Virtual Battlefield Simulation Agents, Experience with the SIM_AGENT Toolkit

Jeremy W. Baxter, Richard T. Hepplewhite

DERA Malvern
St Andrews Road, Malvern,
Worcs. WR14 3JP, UK
(jbaxter , rth)@signal.dera.gov.uk

Abstract

This paper describes our use of the SIM_AGENT toolkit for research into agents for battlefield simulations. We briefly describe our work and then try to classify the domain in which we are working with respect to the environment, agent's goals and actions, the architecture of our agents and environmental representation. The facilities of the toolkit are outlined and discussed with respect to the ease with which they enabled us to apply the toolkit to our domain. We conclude by discussing some benchmarks we believe could help in categorising toolkits and describing some features we would like to see in future agent toolkits.

Introduction

Our work on agents has been in the area of Computer Generated Forces (Hepplewhite 1996). More specifically we have used agents to control groups of tanks within virtual reality type simulations. This paper concentrates on a characterization of the domain and the agents and discusses how this domain affected our choice of toolkit and considers how well the features provided by the toolkit used (SIM_AGENT (Sloman 1995)) have matched up the evolving needs of our research and implementation.

The simulations which we are working with are designed to operate as part of a training system for military commanders. These commanders interact with the simulation from a map display or an 'out of the window' view into the simulation. The virtual battlefield needs to be populated by opposing and supporting forces, which in order to reduce manpower costs, need to be as independent as possible. The agents are therefore expected to interact with each other and the trainees as realistically as possible, and in real time. The behaviour of tank agents is governed by two main factors, the terrain over which they are moving and their beliefs about the enemy. In trying to produce battlefield behaviour which mimics that of a human tactician it is advantageous to model the existing command structure used by the army. This helps with the gathering of knowledge from subject matter experts and enables a hierarchical decomposition of the problems. High level commanders are given objectives which are used to produce lower level objectives for their subordinates. Information flows both up and down the command chain and agents need to co-operate with their peers to achieve the overall goal set by their commander. This natural decomposition of the problem allows higher level agents to work on long term plans while the individual tank agents carry out orders designed to achieve more immediate objectives. Our agents fulfill the roles of both individual tank commanders and the commanders of groups. Their roles are closely tied in to the roles performed by military commanders in the army's command and control structure.

Classifying the domain

The categories used to describe our domain are based on those suggested by Logan (Logan 1998). They are divided into consideration of the agents' environment, their goal types and representations, the properties of the agents actions, internal architectural systems and internal representation of the environment.

The Environment

The environment of our agents can be regarded as inherently dynamic as there are many agents forming the supporting and opposing forces who can move freely in the environment. The terrain however, is a largely static portion of the environment which changes little. In our simulations the terrain is fully observable as each agent has access to the terrain database but the actions and positions of other agents are only partially observable through simulated sensors since agents need line of sight across the terrain to identify other agents. The actions of these other agents are only partially predictable, depending on their types and relationship to the agent observing them. Given these features we classify our environment as a fairly complex one, made dynamic and unpredictable by the presence of other agents.

The Agents' goals

An agent's goals, viewed from outside the agent, are relatively simple. Goals are assigned externally, either by a human operator or from agents higher up the command

hierarchy. The agents are inherently cooperative in that they will always try to achieve the goal they have been set, in conjunction with their peers. The agent has a single goal, which may be a maintenance goal but its goal is almost certain to conflict with the goals of opposing agents.
Agents can only abandon a goal when told to do so by an agent above them in the hierarchy. Even if they believe the goal to be unachievable it cannot be abandoned, only reported as such. The external goals usually have constraints associated with them covering the time within which they should be completed and the area of terrain over which the agent can operate. One of the difficulties however is that there are no clear cut utility functions or optimal policies available for the domain. Advice and comments can be sought from military experts and tactics manuals but conversion to the sorts of utilities usable by computer algorithms is extremely difficult. In many cases the best that can be hoped for is a measure of how reasonable the resulting behaviour appears to be. One of the problems is that adopting some fixed policy is inherently sub-optimal since one of the desired features of tactical behaviour is to be unpredictable.

Internal goals can be generated by the agent either in response to goals received from outside or be an implicit part of the agent's nature. For example agents operate with an implicit goal of self preservation and an implicit goal to destroy opposing agents. Agents also generate subgoals internally as part of the planning process to achieve their top level, externally supplied goal. These goals are more complex since unlike the externally supplied goal there may be several competing goals. An agent may decide to abandon an internally derived subgoal if it believes it is unnecessary or no longer achievable. This ability is a necessary part of operating in a dynamic or partially observable environment since the information which caused an agent to adopt a goal may change or be found to be inaccurate.

