# Artificial Actors for Real World Environments

**Matthew Roberts**

Department of Computing
Macquarie University
Sydney NSW 2109 Australia
mattr@ics.mq.edu.au

**Abstract**

We have developed a simulation environment called CreatureSpace that allows us to test our agent theories on intelligent agents in a complex realistic environment. We present the CreatureSpace architecture and our experiences in combining multiple artificial intelligence techniques in a uniform environment. CreatureSpace is a combination of a realistic environment and intelligent agents that populate this environment using Half-Life as a rendering engine. The agents we create are deliberative agents with the ability to exist in a realistic environment. They can process what they see and behave according to this and their own desires. We focus on how agents can manage large amounts of information and describe our embedded knowledge solution. We have tested the ideas in the paper by using CreatureSpace to run fire evacuation simulations with synthetic characters developed using our embedded knowledge solution. We find that combining embedded knowledge and sketchy planning results in simple solutions to potentially difficult problems in realistic environments.

## Introduction

Simulations of the real world have many uses. In computer games and films they are used to entertain, but they can also be used to demonstrate possibilities or as vehicles to explore scenarios.

We say that a virtual environment seeking to emulate the real world for one of these purposes is a *realistic environment*. Such environments are required to appear realistic to the observer. They must be immersive and it must be possible to populate the environment with intelligent agents. Computer game technology has many of the required characteristics but the agents in the games are often not suitable as general purpose intelligent agents in such environments since they are often scripted for a particular behaviour and do not have general planning or problem solving knowledge.

We define *synthetic characters* (or *artificial actors*) as agents that operate in realistic environments and substitute for human players. Figure 1 shows a synthetic character in a realistic environment. When using realistic environments for scenario testing, complex simulations can be run using a combination of agents and human participants, with some real human participants and some artificial participants behaving in a rational manner. Realistic environments are also useful for testing agent theories. Realistic environments allow us to challenge our agents with real world problems and allow us to avoid developing agents in sterilised environments where many of the real challenges can be ignored. The complexity inherent in realistic environments also highlights challenges that were not expected or may not have otherwise been considered. Although robotics offers the ultimate testing ground for agents, realistic environments come a close second and at a much cheaper cost. Some first person shooters, like Half-Life, provide advanced technology at a low cost and allow programmers access to those parts of the code required for creating a general-purpose realistic environment.

**Figure 1: A synthetic character (artificial actor)**

In some cases (for example, running a simulation to explore fire safety) we may want to know what the agents were "thinking". For this reason we require our agents to not only appear to act rationally, but also to be able to explain their choices in a way that can be easily interpreted by humans.

Intelligent agent architectures can be categorised based on their trade-off between computation time and the realization of goal driven behaviour. The least goal driven and fastest are reactive agents which use stimulus response rules, followed by triggering agents which are reactive but maintain an internal state. Deliberative agents are the most goal driven and least efficient. Hybrid agents are

deliberative agents that use reactive systems for part of their operation. In this project we consider deliberative planning to be important for goal driven behaviour and are not concerned if other parts of the agent's operation are not deliberative. Thus we define any agent which is driven by long-term goals as a deliberative agent, and do not distinguish them from hybrid agents.

In the space of an eight month part-time research project, we have created a simulation environment capable of rendering realistic scenes populated with intelligent agents that plan and choose actions in a human-like fashion. The basis for these agents is a knowledge management framework called *embedded knowledge*, which we describe below.

### Related Work

There has been an increased interest in extending computer game environments. For example, SOAR has been used to control agents playing Quake and Descent in the SOAR games project (Laird 2000). We have taken a similar approach, but instead of focusing on researching game agents, or increasing the game agent's capabilities, we have chosen to focus on creating a realistic simulation environment. We have adopted a planning architecture similar to one described by James Firby (Firby 1989) that separates planning and execution. Firby suggests that agents should plan at a higher level of abstraction and that a separate execution mechanism should take care of executing actions. Execution can then be specified in a reactive fashion. There can be more than one way to execute an action and the agent no longer needs to plan through the details of performing actions.

