A communication and coordination model for ‘RoboCupRescue’ agents.

masters thesis

S.B.M Post
sbmpost@science.uva.nl

M.L. Fassaert
mfassaer@science.uva.nl

10th June 2004

University Of Amsterdam

Abstract

This thesis will focus on software agents that operate within a simulation environment called the RoboCupRescue Simulator System (RCRSS). In this system an earthquake is simulated and agents are programmed to counter the aftereffects. After a thorough description of the system itself, we present various theories and approaches we used to develop these software agents.

Because many design decisions depend on how agents represent their environment, we will first tackle the problem of creating a suitable world model. Then we will look into communication and show how effective teamwork can be achieved by sharing and distributing information. After formally describing the problem, we glue the pieces together and present a coordination scheme for the generation of behaviours to enable agents to 'execute' these. We then continue with a discussion of several other approaches found in literature regarding world modelling, communication and coordination.

Since RCRSS is relatively new, it was not always easy to find documentation. For that reason a lot of effort had to go into researching the internals of the system. Therefore it is hoped this thesis will not only be of scientific value, but may also serve as a reference point for future research.

Acknowledgements

The authors of this thesis would like to thank their supervisor Arnoud Visser for his persistent support throughout the whole project. Whenever we got stuck, Arnoud was always prepared to provide us with new ideas. His active involvement in the project led to many discussions and at times much disagreement. In the end we owe it to exactly these opposing views that our efforts have resulted in what we believe is a well balanced thesis. We would also like to thank Arnoud for giving us the chance to compete in the RoboCupRescue league and helping us organize and fund the trip to Padova to attend in person. Finally we would like to thank him for giving us access to the computational and caffeinated resources at Science Park Watergraafsmeer. Without these, no master thesis would have been possible.

S.B.M. Post would further like to thank his parents Nelleke and Gerard for their support. By recognizing his interest in computer science they allowed him to fully concentrate on his study without worries. Furthermore he would like to thank his brothers and sisters Peter, Tineke, Marjan and Annemiek for being there when needed. He also thanks Luc Negrijn for going over the English in this thesis. Finally, thanks go to all his friends and fellow students, especially Magiel Bruntink, Steven de Rooij, Jan de Vos, Mark Thompson and Irwin Oppenheim.

M.L Fassaert would like to thank his parents for the financial and motivational support during his entire studies. He thanks his brother for choosing a medical study, thereby saving him from the disgrace of graduating after his brother. He would also like to thank his fellow students who proofread this thesis, Stijn Oomes for showing us the nicer parts of Padova and his employers at RouteVision Benelux and UvA Rechten ICT for letting him combine his study with a source of income.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

On a regular basis the media confront us with shocking images of natural disasters such as earthquakes. Often these disasters result in the loss of many lives or the destruction of entire cities. On January 17, 1995 "The Great Hanshin Earthquake Disaster" in Japan claimed the lives of more than five thousand citizens of the city of Kobe. Most victims were crushed to death when their houses collapsed or burned to death by the fires that followed the earthquake. On September 26, 1997 an earthquake struck Central Italy and seriously damaged several buildings and artworks in the town of Foligno.

These events inspired a group of researchers to develop a computer program for simulating earthquakes. The prototype version of this program was called the 'RoboCupRescue Simulator System' (RCRSS). To analyze ways in which rescue operations are carried out in the aftermath of an earthquake and assess their effectiveness, intelligent robots were introduced. The idea was to let these robots play an active role in the simulation and to have them operate either autonomously or as members of a team to deal with the aftereffects of the disaster. This thesis concentrates on the development of the intelligence required to allow the simulated robots to execute their tasks.

1.1 RoboCupRescue

To stimulate further research on intelligent agents, a 'RoboCupRescue' league was started as part of the RoboCup project [1]. The RoboCup project numbers several leagues, notably the 'RoboCupSoccer' league, which belongs to another branch of research at the University of Amsterdam [12], [16].

In the RoboCupRescue league, teams from different countries compete for the best simulation scores. The authors of this thesis have joined in two of these competitions in the year 2003; one in Paderborn (Germany) and the other in Padova (Italy). On various occasions all participants came together to discuss techniques and to present their work, allowing other teams to benefit from the exchange.

1.2 The simulation

In the disaster simulation, software agents are presented with a city map pinpointing areas affected by the earthquake. This is illustrated in figure 1.1.

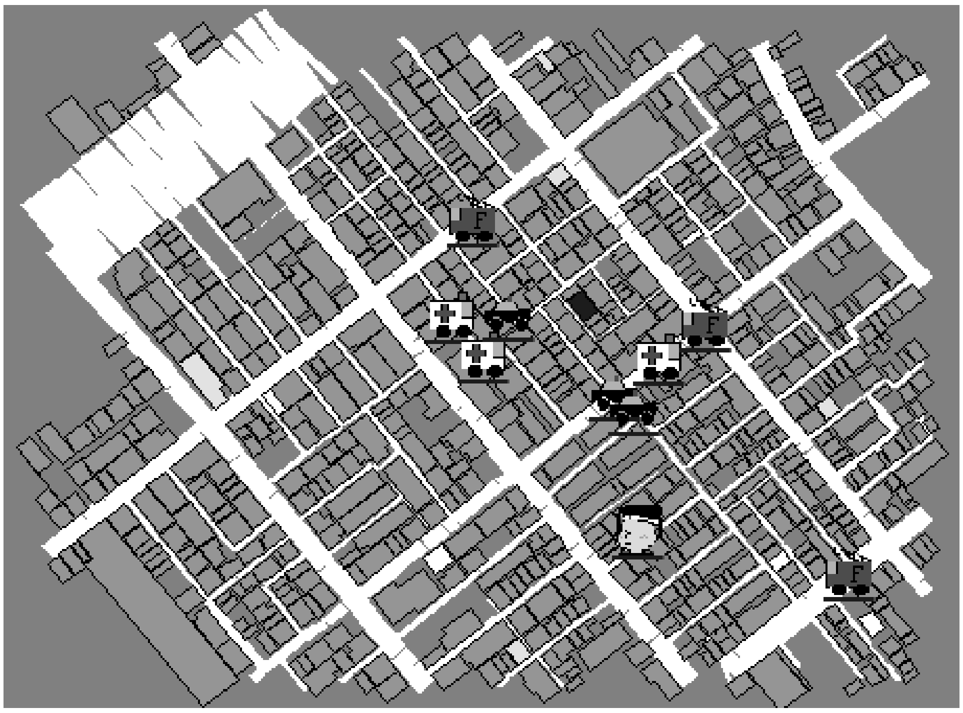


Figure 1.1: *This is a map of the city of Kobe. Some buildings are shelters for the civilians, others are on fire and should be extinguished. Police agents clear the blocked roads and ambulances rescue the civilians. Each car represents a humanoid agent.*

The city has been simplified showing only objects like buildings, roads, cars and agents. When the earthquake strikes buildings collapse, roads are blocked, civilians are buried under the debris, and fires break out. These events affect the agents on the simulated map, each one being capable of a different response. The main objective is to save as many civilians as possible and to minimize the damage from the fires.

In the simulation the lapse of time is defined in terms of simulation cycles. An ordinary simulation run consists of 300 simulation cycles although the number of cycles is configurable. The actual time lapse per cycle too can be altered, but generally equals one second, bringing the total simulation run to 5 minutes. In each cycle agents can submit actions to the simulator system and receive new input based on fresh observations. It is important that these observations are processed quickly to prevent compromising intented actions. A detailed discussion on this topic can be found in section 2.4.

1.2.1 Agents

The simulator system has three different controlled entities: civilians, mobile agents and stationary centers. Civilians represent ordinary people. Their standard behavior is to flee from danger and to seek shelter in refuge buildings. The agents and centers are to be programmed and belong to any of the following types: ambulance staff, police or fire fighters. Each has its own control center staffed by agents of the same type. Agents have different tasks assigned to them:

- Ambulance agents rescue civilians from under the debris and transport them to refuges.
- Police agents clear roads that are blocked as the result of collapsing buildings.
- Fire agents put out fires in burning buildings.

Throughout this thesis we will use the term 'platoon agents' for the mobile agents in the above list, whereas 'centers' will be used to denote the stationary control centers.

1.2.2 Maps

Three city maps were used for the simulation, as shown in Table 1.1. The most prominent objects were counted and the size of the map was determined by taking the length and width of the smallest rectangle containing the full city map.

map name	total area (m^2)	#buildings	#roads	#nodes
Kobe	$417.4 * 316.5$	739	820	765
Foligno	$2038.1 * 1517.4$	1084	1480	1369
Virtual City	$413.1 * 417.8$	1266	621	532

Table 1.1: *The three maps and some key figures. In general the number of civilians in the simulation lies between* 70 *and* 90*, whereas the number of agents is* 15 *to* 30*. Agents can only see (sense) objects at sufficiently close range.*

1.3 Analyzing the problem

One of the many problems in developing agents is the need to base their actions on recent and relevant world information. For instance agents should have an indication of what other agents in the vicinity are doing. When several agents know about a single small fire, it would not be useful for all of them to respond. Only a few fire agents would be sufficient to attend to the problem area, preferably the ones nearest by. In order to cope with these issues we have identified two main problems:

1 How do we represent the world and how do we obtain information?

2 How do we progress from one internal state to the next?

Although it would be nice to have a clear division, both problems are heavily interdependent. The internal state is largely determined by the world model and by the input of new observations, whereas proceeding to the next state may require some special adaptation of the world model. It seems impossible to come up with an alternative that does not suffer from these interdependencies so we will use this framework to break the vicious circle. The first problem centers primarily around knowledge and efficiency, whereas the second is about behaviours and reasoning. For both problems we will see that communication between agents is essential.

1.3.1 Knowledge and efficiency

It might seem that the best thing to do is to store all world information for agents to base their actions on. In reality agents need time to compute their actions. If they do not act within their allowed time slots, they may run behind in the simulation. Moreover such an implementation would not be very realistic when compared to the real world, where resources are limited. To ensure maximum efficiency one must think carefully about the data structures that contain the world representation. It must be decided which properties are to have priority in the reasoning process, suppressing unnecessary details in the world model. These aspects will be treated in chapter 3.

1.3.2 Communication

Communication is necessary for both information retrieval and coordination. We aim to achieve coordination by the concept of ‘common knowledge’. The idea is to supply agents working on the same problem with the same information. Consequently algorithms using this information will produce the same results, thereby allowing agents to know about each other’s situation in advance. Using this strategy we can fully utilize the communication lines for the distribution of relevant information only. The communication protocol, explained in chapter 4, has been specifically designed for this purpose.

1.3.3 Behaviors and reasoning

The function of rescue agents is to turn information into action. Selecting the right actions to arrive at the best results poses a problem that can be solved in many different ways. Many other subproblems need to be addressed, like making sure the world is represented in a meaningful way, agents cooperate efficiently, accurate predictions are made, information is gathered and, most of all, the right actions are taken. Though we are primarily interested in making the right decisions, this cannot be done without finding at least a working solution for the other problems. We have chosen the slightly unorthodox method of selecting actions by using game trees. This is a familiar method with well known benefits, as well as drawbacks. Most of the agent's design is specifically intended to reduce these drawbacks. In chapter 5 we will show how we used game trees and evaluate if we have succeeded in reaping the benefits of this experiment.

Chapter 2

RCRSS description

In this chapter we provide a detailed description of the 'RoboCupRescue Simulator System'. Such a description is necessary for the reader to fully understand the role of agents. So let us have a look at the various components that make up the total system.

2.1 Components

Figure 2.1 shows that the components connect with the simulator kernel. The kernel will be described later in this chapter in section 2.3. The components establish connections by means of sockets, according to the UDP protocol[1].

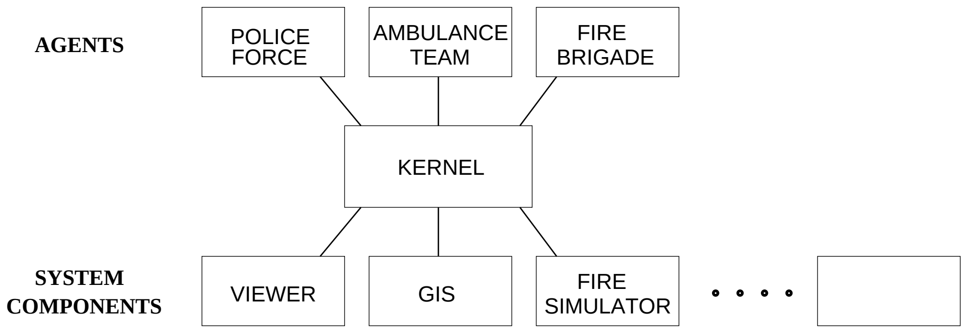


Figure 2.1: *Components of the RCRSS linking up with the kernel using sockets.*

Using sockets for communication enables components to run as separate processes. However, it is also possible to merge several agents into a single

[1]UDP favors speed above reliability: there is no handshake procedure and there is no congestion control.

component. In this way they can share the same communication socket[2], but are denied any additional memory sharing.

2.1.1 System components

The system components, along with the kernel, constitute the actual simulator system. The following list provides an overview of the system components:

- The viewer: displays the simulated map on screen, and shows what agents are doing.
- The gis (geographical information system): provides the initial world status.
- The simulators: implement transitions from one world state to another.
    - Fire simulator: scatters seats of fire and causes fires to spread.
    - Traffic simulator: manages traffic on the map by moving humanoids.
    - Collapse simulator: causes buildings to collapse.
    - Blockage simulator: causes debris to block roads.

The viewer shows the map and displays movements and actions of agents. A screenshot of the standard viewer is depicted in chapter 1. In addition to the viewer supplied with the kernel package, another viewer component was created to facilitate the development of our agents. A detailed description is provided in appendix B.

The gis component is responsible for providing the initial world status. When the system starts it is accessed by the kernel to create an internal representation of the world in a datastructure called ‘object pool’. When activated, the simulators cause changes to it (see section 2.3). It is possible to disable simulators, simply by not starting the appropriate processes. This makes it easy to isolate particular parts of the simulation without any undesirable side effects.

2.2 Simulators

In the following sections we will provide additional details for each simulator as and when appropriate. For more details about the low level kernel interface and the message protocols we refer to the agent development manual written by T. Morimoto [19].

[2]Actually this is a feature of the ADK [9] which was used by the authors of this thesis.

2.2.1 The fire simulator

The fire simulator controls the incendiary properties of all buildings. At the start of the simulation a configuration file (gisini.txt) determines the initial situation on the map. Among the various settings it controls in which buildings fires break out. At startup the gis sets these buildings to the *ignited* stage. From then on the fire simulator controls the spreading of the fire. During each cycle it considers all the buildings that have their ‘ignite flag’ up and determines when these should move to the next stage. A building can be in any of the six stages shown in table 2.1.

Stage	Value	Ignitable	Extinguishable
Not burning	0	*yes*	*no*
Ignited	1	*no*	*yes*
Burning	2	*no*	*yes*
Smouldering	3	*no*	*yes*
Extinguished	5, 6	*no*	*no*
Burned	7	*no*	*no*

Table 2.1: *Fire stages a building be in.*

The first stage *ignited* represents a small fire in the building. From then on the fire will grow in intensity, consuming more of the building as it spreads. As soon as one third of the building has been consumed, it becomes a *burning* building. When two thirds of the building has been consumed, the building becomes *smouldering.* In a previous version of the fire simulator, the stages *burning* and *smouldering* played a part in determining if the fire was big enough to spread to other buildings. In the present version this seems to be determined by the fire extent only, a property which is not discernable to the agent system. When there is nothing left to feed the flames, the building is *burned* and the flames stop.

If the fire in a building is big enough, it starts spreading to nearby buildings. The fire simulator puts the maximum range at which this can occur at 20 metres. Fires cannot ‘jump’ over buildings and so can only spread to the nearest adjoining premises.

The fire simulator also monitors extinguishing actions. This information is passed on by the kernel to the simulator and checked for validity. The amount of water used is added to the total amount of water used on that particular building. Water reduces the size of the fire, slowing the rate at which it consumes the building as well as lowering the risk that it will spread to other buildings. If the intensity of the fire is reduced to zero, the building is considered extinguished and cannot ignite again.

2.2.2 Traffic simulator

The traffic simulator simulates the movements of platoon agents and civilians. It also controls ambulance agents ‘loading’ and ‘unloading’ civilians. In the following description we provide the reader with enough information to understand how agents move. For a detailed description of route plans and load commands we refer to the documentation supplied with Morimoto’s traffic simulator.

In simulating movements, the simulator takes into account road blockages and traffic jams. In either case, platoons and civilians are allowed to travel roads with at least one safe lane left. A lane is considered safe when there is no obstruction impeding forward movement. A minimum distance is to be observed to other platoons and civilians in the same lane. Civilians are assumed to walk and always make way for motorized platoons. Traffic lights are assumed to be disabled during the crisis.

Agents cannot change lanes by themselves. Changing lanes is controlled automatically by the simulator. Therefore agents can only determine whether a road is passable or not. In order to travel agents submit their route plans by AK_MOVE commands which contain lists of object id’s. Figure 2.2 shows an agent moving out of a building (B1), passing roads (R) and nodes (N) and avoiding a blockage by automatically changing lanes. The blocked lanes have been marked by the crosses. In this case the route plan is: {B1,N1,R1,N2,R2,N3,R3,N4,R4}.

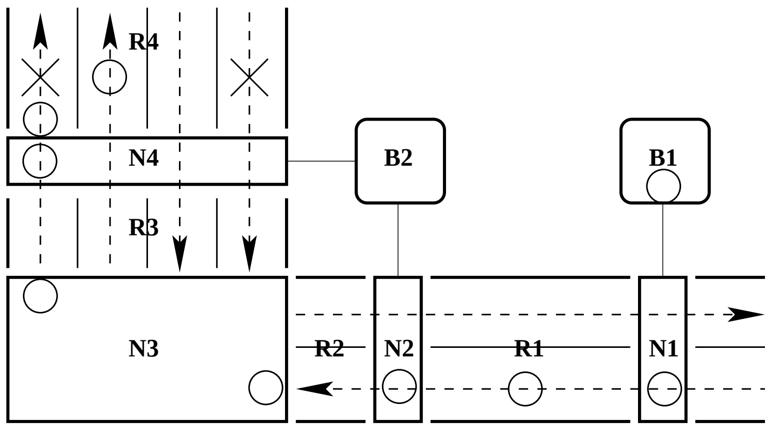


Figure 2.2: *Roads with lanes. The circles represent an agent on the move, bypassing a blockage. The agent moves out of building B1 and travels to road R4.*

The traffic simulator always assumes that a blockage is equally distributed across the road, starting from the outer lanes towards the inner lanes. In the current version of the simulator the severity of a blockage is predetermined at the moment buildings start to collapse, at the beginning of the simulation. It is proposed that later versions of the RCRSS will be able to take into account the impact of aftershocks.

2.2.3 The collapse and blockage simulators

At the start of the simulation, when the earthquake strikes, the collapse and blockage simulator decide to what extent buildings collapse and roads are blocked. In the following description we will not go into the internal operation of the simulators. Instead we provide a survey of the data by which earthquakes are represented.

During the earthquake some areas of the map may sustain more damage than others. To mark the differences, each area is given a numeric value. By connecting areas that are damaged to the same degree, isolines can be projected onto the city map. This is illustrated in Figure 2.3.

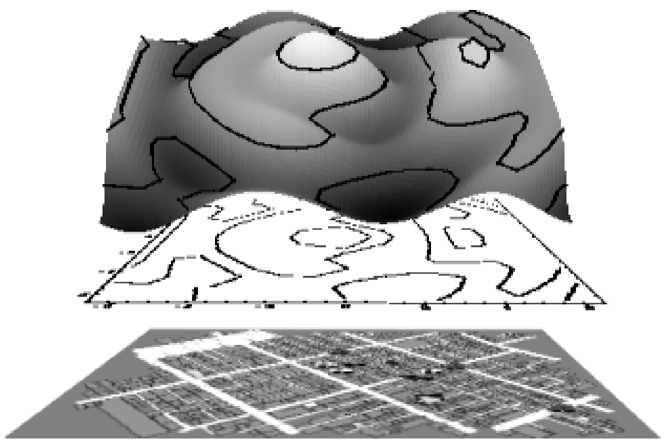


Figure 2.3: *Isolines constructing a 3d-landscape. The isolines connect areas of equal damage values and are projected onto the city map.*

On implementation level the isolines are represented by polygons written to data files. The collapse and blockage simulators operate on individual sets of isolines so each city map comes with two different polygon files. Java tools were used in the creation and scaling of these files. For more information we refer to [28].

2.3 The kernel

The kernel controls all communications between the various components. Since all state transitions within the simulation are the direct result of interaction between components, we will offer a detailed analysis of the kernel simulation loop. Main points of interest are the arranging of messages and the synchronizing of agents with the kernel.

2.3.1 Simulation loop

The simulation loop runs inside the kernel and is the core of the entire RCRSS. Before it is started some initialization steps are necessary. First each component must be connected to the kernel by means of the sockets. When the geographical information system (gis) has been linked up, it provides world information so the kernel can build its own view of the world. When all agents have been connected and successfully supplied with the city map, the simulation loop starts. Now the following sequence of tasks is executed during each simulation cycle. They illustrate how agents can manipulate the world during each cycle and how transitions from one world state to the next are implemented:

- part1:
  - receiveMessages(agent socket)

    SAY/TELL/ACT messages arrive at the kernel
  - sendToSimulators(public socket)

    pass ACT messages to the simulators: they provide updates.
- part2:
  - receiveMessages(simulator socket)

    Receive simulator updates, merge them with the kernel world view.
  - sendToAgents(agent socket)

    Send sensory information (SENSE) to the agents.
  - sendUpdate(gis/simulator socket)

    Send the simulator updates to the gis and other simulators.

Agents can only communicate with each other through the kernel. This is done by means of SAY and TELL messages (further explained in chapter 4), which are relayed as soon as they arrive during part 1 of the simulation loop. Travelling from one point to another or manipulating world objects is a different matter. For these purposes ACT messages are used. Their handling involves the simulators and both parts of the simulation loop. These parts are executed in an alternating sequence. Before each part can be entered at least half a second must pass. This allows both agents and simulators to respond within the allotted timeslot. So ideally one simulation cycle equals one second of real world time.

In part 1 of the simulation loop ACT messages are received and passed on to the simulators. In part 2 these simulators respond by providing updates to the kernel object pool, which is a collection of world objects. For instance the traffic simulator responds to an ACT_MOVE command by updating the position attribute of the matching agent object. The kernel then informs the issuing agent on its new position by sending a SENSE message. This message also contains information about the objects that agents perceive (more on this in appendix A).

In reality, agents are not the only entities that can change the world. The non-moving objects have their own dynamics and their own events. For instance, a collapsing building can cause a new road blockage, which has its implications for the traffic simulator. At the kernel level this translates into state transitions which depend on each other. This is implemented by supplying each simulator with the updates of the other simulators (sendUpdate).

