Learning Through Performance Feedback from Computer Simulation

Robin Broad
MSc Computing Science
Newcastle University

August 2008
Tutor: Dr. Christian Kray

[LITE-VERSION]
(The LITE version is for general viewing with some technical sections removed)

Abstract: This dissertation details the work carried out by Robin Broad during June, July and August 2008 on a project to build a simulation game based upon a search and rescue scenario with an emphasis placed on the processing of free text input and the feeding back of useful comments on the trainee's performance at the end of the simulation.

The software was written using Java and the tuProlog Java implemetation of Prolog created at the Alma Mater Studiorum, of Universit`a di Bologna a Cesena, Italy. The simulation was written for the cadets of the 346 (Tynemouth) Squadron Air Training Corps in particular and it was evaluated and tested by them.

[Word Count: 19,924]

Declaration

I declare that except where otherwise stated, this project and dissertation is my own work.

Robin Broad

August 2008

Acknowledgements

The writing of the software used in this project would not have been possible if it were not for the excellent standard of teaching provided by the staff in the department of Computing Science at the University of Newcastle upon Tyne together with the outstanding facilities, resources (especially the range and sheer number of computing science books) and the staff of the Robinson Library who work late in the evenings and over the weekends to provide a service to both the staff and students of the university.

I would like to thank Dr. Christian Kray, my project supervisor, for his support, and in particular his guidance with the software evaluation, Chris Young of Netskills who really equipped me during the course to do all of the XHTML, CSS and PHP work, Dr. Graham Morgan who got me started on my very first website and taught me how to use the Linux/UNIX commands, Prof. Isi Mitrani and Dr. Nick Cook who taught me Java, Dr. Marta Pietkiewicz-Koutny who taught me concurrent programming which was used extensively in this project, Dr. Chris Holt who supported me with my initial ideas and encouraged me to learn Prolog, Dr. Cristina Gacek for her support throughout the project and Prof. Pete Lee for his guidance on completing the project and dissertation. I would also like to thank the Faculty Liaison Librarian, Moira Bent, for explaining the use of the library, referencing and the searching of digital journals and e-books, and Ms. Shirley Craig, the Computing Science Librarian for her expert advice. Finally, I would like to thank my night-class English teacher, Liz Purvis whose encouragement and enthusiasm has generated for me, a renewed interest in the English language, which has helped in many ways, including the writing of this dissertation.

Good use has been made of open source software during this project. This includes the Standard Java Development Kit and the Java Speech API of Sun Microsystems, the tuProlog IDE Java implementation of Prolog created at the Alma Mater Studiorum, of Universit`a di Bologna a Cesena, Italy, a Java MP3 playing class from the Computing Science Department at Princeton and the ROT13 cipher class written by Jeffrey M. Hunter.

I am grateful to Flight Lieutenant William Close-Ash and the cadets of the 346 (Tynemouth) Squadron Air Training Corps for their input with initial ideas during the early stages of development of the game and for the valuable time spent with them in evaluating the beta 1.0 release of the software.

Special thanks are due to the staff of the Scottish Borders Library Service and Mrs ---- who assisted with Internet access in the Scottish Borders. Thanks are also due to Mr ---- who allowed me to use his Apple Mac mini to create the Mac compatible version of the software.

Finally, I would like to thank my wife, ---- who provided financial support throughout the year. I would also like to thank ----, and my son ---- for their patience and understanding during the very many hours spent in writing not only this dissertation but also in developing the related software, documentation and web resources associated with this project. ---- has taken an interest in the development of the game and as an experienced games player, his opinions have been valued.

Contents

Declaration

Acknowledgements

1 Introduction Page

1.1 What the project is about ……… 5
1.2 The reasons for this choice ……… 5
1.3 Expected achievements ……… 6
1.4 Approach to the problem ……… 6
1.5 Aims ……… 7
1.6 Objectives ……… 7

2 Background Research

2.1 Literature review

(i) A brief history of AI ……… 8
(ii) Logic ……… 8
(iii) Game Playing Machines ……… 12
(iv) Natural Language Processing ……… 12
(v) Expert Systems, CLIPS and Prolog ……… 13
(vi) Training simulation and search & rescue ……… 14

3 Building the Simulation Game

3.1 Approach taken

3.1.1 ---- Family Tree ……… 16
3.1.2 Fox-Duck Adventure Game ……… 17
3.1.3 Design ……… 19
3.1.4 Developing the tuProlog Solution ……… [not in LITE]
3.1.5 Developing the Speech Synthesis ……… [not in LITE]
3.1.6 Developing the GUI and Concurrency Solutions ……… [not in LITE]
3.1.7 Developing the Web Based Resources ……… [not in LITE]

3.2 High-level explanation

3.2.1 Prolog: RadioHam4 ……… [not in LITE]

3.2.2 Java Classes

(i) Java Classes Overview ……… 23
(ii) Free Text Processing ……… [not in LITE]
(iii) Speech Synthesis ……… 24
3.2.3 HTML and CSS ……… 26
3.3 Testing ……… 27
3.4 User Manual ……… 29

4 Results and Evaluation Page

4.1Presenting the results………………………………………………….... 32
4.2Discussion……………………………………………………………. 38
4.3 Evaluation of the approach…………………………………………….. 39

5 Conclusions

5.1 Meeting the original objectives………………………………………… 41
5.2 Positive and negative aspects…………………………………………… 42
5.3 Future work……………………………………………………………... 43

6 References

6.1 Books……………………………………………………………………. 45
6.2 Papers and Journals……………………………………………………… 46
6.3 Websites…………………………………………………………………. 46

7 Appendices

7.1 Planning Gantt Chart…………………………………………………….. 47
7.2 CLIPS Quiz Listing…………………………………………….. [not in LITE]
7.3 Prolog Program Listings
    7.3.1 ---- Family Tree……………………………………… [not in LITE]
    7.3.2 Fox-Duck Adventure Game………………………………………… 49
    7.3.3 RadioHam4……………………………………………… [not in LITE]
7.4 Java Class Listings
    7.4.1 BabelFish………………………………………………… [not in LITE]
    7.4.2 BinarySemaphore………………………………………… [not in LITE]
    7.4.3 BistableClock…………………………………………….. [not in LITE]
    7.4.4 GameMinsCounter……………………………………….. [not in LITE]
    7.4.5 Monostable……………………………………………….. [not in LITE]
    7.4.6 MP3……………………………………………………… [not in LITE]
    7.4.7 RMBGameTalk2…………………………………………. [not in LITE]
    7.4.8 RMBGameWindow2……………………………………... [not in LITE]
    7.4.9 ROT13…………………………………………………….. [not in LITE]
    7.4.10 Say2…………………………………………………… [not in LITE]
    7.4.11 Semaphore……………………………………………….. [not in LITE]
    7.4.12 TeaBoy………………………………………………… [not in LITE]
7.5 HTML and CSS
    7.5.1 Search & Rescue Web Page……………………………….. [not in LITE]
    7.5.2 HTML Instructions……………………………………… [not in LITE]
    7.5.3 Web Form Survey………………………………………….. [not in LITE]
    7.5.4 Web Survey PHP Script…………………………………….. [not in LITE]
7.6 A beginner’s two game dialogue………………………………………… 50

1 Introduction

1.1 What the project is about

One of the problems faced by teachers is that of providing sufficiently detailed individual and useful feedback to students. If a typical classroom teacher were to speak to every student individually during a lesson, then they would be able to afford to spend at most two minutes with each student. In any case, this situation is very unlikely to arise in practice and the teacher will, more often than not, be addressing the class as a whole. Even following a test, it is difficult for the teacher to feedback anything more than an individual mark to each student.

