Teaching Multi-Agent Systems with the help of ARES: Motivation and Manual

Melissa Bergen Jörg Denzinger
Jordan Kidney
Department of Computer Science
University of Calgary

November 7, 2002

1 Introduction

In the last years, multi-agent systems (MAS) have become a very active research area that has connections to many other areas, both inside and outside of Computer Science. Consequently, courses about MAS are starting to be developed and even the first text books are on the market (see [AL02] and [Wo02]). Perhaps even more than in other areas of Computer Science, teaching MAS has to involve practical experiences by the students. The interaction of agents has many surprises (as has the interaction of human beings) and hands-on experiences with issues like timing of actions to achieve cooperation, communication and the effects of it, changes in the surroundings, and so on, are necessary to let students understand not only the basic problems but also why certain concepts are the way they are.

For getting practical experience with developing multi-agent systems the students need an environment (or testbed) in which their agents will interact, that sets the basic rules for the agents and guards these rules against violations by the agents. There are already such environments available, namely the environments used in various competitions, as for example the RoboCup Simulation League Soccer Server (see [Ku02]) or the TAC Game Servers of the Trading Agent Competition (see [We+01]). But, for teaching purposes, the goals for the development of these environments do not totally agree with the goals we need for a teaching environment: having successful systems available via WWW allows for a lot of cheating (resp. requires a lot of work of the instructor spend on counter actions) and results in students not making the experiences they are supposed to make. Also, the two cited environments are rather specialized, so that certain experiences are outside of their scope. There are a lot of other didactic reasons for not using testbeds that were developed and are used to evaluate research systems, as we will see later in this report.

In this report, we present the ARES system (**A**gent **R**escue **E**mergency

Simulator) that is intended to be a testbed for multi-agent systems and to be used for teaching MAS. ARES follows the lead of the RoboCup Rescue Initiative (see [RR02]) in choosing as the application scenario rescuing survivors in a disaster zone. The basic tasks the students have to include into the agents that form their multi-agent system that is employed within ARES are locating survivors and removing rubble to reach and rescue those survivors.

ARES allows for many different variants of the basic setting, by varying the information the agents have when starting, the cost of communication, the methods for agents to regain energy, and so on. While many basic requirements of acting in a real disaster scenario are touched, nevertheless they are simplified within ARES towards a game-like scenario that allows students, resp. student teams, to develop agents that act as team in ARES in the 4 months a beginners course in MAS takes.

This report is organized as follows: After this introduction, we take a closer look at the requirements on a testbed for MAS (resp. MAS concepts) that is aimed at helping in teaching MAS basics. This then leads to stating our goals in developing ARES. In Section 3, we present the system using two different views: the view of a student using it and the view of an instructor configuring ARES for his/her course (and we will also provide some information about the implementation of ARES). In Section 4, we present observations we made when using ARES for teaching MAS to a mixed class of graduate and undergraduate students at the University of Calgary. Finally, we will conclude with some remarks on future work. The report also contains as appendices descriptions of the actions ARES allows agents and their syntax, of the graphical viewer that allows building scenarios and observing a rescue team, of the installation requirements and procedure, and of the parameters for defining the "world laws".

2 Our Goals in Developing ARES

The term *agent* is used by many people in many different areas and even if we look at its uses just in Computer Science we still see rather different definitions of it. In fact, as pointed out in [SV00], the only thing people agree upon regarding what an agent is is that they do not agree. If we want to develop an agent testbed to enhance teaching MAS, then this problem is less severe, since the agents will be defined by the students based on the different definitions presented to them in the lectures. Nevertheless, there are two extremes regarding multi-agent environments in literature and a lot of systems in-between: cooperating agents with a common goal and competing agents with rather different goals. While we present concepts and examples for both extremes and some in-between settings in our course, nevertheless we think that for the assignment of the course –that is implementing a multi-agent system acting in our testbed– we have to make a decision between a common goal for all agents and selfish agents. Since we see at the moment more to be gained by developing very cooperative systems, we have chosen the common goal, which is also the situation the student teams are in (or should be).

For deciding which general problems the agents developed by the students have to solve cooperatively, we did not have to look far. The reasons layed out in [Ki00] by the RoboCup Rescue initiative are very good and convincing and the idea of robots taking on risky tasks in dangerous environments is with us since the beginning of Robotics (and even before if we look into science fiction). As for precisely what tasks we want to have in our testbed, there are a lot of things to consider from a didactic/teaching point of view:

- Teams of students have to be able to implement a team of agents acting in our testbed (i.e. the ARES system) within the four months the basic course in MAS takes and we can only assume that these students know how to program in general (and that they take the course concurrently).

- We want the students to apply the basic concepts they are supposed to learn in the course. This means that the requirements of ARES have to be in such a way that there is not much emphasis on implementation tricks and fine-tuning of the agents (although we will never be able to totally avoid this).

- The ARES testbed and the tasks to fulfill in it have to be easy to handle for students. This means that there has to be enough detail to make the tasks reasonable, but also enough abstraction and simplification to easily grasp the application area and the "laws" of the world the agents act in.

- Naturally, motivation is a big factor here. Our first three points already are important with regard to the motivation of students in that if they are not fulfilled the students' motivation will suffer. But there are other factors that can affect the motivation. One important factor is not to use testbeds for competitions of research systems, especially if these competitions have already been held several times. Naturally, the research systems that are successful in these competitions are the results of a lot of effort by the groups producing them and they often reflect a lot of experiences made in earlier competitions. Systems by students doing their first steps in MAS will not produce results that even come close to the competition winners. Also, making not-so-obvious mistakes is a good learning experience.

  On the other side, having the student teams compete with each other often is a plus with respect to motivation (especially if the competition is not about grades but something else; although the best system should be guaranteed an A).

- Finally, for an instructor it is very important that there are variations in the assignments for a course over the semesters. We are not only talking about avoiding cheating, although obviously this has to be a concern, but also about motivation, again, and having a research experience. If each class has to deal with a different "world" their agents act in, with different tasks and laws governing it, even just hearsay from previous classes does not have a lot of effect on what the students will try out, while without

such a difference the systems of a class tend to cluster around the concepts that won the competition of the student teams the last semester.

In addition to these didactic requirements/goals on/for our testbed, naturally there are requirements with regard to the MAS topics that we want the students to learn by implementing agents acting within the testbed:

- As already stated, we want the students to learn about cooperation of agents and they should have to deal with the problem of planning in and for a MAS, together with cooperative decision making and finding a good organization.
- Communication and its costs are a very important factor in many multi-agent systems. Very often the concrete communication costs (including the amount of processing needed to understand what is communicated to an agent) decide which cooperation concept is best suited for a certain application. Students have to become aware of this. In our testbed, we want to be able to vary communication costs as part of the world laws.
- Not only in a rescue scenario, but in many other applications, acting in real-time is a basic requirement. Unfortunately, full real-time requirements make the systems much more complex and turn the development focus to subjects not at the core of MAS. Therefore we see them as beyond of what the students should be dealing with.

  Nevertheless, we think that some real-time aspects should be included into the students' task and therefore organize the world within ARES as a sequence of turns (points in time, events, or moves) and each turn stands for simultaneous actions by all agents. And for the decision what to do in such a turn, the agents have a limited amount of computing time available.

Our goal in developing ARES was to fulfill the requirements stated above to the highest degree possible. Therefore we have as the basic setting that ARES presents to the agents acting in it a search and rescue mission. In each particular scenario, a certain number of survivors more or less near death is placed on a grid world and many of them are buried under layers of rubble, where each layer may require the combined effort of several of the agents to be removed. The goal for the agents is to rescue as many survivors (while they are still living) as possible within a given amount of time, i.e. a given number of turns. The agents developed and implemented by the students will be given several such grid worlds and the grading and winner determination is based on the performance in all these example worlds.

3 The ARES Simulator System

As already stated in Section 2, the ARES simulator system (multi-agent testbed) puts a group of agents (that supposedly control robots) with the same capabilities into a world that is an urban disaster zone, more precisely, we envision a

city struck by an earthquake. The world is structured into grids that describe the potential zone of direct influence a single agent has. At the moment, there are 4 types of grids:

- instant killer grids (think of them as holes our agents fall into and vanish),
- fire grids (that might spread in later versions),
- recharging grids, and
- "normal" grids consisting of a stack describing layers of rubble and survivors.

Naturally, the goal of the agent team is to rescue as many survivors as possible in a given amount of turns. But in different "worlds", the definition of rescued can differ.

Starting with some initial information about the world (that is more or less accurate), the agents have in general to perform a search and rescue operation where cooperation is more or less part of both search and rescue. In the later, cooperation is needed, because some pieces of rubble need more than one agent to remove them.

In the following subsections, we provide more detail about ARES from 3 different perspectives. First, we describe ARES for a student taking a basic MAS course and working with other students to implement agents running within ARES. Then we take the perspective of an instructor and look at the world features ARES allows to manipulate in order to set different foci for courses. Finally, we take a look at the implementation of ARES itself and provide information that, together with the documented source code, should allow people to modify ARES to include additional features or additional feature instantiations.