2.3.2 Message arrangement

The simulation loop just described works with many messages, originating from different components. Because all components run as separate processes it is difficult to establish the order in which these messages arrive at the kernel. Therefore the behaviour of the simulation system is not deterministic. The message arrangement within a single communication line however is preserved by the RCRSS LONGUDP layer. The layer ensures that a message covering more than one UDP packet arrives correctly[3]. It operates by collecting and rearranging message packets according to their transmission number in order to rebuild the message. For two messages to arrive in the wrong order, it is necessary that the last packet of the first message arrives after the last packet of the second message, which is very unlikely. We will assume that messages which are exchanged between components are not rearranged, and exploit this property in the communication protocol.

2.3.3 Cycle time

As long as there are messages coming in the simulation loop keeps handling them. This means the loop can only advance one cycle when all messages have been processed. Though it may seem possible to overload the kernel, the risk is reduced by the limited number of messages per cycle each

[3]Since the UDP protocol does not guarantee delivery of packets, messages may become lost. Fortunately this almost never occurs in the RCRSS.

component can generate. The actual cycle time depends on how many messages arrive during each part of the simulation loop and how fast they are processed. Generally speaking, messages can be dealt with within the half second limit mentioned earlier, but naturally this also depends on the computer system deployed. We will assume that the kernel manages to deal with all messages within the available timeslot, so the time lapsed per cycle is approximately one second.

2.4 Agent synchronisation

It is important to synchronize agents with the kernel, to ensure messages are not left unprocessed. If agents want to attain optimum performance, we need to know the frequency with which new observations become available and how fast agents should respond in order to keep up with the kernel. We would also like to know what the effects are of agents responding too fast. Both issues have to do with how we synchronize our agents. So we will first explain how we have accomplished this, it being clear that without synchronization, problems would arise.

2.4.1 Processing SENSE messages

For each repetition of the simulation loop, agents are supplied with a SENSE message (see section 2.3.1). We will use this message to synchronize our agents. The following list summarizes what an agent does when a SENSE message arrives.

- Receive and process SENSE message.
- Produce ACT/SAY/TELL messages (think & communicate)
- Send() all messages back to the kernel.
- Receive and process HEAR messages (SAYs/TELLs from other agents)

When a SENSE message arrives, the agent updates its own world model, prior to deciding what action to undertake and what to communicate. The SENSE message is programmed to be dealt with as soon as the send() command has been given. Figure 2.4 shows the time lapses in the communications between an agent and the kernel in accordance to the sequence of tasks above. As can be seen the agent is synchronized with the kernel. SENSE and HEAR messages are dealt with in time, allowing the agent to communicate its actions before the kernel simulation loop reaches the deadline for that particular cycle.

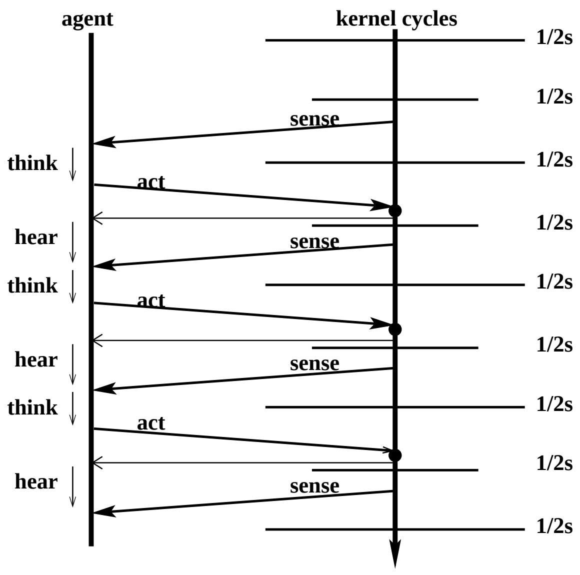


Figure 2.4: *Synchronization of an agent with kernel. Cycles progress along the kernel cycles line. Each horizontal line marks the lapse of at least half a second. The bullets mark the deadlines before which ACT, SAY or TELL messages must be dealt with.*

2.4.2 Time requirements

In the previous section we have shown how agents are synchronized. Evidently this mechanism only works as long as agents can process messages in time. In this section we will show what happens if this is not the case. Figure 2.5 illustrates what happens if the think and hear sequence takes up too much time. When the agent is ready to send its intended actions to the kernel, the deadline has already passed. Therefore messages arrive one deadline later. This effect accumulates with each cycle, since we are assuming think and hear times to be longer than the SENSE message time interval. This causes SENSE messages to arrive more frequently than agents can process them. Note that the last SENSE message in the figure arrives as the previous SENSE message is still being processed. When the simulation loop terminates, agents are still dealing with SENSE messages in the queue. Obviously this is undesirable, because the actions submitted do no longer contribute to the simulation. Therefore we have set an upper limit to the think and hear times, with a view to achieve proper synchronization:

$$T_{think} + T_{hear} < T_{sense\ interval}$$

As shown in figure 2.4 there may exist gaps between the think and hear times: after all, agents can only become aware of messages after these have arrived. This moment is indicated by the horizontal arrow pointing from the kernel to the agent. Please note the additional conditions which pertain to this figure: $T_{think} < \frac{1}{2}s$, $T_{hear} < \frac{1}{2}s$.

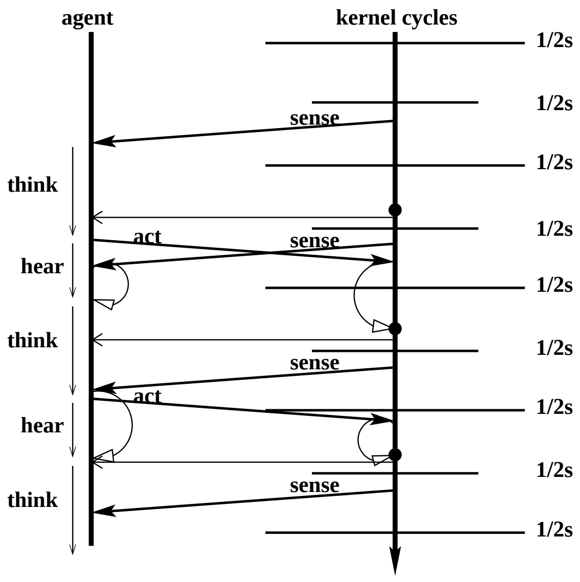


Figure 2.5: *Agent out of sync. Cycles increase along the kernel cycles line. Each horizontal line marks the lapse of at least half a second. The bullets mark the deadlines before which ACT, SAY and TELL messages must be dealt with. The arcs show the delays and the actual times at which messages are processed.*

We conclude this section by noting that it would be possible to increase the cycle times of the kernel. For example one could choose each cycle to last two seconds instead of one. In that case all other time values mentioned should be scaled accordingly, including those of section 2.3.1.

Chapter 3

World Modelling

If agents are to act efficiently, it is necessary to represent the environment in such a way that there is just enough information available for them to determine how to act. Obviously, having to deal with too many details compromises performance and delays the execution of actions. By the time an agent responds to a certain environmental change the entire situation may already have changed again, possibly rendering the action obsolete. Therefore it is useful to make a distinction between different levels of accuracy within the world representation, thus enabling the reasoning process to quickly determine the best action to execute.

3.1 Sectors

Agents work with many world objects collected within one huge object list. These objects are structured in the sense that each one has a geographic location on the simulation map. It would be a waste to ignore this structure, as agents would then have to go through the entire object list repeatedly. Also, it would be helpful if a structure could be devised to give agents greater understanding of the situation at hand.

It was decided to divide the simulation map in sectors. Sectors were already discussed in an earlier article [14], published prior to this thesis. In this chapter we present additional details and provide a complete description of the sector algorithm.

3.1.1 Advantages of using sectors

Dividing the map into sectors is a useful way of breaking down the total problem into a number of smaller subproblems. This makes the general problem of fighting the effects of the disaster more manageable. Evidently this leads to the question which subproblem to solve first. In chapter 5 we will show that the sector method is more efficient. A breakdown to separate problem areas also helps understanding the overall problem better. One area is concerned with the decision which subproblem to tackle first, the other area is devoted to the actual problem solving.

In addition to reducing the scope of the disaster problem, the use of sectors also generates more ways to describe the problem area more efficiently. One way of improving the representation of the map is by precomputing paths between world objects. In section 3.3 it will be explained how sectors can be useful in this process.

The use of sectors makes it possible to assess risks on a per sector basis. This helps to identify areas where agents should focus their attention on. In section 3.5 we will discuss the factors that contribute to the risk assessment of sectors.

3.2 The sector algorithm

It is important to realize that the sector division algorithm we are about to describe is generic and applicable to different city maps. So suppose a new city map were introduced, no adjustments of the algorithm would be necessary. When we explain how it works it will become clear that the algorithm used is uncomplicated and powerful.

A sector constitutes a piece of the map as shown in figure 3.1. In this

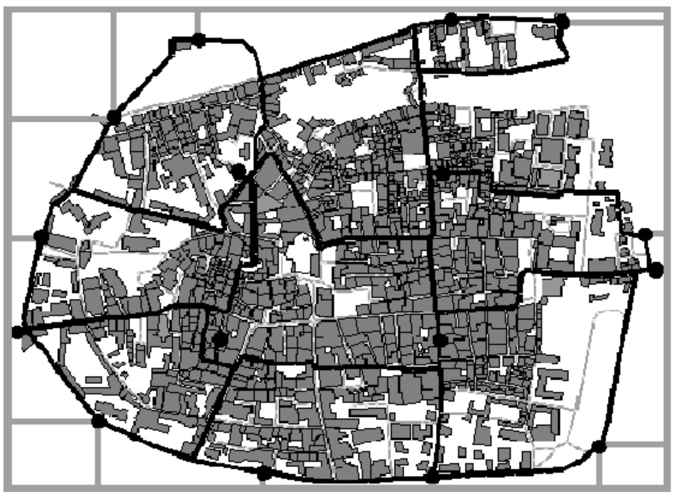


Figure 3.1: *The Foligno map, divided into* 5 × 5 *sectors.*

example the Foligno map has been divided into 5 × 5 sectors. The black lines denote the roads on the map. The black dots represent the nodes used to connect roads. The use of sectors makes it possible to structure world objects according to their actual location. An object is assigned to a sector whenever it lies within the boundaries of that sector.

3.2.1 Gridpoints

As described earlier, each agent is fitted with an object list representing the simulated map. The first step of the sector algorithm is to locate the node objects with the smallest and the largest coordinates, horizontally as well as vertically. By subtracting these coordinates a rectangle can be defined which either lies within, or coincides exactly with the rectangle containing the entire map. The lines of the inner rectangle are now divided into segments of equal length:

$$length_x = \frac{max_x - min_x}{sector_x - 2},\ length_y = \frac{max_y - min_y}{sector_y - 2}$$

This enables the creation of a grid as illustrated in figure 3.2. The variables $sector_x$ and $sector_y$ are commandline options defining the size of the grid. In the example both were set at five. For reasons which will become clear soon, we reserve some sectors by subtracting the value 2 in each direction.

Now comes the key part of the algorithm: we try to match the grid points we have obtained with the node objects on the map in the best possible way. This is accomplished by ascertaining the node object with the smallest euclidian distance for each grid point.

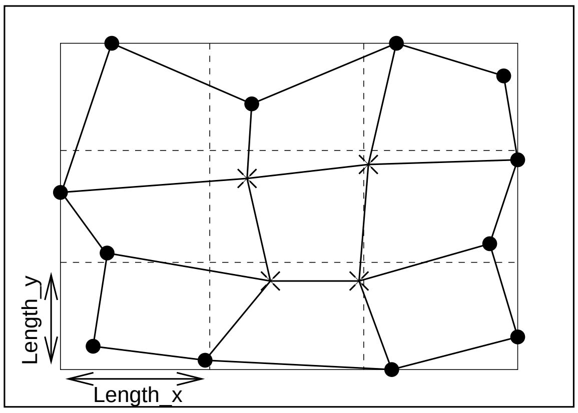


Figure 3.2: *The matching of gridpoints with nodes on the map. The crosses represent inner grid points, bullits represent outer grid points.*

The outer grid points are dealt with in a slightly different manner. The node objects which lie along the grid border are given extra priority by changing the distance function. This is illustrated by figure 3.3. The figure also displays a few gridpoint paths, represented by arrows. Their exact meaning will be explained in the next section.

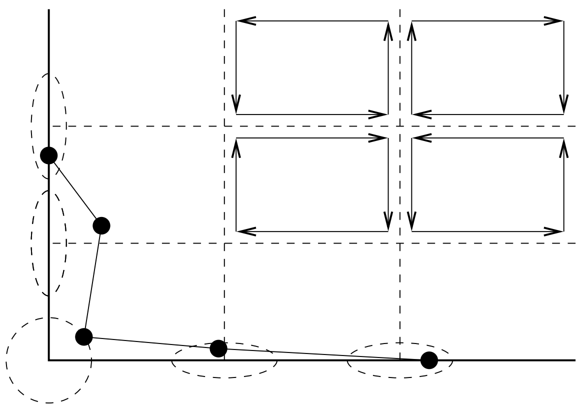


Figure 3.3: *The outer grid points with different distance measures. When looking at the distance measure of the vertical border, coordinate changes in the horizontal direction are attributed greater values. Consequently, points that have the same distance are on the ellipses. The arrows indicate the directions in which paths are found.*

3.2.2 Gridpoint paths

As figure 3.1 illustrates we have now located nodes represented by black dots. The next step is to link up these gridpoints by means of the roads, thereby creating sectors. To do this, we use the path planning algorithm supplied with the ADK [9]. The algorithm helps us find short paths between two objects. To do so quickly, it uses a heuristic function that speeds up the process of finding suitable nodes and roads that make up a path. For our purpose, we supplied the algorithm with a dummy heuristic function in order to find the shortest paths possible. The arrows in figure 3.3 show the directions of the paths that are found. By finding paths in these directions we can make sure each sector joins up perfectly with its adjoining sectors. Code fragment 3.1 shows how this is achieved.

3.2.3 Object assignment

The final step of the algorithm is to assign objects to the right sectors by the application of some elementary geometry. The idea is to view the sectors as if they were polygons and the objects as if they were merely vertices. The polygons are created by concatenating the gridpoint paths surrounding each sector. The problem of object assignment has now been reduced to determining whether a point lies within a closed polygon or not. Several solutions to this problem exist, like those presented by O'Rourke [20] and by Sunday [27]. We opted for a modified version of Franklins [15] algorithm, programmed in C, and based on the *jordan curve theorem.* We quote:

```
createSectors(int sectors_x,int sectors_y) {
  // m2 -- m3
  //  |    |
  // m0 -- m1
  Nodelist path; Node m[4]; // sector markers
  for(int j=1;j <= sectors_y-2;j++) {
    for(int i=1;i <= sectors_x-2;i++) {
      int x1 = i-1 * length_x; int x2 = x1 + length_x;
      int y1 = j-1 * length_y; int y2 = y1 + length_y;
      findMarkers(m,x1,x2,y1,y2); // find node points

      // find node paths, directions depend on the sector
      if((i+j) % 2)
        path = find_path(m[1],m[0]) + find_path(m[0],m[2]) +
               find_path(m[2],m[3]) + find_path(m[3],m[1]);
      else
        path = find_path(m[0],m[1]) + find_path(m[1],m[3]) +
               find_path(m[3],m[2]) + find_path(m[2],m[0]);
      Sector s = new Sector(path,m));
    }
  }
}
```

Table 3.1: *Code fragment for dividing the city map into sectors.*

"I run a semi-infinite ray vertically up from the test point, and count how many edges it crosses. At each crossing, the ray switches between inside and outside."

3.2.4 Loose ends

So far we have not explained how the sector algorithm deals with the sectors along the borders of the map. When looking again at figure 3.1 it is easy to see that the central area has been divided into 3×3 sectors. That leaves us with some objects that could not be assigned since they are not contained by any sector. The top right sector of the figure clearly demonstrates this. Here the gridpoint paths partly overlap simply because there is no shorter road to travel by. As a result many objects cannot be assigned to the sector. This is solved by defining a rectangle containing the entire city map. The grey color in the figure illustrates this. Furthermore, the figure depicts virtual paths in grey color which cannot be used to travel by. These modifications result in two additional sectors for each grid row and column. Together with the corner areas this adds up to 5×5 sectors. Now each object can be assigned to a sector.

3.3 Route planning

Agents can perform either move actions to change their position or a rescue action specific to their type, which will only work if the target of that action is within range. These move actions are communicated to the kernel by providing a list of non moving objects, roads nodes and even buildings, over which an agent wants to travel. We will call such a list a *Route*. The kernel passes the routes of all agents to the traffic simulator, which is described in greater detail in section 2.2. It is up to the agent system to figure out the fastest route to get to a desired position. Finding the fastest route includes avoiding roads which are known or suspected to be blocked by debris.

Finding an optimal route is a problem with high computational complexity and even the near-optimal routes that the A^*-algorithm in the ADK provides, take too long to compute when traveling large distances. Because our agent system not only generates routes that are going to be traversed, but also checks routes to a goal as part of estimating the attainability of this goal, many routes must be generated during a cycle. To make it possible to check a large number of routes we compute all of the routes between all nodes in a map during the startup period of the simulator. Since each object has directly adjacent nodes we can find routes for all objects of the map.

3.3.1 Reducing complexity

There are a large number of nodes in a typical RoboCupRescue map (See table 1.1). If we have $\mathbb{N}$ nodes we would have $\mathbb{N}^2$ routes between them. Because roads can be one way streets it is not correct to assume that every route between two nodes can be reversed to go back from the end node to the starting node, so the number can not be halved. Even if we could use this to reduce the number of routes, it would still be too large to be computed in the time allowed to play a competition game. The sectors we divided our map in can help us reduce this number to more reasonable proportions.

Suppose we precompute the set of routes within a sector. Then we do not have all possible routes in the map, but by stringing together ‘partial routes’ (see the next section), we can still travel to all locations of the map. This allows us to obtain routes significantly faster than having to find them from scratch. With s sectors each with $\mathbb{N}_i$ nodes the number of partial routes would be as in equation 3.1.

$$\sum_{i=1}^{s} \mathbb{N}_i^2 \tag{3.1}$$

This number is smaller than the total number of possible routes $\mathbb{N}^2$ when at least two sectors contain nodes. To prove this, imagine a sector with the set of all possible paths between its nodes equal to A. Take another sector with a similar defined set B. Then the total set of all possible paths between the sectors nodes will be larger than $A \cup B$ since not only does it contain all nodes in A and B, it also contains paths between the nodes of both sectors. When we do not compute and store the paths between nodes in different sectors we significantly lower computation times and memory requirements.

3.3.2 Construction of paths

The paths within a sector are computed by Dijkstra's algorithm [29], which is a greedy algorithm to find all shortest paths to a set of destinations for a single start node. By applying the algorithm to each node of a sector, with the remaining nodes of the sector as destinations we obtain all shortest paths.

3.3.2.1 Dijkstra's algorithm

In the first stage of Dijkstra's algorithm, three sets $\mathbb{V}, \mathbb{D}, \mathbb{P}$ are defined. $\mathbb{V}$ is the set of visited nodes and initially contains only the start node. The set of destination nodes $\mathbb{D}$ contains all other nodes of the sector and the set of generated paths $\mathbb{P}$ contains the trivial path to go from the start node to itself. During the computation a new path will be added to $\mathbb{P}$ for each shortest path from the start node to any of the destination nodes. To do this efficiently the neighbors of all the nodes in $\mathbb{V}$ which are still in $\mathbb{D}$ are first moved to a temporary set $\mathbb{R}$ of *relaxed* nodes. The candidate node for the end point of the next shortest path is surely to be found in $\mathbb{R}$, because getting to any of the other nodes in $\mathbb{D}$ would mean passing one of the relaxed nodes first. For all nodes in $\mathbb{R}$, the total distance traveled to get to it must be obtained in order to select the shortest path. The distance to a node r in $\mathbb{R}$ can be computed by retrieving the path p already in $\mathbb{P}$ to r its predecessor v in $\mathbb{V}$. The distance of a new path is found by adding the distance of p to the distance from v to r. The shortest of all these paths is selected and added to $\mathbb{P}$. The end node is moved from $\mathbb{R}$ to $\mathbb{V}$, because this node is now visited. In the next cycle all its neighbors that are still in $\mathbb{D}$ are moved to $\mathbb{R}$. Once $\mathbb{D}$ and $\mathbb{R}$ are empty, $\mathbb{V}$ contains all nodes and therefore $\mathbb{P}$ contains all shortest paths between the starting position and all destination nodes.

3.3.2.2 Routes and Highways

The paths between the gridpoints that were used to define a sector, can be connected together to form *highways*. When a route is needed to travel from one sector to another we can use the highways to connect two sector paths together and create a total route. In this case a route between a point a in sector A to a point b in sector B consists of:

1. The sector path from a to the gridpoint A_i closest to B

2. The highway between A_i and the gridpoint B_i closest to A
3. The sector path from B_i to b

3.3.3 Computation times

As mentioned earlier, the number of sectors can be configured at startup time. With more sectors one gets less paths and more highways. Figure 3.4 shows the precomputation times for different sector configurations.

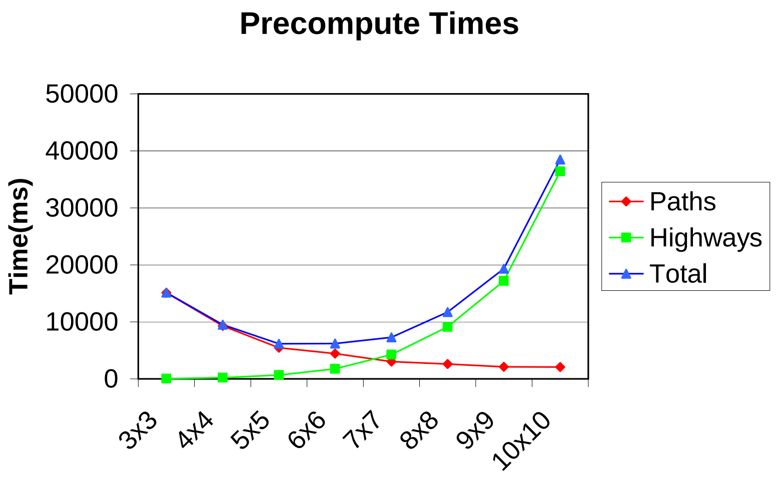


Figure 3.4: *The time required to construct all routes in the Kobe map for different numbers of sectors on a 1.6Ghz system running linux.*

The goal of pre-computing the paths was to reduce computation time of actions by lowering the time that is spent in finding routes. As shown in figure 3.5 below, the *number* of sectors plays a small part in how effective this is, however a significant performance boost is gained by looking up the paths from memory instead of using conventional pathplanning techniques.

So choosing an appropriate number of sectors leads to efficient computation of routes, although this is not our primary concern in choosing the optimal number of sectors. The advantages of sectors we aim for, have an impact on the quality of rescue tasks, which we consider more important than the startup time of the agent system. Also, figure 3.5 indicates that little additional speedup is gained above 4×4 sectors. Therefore the main motivation for choosing the right number of sectors is the detail of the team allocation and summary information rather than tweaking performance. We use a higher number of sector depending on the size and node density of a map. For the well known Kobe map we had 5 by 4 sectors.