## The CreatureSpace System

An important part of developing techniques for intelligent agents is to create a working system. This is required to demonstrate the concepts, to explore their effectiveness and to learn about and review the ideas developed. CreatureSpace is such a system; it consists of two interacting programs that combine to provide an environment based on computer game technology, and agents that populate the environment. These agents can plan and act based on their own desires and what they see in the environment around them.

Part of our aim was to develop our agents in a realistic environment using existing technology. We make use of computer game technology, which can provide very realistic environments that provide detailed information to the player and to the game controlled agents that populate them, at low cost and with little effort. Half-Life can provide the agent with a body in the environment, including "eyes" and "ears" to see and hear its surroundings, giving it human-like capabilities in vision, hearing, and communication. Turning Half-Life into the environment renderer for CreatureSpace involved modifying it slightly to ensure it interfaces correctly with the agents we wish to place in it. This involves specifying instructions to the environment and responses to these instructions.

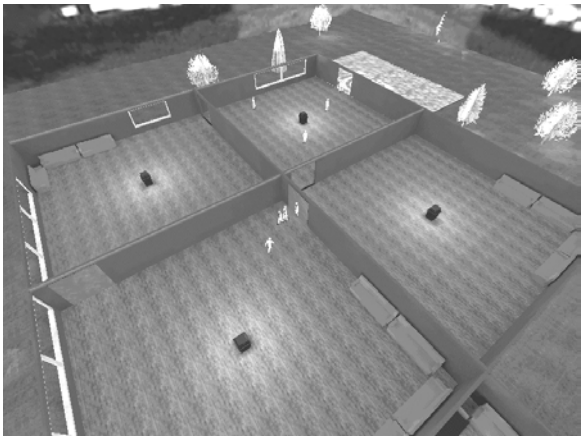

**Figure 2: CreatureSpace in action**

The ideal situation would be to have agents that can exist in any environment and always behave as a human would behave. Since creating such an agent was well beyond the scope of this project, we focused on just one domain, fire evacuation. Regardless, the agents are expected to exist in the environment as a human would, emulating human desires and responding appropriately to input. To achieve this purpose, agents need to be able to interact with their environment in the same way as humans do in theirs. While our agents are restricted to a single domain, they do exhibit

- Curiosity
- Self-preservation
- Social responsibility

Agents exhibit curiosity by exploring their surroundings, social responsibility by warning others of danger and self-preservation by following safety procedures in the event of a fire.

The evacuation simulations take place in a large building. The building has a number of rooms that are all connected by doors. Some of the doors are unreliable and can refuse to open, forcing the agents to find an alternative route. The building has multiple routes from one room to another and there is a predefined safe area outside the building at one end. During the simulations a fire breaks out somewhere in the building and we are able to observe what the agents do in response to this. We can directly observe their behaviour on the screen and CreatureSpace generates logs so we can observe their "thought processes" in detail.

## Agents Operating in Realistic Environments

One of the challenges that agents operating in the real world face is that of managing large amounts of information. We begin with Firby's concept of *situation driven execution* (Firby 1989). Agents plan at an abstract level, generating *sketchy plans* made up of higher level

actions than those that can be directly executed in the environment. Sketchy plans leave much of the detail of executing actions to a separate execution mechanism. The execution mechanism then makes use of the situation at the time of execution (as distinct from the time of planning) to determine how to execute the actions from a given set of predefined options. The execution mechanism does this in a reactive fashion.