There is potential however to make use of the extensive provision of computers in schools in the UK as tools to improve the students' experience of constructive feedback, not only from the teacher but augmented by machines. These machines would be available to give feedback on a more regular basis, or even on demand. This would be an improvement on the current situation of an occasional one-to-one conversation with the teacher, a comment written on some work, a test score or waiting for an end of term (or even an end of year) report.

This project is about providing individual and detailed performance feedback to the user of a piece of software, which takes the form of a search and rescue simulation game. The theme of search and rescue is used only as a vehicle upon which to build the infrastructure of an AI based software application, which will be both intellectually challenging and fun to play.

In their article: *Serious games for serious topics,* Quinn & Neal expound the virtues of game play in education:

"Properly done, serious games are highly effective for serious topics. While the notion of a game may seem frivolous, the design and content are not. In fact, a serious game can introduce tension and crises to simulate the realistic experience of practicing a particular skill, or depict consequences, more easily than other types of learning. In the game *Darfur Is Dying*, for example, the difficulties of daily life such as trying to secure water while avoiding the Janjaweed militias is challenging and nerve-wracking, bringing home the simple challenges of survival for the people caught in this disaster and raising awareness of the importance of a solution." [Quinn & Neal08]

1.2 The reasons for this choice

There are many advantages for both teachers and students in using computer-based simulation as part of the learning process. It is often not possible for a teacher to give students "hands on" experience of subjects such as "the generation of electricity using nuclear power" or "experiments in particle physics" but computer simulation offers a way forward in providing firsthand experience in an otherwise inaccessible field.

Although live demonstrations are of course vital, there are situations where such demonstrations are either very difficult or impossible to provide. In this case, computer simulation, as is increasingly the case, may be used as an alternative. Blake & Scanlon have pointed out in their work on *simulations in science education* that:

"…they (simulations) free up teacher time so that they can interact with students instead of dealing with the management of the experimental set-up/apparatus and supervision." [Blake & Scanlon07] (However, they neglect to mention the technical skill needed in actually running some of the more sophisticated computer simulations).

Not only are there benefits for the teacher, but of course for the students as well. They also point out how the use of multimedia has been shown to bring with it its own benefits:

"An additional advantage of simulations is presenting a variety of representational formats including diagrams, graphics, animations, sound and video that can facilitate understanding (Ainsworth & van Labeke 2002)." .[Blake & Scanlon07]

1.3 Expected achievements

One of the primary goals of this project is to produce a software application using a declarative programming language, rather than the procedural paradigm of Java. This should assist in making logical deductions about a situation, which can be compared with the actions of the game player, so that inferences can be made about the game player's performance. These inferences will be logged during the game play and returned to at the end of the game whereupon the machine will be able to comment, in the form of straightforward statements, about the player's performance during the game.

Furthermore, it is anticipated that this declarative language will be able to be integrated with Java so that a portable application, capable of running on any machine that supports the Java Virtual Machine, can either be downloaded from the Internet and executed as a standalone application or, be run as a Java applet within a browser.

1.4 Approach to the problem

One of the difficulties in creating an initial system design for this project was that the techniques for declarative programming were not well understood at the start of the design phase, and these skills would be developed during the course of the project. A decision was made fairly early on in the project to use Prolog for the core program of the search and rescue game. Prolog was chosen because it seemed to be the most mainstream, widely supported and well used of the declarative programming languages researched which included CLIPS, LISP and Prolog.

When considering an appropriate software process model, the waterfall model was thought to be unsuitable, since without a good understanding of Prolog it was not possible to come up with a reasonably complete system design in the early stages. It was decided therefore that an evolutionary prototyping model would be the most appropriate in this situation. In this way, an initial small, but working, prototype Prolog program could be gradually evolved into something more complex as the project proceeded.

However, software development using Java was well understood and it was possible to draft an overall system diagram, which included some of the anticipated Java components that would be required (see section 3.1.3).

1.5 Aims

To create a software application that uses a declarative programming language to follow a route through a problem under the guidance of the user and gives feedback on their performance in a summary report. This report should help the user to learn something new about the problem through experience.

1.6 Objectives

- To find out about, and make use of, one or more programming languages that are related to logic programming and AI. For example Prolog, LISP, CLIPS, Jess etc.
- To learn more about problem solving, predicate logic and reasoning with uncertainty.
- To research the use of performance feedback in currently available educational software.
- To produce a feature that will *explain* to the user the reasoning process that was used to achieve the solution in an attempt to help the user understand the steps needed in solving a particular problem.
- To use the problem domain of a *search and rescue training scenario* as a framework within which the AI logic programming can operate.
- To write and test a suitably robust and intuitive program which will address the stated aims.
- To evaluate the usefulness and usability of such a system.

2 Background Research

2.1 Literature review

(i) A brief history of AI

Artificial intelligence is a branch of computer science, which is concerned, with the building of machines (electronic computers at the present time) that are capable of solving complex problems in a manner similar to that used by human beings. Although the first electronic computer (the Atanasoff–Berry Computer, built in the United States in 1942), was built less than 70 years ago, the study of formal logic has been pursued for over 2000 years and Boolean algebra (a branch of mathematical logic) has been understood since the early 19th century. In their book *Artificial Intelligence: A Modern Approach*, Russel & Norvig refer to the contribution made by Aristotle:

"The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking" that is, irrefutable reasoning processes. His syllogisms provided patterns for argument structures that always yielded correct conclusions when given correct premises- for example, "Socrates is a man; all men are mortal; therefore, Socrates is mortal." these laws of thought were supposed to govern the operation of the mind; their study initiated the field called logic." [Russel & Norvig03, p.4].

There has been much philosophical debate over the last half century surrounding the issue of machines and intelligence. On one hand, proponents of strong AI advocate that machines one day will match, or even exceed, the intelligence of humans whereas the advocates of weak AI argue that machines cannot think in the same sense that human beings can. The philosopher John Searle is well known for his Chinese room argument, which seeks to refute strong AI. Searle's main objection is to any notion that a machine can be said to have a mind or consciousness. These are matters for philosophical debate and will not be pursued here.