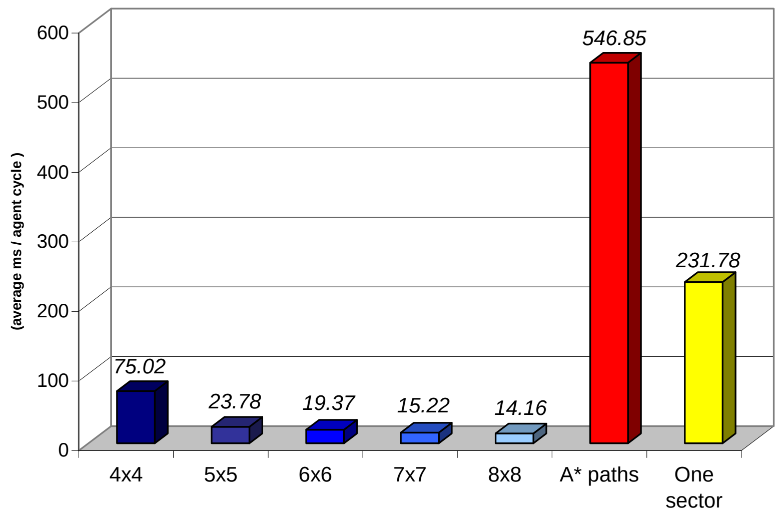


Figure 3.5: *The average time spent by agents computing the best action for different numbers of sectors compared to not using pre-computed paths and not dividing the map in sectors on a 1.6Ghz system.*

3.4 Fire danger

The fire simulator decides each cycle if a fire in a burning building jumps to each of the neighboring buildings within a fixed distance. This is a very detailed simulation, taking into account the material and size of the building, the efforts of fire agents and even the direction of the wind. The fire simulator is one of the most taxing simulators on the hardware that is used to run the simulation. To illustrate: a recent improvement reduced the detail in the simulation to improve the performance after, during the German Open in 2003, it was noticeable that the process running the fire simulator could disrupt the timing of the entire simulator system. Because our approach depends on predicting of the outcome of the simulator system many times to just come up with a single response, we need to do this as fast as possible. It is not possible to call an equivalent of the approach in the fire simulator multiple times or our agent system would require the computational requirements of this simulation multiple times, so we have opted for a less detailed prediction model. By collecting all the static properties, like material, size and distance to neighbors, into one fire risk property we can quickly estimate the importance of getting that building extinguished.

Agents need to know quickly which building is more important to extinguish. This depends on three properties of the building, two of which can be computed before the fire starts. First of all buildings have a size (A) and a material type (m), which determines the amount of burnable material. This amount determines how fast the building goes through the stages of the fire, that are described in section 2.2.1. When a building is burning it can infect its neighbors and therefore buildings with more burnable material are more dangerous than other buildings. This gives us the first part of the priority of a burning building, the *vulnerability* (V) as shown in the two equations below

$$M = \begin{cases} 1.2 & m = concrete \\ 1.2 & m = iron \\ 3.0 & m = wood \end{cases} \tag{3.2}$$

$$V = \frac{1 - \sqrt{(}\frac{1}{1+log(1+A)})}{M} \tag{3.3}$$

The fire can spread from a burning building to other buildings within a certain radius (r). The closer these buildings are, the more likely it is they will catch fire. This process was described in section 2.2.1 as well. Our approximation of this process finds all buildings within the radius, and assumes the chance that they will catch fire is proportional to the percentage that the distance between the position of this building (P) and that of each neighbor (P_n) is of the maximum radius. This is actually the way this chance was handled by an older version of the fire simulator. For all these neighbors we can already estimate their vulnerability and since being close to vulnerable buildings reflects on the risk this building poses. We add all the vulnerabilities of the surrounding buildings, weighted by their distance related chance and get a new factor *Infectivity* (I).

$$I_n = 1 - \frac{|(\overrightarrow{P_n} - \overrightarrow{P})^2|}{r^2} \tag{3.4}$$

$$\sum_{n \in neighbors} S = \frac{1}{1 + log(1 + I_n)} \cdot V_n \tag{3.5}$$

Together these two static values, vulnerability and infectivity, form the static part of the priority of a building as seen by a fire agent. We add them together with two pragmatic importance percentages (α and β) to form the fire *risk* (R). The fire risk of a building is a static value that is computed at the start of the simulation for each building. This value is used in computing the fire priority, on which the extinguish behavior, which we will explain in section 5.3.3, uses to determine its own priority.

$$R = \alpha \cdot V + \beta \cdot S \tag{3.6}$$

The fire risk already contains a hint about how important this building is towards its neighbors, but if a burning building has only burning neighbors then extinguishing it will not accomplish anything in stopping the further spreading of the fire. The risk value reflects how dangerous it would be if none of that building's neighbors were burning. To quickly compute how true this assertion is, so we can find out the actual *danger* (D) this burning building poses during the simulation, we need to count how many of its neighbors are burning. Better yet is to weigh the importance of those buildings, by using their risk value. If we multiply this by the risk value for the building we are considering its score will approach zero as less neighbors are burning.

$$D = R \cdot \frac{\sum_{n \in \text{ neighbors}} \begin{cases} 0 & phase_n \notin \{\text{not burning}\} \\ R_n & phase_n \in \{\text{not burning}\} \end{cases}}{|neighbors|} \tag{3.7}$$

$$P = \begin{cases} 0.0 & phase \in \{\text{not burning, burned,extinguished }\} \\ D & phase \in \{\text{ignited, burning}\} \\ 0.05 & phase \in \{\text{smoldering}\} \end{cases} \tag{3.8}$$

The final step in determining the priority of a building is the consideration whether this building is actually burning itself. If a building is not on fire because it never was ignited, is already extinguished or has burned down then extinguishing it makes no sense, so the priority is zero. If the flames in the building are dying down and the building is smoldering then the building can no longer spread to other buildings and extinguishing it will only save it from collapsing any further, so we assigned a slightly higher score to this. If the building is ignited or burning then it needs to be put out as soon as possible to prevent the spreading of fire and the fire danger that we computed is used as the priority score.

3.5 Summary

The topographical information of the objects in the disaster map doesn't change, which allowed us to create the route plan map and the fire risk values described above. Objects are however also affected to some degree by the disaster which changes throughout the simulation. Most of these disasters effects are concentrated in clusters of objects. Adding the total degree of disaster damage in a sector is one way to describe the location of these clusters. The disaster damage is divided in three percentages that describe for each sector how far it is between perfect condition and total destruction. For each agent type these damage factors have a slightly different meaning in determining the risk that that sector will turn to worse. By giving the highest priority to fighting the danger in the sector with the highest risk the agents are more likely to have the greatest effect on the total outcome of the disaster. Of course there is also attainability to account for, which is a possible focus for future work that will be discussed in chapter 6. These three values for each sector give a global overview and is therefore called the *Summary*. This information is used to globally directed the agents to the correct sector, by the task and team assignment algorithm that we will describe in section 5.3.2.

3.5.1 Fire

During the disaster, buildings are burning and fires spread to other buildings if not extinguished by fire brigades. These buildings are clustered because fire spreads from only a few initial fire points. Adding the total number of burning or ignited buildings together and dividing it by the total number of buildings gives us a very rough percentage of the size of the fire in a sector. It would be possible to provide a more realistic number by using the fire danger of each building, which we described in section 3.4, but for now we have seen no reason to use anything but the percentage of burning buildings regardless of their size or infectivity.

3.5.2 Damage

Buildings collapse due to the earthquake that starts the disaster or because of fires. The collapsed buildings can trap civilians until they are freed by ambulances. The chance that a building is collapsed depends on the distance to the epicenter of the initial earthquake. The sector in which the most buildings are collapsed has the most chance of trapping civilians in them, assuming that the starting position of the civilians is random. It is not random, in fact it is determined by the data files created for the simulation by hand, but since it is not known by the agents where the civilians are until they are found it is best to assume that they are divided equally over the map as there is no way to predict the decisions of the person who designed it. In a real disaster an estimate of the number of inhabitants of each sector could be used to provide additional information of the risk of civilians dying in collapsed buildings. Because this detail is not provided, ambulance agents will simply go to the sector with the most collapsed buildings to look for civilians. The damage percentage is again calculated as a percentage by dividing the number of collapsed buildings by the total number of buildings. Unlike the fire risk, both building collapse and road block are determined by the distance to the epicenter. This makes another variable for attainability more important, namely how easy it is to travel through the sector and how long it takes for the police agents to make all position in the sector reachable. This is not yet considered by our simple sector assignment algorithm.

3.5.3 Roads

Debris of collapsed buildings blocks passage over nearby roads until they are cleared by police agents. The percentage of the total roads that are blocked in a sector has no direct influence on the progress of the disaster. No one dies because of blocked roads and no additional buildings are destroyed by it. It does however hamper the rescue attempts of the agents, making a sector harder to travel in and thus harder to save. No agents are however sent to sectors with a high concentration of blocked roads, solely because the sector needs to be repaired. It could however decide the amount of police agents needed to aid the fire and ambulance units, but in our implementation this number is fixed.

3.5.4 Intel

The last risk factor is a little different as well. It gives a percentage of recency of information. If a building or road has been spotted by an agent during the last cycle its information on fire and roadblock respectively is recent. If it has never been visited by an agent its information is old, from before the disaster happened during the first three cycles, and therefore unreliable. The percentage is calculated by assigning a score to each building based on the time the buildings was last observed (τ), the time elapsed during the simulation so far (δ) and then averaging it over all buildings (β) as shown in equation 3.9

$$intel = \frac{1}{|\beta|} \cdot \sum_{n \in \beta} 1 - \frac{\tau_n}{\delta} \tag{3.9}$$

by adding the time that the item was last observed divided by the time elapsed during the simulation and dividing that by the total number of items. This risk factor has both a direct effect on the priority by giving police agents a reason to travel to unknown sectors and gather information, but it can also be used to provide a reliability of the other risk factors.

3.6 Keeping the worldmodel up to date

During the simulation the agent will receive observations and communications that notify it about change in the disaster. These changes are kept in the world model. The objectpool that forms the core of the worldmodel is designed to store observations. An observation consists of the id of the object it contains, the time of the observation and a list of changed properties of that object. These properties are stored in the objectpool along with a timestamp of the last observation. The timestamp can be used to evaluate the reliability of the information or to resolve conflicting observations.

The observation of properties is pretty straightforward except for the position of other agents. When an agent sees another agent, it will remember the position of that agent. When the agent moves away from that agent it will no longer receive observations from it, so it will remember that that agent is still there. Of course the agent will move around, but it will not necessarily run into that agent again. If it does, there is no problem and a new position is stored. However when the agent passes by that same stored position and sees the agent it previously encountered is no longer there, it can no longer rely on that information. We solve this by generating a fake observation event that states since the time since the previous encounter at that position, the agent is standing 'nowhere'. This is a very important improvement, because the other position of other agents is crucial in predicting its behavior. If the agent was standing on a road, it means the lane which it occupied is no longer blocked by its presence. If the lane is not labeled as cleared in this way the agent could be waiting for the rest of the simulation, waiting for the phantom of an agent to move.

Another source of information for the worldmodel is communication, further explained in chapter 4. The changes to the world model coming in from the communication module are stored in a separate list that is comparable to the object pool. This list also contains the time of the observation was received, so an agent will know what information is more up to date should conflicts arise. When retrieving information from the world model during the simulation, the information from either the objectpool or the communication buffers will be provided whichever is most recent.

The summary is also part of the world model. It allows the coordination to quickly get a simplified picture of the total disaster. For this it has to be kept up to date throughout the simulation as well. All the disaster risk factors described above in section 3.5 are computed by taking the average severity of every evolving property in each sector. Every cycle that information can be kept up to date by recomputing these averages.

Figure 3.6 gives an overview of the different structures in the worldmodel. The arrows denote information processes in the worldmodel during the simulation. We can see how incoming communication is stored in the communication buffers and outgoing communication comes from both the observations ('See') stored in the objectpool and the contents of the communication buffers. Together this information is used both to update the summary and to provide information to the coordination module.

3.7 Conclusion

When the agent is started it first receives an initial map from the kernel. This is the situation before the disaster takes place. It contains topographical data, which will not change during the situation. All buildings are assumed

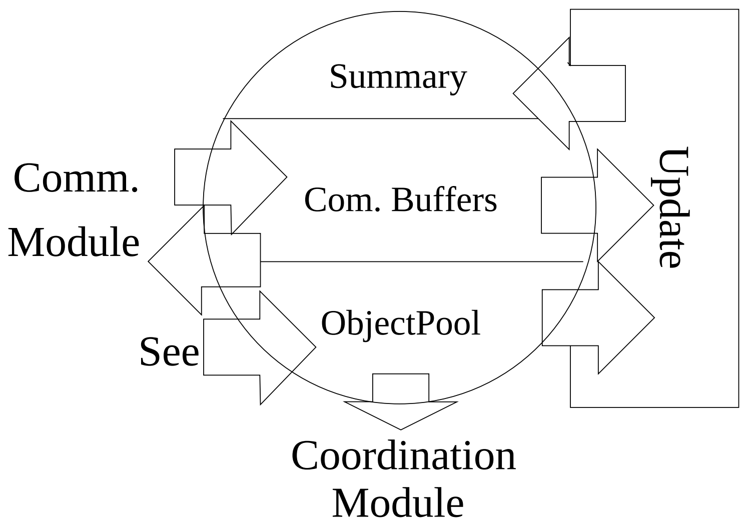


Figure 3.6: *The information flowing in, out and between the various parts of the worldmodel.*

to be in good conditions, none are on fire and all roads are traversable. Before starting up the agent the static supplementary data is computed, which will help in retrieving the static data in the most efficient way. The supplementary data structures are: sectors to quickly retrieve a part of the map, precomputed paths to quickly get a valid route from one place to another and fire risk to quickly predict the spreading of fire and summaries to get a global overview.

Besides providing easy answers to complicated information requests the worldmodel can also give up to date detailed information that is incorporated from two sources: observation and communication. Conflicting inputs are resolved by using a time stamp. The world model provides one complete package of all the information an agent needs to know to execute its communication and coordination, which we will describe in the following two chapters.

Chapter 4

Communication

As their viewing distance is limited, agents are hampered by an incomplete picture of their world. Each simulation cycle generates new observations but agents only perceive a subset of these. Luckily, agents can improve their knowledge by communicating with each other. In section 4.1 we will analyse the communication system and explain why common knowledge within a team of agents is important. It will be clear that it is easy to overflow the system with information, forcing agents to ignore messages. To counter this, we present a communication model [22] in which agents reduce the number of sent messages to a number that the receiver is capable of handling. We shall also introduce an advanced 'information distribution system', which allows agents to share a common view of the world which is but few cycles old. After a detailed description of the actual communication protocol in section 4.2, we shall continue with an evaluation of its performance.

4.1 Analysis

Agents have two kinds of messages at their disposal. An agent can broadcast SAY and TELL messages which will be succesfully transmitted if the receiving party meets certain requirements. The basic analogy to a real world disaster is that SAY messages represent direct oral communication, while TELL messages are broadcast by radio.

Agents can exchange spoken SAY messages across distances of up to 30 metres, regardless of the type of agent involved: ambulance agents can communicate with police agents, etc. In contrast, TELL messages can only be shared among agents of the same type, but distance is irrelevant. Centers can always hear each others TELL messages regardless of the type they belong to. Figure 4.1 shows an example of agents communicating.

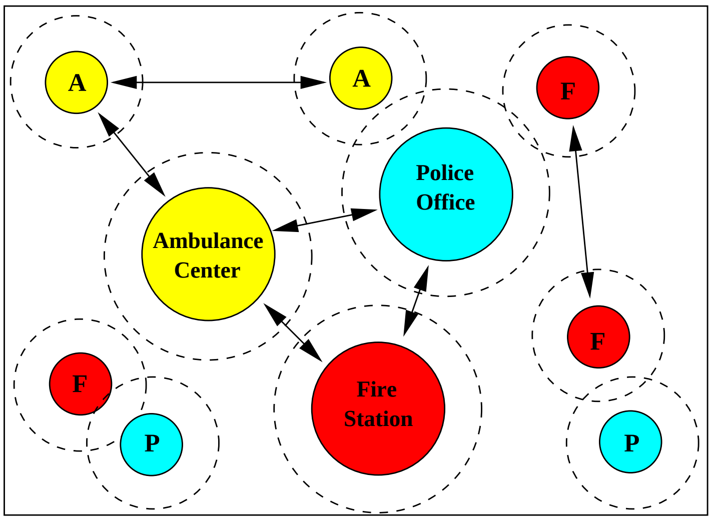


Figure 4.1: *Part of a simulation environment occupied by several agents. The dotted circles around agents show the SAY message range. TELL communication is represented by bidirectional arrows.*

4.1.1 Restrictions

Additional communication restrictions have been introduced to make the simulation more realistic[1]. In a real world crisis situation it would be impossible to keep all agents informed of every detail. To simulate this only a limited number of messages may be sent or received during each simulation cycle. Enforcing these restrictions is the responsibility of the agents themselves. In the case of receiving messages, if too many messages arrive the receiving agent is allowed to make a selection of the messages it wants to hear, completely discarding the other messages. In this process only sender and receiver identities may be accessed. The second restriction is that each message should conform to a prescribed format. The details of both restrictions are given in table 4.2 and table 4.1.

SAY/TELL :	SENDER_ID	MESSAGE	
HEAR :	RECEIVER_ID	SENDER_ID	MESSAGE

Table 4.1: *Message format for agents. The length of the MESSAGE field cannot exceed* 80 *bytes.*

[1]These restrictions vary from year to year. Here we will give the restrictions prevailing in the year 2003.

AGENT CATEGORY	SAY/TELL	HEAR
Platoon agent	4	4
Center agent	$2 * n$	$2 * n$

Table 4.2: *The maximum number of messages agents are allowed to receive during each simulation cycle. Agents discard messages they do not want to hear, using only sender and receiver identities (see Table 4.1).* n *is the number of platoon agents that match the type of the center.*

4.1.2 Communicating observations

The post disaster situation is complex, posing various tasks to various teams of agents. A typical disaster situation will include several fire clusters, a considerable amount of blocked roads and any number of civilians in need of rescue. So several problem areas should be dealt with simultanously [2]. Also some problems are interrelated. For instance, police agents may have to team up to clear roads, while at the same time a fire fighting team may be engaged in extinguishing buildings in that area. The concept of teams assigned to tasks will be further illustrated in section 5.3.2.

In order to make a team operate efficiently it seems convenient to let each team member know about the activities of the other team members. However we wish to point out that the actions of an agent are based solely on the data observations. Suppose we were able to supply each team member with the observations of the other agents. Then it seems obvious that there are no other data available (within the team) that can improve the information value of all observations combined. Compared to communicating activities, it is quite possible that agents extract information on actions of other agents from the combined observations, given that they use a deterministic reasoning algorithm. Indeed it has been shown by Pynadath and Tambe [23] that when communications within a team can be exchanged at no cost, it is best to communicate all observations. Notice however that Pynadath and Tambe say nothing on communications between *teams*. The communication system of the RCRSS does involve communication costs and sharing information within a team does not come easy.

4.1.3 Domain analysis

In the RCRSS we have to cope with communication restrictions. Therefore it is impossible to attain the ideal situation, in which all observations would be shared. As we will see in section 4.1.4, certain concessions will have to be made. But before we can determine what trade-offs are necessary, we need

[2]In contrast, if one would deal with each fire area consecutively, some areas run the risk of total destruction by fire before any remedial action can be undertaken.

to know exactly how much information we are dealing with. In the following analysis we will try to find an average value for the number of messages we expect to transfer.

At the start of the simulation, just before the disaster takes place, each agent has full knowledge of all the objects on the map. From then on agents only sense objects that lie within a distance of 10 metres, except for seats of fire which can be sensed from larger distances (depending on the severity of the fire). So let us focus on the nearby objects first.

4.1.3.1 Nearby objects

Finding the number of nearby objects an agent can sense is relatively simple. It suffices to scan all possible map locations a platoon is likely to visit and to determine for each location the number of objects within a $10m$ radius. By taking the average of these measurements, we obtain an estimate of the number of nearby objects an agent can sense during each simulation cycle. Table 4.3 shows the average numbers for three different city maps.

MAP	NEARBY OBJECTS
Kobe	5.64
Foligno	5.69
Virtual City	4.07

Table 4.3: *The average number of local objects an agent can sense.*

When taking these measurements, we only considered buildings and roads. Node objects can safely be ignored, as they are not affected by the effects of the disaster and are merely used to connect buildings and roads. When taking into account that agents can also sense other moving objects (such as platoons and civilians) once they enter the $10m$ range, it seems safe to conclude that on average 10 local objects can be sensed during each cycle.

4.1.3.2 Distant fire points

Fires can be seen from larger distances. To determine the amount of information resulting from this category of observations we must fall back on analysing the simulation during run-time. To be on the safe side, we chose to run the simulation without any fires being extinguished. Then, for each cycle we determined the number of objects that had their incendiary properties changed. Again, measurements were taken of the maps of Kobe, Foligno and Virtual City. The results of the Virtual City map have been depicted in figure 4.2. The results of the other maps being quite similar, these were omitted.

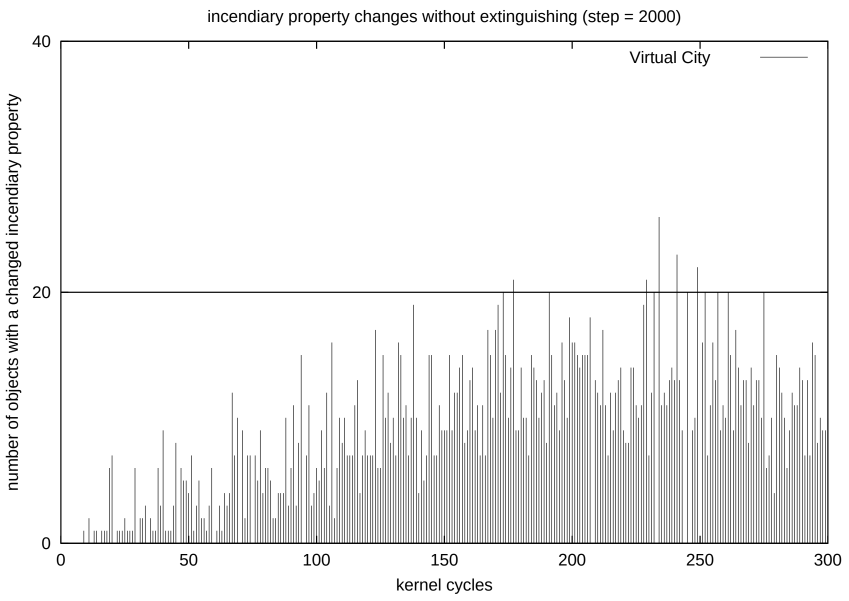


Figure 4.2: *The number of objects showing change in incendiary properties, measured during each simulation cycle in the kernel. We assume four initial seats of fire.*

As there are only five peaks above the $y = 20$ line out of a total of 300 measurements, it follows that approximately 98 percent of all measurements are below the $y = 20$ line. Note that these measurements correspond to a situation where the fire agents remain passive. Moreover agents have a limited view whereas the kernel has full knowledge about all objects in the map. Therefore it seems fair to assume that during a normal simulation an agent can sense about 20 distant objects with changing incendiary properties per cycle.