In this project we have extended this distinction between planning and executing, from agent operation, to the management of an agent's knowledge. We define a distinction between the knowledge needed for planning and that needed for execution. The planning knowledge is used by a deliberative planner to generate sketchy plans and the execution knowledge is used by an execution mechanism in performing reactively specified actions. We have called this split in knowledge *embedded knowledge* and discuss it in more detail in the next two sections. Embedded knowledge also allows for an organisation of detailed information, modular addition and deletion of knowledge, and maintains distinctive representations for the planning knowledge and execution knowledge. The CreatureSpace system has been built around the embedded knowledge framework. We have found that this framework minimises costly planning and helps provide solutions to problems such as knowledge updating, agent attention, agent communication, re-planning and concurrent actions.

## Managing Knowledge

An agent in the real world is bombarded by information and has to explicitly choose what information to incorporate into its knowledge base and which information to use at a particular time. A simple act such as opening a door requires a great deal of numerical information regarding spatial locations of objects. While the amount of information in a simulation is still less than what can be directly perceived in the real world, a good simulation has plenty of detail and as a result, large amounts of information for the agents to process. An agent that does not filter information would need to maintain a large database of information that would become unwieldy for use in planning. We seek agents that have a set of goals, can make multi step plans in response to those goals, execute multi-step plans and re-plan as a result of observation. As the size of the knowledge base increases, making plans in real time becomes more difficult. To solve this problem, we restrict the information that the agent makes use of when it does its planning. When using sketchy plans, planning is abstracted to a higher level, so most of the information in the environment is needed only for the details of execution.

To facilitate this, some of the agent's knowledge is labeled *cognitive knowledge* and the planner is restricted to working only with this knowledge. As all the details of the environment are potentially required for execution, we choose not to exclude any knowledge from the knowledge the execution mechanism has access to. We call this *reactive knowledge* and it is exactly all the knowledge an agent has about its environment. We call this framework embedded knowledge because the cognitive knowledge is a subset of reactive knowledge; it is embedded within the reactive knowledge.

Consider the example of a door. The cognitive knowledge about a door might be that it opens, and which rooms it leads to. The reactive knowledge about this door would be that it opens, which rooms it leads to, what type of handle it has, where it is, how to open it, which direction it opens, etc. The reactive knowledge contains all the details required for opening doors and walking through them, while the cognitive knowledge contains only the information needed for making plans that may include walking through the door. Notice that the cognitive knowledge is a subset of the reactive knowledge.

## Extending Embedded Knowledge

There are two additional concepts in embedded knowledge that improve the agent's ability to manage knowledge and add to the system's flexibility. These are: different representations for cognitive and reactive knowledge, and a contextual organisation of reactive knowledge.

Because the planner and the execution mechanism are different systems doing very different jobs, it is entirely possible for the two systems to need two different representations of knowledge. For this reason, we define cognitive knowledge to be more than simply a subset of reactive knowledge; it is a subset of reactive knowledge that may then be transformed into a new form.

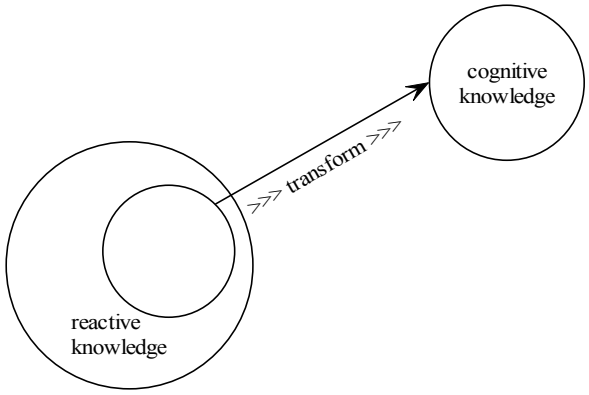


**Figure 3: The relationship between reactive and cognitive knowledge**

We also define a special organisation for reactive knowledge. It is a contextual organisation where related knowledge is kept in nodes and the various nodes are connected in a graph. The connections in the graph denote relationships between nodes of information, similar to a semantic network. We get two major advantages from this organization. Firstly, information that is related is stored together so that access from one to another is fast and simple. The second advantage is that each node now has a surrounding context, information that is related to it and can provide extra information about this node when it is

required. An example of where we use this context is in knowledge updating. The context in which knowledge is stored aids in choosing how to update it when conflicts occur.

