An Anytime Planning Agent For Computer Game Worlds

Nick Hawes

School of Computer Science, The University of Birmingham, United Kingdom
nah@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~nah

Abstract. Computer game worlds are dynamic and operate in real-time. Any agent in such a world must utilize techniques that can deal with these environmental factors. Additionally, to advance past the current state-of-the-art, computer game agents must display intelligent goal-orientated behaviour. Traditional planners, whilst fulfilling the need to generate intelligent, goal-orientated behaviour, fail dramatically when placed under the demands of a computer game environment. This paper introduces A-UMCP, an anytime hierarchical task network planner, as a feasible approach to planning in a computer game environment. It is a planner that can produce intelligent agent behaviour whilst being flexible with regard to the time used to produce plans.

1 Introduction

As the standard of interactive entertainment (and computer and video games in particular) advances, the need for intelligent autonomous non-player characters is growing. The reasons for this need have been discussed in numerous other places (e.g. [17]). There are a number of projects that are currently exploring approaches for designing and building intelligent agents for interactive entertainment. These include the Excalibur project (e.g. [11]) and the Soarbot project (e.g. [9]). This document will introduce an approach to developing intelligent agents for computer games that is based in part on the ongoing work of the Cognition and Affect group[1]. There are two main points of interest in this approach, the first is the agent architecture, and the second is the anytime planner used as the core of the agent's behaviour generation facilities. The rest of this document will start by introducing some ideas about agent architectures, and will then go on to discuss some of the issues involved with using a planner based on an anytime algorithm as part of an agent architecture. This will be followed by a description of some of the more important sub-systems of an implemented agent that uses an anytime planner to generate future behaviour. Finally, some thoughts on the benefits (for both game agents and developers) of using an anytime planner will be presented.

[1] See http://www.cs.bham.ac.uk/~axs/cogaff.html for a collection of papers and resources.

2 An Agent For Real-Time Intelligent Behaviour

Figure 1 represents the CogAff architecture schema for intelligent agents [16]. It represents a superset of a wide class of possible architectures that could be designed to support an intelligent agent. One possible subset (i.e. a selection and instantiation of grid sections) of this represents agents that will be able to perform intelligently in real-time, game-like environments. It is the ultimate aim of the research presented here to develop such an agent. Although this will mean deciding on one particular architecture instance, this will not be the definitive agent architecture for real-time behaviour in game-like worlds. It is more likely that there will be a group of features that will be necessary in most examples of such agents, with additional features being added depending on the specific demands of the particular game world. Two components that will certainly be necessary in any agent intended to achieve a variety of goals in a dynamic environment are a deliberative system and a reactive system.

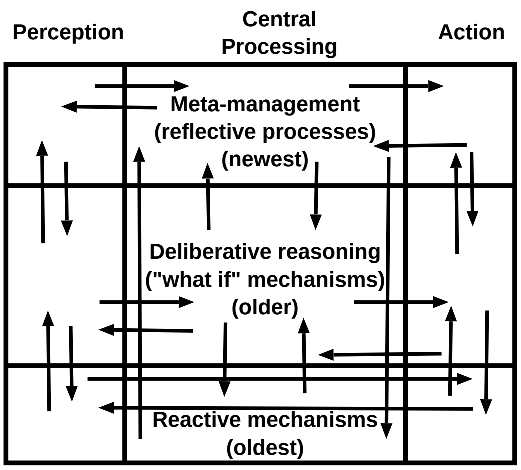

Fig. 1. The CogAff Architecture Schema

The functionality of a deliberative system can range from using simple affective states, such as those demonstrated by the A-Agents in [15], to something more complex such as anticipation [8]. This must be complemented by the necessary reactive mechanisms to keep the agent up to date with the current world state, extricate the agent from dangerous situations that may occur suddenly, and to perform actions that require feedback-based control (e.g. targeting a moving object).

