AASMA Project Report

Emergency responses - Group 31

João Antunes nº 87668

Maria Inês Morais nº 83609

Stefano Gonçalves nº 87706

1.ABSTRACT

The challenge of emergency response resource management is one with many possible variables that requires different response behaviors, advanced-making capabilities, and complex negotiation.

This problem is defined by the need distribute available resources in a timely fashion, prioritizing emergencies correctly and when shortages arise to negotiate these resources so that all emergencies can be answered. Its relevance resides in the fact that emergencies take place every day and the way they are managed can be a determinant factor in people's lives.

Hence, the ability to create a system that effectively dispatches ambulances in order to respond to emergencies so that the response times are minimized, there are as few shortages of resources as possible and the utmost number of emergencies are successful, i.e., the utmost number of lives are saved, is what motivates us to create this optimal solution.

2. INTRODUCTION

2.1 Problem definition

For our system we developed the following problem to be tackled:

We designed a world consisting of a map containing Communication Emergency Centers, Hospitals, Ambulances and Emergencies. The map is split up in zones, and the Communication Center of each zone controls the hospitals and ambulances of that zone. The Emergencies are static and laid out randomly throughout the map, each Communication Center will be responsible for the emergencies in their own area. This way, the Communication Centers are responsible for tackling the emergencies in their own area with the right resources and in the least amount of time. However, if the Centers run out of resources, for instance, don't have any free ambulances at that time or their hospitals are full, they'll also have to tackle the challenge of communicating with the other centers to ask for resources, for instance, asking for the closest available hospital out of their own area.

3. APPROACH

In this project, we implement a multi-agent system, in which we have two kinds of agents, the Communication Emergency Centers and the Ambulances. The main agents are the Communication Centers, they control the Ambulances and the Hospitals and are implemented as deliberative agents i.e., following a Beliefs, Desires, and Intentions model (BDI).[1] The Ambulances have their own set of sensors and actuators, but only follow orders coming from the Communication Centers.

The Communication Centers are able to receive emergency calls and manage their hospital's resources, namely, allocate which emergency transports should be sent to which emergencies. The Communication Centers can also communicate with each other, coordinating their approach in the distribution of resources.

For this purpose, we defined the following types of emergencies and ambulances:

- Types of Emergencies:
  - Transportation of Patients
  - Basic Life Support
  - Intense Care Support
- Types of Ambulances:[2]
  - Vehicle for transportation of patients
  - Basic Medical Aid Ambulance
  - Intense Care Support Ambulance

Each type of vehicle is specified for the respective emergency.

For the types of Emergencies, we attributed the following possible intervals of severities:

- Transportation of Patients: severity between 1 and 3
- Basic Life Support: severity between 4 and 7
- Intense Care Support: severity between 8 and 10

All Ambulances have a maximum capacity of patients they can carry. Hospitals also have a maximum capacity of patients they can take in, becoming full, upon hitting their maximum capacity.

3.1 Environment Definition [3]

The project's environment consists of 4 Communication Emergency Centers (each at a corner of the map in *Figure 1*), 6 Hospitals and 9 Ambulances, which are all static. We also considered that each tile of the model is a 4-way passage, meaning that when there are at least two objects at the same tile there will be no collision between those objects.

In the example of *Figure 1* below there are also 5 static emergencies, that have a number from 1 to 10 to represent their severity. The emergencies are static in the sense that throughout the execution of a run they do not change, nor new ones appear, but on executing a new run, different new emergencies with random locations and severities will appear. To try to simulate the real world, Emergencies with higher severity appear with less probability than ones with lesser probability, (It is also possible to change the number of emergencies that appear in the code, which we will do later on to evaluate the project.) As it is also possible to see in *Figure 1*, each Communication Center is responsible for one area, with the hospitals controlled by that Communication Center being the same color as the center.

In the beginning of the program the ambulances of each hospital are stationed at the location of their hospital, so it is not possible to see them while they are stationed, only when they're in transit.