4.1.3.3 Average number of messages

Now let us try to translate the established results into messages. Only when object properties have changed since the last observation will the altered values be communicated. For nearby objects we shall assume that no more than two relevant properties will change per simulation cycle. Buildings only have one dynamic property (inflammability); roads have two dynamic properties (degree of blockage and repair cost). For agents and civilians it is assumed that the only relevant properties are position and physical state.

Thus for each simulation cycle we have $(2 \cdot 10) + 20 = 40$ potentially changing properties worth communicating. When encoded into 32 bits (including the time value of when the property change occured) this translates into exactly two messages (see table 4.1).

4.1.4 Approaches

Now that we have determined how much information we have to transfer, we can return to where we left off at section 4.1.2 and select a suitable communication method. There are three different approaches to choose from:

1. Distribute observations in an uncoordinated way, thus fully utilizing the send and receive capacities (bandwidth) of the agents.

2. Distribute observations in a coordinated way at the expense of losing bandwidth (due to alignment problems).

3. Communicate something different from observations.

In the first approach agents simply start to broadcast SAY or TELL messages at the moment they sense something. Using SAY messages seems rather pointless as in most cases the distance to be bridged exceeds the preset 30 metres limit, so messages will not reach their destination. When using TELL messages there is the obvious restriction that these are confined to agents of the same type only. Moreover, as we saw in the previous section, each agent would then have to send at least two messages. Since platoons can only receive four messages per cycle according to table 4.2, they would not be able to receive all the messages if they have more than two colleagues. It seems clear that some form of coordination of the communication method is called for.

We chose a mixture of the approaches mentioned. In our communication system, distribution of *information* in a coordinated way has been fully integrated. Yet there is still room for the platoon agents to engage in uncoordinated communication, by using SAY and TELL messages. Coordination has been realized by making platoon agents communicate the two messages to their centers instead of to each other. Because the centers are allowed to receive more than two messages, they can combine them and inform each other before reporting back to their platoon agents. Using the centers as intermediate communication nodes leads to the following advantages:

- Teams can benefit from some form of common knowledge, even when members belong to different types.

- The centers can benefit from some form of common knowledge, making it easier for them to control teams.

- There is a central place were all information is available. This means available knowledge can be shared among different teams.

The benefit of common knowledge should not be underrated. It allows team members to attain a concerted effort. It is possible to predict the behaviour of the other members, based on shared information. Also shared knowledge among teams simplifies the dynamic adjusting of team compositions. But coordination of communication carries a price tag. In section 4.3 we will see that it takes several simulation cycles for information to be fully shared, which means that the information received may no longer be up to date.

4.2 The communication protocol

In the previous sections we motivated why common knowledge is a condition for efficient teamwork. Now suppose a platoon agent has information it wants to share with other agents. The platoon should then broadcast a TELL message to its center which takes care of further distribution. In the distribution process only TELL messages are used as their reception is not restricted by any distance limits[3]. We refer to this kind of transfer as coordinated communication. But what if platoons want to communicate one on one to resolve local conflicts in the problem area? They can then broadcast either a SAY or a TELL message in the uncoordinated communication mode of the protocol.

4.2.1 Coordinated communication

In order to achieve common knowledge it is necessary that the distribution process is synchronized. When a platoon agent sends messages intended for its center, a new distribution sequence is started. At this point the platoon should wait for its center to report back before starting a new sequence. During the sequence the centers collect messages, communicate with each other and then report back to their platoons. In this process it is necessary that the senders and receivers participating in these transfers are properly aligned. If they are not, a receiver may be forced to discard some messages since it may already have exceeded its reception limit for that particular simulation cycle. Therefore it is necessary that the sender somehow knows the status of the receiving side before starting any transfer.

4.2.1.1 Synchronisation considerations

Suppose one could guarantee that all agents were properly synchronized: i.e. all agents are in the same simulation cycle at the same time. The sender would then automatically have an idea of what the receiving side is doing and could decide exactly when to start broadcasting. Remember that in

[3]We allow for TELL messages to be delayed, as is often the case in crisis situations.

section 2.4 we showed that when an agent responds to its SENSE messages within the allowed time intervals, it is synchronized to the kernel. If all agents synchronize in this way, they would also be synchronized to each other.

Due to the complexity of the environment however, we were unable to grant agents sufficient thinking time. As will be explained in chapter 5, the reasoning process claims a considerable amount of computation time. In chapter 3 we showed how the map was divided into smaller sectors to reduce the complexity of the environment and of path planning. Still, these improvements were not sufficient for the agents to synchronize properly.

Instead it might have been possible to use a separate listening thread responding to SENSE messages within the available time, but we chose not to use the thread approach since it would result in adverse race conditions[4]. Neither did we want to make assumptions about the run times of agents and the kernel, or the order in which SENSE messages are dispatched by the kernel. Moreover the ADK [9] was not designed with threads in mind.

Eventually it was decided to incorporate a synchronization method in the communication protocol itself. The idea is that each agent keeps track of the status of the agents it can hear by counting the number of received messages. It is safe to count messages continiously, as this does not involve the MESSAGE field as specified in table 4.1. In this way the sender can always accurately assess the status of the receiving side before broadcasting. For the eventuality that the receiving side is still in the middle of an unfinished transfer, 'dummy' READY messages were added to the protocol. This allows the receiver to indicate its readiness to accept new messages. As soon as one simulation cycle has passed since the last transfer, the receiver is considered to be ready for the next transfer.

It could be argued that there are other more elegant synchronisation protocols imaginable. But these protocols probably involve explicit use of the MESSAGE field. But when messages are discarded by the receiver during the selection process, this field cannot be accessed! This means that the protocol would also have to be able to cope with lost messages, causing additional overheads.

4.2.2 Implementation

We will now discuss the communication protocol in more detail. In the protocol, platoon agents exchange coordinated and uncoordinated communications in an alternating way, whereas the centers are capable of coordinated communication only. We define communication cycles with reference to the underlying simulation cycles. Each communication cycle the agents move through various stages. Figure 4.3 and table 4.4 illustrate how agents go

[4]A situation in which a program variable is accessed simultanously, leading to inconsistent behaviour

through consecutive stages, which enables them to know what to communicate.

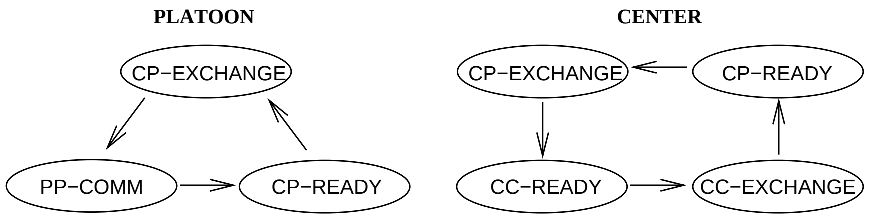


Figure 4.3: *The stages agents go through during one communication cycle.*

STAGE	DIR	N	COMMUNICATION
CP-EXC	P ⇒ C	2	Platoons send information they wish to share with their centers.
CP-EXC	C ⇒ P	4	Centers supply their platoons with shared information from others.
PP-COMM	P ⇒ P	4	Platoons say/tell communication, uncoordinated.
CC-READY	C ⇒ C	1	Centers inform each other when ready to enter stage CC-EXC.
PP-COMM	P ⇒ P	4	Platoons say/tell communication, uncoordinated.
CC-EXC	C ⇒ C	4	Centers distribute information among each other.
CP-READY	C ⇒ P	1	Platoons inform their centers when ready to enter stage CP-EXC.
CP-READY	P ⇒ C	1	Centers inform their platoons when ready to enter stage CP-EXC.

Table 4.4: *The stages platoon agents and centers go through during one communication cycle. Each box in the table represents a communication stage of platoon agents and centers. When an agent progresses to the next stage, consult the box underneath to follow the process in consecutive order. The DIR field specifies the direction of communication. N denotes how many messages the receiving party should expect.*

4.2.2.1 Alignment

Even though table 4.4 might suggest otherwise, platoon and center agents may not reside in the specified stages at the same time. It could just as easily be the case that platoons reside in the PP-COMM stage, when their centers have already re-entered the CP-EXC stage. As mentioned earlier, agents keep track of each other's status by counting messages. As soon as an EXCHANGE stage is entered into, the agents must realign with all participating agents before exchanging information. Upon leaving the EXCHANGE stage an agent verifies that all messages have arrived. This is how the synchronization part of the protocol works: alignment between the platoons and centers occurs in the CP-EXC stage and alignment between the centers occurs in the CC-EXC stage, as illustrated in figure 4.4.

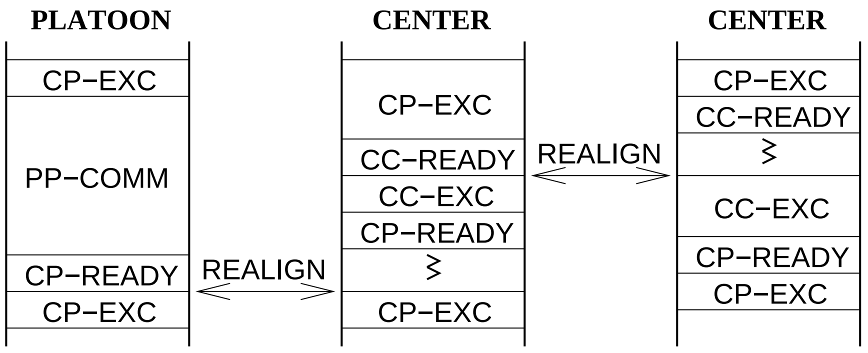


Figure 4.4: *Realignment of a platoon with its center in the CP-EXC stage and realignment of the same center with another center in the CC-EXC stage. In this way agents remain synchronised.*

4.2.2.2 One communication cycle

We will now describe the operation of a single communication cycle. Initially both platoon and center agents are in the CP-EXC stage. In this stage the centers collect messages from their platoon agents and provide them with information from the previous communication cycle. Since both sides are in an EXCHANGE stage they both have to wait for all messages to arrive before being able to proceed to the next stage. When a center has collected all its messages, i.e. has received two messages from each of its agents, it is allowed to leave the CP-EXC stage and to enter the CC-READY stage. In this stage it checks that new messages can be accepted before notifying the other centers by means of a CC-READY message. New messages can be accepted as soon as one *simulation* cycle has passed and all platoon messages have been received. Upon notification, the center immediately proceeds to the CC-EXC stage and waits for the other CC-READY messages before sending any message to the other centers. Upon finishing the exchange operation the center is allowed to enter the CP-READY stage. Again the center checks if new messages can be accepted before notifying its platoon agents. The CP-EXC stage is re-entered, but the center cannot continue until all of its platoon agents have also re-entered the CP-EXC stage. So it would be perfectly possible for some of the platoon agents to remain in the initial CP-EXC stage, because they have not received all center messages yet. Only when the center has verified that each platoon agent has gone through all of the stages of its cycle, a new communication cycle can begin. Likewise it is possible that platoon agents have to wait for their center to cycle through all of its stages.

It is easy to see how the communication restrictions are respected. For instance, the top box of table 4.4 shows that each platoon agent sends up to two messages to its center. Table 4.2 shows that a center is allowed to receive $2n$ messages, n representing the number of platoon agents that belong to it. In this way all platoon messages arrive without complications.

The CC-EXC stage deserves further attention. Since it is not necessarily true that all centers are assigned the same number of platoons, their send and receive capacities may differ. By assuming that a center is assigned at least four platoons, each center becomes capable of accepting at least eight messages per simulation cycle. Since there are two other centers to communicate with, each center can send and receive four messages.

Finally it should be mentioned that agents residing in the PP-COMM stage may sometimes be forced to discard messages because this stage does not allow for alignment. This is the price to be paid for uncoordinated communication.

4.2.2.3 Practical issues

Now that the working of the protocol has been explained it is time to say something about our implementation. We discovered that centers were able to pick up SAY messages from platoon agents. We consider this a flaw in the RCRSS, since centers were intended as building objects. Moreover the RCRSS offers no way of determining whether a SAY message or a TELL message was broadcast. These shortcomings make it impossible to determine if the agents register the correct number of received TELL messages. For now we 'solved' the problem by allowing platoon agents to issue TELL messages only. We suggest a clearer distinction between SAY and TELL messages will be made in future versions of the RCRSS.

4.3 Evaluation

In this section we will see that coordinated communication has its disadvantages too. Before we look into them, we will first determine the maximum number of communication cycles we expect the entire simulation to last. After considering the disadvantages, we will then demonstrate how the distribution system may be employed to convey information on fires, enabling agents to increase their knowledge about fire hazards.

4.3.1 Communication cycles

Looking more closely at table 4.4, one might conclude that it takes about four simulation cycles to complete one communication cycle. But bear in mind that some simulation cycles may not be used for communication purposes due to delays in the reasoning process. This relation becomes clearer when looking at figure 4.5.

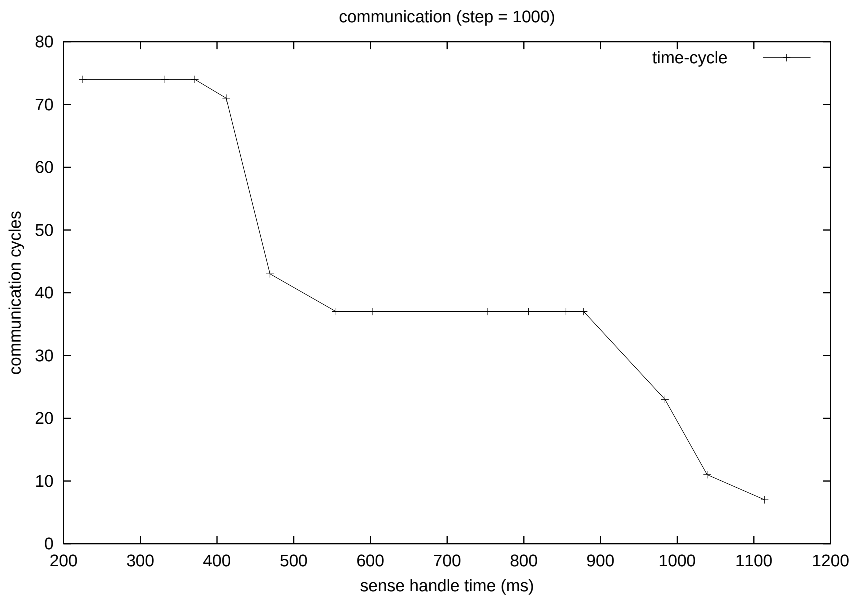


Figure 4.5: *response time versus the number of communication cycles. The crosses represent the actual measurements when the simulation runs* 300 *cycles and the kernel cycle time is set to* 1000*ms.*

Figure 4.5 confirms that at best $\frac{300}{75}$ simulation cycles are required to complete one single communication cycle. Recall from section 2.4 that an agent has to process SENSE messages within $\frac{1000}{2}$ms if ACT messages are to arrive before the deadline set by the current kernel cycle. Indeed the figure shows that the number of communication cycles is reduced to approximately 36 when agents take longer than 400 $\sim$ 500ms of thinking time. The situation becomes even worse if an agent fails to synchronize with the kernel, which will start occurring from 900 $\sim$ 1000ms. We would like to remind readers that the kernel cycle time can always be increased in order to allow agents more thinking time.

4.3.2 Disadvantages

Let us now survey the disadvantages of our chosen method of communication. First of all, when sharing information, distribution takes as long as one communication cycle. Therefore newly shared information always arrives with a certain delay. Also, since one communication cycle takes up multiple simulation cycles, the data of observations accumulate before a new distribution sequence is entered. So when agents wish to share their observations, they run the risk of exceeding the two messages limit[5] which we mentioned in section 4.1.3. Fortunately, it is sufficient to communicate only the most recent changes in object properties. Therefore when any property changes several times during a distribution sequence, only the last change is relevant and must be communicated. This kind of 'information overlap' is likely to occur regularly since agents will not suddenly jump from one end of the map to the other.

Another drawback is that the READY messages used for synchronization take up simulation cycles that could otherwise have been used to transfer information. As can be seen in figure 4.3, platoon agents spend $\frac{1}{3}$ of the time in the READY stages sending READY messages, whereas the centers actually spend half of the available time sending them. Moreover it follows from table 4.4 that READY messages add to the number of simulation cycles that are required to complete one communication cycle.

Thirdly, according to table 4.2, in some stages centers send and receive far less messages than they are capable of. The same is true for platoons in the CP-EXC stage, each one of them sending only two messages at most. The guarantee that centers can receive all platoon messages clearly is given at the expense of bandwidth lost. Still the situation is not as bad as it might seem. Utilising 'outgoing bandwidth' is less important than utilising 'incoming bandwidth' to the full. When there is nothing more to receive it is of no use to improve sending capacity. We have illustrated how platoons and centers fully utilize their incoming bandwidths in the CP-EXC stage.

Finally, the protocol assumes that TELL messages always arrive, though are sometimes delayed. This sounds worse than it really is, since in a real life setup it is not inconceivable that a proper network layer has been installed to take care of retransmitting failed messages transparently to the protocol. Still a platoon agent may not be capable of sending any TELL messages at all, due to damaged equipment or weak radio signals. In which case its center may decide that if the platoon does not answer within a certain time interval, it is permanently banned from the distribution sequence. The platoon however will always be capable of exchanging uncoordinated communications.

[5]Actually in the year 2004 the rules became more flexible when the MESSAGE field was increased from 80 bytes to 256, which reduces the information overflow problem considerably.

4.3.3 Increased knowledge

The distribution system we presented enables agents to increase their knowledge. But what is the performance of this communication method? In order to evaluate the performance, an attempt was made to communicate information on fires. Figure 4.6 illustrates how a center increased its knowledge by coordinated communication.

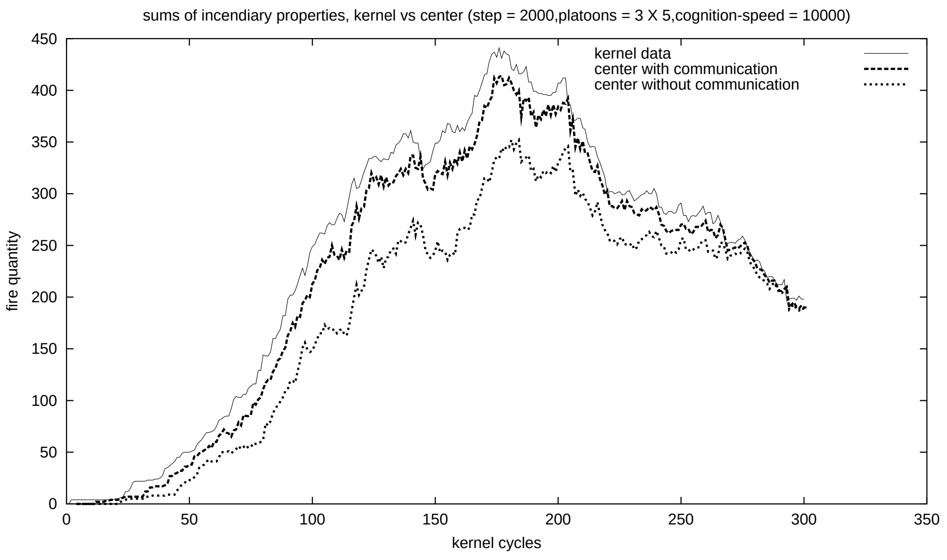


Figure 4.6: *The total sum of incendiary properties of objects. When an object has been totally destroyed by fire, it is excluded from the count. Incendiary properties range from 0-3.*

The curves in the figure represent the sum total of all the incendiary properties of the objects on fire. The topmost curve of the figure represents the measurements of the kernel which is aware of the actual situation. The other two curves represent the measurements of centers operating either with or without communication. It is obvious that the benefits of communicating are substantial.

Agents retain their increased knowledge on top of their standard knowledge. This is accomplished by equiping them with a 'burning objects' list containing copies of objects an agent has more information about. During each simulation cycle, the list is scanned to determine the sum of the incendiary properties. Although figure 4.6 does not reveal which objects contributed to the sum, it is still possible to verify the results. One would expect that the curves measured by the center follow the 'movements' of the kernel curve more closely as knowledge increases. Indeed, after 150 cycles the 'without communication' curve drops a while, whereas the 'with communication' and kernel curves do not.

In our setup the actual distribution process was as follows. At the moment a platoon registered changes in the fire properties of the objects it could sense, it created messages with the identities of the objects along with their incendiary properties. The message also contained the actual times of when the observations were made. When the center combined all this information it compared the time values for each changed property and merged the most recent property changes into four messages. As the four messages arrived at the other centers a similar 'time value merge function' filtered only the most recent information.

It should be noted here that even though many property changes are omitted from the four messages, it is not always feasible to condense the information of $2 \cdot n$ messages into only four messages. Although we did not experience any problems when setting the MESSAGE field to 256 bytes while encoding changes into 32 bits, we could have chosen to communicate less information, e.g. only information on whether an object is on fire or not, without any quantification.

4.3.4 Conclusion

We have seen that common knowledge among agents is important. A communication model has been developed that provides the means to share information between agents, either within teams or at a more global level. Two distinct modes of the communication model were offered. Uncoordinated communication, in which platoon agents communicate at a local level, using all the bandwidth available to them, and coordinated communication, providing platoons with a more reliable way of distributing information. Though suffering from a few disadvantages, performance is promising. Altogether we think the distribution method suggested may prove very useful for agent coordination, which is the subject of our next chapter.

Chapter 5

Agent coordination

5.1 Introduction

Agents have to cooperate to combat the disaster. This means they have to work together on the same subtask. The most common way to do this, among RoboCupRescue competition participants, is a model where one agent or center takes the lead and decides which task to do. (See chapter 6). Our method is a little different in that it imbues the field agents with more autonomy and a flatter hierarchy. The agents do not communicate about their actions, only about their observations. By doing this they are able to gather as much knowledge about the situation as possible. This information is used to quickly decide on a strategy to combat the disaster, without having to confer with the other agents first. Agents select their strategy from several plans. The way our agents decide on a strategy is by structuring them as a game tree. Game trees consist of states that represent possible situations. From each state the tree branches out to possible future situations. The branches are state transitions representing valid actions by a player in that situation or other occurrences that cause a change in change the state of the game. There are many varieties of game trees, but the concept a graph consisting of future situations connected by occurrences is always present.

There are a few advantages of this method, yet it has a couple of problems to overcome, which we will cover in this chapter. We are not necessarily claiming that this approach is more suited to compete in the RoboCupRescue league, we merely aim to research if using game trees in the decision making process is a viable option that promotes desirable qualities of a multi agent system, like cooperation and being able to deal with complicated problems. Our hypothesis is that an agent that is aware of its team members intentions can predict their actions and cooperate with them towards a common goal.

There are a couple of choices that need to be made on how to use game trees, for example what qualifies as a player, as an action, as a state, when actions are considered valid and what the goal of each player is. In this chapter we will explain our chosen approach. In the theory section we will first formally describe the operation of the decision process. The concepts used in the process that require further specification will be described in the succeeding sections.

