Logical Agents for Language and Action*

Martin Magnusson and **Patrick Doherty**
Department of Computer and Information Science
Linköping University, 581 83 Linköping, Sweden
marma@ida.liu.se,patdo@ida.liu.se

Abstract

Game developers are faced with the difficult task of creating non-player characters with convincing behavior. This commonly involves an exhaustive specification of their actions in every conceivable situation that might arise. The process is laborious and the results do not apply in new and unforeseen circumstances. We present an agent architecture where game characters have a basic understanding of their environment that enables them to figure out what to do by themselves. Knowledge about facts and actions is encoded in a formal logic where automated reasoning technology can apply the knowledge to answer novel questions in dialog and to plan actions in new situations. A number of examples serve to demonstrate how the technology meets the challenges of application in a simple computer game prototype. We envision this technology being used to create more flexible game characters with a deeper understanding of the world they inhabit.

Introduction

The behavior of most non-player characters (NPCs) in computer games relies on the developers' foresight. Game play programmers meticulously design finite state machines that specify what NPCs should do in every state they can end up in, script writers painstakingly construct dialog trees that completely determine all possible conversational exchanges, and game designers carefully craft quests where the NPCs play out predefined scenarios. But a hallmark of intelligence is the ability to deal with the unexpected. Unforeseen situations pose problems for state machines, novel questions are impossible in scripted dialog, and dynamic scenarios call for the consideration of alternative courses of action.

A few games have successfully attacked one or more of these challenges using artificial intelligence technology. One example is the application of a planning algorithm, reminiscent of the classic STRIPS planning framework, in Monolith's game F.E.A.R. (Orkin 2005). Enemy soldiers plan sequences of actions for taking cover and attacking the player instead of relying on finite state machines. The tight time constraints enforced by the real-time combat situation necessitated a very limited planning model, but it is nevertheless a demonstration of feasibility.

EA's game series The Sims features characters that display a broader range of autonomous action (EA 2007). Part of the game's appeal is the player's ability to modify the environment and see how this causes the Sims to modify their behavior. This behavior is reactive, in response to the characters' needs and desires, rather than being based on a planning framework. The game adds an element of dialog, which is an important behavior that Sims engage in. But the sounds that they utter have no content or meaning, and the player has no means of participating in any conversation.

This is possible to some degree in the game Creatures from Millennium Interactive (Grand, Cliff, & Malhotra 1997). The player is able to teach life-like creatures with neural network "brains" limited language skills and to watch them reproduce and evolve over generations through genetic algorithms. Creatures successfully manages to build an entertaining game using advanced artificial life technologies.

Academia has approached the challenge of intelligent game characters from at least two different viewpoints. The Oz project (Bates 2002) and the Virtual Theater project (Doyle 2001) emphasized *interactive drama*. They attempted to provide programming languages and tools for creating dramatic stories that involve *believable agents* who appear intelligent to the player. This is an author intensive solution where character behaviors are scripted beforehand to ensure a high degree of control over the result.

In contrast, more traditional artificial intelligence research, such as the Soar architecture, emphasize agent autonomy. Soar is based on the use of production rules, which the Soar/Games project coupled to Quake 2 and Descent 3 (Laird & van Lent 1999). This allowed the researchers to build reusable rule bases for autonomous agent behavior.

Unlike Soar, which is not primarily a planning framework, SquadSmart (Gorniak & Davis 2007) demonstrates how hierarchical task network planning can be used for multi-agent planning in games. An incremental algorithm ensures bounded computation time and an execution module synchronizes actions and copes with failures.

Game contexts also provide excellent opportunities to study natural language understanding problems such as reference resolution in circumscribed environments (Gorniak,

*This work is supported in part by the National Aeronautics Research Program NFFP04 S4203, CENIIT, and the Strategic Research Center MOVIII, funded by the Swedish Foundation for Strategic Research, SSF.

Orkin, & Roy 2006). But the complexity of unrestricted language makes for prohibitively difficult challenges, even given the limited domains of a game applications. We think a more immediately practical approach is to restrict the natural language input, e.g. to exclude ambiguity, while still allowing much more freedom than in traditional dialog trees.