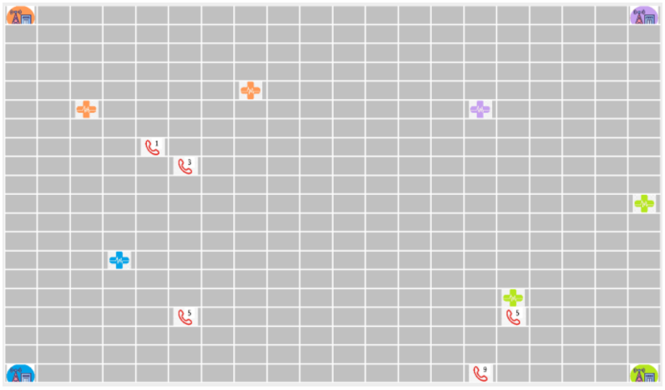


Figure 1 - Environment of the Project

The colors of the ambulances represent their type, as we can see in *Figure 2*:

- Vehicle for transportation of patients: white
- Basic Medical Aid Ambulance: yellow
- Intense Care Support Ambulance: red

As an ambulance picks up patients from an emergency, the emergency disappears from the map. When the ambulances are carrying patients, a black dot appears on top of them.

If a hospital becomes full, the word "full" appears on it.

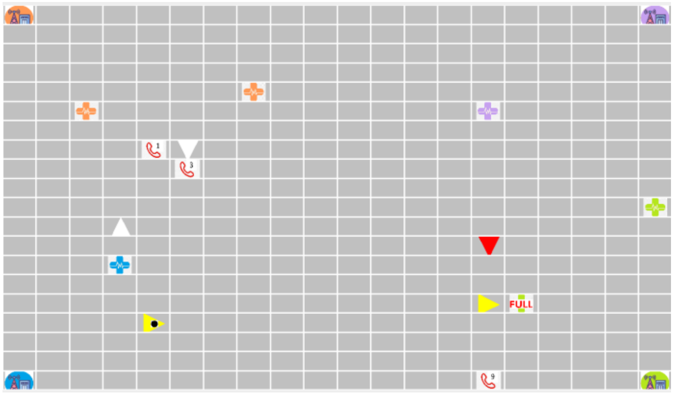
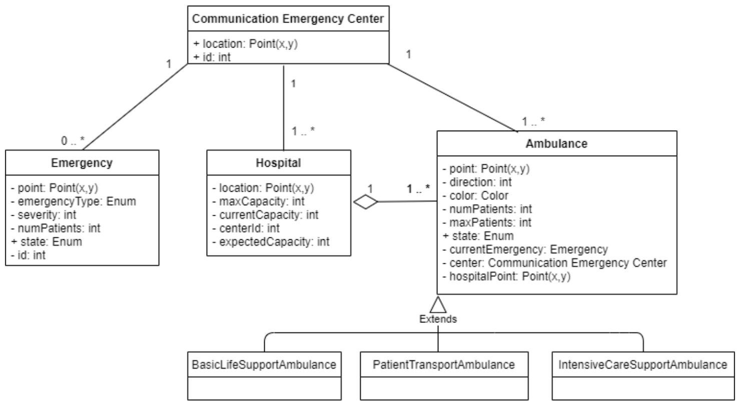


Figure 2 - Environment of the Project with Ambulances

- Inaccessible: The agents cannot obtain complete, accurate, up-to date data about the environment's state since each center does not have information about all hospitals and ambulances, other than his own. The centers also don't know the other center's location, only their owns.
- Non-deterministic: An action does not have a single guaranteed effect since an emergency response is not guaranteed to be successful.
- Dynamic: The world changes while the agent is deliberating since hospitals can become full.
- Discrete: The number of possible actions and percepts our agents can execute in the environment are finite.
- Episodic: The world can be divided in a series of intervals (episodes) independent of each other since each new execution deals with a new model. A new execution will provide new different emergencies in random locations with random severities.

3.2 Agent Definition [3]

3.2.1. Architecture



3.2.2. Agent's model

The Communication Center's desires are the following, by order of preference:

- requestHelp
- dispatchToHospital
- wait
- dispatchVehicle
- returnToOriginalHospital

The Communication Center's actions are the same as the desires.

The Ambulance's actions and states are the following:

- Actions:
    - pickUpPatients

- dropPatients
- moveForward
- turn
- turnRight
- turnLeft

- States:
  - free,
  - ongoing
  - arrivedAtEmergency
  - arrivedAtHospital

At the beginning of the programs all ambulances are at the state "*free*".

