Architectures by Design: The Iterative Development of an Integrated Intelligent Agent

Nick Hawes

Abstract In this paper we demonstrate how a design-based methodology can be used to iteratively produce designs for an information-processing architecture that integrates various intelligent capabilities. This methodology allows us to explain system performance in terms of changes to an existing architecture design, with the explanations being supported by performance data from an implementation of the design. We present an instance of this design methodology applied to the development of an architecture that integrates anytime deliberative capabilities with reactive behaviours and goal management. Iterations of the design are implemented and evaluated in the computer game *Unreal Tournament*.

1 Introduction

This paper presents an example of the use of a *design-based methodology* applied to the design of an intelligent agent. This example demonstrates how the empirical evaluation of an implementation of an intelligent system can help refine a set of requirements for the system and the derived designs. This development approach allows agent designers to account for and explain the eventual behaviour of systems by reference to these requirements and designs, and the informed changes made to them based on experimental results.

Nick Hawes
Intelligent Robotics Lab, School of Computer Science, University of Birmingham, UK e-mail: n.a.hawes@cs.bham.ac.uk

2 Design-Based Methodology

It is an agent's information-processing architecture that is key to its ability to integrate various intelligent capabilities. It is therefore crucial that such architectures can be evaluated in a way that is objective and that allows comparisons with similar architectures. In the past, information-processing architectures have been developed in various ways. Some have been the modelled directly on biological systems, some have been designed to explicitly produce certain phenomena, and some have emerged directly from the need to place certain components in a single system.

When examining this previous work, objective evaluation and comparison of architectures is not always possible because researchers do not regularly reference design-neutral requirements for their architectures, or the trade-offs inherent in making design decisions. It is our belief that these parts of the design process should be made explicit in order to contribute knowledge to the science of constructing integrated cognitive systems. To support this, we follow a design-based methodology when developing information-processing architectures. This is an approach to explaining how systems (biological, non-biological, existing, hypothetical etc.) work, based on how they are designed. The term was introduced in [16] where the approach was used to analyse familiar concepts, such as emotion. This design-based methodology can be considered as directly related to Dennett's design stance [3].

Although most AI researchers would argue that they work from a design stance, as they produce abstract representations of their systems, many do not consider the *design space* in which their single design is located. It is only by comparing designs to other designs that occupy neighbouring design space that we can develop an understanding of particular design features (e.g. when comparing the trade-offs between two designs that differ on one feature) [17].

The methodology we follow is based on the following steps:

1. Determine the requirements of the system you are trying to design. These should be based on the required functionality of the final system. Requirements determine the *niche* or role that the proposed design is intended to occupy (for discussions of the relationship between design space and niche space see [17]).
2. Produce designs which satisfy these requirements. When developing agents, the proposed designs will be agent architectures (which could include detailed specifications of representations, processing styles, information exchange strategies etc.).
3. Implement the design or designs. Whilst the main purpose of implementation is to produce a working system, this step may also lead to a better understanding of how parts of the design interact, and provide a deeper understanding of the requirements of the design.
4. Evaluate how well the implementation reflects the design, and to what degree this design meets the requirements specified in Step 1. This evaluation can done by experimental or analytical means.

5. Examine the design space surrounding the implemented design to study how changes to it could provide additional functionality that may or may not allow it to better meet the specified requirements.

Although the steps are discretised here to allow clearer explanation, they can overlap or be performed in parallel. When this is the case, later steps can feed back into earlier steps, modifying the overall direction of the research.

The following sections summarise four iterations of the above methodology applied to the task of developing an intelligent agent for a computer game. Due to space constraints we mostly focus on how the evaluation of an implementation (Step 4) supports the exploration of design space (Step 5).

3 Background & Assumptions

The initial motivation behind the work presented here was to design an intelligent agent for use in action-orientated computer games. To challenge the state-of-the-art (both in terms of the games industry and AI research on games), we decided to investigate methods of integrating deliberative planning with reactive behaviours in a hybrid agent architecture. We decided to take this approach because we view the behaviour generated by an agent following a deliberative plan as more goal-orientated than the behaviour generated by an agent just reacting to its world. We also assume that agent behaviour that is more goal-oriented appears more intelligent to observers (e.g. humans playing a game against the agent), and therefore prefer agent designs which provide better support for this. Although this is quite a limited characterisation of the space of possible approaches to generating intelligent behaviour, we use it as one of the key requirements for the subsequent design work.