A principle component of most deliberative layers will be something that can reason explicitly about future actions and states. Such a component can be used by an agent to generate behaviour which will allow it to achieve non-trivial goals in its environment (such as a capturing a flag, or accessing a restricted area). Traditional planners fulfill the role of generating future behaviour, but do not lend themselves well to the kind of interactivity required for successful performance in an architecture for an agent situated in a dynamic, complex environment. To deal with such problems, an anytime planner (a planner implemented as an anytime algorithm [3]) has been developed to act as the deliberative behaviour generation method for agents in computer game worlds. The remainder of this paper will present the reasoning behind this approach, some of the issues involved in developing A-UMCP (the Anytime Universal Method Composition Planner[2]) and finally some interesting features from the agent that uses it.

3 An Anytime Planner For Agent Behaviour

As mentioned in the previous section, an anytime planner is intended to form the core of the game agent's deliberative layer. As such, it will work in combination with other deliberative modules (belief maintenance, goal generation etc.) to generate the majority of the agent's goal-directed behaviour.

A planner is a desirable part of the agent's deliberative layer for a number of reasons. Planning allows an agent to choose efficient orderings of possible actions, and to work within resource bounds (e.g. produce a plan that only uses a certain amount of fuel). By having an explicit representation of its future actions (as opposed to an implicit one encoded in reactive rules), other parts of the agent can query the representation with regard to what is expected to occur in the future (e.g. if the agent is aware that the current plan will take it into a dangerous situation it could increase the amount of processing time devoted to its senses). Also, if plans are successful they can be stored for future reuse (or modification for similar situations).

Unfortunately, traditional planning systems have a number of drawbacks that are particularly relevant in the field of interactive entertainment [3]. Because planning suffers from a combinatorial explosion, it can be a slow, processing intensive problem. In domains which require the agent to formulate behaviour quickly (e.g. the majority of 'action' games) and/or systems with only a limited availability of processing time for AI, a traditional planner could prove more of a hindrance than an advantage for an agent that used one regularly. If the planner could be made to operate as an anytime algorithm [3], it would be possible to lessen the impact that a planner would have on the resources available to an agent [5]. It is important to note that using an anytime planner will not provide solutions to all of the problems planning faces in a dynamic environment. For instance, the fact that the world can change during a planning process is not explicitly handled by an anytime planner. It would be the responsibility of the agent architecture to ensure that this did not effect the agent's overall behaviour. For example, the

[2] A-UMCP is an anytime extension of the UMCP hierarchical task network planner [4].

[3] See [13] for a full summary of the weaknesses of traditional planners

agent could interrupt the planning process in advance of predictable changes to the world, or it could invoke a separate plan repair mechanism.

An anytime algorithm is a type of algorithm that can be interrupted at any time and will immediately return a usable result. A constraint on this behaviour is that the quality of the result returned must never diminish as processing time increases. An anytime planner is a planner that behaves in this manner. The two main issues that must be addressed when developing an anytime planner are; what happens after an interruption occurs, and how can the quality of partial results be measured? The full requirements for a functioning anytime algorithm are laid out in [19], and a more detailed description of how they can be applied to planning can be found in [6].

3.1 Interrupting A Planner

Because planners typically work in discrete cycles, actually halting one is a trivial problem. The real problem is what agent behaviour will be generated from the result that is returned. If a forward-chaining total-order planner is interrupted it will return a plan that would take the agent from its start state, to an arbitrary end point in the search space. There would be no guarantee that this end point would be any closer to the agent's goal. If you were to interrupt a backward-chaining total-order planner, you would be left with a plan that would take an agent from an arbitrary point in the search space to the goal state, and there would be no guarantee that the agent could reach the start point of the plan fragment. If a partial order planner was interrupted the result would be a series of discontinuous plan fragments with no guarantee that the agent could pass successfully between them. Any of these results would be unhelpful to an agent requiring a functional plan.