3.1 The student view

For a student, the assignment starts with downloading and installing the ARES system on his/her team's account. At the moment, ARES runs only under UNIX (more precisely Sun OS) and has both C++ and Java components (see Appendix C). The agents the student teams will develop are, from the point of view of ARES, processes that the ARES kernel is communicating with. As a result, agents can be implemented in whatever programming language the students want to use, but it has to be taken into account that ARES interrupts and restarts the agent processes in order to limit the amount of processor time available to an agent each turn, so that a certain efficiency of the agent processes is definitely required.

The main facts students have to make themselves familiar with are on the one hand the information that ARES sends to an agent after each turn and on the other hand the commands that an agent can send to ARES. While the exact details of the syntax and semantics of these messages can be found in Appendix A, in the following we will give a short overview of both message

sets. An agent first has to CONNECT to the ARES system to tell it that it will participate in a run. Then an agent can MOVE from its current grid to a neighboring grid, loosing as a result a certain amount of energy (depending on the grid it moves to). An agent can SEND a MESSAGE to one, some or all other agents, where ARES allows any bitstring as message text, thus giving the students total freedom in developing the communication language of their agents. Since an agent only receives information about the current grid and the direct neighbors from ARES, it also has the possibility to OBSERVE a grid farther away. The quality of the information resulting from this action depends on the distance between current grid and observed grid, so that agents have to deal with vague and even incorrect information.

Since the main goal of the agents is to rescue survivors, they have first to DIG them out as a (sub)TEAM and then SAVE the SURVivors. Since rubble might require several agents digging, ARES will remove the top piece of rubble on a grid only if at least the necessary number of agents (indicated by the piece of rubble) stand on the grid and perform in the same turn the TEAM_DIG command. If a survivor or a survivor group surfaces to the top of the stack of the grid, then one agent has to perform the SAVE_SURV action to get them to safety.

Regarding survivors, it should be noted that they start out with a certain amount of "life energy" and loose some of the energy every turn. If the energy level of a survivor goes below zero, this survivor dies. In contrast to this, the agents can replenish their energy by performing the SLEEP action (depending on the world rules either they can do this action everywhere or only on the recharging grids).

The messages back from ARES to the agents mirror mostly the actions that the agents can perform: acknowledging the CONNECT action, forwarding the messages the other agents send to an agent, and sending the result of a MOVE command, which includes information about the surrounding grids, as do the messages that acknowledge the SLEEP, OBSERVE, TEAM_DIG and SAVE_SURV commands. Additionally, ARES can tell the agents that it shuts down using the DISCONNECT message, it can shut out a dead agent by the DEATH_CARD message, it tells the agent to start (continue) its computations with ROUND_START and it interrupts the computations with ROUND_END. If no command has been sent from an agent to ARES before the ROUND_END, the agent will not perform any actions this turn. If several commands were sent, ARES will execute for the agent the last one (except if communication is free, then all SEND_MESSAGE commands will also be executed). Finally, ARES has a very generic error message, UNKNOWN, that tells an agent that its last command could not be recognized. If a command is understood, but could not be executed -for example, a move out of the grid world- then ARES simply does not execute the command (and the agent looses this turn). There is no error message in this case. When ARES acknowledges the CONNECT action of an agent, it sends this agent some initial information about the world scenario it will be acting in (this information is more or less accurate) and its initial position in this world.

3.2 The instructor view

From the instructor's point of view, the relevant components of ARES concern the generation of world scenarios and the observation of the performance of a team of agents developed by a team of students. For the generation of world scenarios, first the general world rules are needed that will be common to all scenarios and that have to be told to the students at the beginning of the course, so that they can focus their development on these world rules.

As already stated in Section 2, being able to have different world rules in different semesters has a lot of advantages. Therefore ARES consults the so-called world_rules file that the instructor has to distribute in addition to the standard ARES distribution. So far, an instructor has the following world rules for which he/she has to choose an instantiation:

- **communication cost**: either a communication action is treated as every other action, which means that communication does not allow for any other action for an agent in a turn, or ARES forwards every message by an agent to the intended receivers and allows additionally in each turn for performing one of the other possible actions.

- **distortion of initial world**: obviously, after a disaster has struck, some of the knowledge from before the disaster allows to at least get some idea where there are chances for finding survivors. Therefore the agents acting in ARES get initially a "map" of the world scenario that has some resemblance to reality. ARES just takes its real world model and distorts it (randomly changing numerical values, like accumulated life energy per grid, within a given limit). How much distortion takes place, i.e. how big the limits are, can be defined by this world rule, thus influencing the initial planning capability of the agent team.

- **charge vs. sleep**: agents loose energy every turn and therefore there has to be a way to replenish this energy. In ARES, we can select from two different ways of doing this: charging at a charging grid or sleeping on an arbitrary grid. If agents can only charge at a charging grid, long-term planning of actions is much more important, because if they do not reach the charging grid with the remaining energy, they are dead (i.e. out of the run). Using sleep as the world rule allows for a more reactive planning without the need for a long-term perspective.

- **maximum lifting requirement**: the number of agents necessary to lift and remove a piece of rubble has quite an impact on the strategies that agents use (as was demonstrated in our first usage of ARES in a multi-agent course, see Section 4). If all agents might be needed, they should not get to far apart, while the knowledge that, for example, only three agents can handle every situation allows for more diversified plans. Therefore we allow an instructor to set a limit for the number of agents needed for lifting that will be observed in all scenarios the agent teams are facing.

There are a few more candidates for world rules that we are considering at the moment, but already the four from above allow for quite different foci for the development of agent strategies.

After having defined the common rules for all world scenarios, for testing the agent teams different scenarios are needed to avoid having the students do some over-optimization (which with only a few scenarios is possible, by simply encoding optimal strategies for each of the scenarios). For creating a world scenario, ARES provides an instructor with a graphical interface (and this interface, as part of the world viewer, is also available to students for testing particular aspects of their agents). Details of this "world builder" can be found in Appendix B.6. In general, every detail of a grid and the stack of rubble and survivors of the grid can be observed/changed, including life energy of survivors, energy necessary to lift rubble and energy lost when moving on a certain grid. It is possible to start with an empty world (i.e. a world just containing normal grids with empty stacks) or with a randomly generated world, where certain limits can be put on how random the created world really is.

At the end of the semester, the important requirement for ARES is to allow for evaluating and comparing the agent teams produced by the students. Test runs are protocoled, so that the basic statistics can be easily accessed and the protocols can also be used to visualize the run in the world viewer (in different modes, like a slow or fast animation, but also turn by turn, and with different focus to observe each agent individually). More details about this can be found in Appendix B.

3.3 Some implementation details

Due to the well-known software engineering advantages, ARES itself also has been designed as a multi-agent system, consisting of the Kernel agent, the WID agent and the viewer agent. each agent is realized as a different process and the agents communicate using UDP sockets. All processes are running on the same computer. The Kernel handles all connections to the agents written by the students, i.e. it receives their actions/messages and sends back to them the results of these actions/messages. The WID agent (**W**orld **I**nformation **D**atabase agent) handles the simulation of the world scenarios. Triggered by the Kernel agent, it modifies the current world scenario state based on the received actions (the Kernel already makes sure that only the last action submitted by a student agent is forwarded and, depending on the world rule, also the messages) and send back to the Kernel the new state. The viewer agent also communicates with the WIF agent to get the world state that it then displays for the user. Having the WID agent kind of in-between the other two agents allows us to use only the WID and the viewer to replay runs using the run protocol, which is a very important feature of ARES, because this allows the user to set a focus on, for example, different rescue agents (in different replays of the same run) to analyze their behavior closely.

While for the moment, we see ARES giving us enough flexibility in our teaching, it has to be expected that in the future we want to add features to the

system. The internal architecture of ARES and our implementation allow for easy addition of new objects to the world or of additional actions/commands for the rescue agents. The division into Kernel and WID agent also allows the easy integration of simulations changing the world (in addition to letting survivors die) into the WID agent (that just runs such simulation components as it "runs" the agents' actions).

4 Experiences with ARES

We used ARES for the major assignment of our multi-agent course (see [De02]). Naturally, this assignment was to develop and implement a number of agents that as team had to rescue survivors in several world scenarios of ARES. More precisely, each team of students was given the task to develop a (team) strategy for five agents. In preparation for this, each individual student was given one of five general cooperation ideas from literature and he/she had to produce a midterm paper describing the idea in general and suggesting ways how this idea could be employed by the agents acting in ARES. Each student in a team was given a different cooperation idea out of

- Cooperation by giving out selected information
- Cooperation using blackboards/shared memory
- Cooperation via a Master-Slave Approach
- Cooperation using Negotiations
- Cooperation using Market Mechanisms

Naturally, the aim of this midterm paper was not to have one student and his/her assigned idea to come out as "winner" with regard to developing the team strategy. We expected that some kind of mixture of the ideas would be what the students would be deciding upon. But the midterm papers made sure that the students started thinking about the task early and that they were aware of a broader set of possibilities.

