COMP6006: Intelligent Agents Report

The BeadyEye Agent

Rikki Prince
University of Southampton
Highfield
Southampton, UK
rfp102@soton.ac.uk

ABSTRACT
This is the abstract of my report.

1. INTRODUCTION
This report describes the approach taken in creating a software agent to compete in a TAC Classic contest hosted at the University of Southampton for the module COMP6006.

1.1 TAC
The Trading Agent Competition (TAC) is an annual contest organised by, and for, the agent research community, since 2002. The hope is that by fostering a competetive spirit, the advancement of the field will be accelerated. The competition current consists of two scenarios: TAC Classic and TAC SCM [**?**].

It is the TAC Classic scenario which was faced in this challenge. This scenario involves acting as a "travel agent", to create holiday packages to suit the requirements of a group of eight customers. A holiday package consists of inbound and outbound flights, a hotel room for each night of the stay and tickets to entertainment venues. Further complexity is added by the fact that there are two hotels to choose from and three entertainment venues. In total, the agent is required to co-ordinate bidding strategies in up to 28 different auctions.

The auctions also vary in format. The flight auctions clear instantly, but have a sell price which varies every 10 seconds, based on a stochastic function. An auction is held for each night, in each hotel, though only 16 rooms are available per hotel. Therefore the auction is an ascending-price, multi-unit English auction. The quirk is that one hotel auction closes every minute, in a randomly selected order. Finally, entertainment auctions allow agents to buy and sell tickets from each other once they negotiate a price [**?**].

Bidding in auctions is performed by sending bid strings to the TAC server, containing the number of items required and the price the agent is willing to pay. Fortunately, the process of submitting bids and retrieving information about the clients' preferences and current state of the auctions is made slightly easier by the TAC Classic Java Agent Ware [**?**].

At the end of a TAC game, the competing agents are ranked against each other based on how well they fulfilled their clients' requirements. The measure of fulfillment is known as the utility, and is based upon how close the package is to the clients' preferred start and end dates, as well as whether the package matches the quality of hotel the client wanted and includes their favoured entertainment. The final score for the agent is the total utility, minus the costs of purchasing the various items which make up the packages.

1.2 The running of the Soton contest
The contest held for the COMP6006 module ran over two days. Entrants competed on one of the two days, against up to 25 others. However, as each game only allows for eight competitors at a time, the two days were arranged so that each agent would compete in around 10-15 games, against a different group of opponents in each game.

2. APPROACH
The initial aproach taken was to better encapsulate the data available into Java objects, so that programming the decision making aspects of the code were more intuitive to write and manipulate. Agent Ware provides access to client preferences with lines of code such as:

```
agent.getClientPreference(c, TACAgent.ARRIVAL);
```

Once the agent logic becomes more complex, calling this line of code just to find a client's preferred arrival day could make the code complicated to read. The intended alternative is to create an object for each client, which provides a method for requesting the arrival day, as well as all other preferences, for example:

```
client[c].start();
```

The added advantage of this approach is that certain decisions and calculations can be performed within the confines

of the object, rather than in some arbitrary place within the agent. For instance, one of the important tasks that can be processed within the client object is to allocate hotel rooms, flights and entertainment tickets that have been won, and therefore provisionally calculate the utility and cost of a particular client's package.

As well as the client, other concepts within the TAC contest that can be compartmentalised into an object are the flight, hotel and entertainment auctions.

3. DESIGN

As discussed above, the approach taken was to construct objects for storing the available data in a more logical format. This section describes the design of the classes that store and process this data. These classes work on top of the Agent Ware code, the design of which is not examined here.

3.1 Client

The primary abstraction from the data provided by the Agent Ware is that of the client. The client is the main focus of the agent's efforts, so it makes sense to encapsulate some of the decision-making functionality of the agent within a client object.

Within this agent, the Client had to be designed to store both the original client preferences, and any updated preferences that result from changes within the game environment. Once a hotel auction closes, if the agent did not win the required number of rooms for that night, the client's preferred package could become invalid, and hence the requirements of that client would have to change. Additionally, the Client must keep track of any rooms, flights or entertainment tickets which have been allocated to it, and how much they cost to purchase.

3.2 Flights

4. HEURISTICS

5. RESULTS

6. EVALUATION

7. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

8. ACKNOWLEDGMENTS

PV for help.