This organization applies only to reactive knowledge. It is not inherited by the cognitive knowledge that is derived from the reactive knowledge. This is related to differing representations. The reactive execution mechanism requires organized knowledge for fast access and for providing a context for knowledge updating. On the other hand, the deductive planner specifically does not want to be constrained in how it can use cognitive knowledge, so it has no such organization.

## The CreatureSpace Architecture

As described above, we created a system consisting of two interacting programs that is capable of running simulations. In particular, we have focused on the evacuation of a building in case of fire. The system consists of a rendering and basic mechanics engine (Half-Life), and an agent controller.

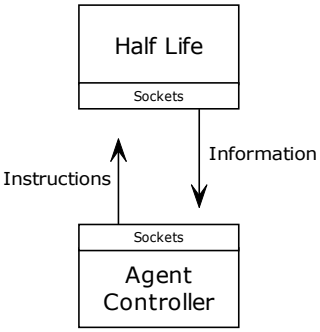


**Figure 4: Overall CreatureSpace architecture**

Half-Life is responsible for providing the environment and the representation (body) of the agents in that environment. The agent controller is responsible for providing the intelligence of the agents. It sends instructions to its representation when it wants to perform some action. It then receives information about how the action proceeded.

The two components communicate across sockets using a simple language that operates as a request-response system. The agent controller will request something of the engine (perhaps *"move my representation to position x, y"*) and the environment (Half-Life) will respond. Normally responses are simply true or false indicating success or failure to perform the request. When appropriate, the responses incorporate additional information on the success or failure. For example, the failure response for walking is *"failed - was stopped at position x, y"*. This simple system is powerful and has easily accommodated the functionality required for CreatureSpace. Table 1 enumerates some of the possible request - response pairs.

| **Request** | **Response** |
|---|---|
| Walk for x meters | True - finished at location x y<br>False - finished at location x y |
| Turn x degrees | True<br>False - stopped after x degrees |
| Where is the closest agent? | True - location x y<br>False |
| Open a door | True - Door will take x seconds to open<br>False |
| Yell *"sentence"* | True |
| Look around this room | True - positions of corners and doors in this room. |

**Table 1: Part of the agent controller to Half-Life communication protocol**

The rendering and basic mechanics engine is provided by Half-Life. A *mod* (modification and recompilation of the source code) of Half-Life was created with some new code specifying the agents that would populate the world as well as effects such as fire. Half-Life provides a particularly realistic environment in which the agents can operate. It is flexible, easy to extend and provides good quality rendering of scenes.

The agent controller can be further broken down into an *action selector*, a *planner*, a *knowledge base* and an *execution mechanism*.

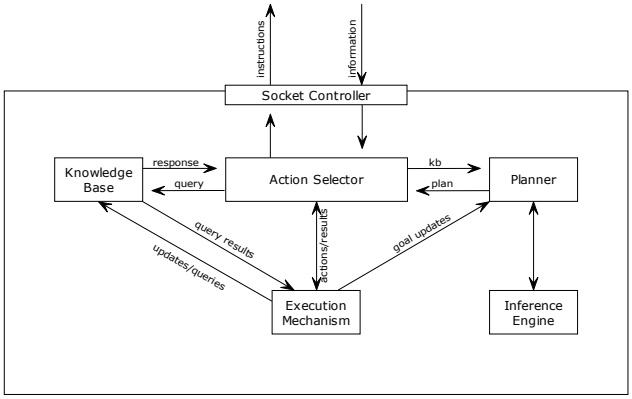


**Figure 5: The agent controller organisation**

The action selector is responsible for executing the basic thinking loop for the agent. This loop is an *observe-> plan-> act* loop. The knowledge base provides an encapsulation of all the functionality for managing knowledge using embedded knowledge. The planner is responsible for generating sketchy plans from cognitive knowledge and in this task it makes use of an inference engine. In the CreatureSpace system planning is done with a STRIPS style planner (Fikes, Hart & Nilsson 1972). We use a prolog-based system as the inference mechanism and as our STRIPS planner. The execution mechanism defines a set of actions that the planner can use for planning and contains reactive specifications for executing each of the actions.