In this paper we present an agent architecture where logic is used to represent the knowledge of NPCs in a declarative form that is applicable even in unforeseen situations, theorem proving derives answers to dialog questions that were not scripted beforehand, and reasoning about action finds plans for scenarios that were not predetermined. The technology has been applied to a simple dialog-based adventure game depicted in Figure 1. The reader is encouraged to experiment with our demonstrator, ANDI-Land, that is available for download at `www.andi-land.com`.

Logical Agents

The characters of ANDI-Land are autonomous agents with Augmented Natural Deductive Intelligence called ANDIs.

Each ANDI is equipped with a knowledge base, i.e. a database of logical formulas that encode the character's knowledge about the laws that govern the game world and its memory of recent events. The representation is based on Temporal Action Logic (Doherty & Kvarnström 2007), a very expressive logic designed for commonsense reasoning problems that involve action and change, but the computer game application necessitated new extensions for reasoning about other agents, beliefs, and communication.

ANDIs are also equipped with an automated theorem prover for reasoning with the content of the knowledge base. Rather than working with an off-the-shelf resolution prover we have developed an automated natural deduction system based on the use of an efficient quantifier-free form of logic (Pollock 1999). The proof rules used in natural deduction are more intuitive and the resulting proofs are similar to human reasoning (Rips 1994). Moreover, the rule set is extensible and easily accommodates special purpose rules, e.g. to integrate an efficient pathfinding algorithm.

An agent architecture based exclusively on logic and theorem proving might not be suitable for the split-second decisions required in first-person shooter combat situations. But achieving satisfactory performance for game applications is certainly possible. NPCs in our ANDI-Land game prototype display no noticeable delay in their responses. While all of the theorem prover's functionality has not yet been integrated with the game prototype, let alone a full size game world, there are many opportunities for improving performance. A few planned enhancements include porting the implementation from Lisp to C++, using constraint solvers for efficient reasoning in common domains (Magnusson & Doherty 2007), and introducing heuristic search.

What we gain from the generality of logical reasoning is a simple but highly capable architecture where the many problems faced by partially or fully autonomous agents are formulated in a unified way in terms of proof. In the following we shall introduce the key features of our current system with the help of example natural language input from a player, formulas from the underlying logical representation, as well as NPC responses.

Figure 1: Magni meets Smith in ANDI-Land

Questions and Answers

Most games that feature a dialog element use pre-authored questions and answers. ANDIs parse natural language input into logical formulas and generate their replies on the fly. It would, however, be unrealistic to attempt a deep understanding of *arbitrary* natural language input. Full natural language understanding is believed to be an AI-complete problem, i.e. a problem impossible to solve without solving all other problems in artificial intelligence at the same time. Instead, the ANDI-Land game's input interface uses a unique interactive natural language parser to guide the player towards sentences that are relevant in the particular game. A chart parser, based on Russel and Norvig's description (2003), produces a partial parse as soon as the player starts typing. Though the initial words do not constitute a complete sentence, the resulting parse chart still contains useful information. Specifically, by looking at active chart edges we can collect all words that would advance the parse if they were next in the input. These word suggestions are presented to the player who chooses one by typing or clicking it. E.g., if the player start typing "who owns", the list of suggestions include all of the objects previously encountered in the game.

The interactive input interface supports an open dialog that does not limit the player's choices to a handful of alternatives, as dialog trees do. This is a challenge for any mechanism used by the NPCs to find appropriate responses. Suppose, e.g., that Magni (the player) enters the shop of Smith (a metalworker) and sees an axe. There is no scripted scenario or dialog that the player must adhere to, but he thinks the axe might be of importance. He asks:

Magni: Who owns the axe?