On a run of the program, the agents have the following behaviors:

- Desire wait:
  The Communication Centers start by detecting all emergencies on the map, sorting the emergencies by severity, and picking the one with the highest severity. After picking an emergency, each center will communicate with the other centers and send each other their distance to that emergency. The center that is the closest to that emergency, is the one that stays responsible for that emergency. All the other centers, that are not the closest to the emergency, will then drop the emergency, and move on to a new emergency repeating the same process. This is necessary since the center's only know their own location and not the others center's location. While each center waits for a response from the other centers it stays on the desire ***wait***. Once a center receives a response from all the other centers and becomes responsible for an emergency it moves on to the desire ***dispatchVehicle***.
- Desire dispatchVehicle:
  The Communication Center accesses the information of the emergency its responsible for and determines the type of emergency and the number of patients involved. Then it searches for a free ambulance in the Communication Center's area that is of the appropriate type to deal with that emergency, has capacity to transport the patients involved and most importantly is the closest to the emergency. Once it finds an ambulance it changes the ambulance's state to "*ongoing*", builds a plan for the ambulance to get from its location to the location of the emergency and sends a message to the ambulance with the plan to execute. Subsequently the ambulance uses its movement actions (e.g. moveForward, turnRight, turnLeft, turn) to follow the plan and get to the emergency. Once the ambulance gets to the emergency, it executes the action "*pickUpPatients*", picking up the patients, setting its own state to "*arrivedAtEmergency*" and sending a message to the Communication Center, informing the center it has arrived at the emergency. The Center adds this ambulance to an array of "ambulancesArrivedAtEmergency".
- Desire dispatchToHospital:
  While deliberating, the Communication Center checks to see if it has ambulances that have arrived at emergencies, if so, it moves on to the desire ***dispatchToHospital.*** This allows for the Communication Centers to dispatch various ambulances to emergencies, and only when the ambulances get there to decide how to proceed next. Since ***dispatchToHospital*** is higher up in the desire's preferences, when the center has ambulances arrived at emergencies waiting to receive new directions, it will focus first on getting these ambulances to a nearby hospital so the patients don't die, and only later will it focus on new emergencies. This way, when the Communication Center has an ambulance that has arrived at an emergency, the center checks how many patients the ambulance is carrying and looks for the closest hospital in its area with vacancy for that number of patients. If there are no hospitals available in the area of the Communication center, the center sends messages to all the other centers to find the closest hospital with vacancies in the other areas. Once the Communication Center has picked the closest hospital available, it again builds a plan to send to the ambulance, so the ambulance can get to that hospital. The ambulance proceeds to execute its movement actions to follow the plan and get to hospital. When the ambulance gets to the hospital it executes the action "*dropPatients*", dropping off the patients at the hospital by adding to the hospital's current capacity the number of patients the ambulance was carrying. The ambulance also changes its own state to "arrivedAtHospital" and sends a message to the Communication Center to inform it that it has arrived at the hospital. The Center adds this ambulance to an array of "ambulancesArrivedAtHospital"
- Desire returnToOriginalHospital:
  While deliberating, the Communication Center checks to see if it has ambulances that have arrived at the hospital, if so, it moves on to the desire ***dispatchToOriginalHospital*** to send the ambulance back to its original hospital. This

desire is of the least priority, so the Communication Center will only get to it once it has dealt with all the existing emergencies. The Communication Center checks the "hospitalPoint" attribute of the ambulance, which marks its original hospital point, builds a plan for the ambulance to get back to its original hospital and sends it to the ambulance. The ambulance once again uses its movement action to follow the plan and get to the original hospital, when it gets to the hospital the ambulance sets its own state to "free".

- Desire requestHelp:
  This desire is a special case of when the Communication Center tries to follow the desire ***dispatchVehicle*** but realizes it either does not have an ambulance of the correct type to answer the emergency or all its ambulances are already busy. In this case it will get to desire to ***requestHelp***. Upon pursuing this desire, the Communication Center will communicate with all other centers to find the closest Center to the emergency with a free ambulance of the correct type. Once the Communication Center finds this center it sends the emergency to this center, so that now this new center is responsible for the emergency.