Actions

The agents can affect their environment in several ways, they can move, shoot and communicate. These actions are fallible but only infrequently so. Agents may be unable to move due to impassable terrain or the risk of collision. They may be unable to shoot due to the loss of sight of the target, battle damage or ammunition supply and the effects of the action are probabilistic. Communication acts are usually successful but their effects are not guaranteed since orders given to subordinates may turn out to be unachievable. In our implementation agents communicate using a very simple language of orders and reports with well prescribed meanings. There are no complex semantics and so communication should be regarded as data passing rather than natural language communication. Communication with the operator is very limited, as is discussed below in the section on debugging.
The actions of agents affect their ability to perceive the environment, either through motion to a different vantage point or by tasking other agents to move and gather information. The actions have different costs and could be subject to resource constraints although our present agents ignore their fuel and ammunition supply states. The primary cost considerations are of time and exposure to danger.

Architectural properties

Our system is hierarchical, with multiple agents cooperating to achieve a single goal, this is based on the real world military command structure. Agents are therefore organised in a semi-rigid structure within which they have to support and cooperate with their peers. Their goal is given to them by a superior in the hierarchy and they may plan to achieve this by giving tasks to subordinates in the command structure. At the lowest level (individual tanks) there is a considerable reactive component to the architecture, agents react to the detection of new enemy vehicles immediately and this reaction consists of several actions, considering the new vehicle as a target, turning to face this new threat and communicating its sighting information to other agents in its group.
All agents and particularly those higher up the command chain have a considerable deliberative component. In most cases this deliberation takes the form of planning and search over the complex terrain to identify suitable routes for movement or positions on the terrain to provide protection or concealment. The ability of our agents to reflect about their own reasoning processes is extremely limited, instead most of our techniques rely on an 'anytime' component to provide plans within suitable time frames. No learning takes place within our agents, their capabilities are fixed before execution and do not change.

Environmental representation

The agents manufacture and hold several different representations of the environment. The terrain surface is held as a static database and the agent can create abstractions of this data for different reasoning purposes. All these representations however are essentially spatial maps of the environment, holding additional information about the benefits or dangers of particular locations. In addition to this agents have dynamic beliefs about the locations of other agents which they build up from sensor information and radio messages. The present formulation relies on the fact that such information is always accurate and consistent so that agents do not have to process the sensor information to maintain hypotheses or belief measures. Perceptual information is not complete however so an agent cannot assume that it knows of all other agents in its environment and do have to make assumptions about the continuing existence of agents which are not currently visible to sensors.
The agents do not try and reason about the beliefs and intentions of other agents, however some simple assumptions about the short term actions of agents are made. The complex representation of the beliefs and

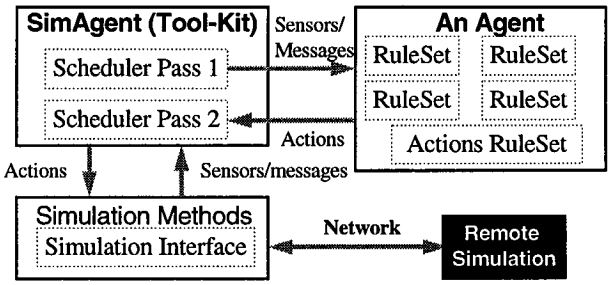

Figure 1. Tool-Kit Overview.

intentions of other agents may well be required in the future development of the agents.

The SIM_AGENT Toolkit

The toolkit which we have been using was developed in collaboration with Prof. Aaron Sloman at Birmingham University as the SIM_AGENT (Sloman 1995) tool-kit, written in Poplog. The tool-kit executes multiple agents, controls the message passing between them and allows physical simulation of the agents. It was designed as a general purpose toolkit for exploring different agent architectures and has been used for several different implementations at Birmingham University in addition to the work at DERA.

The tool-kit provides the facility to support different architectures between the agents, and possibly a number of sub-architectures within each agent to support all its functionality. The agents need to interact with each other and possibly with other entities and so must be physically simulated. This can be achieved either using modules internal to the tool-kit, or by enabling the agents to control the actions of a separate simulation system, in the work described here an external simulation has been used. Figure 1. shows the relationship between the agent objects, agent rule-sets, the tool-kit and remote simulation.