(ii) Logic

One of the most interesting areas of study relating to the understanding of artificial intelligence in machines has been the study of Formal Logic and in particular Predicate Logic. The study of logic is ancient, dating back to the time of the ancient Greeks, thousands of years before the invention of the first electronic computer. Before 300 BC, Aristotle was interested in the truth of statements, the application of rules and the conclusions, which could be deduced as a result. He developed the concept of the *syllogism* where a conclusion could be drawn from two earlier statements known as *premises*. Inferring a conclusion from two connected statements is known as *logical deduction* and may be regarded as a natural consequence of intelligent human thought. An example of an Aristotelian syllogism is shown in Figure 2.1(c). This will be explained once some of the symbols used in Predicate Logic have been introduced.

Logic can of course be used to support the validity of an argument. Bergmann, Moor & Nelson define the term "argument" as follows:
"An argument is a set of sentences one of which (the conclusion) is taken to be supported by the remaining sentences the premises." [Bergmann, Moor & Nelson98, p.7].

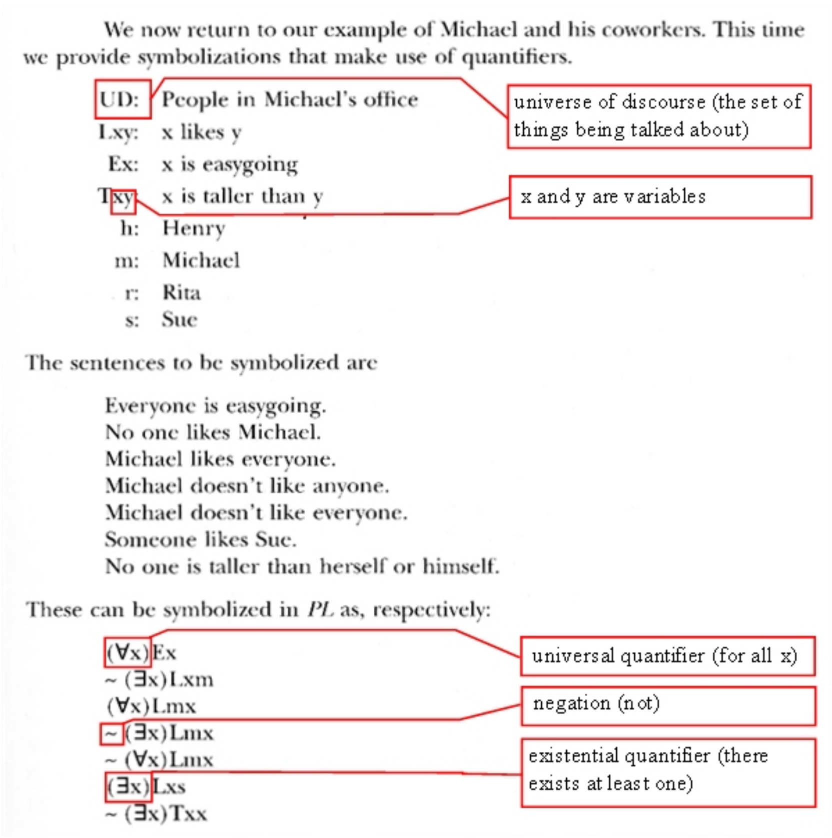
Under Aristotelian logic, arguments can be scrutinised to assess their validity. In an invalid argument, the conclusion would not be connected to the premises in any logical fashion and the argument could then be demonstrated to be false. Similarly, the conclusions reached from

valid arguments can be demonstrated to be true. This is concisely defined here by Bergmann, Moor & Nelson:
"An argument is deductively valid if and only if it is not possible for the premises to be true and the conclusion false. An argument is deductively invalid if and only if it is not deductively valid." [Bergmann, Moor & Nelson98, p.11].

Since the Ancient Greeks, logic has been developed by various civilisations throughout the world and was of particular interest to the philosophers of the middle ages. Running in parallel with the development of formal logic has been the development of mathematical logic and Boolean algebra in particular. In Boolean algebra variables can be assigned the values *true* or *false* and can be manipulated with the logical operators *and*, *or* and *not*, to name a few. Bergmann, Moor & Nelson proceed to explain the use of symbols in predicate logic, which is a more advanced form of logic than the Aristotelian syllogisms already outlined. In Figure 2.1(a) algebraic symbols are used to stand for objects (someone or something), in this case people. The symbols *x* and *y* are known as variables and are able to stand for any person or any thing. The concepts of the universal quantifier (meaning all things) and the existential quantifier (meaning that at least one thing exists) are also introduced, together with the negation (not) symbol.

Although Bergmann, Moor & Nelson's book is classified as a work in philosophy, its contents are extremely closely related to the techniques used by declarative programming languages and indeed, it underpins the operation of Prolog.

It is interesting to note that predicate logic declares a "universe of discourse" (in this case, the people in Michael's office). This limits the scope of the application of the arguments, an idea that will be returned to later, in the context of the realistic implementation of artificial intelligence in machines.



We now return to our example of Michael and his coworkers. This time we provide symbolizations that make use of quantifiers.

UD: People in Michael's office
Lxy: x likes y
Ex: x is easygoing
Txy: x is taller than y
h: Henry
m: Michael
r: Rita
s: Sue

universe of discourse (the set of things being talked about)

x and y are variables

The sentences to be symbolized are

Everyone is easygoing.
No one likes Michael.
Michael likes everyone.
Michael doesn't like anyone.
Michael doesn't like everyone.
Someone likes Sue.
No one is taller than herself or himself.

These can be symbolized in *PL* as, respectively:

$(\forall x)Ex$
$\sim(\exists x)Lxm$
$(\forall x)Lmx$
$\sim(\exists x)Lmx$
$\sim(\forall x)Lmx$
$(\exists x)Lxs$
$\sim(\exists x)Txx$

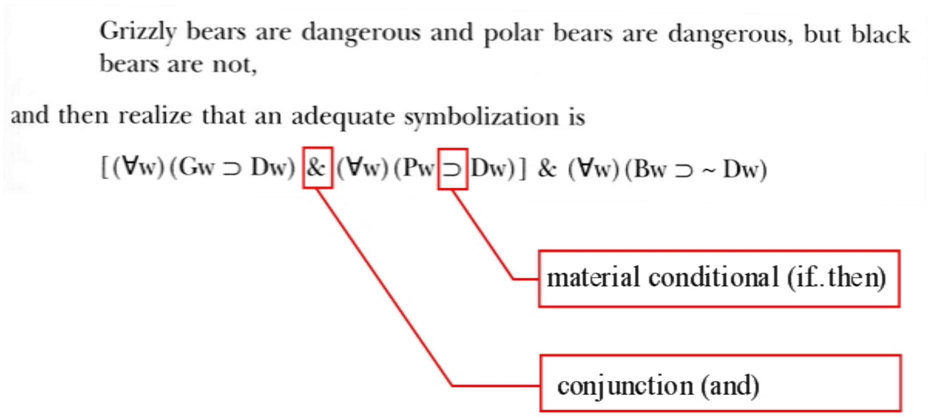
universal quantifier (for all x)

negation (not)

existential quantifier (there exists at least one)

Figure 2.1(a) Symbolization in Predicate Logic, adapted from [Bergmann, Moor & Nelson98, p.255].

