

# Caspian RoboCup Rescue Simulation Agent Competition Team Description

Seyed Hamid Hamraz, Seyed Shams Feyzabadi, Amir Khayati Motlagh

Computer Engineering Department,  
Iran University of Science and Technology, Tehran, Iran  
hamid\_hamraz@yahoo.com, sh.feyzabadi@gmail.com

**Abstract.** RoboCupRescue Simulation League is held each year, and various teams from all over the world participate in the league. Each team is asked to prepare a team description as a qualification material. In this document we briefly describe Caspian Rescue Simulation Team structure for 2005 competitions. We assumed that the reader is familiar with the RoboCup Rescue Simulation Project. We'll start with *world modeling*, *knowledgebase* and *learning* in our team. We'll talk about some *path finding* algorithms in a *dynamic network* and finish our discussion with Caspian *decision making* phase.

## 1 Introduction

One of the most important and primitive issues in designing an agent is to develop a model of outer world for the agent or a *world model*. This model should be updated as well as possible to be used efficiently in *decision making* phase.

Sometimes we'd like to assign agents learn from what has happened to them. In this case a *world model* is not enough, because it just talk about the last state of the world. Solving this problem, we should improve the *world model* to be able to save all the events happened since the start to the end of the agent's assignments.

Another important problem in developing a *rescue agent* is *path finding* in a *dynamic network* which means that the cost of edges may vary and the agent may not get aware of new costs. This dynamic situation makes *path finding* a complex job and no algorithm would be able to produce the shortest path for all the cases. Therefore we had better to look for algorithms which give us the shortest path for most cases. We should not forget that efficiency of the algorithms is crucial and we can not apply any algorithm for our purpose.

Final phase in developing an agent is *decision making* in which agent decides what to do according to information in its *world model*. This phase is the most fateful and can be very complex. A definite claim is that, this phase can not be started until *world modeling* phase finishes, or we can say that for developing a powerful *decision making* module we have to implement an appropriate *world model* first.

In this document we will briefly describe Caspian team structure for the issues mentioned above. We assumed that the reader is familiar with RoboCup Rescue Simulation Project and we won't talk about rules and features of the simulation (see

[1]). In section 2 we will describe Caspian *world modeling* structure. *Knowledge base and learning* is studied in section 3. Section 4 describes algorithm we used for *path finding* and finally we'll briefly present *decision making* unit for different agent types in section 5 and 6.

## 2 World Modeling

One of the most important and primitive works in developing an agent is design and development of a *world model* and good procedures for updating it as well as possible. The better the *world model* is designed and the better this structure is updated, the better and more natural *decision making* is done by agents. Therefore, before any discussion about other jobs we have to do, we decided to accomplish the *world modeling* unit.

Our *world model* is a complete and uniform module for all kind of agents. For example, if a fire brigade extinguishes a building, all the rescue team will get aware of this action after a short time. This perfect *world model* will help us through *decision making* phase. In the system, each agent will tell everything it is sensed during the current cycle through an arbitrary protocol, and others will update their model of world (see [1] for understanding who the listeners of the messages are). Here is a problem, consider we have 10 fire brigades and they can hear only 4 messages every cycle, so each fire brigade would get aware of only 4 other brigades each cycle. Solving this problem, we appointed the center agents to be the center of communications. In each cycle they gather all the information received from their homogenous humanoid agents, and retell a message consisting all of them. On the other side, the humanoids will prefer to hear their center's messages rather than other humanoids. By now, communication between homogenous agents would be well done. For extending this system to support world modeling among heterogeneous agents the centers should also gather other centers' information, but another problem will occur and that is sending and receiving repetitive messages in which a center will receive its own messages and resend them again and again. Simple way to overcome this problem is to recognize such repetitive messages and to avoid resending them. For this purpose we added a *time* field to each object of *world model* which indicates the cycle the object last time updated. For the messages, the *time* field of each message part is also sent. The receivers won't take care of the message parts which have expired *time*.

## 3 Knowledge Base and Learning

Developing a model of world for *decision making* is a crucial job. Now, suppose that the agents would like to look back at what had happened for them in order to learn something useful for their future assignments. In this case *world model* would not suffice, because it just keeps the last state of the world, in the other words it is just a temporary memory for *decision making* jobs and it can not be utilized for learning purposes as well.

In order to overcome the problem, we decided to improve the previous *world model*. The *world model* was a set of objects with their individual properties. For each *dynamic property*<sup>1</sup> we defined an array instead of a single variable. This would allow us to save the value of the property for each cycle in its own place. By now, we can indicate that we changed the *world model* to *knowledge base*.

Learning activity is a costly process and it would be a risk to apply it during simulation time. Therefore we decided to allocate the last cycle of simulation to this job. We could also save the *knowledge base* in a file and perform offline learning. By the way, a simple pseudo code for our agents would be as following:

```
if (time < simulating_duration) {
    updateKnowledgeBase();
    doAction();
}
else {
    learnFromKnowledgeBase();
}
```

The code indicates that the agents will only save information during rescue jobs, and at the end they will learn from what is saved in their memory. This learning will be useful for their future rescue assignments.

