To appear in proceedings Game-On2000
Imperial College London, 11-12 November, 2000

REAL-TIME GOAL-ORIENTATED BEHAVIOUR FOR COMPUTER GAME AGENTS

Nick Hawes
School of Computer Science
The University of Birmingham
Birmingham
B15 2TT
United Kingdom
E-mail: N.A.Hawes@cs.bham.ac.uk

KEYWORDS

Agent architectures, anytime algorithms, computer game agents, planning, real-time performance.

ABSTRACT

To increase the depth and appeal of computer games, the intelligence of the characters they contain needs to be increased. These characters should be played by intelligent agents that are aware of how goals can be achieved and reasoned about. Existing AI methods struggle in the computer game domain because of the real-time response required from the algorithms and restrictive processor availability. This paper discusses the CogAff architecture as the basis for an agent that can display goal orientated behaviour under real-time constraints. To aid performance in real-time domains (e.g. computer games) it is proposed that both the processes encapsulated by the architecture, and the information it must operate on should be structured in a way that encourages a fast yet flexible response from the agent. In addition, anytime algorithms are discussed as a method for planning in real-time.

INTRODUCTION

Computer games are particularly suited to agent based AI because almost every computer game features characters. Whether these characters are faceless guards in a top-secret facility, players on a football pitch, or evil plumber-battling princess-kidnapping despots, they can all be viewed as examples of (more or less) intelligent agents. The field of agent building in AI is very wide ranging, incorporating robotics, simulation, philosophy, vision and others. Computer games need the application of these techniques if they are to increase the intelligence of their characters, and consequently the appeal and quality of the games (van Lent et al., 1999).

There are currently two existing projects that aim to produce intelligent agents specifically for computer games. These are the EXCALIBUR project and the Soarbot project. The Soarbot project aims to create bots ("human level" computer characters) for Quake2 using the the Soar architectural framework for intelligent agents. The EXCALIBUR project is less game specific and is aimed more toward developing a system for controlling intelligent agents in computer game-like environments.

The rest of this paper will first briefly outline why agents should demonstrate goal orientated behaviour, and then introduce the CogAff architecture. This will be followed by some thoughts on one part of the architecture, how the data an agent processes should be structured, and how anytime algorithms can be applied to real-time planning.

GOAL ORIENTATED BEHAVIOUR

When an agent is situated in an environment such as a computer game, there are a number of things that the agent can do to appear more intelligent (and hence a more believable character). One of the most important of these is to act in a way that demonstrates an awareness of goals. This awareness can range from simply having "built-in" goals (e.g. the hard-wired goal to eat when hungry), to generating completely original goals when the situation requires it (e.g. the goal to help the little old lady that has fallen over in front of you), and reasoning about the relative importance of different goals (e.g. deciding that helping the old lady is more important than buying a sandwich).

By achieving goals the agent demonstrates that it is proactive (not just responsive), that it can reason about what should be important in the current situation (e.g. social interaction, accumulation of important physical objects, protecting its family), and that it can find ways in which it can affect the achievement of these goals. Goal orientated behaviour is also one of the more widely accepted properties that distinguishes an agent from other artifacts (Franklin and Graesser, 1997). A typical agent will have a large number of interacting goals, ranging from the critical (e.g. survival), to the obscure (e.g. collecting Pokemon trading cards), to the mundane (e.g. getting to work).

THE COGAFF ARCHITECTURE

The CogAff architecture is a three-layer agent architecture developed by the Cognition and Affect project at the University of Birmingham. Its structure can be seen in Figure 1. The ideas and design decisions behind the architecture are detailed in a number of the Cognition and Affect project's publications e.g. (Sloman, 2000; Sloman, to-appear). The following list provides a shallow, whistle-stop tour of the architecture's key features.