By choosing to design an agent for a game world we already face a set of general requirements for the system: it must be able to cope with a world that is *interactive*, *real-time* and *complex*. This means that the agent must deal with a world that changes through the actions of both it and others, at a rate that is out of its control, and the choice of potential actions available is not trivial. Reasoning about future actions in such a world challenges a number of assumptions that have been made by many planning approaches in the past [15]. Rather than build mechanisms to tackle such problems into a planning algorithm, we view it as the task of the whole agent architecture to support intelligent behaviour in such worlds in an integrated fashion. We have chosen to design the architecture from scratch (rather than adopt an existing solution) to ensure that the planner's functionality is fully integrated with the functionality of other components within the architecture.

4 Scenario

The design and implementation of the agent is based around a scenario called *Capture The Flag* (CTF), which is a type of game found in many multi-player computer games. It is a game played between two or more opposing teams, where each team is made up of one or more players. The aim of the game is to score a set number of points, or to have the most points when the game ends (after a fixed amount of time). Each team has a base, on which a flag rests. To score a point, a player must take the opposing team's flag from their base and then get it back to their own base. This action will score a point only if the team's own flag is at their base when the player returns with the opposition's flag.

Agents in our CTF scenario can only perform a small number of actions. These are limited to running, jumping, picking up and putting down objects, and using weapons (this is typical for many computer games). In the CTF scenario, typical behaviours for agents include guarding their team's flag, attempting to steal the opposition's flag, trying to prevent the opposition from scoring, and assisting a team mate that has captured the opposition's flag.

One of the principal motivations for the selection of this scenario was the desire to evaluate our designs in a real game environment. Only very few games provide the kinds of interfaces necessary to support the addition of an AI character written from scratch, and most of these are first-person shooters. The CTF game type is prevalent in these types of games, whilst providing a level of complexity above that of the typical "kill one another until the time runs out" scenarios found in many other games types.

Even with these advantages, there are reasons why the CTF domain is flawed as a testing ground for AI techniques. First, although it is more complex than many other game types (and hence more challenging for the applied techniques), it falls short of the level of complexity represented by many existing AI testbeds. As such, it provides a less than adequate test of the agent's problem solving abilities. A second reason the domain is flawed is that success in CTF is usually determined by 'physical' skills such as aiming ability and reaction times, rather than cognitive abilities (cf. [2]). Because of this, a purely reactive, rather than hybrid or deliberative, agent would generally perform better at most aspects of the CTF scenario.

The agent implementations discussed in the following sections were produced using Pop11 with the SimAgent toolkit [19]. They play Capture The Flag in the game *Unreal Tournament* [4] via the Gamebots interface [8].

5 Iteration One: The Basic Agent

The architecture for our initial attempt at designing an agent for the CTF scenario can be seen in Figure 1. It is based around the CogAff architecture schema [18], with the thick lines representing segments of the schema that are being utilised. The overall design was informed by previous work within our research group, and by the constraints of the scenario. The following paragraphs discuss the design and implementation of the components within the architecture. This is followed by an evaluation of the implementation. For more detailed descriptions of all the components see [7].

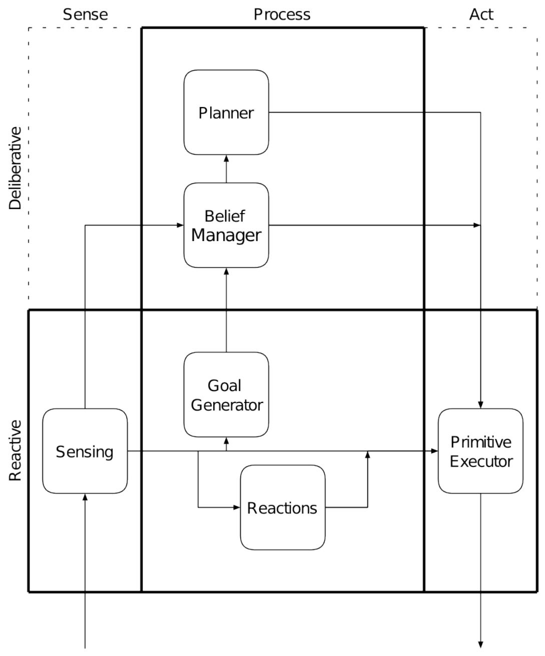

Fig. 1 Architecture Design For Iteration 1.

