DAMAS-Rescue Description Paper

Sébastien Paquet, Nicolas Bernier, and Brahim Chaib-draa

DAMAS laboratory, Laval University, Canada
{spaquet;bernier;chaib}@damas.ift.ulaval.ca

Abstract. In this paper, we describe DAMAS-Rescue, a team of agents participating in the RoboCupRescue simulation competition. In the following, we explain the strategies of all our agents that will be used at the world competition in 2004 at Lisbon in Portugal. In short, *FireBrigade* agents are choosing the best fire to extinguish based on the knowledge they have learned with a selective perception learning method. *AmbulanceTeams* are always rescuing the same civilian based on the messages received by the *AmbulanceCenter*. This center chooses the civilian that maximize the number of civilians that could be saved afterwards. *PoliceForces* have a sector assigned to them at the beginning of the simulation and they are responsible to clear all roads in this sector.

1 Introduction

It is not an easy task to develop a multiagent system to act in the RoboCupRescue environment, since it is a complex environment with a lot of challenges. To share our work on this project, we explain in this article how the DAMAS-Rescue [1] team works by describing the strategies that we have implemented for the 2004 world competition.

The main part of our work has been the conception of a selective perception learning method to enable the *FireBrigade* agents to learn their effectiveness when they are extinguishing fire. By evaluating their capacity and the utility to extinguish a fire they are able to coordinate their fire choices on the most important fires to extinguish. In section 3, we describe in more details the approach we have used for the *FireBrigade* and the *FireStation* agents.

In this paper, we also describe the strategies used by the other agents. For instance, the *AmbulanceTeam* agents are rescuing the civilian that should give them the time to save the most civilians afterwards. The *PoliceForce* agents receive a sector at the beginning of the simulation for which they are responsible to clear all the roads. For example, if an agent asks for a road to be cleared, the assignment of a police force agent to clear the road is automatic, it is the *PoliceForce* responsible for the sector in which the road is.

Additionally, we would like to mentioned that all our agents have been developed with the *JACK Intelligent Agent*^{TM} programming language [2]. This agent oriented programming language enabled us to write in a convenient way all the plans an agent needs to encounter every possible situations.

Even though we consider that most readers of this article already know about the RoboCupRescue Simulation environment, we describe it very briefly in the next section to enable all people to understand it. Afterwards, we describe the strategies that we intent to use at the 2004 RoboCupRescue world competition. We begin by describing the *FireStation* and the *FireBrigades*, then the *AmbulanceTeams* and *AmbulanceCenter* and finally, the *PoliceForces* and the *PoliceOffice*.

2 The RoboCupRescue Environment

The goal of the RoboCupRescue simulation project aims to simulate rescue teams acting in large urban disasters [3, 4]. Precisely, this project takes the form of an annual competition in which participants are designing rescue agents trying to minimize damages, caused by a big earthquake, such as civilians buried, buildings on fire and blocked roads. In the simulation, participants have approximately 30 to 40 agents of six different types to manage:

FireBrigade There are 10 to 15 agents of this type. Their task is to extinguish fires. Each *FireBrigade* agent is in contact by radio with all other *FireBrigade* agents as well as with the *FireStation*.

PoliceForce There are 10 to 15 agents of this type. Their task is to clear roads to enable agents to circulate. Each *PoliceForce* agent is in contact by radio with all other *PoliceForce* agents as well as with the *PoliceOffice*.

AmbulanceTeam There are 5 to 8 agents of this type. Their task is to search in shattered buildings for buried civilians and to transport injured agents to hospitals. Each *AmbulanceTeam* agent is in contact by radio with all other *AmbulanceTeam* agents as well as with the *AmbulanceCenter*.

Center agents There are three types of center agents: *FireStation*, *PoliceOffice* and *AmbulanceCenter*. These agents can only send and receive messages. They are in contact by radio with all their mobile agents as well as with the other center agents. A center agent can read more messages than a mobile agent, so center agents can serve as information centers and coordinators for their mobile agents.

