FOL in an Action Game

Josh Brown and **Ben Scott**

Abstract

We have implemented a learning agent that navigates a virtual city with the goal to seek out and destroy other entities deemed enemies to it using First Order Logic. The city will be composed of a discrete 2D grid. Each node either contains a building, an agent, or is free.

Introduction

All real-time action games today are dependent on AI to allow the computer to compete with a human player. Until recently, however, there has not been enough CPU time in a real-time application to spare to the AI system to allow games to have complex AI. As such, AI routines have been hard-coded to work well when certain states are reached. Unfortunately, the computer is then not able to adapt to the choices made by the human player, severely hindering the playability of the game.

Only recently have academic AI methods attracted the interest of game developers. Specifically neural networks have been popular lately. However, such a system does not account for the accumulation of knowledge that an agent would need in order to truly adapt the the player. We have decided to investigate the use of First Order Logic (FOL) knowledge bases in the creation of agents that can infer information about the environment and make logical decisions on actions to take given that knowledge. It is our hope that this will lead to creation of agents that can operate independently and logically in a game environment.

Rules of the Game

In order to test our inference agents, we have created a synthetic environment in which agents can navigate about and search each other out. We start by giving the PAGE description of the simulation we have created.

Percepts

The agent receives one percept at each turn: vision.

- The visual percept consists of the relative location and type of object in each cell it can see. The agent has a 180^o field of view limited to a distance of 4 squares.

Actions

The agent can choose between five actions and may only effect one each turn.

- Go one square forward
- Go one square backward
- Turn 90^o to the left
- Turn 90^o to the right
- Shoot weapon

Goal

The agent's goal is to survive long enough to be the last agent left alive in the city. Its sub goal is to kill all other agents in the game.

Environment

The environment is a city broken up into a discrete 2D grid. Each cell is either a street (and thus traversable by an agent) or a building (and thus impassable). The agent may be in any street cell at any time provided that no other agent is already occupying that square. If two or more agents attempt to enter the same square at the same time, none are allowed to enter that square causing nobody to move for that turn.

Implementation

Implementation of the test harness and knowledge base was done in standard C++ for portability.

Development of the Test Harness

The test harness for this game is a 3D depiction of the 2D environment. The city is represented by a grid in which each may be occupied by a building, an agent, or nothing. The game can be played in 3 configurations: human vs. computer with a top-down view, human vs. computer with a "chase camera" view, and computer vs. computer with a top-down view. We used SDL as our base system interface in order to allow the application to open a window suitable for OpenGL in a cross-platform manner. OpenGL was used as the low-level graphics library for performance and system independence.

Development of the FOL Knowledge Base

The first task toward a working KB was to write a parser for the language so that we could read in sentences as strings from a file. We chose to use ANTLR to generate a lexer and parser for FOL. This turned out to be harder than we initially anticipated since a full grammar for FOL is context-sensitive. Since ANTLR and most other parser generators can only handle context-free grammars, we had to modify the language a bit to make it context-free. FOL is an extremely expressive language. Implementing the entirety of the FOL language is beyond the scope of this project. Due to time constraints, we decided to limit ourselves to using a subset of FOL that applied to our needs. As such, we removed support for equality and existential sentences from our subset of FOL.

Once the parser was in order, we tackled the construction of the knowledge base itself. The KB is implemented as a naive, unordered list of sentences. Backward chaining using Modus Ponens is the only inference method used at this time. Given this configuration for the KB, it is only natural that it only supports sentences in Horn Normal Form.

Results

Not surprisingly, implementing the naive algorithms for doing backward chaining and unification put a huge performance hit on the application. While this was expected, we did not think it would overtly affect the test harness enough that it could no longer run in real time. As the knowledge base grew the search space exploded causing us to be able to infer little more than one action every 10 seconds. We were forced to make the test environment much smaller and reduce the number of complex sentences from our knowledge base. This has made us seriously reconsider the use of a knowledge base in real-time applications such as games.

Despite performance issues, we were relatively happy with the logical choices made by the agents as they moved about their environment. However, even with mutations or other methods that would add to the knowledge base, we feel that the performance of this technique would not match the performance of current AI techniques used in games such as neural networks or state machines.

Future work

Mutation

We would like to increased the speed of the learning process for the agent by pitting it against itself over and over again. We would use genetic algorithm techniques on the search and attack algorithms in order to speed up the learning process. We have already started work on this but unfortunately did not have enough time to complete it.

Performance

Obviously the performance of the knowledge base engine was a disappointment to us. We would like to spend some time reworking the algorithms used in order to make FOL knowledge bases feasible for real-time applications.