Sensing: The sensing component translates events from the external world (i.e. the Gamebots interface to *Unreal Tournament*) into representations that the agent can process.
Execution: The primitive executor takes action commands (e.g. move to the opponent's base, turn to face opponent etc.) from other components and produces the appropriate actions in the world. This component is implemented as a set of Teleo-Reactive Programs [13]. The primitive executor is essential in the architecture as it separates the mostly continuous, regularly changing, detailed knowledge about the external world (which it operates on to perform actions), from the more discrete, static knowledge useful for reasoning about it (which it receives as commands). Without this component the agent's deliberative components would have to deal with "real-world" type knowledge (such as coordinates) and also run the risk that this information may become out of date as it processes it.
Reactions: The reaction component monitors incoming sense data and quickly generates appropriate action responses to the current situation (e.g. it attempts to move

the agent to safety when it is being attacked). These actions are send to the primitive executor. The reactions are implemented as a set of condition-action rules.
Goal Generation: The goal generator also monitors incoming sense data. Its task is to generate new goals based on this information. It can generate six different goals, each of which has a fixed importance value. The goal-generator has no access to the agent's current belief state, but is entirely driven by changes in the external world. This enables a level of reactivity that ensures goals are generated quickly when relevant events occur. This simple approach to goal generation is feasible in limited game scenarios, but may not scale to more complex domains.
Beliefs: The belief manager stores and manages the agent's internal state, including beliefs about the world. It also manages the selection of goals and handles the subsequent planning and plan-execution processes. The belief manager selects goals based on their importance, with the selection of a new goal only being allowed when neither a planning or plan-execution process is active.
Planning: The planner constructs action plans to achieve the goal held by the belief manager. It is implemented as a hierarchical task network (HTN) planner based on UMCP [5].

5.1 Evaluation

In this agent design, behaviour can be produced either by reactive mechanisms responding to the immediate situation (e.g. attempting to get the agent out of harm's way), or by the execution of an action plan produced by the planner (e.g. attempting to capture the opponent's flag). As stated previously, we consider the latter of these types of behaviour as inherently more goal-orientated, and therefore prefer our agent to spend more time acting in this way. This requirement has led us to select particular metrics by which we can measure the performance of the implementations of our designs. The metrics we use are:

- **Planning time**: The amount of time an agent spends planning during a single game.
- **Wasted planning time**: The amount of time an agent spends planning for a goal after the goal is no longer valid.
- **Execution time**: The amount of time an agent spends executing plans during a single game.
- **Wasted execution time**: The amount of time an agent spends executing plans to achieve a goal after the goal is no longer valid.
- **Percentage of goals selected**: The percentage of proposed goals that an agent selects and starts planning for.

Wasted time is bad because it is time that should be used for a more important process, and as such represents a measure of latency between the events in the world and the agent's behaviour. The notion of "waste" is judged by examining the importance of the current planning or execution goal, and comparing it to the importance

Iteration	**1**	**2**	**3**	**4**
Avg. Planning Time	138 secs	209 secs	186 secs	157 secs
Avg. Wasted Planning Time	37 secs	27 secs	0 secs	0 secs
Avg. Execution Time	153 secs	93 secs	105 secs	135 secs
Avg. Wasted Execution Time	95 secs	0 secs	0 secs	0 secs
Avg. Percentage of Goals Selected	24%	22%	26%	32%

Table 1 Results from the four iterations of the design process.

of goals that are being generated in the meantime. If a more important goal is generated, then all subsequent effort is considered wasted, because it could be spent pursuing the new goal. Goal *achievement* is not used as a metric because it proved very hard to reliably attribute the achievement of a particular goal to the actions of a single agent in our scenario.

Four instantiations of the previously described architecture were used to create two teams of two agents to compete across ten CTF games in *Unreal Tournament*. The values for the evaluation metrics were averaged across these forty separate agent instantiations, and are presented in column 1 of Table 1.

From examining these results, along with the basic agent design, it can be seen that the fact that the agent continues to execute plans once its goals are no longer valid appears to be the biggest source of wasted time. As this is the first iteration of the design process, we have nothing to compare the average percentage of goals selected against. Without a suitable comparison, it is hard to know how this value reflects the performance of the agent. The need for comparison is not as important when analysing the amount of time wasted though, as any amount of wasted time reflects a drop in the agent's real-time performance.