In the simulation, each individual agent receives visual information of only the region surrounding it. Thus, no agent has a complete knowledge of the global state of the environment. This uncertainty complicates the problem greatly because agents have to explore the environment and they also have to communicate to help each other to have a better knowledge of the situation.

3 FireBrigade and FireStation Agents

In this section, we focus on the *FireBrigade* and the *FireStation* agents. As mentioned before, the task of those agents is to extinguish fires. Therefore, at each time step, each *FireBrigade* agent has to choose which building on fire to extinguish. Furthermore, in order to be effective, they have to coordinate themselves on the same buildings on fire, because more than one agent is often needed to extinguish a building on fire.

A *FireBrigade* agent is faced with the problem of choosing a fire to extinguish between a list of buildings on fire. Since there could be a lot of fires, agents do not consider all fires at once. They separately choose which fire zone to extinguish and which specific building in the chosen fire zone to extinguish. Fire zones are simply groupings of near buildings on fire.

To stop a zone from spreading, the *FireBrigade* agents have to extinguish all fires at the border of the zone. It is pointless to constantly change from one fire zone to another, because by doing so, all zones would spread. A better strategy is to choose a zone and stop the spreading or really slow it down, before choosing another zone. Therefore, when an agent has to choose a building to extinguish, it firstly has to choose the fire zone. In other words, agents are using a two level decision making process. First, they look at the global view of the situation, i.e. groups of buildings on fire. Afterwards, they use more detailed information to choose which specific building to extinguish in the chosen fire zone.

Since each mobile agent has only a local view of the situation, it is often hard for a *FireBrigade* agent to have a good view of the global situation. Therefore, it is difficult for *FireBrigade* agents to choose between the fire zones since they don't have a good knowledge of the fire zones that are far from them. To solve this problem, we have moved the task of choosing a fire zone from the *FireBrigade* agents to the *FireStation* agent. Since the center agent can receive more messages, it normally has a better knowledge of the global situation compared to the *FireBrigade* agents. Therefore, it is the responsibility of the *FireStation* agent to allocate fire zones to *FireBrigade* agents. After they have received their fire zone, *FireBrigade* agents have the possibility to learn how to coordinate their efforts to extinguish the more important fires in a given fire zone. Notice that a building is important to extinguish if it puts civilians or other buildings in danger.

When an agent wants to choose a building on fire to extinguish, it goes through the list of all buildings on fire, it evaluates them independently by calculating the utility and the expected reward of each. The utility is an estimation of the importance to extinguish a given building, calculated by considering the number of civilians and buildings on danger. The expected reward, learned with the algorithm presented in section 3.1, can be seen as an estimate of the capacity to extinguish a given fire.

3.1 Selective Perception

To learn the expected reward of extinguishing one building on fire or one fire zone, we have used a selective perception technique [5], because the description of our states is too big. More precisely, both the *FireStation* agent and the *FireBrigade* agents use this algorithm to learn the expected reward of their respective tasks. With this technique, an agent learns by itself to reduce the number of possible states. The algorithm uses a tree structure similar to a decision tree. At the beginning all states are considered to be the same, so there is only the root of the tree. After some experiences, the agent tests if it would be interesting to divide the different states, represented as the leaves of the tree. By doing so, the agent creates new states, by expanding a leaf of the tree, and thus it refines the view it has about the state space. An advantage of this algorithm is that it distinguishes only states that really need to be distinguished. This has the effect of reducing the state space of the reinforcement learning algorithm and thus facilitating the learning process.

Recording of the Agent's Experiences At each time step t, the agent records its experience captured as an "instance" that contains the observation it perceives (o_t) and the reward it obtains (r_t). Each instance also has a link to the preceding instance and the next one, thus making a chain of instances. Consequently, an instance at time t is defined as:

$$i_t = \langle i_{t-1}, o_t, r_t, i_{t+1} \rangle \tag{1}$$

In our case, we have one chain for each building that an agent chooses to extinguish. A chain contains all instances from the time an agent chooses to extinguish a building until it opts for another building. An agent opts for another task if the building is extinguished or this agent finds a "better task" to pursue, due to some changes in the environment. Therefore, during the simulation, the agent records many instances organized in many instance chains. It keeps all those instances until the end of the simulation.