First we should map the concepts of the disaster simulator to those commonly used in game theory. To describe the coordination problem as a game it needs players. The decisions these players make determine which course the progression of the game will take. Obviously our agent system makes decisions that influence the state of the world, so it can be seen as a player. In a game a player decides on a strategy based on the situation. Would the situation simply consist of the actual situation in the disaster, then this would be a good choice. The agent system would be like a chess player who gets to move all his pieces every turn. This is however not the case because of the availability of information. Nowhere it the agent system is there complete knowledge about the real situation, nor is it completely known what each agent knows or does not know. To solve this problem the decision making process needs to be able to handle uncertainty. To get the knowledge and the certainty thereof to a central decision making unit, like a disaster center, takes time and to get the decision back to the executing agents takes time as well. When deciding what action to take the information of what the agent knows is more important than what the other agents know, therefore it is smarter to let the agent make his own decisions. By placing the responsibility at the agents we have designated them as players. These players have a common goal: to get the highest score, which is one of the few things shared among the agents at the start of the game. Due to the common goal the agents are not competing. The disaster can be seen as a common adversary, but because the disaster is not able to make decisions or achieve goals it is not really a player. In our approach the disaster causes a state transition after every agent has moved that predicts the results of those simulators that are not affected by the actions of the agents, for instance the spreading of fire and the damage taken by buried civilians.

Another reason for giving the agents the responsibility to choose their own action is inspiration from the study of real world solving of an escalating problem. Centralized decision making seems to suffer from the same problem with latency between observation and decision. A possible solution for real world information problems is improving the detail and speed of information on which decisions are based, for instance by using helmet mounted cameras. However the simulator system is designed to explore solutions with limited communication as specified by the rules. We do have one advantage over real world teams: we can create a team that is perfectly matched and adjusted to each other. A team that is guaranteed to follow the designed protocol and where chance and uncertainty are brought down to a minimum. We can guarantee that given exactly the same situation, agents will come to exactly the same decision. We can exploit this by predicting the actions of an agent's teammate rather than communicate them, which frees up communication resources for distributing observations. Our goal however was not to have agents come to the same conclusion as a central command who only has that information that units share, but to come to better conclusions. As stated in chapter 4, the information that centers pass on can be several cycles old when it arrives back at the agents. The current best decision can be the same, or what used to be a very dangerous fire can be under control by now. The agents in the field may have newer information than what they receive from the center through the high communication protocol. This information may not be the same as that of an agent's teammate, which creates a delicate balance of correctness and agreement. In this chapter we will describe the decision making process in our simulator system and show how it intends to operate given this pragmatic information feed.

5.2 Theory

Our agents are separate units who all have the task of finding out what the most advantageous operation of the entire system is and then executing their role in it. In this section we will give a step by step reconstruction of the methods used to discover the best response of the agent system using simple set theory and pseudo-code.

Based on the information in the worldmodel, described in chapter 3, we can build a model of the disaster situation, in which we can distinguish the following sets of entities:

$\mathbb{A} = Agents$
$\mathbb{C} = Civilians$
$\mathbb{B} = Buildings$
$\mathbb{P} = Paths$
$\mathbb{R} = Refuges$

All these sets except the paths are finite collections of single objectpool items. Paths is a set of compound items, where each path contains a collection of roads ordered from start to end. Because of this a single road object in the object pool can appear in more than one path. None of the other sets reference the same same items. Our worldmodel Ω is a set consisting of of these subsets and a number c for the cycle number of the simulator system. The items are parameterized with various properties. Paths can be blocked, Buildings can be burning or extinguished, agents can be damaged. These properties are continues values making the total number of possible worlds infinite in theory and a digital approximation of this in the simulator system.

$\Omega = \{\mathbb{A}, \mathbb{C}, \mathbb{B}, \mathbb{P}, \mathbb{R}, c\}$

In a disaster situation there are a finite number actors $\mathbb{A}_i$, each of whom can perform a finite number of possible actions $\mathbb{X}_i$, because all actions are parameterized with a target object and there are a finite number of targets in the simulated world. The actions we use are described below, with the item they can be parameterized with between brackets. The kernel allows extinguish actions to be parameterized with a water quantity as well, but we always use the maximum allowed water output.

- Move (Path)
- Extinguish (Building)
- Clear current road
- Rescue(Civilian)
- Load (Civilian)
- Clear road
- Unload

We have registered the average number of moves that one fire agent could perform and would cause a change in the situation, during a simulation of 300 cycles of the well known Kobe map. In figure 5.3 we compare this to two other methods of generating moves, which we will introduce later. For this simulation the average size of $\mathbb{X}_i$ is ~ 938. This number is an estimate for just one possible starting situation. Depending on the map this number could be a lot higher or lower, but we will assume that for a normal competition simulation the that size of the set of all possible unrelated actions for each actor will be around 1000.

The agent system can choose and send one action for each of the agents under its control for each step time step. Every time step the simulators get one iteration in which they compute the new situation. We can define the result of our agent coordination system as the set of all actions that are accepted by the kernel.

$\mathbb{Y} = [\mathbb{Y}_1 \dots \mathbb{Y}_{|\mathbb{A}|}]$

$\mathbb{Y}_i$ being the action selected for actor $\mathbb{A}_i$ which is an element of $\mathbb{A}$. The total set of possible responses we call τ. The total number of responses is fairly large, because the responses for each actor are unrelated τ contains all possible combinations of $\mathbb{X}_i$. Sometimes one agent's action can cause a different result for another agent's action. The simplest example would be one agent that moves into another agent's path blocking him from reaching his destination. Of course this doesn't make any difference for the possibility of those moves being sent to the simulator system and only enforces the need to consider all the responses of the agents as a whole. The total size of τ can be computed as in equation 5.1. Given our previous approximation of $|\mathbb{X}_i| \approx 1000$ and a typical number of agents 25 we get $|\tau| \approx 10^{75}$.

$$|\tau| = \prod_{i=1}^{|\mathbb{A}|} |\mathbb{X}_i| \tag{5.1}$$

Now that we have a definition of the input and the output of our decision making system we can start dissecting the function, that our agent system embodies. $\mathbb{Y} = \mathbb{F}(\Omega)$

It can be split up into two steps, one generates τ from Ω and the seconds selects the most desirable response from τ.

$\mathbb{Y} = \mathbb{F}_{agents}(\Omega)$:

$\tau = \mathbb{F}_{responses}(\Omega)$

$\mathbb{Y} = \mathbb{F}_{think}(\Omega, \tau)$

Every cycle the kernel will pass the response to the simulators and combined with the simulated disaster progression this will lead to a new situation Ω'. The entire simulation between agent system and simulator system can be seen as calling the agent system function and simulator function in turn until the cycle of the resulting worldmodel has reached the configured end cycle. During competition simulations this is a simplification because both the simulator and agent system have to provide their response before a configurable time limit. During testing we simply use a high enough time limit to prevent it from interfering with the result of the either system.

$\Omega' = \mathbb{F}_{simulator}(\mathbb{Y}, \Omega)$

The function $\mathbb{F}_{think}$ needs to select the response is the one that will lead to the highest score at the final cycle. If $\mathbb{F}_{think}$ can predict Ω' for each $\mathbb{Y} \in \tau$ it can recursively call $\mathbb{F}(\Omega')$ to generate a tree of all possible response sequences for an entire simulation. This tree could be considered a game-tree [7]. A response sequence determines a path from the start situation, the root of the tree, to the end of the game at cycle ϵ. A set of responses for the all future steps is called a pure strategy ($\mathbb{S}$):

$\mathbb{S} = [\mathbb{Y}_1 \ldots \mathbb{Y}_\epsilon]$

All strategies will lead to a final situation at the last cycle. The game-tree is the set of all possible strategies with their resulting situation.

$\mathbb{G} = [\{\mathbb{S}_1, \Omega_1\} \ldots \{\mathbb{S}_\epsilon, |\Omega_\epsilon\}]$

Selecting the right strategy is a matter of generating a tree of all possible strategies with their resulting end situation and then selecting the situation with the highest score and the strategy that leads up to it. The first response in the strategy is the response that should be given at this iteration of the simulation to get the highest score in the end.

```
$\mathbb{Y} = \mathbb{F}_{think}(\Omega, \tau)$ :
    $\mathbb{G} = \mathbb{F}_{expand}(\Omega, \tau)$
    $\mathbb{S} = \{\mathbb{Y}_0 \ldots \mathbb{Y}_\epsilon\} = \mathbb{F}_{select}(\mathbb{G})$
    $\mathbb{Y} = \mathbb{Y}_0$
```

Figure 5.1: *Basic agent system function.*

Generating all strategies is a recursive process where the game tree is expanded for every cycle until the condition for the end of the game is reached. For our simulation the end condition is simply that the cycle counter has reached the configured end cycle(ϵ). Expanding the game tree means simulating the results of each response and generating a new set of possible responses in that situation.

```
$\mathbb{G} = \mathbb{F}_{expand}(\Omega, \tau)$ :
    $foreach\ \mathbb{Y}_i \in \tau$
        $\Omega'_i = \mathbb{F}_{simulate}(\Omega, \mathbb{Y}x)$
        $c \in \Omega'_i = \epsilon?$
            $\mathbb{G}'_i = \{\mathbb{Y}_i, \Omega'_i\}$
        $c \in \Omega' \neq \epsilon?$
            $\tau'_i = \mathbb{F}_{responses}(\Omega'_i)$
        $\mathbb{G}'_i = \mathbb{F}_{expand}(\Omega'_i, \{\mathbb{Y}_i, \tau'_i\})$
    $\mathbb{G} = \mathbb{G}'1 \cap \ldots \cap \mathbb{G}'_i$
```

Figure 5.2: *Simple game tree algorithm.*

Because of the high computational complexity of the simulation, the total set of possible strategies at each cycle will be extremely large. The computational complexity of a game with an average branch factor $\overline{\mu}$ and a fixed length ϵ is $\overline{\mu}^{\epsilon}$. The branch factor is the number of possible valid responses $|\tau|$, which we have estimated to be particularly large for our agent system as well using equation 5.1

With an average number of valid moves per agent $\overline{\mu}$ the total number of leafs on a tree for a game with a fixed length of ϵ has a complexity of $\overline{\mu}^{|\mathbb{A}|\cdot\epsilon}$.

With most other games the depth of the tree is not fixed because there are clear win and loose situations, which mean the game ends. This is not so for our simulator. The goal of our simulator is to have the highest score by the last cycle as determined by the kernel configuration. We will compare the number of strategies at the start of a tic-tac-toe game and the estimated number of strategies of a chess game to the number of strategies a multi agent system has at the first cycle of a typical 300 cycle simulation run with 25 agents.

Clearly our game tree is too large to be computed entirely, and by far too large to be computed in a real time simulation. Our first adjustment will be to reduce the branch factor by eliminating all actions that are seem-

game	length	branch factor	strategies
tic tac toe	5 to 9	9 to 1	26830 [2]
chess	38.58 [3]	35 [17]	$3.71 * 10^{59}$
RCR UvAC2003	300	1000^{25}	$1000^{25 \cdot 300} = 10^{22500}$

Table 5.1: *Estimations of typical number of strategies.*

ingly useless. Examples of this are extinguish actions on a house that is neither burning or about to start burning, clearing of a free road, moving to a position that is not known to be near a burning building, blocked road or civilian. We do this by combining move and operational actions into behaviors parameterized by disaster targets (See section 5.3.3 for a description of these behaviors). As shown in figure 5.3 this reduces the number of actions that are considered for expansion

Our expand function does not really change when behaviors are used instead of actions. What does change is the prediction function. Instead of the results of an action we have to predict the result of a behavior. This is done by predicting what action will be executed for this behavior in this situation and then applying the effects of that action as normal. $\mathbb{Y}_i = \mathbb{F}_{execute}(\mathbb{B}_i)$ And because of this the $\mathbb{F}_{simulate}$ function, which now works with a behavior strategy, has to predict the actions of all the behaviors in the current strategy to apply them to the expected world state.

```
$\mathbb{F}_{simulate}(\Omega, \mathbb{S})$ :
  $\mathbb{S} = \{B0, \ldots\}$
  $\Omega' = \mathbb{F}_{predict}(\Omega, B0)$
  $\mathbb{F}_{simulate}(\Omega', B0)$
```

Because the execution of a behavior can take more than one turn we have removed a natural advantage of our previous approach, which only considered valid actions. These actions had to be executed in one turn limiting the number of targets to those within range of the agent. All possible targets in the map are considered when creating behaviors and not just those that are reachable in one turn. When there are a lot of blocked roads in the start of the simulation or a lot of burning buildings after a while there will also be a lot of possible behaviors that need to be considered. Because of this the reduction of the branch factor by introducing behaviors is considerable, but not as big as may be expected.

We can reintroduce the action radius by artificially limiting the number of considered targets using our previously described sectors. If we make the assumption that traveling between remote disaster clusters is rarely desired, we can force the fire agents to focus on a connected fire front, the police agents on clearing the roads around it and the ambulances to rescue those civilians in the most immediate danger area first. The sectors in which the map is divided become target locations in an entirely new game, where the goal is to assign the right number of agents to the right sectors. The way this super game attains its distribution is not relevant to understand our coordination function, more on that in section 5.3.2. That reducing the number of agents and targets considered lowers our branch factor even more is shown in figure 5.3.

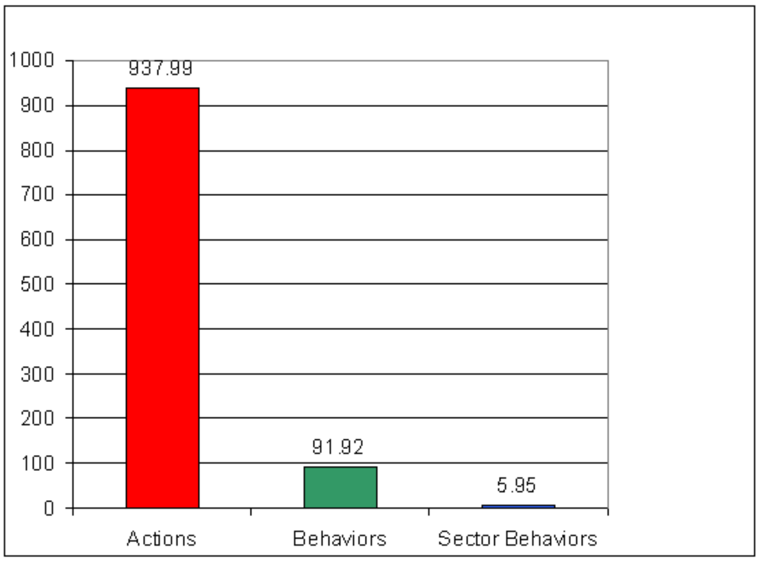


Figure 5.3: *The branch factor without behaviors, with behaviors and with sectors.*

Besides choosing a different representation of actions we can also use well known techniques to prune a game tree to lower the number of strategies to consider. We can limit the expanding of the tree to a certain depth and assume that those strategies that result in situations with good scores to the point where the strategy provides responses are likely to be the beginning of strategies that are good for the end game. This form of pruning quickly reduces the size of the tree to more reasonable numbers. During the execution of the partial strategy we can compute a new set of strategies at every cycle, taking the current situation as the new root for the tree. For a maximum strategy length λ the number of evaluated situation becomes $\epsilon \cdot \mu^{\lambda}$.

This prevents us from knowing if the chosen partial strategy is indeed the very best one. In fact the chance of it being the perfect one grow smaller when we search less of the total game tree. We will have to settle for a strategy that seems like the best way towards a good solution. The further we look ahead the less chance we will have on choosing a shortsighted solution that is not beneficial in the long run. The adjustment needed to algorithm 5.2 is relatively simple. All we have to do is modify our $\mathbb{F}_{expand}$ to stop at a cycle that is λ steps from the current cycle instead of at the end of the simulation (ϵ).

Even with a very small deepening limit our branching factor is far too large to look ahead for more than one cycle given the time it takes to predict a situation and evaluating the result. Limiting the depth of the game tree is very undesirable, but unless we are able to reduce the width of the tree even further we have no choice. We have used several techniques to drastically reduce the branching factor, while attempting to maintain a high average quality of the remaining branches. What we have not done yet is the well known method of heuristically pruning the tree. In competitive two-player zero-sum games the method that is used is Alpha-Beta-Pruning[24]. Our variant however is not a competitive game and instead of choosing a lower limit on what behaviors are considered we select a set with a fixed size. By selecting only a number (Π) of the most promising strategies to expand we can cut down the tree to any desirable size. Again we have to take care not to limit too much or we will increase the chance of pruning the best solution. These two limiting techniques would make the deepening algorithm as shown in fig 5.4.

```
$\mathbb{G} = \mathbb{F}_{expand}(\Omega, \tau)$ :
    $foreach\ Y_i \in \tau$
        $\Omega'_i = \mathbb{F}_{simulate}(\Omega, \mathbb{Y}_i)$
    $\mathbb{G}' = [\{\mathbb{Y}0, \Omega'0\} \ldots \{\mathbb{Y}_{|\tau|}, \Omega'_{|\tau|}\}]$
    $\mathbb{G}'' = \mathbb{F}_{prune}(\mathbb{G}', \Pi)$
    $foreach\ \{\mathbb{Y}_j, \Omega_j\} \in \mathbb{G}''$
        $c \in \Omega_j = \lambda + c' \in \Omega?$
            $\mathbb{G}'''_j = \{\mathbb{Y}_j, \Omega_j\}$
        $c \in \Omega_j \neq \lambda + c' \in \Omega?$
            $\tau'_j = \mathbb{F}_{responses}(\Omega_j)$
            $\mathbb{G}'''_j = \mathbb{F}_{expand}(\Omega_j, \tau'_j)$
    $\mathbb{G} = [\mathbb{G}'''_0 \ldots \mathbb{G}'''_{\Pi}]$
```

Figure 5.4: *Game tree algorithm with width and depth pruning.*

Our number of situations to generate and evaluate has been decreased dramatically. The size of the game resulting game tree is now Π^{λ}, but to reduce the set τ to a size Π we still have to evaluate all possible branches in τ once to select the best Π. This has to be done for every deepening step, with a maximum of λ, so the total number of branches to evaluate is as shown in equation 5.2. If we make Π small enough, this will easily be less than when $|\tau|$ was the factor for the exponent.

$$\lambda \cdot |\overline{\tau}| + \Pi^{\lambda} \tag{5.2}$$

Since $|\tau|$ is the possible set of responses for this cycle, for a give team size and target set, it can still be large. It is in fact $\overline{\mu}^{|\mathbb{A}|}$. Even when $\overline{\mu}$ is pretty small for the average situation as shown in figure 5.3, it can still get large if for instance the fire in a sector gets out of hand. At this point we have done a lot of things to reduce this number already and the only thing left to do is limit the effect of $|\mathbb{A}|$.

All these responses are simulated and the resulting situations are evaluated before selecting the best ones. We can reduce the number of simulated responses by only generating the best ones. The way to do this is split up this node in the game tree into a smaller subtree, where each agent moves once in turn before the disaster situation and time are advanced as would be done by the simulator system. This allows us to prune the tree when generating τ. All possible actions for the first agent at the first time step are generated ($\mathbb{Y}_1^1$), and simulated. The few best ($\mathbb{Y}'^1_1$) are selected by assuming the rest of the agents do not act and the predicting the resulting situation. Like we did when pruning the game tree for the total response set, we can limit the size of $\mathbb{Y}'^1_1$ to an optimal value (π) and expand only those nodes by generating the actions of the next agent, assuming the previous agent did what it did to lead to this situation and assuming the rest of the agents do nothing. If we do this till all agents have an action generated for them we have simulated less situations. (eq. 5.3)

$$\overline{\mu} \cdot |\mathbb{A}| + \pi^{|\mathbb{A}|} \leq \overline{\mu}^{|\mathbb{A}|} \qquad for \qquad \pi < \overline{\mu} \tag{5.3}$$

This approach can be integrated in the pruning of the main tree if we define $\pi = \Pi$. This makes the algorithm more elegant and it prepares our approach for a possible future adjustment where π is different at each depth of the tree to accommodate methods to use the allowed computation time as effectively as possible.

5.3 Design

5.3.1 Agent architecture

By limiting agents to a sector we have split up the coordination problem in two levels. When the disaster starts this usually creates multiple problem areas. Where the earthquake hit hardest there will be many civilians buried in collapsed buildings. The locations where fire initially starts are not necessarily related to this, so in that case, to be more effective, the agents will have to split up. The ambulances are needed most at the collapsed buildings and the fire fighters at the burning buildings. Both groups need the support of police agents to clear their path and scout out the area. Sometimes it can even be more efficient to divide the fire agents over multiple teams, split over fires based on proximity and urgency of a disaster area.

By the rules of the competition the agents are required to be black boxes that have only the allowed input and output. The input consists of the map

```
$\mathbb{G} = \mathbb{F}_{expand}(\Omega, \lambda, \Pi)$ :
    $\lambda \leq 0$?
        $\mathbb{G} = \Omega$
    $\lambda > 0$?
        $\mathbb{G}_0 = \Omega$
        $foreach\ \mathbb{A}_i \in \Omega$
            $foreach\ \Omega_j \in \mathbb{G}_i$
                $\tau_j = \mathbb{F}_{responses}(\Omega_j)$
                $\tau'_j = \mathbb{F}_{prune}(\tau_j, \Pi)$
                $foreach\ \mathbb{B}_k \in \tau' j$
                    $\Omega_j^k = \mathbb{F}_{predict}(\Omega_j, \mathbb{B}_k)$
            $\mathbb{G}_{i+1} = [\Omega_0^0 \ldots \Omega_{|\mathbb{G}_i|}^{\Pi}]$
        $foreach\ \Omega_l \in \mathbb{G}_{|\mathbb{A}|}$
            $\Omega'_l = \mathbb{F}_{disaster}(\Omega l)$
            $\mathbb{G}'_l = \mathbb{F}_{expand}(\Omega'_l, \lambda - 1, \Pi)$
        $\mathbb{G} = [\mathbb{G}'_0 ... \mathbb{G}'_{|\mathbb{A}|}]$
```

Figure 5.5: *Game tree with pruning of every agent's move.*

of the city at the start of the simulation, visual information on the changing properties of the disaster area and a fixed number of communication messages from the say and tell channels. The output is one action per cycle and a number of say and tell messages that is not limited, even though the number of messages that may be received is. Direct information exchange between agents is not allowed in any way. This is of course done to simulate a real world disaster where the only possible link between autonomous rescue units is over wireless connections like radio.

What is done inside the agent is more or less free as long as no additional source of map specific information is used. Our information and action go through a couple of transformations that are shown in figure 5.3.1.

At the top of the architecture of our agent is the worldmodel as described in chapter 3, receiving input in the form of say and tell messages and observations. By the communication output this world model is kept as up to date and synchronized as possible with other agents as described in chapter 4. The subject of this chapter is how an action for the agent is constructed for this agent based on the information of this worldmodel. In the previous section we have given an overview of the method we use. We saved going into details about the data structures used in this process for this section.