The course had both undergraduate and graduate students and even within the three student teams we had undergraduates and graduates mixed. The world rules we choose were as follows:

- we made communication expensive by having SEND_MESSAGE as an action like all the others,
- we distorted the initial world by 5 percent,
- we allowed the agents to replenish their energy by sleeping whenever they want, and
- we had as maximum lifting requirement the maximum number of agents, i.e. 5.

For testing purposes, the students were given three randomly generated world scenarios and naturally they could create their own scenarios. At the end of the semester, we evaluated the rescue agent teams of the students using the randomly generated world scenarios and several other "pathological" world scenarios created to test some abilities of individual agents and teams. Such tests were, for example,

- testing how good the navigation abilities of agents/teams were, by locating the only survivors within an area nearly surrounded by fire and allowing a seemingly easy and short path (that cost a lot of energy) and a longer path (that had much less energy consumption when taking it),
- testing the decision making (and its basic philosophy) of the team by placing survivor groups of different sizes in different distances from the agents, with the larger groups being farther away and a rather strict time limit, so that rescuing a larger group becomes impossible if you stop to rescue a small one,
- testing the usage of the initial world information by taking the last test scenario and additionally giving the survivors in the larger survivor groups less life energy, so that in the initial world information small and large groups have approximately the same amount of life energy (since the amount of life energy is what is told to the rescue teams).

Overall, each team of students produced rather different agents and team strategies. The rescue team of each team of students was able to be the best team in at least one world scenario. But there was one team that was best in most scenarios. This most successful implementation of the agents was found to be when all five agents act as one "super-agent". At the start of a run, all agents used the world information to determine the highest concentration of survivors (resp. life energy) to determine the initial meeting spot, rescuing survivors on the way if possible (i.e. if a single agent can remove the rubble). After the agents join together, they stay together and always move to the nearest grid with sufficiently high concentration of survivors. The agents use Dijkstra's algorithm for finding the shortest path between a start and destination grid and since they all have the same information, only very little communication is necessary. In fact, the agents communicate at the beginning their start positions to each other and later they only communicate if one agent is low on energy, so that all agents stop and sleep until this agent has sufficient energy, again.

Obviously, the students tailored their agents well to the chosen world rules: communication is expensive, so try to minimize it. Since the initial world information is rather accurate and common to all agents, base your decisions on it so that you do not have to communicate. And because all agents might be needed to dig out survivors, keeping them together came forth. Also obviously, by having the ability to change the world rules, we can define changes that will make this particular setup to fail. It should also be noted that the students of the best rescue team did not make use of the OBSERVE action and this together

with having the agents not scout around, meant that they did not have data about how weak survivors were getting. So, if we had one very strong survivor in one grid and further away several weak survivors, both grids would have the same accumulated life energy and the rescue agents would first go to the nearer strong survivor and this way might loose the survivor group. This was how the rescue teams of the other student teams were able to beat the team in a few world scenarios.

5 Conclusion and Future Work

We have presented the ARES system, a testbed for rescue agent teams that are developed by students as the major assignment of a beginners course in multi-agent systems. ARES has been especially developed with teaching in mind, so that on the one hand it let students focus on the multi-agent issues cooperation and communication, and on the other hand it fulfills the wishes of instructors, like being able to have different foci in the assignment every semester. We successfully used ARES in our course and so indicated experimentally that our goals in developing ARES are fulfilled.

Naturally, there are a lot of possibilities of how ARES can be improved. Over time, we expect to find a lot of candidates for additional world rules, like starting the rescue agents with the same information versus different information or having fire that spreads versus fire that does not spread (in our first use of ARES, no rescue agent died in a scenario, spreading fire might change this). And with some of the instantiations of such rules it will be possible to make the assignment harder to reflect, for example, the preparation students got from other courses (which differs from university to university) or to use ARES both for undergraduate courses and graduate courses. We also see the need for additional testbeds that, like ARES, are targeted on teaching instead of evaluating research ideas. Such other testbeds could focus on scenarios in which competition between agents is the main issue.

References

[AL02] AgentLink.org: http://www.AgentLink.org/resources/teaching-db.html, as seen on June 5, 2002.

[De02] Denzinger, J.: http://pages.cpsc.ucalgary.ca/ denzinge/courses/cs601-winter2002.html, as seen on June 5, 2002.

[Ki00] Kitano, H.: RoboCup Rescue: A Grand Challenge for Multi-Agent Systems, Proc. ICMAS-2000, IEEE Press, 2000, pp. 5–12.

[Ku02] Kummeneje, J.: http://www.dsv.su.se/∼johank/RoboCup/manual/, as seen on February 25, 2002.

[SV00] Stone, P. and Veloso, M.: Multiagent Systems: A Survey from a Machine Perspective, Autonomous Robotics 8:3, 2000.

[We+01] Wellman, M.P.; Wurman, P.R.; O'Malley, K.; Bangera, R.; Lin, S.; Reeves, D.; Walsh, W.E.: Designing the Market Game for a Trading Agent Competition, IEEE Internet Computing 5(2), 2001, pp. 43–51.

[Wo02] Wooldridge, M.: An Introduction to Multiagent Systems, John Wiley & Sons, 2002.

[RR02] http://www.r.cs.kobe-u.ac.jp/robocup-rescue/, as seen on June 5, 2002.

A ARES command set

This section will cover a description of the protocol for messages agents can send to the ARES system and the messages that are sent from the ARES system to the agents.

A.1 Basic Definitions

This section includes a description of basic parts of the protocol that will be needed to explain the messages protocol in the next sections.

1. <int> → (0-9)* (eg) 1, 2, 456, 1987, etc ...
2. <string> → (\/.a-zA-Z)*
3. <bool> → TRUE — FALSE
4. <direction> → NORTH | SOUTH | WEST | EAST | NORTH_WEST | NORTH_EAST | SOUTH_EAST | SOUTH_WEST | STAY_PUT
5. <message> → A chunk of data, can be string or byte information
6. <idList> → ID <int>, <idList> | ε
7. <object> → <Rubble> | <Survivor> | <SurvivorGroup> | <BottomLayer>
8. <Rubble> → RUBBLE (ID <int>, NUM_TO_RM <int>, RM_ENG <int>)
9. <Survivor> → SURVIVOR (ID <int>, ENG_LEV <int>, DMG_FAC <int>, BDM <int>, MS <int>)
10. <SurvivorGroup> → SURVIVOR_GROUP (ID <int>, NUM_SV <int>, ENG_LV <int>)
11. <BottomLayer> → BOTTOM_LAYER
12. <Grid Types> → <NoGrid> | <NormalGrid> | <ChargingGrid>
13. <NoGrid> → NO_GRID
14. <NormalGrid> → NORMAL_GRID <GridInfo>
15. <ChargingGrid> → CHARGING_GRID <GridInfo>
16. <GridInfo> → (ROW_ID <int>, COL_ID <int>, ON_FIRE <bool>, MV_COST <int>, NUM_AGT <int> , ID_List (<idList>), TOP_LAYER (<object>))
17. <SurroundInfo> → <CurrentGrid>, NORTH_WEST (<Grid Types>), NORTH (<Grid Types>) ,NORTH_EAST (<Grid Types>), EAST (<Grid Types>) SOUTH_EAST (<Grid Types>), SOUTH (<Grid Types>), SOUTH_WEST (<Grid Types>), WEST (<Grid Types>)

18. <SignalsList> → <int>, <SignalsList> | ε

19. <CurrentGrid> CURR_GRID(<Grid Types>), NUM_SIG <int>, LIFE_SIG (<SignalsList>)

A.2 Agent → System Messages

This section will cover an explanation of the messages that can be sent from an agent to the ARES system.

1. CONNECT

   Definition CONNECT

   Explanation Sent to connect to the system.

   Reply Message From System CONNECT_OK

   Example CONNECT

2. SEND_MESSAGE

   Definition SEND_MESSAGE(NumTo <int>, MsgSize <int>, ID_List (<idList>), MSG <message>)

   Explanation Can be used to send messages to other agents indicated in the ID list.
   Note: if NumTo is set to zero then the message will be forwarded to all agents connected to the system.

   Reply Message From System None.

   Example

   - SEND_MESSAGE(NumTo 2, MsgSize 5, ID_List (ID 1, ID 3,), MSG HELP)
   - SEND_MESSAGE(NumTo 0, MsgSize 6, ID_List ( ), MSG 123467)

3. MOVE

   Definition MOVE (<direction>)

   Explanation Can be sent to move the agent in the indicated direction.

   Reply Message From System MOVE_RESULT

   Example

   - MOVE (NORTH)
   - MOVE (STAY_PUT)

4. TEAM_DIG

   Definition TEAM_DIG

Explanation Sent when the agent wants to participate in the removal of a Rubble object.