One possible way of producing a correct, functional plan from an anytime planner is to start with a complete plan, then iteratively modify it (by local search through plan space) to fit the current goal. This is the approach demonstrated by the Excalibur project in [10]. This provides a usable answer after an interruption because a complete plan is available at the end of each iteration (although the plan may not fully satisfy the agent's goals). Another approach to planning that can provide a complete plan at the end of each cycle of its operation is hierarchical task network (HTN) planning [14]. An HTN planner works by reducing an abstract representation of an action into less abstract parts, and then those parts into parts that are less abstract again. This continues until the plan is totally composed of primitive tasks. Primitive tasks are the atomic tasks that the agent can execute in the world (e.g. turn, walk, pick-up or more abstract actions like move-to or follow). After each reduction, a plan produced by an HTN planner can be viewed as a complete solution to the goal *at the current level(s) of abstraction*. This means that if the agent can execute the plan at whatever levels of abstraction it is described at, it can achieve its goal. It is this approach of interrupting an HTN planner that has been chosen for this research. The Universal Method Composition Planner (UMCP) [4] has been used as the basis of the implementation, and the anytime version of this has been named A(nytime)-UMCP.

3.2 Executing Plans Produced by an Anytime Planner

A necessary companion of an anytime algorithm is a method of interpreting the results of an interrupted process. In some cases this may not be necessary as the results may already be in an executable form. In this case, because our algorithm is based on an HTN planner, it is critical that an agent is able to execute a plan that is described at various levels of abstraction. How this can be done is subject that requires further research. One possible method that would allow the agent to generate the necessary behaviour is to equip the agent with a reactive implementation of any abstract method that could possibly appear in a plan. These implementations could be based on the reduction schema used by the planner to work out which abstract methods reduce into which less abstract ones. Unfortunately this approach appears superfluous when an agent already has a planner and a set of 'necessary' reactions. If the agent had a large number of abstract methods then this approach could lead to a storage and indexing problem. An additional problem is that an abstract method may have more than one possible reduction, and hence a single reactive implementation may not be applicable every time a particular abstract method needs to be executed. To overcome this, simple checks could be introduced to ensure the correct action is being used, or the most widely applicable action could be used (if one exists). A second possibility would be to employ a learning mechanism to generate a mapping between abstract actions and their executable forms based on how the agent achieves goals from complete plans. Case based reasoning would be particularly useful for this.

At this point it is important to note that it is crucial to minimise the amount of processing that an agent must do to interpret an abstract plan. This is necessary because when an agent has interrupted its anytime planner, it is assuming that it must execute the result immediately. If much additional processing is required, the agent might as well have left the planning process to continue, and obtained a better result. If the planner had some knowledge about the process of interpretation then it could use this to help formulate its estimates regarding plan utility and execution time.

Any method of interpreting abstract plans has an inherent problem that the more abstract the plan to be interpreted, the more likely it is that part of it will be executed differently than if the action had been fully reduced (this fact is used as the basis of a search heuristic presented in Section 4). An example of this which is likely to occur is overlooking a chance to order steps in such a way that prevents an agent from duplicating actions. For example, this could happen in some plans in the Blocks World. If two separate actions require a condition to be true (e.g. Clear(A)), then this action will be added to the plan twice. It is only at a later stage that this duplication is dealt with (this is done in UMCP by using a 'do nothing' action for one of them). If the planner is interrupted before the duplication has been removed, or even before the two actions requiring the condition have been produced from more abstract methods, then the agent may waste resources by attempting an action twice when it should only be required once. This is a problem from the agent's point of view (due to wasted resources), and is demonstrative of the relationship between processing time and plan quality.