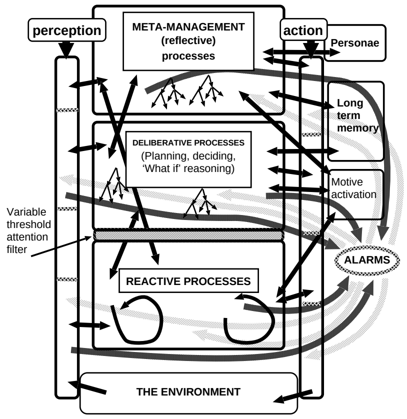

Figure 1: The CogAff Architecture

- Reactive Layer: Reactive processes that act based on the current world state or the agent's internal state.
- Variable Threshold Attention Filter: A filter that prevents less-than-urgent goals from overloading the processing of the deliberative layer (cf. (Logan, 2000)).
- Deliberative Layer: Supports the agent's "what-if" reasoning and deliberative decision making.
- Meta-management Layer: Monitors and evaluates internal control and processing issues.

The design of the architecture is such that an agent can make the most of its limited processing resources, and is able to react to important environmental events in a timely fashion. Both these facts are crucial to the implementation of intelligent agents in computer games. The use of processing time is maximised by the attention filter. The filter is used to inhibit the progress of goals (requests for processing) generated by the reactive layer. Its basic operation can be summarised as follows; every goal generated is assigned a heuristic measure of insistence, if this level is greater than the one encoded by the filter it is allowed to "surface" and influence processing in the deliberative layer (Beaudoin and Sloman, 1993). A timely (although not always optimal) response to environmental change is guaranteed by having concurrently operating layers. If the deliberative layer is engaged in a processing task, the reactive layer will always be available to react to the current situation.

An example of such concurrency in a game could be demonstrated by a player in a football game running with the ball in space. The reactive layer keeps the player running in the right direction and keeps the ball under control. Whilst this is happening the deliberative layer can plan how and when to pass the ball. Distinct levels of processing can also be witnessed in a wide variety of human activities. For example, our breathing is constantly controlled whilst we happily contemplate our day to day problems. Existing examples of agents using the CogAff architecture include Ian Wright's MINDER1 (Wright, 1997) and Steve Allen's concern-centric agents (Allen, 2000).

The Deliberative Layer

Within the CogAff architecture, decision making is carried out in various ways. In the reactive layer, decision making is based on selecting a "hard-coded" action rule that matches the current situation. The deliberative layer's "what-if" reasoning requires the generation and evaluation of possible future states. Because of this, a processing method more sophisticated than reactive rule execution is required. Reactive processing cannot be used because purely reactive methods have difficulty defining paths to future states when their preceding situations cannot be accurately and wholly anticipated. Such situations will often occur in complex and dynamic computer game worlds. A suitable method for deciding actions for future states is generative planning. Examples of generative planners include Cassandra (Pryor and Collins, 1993) and STRIPS (Fikes and Nilsson, 1990). The behaviour of such systems could theoretically be replicated by reactive rules, but such a large set of rules would be required that (a) development time would be extensive, and (b) the processing time required to match a rule to the current state would be prohibitively large.

The deliberative layer is the driving force behind making the decisions that ultimately direct the behaviour of the computer game agent, therefore it is this layer that we must focus on when aiming to develop goal-orientated behaviour. Different instantiations of the CogAff architecture can contain vastly different deliberative layers. At one extreme it could feature some simple pattern matching that evaluates possible future states based on the current one, and retrieves a stored plan from memory. At the other you may see a complex system of interacting processes (i.e. belief maintenance, generative planning, machine learning, plan recognition, opponent modelling and goal management). When using any vaguely complex reasoning system, the deliberative layer is likely to become the agent's processing bottleneck. This is because planning in anything but simple domains will have to deal with computationally intractable search spaces (Chapman, 1985). The consequence of this is that an agent may get caught up in "what-if" deliberation when its attention is really required elsewhere. This will prevent an agent from acting effectively in real-time, as in the long-term its reactions will not be as successful as plans. To combat this problem the functional elements within the deliberative layer must (a) be structured and controlled in a way that allows them to be highly responsive to the real-time dynamics of the agent's environment, and (b) provide affective responses even when their processing time is limited or unpredictable.