3.2.3. Agents' Sensors

- Communication Emergency Center:
  - getCenterId( ): Returns the Center own id.
  - hasAmbulancesArrivedAtHospital( ): Verifies if any there is any ambulance that has arrived to an hospital with the emergency's patients.
  - hasAmbulancesArrivedAtEmergency( ): Verifies if there is any ambulance that has arrived to the emergency point.
  - hasReservedHelpRequestAmbulance( ): Checks if the there is an ambulance that must be dispatched to an emergency which center has requested for help.
  - hasNoAvailableVehicle( ): Verifies if there is no available vehicle for the current emergency.
  - hasNoAvailableHospital( ): Determines if there is no available hospital to receive the emergency's patients.
  - getWorldCentersNumber( ): Returns the number of centers that exist in the model.
  - getWorldDistances( ): Returns an array with the distances of all communication centers to an emergency. This sensor is used when the centers are deciding which center should be handling the emergency according to their distance.
  - detectEmergency( ): Detects an incoming emergency.
  - hasEmergencies( ): Checks if there is emergencies left to handle.
  - isHelpRequested( ): Determines if an emergency is from a center that as sent an help request to handle it.
  - getEmergencyType( ): Returns the emergency type.
  - getEmergencySeverity( ):Returns the emergency severity value.
  - getEmergencyPoint( ): Returns the emergency location.
  - getEmergencyId( ): Returns the emergency Id number.
  - getAmbulanceDirection( ): Returns the ambulance direction. This is used when building a path plan to an ambulance.
  - getAmbulanceNumPatients( ): Returns the number of patients that an ambulance has.
  - getAmbulancePoint( ): Returns the ambulance location.
  - getAmbulanceHospitalPoint( ): Returns the ambulance's Hospital point.
  - getAmbulanceCurrentEmergency( ): Returns the emergency that an ambulance is currently handling.
  - getAmbulanceState(): Returns the ambulance's state.
  - getAmbulanceForwardPosition( ): Returns the position that the ambulance is currently facing forward. This is used when building a path plan to an ambulance.
  - getHospitalAmbulances( ): Returns the hospital's ambulances list.
  - getHospitalMaxCapacity( ): Returns the hospital's maximum capacity.
  - getHospitalExpectedCapacity( ): Returns the hospital's expected capacity.
  - getHospitalPoint( ): Returns the hospital's location.
- Ambulance:
  - getDirection( ): Returns the ambulance's direction.
  - getNumPatients( ): Returns how many patients the ambulance has.
  - getCurrentEmergency( ): Returns the emergency that the ambulance is currently handling.
  - getHospitalPoint( ): Returns the ambulance's hospital location.

3.2.4. Agent's Actuators

- Communication Emergency Center:
  - selectAmbulance( ): Selects an appropriate and available ambulance to handle an emergency.

    - selectHospital( ): selects an available hospital to receive the emergency's patients.
- Ambulance:
    - moveForward( ): Moves the ambulance one position forward.
    - turnRight( ): The ambulance rotates to the right.
    - turnLeft( ): The ambulance rotates to the left.
    - turnRandomly( ): The ambulance rotates randomly.
    - pickUpPatients( ): The ambulance picks up the emergency's patients.
    - dropPatients( ): The ambulance drops the emergency's patients at the hospital.

3.2.5. Agent's Communication

- Communication Emergency Center:
    - sendAmbulancePlan( ): Sends the path plan to an ambulance.
    - receiveDistance( ): Receives an array with the distances of all communication centers to an emergency. This function is called when the centers are deciding which center should be handling the emergency according to their distance.
    - receiveNewEmergency( ): Receives an emergency from another center that has sent an help request.
    - receiveHelpRequest( ): Receives an help request sent from another center.
    - receiveAmbulanceMessage( ): Receives a message from an ambulance. This function is called when an ambulance either has arrived at the emergency point or has arrived at the hospital.
- Ambulance:
    - receivePlan( ): Receives a path plan from a communication center

4. Experiment Analysis

In order to test our system's goals, we decided to evaluate the following metrics:

- Rate of successfully handled emergencies.
- Average number of steps taken to handle all emergencies.
- Impact of shortages of resource levels.

