Controlling Reasoning Within Battlefield Simulation Agents

J. W. Baxter and R. T. Hepplewhite

Defence Evaluation and Research Agency
St. Andrews Road,
Malvern,
Worcestershire. WR14 3PS
United Kingdom

(jbaxter,rth)@signal.dra.hmg.gb

Abstract

Computer Generated Forces (CGFs) are becoming increasingly important in battlefield simulations, in particular for reducing the number of operators required to control entities when training only a few military commanders. These entities must operate in a spatially and temporally continuous dynamic domain, react to any situation in a realistic but non-deterministic manner, using potentially uncertain information about the world. In order to operate large numbers of entities in real time tight constraints must be kept on the computational resources used. In this paper we describe the resource control mechanisms available to us within the "SimAgent" tool-kit and give an example of one of the 'anytime' search techniques we have used to ensure decisions can be made within the available resources.

Introduction

The simulations we are dealing with typically involve decisions being made about movement over a realistic terrain surface. Decisions are made according to how the terrain affects the ability of vehicles to move over it and how it affects their visibility to potential threats. This provides a very complex cost surface over which to search since small movements can dramatically change the cost as visibility to an enemy is made or lost.

The reasoning processes of our agents are based on a "broad agent" architecture (Bates, Loyall and Reilly 1991) implemented within a hierarchy of agents based on the military command and control (C^2) structure. A broad agent is designed to have a broad but perhaps shallow range of capabilities, instead of a few very detailed behaviours. The C^2 hierarchy enables commands to be devolved from high level commanders to subordinates and so each agent need only consider the appropriate level of detail.

In the following section we describe the way resource limits are applied within the agent tool-kit we use and the implications this has on the way we write the rule modules for our agents. The 'anytime' search mechanism used for route planning by the troop command agents to fit in with this framework is described and some comments made on the enhancements needed to the tool-kit to improve the ability to monitor resource limitations

Agent Tool-Kit

The framework for the agents has been developed in collaboration with Aaron Sloman at Birmingham University as a "SimAgent" tool-kit, written in Poplog (Sloman and Poli 1995). The "SimAgent" tool-kit executes multiple agents, controls the message passing between them and allows physical simulation of the agents.

Since, the precise agent architecture was not initially known the tool-kit needed the facility to support different architectures between the agents, and possibly a number of sub-architectures within the 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

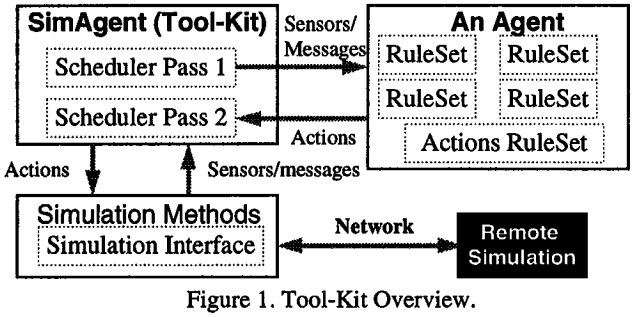

Figure 1. Tool-Kit Overview.

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 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 number of databases. 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 et al 1993)).

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
- The design of the intelligence is generic to any position in the C^2 hierarchy.

Additional or modified behaviour can be easily implanted into the agent by simply loading different, or additional rule-sets into the agent.

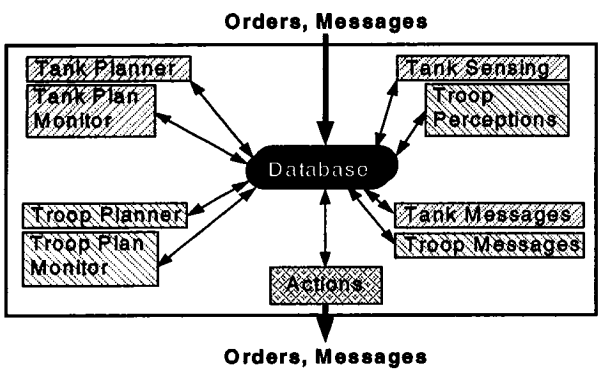

Figure 2. Example Troop Commander Architecture.

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. 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 removed from the list of runable agents until new information arrives.

It is important to note that none of these mechanisms include any reference to real time or the processor time, only to counts of rule firings. It is therefore important to ensure that the actions performed by rules take a small amount of time. In some cases we actively use the rule system to decide actions but in other cases, such as the troop route planner described in the following section, it is used simply as a means of controlling the resources available to an agent.