There is no learning taking place during the simulation since it would take too much time. Agents are evolving in a real-time environment and they cannot afford to take time to learn during the simulation. Thus, the learning process takes place after the simulation, when the agents have time to learn. At this time, the agents regroup all their experiences together, the tree is updated with all those new instances and the resulting tree is returned to each agent. By regrouping their experiences, agents can accelerate the learning process.

Tree Structure To learn how to classify instances, we use a tree structure similar to a decision tree. The tree divides the instances in clusters depending on their expected reward. The objective here is to regroup all instances having similar expected rewards. If two instances have similar rewards, it means that the agent does not have to distinguish between those two situations, since it should act the same way in both situations.

The algorithm presented here is an instance-based algorithm in which a tree is used to store all instances which are kept in the leaves of the tree. The other nodes of the tree, called center nodes, are used to divide the instances with a test on a specific attribute. To find the leaf to which an instance belongs, we simply start at the root of the tree and head down the tree choosing at each center node the branch indicated by the result of the test on the instance's attribute value. Each leaf of the tree also contains a Q-value indicating the expected reward if a fire that belongs to this leaf is chosen. In our approach, a leaf l of the tree is considered to be a state for the reinforcement learning algorithm.

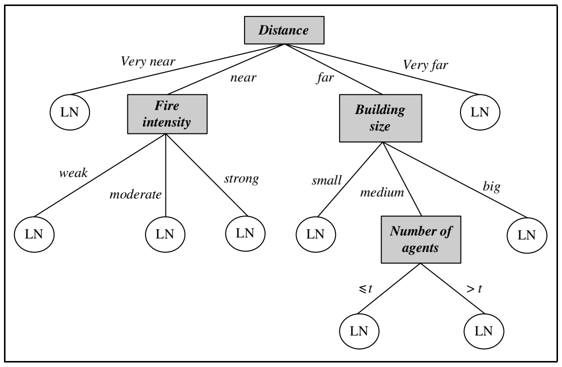

Fig. 1. Structure of a tree.

An example of a tree is shown in Figure 1. Each rectangular node represents a test on the specified attribute. The words on the links represent possible values for discrete variables. For example, the root of the tree is a center node containing a test on the "distance" attribute. This is a discrete attribute that can take the values *very near*, *near*, *far* and *very far*. Therefore, this node has four children nodes, one for each possible value. The tree also contains a center node testing on a continuous attribute, the "number of agents". A test on a continuous attribute has always two possible results, it is either less or equal to the threshold or greater than the threshold. It is why the center node with the test on the "number of agents" attribute has two children: one if the number of agents is less or equal to the threshold t and another if it is greater than t. The oval nodes (LN) are the leaf nodes of the tree where the instances and the Q-values are stored.

Update of the Tree After a simulation, all agents put their new experiences together. This set of new experiences is then used to update the tree. The algorithm adds all recorded instances to the tree and afterwards, it tries to expand the tree. Therefore, the first step is simply to add all instances to the leafs they belong to. We found the leaf of an instance by starting at the root and heading down the tree choosing at each center node the branch indicated by the result of the test on the instance.

The second step updates the Q-values of each leaf node to take into consideration the new instances which were just added. The updates are done with the following equation:

$$Q(l) \leftarrow R(l) + \gamma \sum_{l'} Pr(l'|l)Q(l') \qquad (2)$$

where $Q(l)$ is the expected reward if the agent tries to extinguish a building belonging to the leaf l, $R(l)$ is the estimated immediate reward if a fire that belongs to the leaf l is chosen, $Pr(l'|l)$ is the estimated probability that the next instance would be stored in leaf l' given that the current instance is stored in leaf l. Those values are calculated directly from the recorded instances. $R(l)$ is the average reward obtained when a fire belonging to this leaf was chosen and $Pr(l'|l)$ is the proportion of next instances that are in leaf l':

$$R(l) = \frac{\sum\limits_{i_t \in I_l} r_{t+1}}{|I_l|} \tag{3}$$

$$Pr(l'|l) = \frac{|\{\forall i_t \in I_l | L(i_{t+1}) = l'\}|}{|I_l|} \tag{4}$$

where $L(i)$ is a function returning the leaf l of an instance i, I_l represents the set of all instances stored in leaf l, $|I_l|$ is the number of instances in leaf l and r_{t+1} is the reward obtained after the instance i_t was chosen.