As any computer programmer will be aware, the operators *and*, and *or* are extremely useful and important, together with the *if-then* statement. Predicate logic uses these too and their symbols are shown in Figure 2.1(b), where a statement about the danger of bears has been reduced to a symbolic representation in predicate logic.



Grizzly bears are dangerous and polar bears are dangerous, but black bears are not,

and then realize that an adequate symbolization is

$$[(\forall w)(Gw \supset Dw)\ \&\ (\forall w)(Pw \supset Dw)]\ \&\ (\forall w)(Bw \supset {\sim} Dw)$$

material conditional (if..then)

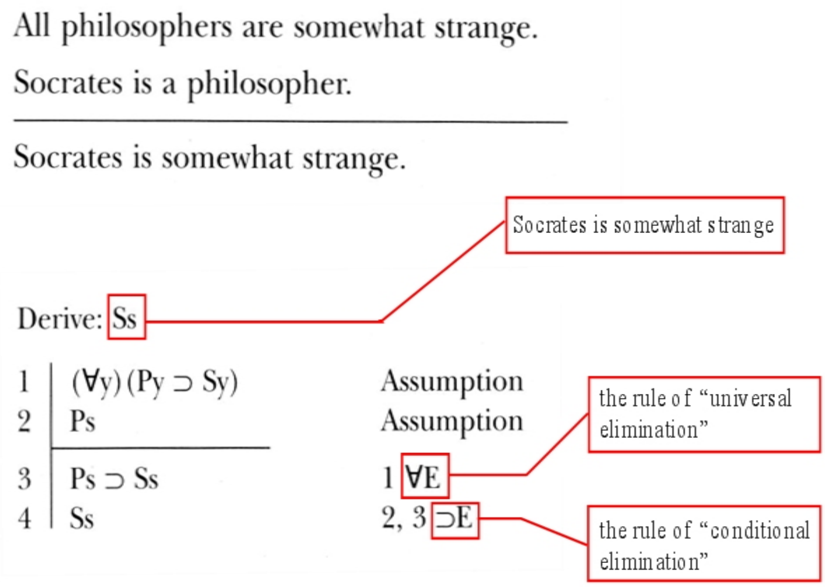
conjunction (and)

Figure 2.1(b) Further Symbolization in Predicate Logic, adapted from [Bergmann, Moor & Nelson98, p.291].

Figure 2.1(c) shows perhaps the most exciting aspect of predicate logic outlined so far. If one accepts, as stated earlier, that making logical deductions is a natural consequence of intelligent human thought, then the ability to derive a new statement (which is a logical consequence of the earlier statements) only by the *mathematical manipulation of symbols,* clearly has important implications for the implementation of intelligent behaviour in machines.

Looking at the Aristotelian syllogism at the top of the figure, the conclusion "Socrates is somewhat strange" may seem obvious to us, but what is not so obvious is the symbolic derivation which follows: The two premises have been coded into symbols as described earlier, and on line 3 we see the application of an elimination rule known as the rule of "universal elimination". Here we move from the *general (universal) claim* shown on line 1 (that all things that can be identified as being a philosopher must therefore be strange), to the *specific instance* of Socrates. Line 3 follows not only from line 1, but also from line 2 which states that Socrates is indeed a philosopher (this must be true in order to deduce that he is strange).

On line 4, the rule of "conditional elimination" (Modus Ponens) is used to assert the fact that *Socrates is strange* as a true statement in its own right. This process of deduction and assertion is something that Prolog is quite capable of.



All philosophers are somewhat strange.
Socrates is a philosopher.

Socrates is somewhat strange.

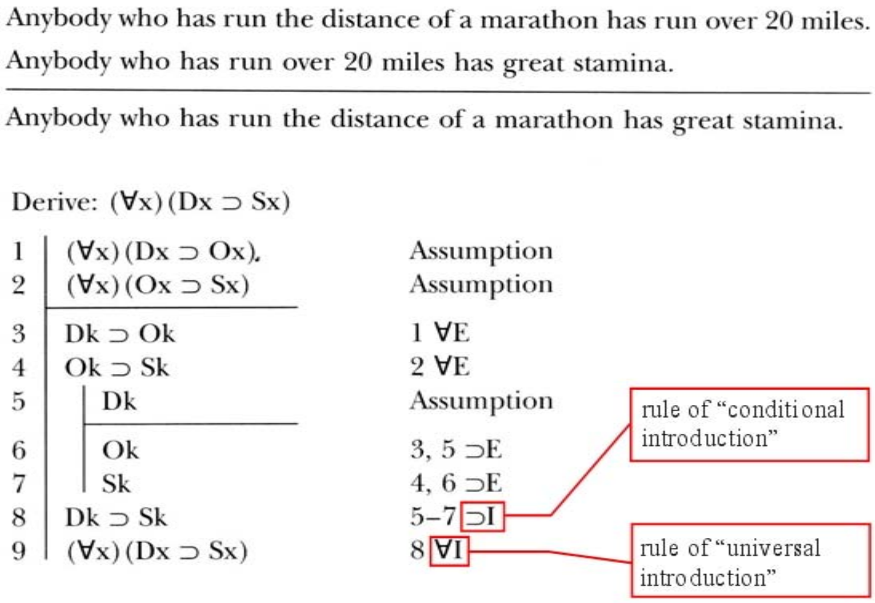
Derive: Ss

1 | $(\forall y)(Py \supset Sy)$ Assumption
2 | Ps Assumption
3 | $Ps \supset Ss$ 1 $\forall$E
4 | Ss 2, 3 $\supset$E

Socrates is somewhat strange
the rule of “universal elimination”
the rule of “conditional elimination”

Figure 2.1(c) Derivation in Predicate Logic, adapted from [Bergmann, Moor & Nelson98, p.444].