Though Smith has never seen this question before, he uses his general reasoning mechanisms to answer it. To be able to translate the input into a logical formula that he can reason with he needs some logical representation of questions. We follow Searle (1969), and many others, in treating communication as consisting of a special kind of actions, so called *speech acts*. The above input can be viewed as a request to give the speaker a reference to the current owner of the axe:

$$\textit{informRef}(\mathrm{magni}, \textit{value}(12{:}15, \mathrm{owner}(\mathrm{axe})))$$

The function *value* denotes the value of owner(axe) at the given time point, where the clock time 12:15 should be read

as an abbreviation of a Unix time stamp uniquely identifying both a date and time. Assuming that Smith owns the axe at that time, the simplest response would be to inform Magni that *value*(12:15, owner(axe)) = $smith$. However, if all that was required to satisfy a request for a *reference* to a person or object was to supply something that was *equal* to that person or object, then an obnoxious NPC might respond with the tautological *value*(12:15, owner(axe)) = *value*(12:15, owner(axe)), i.e. "The owner of the axe owns the axe". Instead we require an answer that provides a name *string* that identifies the axe's owner. Smith uses a special *Id* relation to inform Magni that the axe's owner is identified by the string "smith":

inform(magni,
"*Id*("*value*(12:15, owner(axe))", "smith")")

Simply passing the *Id* relation as an argument to the *inform* action constitutes a syntax error in first-order logic. But the above formula is surrounded by quotation marks. While quoted formulas are not part of any introductory logic text books, they can be incorporated into standard first-order logic through a translation from quoted formulas into regular first-order terms (Morgenstern 1988). The ability to turn formulas into terms and pass them as arguments to speech acts is the enabling feature of our formulation of communication between agents.

Of course, we would prefer Smith's reply to use plain English. But since Smith has never seen Magni's question before, there is no way he could have an English translation of the answer pre-stored. Shieber's uniform architecture for natural language parsing and generation addresses this difficulty with minimal machinery (Shieber 1988). It effectively makes the natural language grammar reversible with relatively small modifications to the basic chart parsing algorithm. First, a simple mechanism introduces personal pronouns based on the current dialog context. Then, running the chart parser in "reverse" on Smith's answer formula generates an English response to the original question, here together with a follow up question about what Smith would charge for the axe, which is processed in the same way:

Smith: I own the axe.
Magni: How much is the axe?
Smith: The axe is 5 gold.

The question answering mechanism just described does not require pre-authored dialog but relies instead on a generative context free grammar. While our intention is to construct a larger grammar with more natural-sounding sentences, even a relatively small grammar can produce an exponential number of novel sentences. We hope that this will encourage players to solve game puzzles by reflecting on what they intend to accomplish with the dialog rather than mindlessly exhausting the handful of alternatives offered by dialog trees.

Ignorance and Learning

Dialog can also be used to teach NPCs simple facts. Consider e.g. the following continuation of the previous dialog between Magni and Smith:

Magni: How much gold do I own?
Smith: I don't know.
Magni: I own 6 gold.

As before, the question results in a request for information that Smith tries to satisfy. But this time he lacks the requested knowledge and gives a default "I don't know" response when his proof attempt fails. Magni tells Smith about his wealth through the *inform* action, again using the *Id* relation as content:

inform(smith,
"*Id*("*value*(12:18, gold(magni))", "6")")

Smith applies his theorem prover to the new input to make sure that it doesn't conflict with his existing knowledge. The proof does not result in any contradictions and Smith adds the content to the other beliefs in his knowledge base and replies:

Smith: I see.
Magni: How much gold do I own?
Smith: You own 6 gold.

Declarative knowledge, like the logical representation used by Smith and other ANDIs, can be additively expanded. This makes it possible to teach the ANDIs new facts. The new knowledge is automatically put to use by the reasoning mechanism, e.g. to answer subsequent questions as was done above.

Reasoning and Proof

Sometimes the answer to a question is not stored in an NPC's knowledge base but is an *implicit* consequence of it. The ANDI theorem prover finds such answers by deductive proof with the formulas in the character's knowledge base as premises. The following question is a case in point:

Magni: Is my gold more than the axe's price?

Yes/no questions correspond to requests for information about whether the statement is true or false. The natural language grammar parses this as an *informIf* speech act:

informIf(magni, "*value*(12:20, gold(magni)) >
value(12:20, price(axe))")

While Smith has neither seen this question before, nor has any sentences of the above form in his knowledge base, he is still capable of answering. His natural deduction prover includes a set of rewrite rules for mathematical reasoning that successively simplify expressions involving relations between known quantities. The proof is successful and Smith replies affirmatively:

Smith: Yes, your gold is more than the axe's price.