STRUCTURAL CONSIDERATIONS

To enable deliberative processing to occur in real-time (at run-time), it will be necessary to structure both the information that the agent is to deliberate with, and the processes that will perform the deliberation. This structuring should aim to encourage fast responses from the deliberative layer of an agent, preferably without the loss of expressive power.

Information Structure

It is important to structure the data that an agent must reason with. This allows search to be performed in a more efficient and directed manner. For example, using heuristics when performing search represents a way of structuring data. Heuristics structure the search space into regions of "good" and "bad" solutions or partial solutions. The following paragraphs will discuss two possible ways of structuring the search space an agent faces in a computer game.

The first method of imposing structure is plan waypoints. These are a similar concept to navigation waypoints: a point (in plan space) that the agent must pass through in order to progress. If we imagine a hierarchy of abstractions spanning the entire of an agent's operation (i.e. a plan from the start of a game to the end of a game), waypoints represent subgoals very high up in this hierarchy. They could be generated in various ways (not just through planning), or could be specified offline to generate desired behaviour (e.g. collect 100 gold coins - buy a sword - find player X - engage in combat). By focusing an agent's deliberative functions on achieving waypoints we can reduce the time an agent must spend planning. This is because the agent will only have to tackle easily manageable chunks of planning, and hence will be able to return a result quicker. Waypoints will also increase the efficiency of the planning process. By strictly limiting the future projection of an agent's plans, we reduce the probability that intended actions will become invalid because of changes due to the dynamic nature of computer game worlds.

A second possibility for structuring the information an agent faces is through the role it plays in the game. If we use the idea of a theatrical role (i.e. a part that an actor must play, and that defines their behaviour), we can view agents as actors playing roles in computer games. A role will determine superficial features of an agent like appearance and voice, but roles can also be used to specify behaviour-relevant characteristics. These can include physical properties such as strength or speed, or more interestingly, cognitive properties such as intelligence, perceptiveness, or preferences for or against objects or behaviours (e.g. a hatred of spiders or a love of guns that make loud noises). These characteristics can be used in a similar manner to search heuristics, in defining and focusing processing on regions within the search space that will more readily offer suitable solutions. Such characteristics could also lead to very varied behaviour being produced by similar agents with different roles.

Architectural Structure

The essence of an agent architecture is the presence of various heterogeneous processes linked together by a flow of information. To produce an agent that can deliberate in real-time, whilst staying responsive to its environment, we must take full advantage of architectural structure. At one structural extreme an agent could have a monolithic deliberative layer which dealt serially with every different type of reasoning and kept track of all the changes in the environment. The problem this presents is that it would risk lagging behind the world if it was processing a problem when it really should be updating its representation, or monitoring important events. The other structural extreme would be a deliberative layer consisting of an independent intelligent process for each necessary job (e.g. a process that did all the planning in isolation, a process that maintained beliefs in isolation etc.). This would lack efficiency because processes would need to have duplicate features (e.g. environmental monitoring, goal representations) if they were to operate successfully, and such a level of functional independence would no doubt throw up a number of conflicts when processes acted concurrently.

Luckily we only have to use such extremes as scare tactics when making a point, and are not restricted to them when developing agent architectures. To create a successful deliberative layer we need to combine the key feature of the first extreme (control over a variety of integrated processes), with the key feature of the second (concurrently active processes), and create something better than both of them.