A final example of the application of predicate logic is that of the rules of *conditional* and *universal introduction*, shown in Figure 2.1(d). Line 8 is interesting because a new rule is being introduced. This new rule has been derived as a consequence of the two rules that form the premises of this argument. Instead of asserting a new fact, as in the previous case, this time a new rule has been asserted in a process known as the *rule of conditional introduction*. On line 9, the deduced rule for *k* has been generalised for any instance *x* by the rule of *universal introduction*. This was possible since no specific information about object *k* was required in deriving the rule on line 8.



Anybody who has run the distance of a marathon has run over 20 miles.
Anybody who has run over 20 miles has great stamina.

Anybody who has run the distance of a marathon has great stamina.

Derive: $(\forall x)(Dx \supset Sx)$

1 | $(\forall x)(Dx \supset Ox)$ Assumption
2 | $(\forall x)(Ox \supset Sx)$ Assumption
3 | $Dk \supset Ok$ 1 $\forall$E
4 | $Ok \supset Sk$ 2 $\forall$E
5 | | Dk Assumption
6 | | Ok 3, 5 $\supset$E
7 | | Sk 4, 6 $\supset$E
8 | $Dk \supset Sk$ 5–7 $\supset$I
9 | $(\forall x)(Dx \supset Sx)$ 8 $\forall$I

rule of “conditional introduction”
rule of “universal introduction”

Figure 2.1(d) Universal Introduction in Predicate Logic, adapted from [Bergmann, Moor & Nelson98, p.448].

Bergmann, Moor & Nelson give us the following assurance about the confidence that can be placed in the results of this type of derivation:

"The derivation rules of PD(Predicate Derivations), like the derivation rules of SD(Sentential Derivations), are truth preserving. Given the semantics developed for PL(Predicate Logic), the derivation rules of PD never lead us from true sentences to false ones." [Bergmann, Moor & Nelson98, p.443].

(iii) Game Playing Machines

In 1997, IBM succeeded in a landmark achievement for AI when their supercomputer *Deep Blue* beat the reigning world chess champion Garry Kasparov in a six-match chess tournament, which observed the three-minute time control for competition chess. The victory for IBM was no landslide. Kasparov drew three games and won one, which is in itself a remarkable achievement considering the incredible computing power of *Deep Blue*. According to IBM's website, Deep Blue was capable of analysing 200 million moves per-second or, a staggering total of 50 billion positions during the three-minute time allowance for each player. It is interesting to note that according to Teuscher, *Deep Blue* utilised a chess-playing algorithm, which was originally conceived by Alan Turing:

"Turing and David Champernowne wrote the Turochamp chess program in 1948, applying a search strategy known as *Minimax* towards choosing the next move. Eventually computers were built that could beat human chess champions, but it took 40 years longer than Turing predicted. Using a variant of *Minimax* known as the *alpha-beta search*, a supercomputer named *Deep Blue* beat the world chess champion Gary Kasparov in 1997." [Teuscher04, p.168].

Some games playing machines are capable of learning through experience and improving as they play. As early as the 1960’s, a checkers playing program written by Arthur Samuel had beaten a checkers champion; Robert Nearly. Again, this program had been developed by IBM as part of a research project into machine learning and although there is some controversy about the actual checkers playing standard of Robert Nearly, a *state* champion rather than a national or international champion, it is clear from this extract that he found this self-improving game playing machine quite a challenging opponent:

"Self-improving program: As a result of learning from its experience, the program carried on improving quickly and soon beat a checkers champion. The champion remarked after defeat in 1965... "In the matter of the end game, I have not had such competition from any human being since 1954, when I lost my last game."" [Brighton & Selina03, p.79].

The games playing record for machines is clearly impressive, but does this really constitute what we would regard as intelligence? According to Brighton & Selina IBM regard *Deep Blue to* be a ‘stupid’ machine, a statement far removed from Allan Turing’s view that machines could ‘think’ and that they could be said to possess intelligence:

"Deep Blue is stunningly effective at solving chess problems, but it is less 'intelligent' than even the stupidest human. - The IBM Deep Blue website." [Brighton & Selina03, p.85].

(iv) Natural Language Processing

How far have researchers come in reaching the goal of a truly intelligent machine that is able to communicate using natural language? Important progress was made by Minsky et al. with their *blocks world* program [Russel & Norvig03, p.20]. One facet of this project, which is of particular relevance, was Terry Winograd's work on the SHRDLU system of natural language processing. Russell and Norvig explain that part of Winograd's success was that the system only needed to deal with a restricted range of vocabulary relating to the limited domain of one specialized subject.

Working only within the realm of the *blocks world* would usefully restrict the range of commands that the system would be expected to interpret. Without this restriction, the task of comprehending natural language becomes immense. As King has noted, we in fact end up with an infinite number of possible utterances to deal with:

"We all speak a language. Only physical limitations like tiredness, etc can stop us from inventing sentences in that language. Each of those sentences may be different from all the ones thought of before and we can go on expressing new sentences quasi-forever. All this means is that in a language there are infinitely numerous different sentences." [King83, p.3].

King goes on to explain the importance and function of *parsers* in natural language processing. He speaks about *recognisers*, which are capable of detecting the difference between correctly formed and spelled grammatical sentences of a language and incorrectly formed sentences. Parsers also produce an output i.e. they are working as *transducer*s. King's explanation of a parser is as follows:

"What is needed is a procedure that will, this time, not just recognise the sentence but also discover how it is built. The execution of that procedure is called parsing and the thing that executes it is called a parser. Therefore, parsers do essentially two things. On the one hand, when presented with the string, they have to recognise it as a sentence of the language they can parse. In this respect, parsers have built-in recognisers. On the other hand, they have to assign to that sentence a structure, which they have to output... Parsers belong to the type of objects called transducers- i.e. in simple terms recognisers augmented with output facilities." [King83, p.8].

(v) Expert Systems, CLIPS and Prolog

In the 1970s and eighties a lot of excitement and interest existed around the subject of *expert systems* which are still in use today. They are useful in the areas of specialist medical diagnosis and training, in oil prospecting and mining. Experts systems are carefully programmed with facts and rules, which have been elicited from experts from within a specialised field. Once successfully constructed and tested, the expert system software could be rolled out across a variety of locations, effectively spreading out the knowledge and skills of the original experts at a reduced cost. In their book *Expert Systems,* Giarratano & Riley list some further advantages of experts systems:

"Advantages of experts systems: increased availability, reduced cost, reduced danger, permanence, multiple expertise, increased reliability, explanation, fast response, steady unemotional and complete response at all times, intelligent tutor, intelligent database." [Giarratano & Riley05, p.8]

It is interesting to note that they explain how an expert system may be used as an intelligent tutor. This is of particular relevance to this project, which aims to give performance feedback to the player at the end of the game and which, is an action, normally expected from an intelligent human tutor. Giarratano & Riley make reference to the term *explanation facility*

"Explanation: the expert system can explain in detail the reasoning that led to a conclusion. A human may be too tired, unwilling to, or unable to do this at the time. This increases the confidence that the correct decision is made." [Giarratano & Riley05, p.8].