## Easy Solutions

In creating this simulation environment, a number of challenges arose. Most of these were simpler to solve than would have otherwise been the case, because we were using the embedded knowledge framework and sketchy plans. We were able to find solutions to the following challenges:

- Updating knowledge
- Re-planning
- Uncertainty
- Agent attention/agent communication
- Concurrent actions

Updating knowledge in this system is relatively easy because the execution mechanism is given direct access to the reactive knowledge in its context. It requires this knowledge to tailor execution to the current situation. When the execution mechanism updates reactive knowledge, the cognitive knowledge derived from it is automatically updated.

Re-planning is relatively simple in this system because the execution mechanism has access to the reactive knowledge and can easily update it as described above. This means that after a plan has failed, the knowledge base has already been updated and a new plan can be generated from it.

Agent attention and communication are related problems because communicating with an agent first involves getting its attention. In CreatureSpace all agents have a list of things that can grab their attention. If any one of those things comes into their field of vision or within earshot, they will cease current execution, update the knowledge base to incorporate whatever drew its attention, and then re-plan based on this new information.

Concurrent actions are important for doing things like walking and talking at the same time. In CreatureSpace we define two types of action, those that can be done concurrently with others and those that cannot. The execution mechanism threads execution of actions together if concurrent actions are required. At no stage does the planner need to plan for concurrent actions. For example, the planner may generate a plan including walking and talking. From the planner's point of view they are being executed one after the other. However the execution mechanism recognises that walking and talking can be done at the same time and interleaves the execution of the two actions. Keeping the planner free from concurrent actions contributes to the simplicity and speed of the planner.

The execution mechanism also takes care of dealing with uncertainty. This frees the planner from this difficult task, further simplifying planning. The execution mechanism can recognise that some actions are particularly uncertain and force re-planning immediately after executing them.

A further advantage that the combination of embedded knowledge and sketchy planning affords us is that the plans generated by the agent are simpler to interpret. The plans are not muddied by details, resulting in cleaner, easier to interpret plans. The low level details can be investigated if required, by interrogating the execution mechanism.

## The Simulations

The usefulness of embedded knowledge in deliberative agents has been demonstrated by running fire evacuation simulations. The fire evacuations all involved the same building that had one or more agents in it. The agents are assumed to be familiar with basic fire drill techniques and will evacuate the building to a safe area outside the building. The scenarios of the fire evacuation that were run are as follows:

- One agent who knows the layout of the building
- One agent who must explore the building.
- Multiple agents.
- Broken doors that force the agents to rethink their plans.
- Trapping an agent to see how it copes.

The simulations were run and recorded in a standard compressed digital video format. Figure 6 is a snapshot from one of the recorded simulations. The agent has just had a fire break out in front of him and is in the process of getting to the exit.