The asynchronous operation of architectural elements is an idea commonly used to facilitate real-time operation (cf. qualities for real-time success in (Hayes-Roth, 1990)). The advantage of this is demonstrated by the concurrent independent layers of the CogAff architecture. If we apply the concept to the elements within a single layer, a similar advantage can be gained. If we divide the functions of the deliberative layer into a number of independent yet communicating modules, each gets the advantage of a relatively autonomous operation (e.g. allowing it to manage its own resources), and the agent gets the advantage of a responsive and flexible deliberative layer. The structure of the deliberative layer should allow for asynchronous processes for at least planning, belief maintenance, and motive maintenance. Motive maintenance should handle the adoption, management and deliberative generation of goals, as well as dealing with roles and waypoints (this could easily be decomposed further).

ANYTIME PLANNING

Even with an architecture that is flexible with regard to time demands, any planning process will pose possible performance related problem. This is because planning in complex environments is, as mentioned previously, intractable. To be a manageable and fully integrated part of the structure of the deliberative layer, a planning algorithm requires some additional features. Primarily we would like a planner that can be monitored and controlled (interrupted, redirected, etc.) without hindering its performance. This would, for example, allow an agent to interrupt planning how to attack an enemy when it needed to plan to escape from imminent danger. It would also be desirable to have a planner that could be executed for a fixed amount of time and then have it return a result. This could be used, for example, if the agent knew how long it would have until certain environmental features changed. Anytime algorithms (Dean and Boddy, 1988) lend themselves ideally to these desired behaviours.

The underlying concept of an anytime algorithm is that as processing time increases, so does the quality of the result returned. For example, drawing a picture could be considered an anytime algorithm; the longer the artist has to spend on the work, the higher the quality of the result. An example of a non-anytime algorithm is searching for your car keys; either you've found them or you haven't. Because of the steady improvement in the results of processing, it is possible to interrupt the algorithm at any time to return a solution (the earlier you interrupt the lower the solution quality). To produce an algorithm that can function in this manner, a number of desirable properties must be present (Zilberstein, 1996). One of the most important of these properties is consistency of improvement. A consistent algorithm provides output quality that correlates well with computation time, allowing a performance profile to be built (Dean and Boddy, 1988). A performance profile is a graph of output quality against time which can be used to probabilistically determine the outcome of an anytime algorithm. Performance profiles are needed to allow reasoning about the operations of anytime algorithms (e.g. whether it is worth interrupting a process yet or if an interruption is forced, whether the resulting plan will be of a usable standard). Unfortunately knowledge-based algorithms such as planning are not always "well-behaved"; their output does not correlate well to their computation time. This makes it difficult to construct performance profiles for them (Mouaddib and Zilberstein, 1995). Some previous work has presented different approaches to this problem.

Work done on the EXCALIBUR project has produced a planning model based on structural constraint satisfaction problems (SCSPs) (Nareyek, 2000a). Local search techniques are used to explore the constraints that define the planning problem. Structural constraints allow the local search method to modify not just the instantiations of the constraints on the plan (as in traditional constraint satisfaction problems), but the entire structure of the plan. The use of local search means the increase in complexity of the plan occurs in an iterative manner and hence results in a predictable performance profile. The early results of this method look promising (Nareyek, 2000b).

(Mouaddib and Zilberstein, 1995) present the concept of progressive processing. This method groups together the knowledge and operators that represent a particular level of granularity. Operators at one level of granularity can only process data that have the same level of granularity, thus limiting the immediate search space. After every reasoning cycle an evaluation is made about whether to continue reasoning at the current level, or to use more specific operators. Processing in this way gives the knowledge based algorithm a much more predictable performance profile because the solution quality increases steadily with the granularity of the processing.