Initially serious consideration was given to using the CLIPS (C Language Integrated Production System) expert system tool, developed by NASA at the Johnson Space Center and available as open source code. Indeed some time was spent learning how to use it and in creating some applications with it. One of these is shown in section 7.2 *CLIPS Quiz Listing.* This program declares a number of facts in the form of questions, which are then asked of the user. Information is stored about whether or not the correct answer was given, using the following CLIPS instruction:

```
(if (eq (lowcase ?answer) (lowcase ?answer-text)) then (bind ?response correct)(bind ?known
yes)else (bind ?response incorrect)(bind ?known no))
```

After asking all of the questions the program goes back to ask any questions that were answered incorrectly the first time. At the time the question was answered incorrectly, the correct answer was given by the machine. The user then needs to recall the correct answer in an attempt to reinforce their ability to remember the answers to the questions. The program would repeat this process until all of the questions had been answered correctly. This was found to be quite effective as a tool to aid recall.

Although CLIPS was first released in 1985, it has been adapted to be compliant with Java and Java implementations such as Jess and JClips have been produced. CLIPS is still supported by Gary Riley and a CLIPS user group exists on Google groups. Despite all of this, CLIPS appears to be waning in mainstream popularity while Prolog continues to have widespread support. Much of the research indicated that Prolog and LISP are still two of the most important AI languages used today.

In choosing a declarative programming language for this project, Luger gives a strong indication about the direction to take:

"Not only are these (LISP and Prolog) two of the most frequently used languages in artificial intelligence: their syntactic and semantic features also encourage powerful ways of thinking about problems and their solutions. The remarkable influence these languages have had on the historical development of AI is as much a product of their ability to function as "tools for thinking" as it is a reflection of their strengths as programming languages." [Luger05, p.635].

Giarratano & Riley also regard Prolog to be a “classic AI language” [Giarratano & Riley05, p.20] and Gazdar & Mellish refer to Prolog as being the “most well-known” of the AI languages [Gazdar & Mellish89, p.5]. Some reference was also found to the fact that LISP, although the leading AI language in the United States, often runs using dedicated hardware and does not run very efficiently on conventional machines [Giarratano & Riley05, p.45].

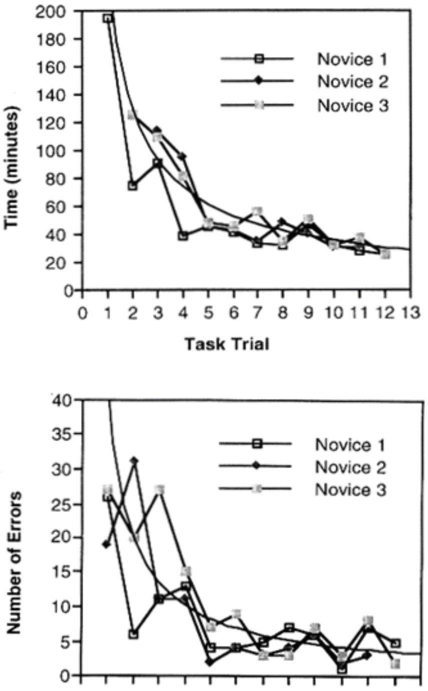
(vi) Training simulation and search & rescue

One of the stated objectives of this project was to use the problem domain of a *search and rescue training scenario* as a framework within which the AI logic programming can operate. There are of course advantages to exploring a scenario in simulation, whether it is a computer simulation or some other kind of simulation, for example role-play. In the case of search and rescue training, one advantage is that mistakes can be made in simulation that may cost lives in reality. Learning from experience within a simulated environment means that the trainee can be made aware of the consequences of their mistakes so that they may be better prepared when faced with the real situation. Indeed while speaking to Flight Lieutenant William Close-Ash during the early design stage of this project, he was quite sure that letting trainees experience the loss of life in simulation was justified if it meant that it reduced the risk of mistakes being made in a real life situation.

Another advantage of simulation is that situations, which rarely arise in reality, can be set up specifically for the purpose of training. They can also be used where reality is “too expensive, complex, dangerous, fast or slow” [Saunders, Coote & Crookall87, p.9]

Bloom & Bowen-Loftin describe the success of a training simulator built for Space Shuttle FDOs (Flight Dynamics Officers) at NASA mission control in Houston. It is worth looking at the improvements in performance following computer-aided simulation training as described here and illustrated in figure 2.1(e).

"An extensive investigation of the performance of novices using the system has been conducted. The figures show two measures of performance: (a) the time required to perform the normal task as a function of the number of training experiences and (b) the number of errors made during the performance of the normal task as a function of the number of training experiences." [Bloom & Bowen-Loftin98, p.220].



The time required to perform a nominal task as a function of the number of ICAT sessions completed by three novice flight controllers.

The number of errors committed by novice flight controllers performing a nominal task as a function of the number of ICAT sessions completed.

Figure 2.1(e) Improvements in Space Shuttle FDO Performance Following Computer Aided Simulation Training, adapted from [Bloom & Bowen-Loftin98, p.220].

In their paper *Emergency Response Simulation Using Wireless Sensor Networks,* Filippoupolitis et al. describe a range of search and rescue simulation programs, which include:

- Agent modelling on a sarin attack in Manhattan
- A simulation framework for emergency response drills.
- Robocup-Rescue simulator
- A game engine based simulation of the NIST urban search and rescue arenas

[Filippoupolitis08]

3 Building the Simulation Game

3.1 Approach taken

Since an evolutionary prototyping model was decided upon as a method of making progress with the unknown language of Prolog, then some of the steps in the evolution from simple working programs into a more complex final solution need to be explained. That is the purpose of this section, which leads up to the high-level explanation of the final solution in section 3.2.

3.1.1 ---- Family Tree

Both *forward chaining* and *backward chaining* are important systems of reasoning used in logical programming. These terms have been defined here by Giarratano & Riley:

"Two general methods of inferencing are commonly used as problem-solving strategies of expert systems: forward chaining and backward chaining. Other methods used for more specific needs may include means-ends analysis, problem reduction, backtracking, plan-generate-test, hierarchical planning and the least commitment principle, and constraint handling. Forward chaining is reasoning from facts to the conclusions resulting from those facts... Backward chaining involves reasoning in reverse from a hypothesis, a potential conclusion to be proved, to the facts that support the hypothesis." [Giarratano & Riley05, p.31].

A program was written early on in the development of the project that was successful in demonstrating the power of deduction in Prolog and its ability to perform backward chaining (also known as backtracking). This program was “---- Family Tree” which is listed in section 7.3.1 of the appendix. The program was given a number of facts detailing the names of parents and children within a family and the names of the males and females only. Despite being given a limited amount of initial information, this program was capable of determining many relationships, provided that the associated rules were declared.