Automated reasoning vastly extends the coverage of an ANDI's knowledge. It is what makes the open natural language dialog feasible since novel input would leave an NPC without general reasoning capabilities perplexed.

Actions and Execution

We have seen examples of speech acts for what-questions (*informRef*), yes/no-questions (*informIf*), and statements of facts (*inform*). But the player's next sentence involves an action that is not only communicative:

Magni: Sell the axe to me.

The grammar recognizes the above as an imperative command where Magni requests Smith to perform the action of selling him the axe. The logical representation of the requested action occurrence uses the *Occurs* relation that takes an acting agent, a time interval, and an action instance as arguments. In this case the time interval $(t_1, t_2]$ is left undetermined while the agent supposed to do the selling is Smith and the agent he is selling to is Magni:

$$\exists t_1, t_2\, [\mathit{Occurs}(\mathrm{smith}, (t_1, t_2], \mathrm{sell}(\mathrm{axe}, \mathrm{magni}))]$$

As always, Smith has no predefined response but passes the input to his theorem prover. Of course, we should not expect a deductive proof to succeed unless Smith has in fact already performed the requested action. But the ANDI theorem prover is not purely deductive. A special *abduction* proof rule can be used to schedule an action for execution, thereby committing to the action occurrence. However, committing to executing an action is not enough to prove that the action will actually occur. One needs to make sure that it is possible to execute the action and that one knows exactly what action one is supposed to execute. The former condition is satisfied, since Smith owns the axe and Magni can afford it, but could otherwise have required additional actions to satisfy. The latter condition is satisfied if Smith knows a name *string* identifying the action, as denoted by an *ActionId* relation. Knowing how to sell the axe to Magni is uncomplicated. More difficult would be e.g. to sell the axe to the highest bidder. Such an action would first involve finding out who the highest bidder is (Moore 1980). Smith's ability to satisfy Magni's request, then, is captured by the following instance of a general action occurrence axiom:

$$\begin{aligned}&\mathit{Committed}(\mathrm{smith}, t_1,\\ &\quad \text{“}\mathit{Occurs}(\mathrm{smith}, (t_1, t_2], \mathrm{sell}(\mathrm{axe}, \mathrm{magni}))\text{”}) \wedge\\ &\mathit{Executable}(\mathrm{smith}, (t_1, t_2], \mathrm{sell}(\mathrm{axe}, \mathrm{magni})) \wedge\\ &\mathit{Believes}(\mathrm{smith}, t_1,\\ &\quad \text{“}\mathit{ActionId}(\mathrm{sell}(\mathrm{axe}, \mathrm{magni}), \mathrm{sell}(\mathrm{axe}, \mathrm{magni}))\text{”}) \rightarrow\\ &\mathit{Occurs}(\mathrm{smith}, (t_1, t_2], \mathrm{sell}(\mathrm{axe}, \mathrm{magni}))\end{aligned}$$

The action that Smith has committed to is then executed by a procedure call at the appropriate time, which is as soon as possible in this case. The logical representation of actions and the abductive proof rule for scheduling actions make it possible to treat Magni's command in the same way as any other request, namely as a formula to be proved.

Persistence and Change

Actions, such as Magni's purchase of the axe, change some aspects of the world while leaving most unaffected. It is not trivial for an NPC with incomplete knowledge of the world to predict which aspects change and which persist. In fact, providing a concise representation that supports such predictions is the (in)famous *frame problem*. This is precisely what the *occlusion* concept in Temporal Action Logic was designed to deal with.

Properties and relations that may change over time, such as owner(axe), are called *fluents*. They can be given different values at different time points using the *value* function. But rather than having to specify their values at every time point, we want to implement the blanket assumption that their values persist over time. This assumption, however, is not always true since actions change the values of fluents. The role of occlusion is to selectively release fluents from persistence, i.e. giving them permission to change. By occluding fluents at specific time points one obtains a fine grained control over their persistence. In particular, actions can change fluents by first temporarily occluding them.

The resulting behavior is exemplified by the following dialog with another NPC, the woodsman Jack, who does not know of Magni's purchase:

Magni: Who owned the axe yesterday?
Jack: Smith owned the axe yesterday.
Magni: Who owns the axe today?
Jack: Smith owns the axe today.