Agent Scheduling

The tool-kit scheduler is responsible for the correct running of the agents. The scheduler runs in a two pass operation. Firstly, it allows the agents to perform their mental processes, this is controlled via POPRULEBASE a forward chaining production system. Secondly, it passes messages between agents and runs any physical simulation, or external actions of the agents. The external simulation returns messages back to the agents reporting changes in state, or information from sensors. Running in this two pass manner ensures the behaviour is independent of the order of the agents, because all agents get to perform sensor and thinking activities before the actual state of any entity changes. It does however require that the time allowed to perform reasoning for all agents during the first pass should be short enough to ensure that updates to and from the simulation happen frequently enough to give reasonable behaviour. In many cases this requires that algorithms are

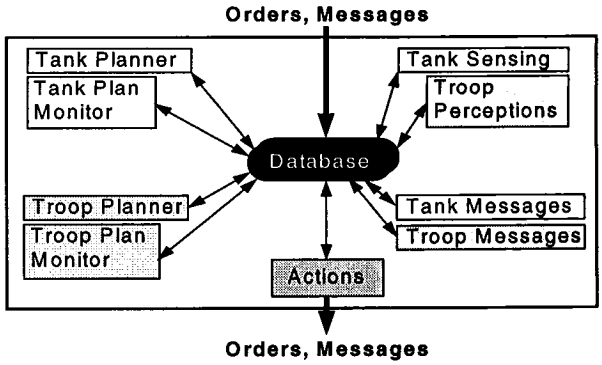

Figure 2. Example Troop Commander Architecture.

interruptible and operate in small incremental stages.

Internal Agent Mechanisms

Each agent has an associated collection of rule-sets known as its rule-system. A rule-set contains a collection of condition-action rules interacting via a database. The condition action components of a rule are not limited to a particular style, since they can invoke any POP11 function, it is possible to call other languages such as C++, Prolog, etc. The rule-sets are a method of grouping similar behaviour components. The rules can switch between rule-sets and databases, push them on a stack, restore them, etc. (c.f. SOAR (Laird 1993)).

The toolkit allows the rulesets to behave in a number of different ways, all applicable rules can be run in parallel or the most applicable rule can be selected by a number of criteria, including precedence (rules are ordered) or weighting schemes. In our application we have used the settings which allow for only one rule to fire at a time with the first applicable rule in an ordered list of rules being selected. The selection of a 'style' of rule matching is usually dependent on the style the programmer is used to.

Although learning is not included in our implementation, it is supported in the tool kit. A rule can introduce new rules or rule-sets within an agent.

Each agent within the hierarchy is based on the same architecture, Figure 2 shows the basic design. The fundamental properties of this design are:

- It contains a central database, through which all the rule-sets communicate. This database can be partitioned on a keyword, each sub-database holding related data, allowing searching to be performed much more quickly.
- Individual rule-sets can be identified to perform fundamental tasks, although their operation may be inter-linked Separating functionality enables parallelism of the rule-sets.
- The modules only simulate the agent's intelligence and do not perform any actual physical modelling. To perform actions the agent sends instructions to the physical simulator, and receives confirmation back about the action via its sensors. This allows separation

of the intelligence modelling from the physical simulation. Additional or modified behaviour can be easily implanted into the agent by simply loading different, or additional rule-sets into the agent.

Control of an Agent's Resources

Within the tool-kit there are several mechanisms for controlling resources, not all of which we use. The prime means of controlling agents is by limiting the number of rules which may be run within a rule-set on any pass. This may be set as a limit specific to each rule-set or apply to a rule-system as a whole. Additionally agents may run their rule-systems a multiple number of times in a pass or include rule-sets multiple times within their rule-system but we have not made use of these mechanisms. One of the effects of this is that rulesets may be written which would keep operating continuously but the scheduling will ensure that they can be considered to run in parallel with all other rulesets. Each time the scheduler is run it is given a list of agents to execute, each agent reports if it did not fire any rules on a pass, allowing it to be temporarily removed from the list of active agents.

Meta-level Reasoning

Reasoning about the computational resources and abilities available to an agent (sometimes called meta-level reasoning) is one of the features of the tool-kit which is still developing. Presently there are mechanisms known as rule-families which can be used to control which of a number of rule-sets are available to an agent at a given time. These can be used to focus attention on a particular problem, such as planning a route, to prevent time being wasted checking all the rules within a module when a clearly identified subset is all that is required.