An interesting relationship is that of cousin. The rule that correctly describes this relationship is given here:

```
cousin2(X,Y):-
parent(A,B),parent(B,X),parent(A,C),parent(C,Y),different(B,C).
```

This rule effectively states that two people (X & Y) are cousins if they share a common grandparent but have different parents to each other.

The following trace shows backtracking in action. In line (1) Prolog is given a query to determine whether ---- and ---- are cousins. In line (3) Prolog takes the first parent-child relationship from its list of facts. This relationship states that ---- is a parent of ----. Prolog then looks to see whether ---- is a parent of ---- (since it is trying to identify a grandparent of ----). This is found to be true on line (5).

Prolog then goes on to look for a grandparent of ----, following exactly the same process as before, and its search is unsuccessful when it finds on line (9) that ---- is not a parent of ----. It then tries to identify a grandparent of ---- by logically continuing through its search of parent-child facts and works on line (11) from the relationship ---- is a parent of ---- and ---- is a parent of ----. This fact is found to be true on line 13 and on lines 14 to 20, it determines that -

--- and ---- are different people and therefore reaches the conclusion that ---- and ---- are indeed Cousins. This is indicated by the word *true* at the end of the program output.

```
cousin2(----,----).
Line(1)      1    1  Call: cousin2(----,----) ?
Line(2)      2    2  Call: parent(_79,_80) ?
Line(3)      2    2  Exit: parent(----,----) ?
Line(4)      3    2  Call: parent(----,----) ?
Line(5)      3    2  Exit: parent(----,----) ?
Line(6)      4    2  Call: parent(----,_129) ?
Line(7)      4    2  Exit: parent(----,----) ?
Line(8)      5    2  Call: parent(----,----) ?
Line(9)      5    2  Fail: parent(----,----) ?
Line(10)     4    2  Redo: parent(----,----) ?
Line(11)     4    2  Exit: parent(----,----) ?
Line(12)     5    2  Call: parent(----,----) ?
Line(13)     5    2  Exit: parent(----,----) ?
Line(14)     6    2  Call: different(----,----) ?
Line(15)     7    3  Call: not(----=----) ?
Line(16)     8    4  Call: '$call'(----=----,not,1,true) ?
Line(17)     9    5  Call: ----=---- ?
Line(18)     9    5  Fail: ----=----l ?
Line(19)     8    4  Fail: '$call'(----=----,not,1,true) ?
Line(20)     7    3  Exit: not(----=----) ?
Line(21)     6    2  Exit: different(----, ----) ?
Line(22)     1    1  Exit: cousin2(----, ----) ?

true ? ;
```

3.1.2 Fox-Duck Adventure Game

An important breakthrough in the development of the search and rescue adventure game happened when the “Fox-Duck” adventure game was found online [URL1]. Although this listing was incomplete and found to contain “bugs”, it was corrected with the help of the trace function in GNU Prolog and some of the output from the game is shown here:

```
| ?- go.
>> listing(location).

location(egg, duck - pen).
location(ducks, duck - pen).
location(fox, woods).
location(you, house).
>> goto(yard).
You are in the yard
>> goto(duck-pen).
You can not get there from here.
>> open(gate).
>> goto(duck-pen).
You are in the duck-pen
The ducks have run into the yard.
>> take(egg).
You now have the egg
>> goto(yard).
You are in the yard
```

```
>> chase(ducks).
The ducks are back in their pen.
>> shut(gate).
>> goto(house).
You are in the house
Thanks for getting the egg.
```

The structure of this program was vital in gaining an insight into how such an adventure game is implemented in Prolog. It was not long before a new nine-node network was encoded in Prolog. Arcs between the nodes were made to be either available or unavailable, depending upon the weather conditions (which were randomly generated at the start of the game). In addition, the player would find out by 'trial and error' what the weather restrictions were.

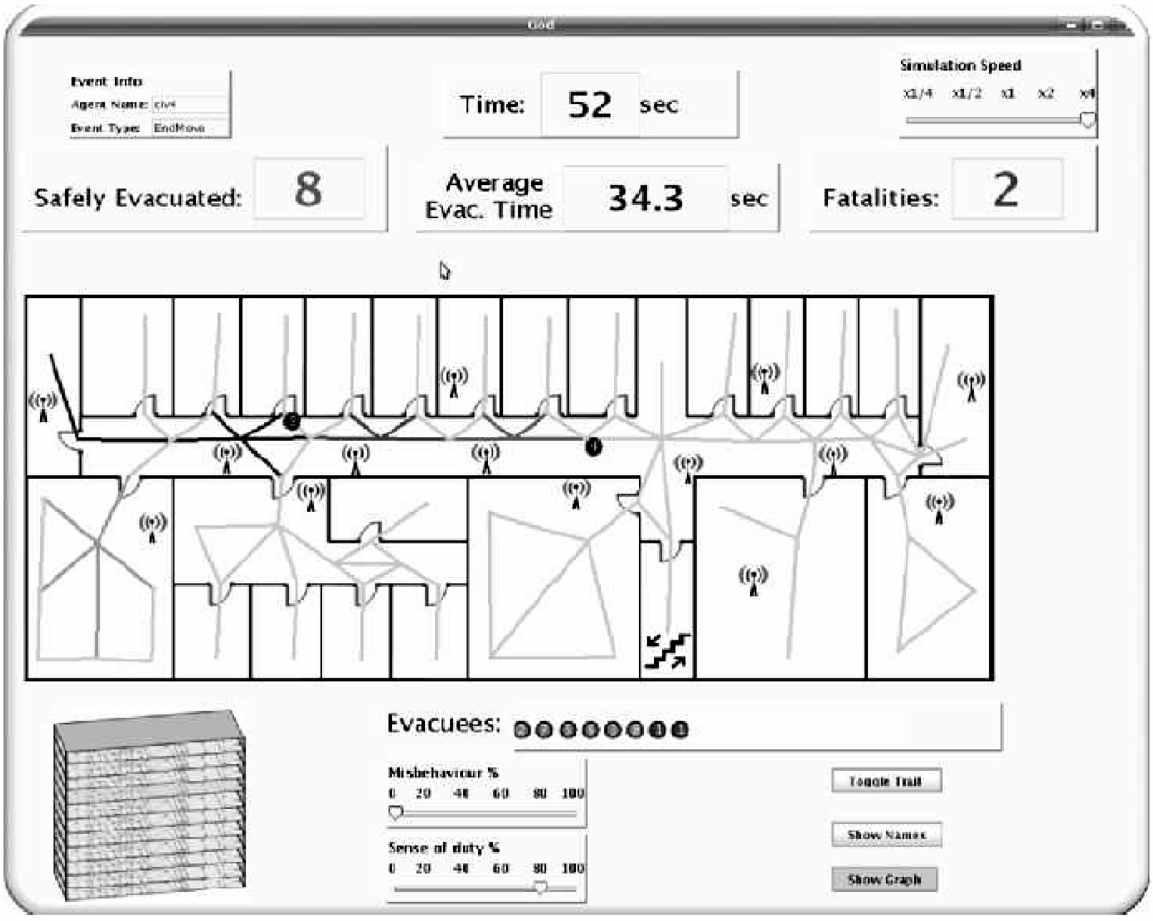
In this example, stormy weather conditions have prevented access to the sea.

```
| ?- start.
Your task is to reach the survivors at the crash_site.
However, today`s weather conditions will limit the routes that
you can take.
Today`s weather condition is: stormy
COMMAND: move(road-b1).
Your current position is: road-b1
COMMAND: move(sea).
It is not possible to reach the sea under today`s weather
conditions.
COMMAND: move(rocks).
Your current position is: rocks
COMMAND: move(woods).
Your current position is: woods
COMMAND: move(crash-site).
Your current position is: crash-site
You have reached the survivors, well done!
G-A-M-E---O-V-E-R
```

