Artificial Actors for Real World Environments

Matthew Roberts
Department of Computer Science
Macquarie University
Sydney NSW 2109 Australia
mattr@ics.mq.edu.au

Abstract

We have developed a simulation environment called CreatureSpace that allows testing 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. We use Half-Life as the rendering engine for CreatureSpace. We focus on how agents can manage large amounts of information and describe our embedded knowledge solution.

Introduction

Simulations of the real world serve many purposes, in many cases they are used for entertainment but in many others they have more serious uses, such as demonstrations or as vehicles for exploring scenarios. Computer games and movies have the market for entertaining real world simulations fairly well covered. But what of the more serious uses? Limitations we often face when trying to use such simulations are that the environment is not realistic enough, it is not possible to immerse yourself in the environment or it is not possible to populate these worlds with intelligent agents. The computer game technology that is used for entertaining environments can be used to create environments suitable for any number of purposes, including, but not restricted to, running simulations as we have done here. A vital part of creating simulations of the real world is being able to populate these worlds with artificial actors that can take the place of real human participants in simulations. More 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. In the space of an 8 month, part-time research project, we have been able to create a simulation environment capable of rendering real world scenes and being populated by intelligent agents that plan and choose actions in a human-like fashion. We have also created a knowledge management framework for the agents that populate these environments. This framework minimises costly planning and helps provide solutions to problems such as knowledge updating, agent attention, agent communication, re-planning and concurrent actions.

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 while the SOAR games project focusses on researching game agents, or increasing game the agent's capabilities, our
focus is on creating a real world simulation environment. We have adapted a planning architecture similar to 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 that one way to execute an action and the agent no longer needs to plan through the details of executing actions.

**Agents operating in real world environments**

Agents operating in the real world face challenges that agents in other domains do not. The challenge that we deal with here is that of managing large amounts of information. We begin with Firby's concept of situation driven execution (Firby 1989). The agent does not plan at a level such that the plan can be passed strait to the environment renderer. Rather, higher level, more abstract, sketchy plans are generated. Sketchy plans leave much of the detail of executing actions to a separate execution mechanism, using higher level, more abstract actions to plan with. What we have created in this project is an extension of this distinction between planning and executing, from agent operation to the management of an agent's knowledge. In order to deal with managing information by splitting planning and execution, we have created an architecture using embedded knowledge that allows us to distinguish between the knowledge we need for planning and the knowledge we need for execution. Embedded knowledge also allows for an organisation of detailed information, modular addition and deletion of knowledge, and distinctive representations for the knowledge needed for planning and the knowledge needed for execution.

**Managing Knowledge**

An agent in a 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 a result, large amounts of information for the agents to process. It would be possible for the agent to simply acquire huge amounts of information and store it in a knowledge base. However, we desire deliberative agents. For our purposes we define a deliberative agent as one that has 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. The larger the knowledge base, the tougher it is to create deliberative plans in real time. We would like to be able to restrict the information that the agent makes use of when it does it's 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 knowledge is labelled cognitive knowledge the planner is restricted to working only with this knowledge. As all the details of the environment are required for execution, we choose not to exclude any knowledge from that which the execution mechanism works with. We call this reactive knowledge and it is exactly all the knowledge an agent has about its environment. The cognitive knowledge is a subset of reactive knowledge, so we say it is embedded within the reactive knowledge.

**Extending embedded knowledge**
The concept above is the basis of embedded knowledge, but it is not the extent of it, there are two more concepts in embedded knowledge that improve the agent's ability to manage knowledge and that 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 that simply a subset of reactive knowledge, it is a function of reactive knowledge. The function will filter out a great deal of information and may or may not transform it into a new representation in some form. 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 object oriented paradigm has been a strong influence in the design of this organisation of knowledge. The connections of the graph denote relationships between nodes of information. This organisation is similar to a semantic network. The cognitive knowledge that gets derived from this reactive knowledge does not inherit its organisation. This is done because this system has been designed for use with a deductive planner, and we do not want to over-constrain the planner.

CreatureSpace

In an effort to both demonstrate and better explore the ideas presented above, we have created a system, consisting of a number interacting programs, that is capable of running simulations. In particular, we have focused on evacuation of a building in case of fire. The system consists of a rendering and basic mechanics engine, and an agent controller. The system created has been dubbed CreatureSpace.

The CreatureSpace architecture

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, is easy to extend, freely available and provides good rendering of scenes.

The two components we have added to Half-Life are an agent server, written in C++ which has been built into the Half-Life mod, and an agent controller written in Java that
communicates with the server using sockets and instructs the agents on what actions to take. The agent controller can be further broken down into an action selector, a planner, a knowledge base and an execution mechanism.

![Diagram of the agent controller]

The action selector is responsible for executing the basic thinking loop for the agent. This loop is a basic observe -> plan -> act loop. The knowledge base provides an encapsulation of all the functionality for managing knowledge using embedded. 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 et al. 1972), this requires an inference engine capable of reasoning with Horn clauses and this is provided by a prolog based inference engine. The execution mechanism is exactly the execution mechanism from the planning - execution organisation described above. It defines a set of actions that the planner can use for planning and contains reactive specifications for executing each of them.

Easy solutions

In creating this simulation environment, a number of challenges arose, most were far 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
- 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. 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 similar problems because communicating with an agent first
involves getting their 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.

Discussion

This project has not been free on unexpected problems, although I should point out that they
were relatively few and far between. Half-Life has not been released for the purpose of
aiding agent theories, it is a computer game. While the results from Half-Life are spectacular
given the overall work required, one does need to spend some time coming to terms with it.
The other problem is that when your agents are operating in such a realistic environment, any
mistake in movement is very obvious. Agents that turn a little left then right when they
should have gone right strait away show up very clearly. Using a reactive execution
mechanism goes some way to solving this problem but it still requires some very careful
execution specifications to make agents perform actions without showing themselves up as
computer controlled. In this project we were able to create agents that could plan and attempt
actions in a believable fashion, but there were still glitches in their action execution that
would stick out. 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. All this has been achieved as a part
time research project over 8 months.

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 problem. 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.

References

http://ai.eecs.umich.edu/~soarbot/