As the agents can interact with human controlled simulation entities, it is necessary for the agents to run in real time. This is achieved 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

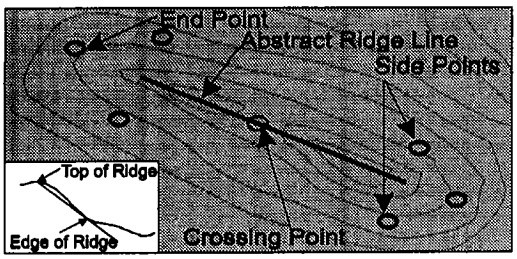

Figure 3. Example of ridge line abstraction.

when desired. One example of this is the troop concealed route planner described in the following section.

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.

Troop Agent

The troop commander agents are one level up the control hierarchy from individual vehicles (Hepplewhite and Baxter 1996) and they cover movements of several kilometers over the terrain rather than the few hundred meters over which the tank commander agent plans. Since the plans will be used as guidelines by the tank commander agent, rather than an exact description of their motion, troop plans can be made at an abstract level reducing the potentially huge search space to a more manageable size.

The troop commander agents make plans over a series of points based upon 'significant' terrain features, currently abstracted ridge lines. This is designed to enable troop commanders to consider large scale moves, such as driving round a hill, as a single step, and reduces the search space significantly to enable the planning to occur in a reasonable amount of time. Troop plans are currently based on consideration of the degree of concealment provided by a route from known enemy positions in addition to the time taken to travel a route.

To represent a ridge in the planning system a simple line is not sufficient, some idea of the size of the ridge is also important and potential crossing points. The ridge lines are abstracted manually and a simple algorithm used to extract the edge points. An example of the points produced from a ridge like feature is shown in Figure 3.

Costing of Routes

The cost function uses a combination of traversal time and exposure to enemy units. To model the effect of terrain, traversal time is based on the slope of the terrain profile. The speed reduction depends on how close the slope is to the maximum vehicle gradient. Only upward slopes have an effect and 'impassable' slopes are given an arbitrarily slow speed. The exposure of a route is calculated by using line of sight to find what percentage of a route segment (sampled at 50m intervals) is visible to observer positions. The costs are combined by multiplying the cost of exposed sections by a constant indicating the relative importance of speed and concealment. The heuristic function used in the search is the straight line distance divided by the maximum speed of the vehicle.

The Search Mechanism

The search is a variant of A* modified to work with complete routes over a fully connected graph. The search is required to find the lowest cost solution in the available time. Since tracked vehicles are able to move across most terrain types there is almost always a valid (if perhaps very slow) route from any given node to all others. The terrain abstractions we are using typically yield on the order of a hundred nodes for start and goal points five kilometers apart.

The search starts by considering the cost of the route direct from the start to the goal and then considers all single stage routes (routes which travel via a single intermediate node). This gives an upper bound on the route cost and also identifies the direction in which search can be carried out with the lowest apparent branching factor. A* search then proceeds, expanding in the direction with the lowest apparent branching factor. The following description assumes expansion from the start towards the goal although either direction is possible.

The node ('significant point' on the terrain) which has the lowest expected path cost, that is the known cost to the node plus the heuristic cost to the goal, is selected for expansion. In general the branching factor at each node would appear to be the number of nodes in the graph. In practice it is considerably lower than this since nodes can be rejected if the expected path cost through them exceeds the current lowest cost plan. The cost of each valid expansion is computed and if found to be less than the existing cost of getting to that node then the cost to it and route to it are amended. The planner also checks the (known) cost to directly complete the route to see if this

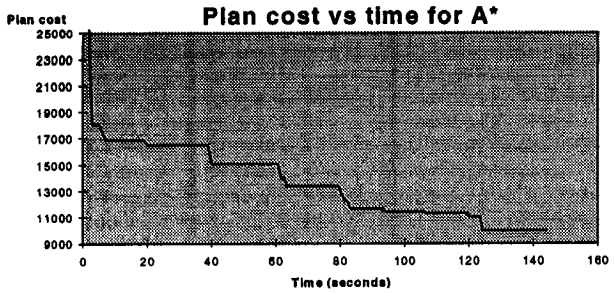

Figure 4. Plan cost against Planning Time

provides a cheaper route than has been found so far and updates its best plan accordingly. This means that at any time the best route found so far can be executed, allowing the troop commander to respond quickly when necessary.

The routine is also memory bounded since all that needs to be stored are the nodes (significant points) and the best known routes to and from the node. The search terminates when no node has any possible extensions which could yield a lower cost route.

Figure 4 shows how the initially high cost for a route was improved over time with an initial very fast reduction in cost as the search quickly identified how to avoid areas which were very exposed to the enemy and then showing a steady improvement as the route was refined.