Of these two methods, the concept of progressive processing is most suited to agent-based real-time planning. This is because it lends itself to the use of hierarchical planning techniques (each level of the hierarchy represents a level of granularity). This is important because it allows the agent to take advantage of goal-subgoal instability (Wood, 1993, p27). This method specifies stable high-level goals first, and then does not specify their less stable subgoals and action primitives until it is absolutely necessary. This is useful in dynamic environments because plans can be generated that encode the goal being planned for without setting in stone the precise (primitive) details of how it is to be achieved (the part of the plan that will probably change). To reflect the situated nature of a computer game agent (i.e. that it is directly effected by the results of its actions, and not an unaffected advisor) it may be necessary to modify the progressive planning model slightly. Instead of always planning at uniform levels of granularity, it may benefit the agent to expand early plan subgoals to a greater degree of specificity before expanding later ones. This would mean that if a planning process gets terminated prematurely, the agent will have concrete actions to execute immediately. Unfortunately such a modification presents two problems. The first is that it may not provide such a predictable performance profile as the original algorithm. This is because the purely progressive nature would be replaced with a series of progressions and regressions. The second is the issue of subgoal interaction. If early subgoals are expanded to a greater degree than later ones, this may prohibit important choices later in the planning process.

A problem that arises when considering anytime progressive processing in a situated agent is: how does the agent execute the incompletely specified plans that are returned after the planner has been interrupted? If we only allow the planner to only return any complete plans found, we lose the predictable performance profile (a similar method to this was used in (Blythe and Reilly, 1993)). But not doing this means that some parts of the plan may still be in an unexpanded or abstract form. A possible (yet far from satisfactory) answer to this would be to take advantage of the architecture that the planner is implemented within. The reactive layer of the agent (assuming a CogAff architecture) must have access to a number of precompiled reactive plans (e.g. the runaway quick plan or the pick up health plan) for use when the planning process is not available. If this plan library was to contain some sketch plans for more abstract actions (e.g. clear the area of enemies, or get to a higher floor in the building), then the agent would be able to act with some degree of intelligence. Some definite restrictions would have to be placed on this method. The sketch plans should be fairly low in specific detail thus allowing them to be executed in many different circumstances. This would also maintain the importance of agents producing more complete plans; a complete plan would still have a higher likelihood of success than an incomplete one. There should also be a restriction on the level of abstraction that the sketch plans can describe. It would be thoroughly pointless for an agent builder to develop a planner and then give the agent reactive implementations of all the possible actions anyway.

CONCLUSIONS & FUTURE WORK

So far there are very few solid conclusions to draw from this work. This paper has outlined more a direction for research than an account of its results. What has been made clear is that to increase the intelligence, and hence the appeal of computer game agents, it is necessary to give them the ability to make goal orientated decisions in real-time. This equates to planning in real-time. Planning on its own will be ineffective, so it must be embedded in an architecture that allows the agent maximum real-time flexibility. The architecture proposed is the CogAff architecture. It is suitable because it is geared toward agents with limited resources in dynamic environments. The deliberative layer of the architecture, and the data it will process should be structured to facilitate real-time operation by the agent. It is clear that the planning system for a computer game agent should be based on an anytime algorithm (or possibly a collection of them), but knowledge based systems do not often display the necessary properties required for a "well-behaved" anytime algorithm. The concept of progressive planning is used as the solution to this, although alterations may need to be made for situated planning. Planning concepts that were not mentioned but need to be addressed are the problems caused by a dynamic environment. These are problems such as information uncertainty and the variable success of actions. Solutions to these will involve interleaving planning and plan execution, and monitoring the environment and the outcomes of processes and actions. All concepts will benefit from prototype implementations and design refinements. The initial implementations will be done using POP-11 and the SIM_AGENT toolkit because the language allows very quick and easy incremental development. A working implementation within a commercial game would require a language that executed a lot faster. A possibility for this would be the RC++ language developed at Sony. It has a number of similarities to SIM_AGENT and so porting the code should not pose any major problems.

NOTES

This research is supported by sponsorship from Sony Computer Entertainment Europe. The Cognition and Affect project's homepage and publications can be found at http://www.cs.bham.ac.uk/~axs/cogaff.html. The homepage for the EXCALIBUR project can be found at http://www.ai-center.com/projects/excalibur/, and the Soarbot project at http://ai.eecs.umich.edu/~soarbot/.