4 An Anytime Heuristic

A key part of implementing an anytime algorithm for any application is developing a suitable quality measure for the partially complete solutions that the algorithm must work with. In this case, it is necessary to develop a quality measure for partial HTN plans. Because the plans are to be interpreted by a system that will be more likely to make mistakes if the plan is more abstract, the quality measure developed for the A-UMCP planner is based on a heuristic estimate of 'abstractness' (a less abstract plan is a better plan). To implement this, the heuristic used in the HSP planner [2] has been modified. Originally the heuristic was used to estimate the cost of achieving a goal in a state-based planning problem. For use in an anytime HTN planner it has been adapted to allow its application to the problem of estimating the cost of reducing an abstract method to a set of primitives. The rationale for using this originally state-based method is that usually the ultimate aim of reducing an abstract method is to introduce a particular atom into the current state[4]. The atomic goal of the abstract method, taken with the current state and the primitive HTN operators, form a state-based planning problem. If we apply the HSP heuristic to this problem, the resulting value represents the amount of action needed to achieve the atomic goal (i.e. how abstract that goal is). This value is directly related to the cost of reducing the associated abstract method into a set of primitives. The resulting heuristic is used to calculate the cost of individual plan steps (the abstract methods), with the total plan cost being based on the sum of its step costs. An empirical study confirms that estimating cost in this manner is valid. Figure 2 demonstrates the comparison between the heuristic cost and the actual cost (measured in CPU cycles) of reducing an abstract task into a collection of primitives for some examples from the Blocks World (although the demonstrated trend is domain independent). It shows that as the actual level of abstraction increases (represented by the CPU time taken to reduce an abstract method) the heuristic estimate of "abstractness" also increases. The fact that the heuristic sometimes attributes different heuristic values to methods that have similar real values (e.g. the adjacent bars with heuristic costs of 2 and 3) can be attributed to the simplifying assumption made during the calculation of the heuristic (the HSP heuristic behaves in the same way).

The results of the heuristic calculations are used in a weighted A* search [12](Chapter 3.2) to guide the A-UMCP planner to reduce the abstractness of the current solution as quickly as possible (as in these terms, less abstraction corresponds to a higher quality plan). After each search cycle, the best partial

[4] In UMCP this atom is used as the name for the actual abstract method.

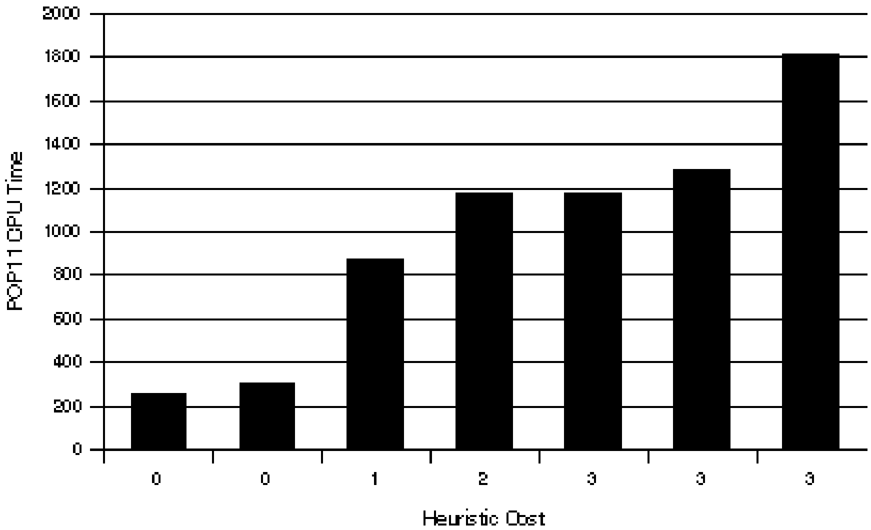

Fig. 2. Comparison Between Estimated Heuristic Cost and Actual Cost (measured in POP11 CPU Time)

solution (abstract plan) encountered so far is stored as the result to be returned if an interruption occurs. This makes the anytime planning process appear to the agent as monotonic with respect to time. The agent will never receive a plan of lower quality than the current result as processing time increases (even though the planner may produce them). Such monotonic behaviour is one of the required properties of an anytime algorithm. Requiring this monotonicity allows the agent to safely make the trade-off between processing time and plan quality, because it knows that the plan quality can never decrease.