Integrating with the Agent

The planner thus has several desirable features for allowing the agent to reason with limited resources. The representation of all nodes is the same throughout the search allowing a fixed structure to be used and stored in the database between invocations by the agent. The progress of the search can be stored simply by recording which node is being expanded and how far along the list of other nodes that expansion has proceeded. The resources used in each invocation of the planner can therefore be limited by controlling how many nodes can be considered each time it is called. The rule-set controlling the planner can therefore call the planner as many times as its rule limits allow before it is forced by the scheduler to pass control to other rule-sets or other agents. The rule-set can use references to a hard time limit, or other context dependent constraints to decide when to pass the best plan found so far out for execution and can decided whether or not to continue looking for improvements to a plan it is currently executing.

Executing troop level plans.

The route planner has been incorporated into the troop commander agent which uses it to plan routes for the troop (group of three to five tanks) to follow. Routes are planned based on known enemy positions. The plans are executed by agents controlling each individual tank in the group who try to keep together and move in a group while responding sensibly to the terrain they are moving over and the presence of enemy tanks. The troop commander

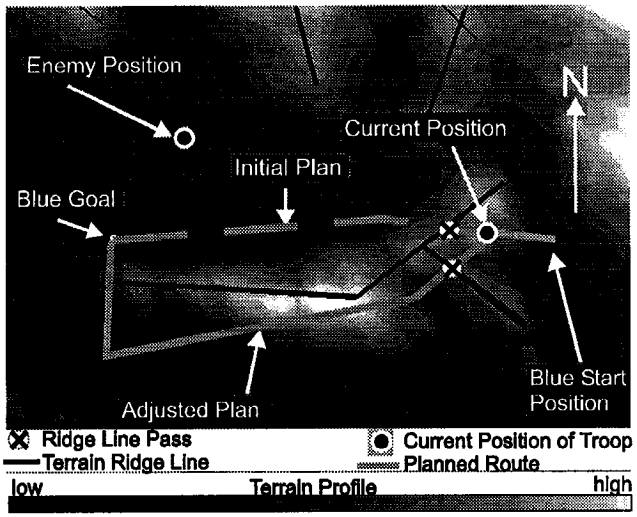

Figure 5. Initial troop plan and a re-plan on spotting the enemy.

monitors the progress of the group and may re-plan the route if an enemy which was not previously known about is detected. The behaviour this produces can be seen in Figure 5.

The blue troop initially planned to cross the ridge in front of it and proceed towards their goal along the northern edge of the ridge. As it reached the top of the hill one of the tanks spotted the red troop in the valley and informed the troop commander. This resulted in a re-plan which caused the blue troop commander to adjust its plan to follow the southern slopes of the ridge instead, first crossing the smaller ridge to the SW.

Conclusions

We have described the features of the SimAgent tool-kit which allows resource limitations to be applied and described how we have used them to enforce real time operation in agents for military simulations. We have also briefly described an 'anytime' planner for concealed routes over terrain used by these agents.

Clearly the present mechanisms allow us to apply some resource limits but place the onus on the programmer to ensure that no particular rule or rule-set consumes an 'excessive' amount of processor time. It is probable that the SimAgent tool-kit will require more sophisticated resource monitoring and control mechanisms to allow a 'meta-management' layer to control the way in which resources are allocated to problems.

The planner described in this paper shows the features which we believe to be necessary for situations where planning or search cannot be guaranteed to continue to completion. It is easily interruptible enabling mechanisms which can adjust its performance and control the overall resources used by an agent to gain feedback from it and temporarily or permanently halt it. It always has the best plan so far ready for execution and continuously improves

on that plan given more time. It is also able to recognise when it has produced the best plan it can find and the search can be terminated.

At least part of the reason that the plans produced can always be executed by the troop commander is because they are made at an abstract level and so are further expanded by the individual tank commander agents for final execution within the simulation.

Further Work

We would like to investigate the control of reasoning within agents further, particularly as they grow more complex. We are interested in applying techniques which maintain plans at varying levels of abstraction, depending on how close they are to execution. We are also interested in examining how to allow agents several different ways of solving a problem with different time profiles and reasoning about how to share resources between these techniques. We believe that search and planning techniques cannot operate simply as 'black boxes' or closed processes they must feed back information about the search and the search space to assist the agent in making decisions about how to balance different planning and execution methods.

Acknowledgments

We would like to acknowledge the assistance of the cognition and effect project at Birmingham University, particularly Aaron Sloman and Brian Logan, for sharing their ideas on agents with us and for producing and maintaining the SimAgent tool-kit. We would also like to thank our colleagues at DERA Malvern for the valuable suggestions they have made about the agents and the route planner.

References

Bates, J, Loyall , A. B. and Reilly, W. S. 1991 Broad Agents. *Sigart Bulletin*.

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.

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.