Knowledge-Gathering Agents in Adventure Games

Brian Hlubocky and **Eyal Amir**
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{hlubocky,eyal}@uiuc.edu

Abstract

Computer text-based adventure games are virtual worlds in which humans or artificial agents act towards a specified goal through a text-based interface. In this paper we describe progress towards an agent that can interact with a game world in a human-like fashion. Precisely, we present the first accurate knowledge-gathering software agent that can track the state of the world in a text-based adventure game. This is nontrivial because such games are characterized by large, partially observable domains that include many objects, actions, and relationship between them. To test our agent, we developed a text-based adventure game world built by augmenting the LambdaMOO code base. We translated this game world into First-Order Logic, and used a logical filtering algorithm to allow the agent to track itself and the world efficiently. We expect that the development of software agents that act intelligently in such games will give rise to new types of games and will contribute to research on human-level artificial intelligence.

1 Introduction

We are interested in creating an agent that can play a text-based adventure game using well-founded, knowledge-based principles. Our motivation for this project is twofold. In the short term, we seek to create smart games in which intelligent agents interact believably in the world. Looking into the future, we expect the technology that will come out of this research to contribute to building human-level AI software that can act in rich, large domains.

Having a world populated by agents that learn and act like humans makes games more realistic, interesting, and enjoyable. Also, one can envision games in which an intelligent agent collaborates with a human player to achieve certain goals, and successfully playing the game becomes dependent on experiencing it together with the agent. In contrast, current agents fill a few narrowly defined roles in most games, never acting outside of those roles.

The technical problems faced when building such agents are not simple. A typical domain in text-based adventure games consists of a partially observable and potentially nondeterministic world, with an extremely large state space (greater than 2^{40} states). This renders unusable traditional planning and learning algorithms, such as reinforcement learning [Kaelbling et al., 1996] and POMDPs. New algorithms for tracking the world, learning, and exploration are required.

In this paper, we describe our experiences, including current progress and difficulties encountered in creating an intelligent agent that plays computer text-based adventure games. Currently, our agent is able to track itself and the evolving world efficiently in response to the agent's actions and observations. We present the overall architecture for our agent, together with the current state of our algorithms and this architecture. Currently, the architecture also includes a hand-made translation of the text interface into a logic-based interface that the agent receives and outputs. Our translation serves as a test case for natural-language processing (NLP) theories, and we expect that it will serve to guide future NLP research and act as a training corpora for future NLP applications.

We are not the first to examine the use of text-based adventure games in the context of cognitive robotics. The challenges associated with using adventure games for AI research are explored in [Amir and Doyle, 2002]. The use of a description logic knowledge base as the core for a text-based adventure game is investigated by [Gabsdil et al., 2001; Koller et al., 2002]. One example of a text-based adventure agent implemented using a MUD is [DePristo and Zubek, 2001], which focuses mostly on the selection of actions necessary for survival in a combat-oriented world.

This paper is organized as follows. In section 2, we introduce text-based adventure games and describe the particular game we chose as a research platform. Section 3 details the process used to translate (by hand) our game world into First-Order Logic. Logical filtering is described in section 4, along with a discussion of our choice of an efficient filtering algorithm. Section 5 describes the desired agent architecture and our current progress toward that goal.

2 Text-Based Adventure Games

Computer text-based adventure games appeared about 25 years ago, reaching a high-point commercially in the late 1980s. They are promising for a new frontier in computer games in the manner that we describe in the rest of the paper.

In the following we describe some characteristics of text-based adventure games and list factors that make them useful for research in cognitive robotics. We also introduce a specific text-based adventure that was used in our research.

The text-based adventure is a computer game in which the player interacts with the world by entering natural language commands using a text-based interface. Some of the text-based adventure's most distinctive characteristics as identified by [Amir and Doyle, 2002] are:

- The game world is relatively static
- Effects of actions are local and immediate
- Actions are discrete
- There is no uncertainty in acting or sensing
- Finishing the game generally involves lots of exploration
- No action can make the game unbeatable
- The player is assumed to possess a required amount of commonsense knowledge

The characteristics of these games make them a fruitful platform for AI research in cognitive robotics. [Amir and Doyle, 2002] describes several factors that make text-based adventures challenging research problems. At the beginning of the game, only a small amount of information is known about the world. The number of transitions from any given state is unknown to the player, as the environment is only partially observable. An agent must simultaneously learn action effects and preconditions while tracking the world. Another challenge is the large state and action space presented by most text-based adventures. A game with 100 objects and 100 rooms has $2^{10,000}$ different states, necessitating efficient algorithms for learning and filtering.