Jack knew that Smith used to own the axe (where "yesterday" is simply interpreted as "24 hours ago"). But when asked about its current owner he has no explicit knowledge and therefore makes use of the blanket assumption of no change. The special abduction proof rule, mentioned previously, lets Jack assume that the axe has not changed owners, i.e. that owner(axe) is not occluded, as long as doing so does not lead to inconsistency. The following axiom then propagates the value of a fluent f over a time interval $(t_1, t_2]$:

$$\neg \mathit{Occlude}((t_1, t_2], f) \rightarrow \mathit{value}(t_1, f) = \mathit{value}(t_2, f)$$

Jack uses the axiom and his non-occlusion assumption to project his knowledge that Smith owned the axe yesterday forward in time. His prediction is reasonable, given that he is not aware of the purchase which changes the owner(axe) fluent (as well as the gold fluent for both the buyer and seller).

The purchase action, however, explicitly occludes the affected fluents. Therefore, if Magni tells Jack that he bought the axe at some earlier time point, Jack will have to conclude that owner(axe) *is* occluded, which contradicts his previous assumption that it was *not* occluded. With this new information Jack believes that Magni rather than Smith currently owns the axe, and he can apply a new persistence assumption to predict that this will remain so:

Magni: I bought the axe from Smith.
Jack: I see.
Magni: Who owns the axe today?
Jack: You own the axe today.
Magni: Who will own the axe tomorrow?
Jack: You will own the axe tomorrow.

Persistence assumptions of this kind make it possible for the NPCs to draw reasonable conclusions despite having insufficient information and being surrounded by a dynamically changing world, such as those in modern computer games.

Goals and Plans

The same logical representation of actions used for reasoning about the changes caused by known action occurrences can be used for goal-oriented action planning (GOAP) to achieve desired outcomes. Suppose, e.g., that the woodsman Jack has a goal to obtain lumber, represented by the following formula:

$$\exists t\,[\mathit{value}(t, \mathrm{owner}(\mathrm{lumber})) = \mathrm{jack} \wedge t > 12{:}25]$$

Jack tries to satisfy this goal in the same way that he would try to satisfy a request communicated to him in dialog, by proving it. The proof uses his knowledge of available actions and their effects on the world. Jack knows that having lumber is an effect of felling trees, but that a precondition of the chop action is the possession of an axe, which he does not currently have. Since Jack now knows that Magni owns the axe he plans to ask Magni to sell him the axe and then use it to cut down a tree. Jack arrives at the following plan of action through an abductive proof of his goal:

$$\begin{array}{l}\exists t_1, t_2, t_3, t_4, t_5, t_6, [\\ \quad \mathit{Schedule}(\mathrm{jack}, (t_1, t_2],\\ \quad\quad \mathit{request}(\mathrm{magni}, \text{“}\mathit{Occurs}(\mathrm{magni}, (t_3, t_4],\\ \quad\quad\quad \mathrm{sell}(\mathrm{axe}, \mathrm{jack}))\text{”})) \wedge\\ \quad \mathit{Schedule}(\mathrm{jack}, (t_5, t_6], \mathrm{chop}) \wedge\\ \quad t_1 < t_2 < t_3 < t_4 < t_5 < t_6]\end{array}$$

Planning is not exclusively used for achieving the agent's own goals. Any questions and commands from other agents may give rise to planning. Suppose e.g. that Jack is asked by the player to go visit Smith, but that he does not know where Smith is. Assuming that Magni does know, he plans to request Magni to inform him about Smith's location and then to walk there:

$$\begin{array}{l}\exists t_1, t_2, t_3, t_4, t_5, t_6, [\\ \quad \mathit{Schedule}(\mathrm{jack}, (t_1, t_2],\\ \quad\quad \mathit{request}(\mathrm{magni}, \text{“}\mathit{Occurs}(\mathrm{magni}, (t_3, t_4],\\ \quad\quad\quad \mathit{informRef}(\mathrm{jack},\\ \quad\quad\quad\quad \mathit{value}(12{:}45, \mathrm{location}(\mathrm{smith}))))\text{”})) \wedge\\ \quad \mathit{Schedule}(\mathrm{jack}, (t_5, t_6],\\ \quad\quad \mathrm{walk}(\mathit{value}(12{:}45, \mathrm{location}(\mathrm{smith})))) \wedge\\ \quad t_1 < t_2 < t_3 < t_4 < t_5 < t_6]\end{array}$$

If the player last saw Smith at the town square the scenario would play out in English as follows:

Magni: Go to Smith.
Jack: Where is Smith?
Magni: Smith is at the town square.
Jack: I see.