6 Iteration Two: Execution Interrupts

The previous analysis led to the identification of periods during which the agent is executing a plan when it should really be pursuing a different goal. To attempt to eliminate these periods, we must alter the previously presented agent design. Rather than discuss the design of the whole architecture, we will only focus on the changes made to the previous design. The new design can be seen in Figure 2.

It is the belief manager that holds the key to reducing the amount of time wasted during plan execution. In the basic design, when the goal generator proposes a goal that is more important than the one currently being pursued, the belief manager simply stores this goal. Then, when the agent finishes what it is currently doing (either planning or executing) the goal is selected. To reduce the amount of execution time wasted by the agent, we need to redesign the belief manager to immediately halt the execution of a plan if it is discovered that this plan's goal is no longer the agent's current goal (i.e. a more important goal has been proposed). But, rather than alter the behaviour of the belief manager too drastically, we can make a small alteration

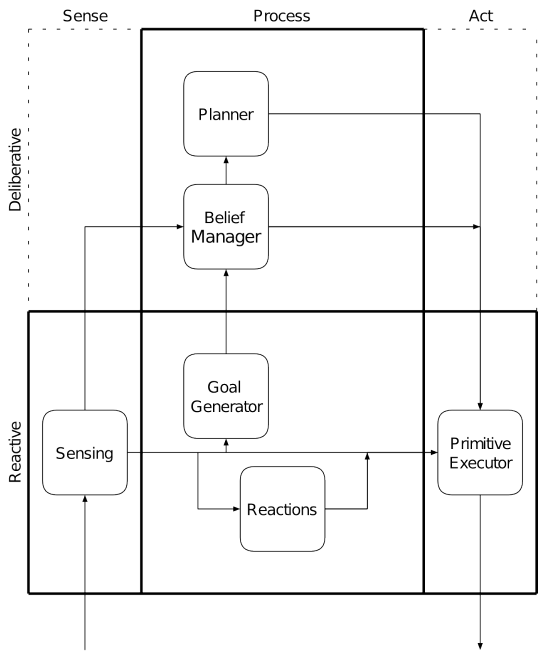

Fig. 2 Architecture Designs For Iterations 2 & 3 (the dotted line indicates the additional connection for Iteration 3).

to it, and add a new component to the architecture to handle the rest of the necessary behaviour. The small alteration to be made to the belief manager is to allow it to immediately select a proposed goal to be the current goal if the proposed goal is more important than the current goal. The new component to be added to the agent architecture is the *interrupt manager*.

The interrupt manager's role within the agent architecture is to monitor the goal held by the belief manager and ensure that the primitive executor is not executing a plan for any other goal. It takes input from the belief manager about the current goal, and from the primitive executor about the current action goal. If the goal supplied by the belief manager differs from the one supplied by the primitive executor (which would happen if a more important goal has been proposed), the interrupt manager generates an interrupt event. This should cause plan execution to stop and allows the belief manager to start a planning process for the new goal.

Given the ability to interrupt plan execution, it is desirable to see what other conditions might warrant an interrupt being generated. Game worlds can change quickly and unexpectedly, and this can lead to situations where the agent is executing a plan for a goal that is no longer valid, but no new goal of higher importance has been proposed (e.g. when an agent is attempting to score a point but another agent has possession of the flag). If this happens then the agent should adopt another goal, but if this new goal is of a lower importance, then it won't generate an interrupt based on importance. To overcome this we must give the interrupt manager the ability to generate interrupts when a plan is being executed, but the associated goal is no longer valid. To do this, not only must the interrupt manager cause execution to be halted, but it must cause the belief manager to deselect the current goal. This is necessary because a valid new goal with lower importance cannot be selected whilst the belief manager holds a goal of higher importance. The interrupt manager plays a similar role in the agent architecture as the environment monitors do in the CPEF continual planning system [11].

The interrupt manager is implemented as a set of reactive rules for comparing the current execution goal to the goal held by the primitive executor, and for evaluating whether the current execution goal is valid given the agent's beliefs about the world. This latter process could be implemented deliberatively (e.g. by comparing the preconditions for actions in the current plan against the state of the world), but a reactive implementation was chosen for speed and efficiency. This is feasible in this case because the number of beliefs that can affect a plan's validity are quite limited.