Multi-User Dungeons (MUDs) are a unique type of text-based adventure game that have multiple players interacting simultaneously with each other and their environment. The natural separation between the game world (server) and its occupants (clients) make MUDs an ideal platform for agent building. MUDs are also programmable, enabling the easy creation of new environments, objects, and actions. In most cases, these programming languages allow almost any situation to be modeled in the game world. As agents are being built, nearly all of the previously described characteristics of text-based adventure games can be relaxed, making the world more challenging. For example, non-local action effects and noise in sensing and perceiving can be added.

In 1990, Xerox PARC scientist Pavel Curtis created LambdaMOO, a MUD whose focus was social interaction and the creation of new environments. After considering several different MUD code bases, we chose LambdaMOO as our research platform for several reasons. First, the MUD is easy to install and run. Second, it separates the details of the world from the core MUD code. This allows us to make changes to the game world without having to recompile the code. It also makes it easy to backup and transmit the database to different computers. Third, the provided database is feature rich. It contains support for most natural actions as well as commands to easily create new rooms, objects, and actions. Fourth, it focuses heavily on player-world and player-player interaction. Many MUDs are combat oriented and are not of interest to us. Finally, it provides a powerful programming language. This language is similar to C and can be used to create just about any behavior imaginable.

3 Text Translation into Logic

In this section we describe the need for text translation, the steps we took to translate the game world into logic, and the difficulties we encountered. For an agent to be able to reason about its environment, it must either have sensors that transmit information in a form it can understand, or be able to convert the sensed information after the fact. Because our work is not concerned with conversion of natural language sentences into logic, we modified our game engine to output logical sentences directly. This requires the translation of all aspects of the game world into logic.

To accomplish this, we first created a small game world (just a few rooms), that was representative of the kinds of objects and actions we required. Then we created all of the predicates needed to fully describe this world in logic. Using these predicates, we defined STRIPS-like actions (preconditions and effects) that were required to interact with the game world. Finally, the MUD was modified to output logical sentences composed of the previously defined predicates.

Although the translation of a game world into logic is a very straightforward procedure, we encountered several difficulties along the way. First, some of the predefined LambdaMOO actions behaved in a manner that contradicted commonsense assumptions. For example, the *open* action, when performed on a locked container, would only succeed if the actor was carrying the required key. When successful, the container would automatically unlock and open, and would lock again when closed. We expect to have to explicitly issue an *unlock* command (specifying the correct key), before the container can be opened. In these situations, the offending actions were redefined to better match commonsense expectations.

Another difficulty we experienced was the underestimation of the number of different action outcomes. Whenever an action succeeds or fails, the agent makes an observation. If the action succeeds, then the action's effects are observed. If the action fails however, any number of different observations might be seen, related to the complexity of the preconditions that must hold on the action for it to succeed. Because a human would be able to observe all of these causes of failure, each needs to be converted to logic and presented to the agent after the action is executed.

4 Logical Filtering

Logical filtering is the process whereby an agent updates its belief state, or knowledge of the state of the world (represented in logical formulae), from a sequence of actions and observations. In our case, the agent receives observations from the game server, and performs actions that the server executes. At every point in time, the agent has knowledge that is represented in ground First-Order Logic (this is equivalent to propositional logic). When it performs an action in the world or receives an observation, it uses a logical filtering algorithm to update its knowledge with this information. At any point in time, the knowledge of our agent is about the current state alone, and not about past or future states.

The agent has knowledge about the effects of actions that is separate from the knowledge about the state (currently, this knowledge is given by the user).

The features of our game that most affect our choice of filtering algorithm are: 1) a nondeterministic and 2) partially observable world. Under these conditions, efficient updating of the agent's belief state can be achieved using the *NNF-filter* algorithm of [Amir and Russell, 2003]. This algorithm maintains a compact belief state representation indefinitely for our game, which is critical when working with game worlds with large state spaces (when we have n propositional features, this is $O(2^n)$ states). This algorithm has a linear runtime performance for an individual action in the number of propositions of the current belief state. Currently, our implementation of NNF-filter (written in Lisp) is quite slow, and we expect to benefit greatly from a future reimplementation.

5 Agent Architecture

One of our goals is to create an intelligent agent that interacts with the game world much like a human player would. This section first describes our progress toward reaching that goal, then discusses the desired agent architecture.

5.1 Current Architecture

Of the planned architecture, we currently have an agent that is able to track the world through the use of the telnet MUD client and filtering modules. The Python programming language was used to create the MUD client, filtering wrapper, and graphical interface. The filtering algorithm was implemented in Lisp and is executed by the filtering wrapper.