The declarative representation of actions gives ANDIs the capability both of executing given plans and of goal-oriented action planning when no predefined plan is given. Jack's plan to buy the axe makes use of the information he learned from Magni previously. His plan to ask Magni regarding Smith's location satisfies the requirement to know an identifier for the walk action and is made possible by the representation of questions as regular speech acts amenable to planning. Such capabilities make the NPCs better equipped to deal with unforeseen circumstances.

Lies and Trust

In the previous dialog Jack proactively posed a question to the player. This hints at an interesting challenge. If the player, accidentally or on purpose, misinforms an NPC then the NPC risks acting on false information and, worse, believing a contradiction. In logic, anything follows from a contradiction so the effect would be disastrous. The poor character would suddenly believe anything and everything, effectively making him go mad.

The solution is to realize that no one believes everything they hear (unless they are already mad). Rather, if you are the hearer h, you believe your own percept of hearing a speaker s telling you something, but only believe the content c of what was said if you find the speaker to be trustworthy. This insight can be captured by an axiom:

$$\begin{array}{l}\mathit{Occurs}(s, (t_1, t_2], \mathit{inform}(h, c)) \rightarrow\\ \quad (\mathit{Trustworthy}(s) \rightarrow \mathit{Believes}(h, t_2, c))\end{array}$$

Furthermore, if you later withdraw your belief in the trustworthiness of the person, you should also withdraw the beliefs that were consequences of the untrustworthy person's allegations. (While various methods of *reestablishing* trust are possible, the simplest being to forget about the lie after a certain time period, we leave this decision open for now.)

The mechanism is illustrated in the following dialog where the player finds Jack with his newly cut small heap of lumber and someone's pick lying on the ground. Jack does not know whose pick it is and the player tries to fool him by claiming ownership of it:

Magni: I own the pick.
Jack: I see.

Jack assumes Magni to be trustworthy and therefore, by the above axiom, believes his claim. But the player, emboldened by this success, tries his luck with the lumber too:

Magni: I own the lumber.
Jack: No!
Magni: Who owns the lumber?
Jack: I own the lumber.

Magni's assertion that he owns the lumber together with the trustworthiness assumption leads to a contradiction with Jack's previous belief that *he* owns the lumber. A truth maintenance system detects the contradiction and withdraws it from Jack's knowledge base. The trustworthiness assumption, and formulas marked as having been derived from it, must also be withdrawn to prevent the contradiction from reappearing. Jack is now undecided regarding the owner of the pick and will only grant Magni a default "maybe" in response to his repeated claim to own it.

Magni: Who owns the pick?
Jack: I don't know.
Magni: I own the pick.
Jack: Maybe.

Default trust makes it possible to fool NPCs, but only if one is careful not to attempt an obvious lie since the ANDIs' truth maintenance system safeguards them from believing contradictions.

Failure and Recovery

Dropping contradictory beliefs may have consequences on the execution of plans. If a belief is necessary for a plan of action to satisfy some goal, dropping it may cause that plan to fail. But there are also other reasons for why a plan may fail, such as disruptive events that are not under the agent's control. Common to all such reasons is that they necessitate a method of graceful recovery from plan execution failures.

Consider again Jack's plan to visit Smith. Jack has already executed the first part, namely asking Magni for directions. But, in light of recent events, these directions can not be trusted. Specifically, Jack's belief that walking to the town square is the same as walking to Smith's location is disbelieved at the same time as Magni's trustworthiness is disbelieved. Since Jack's proof that his plan achieves the goal depends crucially on this belief, the truth maintenance system must also withdraw the fact that the goal has been satisfied. The ANDI theorem prover is forced to reestablish the goal by finding an alternative proof, e.g. by asking another NPC for the information. This plan revision is an automatic consequence when predictions made during planning are contradicted by percepts during execution and enables ANDI agents to pursue goals even in unpredictable environments where plans have no guarantee of success.