To perform this test, we used the previously defined environment. We distributed the ambulances and hospital's capacity the following way to really challenge the system (The hospital's ids can be seen in the *Figure 3* below):

- Hospital 1 (maximum capacity: 3):
    - 1 Intense Care Support Ambulance
    - 1 Basic Medical Aid Ambulance
    - 1 Vehicle for transportation of patients
- Hospital 2 (maximum capacity: 1):
    - 1 Basic Medical Aid Ambulance
- Hospital 3 (maximum capacity: 2):
    - 1 Vehicle for transportation of patients
- Hospital 4 (maximum capacity: 1):
    - 1 Basic Medical Aid Ambulance
    - 1 Vehicle for transportation of patients
- Hospital 5 (maximum capacity: 2):
    - 1 Intense Care Support Ambulance
- Hospital 6 (maximum capacity: 1):
    - 1 Vehicle for transportation of patients

For simplicity of the example, we assumed there is always only one patient per emergency.

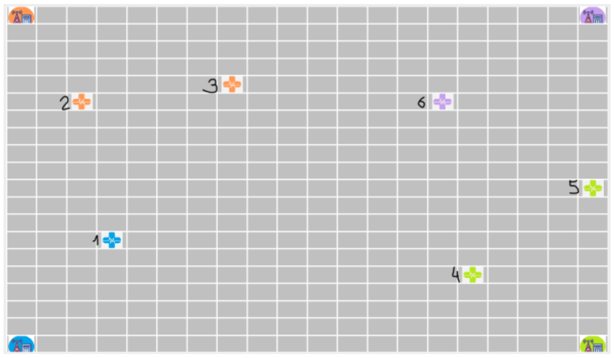
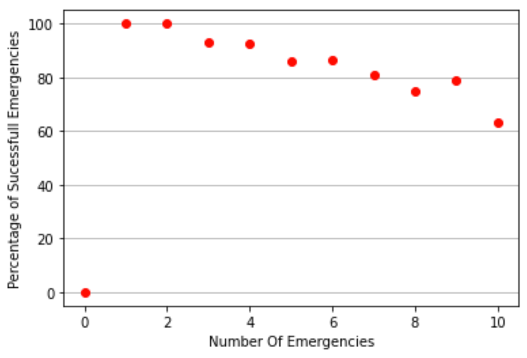


Figure 2 - Hospital's id

To generate the graphs in this report, we executed 10 runs, for each n, with n being the number of emergencies from 1 to 10. We then calculated the average of the values of interest for each of the 10 runs of n.

It's important to note that to get more conclusive results many more iterations for each test were needed, but since the project is slow to run and there wasn't much time only 10 iterations were done for each test.

4.1 Rate of successfully handled emergencies



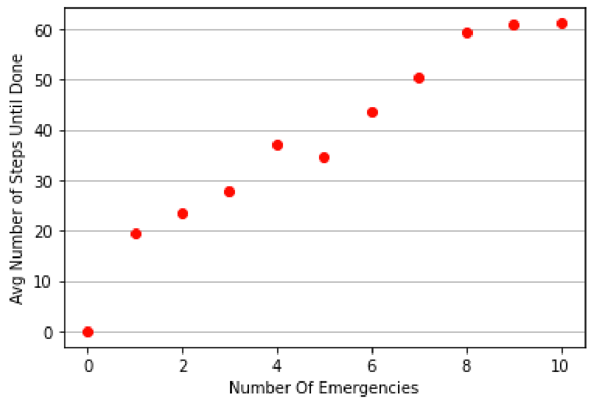
Graph 1 - Percentage of Successful Emergencies per number of Emergencies

To calculate which emergencies are successful and which are not we created a function that is updated at each step, taking into account the type of emergency and a threshold that increases the probability of the patient dying the longer it takes for the emergency to be resolved and the higher the emergency severity.

Interpreting the graph, we can see that with increasing number of emergencies, the percentage of successful emergencies starts to decrease. This is because the system starts running out of resources, taking longer to respond, and hence having more patients dying.

At 10 emergencies the system starts to hit a breaking point with on average only 60% of successfully handled emergencies, but with some executions going as low as only 10% of successfully handled emergencies.