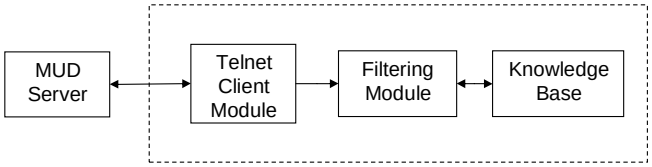

Figure 1: Our current implementation

To illustrate the operation of our system, we provide an example of our agent acting and observing in the game world. First, the agent code parses the output of the game server and constructs an observation. Below is a description of a room with an exit leading down, a tree trunk, and a key. This description is automatically generated by the MUD and sent to the agent.

(AT ME Tree_Branches) (AND (EXIT Tree_Branches down)
(INDIRECTION Tree_Branches down Base_of_the_Waterfall))
(AND (AT tree_trunk Tree_Branches) (AT gold_key Tree_Branches))

These observations are added to the agent's current knowledge base through the use of the (observe) action, which is the equivalent of issuing a "look" command to the MUD. We assume the agent is curious about the tree trunk, so it issues a "look tree_trunk" command to the MUD, which produces the following observations:

(CONTAINER tree_trunk)
(AND (NOT (CONTAINER-OPEN tree_trunk))
(CONTAINER-HASLOCK tree_trunk))

Filtering with the (observe) action again adds this new information to the agent's knowledge base. Now the agent knows that the tree trunk is an open container that has a lock on it. Next, the agent attempts to open the container by issuing the "opencontainer tree_trunk" command. In response, the MUD informs the agent that the action failed because the container was locked:

(ACTION-FAILED)
(CONTAINER-LOCKED tree_trunk)

Next, the agent adds this new information to its knowledge base and attempts to pick up the key in the room. To do this, it issues the "get gold_key" command. On seeing the action succeeded, the agent filters using the "(GET ME Tree_Branches gold_key)" action with the following observations:

(AND (PLAYER-HAS ME gold_key)
(NOT (AT gold_key Tree_Branches)))

The final belief state then becomes:

(AND (NOT (CONTAINER-OPEN TREE_TRUNK)) (NOT (AT GOLD_KEY TREE_BRANCHES)) (AT ME TREE_BRANCHES) (EXIT TREE_BRANCHES DOWN) (INDIRECTION TREE_BRANCHES DOWN BASE_OF_THE_WATERFALL) (AT TREE_TRUNK TREE_BRANCHES) (CONTAINER TREE_TRUNK) (CONTAINER-HASLOCK TREE_TRUNK) (CONTAINER-LOCKED TREE_TRUNK) (PLAYER-HAS ME GOLD_KEY))

In the current system, the agent must know about all of the actions, their preconditions, and their effects to track the world, however, in the future, the agent will be able to learn action preconditions and effects as it explores.

5.2 Planned Architecture

A human playing a text-based adventure game performs many important functions that help him/her to reach the goal (win the game). First, the player must track the world. This means the player's knowledge of the world must be kept up to date as a result of performing actions and receiving observations. This requires the player to learn which actions can be performed in a given situation (preconditions), as well as their effects once executed. In addition to learning, the player must be able to explore and make decisions that will help him/her reach the goal.

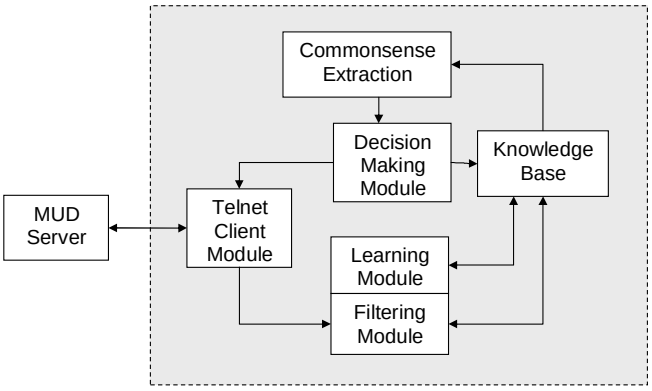

Figure 3: Our planned architecture

All of these functions are a part of our planned agent architecture. The telnet client module handles communication