Conclusions

Game characters are usually not equipped to deal with new circumstances, novel dialogs, and unscripted scenarios. The problem is that they have no real understanding of the world that surrounds them. They are told what to do rather than taught how the world works. If what they were told to do is no longer appropriate they have no means of figuring out an altered course of action that *is* appropriate. Research on autonomous agents, such as that presented in this paper, is focused on precisely this challenge of building agents that use their knowledge to figure out what to do by themselves.

To meet the challenges we have made extensive use of logic. While the use of logic as a *theoretical* foundation is relatively commonplace, it is often considered too inefficient and brittle for practical application. A theorem prover can potentially churn away indefinitely on a problem and a single contradiction can throw it into permanent insanity. But this is only true for applications that insist on agents with *complete* reasoning and naïve trust. If NPCs use incomplete, time bounded, reasoning and only believe statements as long as there is no reason to judge the source as unreliable, these problems can be effectively dealt with.

With these premises we have built a logical agent architecture and a practical application in the ANDI-Land computer game prototype. The logical representation enables a uniform treatment of problem solving in terms of proof. ANDIs use theorem proving technology to apply world knowledge in new situations, find answer to novel questions, and plan courses of action to satisfy their goals. Much work remains before the technology is sufficiently efficient and robust for larger scale applications, but it has the potential to create NPCs with an understanding of their environment that finite state machines and decision trees lack.

Our vision is to populate game worlds with characters that live out their lives autonomously. High level goals to survive and thrive is the source of more immediate goals, plans, and proactive action. One possible application is a functioning miniature economy where the participants produce and trade with necessities like food and tools. An inquisitive player questioning the characters is not met by mindless machines but with autonomous agents that know what they are doing and why. Such NPCs present challenging problems for any logical theory of agents. But a successful logical theory of agents can help in the challenge of inhabiting games with compelling NPCs.

References

Bates, J. 2002. The Oz project. `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/oz.html`.

Doherty, P., and Kvarnström, J. 2007. Temporal action logics. In *Handbook of Knowledge Representation*. Elsevier.

Doyle, P. 2001. The virtual theater project. `http://www.ksl.stanford.edu/projects/cait/`.

EA. 2007. The Sims. `http://thesims.ea.com/`.

Gorniak, P., and Davis, I. 2007. Squadsmart: Hierarchical planning and coordinated plan execution for squads of characters. In *Proc. of AIIDE'07*, 14–19.

Gorniak, P.; Orkin, J.; and Roy, D. 2006. Speech, space and purpose: Situated language understanding in computer games. CogSci'06 Workshop on Computer Games. `http://petergorniak.org/papers/gorniak_games_cogsci_2007.pdf`.

Grand, S.; Cliff, D.; and Malhotra, A. 1997. Creatures: Artificial life autonomous software agents for home entertainment. In *Proc. of Agents'97*, 22–29.

Laird, J. E., and van Lent, M. 1999. Developing an artificial intelligence engine. In *Proc. of GDC'99*, 577–588.

Magnusson, M., and Doherty, P. 2007. Deductive planning with temporal constraints. In *Proc. of Commonsense'07*.

Moore, R. 1980. Reasoning about knowledge and action. Technical Report 191, AI Center, SRI International.

Morgenstern, L. 1988. *Foundations of a logic of knowledge, action, and communication*. Ph.D. Dissertation, New York, NY, USA. Advisor: Ernest Davis.

Orkin, J. 2005. Agent architecture considerations for real-time planning in games. In *Proc. of AIIDE'05*, 105–110.

Pollock, J. 1999. Natural deduction. Technical report, Department of Philosophy, University of Arizona. `http://www.sambabike.org/ftp/OSCAR-web-page/PAPERS/Natural-Deduction.pdf`.

Rips, L. J. 1994. *The psychology of proof: deductive reasoning in human thinking*. MIT Press.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition.

Searle, J. R. 1969. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press.

Shieber, S. M. 1988. A uniform architecture for parsing and generation. In *Proc. of COLING'88*, 614–619.