The sector map we extract from he object pool is used to make a task and team distribution. In the task and team section below we will give a list of tasks we implemented and the way we distribute our agents over them. For now it is enough to know that from its appointment to a task the agent can simply see what sector he needs to get to, what he is supposed to do

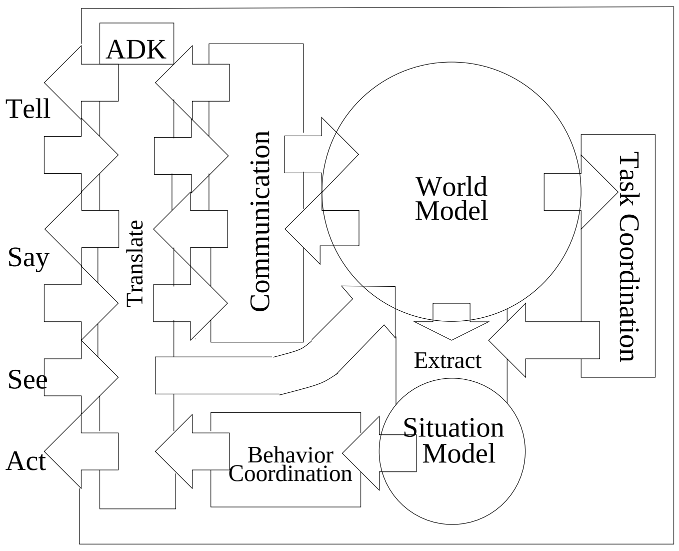


Figure 5.6: *The process of turning information provided by the kernel into actions.*

there and what other agents he will be working with on this task. These three specifications of the agent's task are used in selecting only the relevant information from the world model. This sub-selection is called a situation model and in the section to devoted to this data structure we will explain how to determine what information is included in this extraction.

The situation model is the information that is used by the agent to create an action. The process used for this was formally described in section 5.2. Since it is the focus of this chapter, we will further explain its operation in the next sections. After that we will evaluate its operation in section 5.4.

5.3.2 Tasks and Teams

The decision of the distribution of agents over the problem areas is done by the *task coordination.* It can be seen as the top level of coordination, because it makes decisions that would be the responsibility of officers in the hierarchy of human fire- or police forces, which often resembles a military structure. The resemblance to human disaster control teams is not entirely accurate. The immobile center agents, which are intended as a sort of headquarters from which disaster managers can steer the units in the field, do not make any decisions in our solution. They are used like a communication center only. The information on which the task allocation decisions are based is merely composed here from reports send in by the agents and send back to the agents to make the final decision themselves.

Each agent decides for itself what task to commit to and what team to cooperate with based on the sector information in the worldmodel. The sector map has cut the objectpool into parts. For each of these parts the agents computes a threat value. In chapter 3 we elaborated on how these values are computed. When all agents agree on this information they can all decide on the same task and team distribution, because they use the same knowledge. There are various interesting ways in which this distribution can be done, but because this part of the coordination is not our focus we have used a fast and predictable method for the competition and to implement and analyze the coordination within a team. We simply divided the total set of agents into three teams.

The first team contains all fire agents and two police agents to support them. This team has the task to fight fire in the sector that has the highest fire value. The fire agents will consider all burning buildings in the target sector as potential targets for the extinguish behavior. To do this a couple of requirements must be met. First of all each fire agent of the team needs to be able to get to the desired sector. If they have a clear path to the sector this will be done by a simple traveling behavior. If the route to the sector is blocked however, the fire agent needs help from a police agent. The police agent moves to the trapped fire agents, removing roadblocks on the way. This is one of the most interesting examples of cooperation in our agent system. When the fire agent gets to the sector, he has to choose between all the different burning buildings. A large factor of this is of course which building is the most important in stopping the spreading of fire. For this the fire danger values from section 3.4 are used. Buildings who are not reachable because of known roadblocks inside the sector are quickly dismissed, no matter how important they are. The police agents who are working inside the sector will clear the roads near high priority burning buildings sooner because they will predict that if they do that, more good options for the fire agents after them become available. Another coordination decision that is solvable by the game tree. The last factor for choosing the building to extinguish gives us our last cooperation example. Assuming they have the same information most agents will be drawn towards the highest priority buildings anyway. Because our prediction function simulates the effect in the fire simulator where there is an optimal number of fire agents to extinguish the same building together, the agent will see a bonus priority in choosing the same building as his peers or see diminishing returns if he predicts enough other team members who will concentrate on that building. When a fire agent is nearly or completely out of water he can travel to the nearest refuge to refill.

The second team consists of all ambulances with two supporting police agents and of course this team has the task to get all civilians out of the most damaged sector. The traveling to the sector and cooperation with the police units is the same as the fire fighting task. The ambulances have to dig out civilians, load them and bring them to a refuge in one complicated behavior. Buried civilians are a lot harder to spot than burning buildings so when an ambulance knows no civilians in the area it reverts to driving around to find them.

The remaining police agents are in a team together and have the task to travel along the highways to improve the information in the sector map and at the same time they work on removing debris from these important routes. These agents have no sector to focus on. Their first responsibility is, to keep the threat information (section 3.5) up to date, which is important for the task coordination. These agents will be the first to spot a starting fire in a different sector and the information they provide about the highways and what they have seen when traveling near these sectors aids in deciding whether it will be profitable to redirect the rescue or the fire fighting team to other threatened sectors. They select the places to travel to by using the intel value from the threat map. When this value gets too low, paying it a visit will take priority over removing debris on the highways. Since this behavior chooses targets based on threat information, which is *not* part of the situation model yet there is not a lot of coordination in this task yet.

5.3.3 Behaviors

Behaviors are small programs that are generated based on a situation model. After the most desirable one is selected based on predicted result, it generates one action for that agent that round. We have described how a behavior is selected and will describe now how the behavior is executed.

Agents can choose between a limited set of elementary actions to send to the simulator system. Every agent can move over roads and into buildings, but the different agent types have specific rescue capabilities to affect other objects. Fire agents can perform an extinguish action on a building, police agents can perform a clear action on a blocked road and ambulance agents can dig civilians from collapsed buildings or load and unload them to take them to a safe area.

An agent can perform either a move action or a rescue action at every cycle. To perform a rescue action successfully the agent must be at the right location. For extinguish actions the agent has to be within a fixed distance of the building he is targeting. To clear a road an agent needs to be on that road or on one of two nodes that road is connected to. To dig a civilian out of a building the agent needs to be either inside of that building or on one of the nodes that define the entrances of the building. So an agent has to be able to reach the required position or the attempt to perform the action will be futile. Sometimes, there are also other requirements to perform an action. Fire agents planning to release a certain amount of water on a fire, should have at least as much water in their supply tank. Ambulances need to be empty before picking up more civilians and of course all agents need to be able to move and not get stuck in a collapsed building or be destroyed.

If agents do a move action they do not affect the situation in a way that could be considered a direct improvement. To decide whether a move is desirable the agent would have to evaluate the possible actions in the possible next cycle and propagate the results of that effect back. Covering larger distances makes this effect even stronger. Considering the situation keeps getting worse without action, because of fires spreading and buried civilians dying, a move action can be seen as going through a local minimum in the jargon of search algorithms. Linking the move action to the goal oriented action for which it is required avoids this problem. Checking if the requirements have been met can also be done for the entire plan of a compound action, making it easier to reject it early. In chapter 5 we will explain how we use a search tree to decide the agent's active behavior. Pruning this search tree becomes a lot easier by selecting from compound actions to avoid local minima.

To make the possible actions more easy to handle they can be grouped into a composite action, that also checks for the required circumstances. The reason for grouping actions into behaviors is to reduce the number of choices and make the evaluation of their effectiveness easier. Behaviors are goal oriented and there will almost always be less attainable goals than possible elementary actions. The relation between the fulfilling of this goal and its effect on the total situation is also more predictable than that of an elementary action. At each cycle the agent has a large but limited set of actions it can perform. As an example we will use a fire agent. This fire agent can spray water on any building within its extinguish radius, even the ones that are not burning yet. The agent can also attempt to move to any position that does not require more distance to be traveled than is allowed by its speed. Certain positions may be blocked by roadblocks, but the attempt to get there is still an elementary action that requires the same process to be rejected as one that is more feasible. The maximum amount of extinguish-behavior goals that this agent has is equal to the total number of buildings in the map. All buildings that are not on fire can be rejected early and after that all buildings that are not reachable because of blocked roads. If the entire city is burning this is still a large number of behaviors, so we cut down on this number one last time by using sectors, which were explained in section 3.1.

We will now describe the set of behavior used by us during competition and research. You may notice the absence of active behavior for ambulance agents. This is because we focused on fire fighting when evaluating our agent system and our competition version of ambulance behavior slightly deviated from the design pattern presented here which would dilute the point. Our simulations focus on the cooperation between fire brigades and police vehicles instead.

5.3.3.1 Travel

Units Ambulance, Fire brigade, Police

Task All tasks

Requirements Unit is operational and not in the desired sector

Position The gridpoint of the destination sector that is closest

Target Appointed task sector

Action None

Description Travel is a behavior that is used for long distance movement. It is generated for all agents who are not in the target sector. Most of the time these agents will not have other behaviors to choose from, since all goal oriented behaviors are defined as only being legal when initiated from within the active sector. This cuts down the branch factor for those agents who are not going to actively participate in fighting the fire, saving the civilians or improving the roads in the disaster area for the first couple of kernel cycles. Separating the travel behavior from the *short distance* behaviors also makes it more realistic to assign a heuristic value to their desirability without having to apply penalties for travel time. Lastly making the travel behavior the only valid one when not in the sector avoids the trap of local minima in the search tree that is caused by traveling not providing any direct improvement to the situation. Sometimes agents have a specific reason to be outside the sector and do get the choice between doing their task and traveling back. The two behaviors that can exist as choices beside travel are RefillWater and FreeTeammember. Obviously these are important goals so the agent will almost always pick them over traveling back until they are fulfilled.

5.3.3.2 Extinguish

Unit Fire brigade

Task Fight Fire

Requirements The unit is operational, has water in the tank and is in the desired sector

Position The road or node that is within extinguish distance, but furthest away of the target building

Target A burning building

Action Extinguish

Description This is the main operational behavior of the fire unit, so it is only generated when the actor is a fire agent. This is one of the short distance behaviors, therefore being in the target sector is a requirement. The agent also needs enough water to extinguish or the behavior will not be generated. This avoids the local minimum where a fire agent wants to move to the right position because the heuristic favors extinguishing, only to compute at the next time step that moving towards an extinguish target without water has been useless. This could happen for all reachable burning buildings in a sector, wasting a lot of tree expansion operations.

5.3.3.3 Clear Path

Unit Police

Task Fight Fire, Search and Rescue

Requirement The unit is operation and in the desired sector

Position The closest node or road on the path that is blocked.

Target A path from the current position to a blocked road

Action unblock the blocked road

Description Police agents who are in a heterogenous team with fire brigades or ambulances help them travel the assigned sector by removing any blockades they encounter. This is done in a quite opportunistic manner by clearing from the current position to another position with a known blocked road. Proximity is the most important priority for selecting the roads to clear, to avoid the police agent wasting time on traveling and getting sidetracked by other blockades along the way to more urgent positions.

5.3.3.4 Recon

Unit Police

Task Highways

Requirement Agent is operational

Position The closest gridpoint of the sector who's intel value is lowest

Target The same gridpoint

Action None

Description Recon is one of the two information gathering behaviors. Recon is the abbreviation of the military term reconnaissance. The goal is to gather information concerning the general disaster level of the target sector. This information is used by the task coordination to prioritize the assignment of teams to sectors. This is done based on the perceived percentage of burning and collapsed buildings and the number of blocked roads. The recon behavior is generated for police agents that are in the team specifically formed to work outside of the sectors. They do not have to cooperate with fire brigades or ambulances. Their situation model does not contain a specific sector, just the highways.

5.3.3.5 Patrol

Unit Police

Task Fight Fire, Search and Rescue

Requirement Agent is operational and in the desired sector

Position The node, road or building who's information is oldest

Target The same position

Action None

Description The second information gathering behavior is targeted at a specific sector. Its goal is to increase the knowledge of the disaster fighting team about their surroundings. Again its generated for police agents, but unlike recon these police agents are in a team with fire brigades or ambulance units. In a mixed team like this the first task of the police agents is of course to clear the roads so the disaster fighters can do their job. If no known blocked roads are obstructing their team-members the police agents can either search for clear work in the neighborhood or confirm to the team that the sector is freely travelable.

5.3.3.6 Refill

Unit Firefighter

Task Fight Fire

Requirement The agent is operational, in the desired sector and his water tank is not full

Position The refuge that has been designated as the closest refuge for the target sector

Target The refuge

Action Wait

Description Fire agents have a fixed size tank of water that is emptied whenever an extinguish operation is used. To remain functional they have to return to a refuge and wait there while their tank is automatically refilled. This behavior is generated for each fire agent who's tank is not full. In rare occasions it can therefore happen that a fire agent decides to go for a refill because the tank is empty. When the tank is empty all extinguish behaviors are judged to be illegal moves, so what remains for the agent is usually to go for a refill. There are a number of refuges on the map, who's location is known at the start of the simulation. During the division of the map in sectors the closest one is computed for each sector and stored as the assigned refuge for any team who is operating in that sector. This refuge and the path to it from each of the sector's gridpoints is added to the situationmodel for that sector, so fire agents can make predictions about the reachability of the refuge and police agents can improve it.

5.3.3.7 Free Teammember

Unit Police

Task Fight Fire, Search and Rescue

Requirement The agent is operational

Position On the path between actor and target team member

Target The team member who is blocked in

Action Clear road

Description Team members outside of the target sector need to perform the travel behavior. They do this along one precomputed path between their position in another sector and the closest gridpoint of the target sector. The paths from agents outside the sector to the nearest sector corner is always added to the situation model, so this path is always available. If this path is blocked it needs to be cleared so the team member can join the group in fighting the disaster. Ambulances and fire brigades can not do this themselves. The police units added to the fire fighting or ambulance group need to help them out. One free behavior is generated for each ambulance of fire fighter not in the target sector.

5.3.4 Situation model

The situation model is a structure that defines a static state in the decision tree of an agent. It is not a model of the real world, but rather of an agent's *view* of the real world. The representation of the knowledge an agent has about the real world is contained in the world model, described in chapter 3. At the start of the thought process of the agent, as shown in figure 5.3.1, the situation model is *extracted* from the world model. This process applies the focus of an agent on its view of the world. Selecting only those objects relevant to the current task of the agent is the most important way of focussing. Besides that properties of objects that can be ignored, like the exact topography of a building, are not extracted and others are combined in a more qualitative value like material, size and fieriness into fire danger.

The agent uses the decision of who its team members are and what its task is to determine what information is relevant. The task will usually involve a target sector, so all objects from this sector are relevant. A refuge is needed in behaviors, like refilling the water tank and evacuating civilians, so the nearest refuge and the shortest path through it from the target sector is also required. The team consists of set of agents that the agent expects to also have the same task. The agent needs to cooperate with these agents in various ways, so they are also part of the situation. If the agents are outside the sector they need to travel to it first, possibly with the help of police agents to clear their route. This route is also part of the situation.

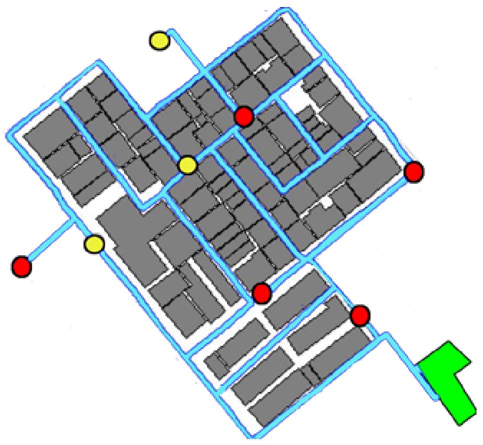


Figure 5.7: *A possible situation model of one of the center sectors of the Kobe map.*

The construction of a situation model is done mostly for efficiency reasons. It would be possible to imagine an alternate situation model consisting of a copy of the world model and the task and team decision that is used to filter only the relevant information from it. The use of a customized structure allows us to describe the world in terms that are often used to validate behaviors and score their effectiveness, which saves having to compute these over and over again. Storing only required information makes a state use much less system memory and also we can store the tree of situation models by letting child nodes only store the information that has changed due to the state transition from its parents. When an agent performs a move action the resulting situation model will only store the agent and its new position, when an agent extinguishes a building only the agent's new amount of water is stored and the fact that the target building is now more extinguished than it was.

5.3.5 Behavior Coordination

The situation model is the root of the earlier discussed game tree and extracting it from the world model is the first step each cycle in the process of selecting an appropriate action. This is done in a few phases as show in figure 5.8. After the situation model is extracted from the worldmodel as described in the previous section, this situation model is expanded into a tree of possible strategies and resulting situation models using the method described in section 5.2. The current situation is the root of the tree and the leafs of the tree are possible futures.

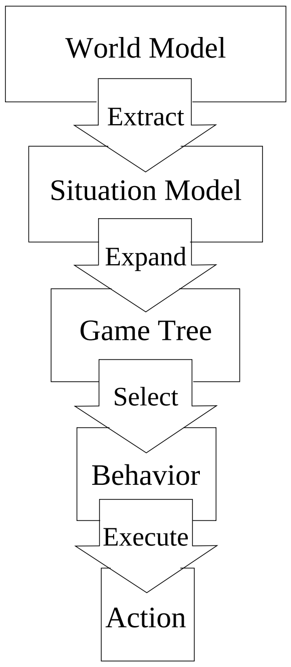


Figure 5.8: *The behavior coordination process.*

For each of these futures it is possible to calculate a score that resembles the score used to determine the performance of an agent system in a competition game. It is not exactly the same, because the situation model is not the same as an objectpool on which the official score is based. For instance, a situation model only describes one single sector and the agents can not be held responsible for what happens in other sectors. Also the properties of the buildings and humans in the sector are not exactly in the same proportions. They are computed by approximated of the simulators and some of them may not even be known. The scorings function does however use the same criteria: damage to buildings and damage to civilians and agents. This score is calculated for all of the leaves in the tree and one of them will be the highest. In the unlikely case that two situations have the same score, the tree has been constructed so the first leaf generated was for the strategy with the highest heuristic scores so, this one will be selected first as a tie breaker. We now have a list of behaviors leading up to the predicted best situation. This list forms a strategy for the entire team. The first behavior for the agent that is computing the thought process is the behavior that this agent should execute and as such it is selected from the game tree.

Executing the behavior has already been simulated while predicting its attainability and result. The agent knows which position it should be to perform the desired action, whether it is already at the position or should move first and if it should move, what path to take that according to the best of its knowledge does not pass any blocked roads. For efficiency this information has been stored during prediction and executing the behavior is therefore a matter of just passing the desired action to the ADK [9] which will take care of sending it to the kernel.

5.4 Evaluation of the Effect of game trees

The expected effect of game trees is that goals with low short term pay offs but higher long term pay offs are chosen above goals that high short term pay offs. Among agents this should increase the level of cooperation by agents foregoing their own short term success for the sake of the long term success of the team. The typical situation where this happens is when an agent moves towards another team member so they can cooperate on extinguishing the same building instead of them both extinguishing different buildings that they can reach right away.

To observe this effect as clearly as possible we have create a very simple map with eighteen buildings and two fire agents which is shown in figure 5.9. In our test situation we only used the two fire agents on our small test map and allowed the agents more than enough time to compute game trees of the configured size. The sizes we are especially interested in are game trees that only think one step ahead, in other words game trees with a depth of one, and game trees that do not consider other possibilities than the ones with the highest heuristic score, those with a width of one. Our test situation is specifically set up to lure both agents into extinguishing their own closest building, because the other building is out of extinguishing range and would require moving towards it first. With two burning buildings there are exactly four behaviors that are generated for a fire agent performing the fire fighting task: Extinguishing either building, patrolling or refilling the water tank.

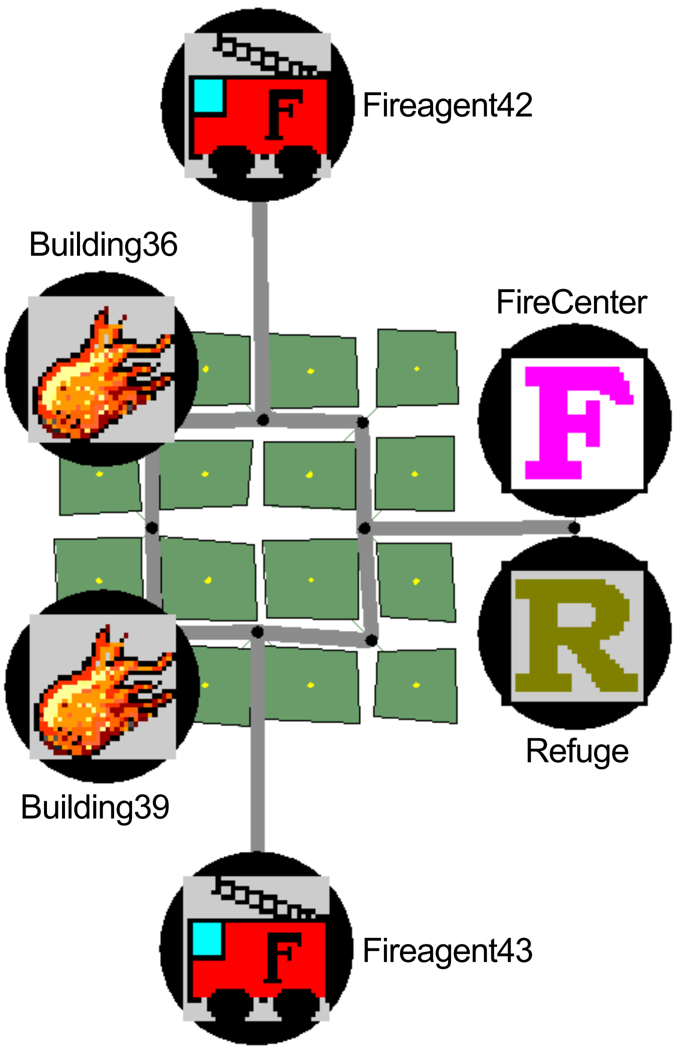


Figure 5.9: *The map used for evaluating the game trees. The vehicle icons show the starting position of the two fire agents, the flames are where two buildings are ignited and on the right are the refuge (R) and the fire station (F).*

Since we want to determine the working of the decision making process and not how this decision making process deals with the characteristics of the communication module, we have shut this off. The action radius of an agent is much larger than its visual range, so encountering a situation where one agent has the exact same knowledge about a possible target, yet has to move first to reach it is impossible. The working of the communication method will always have a large impact on the effectiveness of the game tree coordination. Just for evaluation purposes it is possible to ignore the rules and spread all observations to all agents immediately, which is what we did to obtain our results. We passed on the observations of each agent directly into the world model of the other agent, removing the need for communication

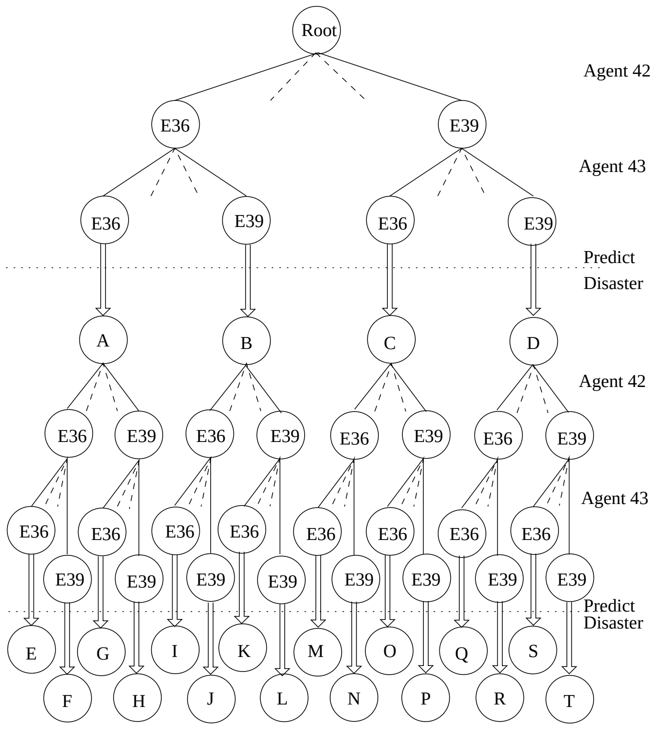


Figure 5.10: *The game tree for a team of two agents with a depth of* 2 *and a width of* 2.