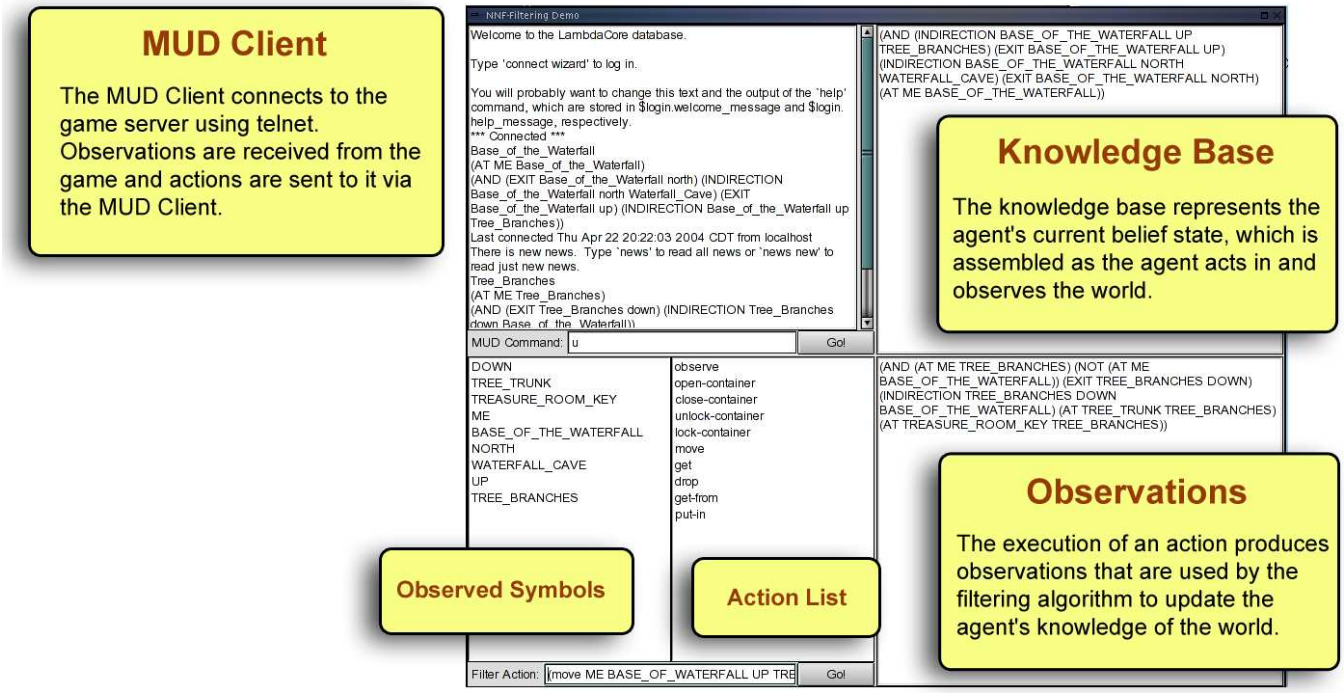

Figure 2: A screen capture of our current user-interface

with the MUD server. The MUD output is used by the filtering and learning modules to update the agent's knowledge base. The exploration and decision-making module combines the agent's knowledge of the world with a source of commonsense knowledge to plan and take action. These actions are then returned to the telnet MUD client module to be sent to the game server.

Our filtering algorithm enables our agent to track itself and the world, but it currently has no capability to learn about actions and their effects. There are many solutions to the learning problem, but in the context of text-based adventure games, none are well suited. In the game world, the agent is assumed to have no a priori knowledge of action effects or preconditions. It needs to be able to simultaneously filter (track) and learn with a partially known action model.

The second problem that needs to be addressed is that of exploration (planning). Traditional planning algorithms will not work with text-based adventure games, whose worlds are partially observable and whose state space is exponentially large. Also, the number of potential actions that can be applied in a given state is very large, making the problem even more challenging. External commonsense knowledge from projects such as OpenMind and Cyc can be used to aid in planning.

6 Conclusions

In this paper, we presented the first accurate knowledge-gathering software agent that can track the state of the world in a text-based adventure game. We described the problems associated with using text-based adventure games for research in artificial intelligence. We also discussed the process of translating the game world into First-Order Logic, our specific logical filtering algorithm, and our current and desired agent architecture. Our work thus far shows that our goal is feasible. We are in the process of developing the theory and implementation that are needed for a software agent that interacts on the level of a human with its environment.

References

[Amir and Doyle, 2002] Amir, E. and Doyle, P. (2002). Adventure games: A challenge for cognitive robotics (full version). AAAI'02 workshop on Cognitive Robotics. Also, available at the author's website (http://www.cs.uiuc.edu/~eyal/papers).

[Amir and Russell, 2003] Amir, E. and Russell, S. (2003). Logical filtering. In *IJCAI '03*, pages 75–82. MK.

[DePristo and Zubek, 2001] DePristo, M. and Zubek, R. (2001). being-in-the-world. In *Proc. of the 2001 AAAI Spring Symp. on AI and Interactive Entertainment*.

[Gabsdil et al., 2001] Gabsdil, M., Koller, A., and Striegnitz, K. (2001). Building a text adventure on description logic. In *Proceedings of KI-2001 Workshop on Applications of Description Logics*, Vienna.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: a survey. *JAIR*, 4:237–285.

[Koller et al., 2002] Koller, A., Debusmann, R., Gabsdil, M., and Striegnitz, K. (2002). Put my galakmid coin into the dispenser and kick it: Computational linguistics and theorem proving in a computer game. Journal of Language & Computation. To appear.