## 4 Path Finding

Another important issue in developing a rescue agent is *path finding*. In a system like rescue simulation environment, finding routes would not be a simple job to be done with *Floyd*, *Dijkstra* or other simple algorithms, because we are not aware of the roads situation exactly and we don't know whether they are open or blocked. For this problem, according to the current status we used different algorithms:

- *When the agent is sure that all the roads are open:* In this case, using the static algorithms like *Floyd* is a good choice. Routes between each two objects can be cached at the beginning.
- *When the agent is not aware of some roads situation and it has a set of destinations to go:* In this case we start from the source and examine all the routes to all possible directions parallelly. For blocked roads we add an infinite value to the length of the route. For open roads, length of the roads and for unknown roads  $c * roadLength$  is added. For extending the routes, smallest-length route is chosen and extended. The process continues until a destination is reached after an extension.
- *When the agent is not aware of some roads situation and it has one destination to go:* This case is similar to previous one and can be accomplished with that algorithm too, but for upgrading the performance we changed the algorithm a bit. We start from the source and we extend the routes to all directions. Values are added to route's length the same as before, but the route which is the closest to the destination, is chosen for extension each time. If the closest route was blocked we should

---

<sup>1</sup> Property which, its value varies during simulation (e.g. *area of a building* is static but its *fieriness* is dynamic).

choose the second closest one. The process continues until the destination is reached.

It is trivial that, these algorithms will surely return a route, as long as the maps are connected.

## 5 Decision Making

All the units described before were communal among all agent types. In this section we are going to describe the final phase, *decision making*, for different agent types. *Decision making* unit would be individual for each type of agents, while they have different capabilities.

### 5.1 Center Agents

Center agents will not have an explicit *decision making* unit, because they are not able to do actions. As indicated before their main task is *world modeling*, according to their strong ability of communication. They may also used for some coordination purposes.

### 5.2 Police Forces

Police forces are the simplest agents in designing phase and they can be really fateful sometimes. In our system, each police agent extracts the blocked roads from its *world model* in each cycle. It will choose the most important one to be opened through a *priority assigning* system. This *priority assigning* depends on various factors like opening the way for fire brigades, opening the way to refugees and etc. We'll talk about *priority assigning* system later. Polices are always careful not to go collectively for a specific blocked road, because not only they can not help each other, but also they barricade themselves. After opening the main roads of the city in a way that transportation becomes easy, police agents will divide the city into several zones like pieces of a pizza. Each agent will go to its zone in order to search buried civilians and blocked roads. After finishing the search, each police agent will take care of the civilians in its zone and reports the civilians' situation periodically. This will help ambulance agents to be better aware of civilians and to do a better rescue job. If a police agent does not find a buried person in its zone, it'll move to refuge to protect itself.

### 5.3 Ambulance Teams

Rescuing people in a disaster space is the most important goal for rescue team (see scoring formula [2]). Therefore ambulance agents are the most fateful agents in situation which fires are extinguished soon. Caspian ambulances will extract buried civilians from their *world model* in each cycle and through *priority assigning* system they

choose the most important civilian to be rescued. They always work collectively and the task for coordinating them is assigned to one of them called *leader*. *Leader* will decide which civilian should be rescued and tells its decision to others. If ambulance agents did not find any buried civilian in their *world model* they start searching the city. They do not divide city into zones, because if a buried civilian found, all of them should gather. They search the city collectively and they are also careful not to get in a building collectively or sequentially. Each building is reserved only for one ambulance team to be searched and this will prevent them from searching a building more than once. This process continues until they find a buried person. Finally after finishing the entire city and rescuing all the buried people they'll move to refuges to protect themselves.

#### 5.4 Fire Brigades

Spreading fires and igniting buildings is another factor for calculating the score. However it is not as important as people's death, dying civilians in fires makes fire brigades job so fateful though. The brigades should also act collectively, but sometimes they had better to get separated in two or more groups. Fire agents extract burning buildings from their *world model* in each cycle. They choose the most important one for extinguishing through a *priority assigning* system and they act collectively. In order to prevent them from barricading each other we decided to make them do extinguish action from inside buildings. The coordination task between them is assigned to one of them called *leader*. *Leader* will decide which fire should be extinguished first and tells its decision to others. After finishing all the fires, these agents will divide city into zones like pieces of pizza and they start searching buried people in their own zones. After finishing the entire city each of them will take care of buried civilians in its zone and reports the civilians' situation periodically. If a fire brigade does not find a buried person in its zone, it'll move to refuge to protect itself.

### 6 Priority Assigning

We pointed to a *priority assigning* system in previous sections, but we didn't talk about its implementation. In this section we are going to discuss about the system more precisely. Each agent type deals with a set of homogeneous jobs and should choose the most important one to do first. For example, police agents should find out which blockade should be removed first, ambulance teams deal with buried people and fire brigades should find the most dangerous fire to be extinguished first. Thus we have a set of jobs and we'd like to determine which one should be done first. Here is a simple pseudo code for choosing the most important fire:

```

Allocate new array PRIORITIES with the same size as
FIRES array;
Initialize each element of PRIORITIES by 1.0;

For i = 1 to FIRES.size do
  If FIRES[i].burningTime < 3 then
    PRIORITIES[i] *= 10;
  If FIRES[i].fieriness > 2 then
    PRIORITIES[i] *= 0;

    PRIORITIES[i] /= (self.distanceTo(FIRES[i]))2;
    . . .

Find the FIRE with the greatest PRIORITY and do action;

```

A priority array is allocated first. The priority is set for each job and the one with the greatest priority is chosen to be done. The same procedure is done for blockades and buried people in their related modules.

## References

1. Morimoto, T.: *How to Develop a RoboCup Rescue Agent*,  
<http://robomec.cs.kobe-u.ac.jp/robocup-rescue>
2. *RoboCup 2004 Rescue Simulation League Rules* V1.01, January 20, 2004  
<http://kaspar.informatik.uni-freiburg.de/~rcr2005/rules.php>
3. Russell, S. J., Norvig, P.: *Artificial Intelligence A Modern Approach*, Prentice Hall, Englewood Cliffs, New Jersey 07632 (1995)
4. Mitchell, T. M.: *Machine Learning*, McGraw-Hill (1997)