Reply Message From System TEAM_DIG_RESULT

Example TEAM_DIG

5. SAVE_SURV

Definition SAVE_SURV

Explanation Sent when the agent wants to save a survivor.

Reply Message From System SAVE_SURV_RESULT

Example SAVE_SURV

6. SLEEP

Definition SLEEP

Explanation Sent when the agent wants to sleep for one round (this allows the agent to regain some energy).

Reply Message From System SLEEP_RESULT

Example SLEEP

7. OBSERVE

Definition OBSERVE (ROW <int> , COL <int>)

Explanation Can be used by an agent to look at a specific grid in the world. Note: the further this grid is away from the agent's current location the more distorted the information will be.

Reply Message From System OBSERVE_RESULT

Example OBSERVE (ROW 2 , COL 15)

A.3 System → Agent Messages

1. CONNECT_OK

Definition CONNECT_OK (ID <int> , ENG_LEV <int> , LOC (ROW <int> , COL <int>), FILE <string>)

Explanation Sent as a responce to the CONNECT message from an agent, to indicate that the agent is now connected to the system.

Example CONNECT_OK (ID 2, ENG_LEV 500,
LOC (ROW 2, COL 3), FILE WorldInfo.txt)

2. UNKNOWN

Definition UNKNOWN

Explanation Sent by the system to the agent when the command sent by the agent cannot be recognized by the system.

Example UNKNOWN

3. DISCONNECT

   Definition DISCONNECT

   Explanation Sent by the system when it is shutting down, to allow agent to shut down also.

   Example DISCONNECT

4. FWD_MESSAGE

   Definition FWD_MESSAGE (IDFrom <int>, MsgSize <int>, NUM_TO <int>, IDS (<idList>), MSG <message>)

   Explanation Sent by the system by request of another agent to send a message to the agent receiving this message.

   Example FWD_MESSAGE (IDFrom 2, MsgSize 6, NUM_TO 2, IDS (ID 1, ID 3,), MSG HELP)

5. ROUND_START

   Definition ROUND_START

   Explanation Sent by the system to indicate that the current agent's round at having processor time has started.

   Example ROUND_START

6. ROUND_END

   Definition ROUND_END

   Explanation Sent by the system to indicate that the current agent's round at having processor time has ended.

   Example ROUND_END

7. MOVE_RESULT

   Definition MOVE_RESULT (ENG_LEV <int>, <SurroundInfo>)

   Explanation Sent as a result of an agent's request to move

   Example MOVE_RESULT (ENG_LEV 455 , <SurroundInfo>)
   Note: We have omitted showing an example of the surroundinf information because it would get too large and as a consequence the example would not be easily readable.

8. SLEEP_RESULT

   Definition SLEEP_RESULT (RESULT <bool>, CH_ENG <int>)

   Explanation Sent as the result of the SLEEP command from the agent.

   Example SLEEP_RESULT (RESULT TRUE , CH_ENG 555)

9. OBSERVE_RESULT

   Definition OBSERVE_RESULT (ENG_LEV <int>, GRID_INFO
   (<Grid Types>), NUM_SIG <int>,
   LIFE_SIG (<SignalsList>))

   Explanation Sent as the result of an OBSERVE command from the agent.

   Example OBSERVE_RESULT (ENG_LEV 455 , GRID_INFO
   (NO_GRID), NUM_SIG 0, LIFE_SIG ( ))

10. TEAM_DIG_RESULT

    Definition TEAM_DIG_RESULT (ENG_LEV <int>, <SurroundInfo>)

    Explanation Sent as the result of the TEAM_DIG command from the agent.

    Example TEAM_DIG_RESULT (ENG_LEV 555, <SurroundInfo>)
    Note: We have omitted showing an example of the surroundinf information because it would get too large and as a consequence the example would not be easily readable.

11. SAVE_SURV_RESULT

    Definition SAVE_SURV_RESULT (ENG_LEV <int>, <SurroundInfo>)

    Explanation Sent as the result of the SAVE_SURV command from the agent.

    Example SAVE_SURV_RESULT (ENG_LEV 355, <SurroundInfo>)
    Note: We have omitted showing an example of the surroundinf information because it would get too large and as a consequence the example would not be easily readable.

12. DEATH_CARD

    Definition DEATH_CARD

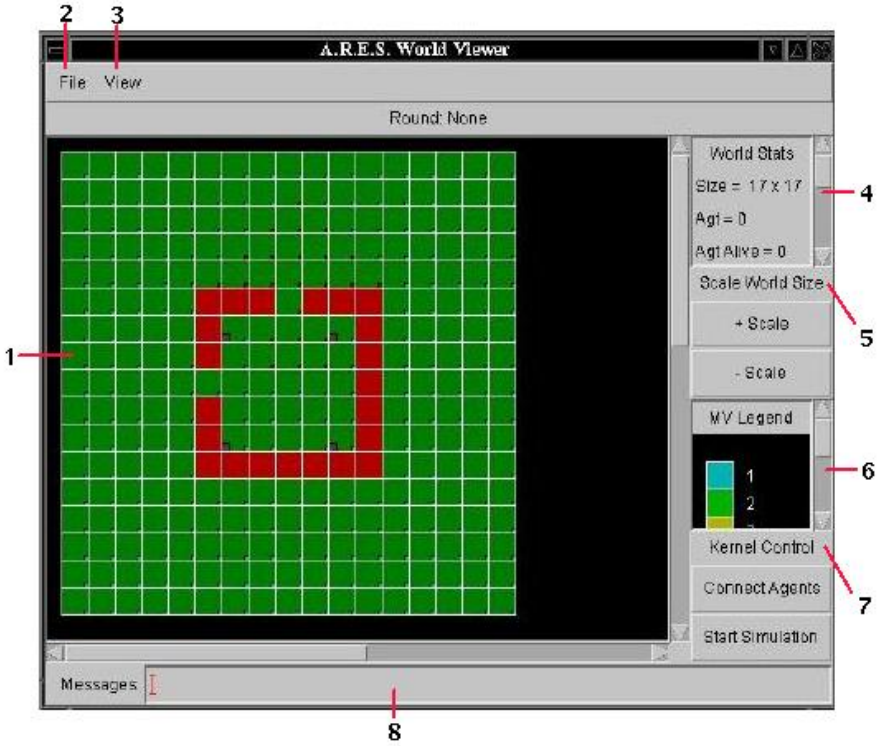
    Explanation Sent by the system to indicate that the agent is dead.

    Example DEATH_CARD

B The world viewer

This section will cover each GUI/feature in the ARES world viewer and how to use them.

B.1 The Main GUI



GUI Description: This GUI is used to display the world scenarios and to access the main functionalities of the viewer. This is the first GUI to appear after the world viewer is started.

GUI Section Descriptions:

1. World Display: this area of the GUI is used to display the world scenario graphically. If the mouse is clicked over top of a grid then a new GUI will appear containing the information about the selected grid.
2. File Menu: this menu contains the functionalities for saving a copy of the current world scenario, building a world graphically and replay a system run from a protocol file.
3. View Menu: this menu contains the functionalities for viewing a world scenario and viewing information about a specific grid or agent.

4. World Statistics: This area of the GUI displays statistics about the world scenario. Below is a description of the meaning of each value shown:

   Size: The size of the world as Row x Col

   Agt: The number of agents in the world

   Agt Alive: The number of agents that are alive

   Agt Dead: The number of agents that are dead

   SV: The number of initial survivors in the world

   SV Dead: The number of initial survivors that are dead

   SV Alive: The number of survivors that are still alive

   SV Saved: The number of survivors that have been saved

   -SV: The number of initial survivors that were saved when already dead

   +SV The number of initial survivors that were saved while still alive

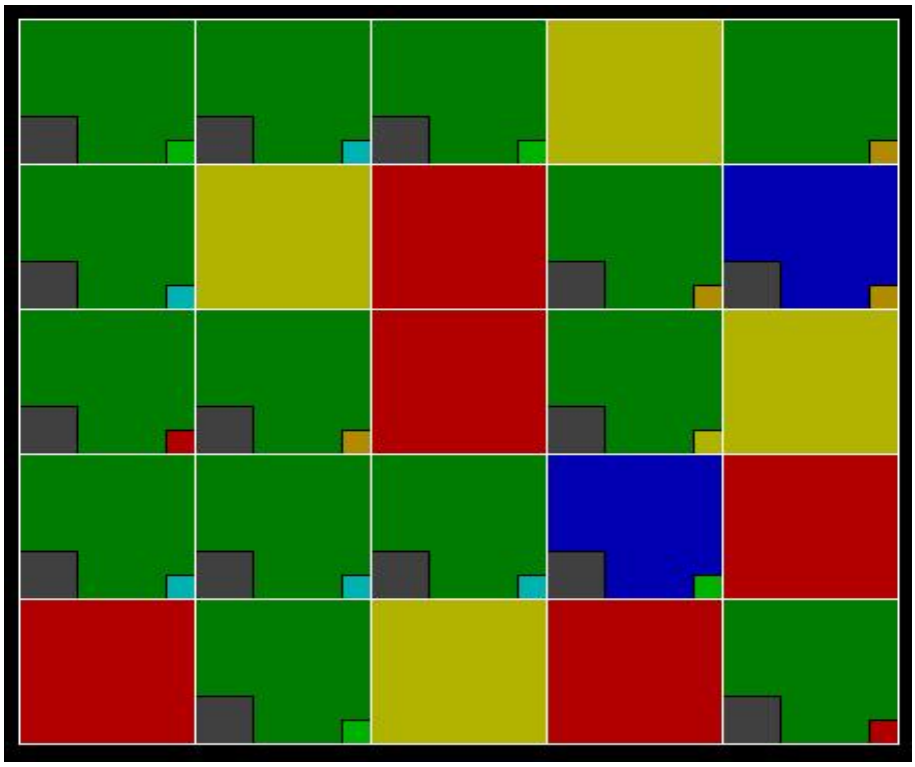
5. Changing the scale of the world: This area of the GUI can be used to change the displayed scale of the world. The "+ Scale" button will increase the display scale of the world (i.e. we zoom in) and the "- Scale" button will decrease the displayed scale of the world (i.e. zoom out).

6. Move Cost Legend: This legend displays graphically the move cost values with their associated colors. Remember that these values are just close estimates of the real move cost; to find out the exact move cost value of a grid, view the information for the desired grid in the special window with grid information (see later).

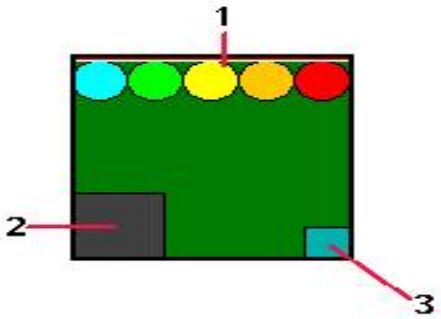
7. Kernel Control: This area of the GUI contains buttons that can be used to control the ARES system when it has been started up with the world viewer. The "Connect Agents" button should be selected when the user wants to connect his/her agents to the ARES system. The "Start Simulations" button should be selected by the user when he/she wants to start the simulation after the agents have been connected.

8. Error Message Display: If there are any errors that occur when running the system or using the world viewer, the error message will be displayed in this area (in a red font).

B.2 The composition of the world

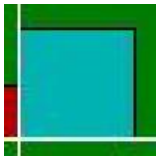


This section will cover an explanation of how the world and the objects within it are depicted graphically. The image above shows a screen shot of an example world scenario. A world (scenario) as it is set up in the ARES system is made up of a collection of "grids" that contain layers (resp. a stack of layers) of objects. For the representation of information about a grid the graphical display of a grid has been split up into different sections; their breakdown is described below.

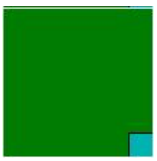
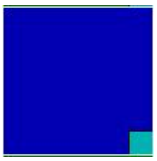


Grids in the ARES system have three properties, they are a move cost (3), a

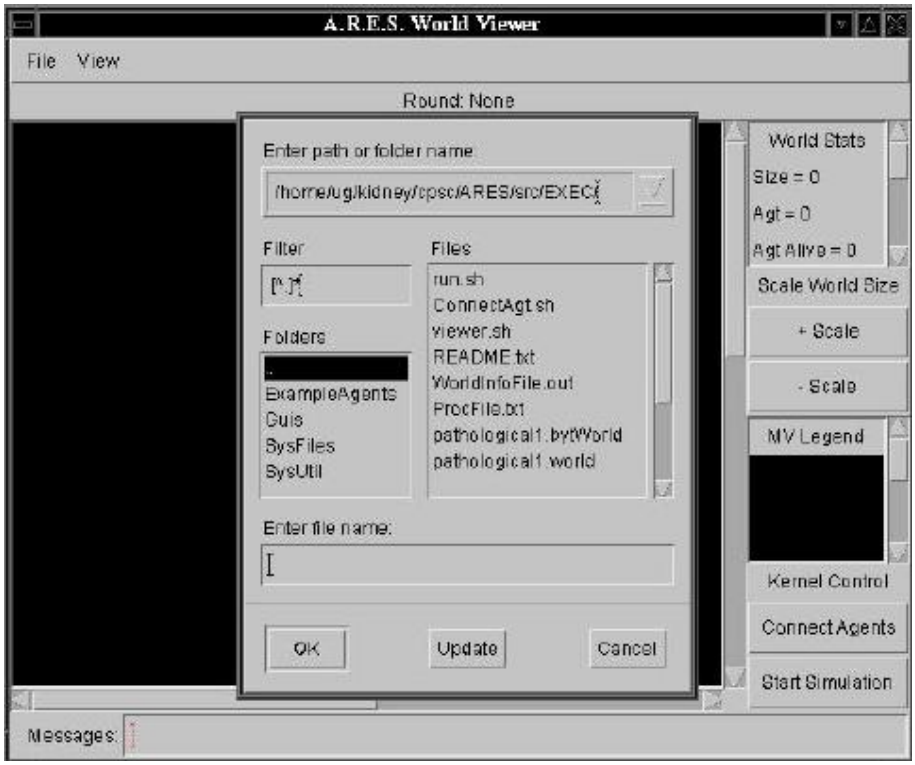
stack of layers (2), and agents that are on the grid (1), as depicted in the above picture. Below are pictures showing each type of object that can be on a grid's stack.

1.  Rubble
2.  Survivor
3.  Survivor Group

Finally, there are four different types of grids in the world, the picture for each type is shown below:

1.  Normal Grid
2.  Charging Grid
3.  Fire Grid
4.  Killer Grid

B.3 View a world scenario

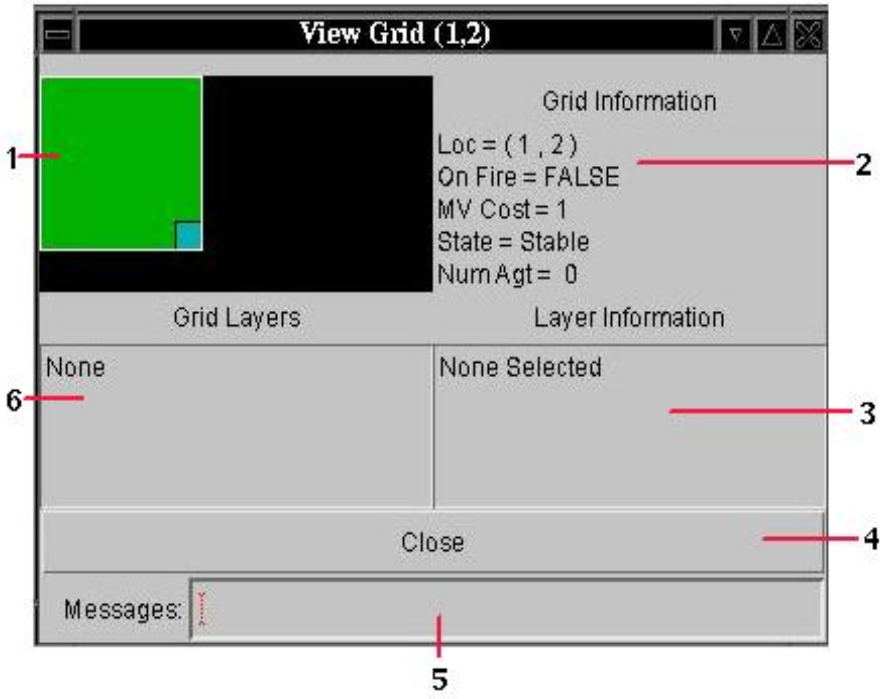


This feature allows to view an ARES world scenario file graphically and explore it with the other features built into the viewer. This feature can be accessed by selecting "View World" from the View menu or by pressing Ctrl-V. When this feature is selected, the user will be asked to enter the file to display (as shown in the figure above). If there were no errors, the world should then be displayed in the World Display area of the Main GUI.

B.4 View information about a grid

To view the information about a grid, there are two ways of going about this, below both ways will be explained.

B.4.1 Selection with the mouse



GUI Description: Displays information about a single grid. This GUI will appear when the mouse selects a grid from the World Display in the Main GUI.

GUI Section Descriptions:

1. Grid Display: graphically displays the selected grid.
2. Grid Info: displays in text format the grid's statistics, these values are explained in more detail below:

   Loc: the location of the grid shown as (Row,Col).

   On Fire: indicates if the grid is on fire or not. TRUE means the grid is on fire and FALSE means the grid is not on fire.