6.1 Evaluation

Experimental results for this iteration of the design process can be seen in column 2 of Table 1. They show that the agent no longer wastes any time executing plans when the associated goals are no longer appropriate. As a result of this, the overall amount of time the agent spends executing plans has decreased, and consequently it has more time to spend planning. Although the amount of time the agent spends planning has increased, the results demonstrate a decrease in the amount of planning time that is wasted by the agent. It is likely that this change isn't connected to the alterations made to the agent's design, because none of the interfaces to the planner were modified. Along with this unexpected positive change, the results show an unexpected negative change. The percentage of goals selected has dropped below its value in the first iteration. With less time spent executing plans, it should be expected that this value would rise as the agent is able to complete more plan-act cycles and therefore present the belief manager with more opportunities to select goals.

Although the design proposed in this second iteration improves upon the previous design in most respects, there still appears to be a significant amount of time being wasted by the agent during planning processes. This time wasting behaviour is not strictly due to the speed of the planner (although a faster planner would waste less time), but is due to the fact that unlike the execution process, the planner is not interruptible in this design. The next iteration of the design process will address this.

7 Iteration Three: Simple Planner Interrupts

In this iteration of the design process, the changes made previously to allow execution processes to be interrupted are extended to apply to planning processes too. Only two changes are made to the design and implementation of the agent in this iteration. The interrupt manager in connected to the planner, and the implementation of the planner is altered to allow interrupts to occur. This can be seen in Figure 2.

7.1 Evaluation

The results in column 3 of Table 1 show that altering the design to allow planning to be interrupted as well has had positive effects. The agent now wastes no planning or execution time, and the average amount of time an agent spends planning has dropped by a similar amount to the amount of time it was previously wasting during the planning process. The average amount of time the agent spends executing has also risen due to the reduction in planning time. The complete eradication of wasted time during planning and execution has led to an increase in the percentage of goals that the agent can select, as the new design provides a greater number of opportunities for the agent to adopt a new goal.

8 Iteration Four: Anytime Planning

Now that the amount of time wasted by the agent has been reduced to zero, we can look for other, less obvious flaws in the agent design. The results from the third iteration of the design process show that the agent still spends a great deal of time planning. Although none of this time is "wasted" using our (admittedly limited) metrics, it is still possible to argue that the agent is wasting time during both planning and execution. If a planning or execution process is interrupted and halted, then the time spent on that process is effectively wasted as the agent has not gained any benefit from the process (except possible inadvertent gains), and the partial results of the process are discarded.

If the agent could anticipate when interrupts to a process are likely to occur, and make use of the results of the incomplete process before this, then the deleterious effect of interrupts could be overcome. Although this is an interesting design choice, it is not possible to apply this to plan execution processes. Despite the fact that execution processes can be interrupted at any time, at no point is there anything concrete that can be retrieved from this process to use at a later date. Any gains made from partially enacted execution processes will be the results of actions taken during execution (e.g. gaining a favourable world position or observing an interesting event), not as the result of anything returned or stored by the execution process.

For the agent's planning process, this idea of using partial results is much more viable. Any processing effort expended during planning a process results in the space of possible plans being pruned of invalid plans, so that the planning agent gains a better understanding of what a likely solution will be. If the planning process can be halted in sufficient time before an anticipated interruption, then this partial result can be used to guide subsequent execution. This approach will allow the agent to achieve its selected goals before they become invalid (i.e. before an interruption should occur) and reduce the amount of time spent on planning processes that are eventually discarded. This fourth iteration of the design process will investigate the design changes necessary to support this behaviour. The resulting design can be seen in Figure 3.

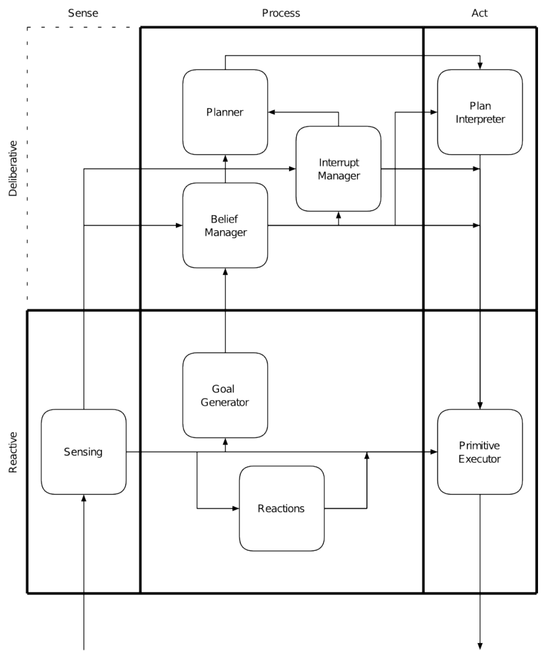

Fig. 3 Architecture Design For Iteration 4.