4.2 Average number of steps taken to handle all emergencies

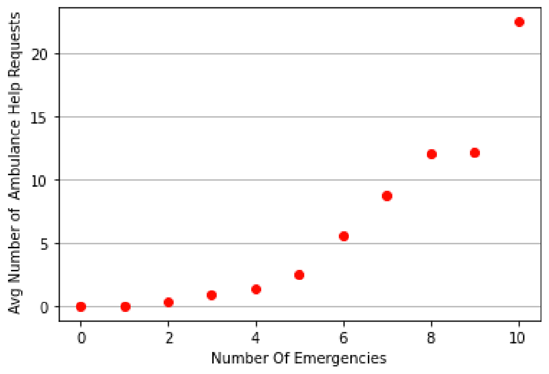


Graph 2 – Average number of Steps taken per number of Emergencies

Once again, the higher the number of emergencies the longer it takes, i.e., the more steps must be taken, to solve all emergencies.

This is because with a higher number of emergencies each zone will start having their ambulances busy and hospitals full, having to ask other centers further away from the emergency to handle it or having to divert patients to hospitals that are not the closest, since the closest are full.

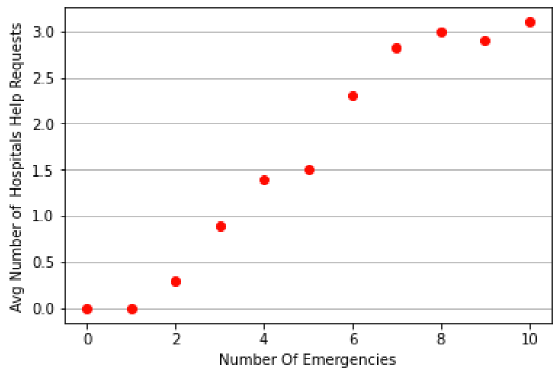
4.3 Impact of shortages of resource levels



Graph 3 - Average number of Ambulance shortages per number of Emergencies

In graph 3 we can see, the number of times a Communication Center had no free ambulances to respond to an emergency and had to ask another Center to resolve the emergency.

It can be seen that up until 5 emergencies, there's relatively low shortages of resource levels, and the system copes well with them. After that the shortages of resources increase, with 10 emergencies being once again at a breaking point of the system.



Graph 4 - Average number of Hospital shortages per number of Emergencies

In graph 4 we can see, the number of times a Communication Center had the hospitals in its area full and had to ask other Centers for hospitals with vacancies.

It is possible to observe that up until 5 emergencies there were at maximum 2 times the center's needed to deliver their patients in a hospital outside their area. After this it increases somewhat, with the highest being over 3 times the center's needing to deliver their patients out of their area. This time, there is no breaking point. This is because it was also not our goal with this example, since if all hospitals become full, there is no more actions the agents can perform.

5. Conclusion

Taking the Experiment Analysis into consideration, our project became relatively less efficient than we initially expected, since when our agents deal with more than 5-6 emergencies the success rate of our model begins to decrease, and when it reaches the breaking point, which seems to start when the model deals with 10 or more emergencies, it starts to get bad results. This leads us to assume that, with more than 10 emergencies the agents will no longer be able to deal with so many emergencies. However, we also attributed relatively low resources to our agents, in this case not many ambulances per hospital, so we could observe it hitting a breaking point. Nevertheless, in a real-life scenario, this system would need to be improved and made more efficient in order to be a more reliable system. With more time, we could've improved our agents.

REFERENCES

[1] Deliberative Agents - slides provided at the course page: https://fenix.tecnico.ulisboa.pt/downloadFile/1689468335703219/06%20DeliberativeAgents.pdf

[2] Auto Ribeiro - Tipologias de Ambulâncias: https://autoribeiro.com/tipologias-de-ambulancias/

[3] Model's and agent's code implementation adapted from the provided solution code to the resolution of the laboratory 3 of the course's page: https://fenix.tecnico.ulisboa.pt/downloadFile/1689468335703786/Lab03.pdf

[4] Agent and environment properties - slides provided at the course page: https://fenix.tecnico.ulisboa.pt/downloadFile/1970943312353657/02AgentProperties.pdf