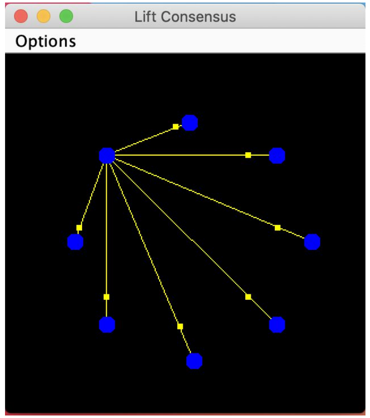
Using histograms we were able to find out in real time how many lifts are in a given weight range and also observe how the average time for handling requests by lifts is distributed.

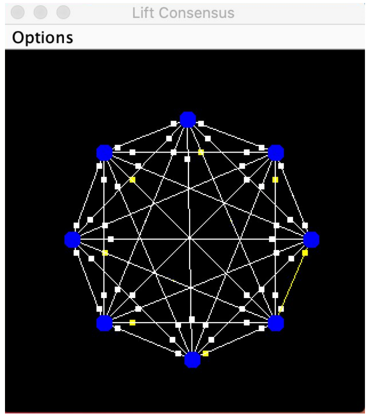




Network space

Graphical representation of the consensus algorithm execution, responding to a floor Panel request. The agents are represented by the blue nodes, the INFORM message for the proposed time is represented by white node connections and the HALT message by the yellow ones





Experimental Results and Analysis

For experimental purposes we were interested in viewing the correlation between the independent Variables **Number of Lifts** and **Weight** and the dependent variables **Average occupation** and **Waiting time**.

With this in mind we ran the simulation for a building with **100 floors**, **2.5 Lift Speed**, 5 **floor distance** and **1 Time at floor**.

As evident in the tables on the right, we were able to define a **strong correlation between higher max Weight and lower average occupation**, as expected. The correlation between a **higher number of lifts and lower occupation** was **small** but significant.
We were also able to determine that the impact of max Weight on Average Wait Time is **almost insignificant** in comparison to the **impact of the number of lifts available.**

Average occupation Percentage

Number of Lifts\Weight	300	600	900
4	0,361847	0,236887	0,1715397
9	0,318032	0,206038	0,1570168
14	0,308168	0,175866	0,1248112

Average Waiting Time

Number of Lifts\Weight	300	600	900
4	77,22	81,95398	89,322715
9	29,52606	30,97316	36,014745
14	25,36657	26,04132	27,743447

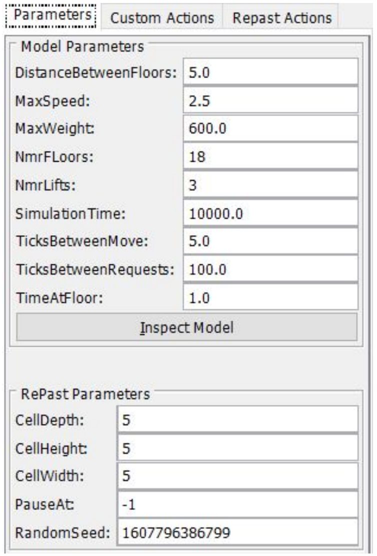
Conclusions / Further Work

As it stands, the project has the ability to simulate several distinct configurations of buildings with different characteristics. It presents useful information about the Building and each Lift in real time.

Since we have a lot of independent variables available at the start of the run, we can conduct a limitless number of experiments to try to find the best Lift configuration for a given Building.

That would actually be the work we had planned for the future of the project.

We did not have much time to do this deep experimenting but that would definitely be an interesting thing to do. Finding the most efficient Lift heuristic and configuration for every type of Building.