After the Q-values have been updated, the third step checks all leaf nodes to see if it would be useful to expand a leaf and replace it with a new center node containing a new test, thus dividing the instances more finely.

To find the best test to divide the instances, we try all possible tests, i.e. we try to divide the instances according to each attribute describing an observation. After all attributes have been tested, we choose the attribute that maximizes the error reduction as shown in equation 5 [6]. In fact, the test is chosen only if the expected error reduction is greater than a certain threshold, if not, it means that the test does not add enough distinction, so the leaf is not expanded. The error measure considered is the standard deviation ($sd(I_l)$) on the instances' expected rewards. The expected error reduction obtained when dividing the instances I_l of leaf l is calculated using the following equation where I_d denotes the subset of instances in I_l that have the d^{th} outcome for the potential test:

$$\Delta error = sd(I_l) - \sum_d \frac{|I_d|}{|I_l|} \times sd(I_d) \tag{5}$$

The standard deviation is calculated on the expected reward of each instance which is defined as:

$$Q_I(i_t) = r_t + \gamma Pr(L(i_{t+1})|L(i_t)) \times Q(L(i_{t+1})) \tag{6}$$

where $Pr(L(i_{t+1})|L(i_t))$ is calculated using equation 4 and $Q(L(i_{t+1}))$ is the value returned by equation 2.

As mentioned earlier, one test is tried for each possible instance's attribute. For a discrete attribute, we divide the instances according to their value for this attribute and for a continuous attribute, we test different thresholds to find the best one. Finally, after the tree has been updated, we update the Q-values again to take into consideration the new state space.

3.2 Agents Coordination

During the simulation, the agents use the tree created offline to choose the best fire zone and the best building on fire to extinguish. Since the *FireStation* agent has a better global view of the situation, it is its responsibility to suggest fire zones to *FireBrigade* agents. Those agents have however a better local view, so they can choose which particular building on fire to extinguish in the given zone. In short, both agents are using a reinforcement learning algorithm with selective perception, but for different tasks at different perception level. By doing so, we can take advantage of the better global view of the *FireStation* agent and the better local view of the *FireBrigade* agent at the same time.

Fire Zones Allocation To allocate the fire zones, the *FireStation* agent has a list of all fire zones. For each fire zone, it retrieves from the tree all the expected rewards for all possible number of agents. More precisely, it finds the leaf for the first fire zone with one agent and it records the expected reward. Then, it tries with two agents for the same fire zone and it continues like this until the expected reward is greater than a specified threshold. If this happens it means that considering past experiences, the *FireStation* agent expects the chosen number of agents to be enough to extinguish the zone. The *FireStation* agent does this with all fire zones, ending up with an expected reward and a number of agents for each zone. Then, it chooses between all zones with an expected reward greater than the threshold the one that uses the least agents. Afterwards, it removes this zone and the assigned agents from its lists and continues the process with the remaining agents and the remaining fire zones. It continues until there is no agent or fire zone left. When a zone is chosen, the assigned agents are those that are closer to this zone. At the end, the *FireStation* agent sends to each *FireBrigade* agent their assigned fire zones.

Choice of Buildings on Fire When choosing a building to extinguish, a *FireBrigade* agent has a list of buildings on fire it knows about in the specified fire zone. For each building in this list, the agent calculates the utility of extinguishing this building. The utility is calculated by considering the area in danger and the number of civilians in danger. The area in danger is composed of all intact buildings near the building on fire. The civilians in danger are the civilians trapped in those buildings.