To support the use of partial results, the planner must be redesigned as an *anytime planner*. For this we use the approach described [7]. The result is an anytime extension of the UMCP planner which produces plans that increase in quality monotonically with respect to time. This enables rational judgements to be made about interrupt times.

The anytime planner produces partially complete HTN plans in the UMCP formalism. The plans are complete in that they contain specifications of actions for the entire plan, but they are incomplete because some of the steps are represented at an abstract level (i.e. they are not suitable as input for the primitive executor). To enable the agent to act out these plans, they must first be translated into primitives. This is the task of the *plan interpreter*. The plan interpreter must be implemented in a way that ensures that the minimum of time is wasted after an interrupt has occurred. Therefore we implemented it as a cut-down reactive planner with fixed plans for each possible abstract action. Given a plan containing abstract actions, it quickly produces a plan that is completely primitive, but that is not always correct. This is typical of the trade-offs that must be made when using an anytime algorithm.

In this iteration, the interrupt manager must be able to spot situations in which, in order for the agent to avoid wasting planning time and to maximise its ability to achieve goals, it is necessary for the agent to interrupt its planner and use the partial results to guide execution. Rather than being based on importance-based comparisons, this new type of interrupt is heavily reliant on time. Typically the interrupt manager must be aware of the amount of time left before the current goal becomes invalid and the amount of time it usually takes for the agent to execute a plan to achieve the current goal. The difference between these values represents the time available for the agent to plan in. Because of the dynamism of the domain, it is difficult for the interrupt manager to make consistently accurate decisions about when to generate an interrupt. To address this it could use probabilistic methods (although this may not guarantee success) or assume some value of inaccuracy when examining previously determined time values (e.g. five seconds either way, but this may cause the agent to waste time). In the implementation, the interrupt manager takes the latter approach when making decisions about when to interrupt the planner.

8.1 Evaluation

The results in column 4 of Table 1 show that by altering the agent design to allow preemptive interrupts of an anytime planner we have reduced the amount of time the agent spends planning. This has resulted in an almost identical increase in the amount of time the agent spends executing plans. This is important because it means that the agent is closer to satisfying the original requirement of spending as much time as possible acting out intelligent, goal-directed behaviour, rather than relying on its reactions. The new design has also resulted in an increase in the percentage of suggested goals that the agent selects. This is because the design changes effectively speed up the time the agent takes to run through a plan-act cycle, giving it more opportunities to select proposed goals.

9 Related Work

The agent architecture produced as a result of our design-based methodology is similar to many other hybrid architectures in the literature. Many architectures for intelligent agents combine deliberation with reactive plan execution, e.g. [20, 1], but do not take the additional step of using an anytime planner. The SOMASS system [10] employs HTN planning and reactive action interpretation as our final agent design does, but also does not have the additional mechanisms for anytime behaviour. The Excalibur project [12] employs an anytime planner for controlling computer game agents, but does not integrate it into an architecture containing the reactive control mechanisms necessary to cope with dynamic game worlds. Many other agents have been developed for game worlds (e.g. [9, 6]), but they tend to be variations on purely reactive systems.

In terms of the metholdology we employed to create our architecture, there are not many similar processes documented in the AI literature. The most prominent alternative approach is Bryson's Behaviour-Oriented Design (BOD), which has also been applied to the development of an Unreal Tournament bot [14]. Although BOD is also an interative methodology targeted at system behaviour, it is intended only for behaviour-based systems, and as such is much more limited in scope than the design-based methodology followed here. That said, its limited scope allows it to specify a much more precise (and therefore informative) set of steps which any designer much work though.