5 Agent Construction

An agent has been implemented to test the efficacy of the planner in a game environment. The following sections will discuss the technology used to develop the agent, and the internal structure and features required by an agent to support an anytime planner.

5.1 Agent Technology

The implemented agent is a bot for the first person shooter Unreal Tournament. The interface to the game is provided by the Gamebots interface [7]. The interface provides sensory data from the game engine in the form of strings, and accepts action commands for bots in the same form. Within Unreal Tournament, the game being played is Capture The Flag. This was chosen because it has additional complexity compared with the other available Unreal Tournament game types. As such, it provides a more stringent test for the planner than the other game types would. One dimension of this complexity that is not catered for by the current implementation is the co-operative behaviour required to play as a member of a team. To successfully deal with this, both the agent and the planner would have to be modified to enable explicit co-operation.

5.2 Architecture and Internal Mechanisms

A rough overview of the architecture used for the anytime planning agent can be seen in Figure 3. From this, two interesting modules can be identified for further discussion; the interrupt management mechanism and the plan interpreter.

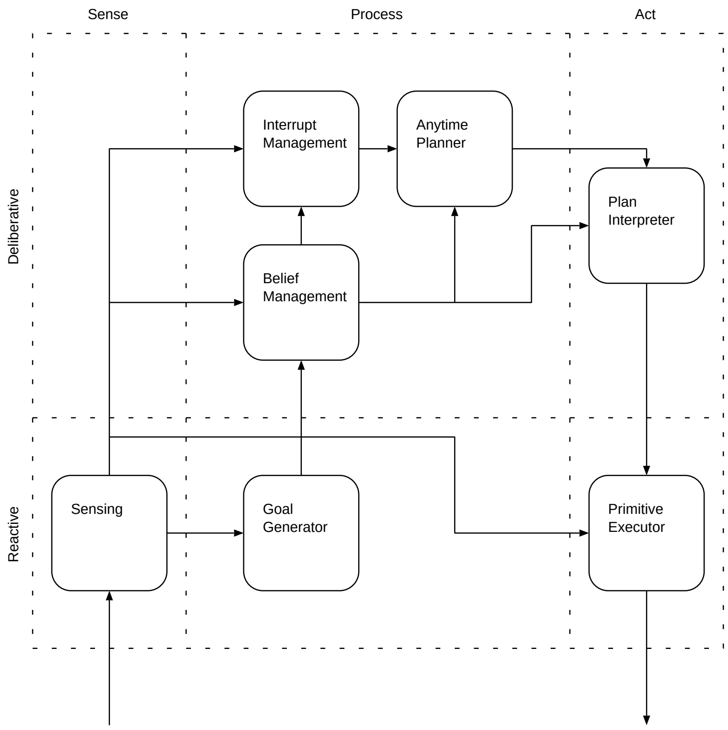

Fig. 3. Architecture of the anytime planning agent

The interrupt management mechanism is used by the agent to redirect its resources under certain conditions. Such conditions occur when the agent is planning for, or executing a plan for, a goal that is no longer desirable. To facilitate arbitration between possible courses of action (i.e. deciding what is currently "desirable"), all goals are assigned a measure of importance [1](Section 3.2.2.1). For each goal this represents how important it is for the agent to achieve it relative to all other goals. The importance measures can be reassigned during a game if the agent's priorities change[5].