Control of other aspects of reasoning, for instance the depth of a search the degree of abstraction to be used and how much real time to allow before execution must commence is done through setting parameters via rules which make modifications in the database. In theory this allows 'meta-management' rules to control the types of reasoning as well as the resources available to an agent depending on the present situation. In practice we have only just begun to explore the use of these mechanisms and most of the parameters used to control the resources an agent has remain fixed.

Discussion

The main feature of the toolkit which benefited our research was its flexibility and extensibility. Each ruleset can be configured to behave in a variety of ways, providing different mechanisms for breaking ties between rules and selecting actions. This combined with the ability to access arbitrary functions in conditions and actions allowed us to develop in a way which matched our domain and re-used existing code without being overly restricted by the toolkit.

The downside of this, however, is the lack of detailed support for specific architectural features which might be expected in a more specialised toolkit. Additions can be made to the toolkit but in most cases it is the programmer who has to design and code the architectural support which their agents need. The benefit of this is that a simple implementation can be made using the basic toolkit and supporting systems added as their need becomes apparent.

In particular for our research into battlefield simulation agents we had to add support for an external simulation, distributing agents across multiple machines, operating in real time, and a hierarchical representation of goals and plans.

Physical Distribution

Adding an external simulation and distributing the agents across several machines proved relatively easy since the design of the toolkit allows the programmer to overload methods used by the toolkit and so provide functions which mapped getting sensor data, performing actions and communicating to other agents onto network messages. Allowing agents themselves to migrate between machines would be a harder task since no absolute boundary is enforced between the agent, the toolkit and the operating system. While this allows all agents to access common data held within the toolkit (such as the large terrain database in our case) it does make mobility harder.

Real-time operation

One of the more difficult tasks which we faced was introducing the concept of real time operation into the toolkit. One of the reasons for this is that the resource control mechanisms described above do not include any reference to real time or the processor time, only to counts of rule firings. If real time operation is desired it is important to ensure that the actions performed by rules take a small amount of time to prevent single rule firings consuming large amounts of real time.

We attempt to achieve real time operation by constraining the amount of processing an agent can perform on a cycle (single pass of the scheduler) and by updating the agents from the simulation at regular time steps (typically two seconds).

The agents therefore operate by all running up to their rule limit (or completion) in the first pass of the scheduler at which point the scheduler checks to see if the next time step has been reached. If more time is available another pass is made through all active agents. This continues until all agents have completed (have no more rules they can run) or the time step is reached.

It is therefore impossible for an agent to predict how much processor time it will get in any slice of real time since this depends on the number of agents running and how much work they are doing. This requires the use of 'anytime' techniques which always have a solution of some sort available so that the agent can start executing it when desired or the use of continuations, functions which return

after a given amount of time with either a solution or the information needed to call the function again and progress towards the solution. For some techniques the requirement to write them in the form of continuations is a fairly onerous overhead (particularly is the code is re-used from a different application). The ability to run multiple threads for each agent would be a useful additional capability.
Related to the issue of running in real time is the response time of a single agent. By operating at a level of two second time steps our agents have a long response time and so have to rely on the simulation to perform fast responses. In many systems there are a variety of different time frames which are appropriate and more sophisticated resource scheduling might make it easier to combine fast reactions with complex deliberative algorithms.

Data storage

There are several ways of storing data in the toolkit. Agents have 'slots' (the equivalent of C++ class variables) which can be used to hold information, for example the position of an agent and information about its status. These slots are easy for external functions, such as those dealing with sensors, to access and their values can be used in the conditions of rules. Procedural knowledge is generally encoded in the rules and rulesets available to agents. In theory these can be altered although we have no experience in doing this. Some shared data can be stored in global objects available to all agents, for example our terrain database or information about the physical capabilities of various vehicle types. The prime storage location for data however is the central database, held separately by each agent. The basic toolkit comes with no distinction between types of data although for different applications different ways of updating and indexing data may be required.
In the work we have done and from experience at Birmingham University one of the desirable features has been shown to be some form of truth maintenance system. That is a means of automatically identifying dependent data entries and removing facts from the database when the evidence supporting no longer holds. One specific example (Baxter 1996) of this is in a hierarchy of goals, sub-goals and plans maintained by our agents. In this case not only is it useful to have unsupported data removed (i.e. sub-goals which no longer apply due to achievement of a higher level goal) but it is useful to be able to identify the relationships between plans and goal easily by some form of data tagging scheme. It is probable that future versions of the toolkit include some form of truth maintenance system. Clearly the support provided by a toolkit for different types of data representation will affect its applications. Learning applications tend to prefer homogenous data representations and physically realised systems (e.g. robots) need tight links to the real world and sensor data. In all cases it is also important to allow the user easy access to the data as an aid to debugging.