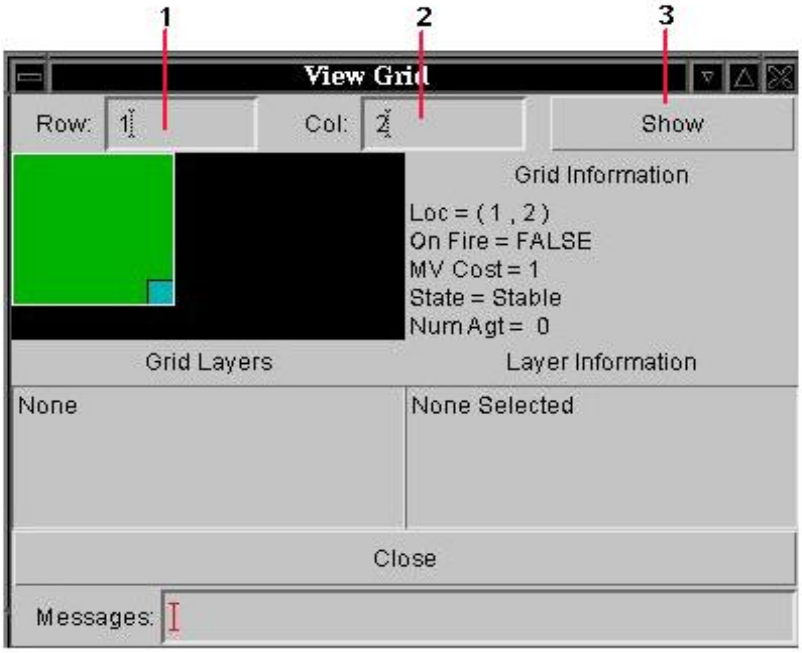
   MV Cost: the move cost of the grid.

   State: the current state of the grid. Stable means that the grid is safe for an agent and killer means that the grid will kill the agent.

   Num Agt: the number of agents on the grid .
3. Layer information: this area is used to display information about a single layer in the grid. The layer can be selected from the Grid Layers List (6). The information that is displayed here is specific to the type of object at the indicated layer in the grid.

4. Close Button: can be used to close the GUI window.
5. Error Message Display: if an error occurs while displaying the information about a grid it will be shown in this section.
6. Grid Layers: This area shows a list of the layers on the grid in a text format. The top layer is displayed first and so on with the bottom layer being displayed last. To see specific information about a layer select the desired layer from the list with the mouse and the information will be displayed in the Layer Information section (3).

B.4.2 Selection by grid location

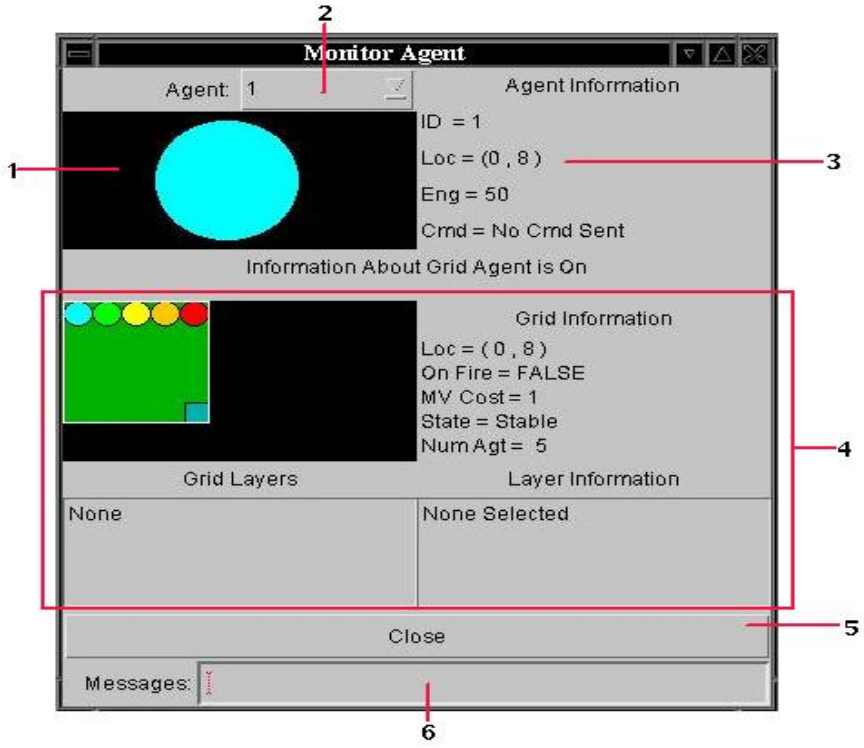


GUI Description: Displays information about a single grid. This feature can be activated by selecting "View Grid" from the View Menu or by pressing Ctrl-G.

GUI Section Descriptions: For more detail see the above section.

1. Row Value: used to enter the desired row value of the grid.
2. Col Value: used to enter the desired col value of the grid.
3. Show Button: once the desired grid location values have been entered this button is pressed to get the information about the desired grid.

B.5 Monitoring an Agent



GUI Description: Displays information about an agent. This feature can be activated by selecting "Monitor Agent" from the View menu or by pressing Ctrl-A.

GUI Section Descriptions:

1. Agent Display: graphically displays the agent, and what color represents the agent.
2. Select Agent: a drop down box that allows the user to select the agent to monitor based upon the agents ID number.
3. Agent Info: displays some text information about the agent. These values are explained in more detail below.

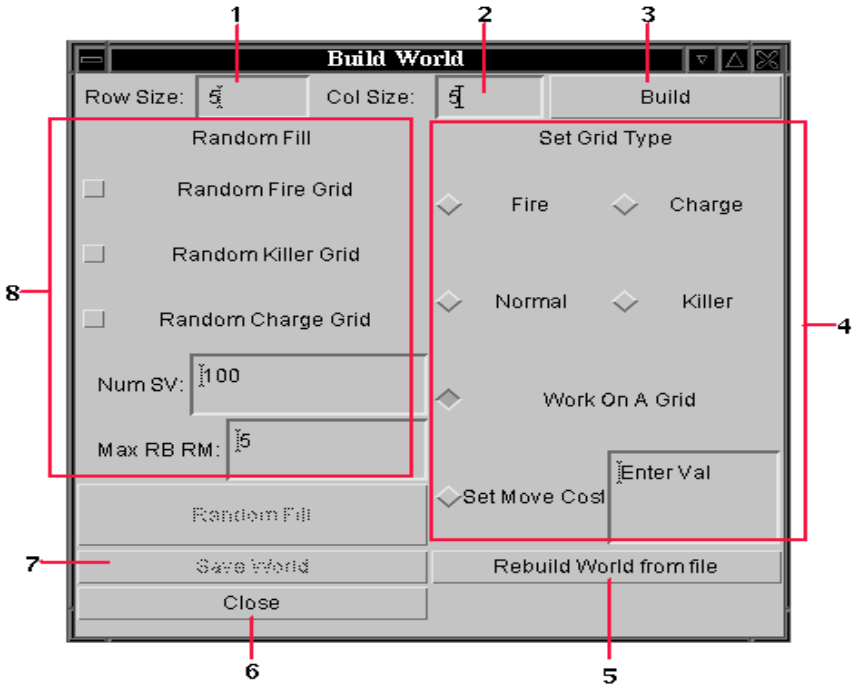
   ID: the id number of the agent

   Loc: the location of the agent in the following format (Row,Col)

   Cmd: the last command sent by the agent
4. Agent Grid Information: see "View information about a grid"
5. Close Button: this button can be used to close the GUI.
6. Error Message Display: if any errors occur while monitoring an agent an error message will be displayed in red text in this area.

B.6 Build a World Graphically

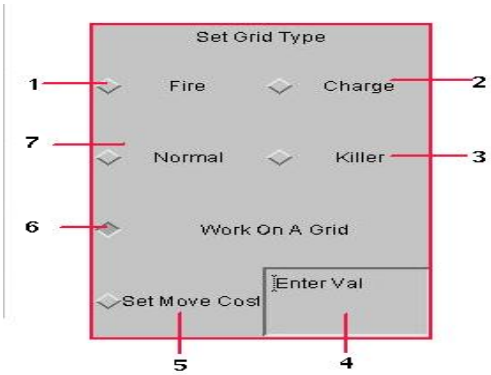
This section will cover how to build a world scenario with the ARES viewer and what each part of the Build World GUI does.



GUI Description: The main GUI used to build a world, this feature can be activated by selecting "Build World" from the File Menu or by pressing Ctrl-B. Note: All error messages will be displayed in the "Error Message Display" area of the Main GUI.

GUI Section Descriptions:

1. Row size: used to specify the row size of the world if building a world scenario from scratch.
2. Col size: used to specify the column size of the world if building a world scenario from scratch.
3. Build Button: once the Row and Col sizes have been entered (see above) then this button can be pressed and a blank world of the indicated size will be created and displayed in the World Display on the Main GUI (as described earlier).
4. Set Grid Info: this section of the GUI can be used to set the information about a specific grid in the world. This section of the GUI is described more precisely below:



Setting information for a grid can be a simple thing, all that has to be done is to select a button from this section and then select the grid that you want to change the information on from the "World Display" in the Main GUI. Below is a list of what each action does to a grid.