In more detail the conditions that trigger an interrupt are;

- Criticality based interrupts:
  - An interrupt is triggered if the agent is planning for a goal, and the goal generator proposes a new goal that is more critical than the current planning goal.
  - An interrupt is triggered if the agent is executing a plan for a goal, and the goal generator proposes a new goal that is more critical than the current goal.
- Goal specific interrupts:
  - Each goal proposed can have specific interrupt conditions (e.g. the goal to retrieve the agent's team's flag is interrupted if the flag is retrieved by another agent). An interrupt is triggered if these conditions occur.
  - Each goal proposed can have specific time-based interrupt conditions (e.g. if the time required to plan and execute an action to intercept an agent stealing the agent's flag is longer than the predicted time that agent takes to score a point with the flag). An interrupt is triggered if these time-based conditions occur.

Depending on the cause of the interrupt and the agent's current state, an interrupt can have one of three possible effects. Firstly, the current action (planning or execution) can simply be halted, causing all information on the current task to be discarded. Secondly, the current action can be suspended, whilst another goal is pursued. This type of interrupt causes the storage of information necessary for an agent to continue planning or executing a plan in the future. The final type of interrupt can only happen during the planning process, and causes planning to be halted and the resulting (potentially abstract) plan to be executed.

This leads on to the second interesting component of the architecture pictured in Figure 3, the plan interpreter (a necessary part of the anytime planner as described previously). Currently this module is implemented in the simplistic first way described in Section 3.2. The input to the component is an arbitrarily abstract task network produced by the A-UMCP planner. The output is an interpretation of this network in terms of primitives that are executable by the agent. The interpretation is done by first using all the information available (task orderings and constraints) to generate an ordering for the plan steps. These steps are then translated into primitives using information derived from the A-UMCP reduction schema. Where multiple reductions are possible, the most widely applicable option is chosen (this usually results in redundant steps in the final plan).

[5] Altering the importance assignments can be used to produce agents with different playing styles.

5.3 Necessary Knowledge

From the previous descriptions of agent components, it is evident that the agent must maintain knowledge about certain aspects of its internal processing and the external world in order to support anytime planning. Types of knowledge required include:

– **Criticality Measures:** The agent must know, or know a way of calculating, how critical each possible goal is to it.
– **Interrupt Conditions:** The agent must know general, or goal-specific conditions which can trigger planning and execution interrupts.
– **Timing Knowledge:** The agent must be aware of how long it may take to plan for and execute particular goals. It may also be necessary to be aware of how long it takes for its opponents or teammates to perform certain observable tasks, and how long is left in the current game.
– **Performance Profile:** A performance profile represents knowledge about the quality of the plans produced by the planner within particular time periods [3]. This information about the potential quality of plans will prove useful when making decisions about when to carry out interrupts.

Most agents that are based around an anytime planner will require some of these types of knowledge in order to make optimal use of their anytime abilities.

6 Empirical Observations

The A-UMCP planner has been completed, and has been integrated into the agent described above. Preliminary results look promising, with the implemented agent demonstrating behaviours required to succeed in a dynamic and real-time environment. A key behaviour available to the agent is using the anytime nature of the planner to take advantage of variable lengths of time available for planning. When doing this, it is apparent that longer processing time allows the agent to create better, more detailed plans (not just plans that never decrease in quality, as was stated as a requirement for an anytime algorithm). The agent can also reuse previously constructed plans (both solutions and abstract results), and can continue planning from points in previously suspended planning processes. These abilities provide the agent with the flexibility necessary to deal with the fast-paced, dynamic world of a computer game. The implemented method of interpreting plans provides a basic method of understanding abstract plans, but due to its static nature occasionally produces bad primitive interpretations when the current state varies from the agent's default assumptions. This may not be entirely the interpretation method's fault as such failures often come after early interruptions to the planning process, leaving the interpretation method with a great deal of guesswork to do. This guesswork could be assisted by learning in a future implementation.