Debugging Agent systems.

As with any software system debugging agents is an important consideration. However this does not seem to feature early in development for toolkits. SIM_AGENT comes with a wide variety of tracing functions which can be configured to produce a wide variety of data about what an agent is doing. The equivalent of 'print' statements are available for inclusion in rules and can be dynamically enabled or disabled. By using similar mechanisms we have included the ability to display visual debugging information overlaid onto a graphical display. In most cases however locating errors from voluminous output files is an art form and potentially a tedious process. Debugging is especially difficult when errors are produced by timing problems in network communications or rule firings. One clear need for future toolkits therefore is a consideration of how to analyse and debug the performance of agents whose behaviour depends upon communication with other equally complex agents. This would seem to point to the need to be able to accurately instrument agent systems not only for debugging but to allow meaningful performance comparisons to be made.

Classifying toolkits

By examining the domain in which we operate and the facilities provided by the toolkit which we use it has become clear that there are several ways in which toolkits might be classified. One of the most complex areas which we have dealt with is the representation of time within our agents and coping with scheduling problems to allow us to run in real time. Even so the response time of the toolkit probably limits its application in hard real time systems where events occur every time a user types something into a desktop application for instance. The agents are also implicitly designed to run in parallel with and communicate with an environment, rather than being embedded within an application.
A major benefit of the toolkit has been its flexibility and the ability to add different types of functionality for different tasks. In categorising a toolkit it needs to be made clear what representation conventions are forced on the user (for example expressing all actions in a temporal logic). A general toolkit should provide libraries or possible representations and techniques rather than overly restricting the user. It must be borne in mind however that specialised toolkits tightly coupled to their domain are almost certain to be more efficient than general ones.
For a user selecting a toolkit we believe it would be useful to be able to measure the response or cycle time of a toolkit and to provide some idea of relative efficiency and flexibility. Flexibility seems too fuzzy a measure to be easily used but response times and a set of benchmarks might be easier to develop. Possible benchmarks could be response time to a simple message and the minimum time delay between posting a goal and generating an external action. This would give a lower bound on the response time of any agent implemented in the toolkit and enable

potential user to identify whether it would be possible to use the toolkit for their application.

Conclusions

We have briefly described the SIM_AGENT toolkit and the way we have used it within the domain of Computer Generated Forces. Without more experience with other toolkits in the same domain, e.g. SOAR (Tambe et al 1995), it is difficult to draw conclusions about how effective other tools would have been applied to this domain. The flexibility of the toolkit made it easy for us to develop domain specific enhancements, however this came at a cost in overall efficiency.

Facilities for making and reasoning about communication would seem to us to be a prime contender for addition to a toolkit (e.g. an Agent Communication Language and a model of expected responses) although it may be useful to have a clear distinction between an internal and external representation to allow the use of common external communication languages. We can also see the need for agent to be able to reason about their own capabilities and performance, which requires the ability to instrument and monitor performance. Finally in many cases we can see a need for internal self-simulation. This would allow agents to run a model of themselves internally to discover what their reactions would be in hypothetical situations and return some results which can be used in decision making.

References

Baxter J W. 1996 'Executing Plans in a Land Battlefield Simulation' In Proceedings of the AAAI Fall symposia, 15-18

Hepplewhite R. T. and Baxter J. W. 1996 Broad Agents for Intelligent Battlefield Simulation. In Proceedings of the 6th Conference on Computer Generated Forces and Behavioural Representation, Orlando, Florida: Institute of Simulation and Training

Laird, J. E. Clave, B. L. Erik A. and Roberts D. 1993. *Soar User's Manual (V6)*, University of Michigan, Carnegie Mellon University.

Logan, B 1998 http//www.cs.bham.ac.uk/~bsl/aaai-98/agent-classification.html

Sloman A. and Poli R. 1995 SIM_AGENT: A tool-kit for exploring agent designs. ATAL-95 Workshop on Agent Theories, Architectures, and Languages, IJCAI-95 Montreal, August.

Tambe, M., Johnson, W. L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloom, P.S., Schwamb, K. 1995 Intelligent Agents for Interactive Simulation Environments. *AI Magazine* 16(1).