Afterward, the agent sorts those buildings according to their utility. Each agent assigned to a specific fire zone has its own list, but since they have, most of the time, the same observations about the zone, their lists are normally similar. This similarity in the situation evaluation enables them to coordinate themselves without any communication. To do so, each agent is considering the buildings in the order of their utility. Therefore, the building with the highest utility is considered first. Precisely, each agent uses a tree, learned with the algorithm presented in section 3.1, to evaluate the number of *FireBrigade* agents that should be needed to extinguish the building. The tree returns an expected reinforcement considering the number of agents, the building area, the composition of the building and the fire intensity. For a given building at a certain time, the last three characteristics are fixed. To evaluate the number of agents needed to extinguish the building, each agent calculates the expected reinforcement for each possible number of agents. It begins by calculating the expected reinforcement with one agent, than with two agents, than with three agents, and so on until the expected reinforcement exceeds a certain threshold. When the expected reinforcement reaches the threshold, it means that the agents should be able to extinguish the building quite fast.

After an agent has found the number of agents needed for the first building, it "virtually" assigns to it the agents following a prefixed order. For example, if there are 5 agents and 4 buildings on fire. If an agent thinks that three agents are required to extinguish the more useful building, than the first three agents will be assigned to this building. Afterwards, the other two agents could be assigned to the second building by doing the same process. We say that they are "virtually" assigned because no agent really assigns buildings to other agents. Each agent assigns a building to itself, but it considers that the other agents are choosing their building according to the same information. If they actually have the same information, they should be well coordinated on the most important buildings to extinguish.

3.3 Other Strategies

At any moment, if the agent's tank is empty, it will go refill it at the closest refuge. The agent waits at the refuge until its tank is full. Also, if there is no building in fire, the agent will go refill its tank to be ready if a new fire is found. Finally, if there is no fire and its tank is full, the agent will search for civilians by visiting every buildings.

4 AmbulanceTeam and AmbulanceCenter Agents

In this section, we present the *AmbulanceTeam* and the *AmbulanceCenter* agents. In their case, the center has a lot of responsibilities, since it is the one making all the decisions about which civilians to rescue and in which order. We first explain how the center makes its decisions and then we explain how the *AmbulanceTeams* act based on those decisions.

4.1 AmbulanceCenter

The *AmbulanceCenter* agent has an important role since it chooses which civilians to rescue and in which order. At each turn, it sends the ordered list of civilians to rescue at the *AmbulanceTeams*. Those agents are described later, but for now it is worth mentioning that every *AmbulanceTeams* are rescuing the same agent to reduce the rescuing time.

The decision process has been centralized in order to reduce the amount of messages. When a building contains many buried civilians, the messages concerning the civilians are big. This increases the chances of those messages to be lost. In the centralized decision process, if the center loses a message, the only consequence is that the civilians would not be taken into account immediately. However, in the decentralized way, each *AmbulanceTeam* may have different beliefs about civilians, so they could choose different civilians to rescue. Since all the reasoning is done with the hypothesis that all *AmbulanceTeams* are rescuing the same civilian, the reasoning process becomes invalid.

Now we will explain how the center chooses the agent to rescue. First of all, the center sends to the ambulances the order to save the other agents of the securing team (i.e. *FireBrigades*, *PoliceOffices* or other *AmbulanceTeams*), if there are some that are buried. It is really important to rescue our agents rapidly to enable them to do their tasks.

When all our agents have been saved, the center agent calculates which civilians to save and in which order. To do that, it uses a greedy planning algorithm to try to maximize the number of civilians that could be saved. Each task corresponding at saving a civilian has a time length giving the necessary time to save the civilian, taking into account the travel time to go to the civilian location, the rescuing time and the time to transport it to the refuge. Each task also has a deadline representing the expected death time of the civilian.

The Algorithm 1 presents how the first civilian is chosen. The chosen civilian is the one that maximizes the number of civilians that could be chosen after this one (*cpt* in the algorithm). When the first civilian has been chosen, it is removed from the list of civilians to rescue and the algorithm is called again to find the second one. Those two best civilians to rescue are sent to each *AmbulanceTeam* at each turn.