1 Fire Button: sets the selected grid on fire.

2 Charge Button: makes the selected grid a charging grid.

3 Killer Button: makes the selected grid a killer grid.

4 Move Cost Value: value used when setting the move cost of a grid.

5 Set Move Cost: sets the selected grid's move cost to the value indicated by "Move Cost Value" from above.

6 Work on a grid: this button lets you work on the specific information about a grid. When a grid is selected, a new GUI will appear, this GUI is explained in more detail in the next section.

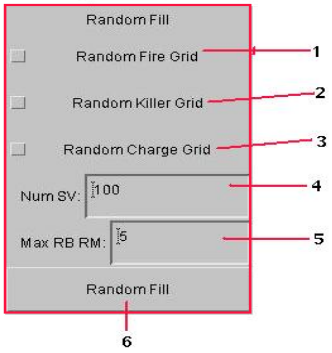
7 Normal Button: this sets the selected grid to be a normal grid.

5. Rebuild From File Button: If you want to modify a world scenario that already exists then this is the button for you. When this button is selected the user will be asked to enter the file to read in and then the world scenario will be loaded into the system and displayed in the "World Display" section of the Main GUI. From this point the user can then use all other functionality to modify the world and then save the changes.

6. Close Button: this will close the GUI (Note: The user must save all changes before closing the GUI, all unsaved data will not be saved by the system and therefore will be lost).

7. Save World Button: This will save the world scenario for the user. When pressed the user will have to specify the file name to save the world scenario into (Note: this name must not include any file tags as the system will add a .world tag to the end of the name).

8. Randomly fill the world: this section can be used to get the system to randomly fill in the world with grids and objects, each part of this section is described in more detail below.



1 Fire: set if the user wants the system to place fire randomly in the world.

2 Killer: set if the user wants the system to place Killer grids randomly in the world.

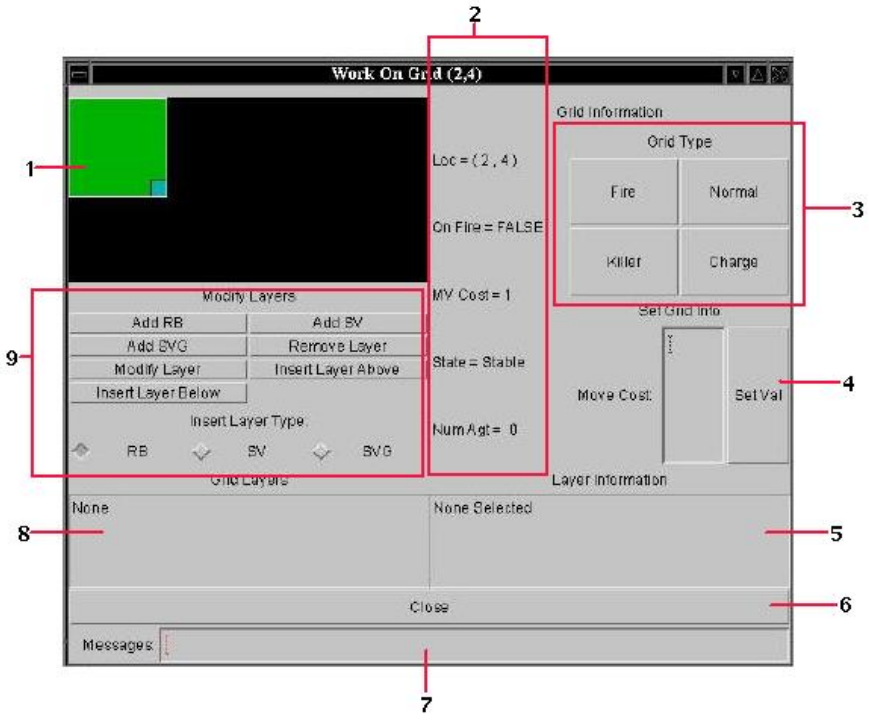
3 Charge: set if the user wants the system to place Charging grids randomly in the world.

4 Survivors: the number indicates the APPROXIMATE number of survivors the system should place in the world.

5 Rubble Remove: the value indicates the maximum number of agents needed to be able to remove the rubble. The value for the number of agents needed to remove the rubble will be set by taking a random number between 1 and the indicated value.

6 Random Fill button: once all values have been set, this button can be pressed to fill the world up randomly.

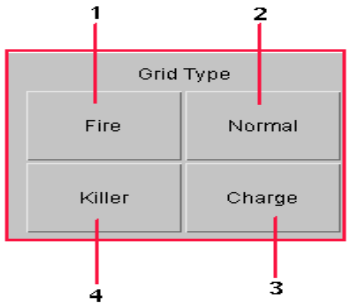
B.6.1 Work on a grid



GUI Description: This GUI is used when building a world scenario and the user wants to set specific information about a single grid. It can be activated by first selecting "Work on Grid" from the Build World GUI and then selecting the desired grid from the "World Display" on the Main GUI.

GUI Section Descriptions:

1. Grid Display: graphical representation of the grid.
2. Grid Information: textual display of statistics about the grid (see "View A Grid" for information about the meaning of each value).
3. Set Grid Type: this section can be used to set the type of the grid, each button is explained in more detail below.



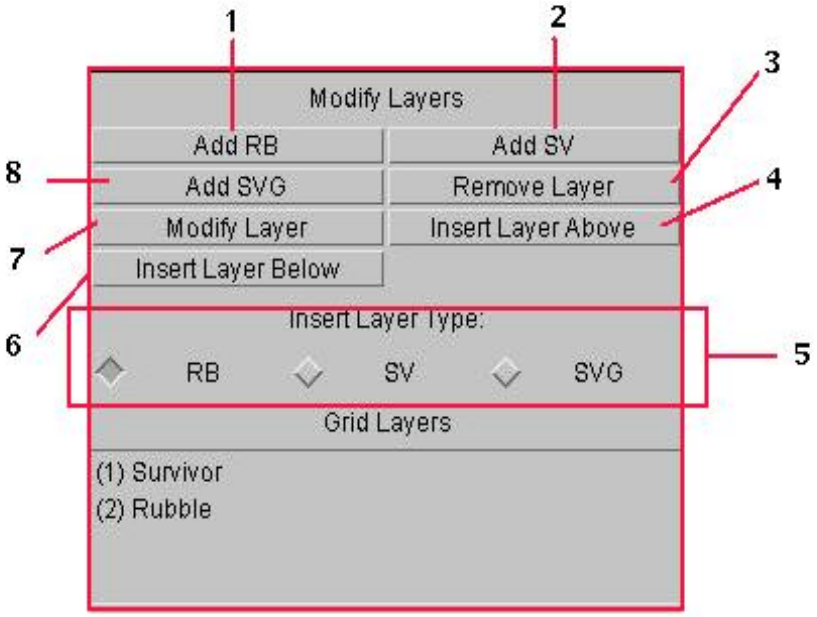
1 Fire: set the grid on fire.

2 Normal: set the Grid to be normal.

3 Charge: make the grid a charging grid.

4 Killer: make the grid a killer grid.

4. Move cost: set the move cost of the grid.
5. Layer information: display area for specific information about a layer of the grid stack.
6. Close button: closes the GUI.
7. Error Message Display: if any errors occur while setting the information about the grid, they will be shown in this area of the GUI.
8. Grid Layers: display area for the layers of the grid stack, starting from the top layer down to the bottom layer.
9. Modify Grid Layers: this section of the GUI can be used to modify the layers on the grid, this is explained in more detail below:



1 Add Rubble: this will add a new Rubble object to the grid (as the new top layer of the stack). Also a GUI will appear where the user can set the information about the object, this GUI will be explained in the next section.

2 Add Survivor: this will add a new Survivor object to the grid (as the new top layer of the stack). Also a GUI will appear where the user can set the information about the object, this GUI will be explained in the next section.

3 Remove Layer: this will remove a layer from the grid. To remove a layer the user must first select the layer to be removed from the Grid Layers List.

4 Insert Layer Above: this will insert a new layer above the the selected layer of the grid's stack. The type of object that is added is determined by the selection made in GUI section (5). Also, a GUI will appear where the user can set the information about the object, this GUI will be explained in the next section.

5 Insert Type: this indicates the type of object that is to be inserted when using the "Insert Layer Above" and "Insert Layer Below" buttons.

6 Insert Layer Below: this will insert a new layer below the the selected layer of the grid's stack. The type of object that is

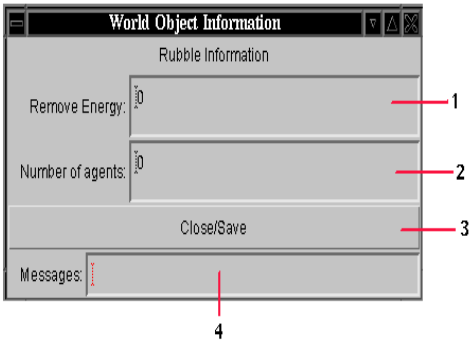
added is determined by the selection made in GUI section (5). Also, a GUI will appear where the user can set the information about the object, this GUI will be explained in the next section.