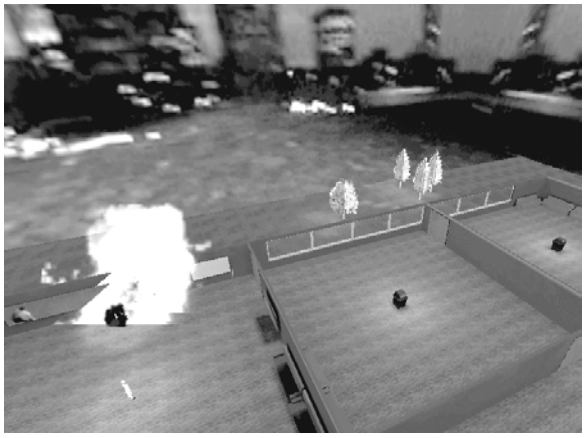

**Figure 6: Agent notices fire in front of him and evacuates the building**

As expected, in scenarios where the doors were broken, agents took longer to evacuate. Similarly, more obstacles and more agents led to bottlenecks at the doors. These bottlenecks slowed the evacuation quite significantly as agents jostled through the doors. Agents in the simulations would yell warnings once they became aware of the emergency. Their voices had a limited range, but anyone within that range would hear the warning and then decide whether or not to pay attention to it. We found that communication of the fire could travel ahead of the first agent to see it, meaning agents far from the fire would hear

of it quickly and get out of the building first, even if the fire was a long way away.

While these specific results might be achieved with purely reactive agents, we could not as easily investigate why the agents chose particular actions and it would not be as simple to scale to more complex domains. Furthermore, the rules for behaviour and the reactive systems behind them would be far more complex and more difficult to understand.

The combination of embedded knowledge and sketchy planning allows the agents in our simulations to plan in real time despite the complexity of the environment in which they exist. We found that the amount of cognitive knowledge required for planning was far less than the reactive knowledge that needed to be maintained, allowing for real time planning. High level actions were easily represented by a reactive specification in terms of lower level actions. As a result of using embedded knowledge and sketchy planning for these agents we were able to take advantage of the simple solutions to the problems of concurrency, re-planning, attention and knowledge update. If we had not used embedded knowledge and sketchy plans it is expected that we would not have been able to implement all of these in the time available.

## Embedded Knowledge in Action

Each of the simulations generated logs from the execution mechanism, the planner and the communications between the environment and the agent. Extracts from these logs that demonstrate the deductive planner of the CreatureSpace agents are included below.

The following is a description of two iterations of the agent's thinking loop. This loop executes indefinitely as long as the agent exists and is responsible for the agent being autonomous. The agent chooses a plan based on its current knowledge and executes it in each iteration. In this case, the agent is standing in a building and there is nothing worth noting going on around it.

Initially, the agent has the following goals (with priorities) and cognitive knowledge, which together make up its state of mind. The agent's cognitive knowledge is captured as the planner uses it, so it is already in a form appropriate for the Prolog based inference engine.

```
Goals : ***************************
  exists(me) : 1
  discover : 2

***** My current cognitive knowledge is *****
burn(_):-
  fail.
clear_of_building:-
  fail.
contains(building1,room1).
...
contains(room2,obstacle1).
contains(room3,door2).
...
directly_connected(room1,room2).
...
directly_connected(safe_area,room4).
discover:-
  fail.
everyone_warned(_):-
  fail.
exists(me).
first_name(person1,john).
holds(contains(room1,me), init).
job(person1,sparky).
leadsTo(door1,room1).
...
leadsTo(door6,safe_area).
outside_room(_):-
  fail.
outside_room(safe_area).
room(room1).
...
room(room6).
```

This shows quite a bit of detail that is required by the planning algorithm but is not of importance here. Although the cognitive knowledge has been abridged, for the sake of brevity, this agent is actually aware of all the rooms, doors, obstacles and people in the building it is in, and is aware of fire evacuation techniques. Notice that the cognitive knowledge used by the planner is expressed at a very high level. No details about the location or sizes of objects are included since this information is present as reactive knowledge, but filtered out of cognitive knowledge.

Some of the rules, such as this one, seem strange.

```
outside_room(_):-
  fail.
outside_room(safe_area).
```

This rule set should be read as "if there are no outside rooms we fail. The room called safe_area is a room". The redundancy in the cognitive knowledge is due to the fact that the agent is acquiring knowledge incrementally and the knowledge base is generated automatically. Note that all this cognitive knowledge is in a form that can easily be used in traditional planning.