4.2 AmbulanceTeam

As we have said before, *AmbulanceTeams* are rescuing civilians according to the order given by the *AmbulanceCenter*. In each message from the center, the ambulances receive two civilians to rescue. Since all *AmbulanceTeam* is receiving the same messages, they are always rescuing the same agent. This reduces the time necessary to rescue an agent.

When a civilian has been rescued, i.e. that it is not buried anymore, only one ambulance will load the civilian and transport it to the hospital. The chosen ambulance is the *AmbulanceTeam* with the smaller identification number, identified as the team leader. It is the same ambulance that will send a message to the center to inform it that the civilian has been rescued. All other *AmbulanceTeams* begin to work on the next civilian immediately.

When there are no civilians to rescue, *AmbulanceTeams* search to find buried civilians. They begin by creating a list of buildings based on the scream for help heard. Each building in a perimeter of 30 meters from the point where the message was heard are added to the list. Agents are then visiting all buildings in the list. If they found some buried civilians, they send a message to the *AmbulanceCenter* indicating the position and the healthiness of each civilian. After that, it is possible that some civilians were not heard by the *AmbulanceTeams*, so they visit all unexplored buildings to try to find buried civilians.

```
**Function** FOUND-BEST-CIVILIAN($Civilians$) return $firstCivilian$

**Inputs:** $Civilians$: all known civilians.
**Return:** $firstCivilian$: the first civilian to rescue.

$firstCivilian \leftarrow$ null
$maxCpt \leftarrow 0$
for all $x$ in $Civilians$ do
  $cpt \leftarrow 0$
  for all $y$ in $(Civilians - \{x\})$ do
    if $x.time + y.time +$ CURRENT-TIME $\leq y.deadline$ then
      $cpt \leftarrow cpt + 1$
    end if
  end for
  if $cpt > maxCpt$ then
    $maxcpt \leftarrow cpt$
    $firstCivilian \leftarrow x$
  end if
end for
Return $firstCivilian$
```

Algorithm 1: Algorithm used to calculate the first civilian to rescue. *time* is the rescuing time for this civilian and *deadline* is its expected death time.

5 PoliceForce and PoliceOffice Agents

Polices are playing a key role in the rescue operation by clearing the roads, thus enabling all agents to circulate. Without them, some actions would be impossible because agents would be indefinitely blocked by roads' blockades. Therefore, it is really important for them to be fast an efficient.

An important aspect of our strategies is that we have divided the map in nine regions. So, at the beginning of the simulation, when the agent receives the information on the map, it begins by dividing the map in nine homogeneous regions and sends its position to the *PoliceOffice*. When the *PoliceOffice* has received all the positions of the *PoliceForce* agents, it assigns a sector to the nine agents that are closer to the center of a sector. Thus, there is one an only one *PoliceForce* affected to a sector and this agent has the responsibility of this sector. By doing so, we have divided our *PoliceForces* in two groups: those with a sector and those without a sector.

In the next subsections, we describe in details all the strategies, but to give an overview, here is a list of the strategies in their priority order:

1. Unblock other agents in the sector (with sector).
2. Clear roads between fires and refuges in the sector (with sector).
3. Clear roads around refuges in the sector (with sector).
4. Clear all the roads of the sector (with sector).
5. Search for civilian based on help calls in the sector (with sector).

6. Search for civilian in all unexplored buildings of the sector (with sector).
7. Clear roads between fires and refuges (without sector).
8. Search for civilian in all unexplored buildings (without sector).

5.1 PoliceForce With a Sector

First of all, the highest priority task of a *PoliceForce* agent with a sector is to help the other agents in its sector. This is very important, because *FireBrigades* and *AmbulanceTeams* would not be able to do their tasks if they are blocked. Therefore, when an agent is blocked it will send a message to the *PoliceForces* indicating its position. When it receives the message, the *PoliceForce* responsible of this sector will add it to the list of roads to clear. In this list, it will choose the closer road and will go to clear it.