We have run a simulation on this map in three configurations. Width a width of one and a depth of two, with a width of two and a depth of one and finally width a depth and width of two. The game tree of the latter is shown in figure 5.10. There are tree types of situation models here shown as circles. The root node is the extracted situation model. The nodes with $E36$ or $E39$ in them are the result of the agent mentioned on the right of those branches predicting an extinguish behavior on either building 36 or building 39, which are the two buildings set on fire in the map in figure 5.9. The circles with just a capital in them (A to T) are situation models that are the result of the prediction of the disaster. A score could be computed for them that would accurately reflect the success of the proceeding behaviors. The lines that go between the situation models are the choices that can be made. The solid lines go to situation models labeled $E36$ and $E39$ because they are the strategies that for extinguishing. The strategies that would go to the refuge to refill water or the ones that would have the agent patrolling the sector are dotted and they do not lead to situation models. With a width of two, they do not make it to the best two behaviors, because extinguishing is preferable according to the heuristic scoring mechanism, as long as there is enough water and buildings to extinguish. The arrows represent the disaster prediction, which is not a choice. It increases or decreases the predicted flames in buildings based on the previous size and the amount of extinguishing in that building and tags all agents to be allowed a new action. The strategies generated by all three of the tested game trees are contained within this one. We will now discuss how and why these were chosen and what that means for cooperation.

Strategy	Step 1		Step 2		Score
	42	43	42	43	
T	E39	E39	E39	E39	0.785556

Table 5.2: *Strategy T as selected by a tree with width* 1 *and depth* 2*.*

With a width of one the game tree does not allow for any options but to select the behavior with the highest heuristic score. Table 5.2 shows this strategy. This would mean only one of the extinguish behaviors is considered, namely the one on the building with the highest fire risk. Although we tried to make the buildings in the map as equal as possible, they are all the same size, building 39 was placed slightly closer to its neighbors, causing it to be more risky. Therefore it was chosen by both agents as a target in the starting situation resulting in a strategy that looks like the agents are cooperating. This is however not the case. Imagine a slightly more complex situation with a team of four agents. We will not detail this situation here because the game tree is huge. In this situation however there are again two burning buildings, building A has a slightly higher fire danger than building B. Because the heuristic score for extinguishing building A is higher and no other options are considered all agents will target building A, even if it can be predicted that only three agents are needed to extinguish this fire immediately. The agents may be picking the same target, but they are not cooperating. It would be better cooperation if the fourth agent, or whoever was in a better situation to do so, would start at getting building B extinguished. In this case however the strategy is selected where both agents extinguish building 39 in this cycle and since it is not predicted that this will put out the fire in one turn they plan to continue extinguishing it at the next turn. In fact agent 42 will have to move to get close to building 39 first so the first turn only one doze of water will be used on the building.

Strategy	Step 1		Score
	42	43	
B	E36	E39	0.767333
D	E39	E39	0.764222
A	E36	E36	0.764222
C	E39	E36	0.761111

Table 5.3: *Strategies as generated by a tree with width* 2 *and depth* 1*.*

Now the case where cooperation actually breaks down is when we do allow the agent to choose, but limit its ability to plan ahead. With a width of two and a depth of one the agent can only see as far as the strategies A, B, C and D. Selecting the most dangerous building to extinguish is the first option considered because of the higher heuristic score, but when the prediction is done it turns out that either of the agents will have to spend one cycle moving towards the furthest building to accomplish this. This means less water is used and in the agents' shortsighted opinion flames will get bigger than they would be if each agent just starts spraying the one building that is within extinguishing range. The strategies are shown in table 5.3. When we ran the entire simulation the end result was not much worse, because the buildings are kept small so the two agents would always be enough to get the buildings extinguished. In fact if they split their efforts they will be able to extinguish both buildings only one cycle slower than when they cooperate. It would however be easy to imagine the agents not being able to offset the rate at which the fire spread by themselves causing a catastrophical situation.

A game tree that looks ahead one step further will see that the loss of moving at the first cycle is compensated for by the combined effort on the same building in the second cycle. The two strategies that either plan to extinguish building 39 together or 36 receive exactly the same score. The former is chosen because, again, according to the heuristics building 39 is more dangerous and therefore this strategy is considered before the latter and chosen as the best one. There are more equivalent strategies like the ones that have both agents move to the one furthest away and then after that not cooperate on the same building. Those are the two at the bottom of the list. This is of course because the map is almost completely symmetric. The only strategy that has no mirrored counterpart is J in third place, which is basically a continuation of strategy B that got chosen by the game tree with a depth of 1. It does not have the benefit of cooperation but it still has a good water output because no time is lost moving.

Strategy	Step 1		Step 2		Score
	42	43	42	43	
T	E39	E39	E39	E39	0.785556
E	E36	E36	E36	E36	0.785556
J	E36	E39	E36	E39	0.768445
L	E36	E39	E39	E39	0.765445
I	E36	E39	E36	E36	0.765445
R	E39	E39	E36	E39	0.764833
F	E36	E36	E36	E39	0.764833
K	E36	E39	E39	E36	0.762445
S	E39	E39	E39	E36	0.762000
G	E36	E36	E39	E36	0.762000
Q	E39	E39	E36	E36	0.761833
H	E36	E36	E39	E39	0.761833
P	E39	E36	E39	E39	0.761333
M	E39	E36	E36	E36	0.761333
O	E39	E36	E39	E36	0.761222
N	E39	E36	E36	E39	0.761222

Table 5.4: *All strategies for a game tree with width* 2 *and depth* 2 *sorted by preference.*

To evaluate how well our approach scaled, we pruned the game to sixteen different sizes and the time required is shown in figure 5.11. It is apparent from these results that the evaluation of adequately sized game trees takes too much time. To make an estimate of what sort of game tree we could use in a competition simulation we have calculated the average computation time per node. This can be calculated from the same data set that was used to make that figure by dividing the recorded times by the total number of nodes in each of the game trees. It was found to be about 1.477 milliseconds with a standard deviation of 0.954 milliseconds. We suspect this rather large standard deviation is mostly caused by the difference between predicting the disaster and predicting behaviors. If we use the standard rules where the entire agent system has five hundred milliseconds to react and has to provide a reaction for an average of twenty-five agents, we have twenty milliseconds for each agent. With our estimated time per node the agent can evaluate a game tree of thirteen nodes. As we explained in section 5.2 the size of the game tree depends on the pruning width, the pruning depth and the size of the team the agent is in. Notice that the size of the team is not directly related to the number of agents in the simulation, but a team size of thirteen, ten fire agents and three police agents to help them, could very well be needed to extinguish a large fire cluster. Since we have to predict the result for each of the members of the team, even considering one action for each member would spent our entire number of allowed nodes. This means the pruning width would be set to one and the pruning depth as well.

There is not a lot more that can be done to bring this time down. The

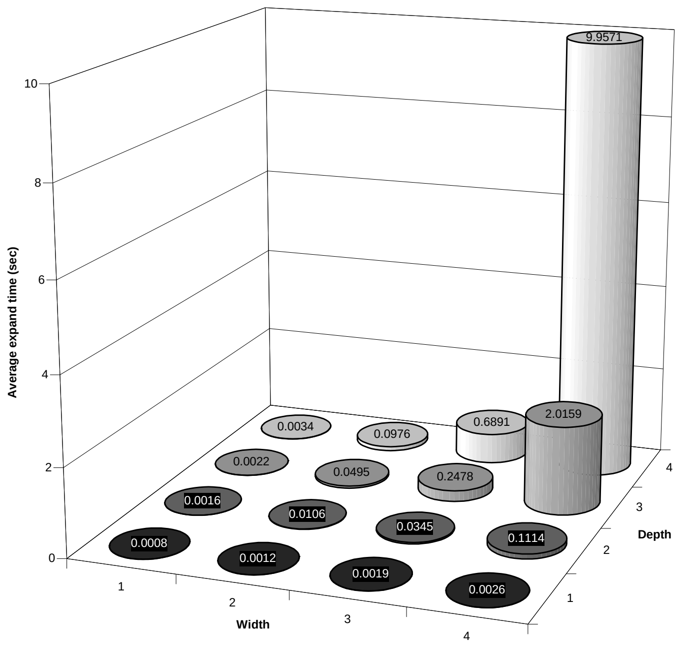


Figure 5.11: *The time in seconds needed to construct game trees of various sizes on a 2Ghz system using the map shown in figure 5.9 over twenty cycles.*

most time consuming operation in evaluating an action, path planning, has been effectively neutralized as shown in section 3.3. There may be other possibilities for optimization that we did not try yet which may allow us to create a bigger game tree. Also with computers getting faster and the allowed computation time during competitions being increased, as was the case during the world championship in Italy, it may be possible to reap the benefits of using game trees eventually, but for now we will have to use a laboratory situation to show what those benefits are.

5.5 Conclusion

Despite our various efforts to speed up the evaluation of a game tree, it is still too slow to use in a competition. We expect that with a couple of more improvements, some of which will be discussed in chapter 6, game trees will be a viable option for the RoboCupRescue problem. For other multi agent systems that do not have the same problems and requirements it may very well be an excellent solution.

We have however shown that it is possible to use game trees to promote cooperation and to avoid making shortsighted decisions. Agents that consider their team members' efforts adjust their strategy to benefit a common goal, even disregarding strategies that would make them as a single agent more effective.

Aside from confirming our hypothesis we have given various ways of structuring the decision making process that could be beneficial even without using game trees. Combining movement and rescue actions in behaviors, being able to predict the result of those behaviors and being able to compute a score for those behaviors could be enough to create a decent decision making system by itself, although the scoring methods would have to be extended to cover the cooperation and long term prediction benefit that we intended to cover in our game tree.

Chapter 6

Discussion

In this chapter we will have a look at work done by others and will compare where appropriate. In doing so, we will retain the same structure of the previous chapters: First, different world modelling approaches will be treated after which communication schemes are explored. Finally several ways of coordination are investigated. The comparison material consists of what other RoboCupRescue teams have done, and work related to multi-agent systems.

6.1 Modelling

6.1.1 Path planning

Two aspects of path planning are important in a RobocupRescue multi agent system. How fast a path can be found and how the possibility of blocked paths are handled. In section 3.3 we described that all our paths are calculated before the first cycle of the simulation, sacrificing a lot of system memory in favor of speed. There is no faster way of obtaining the shortest path than looking it up. The Black Sheep team stores the shortest path as well, but only after the path between the two specific positions has been searched using D^* when it is first requested [26]. This way the paths are cached for reuse, which of course saves storing all those paths who are never used during a simulation. However in light of our agent behavior, we expect the likelihood of needing the exact same path twice to be so small that this approach would not be beneficial enough for our extreme need for fast path planning. Other teams like ResQ Freiburg [10] and S.O.S. [6] just plan their paths when required using A^*, D^* and Dijkstra's algorithm [29].

There is one big problem with storing the shortest path. This path can be in three states: known to be clear, known to be blocked or unknown whether it is blocked or not. Depending on the task of the agent it can be required to choose either the shortest path regardless of whether it is blocked, the shortest path that is known to be free or the shortest path that is not known to be blocked. A police agent can easily clear a way for itself, but an ambulance agent on the way to a dying civilian should not take the chance of running into a dead end. Path planning approaches who do not store their paths, can always use the most recent information to simply discard passages that are blocked or not known to be free to acquire a path with the desired state. The Black Sheep approach has combined the advantages of storing and just in time checking by storing three paths between two positions: one for each possible state. A short one, a certainly clear one and a possibly clear one. Because a shorter clear one can be found when new information is processed or a possibly clear one can be found to be either blocked or clear the possibility of hitting a previously computed path in the lookup table becomes even smaller. Should we follow this method of keeping three copies of paths, but for *all* paths it would require the use of at least three times as much system memory, since the shortest path is equal or shorter than the other two possibilities. This is not desirable, but it is not even feasible to keep the lists of clear or possibly clear paths up to date during the simulation, where new information about roads is constantly collected. There are simply too many. In general our agents ask the shortest path to a target. If the shortest path is clear then they will be able to move to their target and fulfil their task. If the shortest path is blocked they will simply find something else to do, favoring speed over priority and rescue actions over movement. This approach is used under the assumption that in the small space of the sector it is likely that the road will be cleared faster, than that a clear detour is found. A lot of the time the agents will only move a few streets to the next burning building or the next blocked road so the feasibility of wandering all the way around without the perceived situation changing and requiring a new task would not be enough to warrant such methods. For travelling larger distances we have created the specific travelling behavior, that can count on cooperation of police agents. This appears to be sufficient, but it would be possible to create variations of this behavior that implements a more stubborn need to get to a destination by finding ways around roadblocks.

6.1.2 Fire clusters and fire fronts

All RobocupRescue teams recognize the importance of containing the fire by fighting the fire front of each fire cluster. Detecting the fire clusters and choosing the most important one is often done in more or less the same way, by regarding the risk of spreading and the danger to civilians with an approximation of the effect on the total score. Our stores a fire value per sector and in that way describing a fire cluster as a sector with a degree of burning is in fact one of the oddest approaches. One fire cluster covering more sectors is seen as separate problems and two fire clusters in one sector is seen as one problem. A good team assignment strategy should have no problem adjusting for this unintuitive way of describing fires. A fire cluster over two sectors simply requires two smaller teams to combat instead of one large team. Solving problems like these would be an objective for a more sophisticated task coordination, which we will discuss in section 6.4.

To prevent a fire from expanding, which is of course bad for the score and makes it harder to contain the fire in the future, the agents have to pick their buildings to extinguish first carefully. Often this is done by assigning a priority to expected targets that is a lot like our fire danger. Not ignited buildings in the infective range of a target increase the priority. The Black Sheep team also tries to create a containing front by extinguishing neighbors of other extinguished or burned out buildings by increasing their priority. A couple of teams use a neural network to approximate the complex relation between situation and priority. These teams are Eternity [8], S.O.S [6]. The others use a linear function like our fire danger values (Section 3.4) to tweak the priority and focus on containing the fire as fast as possible

6.1.3 Sectors

Like the UvA team, the S.O.S team [6] thought of using a grid structure for the map. However in our approach the objects of the map itself are used for the mapping process of the grid, whereas the documentation of S.O.S suggests the use of a simple grid with evenly sized cells. They see the possible differences in object density of these grid cells as a major shortcoming since it results in an 'unfair task assignment' and a 'decrease in search efficiency'. Indeed the same shortcomings our sector algorithm suffers from, though it can easily be modified to compensate for them. We could for instance create a finer grid and merge sectors to compensate for less densely occupied areas. Also our grid structure features highways which can be used for active searching efforts. Eventually the S.O.S team abandoned the grid idea and opted for heuristic search methods.

6.2 Communication schemes

A key feature of the communication scheme that was described in chapter 4 is that the reception of messages is well defined. By hiding communication details, like the need to select messages upon arrival, one can fully concentrate on the problem at hand. Moreover, this level of abstraction simplifies the design of deterministic algorithms and enables us to implement common knowledge. In our communication model we had to cut down on bandwidth to achieve this.

6.2.1 Randomized gossiping

We will now look at a different approach that was suggested in a paper by Chaturvedi [11] et al. Their communication method is also capable of dealing with limited numbers of outgoing and incoming messages, however message size and message format limitations are not considered. As will become clear, their method allows the use of all available bandwidth but requires more cycles for distribution of information.

They describe an all-to-all broadcast mechanism in which each agent creates two sets per communication round. The first set contains randomly selected agents to which messages will be sent, whereas the second set contains randomly selected agents from which messages will be received. The maximum sizes of these sets are determined by the outgoing and incoming bandwidth capacities. An agent creates new sets, sends and receives until it directly or indirectly has received the messages from all the other agents. For this to work, each agent has a list of tuples with the identity of an agent together with its broadcast message. By constantly broadcasting the list in the send phase, the receivers can update their own lists by a simple merge process. In this way it is possible that an agent is indirectly supplied with messages from other agents.

In the method just described, messages are disseminated at a high rate because agents retransmit the messages of the other agents in subsequent communication rounds. However, extra rounds are introduced by the random nature of the algorithm. When fully utilizing outgoing bandwidth, the receivers may not be able to accept all messages. This problem is solved by randomly selecting the messages to accept. Therefore it may take additional cycles for each agent to receive all the messages.

The results of the paper showed that *on average* distribution took 7 communication rounds (simulation cycles) for 128 agents and a bandwidth setting of 4 for both incoming and outgoing. Furthermore, it was claimed that the randomized gossiping algorithm was able to operate at, or very close to $O(^2\log n)$, n representing the number of agents. So the gossiping method performs quite well, even with large numbers of agents. When comparing these results to our communication method, a few things should be kept in mind.

- In the above gossiping approach, each agent is assumed to have the same bandwidth limitations, whereas our method has different limitations for different kinds of agents. For example, in our approach the incoming and outgoing bandwidth capacities of the centers depend on the number of platoon agents.

- The senders and receivers in the gossiping scheme are determined randomly, whereas in our scheme they are not. This allows us to fix the number of simulation cycles required for distribution.

- In our approach additional restrictions apply to the message size and format, making it difficult to condense several messages from other agents into one message.

6.2.2 Asynchronous communication

So far we have only considered communication schemes in which information is distributed in a synchronous way. At the other end of the spectrum we find the approach followed by Roth [25] et al. The communication system they describe suffers from high communication latencies. To compensate for this, they chose to abandon synchronization, thereby eliminating any synchronization overheads involved. In our communication method this necessitated two additional simulation cycles for each communication cycle. In their approach, shared information is seen merely as a mechanism to fall back on whenever local observations do not suffice, or when it is known that the combined information is more accurate. Just like in our example in section 4.3.3, in which increased knowledge was maintained on top of local knowledge, they also employ two separate knowledge models. We quote:

> *"We introduce an approach for representing the world with two separate world models: an individual world model that describes the state of one robot, and a shared world model that describes the state of the team."*

They conclude that in spite of high latencies, they were able to significantly improve the knowledge of their agents. Still these high latencies are seen as a limiting factor, as lower latency communication may allow the implementation of 'merging observations'.

6.2.3 Other teams

Now that we have attempted to draw up a classification of communication methods on the basis of synchronization, we would like to show how the ideas of other teams fit in. To this end we have made a subselection of all the entries of teams participating in the RoboCupRescue competitions of 2003. In doing so, it became clear that most teams have adopted the idea of using some central place to route information through, but payed less attention to synchronization aspects. We will now briefly discuss what we thought to be interesting approaches towards communication.

6.2.3.1 Black sheep

The black sheep team solved the problem of the bandwidth limitations by using a priority queue. Firstly, they defined the following message types:

- Need rescue (for trapped agents)

- Road request (clear request)
- Road update (property updates)
- Building update (property updates)
- Civilian update (property updates)
- Building searched (for trapped civilians)

Secondly, they attributed a priority value to each message type, allowing for proper scheduling. In order to deal with the outgoing message limitations, they reduce the priority queue by four messages upon each transmittance. The incoming limitations are dealt with by only allowing the platoons to listen to their centers, while the centers can listen to both their platoons and other centers. However, it was however not clearly explained how messages were selected upon arrival. In addition only AK_TELL messages were considered and no explicit communication sequence was described.

6.2.3.2 Rescue freiburg

The communication method of the German Freiburg team has some similarities to ours. In their approach the centers also collect messages from the platoons, but these are immediately transmitted back to the other centers and platoons of the same type. So our method differs in that we have a second stage where another message merge is performed. In the first stage, messages are merged as soon as a center has collected the messages from its platoon agents. In the second stage the messages are again merged when the centers broadcast to each other before reporting back to their platoons. By using two stages, platoon agents belonging to different centers can exchange communications by means of the distribution sequence. Moreover, during the second stage we allow platoons to communicate locally.

The Freiburg team also mentions a second communication mechanism in which centers may use the knowledge gained to give commands to their own platoons. Finally, their documentation did not seem to cover any synchronization solutions.

6.3 Different approaches for coordination

6.3.1 Forming teams

Our agents always try to form teams. They do not literally construct them, but assume that the agents who are in the best position to assist them will do so. As can be expected most other RobocupRescue participants are a lot more explicit in making groups. Some will have agents call for assistance directly to other agents (YOWAI [18]), others will send their requests for help to the center first, which selects the most important ones and commands the other agents to help the requester (SOS [6]). The last possible option is that only the center both recognizes the need for a team and organizes it (ResQ Freiburg [10]).

Unlike our, most other approaches have the option for agents to work individually. This is mostly because their tasks are more specific. These tasks combine extinguishing efforts on one building or a set of buildings. Our tasks cover all the main and support behaviors. This also means that our teams are heterogenous. Police agents in other approaches usually just react to requests by fire agents to clear a road that is in their way. They don't remove the road, because they are aware of the reason why the fire agent wants it cleared. This makes it hard for the police agents to prioritize the requests.

The most remarkable difference with other approaches was that they all have different grouping methods for the different types of agents. Fire agents always need a way to agree to cooperate, since extinguishing a building alone is nearly impossible. Most teams had their police agents handle requests solo. Some teams, like SOS, have the option for ambulance agents to cooperate if needed for the survival of the civilian by computing the digging speed needed to free a civilian before he dies. ResQ Freiburg tries to divide the ambulances as evenly as possible over all the civilians with a chance to survive. Our team approach was designed to decide per situation, like the former approach, but not by using a computation but by predicting the result of combining effort.

6.3.2 Behavior selection

The fact that our agents construct a strategy for multiple future cycles instead of just selecting the behavior which currently has the highest priority is the major difference with all other RobocupRescue participants. There is no other team that we know about that uses this method, therefore it is hard to compare our method to others. We can simulate approaches that do not plan ahead by pruning our game tree to only look one step ahead. We then would only be constructing a game tree to evaluate the different possible ways to cooperate in the next cycle. If we would put every agent in its own team our game tree would have the current situation as root node and only the actions of this one remaining agent as branches. The selection of the best of these branches would be based only on the direct impact they would have on the disaster. This would be comparable to what other participants do, but they have more complicated functions to determine the effect of the action.