REFERENCES

Allen, S. (2000). "A concern-centric society-of-mind approach to mind design." In *Proceedings of the AISB'00 Symposium on How to design a functioning Mind* (Birmingham, UK, Apr.17–20). AISB, 135–136.

Beaudoin, L. P. and Sloman, A. (1993). "A study of motive processing and attention." In *Prospects for Artificial Intelligence*, Sloman, A., Hogg, D., Humphreys, G., Ramsey, A., and Partridge, D., eds. IOS Press, 229–238.

Blythe, J. and Reilly, W. S. (1993). "Integrating reactive and deliberative planning for agents." Technical Report CMU-CS-93-155. School of Computer Science, Carnegie Mellon University.

Chapman, D. (1985). "Planning for conjunctive goals." AI Technical Report 802. MIT AI Lab.

Dean, T. and Boddy, M. (1988). "An analysis of time-dependant planning." In *Proceedings of The Seventh National Conference on Artificial Intelligence* (St Paul, Minnesota, Aug.21–26). Morgan Kaufmann, 49–54.

Fikes, R. E. and Nilsson, N. J. (1990). "STRIPS: A new approach to the application of theorem proving to problem solving." In *Readings in Planning*, Allen, J., Hendler, J., and Tate, A., eds. Morgan Kaufmann, 88–97.

Franklin, S. and Graesser, A. (1997). "Is it an agent, or just a program?: A taxonomy for autonomous agents." In *Intelligent Agents III*. Springer-Verlag, 21–35.

Hayes-Roth, B. (1990). "Architectural foundations for real-time performance in intelligent agents." *Real-Time Systems: The International Journal of Time-Critical Computing*, 2:99–125.

Logan, B. (2000). "A design study for the attention filter penetration architecture." In *Proceedings of the AISB'00 Symposium on How to design a functioning Mind* (Birmingham, UK, Apr.17–20). AISB, 94–101.

Mouaddib, A. I. and Zilberstein, S. (1995). "Knowledge-based anytime computation." In *14th International Joint Conference on Artificial Intelligence* (Montreal, Canada, Aug.20–25). Morgan Kaufmann, 775–781.

Nareyek, A. (2000a). "Open world planning as SCSP." In *Papers from the AAAI-2000 Workshop on Constraints and AI Planning*. AAAI Press, 35–46.

Nareyek, A. (2000b). "Beyond the plan-length criterion." Available from http://www.ai-center.com/projects/excalibur/.

Pryor, L. and Collins, G. (1993). "Cassandra: Planning for contingencies." Technical Report No. 41. The Institute for the Learning Sciences, Northwestern University.

Sloman, A. (to-appear). "Architecture-based conceptions of mind." In *Proceedings 11th International Congress of Logic, Methodology and Philosophy of Science*. Kluwer. Available from http://www.cs.bham.ac.uk/research/cogaff/0-INDEX00-05.html

Sloman, A. (2000). "Architectural requirements for human-like agents both natural and artificial. (what sorts of machines can love?)." In *Human Cognition And Social Agent Technology*, Dautenhahn, K. eds. John Benjamins Publishing, 163–195.

van Lent, M., Laird, J., Buckland, J., Hartford, J., Houchard, S., Steinkraus, K., and Tedrake, R. (1999). "Intelligent agents in computer games." In *Proceedings of The Sixteenth National Conference on Artificial Intelligence* (Orlando, Florida, Jul.18–22). AAAI Press, pages 929–930.

Wood, S. (1993). *Planning and Decision Making in Dynamic Domains*. Ellis Horwood.

Wright, I. (1997). *Emotional Agents*. PhD thesis, School of Computer Science, The University of Birmingham.

Zilberstein, S. (1996). "Using anytime algorithms in intelligent systems." *AI Magazine*, 17(3):73–83.