Model Parameters

DistanceBetweenFloors:	5.0
MaxSpeed:	2.5
MaxWeight:	600.0
NmrFLoors:	18
NmrLifts:	3
SimulationTime:	10000.0
TicksBetweenMove:	5.0
TicksBetweenRequests:	100.0
TimeAtFloor:	1.0

RePast Parameters

CellDepth:	5
CellHeight:	5
CellWidth:	5
PauseAt:	-1
RandomSeed:	1607796386799

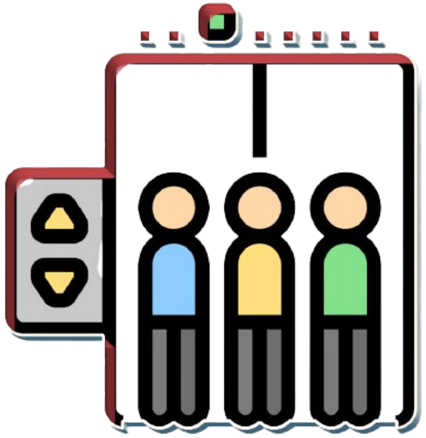
4MIEIC01	**Group 13**
António Dantas	up201703878
João Macedo	up201704464
Vítor Gonçalves	up201703917

Additional Information - From JADE to Repast3

As we have used the same Multi-Agent System as the first assignment, some changes had to be made to the classes that were previously used, in order to be able to run the MAS on top of Repast.

Fortunately, as we are using the **Eclipse IDE**, we were able to use a plugin that makes this conversion automatically, **MASSim2Dev.**

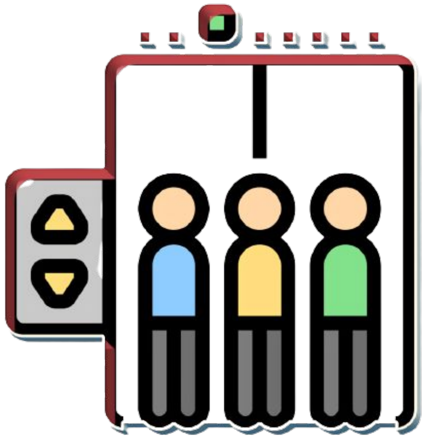
A **RepastLauncher** class was also generated, which extends **Repast3Launcher**. This class is the entry point of the system and is responsible for creating the agents and the visualization elements.



Additional Information - Initial problems

It should be noted that we **had some problems** related to the fact that **Repast** is single threaded, which required some changes, mainly, in the **flow** of the program at startup and **TickerBehaviour**.

After these small changes, the project was dedicated to the **exploration and creation** of the various types of visual information that we wanted to show the user in real time.



Additional Information - Time vs. Ticks

The **Ticker Behaviour** represented a setback on our new agent system.

On our first project we used a scheduler based on real time to get the Lift agents to move. Doing this with our Repast port wasn’t very reliable and would cause big problems in the future, mainly on the Batch-Mode implementation. The solution was to change this behaviour to be fully driven by ticks.

The same happened with our **Request Agent**, that would send requests to the Lifts every 3 seconds. Now, we can specify this interval with ticks at the beginning of every run. The default value is 100 ticks.

Additional Information - Detailed Execution E.g

To run our project we recommend using the Eclipse IDE.
Our source code that was sent with this presentation **must be imported**.
Ensure that all necessary jars are added to the classpath, i.e., *jade.jar*, *sajas.jar* and all jars in the lib folder of repast3 and also *repast.jar.*
Run project as Java application in **launcher.RepastLauncher.**
There are **two ways to run** our MAS, in **batch mode** and **graphical interface mode**.

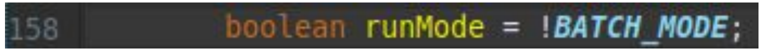
Both in **batch mode** and **graphical mode**, after executing the number of ticks specified in the ***simulationTime***, the systems stops its simulation and saves the information in csv files.

At the end of the execution, each lift will have a csv file(lift_id + date) and there will be a file(analysis + date) with more general information about the system. These files can be found in the analysis folder.

Additional Information - Detailed Execution E.g

Batch Mode

Mostly used to collect data for experiments as it runs faster without a graphical interface. Shows only simple logs on the console that allows us to view the messages sent and which elevator takes care of the request sent.
By default the MAS will not run in Batch mode, so, to make it possible the following(RepastLauncher.java, line 158)



```
158        boolean runMode = !BATCH_MODE;
```

should became: ***runMode = BATCH_MODE***

Input parameters can be changed in *RepastLauncher.java* .

Graphical Mode

Default mode.
Will show, using visual elements such as plots and histograms, as the dependent variables vary over time.
Input parameters can be changed in Repast's GUI.