ResQ Freiburg and Roboakut [13] for example use reinforcement learning to select the best target. These algorithm have learned to associate a couple of features of the situation to the right action. The learning algorithm should after a while be able to predict the long term effects of an action. Actions that in the past led to a good score are encouraged and the ones that led to a bad score are discouraged. The factors that this learning algorithm will discover can also be explicitly engineered by the RoboCupRescue participant based on observation and common sense. Priority scores can be computed based on the danger a burning building poses for its neighbors or the time left before a buried civilian will die. We use similar approximations of the effect of a behavior as an heuristic hint to optimize the pruning of the game tree (see section 5.2). Our theoretic single layer game tree would therefore be comparable to other approaches, yet too simple to be used in a competition. We rely mostly on the game tree to predict the long term effect, so the heuristic hint is very simple, often based on only a single property of the target. Most teams that use the priority approach have found various ways to improve estimating of importance, for example by categorizing tasks as urgent or not. Clearing a blocked road that has another agent trapped would be more urgent(SOS) and so would saving another agent from a buried building (Black Sheep). Some use small neural networks to map features in the object pool to priorities as a hybrid approach, like fire danger and civilian survival chance, as a hybrid approach between learning and common sense.

6.4 Future work

6.4.1 Communication teams

Before designing the present communication model, the authors of this thesis considered various alternatives. We believe that these alternatives may prove useful in the creation of a completely different communication model. To illustrate this, we will take a brief glance at the history of this project.

The first communication model we considered was documented in a UvA team description [21] and involved platoon agents as 'communication managers'. These managers were given the task of maintaining contact with both team members and their centers; they acted as gateways between the platoons and centers. The proposed hierarchical structure has the disadvantage of the managers becoming overloaded with information. It became clear that by having communication managers relay the information of others, simulation cycles would be wasted that otherwise could be used to communicate directly. At that point another idea emerged. To increase the information value of a message, communicating agents could be enabled to remember with whom they had already communicated. The basic assumption being that the information value of a message would increase with the passage of time since the previous exchange. To implement this a 'buddy list' was considered, updated on the basis of time-stamps and possibly other data such as team membership. During the send operation only the messages with the highest priority would be transmitted.

It is conceivable that the above ideas can be used to create an entirely different communication model as compared to the model we currently have implemented. We anticipate a model that focusses less on synchronized distribution of information, aiming more at instant communication between cooperating platoons. It would be interesting to compare such a model with the one described in chapter 4.

6.4.2 Advanced task coordination

We have divided the coordination into two levels and concentrated on the lower level, the level where agents coordinate their short term goals and targets. The long term goals needs coordination as well. Looking ahead can result in great improvements. Examples of this are: sending the police agents towards the next fire cluster first, to prepare it for the fire agents, deciding which fire cluster has priority and which one can be more easily detained at a later cycle and deciding wether it is worth the travel time to have some agents switch to another team.

These features require a coordination approach. Finding out which type of approach is most suited was not one of our research goals. We used a simple system with a couple of production rules to serve as a placeholder. These rules only considered the number of each type of agent and a very small part of the information present in the summary (see section 3.5). Adding and improving the rules to make them aware of time, the characteristics of the fire simulator and the effectiveness of an agent could greatly improve the performance.

Other approaches are imaginable. Looking ahead to evaluate a long term strategy is one of the most important features of a task and team coordination system. As explained in chapter 5, looking ahead to evaluate multiple long term strategies is a task well suited to a game theory approach. It would not be hard to imagine the task and team coordination as a game. The summary serves as an abstract view of the disaster that already resembles a chessboard. The presence of one fire agent in a sector would affect the fire in that sector and the presence of a police agent would reduce the number of blocked roads. Of course putting more agents in the same sector would result in a synergy bonus, for which an approximation could be designed. After a few cycles of simulating the effect of agents and the progress of the disaster an approximation of the official scoring algorithm, using the information present in the summary, could serve to evaluate the used strategy. The strategy of the task coordination game player would consist of moving one or more agents to neighboring sectors each move. The effectiveness of this approach would depend on the computation requirements and the vulnerability to local minima, like our low level coordination.

The assessment of a problem is another important task in coordinating the task and team assignment. The assessment of a problem should of course yield the priority, but also the attainability. A problem that can be solved quickly should be solved before a problem that is more important, but requires a lot of time and effort. This could help to avoid the situation where the small problem grows into a big problem itself, so the agents end up with two big problems. This requires a way to model the passing of time. It is inherently contained in the previously discussed game tree approach, where the strategy that focusses on the big problem first becomes inherently less desirable as the unchecked small problem deteriorates the evaluated score. Another way to for a coordination system to understand the subtleties of task assignment is learning. If given enough examples of a situation where the small problem eventually grows into a big one, many learning approaches will eventually learn to detect other, more subtle, properties of a problem area that determine overall effectiveness. Attainability is just an example of a factor that can come into play. In this way of viewing the disaster as an optimization problem, with a summary with a good score as optimal, algorithms like neural networks and genetic algorithms seem particularly suited.

6.4.3 Game tree communication

In our approach the agents duplicated a lot of work by each computing their own version of the game tree, while the communication channels were reserved completely for passing on observations. Besides that, every cycle the agents start over with a fresh root node of the tree, instead of preserving earlier computations. This approach seems wasteful, but considering the large difference in knowledge between agents or even the knowledge of one agent from one cycle to the next it seemed more sensible. Despite improving the knowledge by making improvements to this communication approach, another radically different approach can be conceived. Most other teams communicate each agent's next action every cycle. This removes misunderstandings of what the agent that receives that information should do to cooperate. This method can be improved upon by not only sending the next action, but a few possible strategies of the cycles after that. This could take the form of a partial game tree with the most promising strategies. Other agents could compare these strategies with their own to figure out which one of the possibilities to actually execute. Also this information could be used to provide pointers on what strategies are worth the computation time to research more thoroughly, giving the tree expanding algorithm an edge that would allow to look further ahead of the first few important cycles. The agents bargain amongst each other providing conditional actions. These conditions could be completely unacceptable to teammembers, like requiring an agent that is trapped under debris to help them. This could lead to other types of misunderstandings and uncoordinated behavior, which would need to be solved for this approach to provide a definite advantage over our current approach. Also it is not sure whether the small message size and communication range is enough to communicate the parts of the strategy required to cover most possibilities. In the rules of the next competition the allowed message size has nearly tripled, which would provide some room for communication and distributed evaluation of game trees alongside the synchronized common knowledge.

Chapter 7

Conclusion

The development of agents requires solutions to three separate problems. There must be reliable systems for the exchange of communications and for the coordination of agents. And thirdly, a reliable world model must be designed. We started out by approaching the communications problem and the coordination problem individually. Then, when designing the world model, we looked closer into their mutual interdependency.

The central idea of our proposed communications system is to provide all relevant agents with the same information, in order to make them perform more effectively as a team. Our communication protocol offers a reliable method for sharing information in a synchronized manner. We have shown that this communication method greatly contributes to the overall amount of knowledge at agents' disposal.

We designed the coordination process to coordinate agent behavior in a way that showed cooperation and understanding of the problem space, without using the communication channels for coming to an agreement. To attain this, we based it on well known principals of game theory while assuming the common knowledge provided by the communication scheme. We created a new layer of qualitatively describing agent actions in the form of behaviors and a new layer of describing the problem in the form of a situation model. We showed that using a game tree to structure an agent's thought process exhibits the advantages we were hoping for, but also that a lot more can be done to improve this approach until it is ready to be used in a competition simulation.

Appendix A

RCRSS specifics

- KA_SENSE messages.
  - During the first 3 cycles agents cannot receive any sensory information at all. They cannot act either.
  - Information about objects within an agent's radius is always sent regardless of the fact that this information might not have changed since the last cycle.
  - The incendiary property of all building objects within range is sent in the 6^{th} cycle. After that we'll only receive updates on this property if it has changed.
  - Agents which are within the radius of the burning object can see it:

    $$O_{radius} = C * O_{burntime}$$

    O_{radius}: Burning object radius.
    C: Cognition speed (by default 10 meters per cycle).
    $O_{burntime}$: Object burn time in cycles.

- Extinguishing fires.
  - The hose length is limited.
  - Extinguish direction matters.
  - Only buildings which are on fire can be extinguished.
  - The building should be close enough.
  - A firebrigade should have enough water. The tank has a capacity of 15 kiloliters, and can be refilled at refuges.
  - Extinguishing and refilling happens with a maximum speed of 1 liter per cycle.

- Evaluation.
  - Hit points for agents are calculated by using a *Damage* property which is determined as follows:

    $$Damage = D_{fieriness} + D_{buryness} + D_{block} + D_{broken}$$

    The agent's hit points are initialised at 10000, and decreased by *Damage* each cycle. When the agent moves to a refuge its *Damage* property is reset to 0.
  - After the simulation has finished the score is calculated as follows:

    $$Score = (N_{agents} + \frac{HP_{remain}}{HP_{init}}) * \sqrt{\frac{AREA_{ok}}{AREA_{init}}}$$

    N_{agents}: Number of agents still alive.
    HP_{remain}: Remaining hit points of all agents.
    HP_{init}: Total hit points of all agents at simulation start.
    $AREA_{ok}$: Area of buildings that are not burnt down.
    $AREA_{init}$: Area of buildings at simulation start.

Appendix B

Agent viewer

Included in the RCR Simulator System is a viewer, which is a program that connects to the kernel to receive all changes to the objectpool, much like a simulator does. This information is then shown in a graphical display so researchers can see the actions of their agents and their effects on the disaster or, during the competition, the public can watch the simulation. There are a couple of different viewers available. Some show the information in a clear and complete two dimensional overview for easy evaluation of the agents, like the one by T.Morimoto [4] and the one by Y. Kuwata [5]. Other viewers are being developed that show the simulation in a more engaging and exciting $3D$ perspective. All these viewers show simply what the kernel tells them, which is the topography of the map in the beginning of the simulation and after that all the disaster changes: burning buildings, blocked roads, building collapses, agent and civilian positions. This makes it easy to see what's going on in the 'real' disaster, but not what agents *think* is going on. Agents may not know about a lot of burning buildings and collapsed roads, which makes their actions sometimes seem illogical. To evaluate the behavior of our agents we created our own viewer who takes its information not from the kernel, but directly from the world model of one specific agent.

Because our agent system usually has multiple agents running in the same executable they can share their viewer. The viewer will only show the world model of one agent at a time, but it can be switched during the simulation. Centers can also be selected. Of this agent different pieces of information can be shown simultaneously. All of these can be turned on or off using a command line parameter when starting the simulation or by pressing a filter key (F1 to shift F12) during the simulation. These *view modes* as we call them are roughly dividable in five categories:

- Object pool information like what is shown in a normal viewer

- Data browser

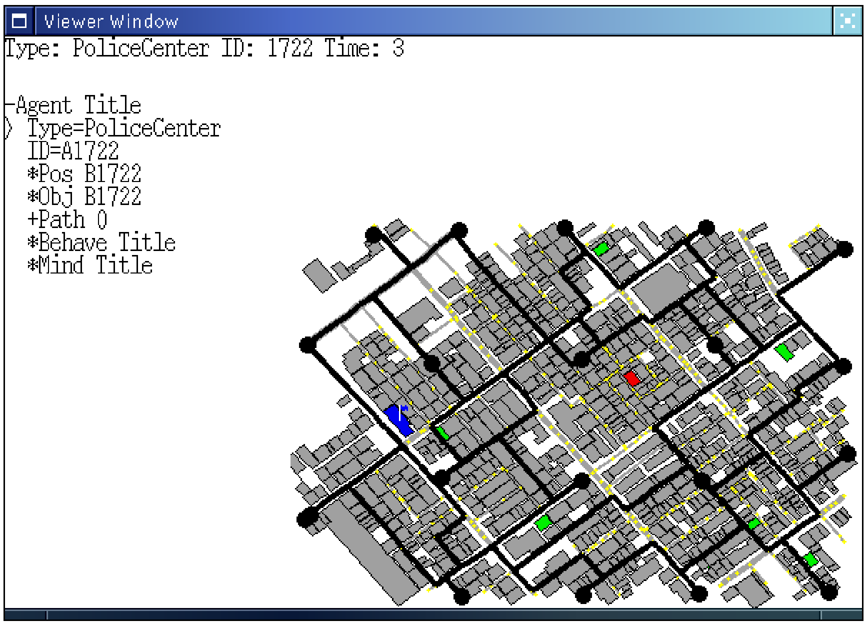
- Sectors

- Error pinpointing

- Coordination structures

The object pool information are simply the locations and condition of buildings, roads, nodes and civilians. Should all of these be turned on our viewer would look a lot like a normal 2D viewer, but it would only show what the agent knew. The data browser is a way to get the details of all the elements that can be shown. In the view of an agent any object can be selected like a building or even another agent. The data browser will then show all the properties of this element. For example its exact position, its neighboring objects or what sector it is in. Most of these properties consist of other properties, which can be selected and expanded to show a tree like structure of all properties of this object and the objects it's linked to.

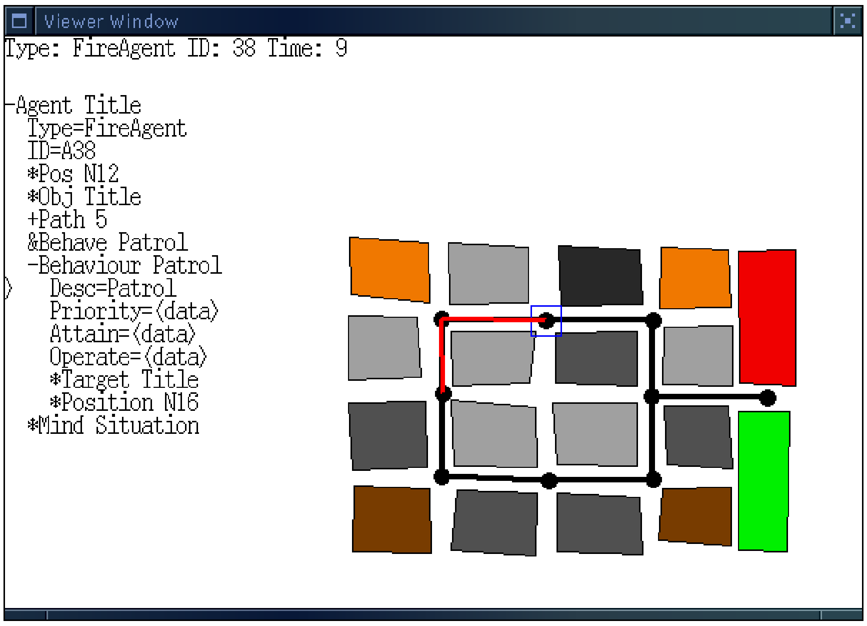
The data browser can also show the properties of high level structures like sectors, or even the current game tree and most of these objects can also be shown graphically. This has helped us often in finding out immediately what was wrong if something undesirable happened, which helps a lot in tracking down bugs and design flaws in a system as complicated as ours.



```
Type: PoliceCenter ID: 1722 Time: 3

-Agent Title
> Type=PoliceCenter
  ID=A1722
  *Pos B1722
  *Obj B1722
  +Path 0
  *Behave Title
  *Mind Title
```

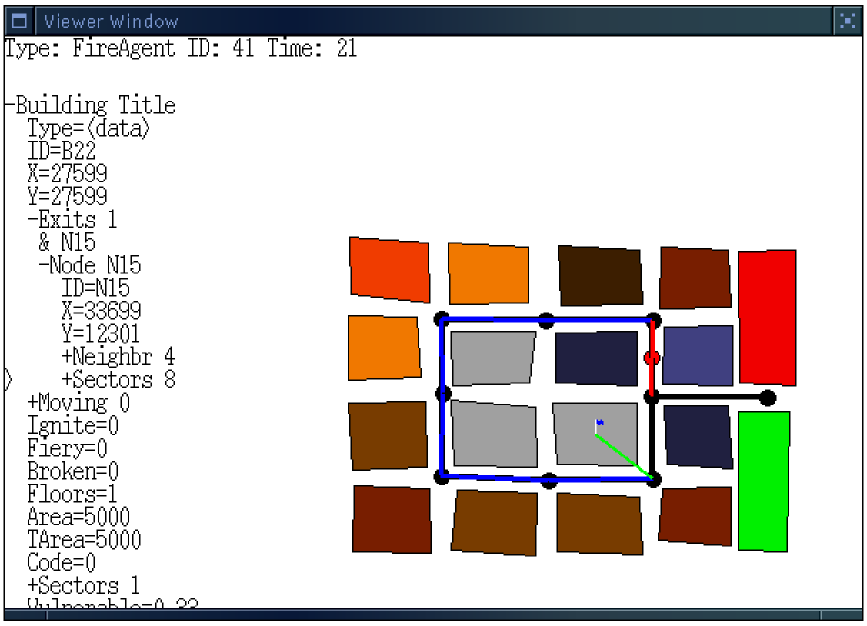
Figure B.1: *The viewer displaying the Kobe map. The thick black dots are the sector grids and the thick black lines are the sector borders.*



```
Type: FireAgent ID: 38 Time: 9

-Agent Title
  Type=FireAgent
  ID=A38
  *Pos N12
  *Obj Title
  +Path 5
  &Behave Patrol
  -Behaviour Patrol
>   Desc=Patrol
    Priority=<data>
    Attain=<data>
    Operate=<data>
    *Target Title
    *Position N16
  *Mind Situation
```

Figure B.2: *The viewer displaying a small test map, with the data browser diplaying a fire agent and it's patrol behaviour.*



```
Type: FireAgent ID: 41 Time: 21

-Building Title
  Type=<data>
  ID=B22
  X=27599
  Y=27599
  -Exits 1
   & N15
   -Node N15
     ID=N15
     X=33699
     Y=12301
     +Neighbr 4
>    +Sectors 8
  +Moving 0
  Ignite=0
  Fiery=0
  Broken=0
  Floors=1
  Area=5000
  TArea=5000
  Code=0
  +Sectors 1
```

Figure B.3: *The viewer displaying a small test map as seen by a fireagent. The data browser is listing the properties of a selected building and the node it is connected to.*

View mode	Mode	Details
Buildings	1	building geometry and their color coded fire state
Roads	2	road geometry and their grey tint coded block
Nodes	3	location of nodes
Agents	4	agent locations
Sense	5	objects that have changed properties in the last cycle are highlighted
Browser	6	display the data browser
Cursor	7	the object that is selected in the browser as highlighted in the map
Borders	9	the sector borders are accentuated as thick edges
Grid points	10	the sector grid points are accentuated as thick black dots
Sector contents	11	all buildings belonging to the same sector are drawn in the same color
Path errors	12	positions that have been found unreachable by any precomputed path are shown
Sector errors	13	objects that are not found to be party of any sector are highlighted
Behavior	8	the goal, target position and name of the behavior being executed are highlighted
Summary	14	Show the risk values of each sector as floating point values
Situation	15	Display the items as represented in the situation model extracted for the root node of the current game tree

Table B.1: *View modes of the agent viewer.*

Bibliography

[1] RoboCupRescue. http://www.robocup.org.

[2] http://www.mathrec.org/old/2002jan/solutions.html.

[3] http://www.chessgames.com/chessstats.html.

[4] http://ne.cs.uec.ac.jp/ morimoto/rescue/viewer/index.html.

[5] http://homepage1.nifty.com/morecat/Rescue/Rescue.html.

[6] Saman Amirpour Amraii, Babak Behsaz, Haamed Gheibi, Mohsen Izadi, Hamed Janzadeh, Farid Molazem, Arash Rahimi, Mohammad Tavakoli Ghinani, and Hamide Vosoughpour. S.o.s. 2004: An attempt towards a multi-agent rescue team. 2004.

[7] K. Binmore. *Fun and Games.* D.C. Heath and Company, Lexington, 1992.

[8] Ali Akhavan Bitagshir, Fattaneh Taghiyareh, Amirhossein Simjour, Amin Mazloumian, and Babak Bostan. Uteternity's team description: Layered learning in robocup rescue simulation. 2004.

[9] M. Bowling. *RoboCupRescue: Agent Development Kit.* Carnegie Mellon University, Pittsburgh, PA 15213-3890, version 0.4 edition, december 2000. http://www-2.cs.cmu.edu/m̃hb/research/rescue.

[10] Michael Brenner, Alexander Kleiner, Mathias Exner, Markus Degen, Manuel Metzger, Time Nüssle, and Ingo Thon. Resq freiburg: Deliberative limitation of damage. 2004.

[11] Alok Chaturvedi, Jie Chi, Shailendra Mehta, and Daniel Dolk. Samas: Scalable architecture for multi-resolution agent-based simulation. In *ICCS-2004*, volume 3038, 2004.

[12] R. de Boer and J. Kok. The incremental development of a synthetic multi-agent system: The uva trilearn 2001 robotic soccer simulation team. Master's thesis, University of Amsterdam, february 2002.

[13] Baris Eker and H. Levent Akin. Roboakut 2004 rescue team description. 2004.

[14] M.L. Fassaert, S.B.M. Post, and A. Visser. The common knowledge model of a team of rescue agents. 1th International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster, Padova, Italy, 6 July 2003., 2003.

[15] W.R Franklin. http://www.ecse.rpi.edu/Homepages/wrf/research/geom/pnpoly.html.

[16] F.C.A. Groen, M.T.J. Spaan, and N. Vlassis. Robot soccer game or science. In M. Ivanescu, editor, *Proceedings CNR-2002*, pages 92–98. Editura Universitaria Craiova, October 2002. ISBN:973-8043-165-5.

[17] A.D. De Groot. *Thought and Choice in Chess.* Mouton, The Hague, 1965.

[18] Masahiro Kyougoku and Ikuo Takeuchi. The team description of yowai2004. 2004.

[19] T. Morimoto. *How to develop a RoboCupRescue agent*, 1st edition, version 0 edition, 2002.

[20] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, second edition, 1998.

[21] S.B.M. Post, M.L. Fassaert, and A. Visser. The communication reduction approach of the 'uva rescue c2003'-team. http://www.science.uva.nl/ arnoud/research/roboresc, 2003.

[22] S.B.M. Post, M.L. Fassaert, and A. Visser. The high-level communication model for multi-agent coordination in the robocuprescue simulator. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida (Eds.), editors, *Lecture Notes on Artificial Intelligence.* Springer Verlag, Berlin, 2003. RoboCup.

[23] David V. Pynadath and Milind Tambe. Multiagent teamwork: Analyzing the optimality and complexity of key theories and models. In *In Proceedings of the 1st conference on autonomous agents and multiagent systems (AAMAS-2002)*, 2002.

[24] E. Rich and K.Knight. *Artificial Intelligence.* McGraw-Hill, Singapore, 1991.

[25] Maayan Roth, Douglas Vail, and Manuela Veloso. A world model for multi-robot teams with communication. In *IROS-2003*, 2003.

[26] Cameron Skinner, Jonathan Teutenberg, Gary Cleveland, Mike Barley, Hans Guesgen, Pat Riddle, and Ute Loerch. The black sheep team description. 2004.

[27] D. Sunday. http://softsurfer.com/algorithms.htm.

[28] T. Takahashi. *Tools for checking & creating ***polydata.dat files*, February 2002. ttaka@isc.chubu.ac.jp.

[29] T.H.Cormen, C.E.Leiserson, and R.L.Rivest. *Introduction to Algorithms.* MIT Press, McGraw-Hill, New York, NY, 1990.