When the preceding list is empty, the *PoliceForce* agents with a sector work to open the roads for the *FireBrigade* agents by clearing all roads between a fire in the sector and the closest refuge. The refuge can be in the sector or not. This is a proactive action to help the *FireBrigades*, because there is a good chance that those agents would ask for a road clearing. *FireBrigades* have a good chance to use those roads because they have to refill their tank at the refuge, so they are often going from a fire to a refuge and in the other direction too. This strategy helps to reduce the communications and the agent movements.

Afterwards, *PoliceForce* agents are clearing roads around refuges. They will clear all roads in a perimeter of 40 meters around the refuge, if it is in their sector. This is also a proactive action, because refuges are intensively used by the *FireBrigade* and the *AmbulanceTeam* agents.

When the three preceding tasks have been done, *PoliceForces* clear all the roads in their sector. They first calculate the best path to visit all the roads in their sector. After this, they will follow this path and clear all block roads on their path. At the end of this task, not only all roads are cleared, but the *PoliceForce* agents have had the chance to hear buried civilians scream for help. It is in fact the next task of the agent: search for buried civilians. If they found some buried civilians, they send a message to the *AmbulanceCenter* indicating the position and the healthiness of each civilian.

To summarize, after all the strategies have been done by the *PoliceForces* with a sector, all the roads are cleared and the *AmbulanceCenter* knows the position and the healthiness of all civilians, if no messages have been lost. After completing all the tasks in their sector, *PoliceForces* will act as if they had not received any sector, see section 5.2.

5.2 PoliceForces Without a Sector

The behavior of those *PoliceForces* are quite simple, they have only two tasks. The first one is to clear roads on the path from a fire to a refuge. Unlike the other type of *PoliceForces*, they are not restrained to a specific sector, so they choose the closest fire. By clearing the path from a fire to a refuge, they are clearing the roads around fires and around refuges. This is interesting since they are really important spots to clear to help the other agents to move freely in the city. When there are no more roads to clear from a fire to a refuge, they will search for buried civilians by visiting all unexplored buildings.

5.3 PoliceOffice

The *PoliceOffice* role is relatively simple in the simulation. First of all, they allocate the sectors for the *PoliceForces* at the beginning of the simulation. After that, they are simply redirecting messages coming from the *PoliceForces* and from the other centers.

6 Conclusion

This paper has presented the strategies used by all the agents of the DAMAS-Rescue team, an intended participant in the 2004 RoboCupRescue simulation world competition. To resume, *FireBrigade* agents are choosing the best fire to extinguish based on the utility of a building on fire and their capacity to extinguish it. The capacity is learned by the agent with a selective perception learning method. *AmbulanceTeams* are always rescuing the same civilian based on the messages received by the *AmbulanceCenter*. This center chooses the civilian that maximize the number of civilians that could be saved afterwards. *PoliceForces* have a sector assigned to them at the beginning of the simulation and they are responsible to clear all roads in this sector.

Finally, we hope that this article could help some new teams in developing agents to participate in the next RoboCupRescue simulation competitions.

References

1. Paquet, S.: DAMAS-Rescue web page. Online (2004) http://www.damas.ift.ulaval.ca/projets/RobocupRescue/index.php.
2. Nick Howden, Ralph Rnnquist, A.H., Lucas, A.: Jack intelligent agents summary of an agent infrastructure. In: Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montral, Canada (2001)
3. Kitano, H., Tadokor, S., Noda, H., Matsubara, I., Takhasi, T., Shinjou, A., Shimada, S.: Robocup-rescue: Search and rescue for large scale disasters as a domain for multi-agent research. In: Proceedings of the IEEE Conference on Systems, Man, and Cybernetics (SMC-99). (1999)
4. Kitano, H.: Robocup rescue: A grand challenge for multi-agent systems. In: Proceedings of ICMAS 2000, Boston, MA (2000)
5. McCallum, A.K.: Reinforcement Learning with Selective Perception and Hidden State. PhD thesis, University of Rochester, Rochester, New-York (1996)
6. Quinlan, J.R.: Combining instance-based and model-based learning. In: Proceedings of the Tenth International Conference on Machine Learning, Amherst, Massachusetts, Morgan Kaufmann (1993) 236–243