The goals labeled `exist` and `discover`, lead to the following query of the inference engine based on the agent's strongest desire. This query asks, *"how do I discover things from my current situation?"* and the answer from the planner is *"explore your surroundings"*. The desire to explore is built into the agents in CreatureSpace, as we desire curious agents, who are not content to sit around if their knowledge is incomplete.

```
*************** Query results***************
Query of : achieve_goals([discover], init, S)
Generated a plan of
explore
```

Now the agent must execute the action `explore`. So the execution mechanism is called into operation. This mechanism executes the explore action based on the reactive knowledge (the current situation). In this case, the agent knows everything it wants to know about the building it is in, so it chooses to simply look around the room. Depending on the actual situation, the agent may choose another action, for example, to investigate one of the other rooms.

In this particular case, the agent notices a fire, which interrupts the execution of the action. The action of seeing a fire triggers a change in the agent's reactive knowledge to include the information about the fire, and also triggers a change to the agent's desires:

```
Goals : ***************************
  exists(me) : 1
  discover : 2
  contains(safe_area,me) : 3
  everyone_warned(fire) : 3
  clear_of_building : 3
```

Note that these three new desires all possess a high priority. The failure of an action always forces the agent to generate a new plan, as action failure often leads to a significant change in the agent's knowledge or desires.

When re-planning, the agent's cognitive knowledge reflects the change in its reactive knowledge by including one new piece of information:

```
...
burn(building1).
...
```

The query generated from the desires is *"how do I get to the safe area, warn everyone of the fire and get clear of the buiding?"*. The plan generated in response to this is:

```
walk(room1,safe_area)
yell(fire)
clear_building
```

These three actions are now executed by the execution mechanism. In this example, the action `yell` can be executed concurrently with the other two actions. The planner did not know this, and generated the plan as a linear sequence of actions. At execution time, the execution mechanism notices that concurrency is possible and interleaves the actions.

In these two iterations of the thinking loop the agent has used its cognitive knowledge, its surroundings and its desires to determine a plan. It then acted out this plan based on its detailed reactive knowledge. When the situation changed the agent re-planed based on the new information. This was all executed in real time.

## Discussion

This project has not been free of unexpected problems, although they were relatively few and far between. One of the problems in specifying agents operating in such realistic environments is that any mistake in movement visually stands out. Agents that turn a little left then right when they should have turned right straight away show up very clearly. Using a reactive execution mechanism partially solves this problem but it still requires some very careful execution specifications to make agents perform actions without showing themselves up as computer controlled. This project has managed to achieve an incomplete, but functional simulation environment with agents that plan and choose actions in a believable fashion and with a particularly realistic rendering of environments. The speed with which we were able to achieve this (four "man months") shows the leverage we have been able to get from the computer game technology, the speed of development with CreatureSpace and the ease with which we were able to find solutions to difficult problems.

### Future Work

We envision that the architecture would be scalable to more complex domains and more complex agents. With agents that perform deliberative planning, we could build agents with differing planning capacities and knowledge. We could then examine how the differences between individual agents may influence the outcome of real life situations such as evacuations. For instance, how the presence of delegated fire wardens facilitates orderly evacuation.

## Conclusion

Creating a simulation environment using computer game technology and embedded knowledge has proven a useful combination. The computer game solves many of our rendering and basic mechanics problems. The embedded knowledge framework has proven itself capable of managing agent knowledge in an effective manner, and when combined with sketchy planning, has led to some unexpectedly easy solutions to problems we encountered.

## Acknowledgments

I would like to acknowledge the help of the entire Computing Department at Macquarie University for their contributions to this project. Particular thanks go to my supervisor Yusuf Pisan, whose ideas began this work and whose advice throughout the year was invaluable.

## References

Firby, R. J. 1989. *Adaptive Execution in Complex Dynamic Worlds*. Ph.D. thesis, Yale University.

Laird, J. 2000. It knows what you're going to do: adding anticipation to a Quakebot. In *Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment* Menlo Park, Calif.: AAAI Press.

Fikes, R. E., Hart, P. E., and Nilsson, N.J. 1972 Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3:251-288.