7 Modify Layer: this allows the user to modify the information of a specific layer. The user must first select the layer of the stack to modify before pressing this button.

8 Add Survivor Group: this will add a new Survivor Group object to the grid (as the new top layer of the stack). Also, a GUI will appear where the user can set the information about the object, this GUI will be explained in the next section.

B.6.2 View or Modify an object's information

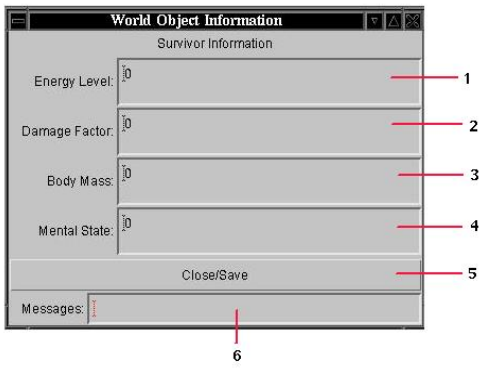
The following GUIs are used to modify objects (i.e. Rubble, single Survivors, or Survivor Groups) of the layers in the stack of a grid in ARES. We start with Rubble objects:



GUI Description: Used to display/modify information about a Rubble object.

GUI Section Descriptions:

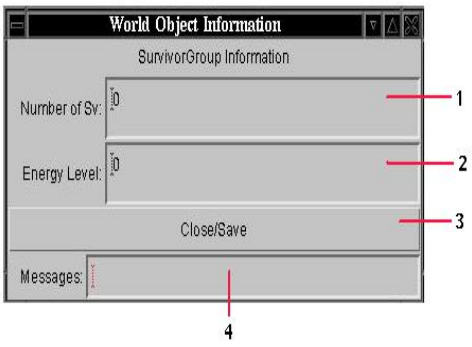
1. Remove Energy: the amount of energy each agent will lose after removing the Rubble object with TEAM_DIG.
2. Number of agents: the number of agents needed to remove the Rubble object.
3. Close/Save button: closes the window and saves the information displayed in the GUI as object information.
4. Error message display: if there are any errors, the resulting message will be displayed here.



GUI Description: Used to display/modify information about a Survivor object.

GUI Section Descriptions:

1. Energy Level: the starting energy level of the survivor.
2. Damage Factor: a number indicating the "damage" the survivor has suffered.
3. Body Mass: a number indicating the weight of the survivor (will be used in later versions of ARES).
4. Mental State: a number indicating just how confused the survivor is (will be used in later versions of ARES).
5. Close/Save button: closes the window and saves the information displayed in the GUI as object information.
6. Error message display: if there are any errors, the resulting message will be displayed here.

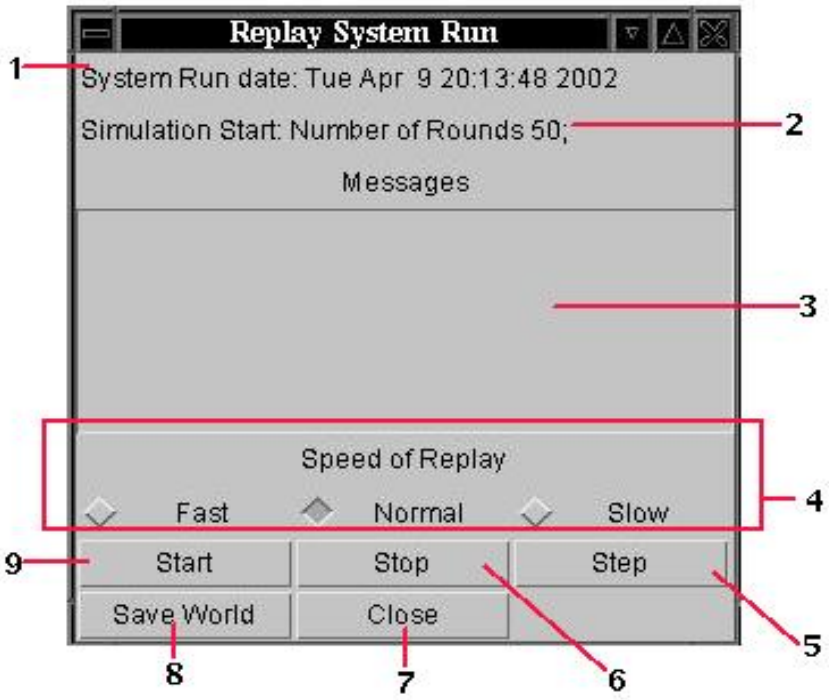


GUI Description: Used to display/modify information about a Survivor Group object.

GUI Section Descriptions:

1. Number of SV: the number of survivors in the group.
2. Energy Level: the total combined amount of life energy of the group (ARES will assume that this energy is equally divided among all group members).
3. Close/Save button: closes the window and saves the information displayed in the GUI as object information.
4. Error message display: if there are any errors, the resulting message will be displayed here.

B.7 Replay a system run



This feature of ARES with its associated GUI can be used to replay a previous run of ARES based on information in the protocol file that was generated for the previous run.

GUI Description: Used to control the flow of the replay. This feature can be activated by selecting "Replay Run" from the File menu or by pressing Ctrl-R. The user will first be asked to enter the protocol file's name and then, if no errors occurred, the world scenario will appear and the replay run GUI depicted above will appear.

GUI Section Descriptions:

1. Run Date: the date of the original system run.
2. Simulation Rounds: the number of rounds in the original system run.
3. Messages: display area for any messages generated by the system during the original run.
4. Replay speed: speed selection for automatic replay.
5. Step Button: used when not on automatic replay to step through the replay round by round.
6. Stop Button: stops the automatic replay feature.
7. Close Button: closes the GUI and ends the replay system run feature.
8. Save World Button: can be used to save the current world settings and configuration into a new file.
9. Start Button: starts the automatic replay feature. Depending upon the speed selected, the simulation will step forwards one round at a time in standard intervals. To change the speed the user must first press the Stop button (6), then select a new speed (4) and finally press the start button again.

C Installing ARES

C.1 System Requirements

The following are the requirements on a computer on which ARES should be installed on.

- Computer is running UNIX
- Your current OS has the make command installed
- Your current OS has the java command installed
- The following compilers are required:
  - g++
  - gcc
  - Java
  - flex
  - yacc

C.2 Installation of and compiling the ARES System

The following procedure contains the step by step instructions for installation of the ARES system:

- Step 1: Download the zipped file containing the code for the ARES system.
- Step 2: Type the following two commands in the order shown into a shell (note: make sure you are in the directory you downloaded the system into):
  - 1. gunzip ARES-System.tar.gz
  - 2. tar -xvf ARES-System.tar

  If successfully executed, a new directory called ARES should have been created.
- Step 3: Change your current directory to the ARES directory. At this point you have two choices:
  - 1. Keep all the code after compilation of the system is done
  - 2. Do not keep all the code after compilation of the system is done

  If you want to keep all the code after the system has been compiled, type make in the shell. If you do not want to keep the code after the system has been compiled, then type make && make REM_CODE in the shell.
- Step 4: You have now finished the installation procedure and can begin using the ARES system.

D Parameters of ARES

This section will cover a description of the ARES input files. These files allow a user to define a world scenario and set parameters for the system.

D.1 Formal Definition

1. <int> → (0-9)+
2. <String> → (_\/.a-zA-Z)+
3. <float> → ((0-9)+.(0-9)+)
4. <ARES World File> → <World Settings> <Grid Types> <Grid Stacks>
5. <World Settings> → Settings { <World Info> <Simulators> <Commands> }
6. <World Info> → World_Info { Size (<int> , <int>) Seed (<int>) World_File_Levels (<int> , <int> , <int>) }

7. <Simulators> → Simulators { <Settings> }
8. <Settings> → <Setting> <Settings> | ε
9. <Setting> → <String> (<ParamList>)
10. <ParamList> → <Param>, <ParamList> | ε
11. <Param> → <int> | <String> | <float>
12. <Commands> → Commands { <Settings> }
13. <Grid Types> → Grid_Types { <Types List> }
14. <Types List> → <type> <Types List> | ε
15. <type> → <String> { <Grid Locations> }
16. <Grid Locations> → <Location> <Grid Locations> | ε
17. <Location> → (<int> , <int>)
18. <Grid Stacks> → Stacks { <Grid List> }
19. <Grid List> → <Grid> <Grid List> | ε
20. <Grid> → Grid (<location> , Move_Cost <int>) { <Stack> }
21. <Stack> → <Object> <Stack> | ε
22. <Object> → <String> (<ParamList>) ;