One criticism that could currently be levelled at the implementation is that it is not sufficiently complex to truly test the planner. The Unreal Tournament domain was chosen because it provided an easily accessible real-time game as a testbed. Unfortunately, because the requirements for success in Unreal Tournament favour physical qualities (e.g. reaction times and aiming ability) over cerebral ones (e.g. producing plans that are better than your opponents), the influence of the planner is considerably diminished. In Unreal Tournament a successful plan usually means doing something which isn't stupid, whilst a failed plan will result in behaviour that is stupid. A better test for an anytime planning agent would be a games domain where the utility gained from planning varied along a more gradual scale, rather than in such a discrete dichotomy. Such variation would provide more meaningful results for the evaluation of the planner, as variations in planning time would result in a wider range of possible plan qualities, and therefore a wider (and more interesting?) range of agent behaviour.

7 The Benefits Provided by using an Anytime Planner

7.1 The Benefits to a Game Developer

Equipping a game agent with an anytime planner will provide a number of advantages from both the agent's and the agent developer's point of view. From the developer's point of view, the main advantages are evident in the way the AI code can be processed. Having a process that can be safely interrupted (and resumed) allows planning to be easily scheduled or spread across the available processing time. This could be achieved, for example, through the use of an AI process manager [18]. Secondly, by being able to specify a flexible bound on planning time (by forcing an interruption to occur after a certain amount of time), it becomes less complex to implement dynamic level-of-detail AI. For distant or relatively insignificant agents, an early interruption of an anytime planner will result in an abstract plan that will (depending on how the agent interprets it) guide the agent to achieve the goal in a basic (possibly clumsy and inefficient) way, after minimal processing. For agents that are significant to the player's experience (either due to being closer or because their actions will have a greater impact on the course of the game) an anytime planner can be run for a lot longer, resulting in more detailed and hence more efficient (or interesting, believable, faster etc.) behaviour as the plan is executed. Another advantage gained by the developer using an anytime planner is an agent that displays 'believable' weaknesses. Humans do not always form correct optimal plans when under pressure (when playing computer games or at other times), so competing against an agent that did would enforce the feeling of interacting with something artificial (although a 'perfect' style of play would be useful for certain game characters e.g. robots or super-beings). An agent executing plans returned from an interrupted anytime planner would almost certainly not display optimal behaviour, and could possibly fail to achieve its goals altogether. Based on this, an upper bound on planning time could be used to characterize certain styles of gameplay and game-agent behaviour. From considered and cautious (a high upper bound, therefore more planning time) to gung-ho and carefree (a low upper bound, therefore less planning time).

7.2 The Benefits to a Game Agent

From an agent's point of view, the benefits gained by using an anytime planner (as opposed to a traditional planner) can be grouped into two categories; optimal use of time, and immediate reactions to environmental change. The time-based benefits stem from the fact that there is always a limit on the amount of processing an agent can perform. This limit is imposed by either software (in the case of game agents with a scheduled amount of processing time) or (and ultimately in all cases) hardware (in the case of robots or human agents). The better use an agent can make of the processing time available to it, the better it can perform. An anytime planner helps an agent do this because of its interruptibility. The simplest of cases may see an agent deciding that it no longer needs to plan for whatever goal it was previously pursuing, and halting the planning process (a function available in principle to all planners). An example of a more advanced use of an anytime planner would be the agent monitoring the progress of the planner (perhaps via a performance profile [3]) and then examining the utility it expects to receive from achieving the goal. If the utility of achieving the goal is decreasing with time (e.g. in a game of capture the flag, the utility of scoring points may decrease with time if a team is considerably behind the opposition) the agent may judge that the continued increase in the plan's quality does not warrant the continued decrease in the goal's utility. If this is the case then it can interrupt the planner and execute the resulting plan. This behaviour would not only save processing time, but it would also prevent any unnecessary loss in goal utility.

In a dynamic environment the agent's (short to mid-term) goals may be quite fluid, and waiting for a traditional planner to return a plan may result in opportunities passing the agent by or current planning goals becoming undesirable. If an agent can anticipate how long it will be until such environmental changes occur, and has some idea about how long it will take to execute a plan, it can interrupt the planner in advance of the changes and take action. This is really a more severe case of maximizing the use of processing time, where instead of a gradual decrease of a goal's utility, there is a sudden one when the environment changes.