10 Conclusion

In this paper we demonstrated how a design-based methodology can be used to produce new iterations of an agent design based on the analysis of implementations of preceding designs. This method provides a framework for demonstrating how changes in a design affect how the design satisfies a particular set of requirements. In this instance we used a small set of empirical metrics to evaluate the designs with regard to the original requirements. The ultimate result of this is that the design process produces agent designs that are a demonstrably better fit for their intended niche than other designs that are nearby in design space. Empirical evaluation demonstrates this, with instantiations of the agents produced in the fourth iteration of the design process winning 70% of 3 vs. 3 games, and 90% of 2 vs. 2 games against instantiations of agents from the first iteration [7].

References

1. Bonasso, R.P., Firby, R.J., Gat, E., Kortenkamp, D., Miller, D.P., Slack, M.G.: Experiences with an architecture for intelligent, reactive agents. J. Exp. Theor. Artif. Intell. **9**(2-3), 237–256 (1997)
2. Cavazza, M.: Merging planning and path planning: On agent's behaviours in situated virtual worlds. In: Proceedings of the AISB'00 Symposium on AI Planning and Intelligent Agents, pp. 17–24. Birmingham, UK (2000)
3. Dennett, D.C.: Brainstorms: Philosophical Essays on Mind and Psychology. MIT Press, Cambridge, MA (1978)
4. Epic Mega Games: Unreal Tournament. GT Interactive (1999)

5. Erol, K.: Hierarchical task network planning: Formalization, analysis, and implementation. Ph.D. thesis, Department of Computer Science, The University of Maryland (1995)
6. Gordon, E., Logan, B.: Game over: You have been beaten by a GRUE. In: D. Fu, S. Henke, J. Orkin (eds.) Challenges in Game Artificial Intelligence: Papers from the 2004 AAAI Workshop, pp. 16–21. AAAI Press (2004)
7. Hawes, N.: Anytime deliberation for computer game agents. Ph.D. thesis, School of Computer Science, University of Birmingham (2004)
8. Kaminka, G.A., Veloso, M.M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S.: Gamebots: A flexible test bed for multiagent team research. Communications of the ACM **45**(1), 43–45 (2002)
9. Laird, J.E., Duchi, J.C.: 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, pp. 54–58 (2001)
10. Malcolm, C.: A hybrid behavioural/knowledge-based approach to robotic assembly. In: Evolutionary Robotics: From Intelligent Robots to Artificial Life (ER'97), pp. 221–256. AAI Books, Tokyo, Japan (1997)
11. Myers, K.L.: Cpef: A continuous planning and execution framework. AI Magazine **20**(4), 63–70 (1999)
12. Nareyek, A.: Intelligent agents for computer games. In: Computers and Games, Second International Conference (CG 2000), *Lecture Notes in Computer Science*, vol. 2063, pp. 414–422. Springer (2002)
13. Nilsson, N.J.: Teleo-reactive programs for agent control. Journal of Artificial Intelligence Research **1**, 139–158 (1994)
14. Partington, S.J., Bryson, J.J.: The Behavior Oriented Design of an Unreal Tournament character. In: T. Panayiotopoulos, J. Gratch, R. Aylett, D. Ballin, P. Olivier, T. Rist (eds.) The Fifth International Working Conference on Intelligent Virtual Agents, pp. 466–477. Springer, Kos, Greece (2005)
15. Pollack, M.E., Horty, J.F.: There's more to life than making plans. AI Magazine **20**(4), 71–83 (1999)
16. Sloman, A.: Prolegomena to a theory of communication and affect. In: A. Ortony, J. Slack, O. Stock (eds.) Communication from an Artificial Intelligence Perspective: Theoretical and Applied Issues, pp. 229–260. Springer, Berlin, Heidelberg (1992)
17. Sloman, A.: The "semantics" of evolution: Trajectories and trade-offs in design space and niche space. In: H. Coelho (ed.) Progress in Artificial Intelligence, 6th Iberoamerican Conference on AI (IBERAMIA), pp. 27–38. Springer, Lecture Notes in Artificial Intelligence, Lisbon (1998)
18. Sloman, A.: Varieties of affect and the cogaff architecture schema. In: Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing, pp. 1–10 (2001)
19. Sloman, A., Logan, B.: Building cognitively rich agents using the sim_agent toolkit. Communications of the ACM **43**(2), 71–77 (1999)
20. Wilkins, D.E., Myers, K.L., Lowrance, J.D., Wesley, L.P.: Planning and reacting in uncertain and dynamic environments. Journal of Experimental and Theoretical AI **6**, 197–227 (1994)