8 Conclusion and Future Work

The conclusion to draw from this work is that an anytime planner will be a useful tool for both an agent in a dynamic world, and games developers implementing such agents. It is an approach to planning that allows judgements to be made about how processing time can be used effectively. The fact that the algorithm is interruptible and a cost measure is available for partial solutions means that the agent can interact with the planner, monitoring its progress and can achieve trade-offs between time spent planning and the expected utility of executing the resulting plan. Future research needs to investigate how abstract and partially complete plans from a hierarchical task network planner can be cheaply and accurately interpreted by an agent. An approach suggested in this paper (rule-based interpretation) may be practical but is inflexible and fallible in situations that vary significantly from the norm.

9 Acknowledgements

This research is funded by sponsorship from Sony Computer Entertainment Europe. Also, this research has been supported by Aaron Sloman in the form of advice, critical assessment and supervision.

References

1. L. P. Beaudoin. *Goal Processing In Autonomous Agents.* PhD thesis, School of Computer Science, The University of Birmingham, 1994.
2. B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence: Special Issue on Heuristic Search*, 129:5–33, 2001.
3. T. Dean and M. Boddy. An analysis of time-dependant planning. In *Proceedings of The Seventh National Conference on Artificial Intelligence*, pages 49–54, 1988.
4. K. Erol. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation.* PhD thesis, Department of Computer Science, The University of Maryland, 1995.
5. N. Hawes. Real-time goal-orientated behaviour for computer game agents. In *Game-On 2000, 1st International Conference on Intelligent Games and Simulation*, pages 71–75, 2000.
6. N. Hawes. Anytime planning for agent behaviour. In *Proceedings of the Twelth Workshop of the UK Planning and Scheduling Special Interest Group*, pages 157–166, 2001.
7. G. A. Kaminka, M. M. Veloso, S. Schaffer, C. Sollitto, R. Adobbati, A. N. Marshall, A. Scholer, and S. Tejada. Gamebots: A flexible test bed for multiagent team research. *Communications of the ACM*, 45(1):43–45, 2002.
8. J. E. Laird. It knows what you're going to do: Adding anticipation to a quakebot. In *Papers from the 2000 AAAI Spring Symposium on Artificial Intelligence and Computer Games*, pages 41–50, 2000.
9. J. E. Laird and J. C. Duchi. Creating human-like synthetic characters with multiple skill levels: A case study using the soar quakebot. In *Papers from the 2001 AAAI Spring Symposium on Artificial Intelligence and Computer Games*, pages 54–58, 2001.
10. A. Nareyek. Using global constraints for local search. *Constraint Programming and Large Scale Discrete Optimization*, 57:9–28, 2001.
11. A. Nareyek. Intelligent agents for computer games. In *Computers and Games, Second International Conference, CG 2000*, pages 414–422, 2002.
12. J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, 1984.
13. M. E. Pollack and J. F. Horty. There's more to life than making plans. *AI Magazine*, 20(4):71–83, 1999.

14. E. D. Sacerdoti. The nonlinear nature of plans. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 162–170. Morgan Kaufmann, 1990.
15. M. Scheutz and B. Logan. Affective vs. deliberative agent control. In *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing*, pages 39–48, 2001.
16. A. Sloman. Varieties of affect and the cogaff architecture schema. In *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing*, pages 1–10, 2001.
17. M. van Lent, J. Laird, J. Buckland, J. Hartford, S. Houchard, K. Steinkraus, and R. Tedrake. Intelligent agents in computer games. In *Proceedings of The National Conference on Artificial Intelligence*, pages 929–930, 1999.
18. I. Wright and J. Marshall. Egocentric ai processing for computer entertainment: A real-time process manager for games. In *Game-On 2000, 1st International Conference on Intelligent Games and Simulation*, pages 42–46, 2000.
19. S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.