3.1.3 Design

Since an evolutionary prototyping development model was chosen, it was not easy to produce a comprehensive design from the outset, especially since the declarative programming language to be used had not yet been chosen and the capabilities of declarative programming were not at that stage understood. In spite of this, an effort was made to produce a software component diagram when only three weeks into the project. This is shown in Figure 3.1.3(b) and it is a surprisingly accurate representation of the finished project.

Some initial research into relevant and existing training simulations led to the discovery of some work by Filippoupolitis et al. on emergency planning and response. Their graphical user interface was of some interest since besides showing the building plan for the evacuation scenario, it detailed the elapsed time and the number of fatalities against the number of people evacuated. It gave some idea of what the user interface for such a simulation may look like and influenced the design of the GUI for this project. It is shown here in figure 3.1.3(a).



Event Info
Agent Name: civ4
Event Type: EndMove

Time: 52 sec

Safely Evacuated: 8

Average Evac. Time 34.3 sec

Fatalities: 2

Graphical User Interface of the simulator

Figure 3.1.3(a) The Filippoupolitis GUI [Filippoupolitis08]

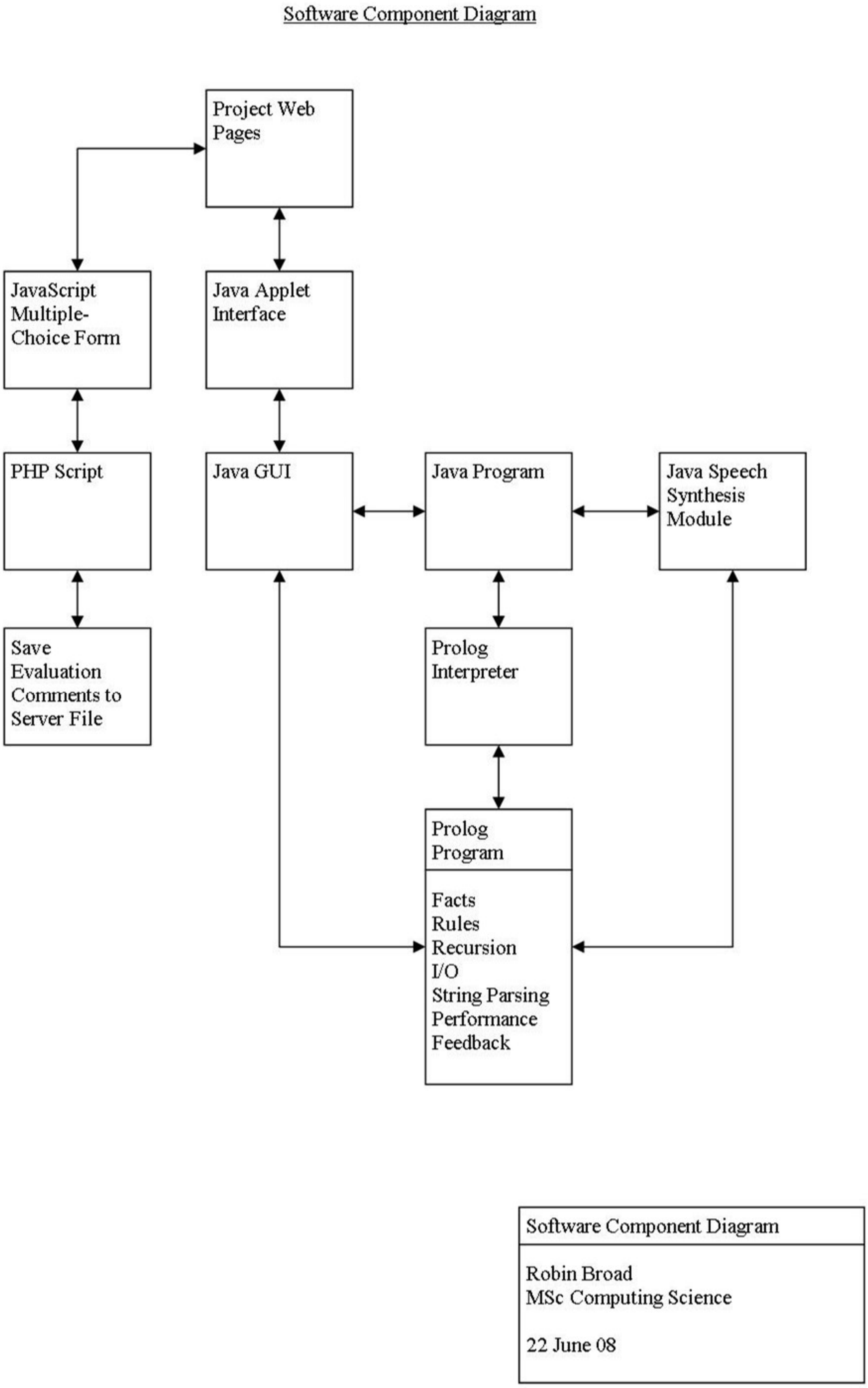


Figure 3.1.3(b) An Early Software Component Diagram

A number of design requirements were considered before the software was developed further, they are listed here:

DESIGN REQUIREMENTS	
R1	The software should be written in Java so that it is capable of being downloaded and executed on a range of platforms including Windows Vista, Windows XP and Mac OS X
R2	The GUI should be usable with a range of different screen resolutions ranging from 800 x 600 pixels to 1200 x 800 pixels
R3	To improve the efficiency and readability of the logic of the game, a declarative programming language should be used in conjunction with Java
R4	The game should take the form of a search and rescue simulation and provide feedback on the user's performance during the game
R5	The weather conditions should be randomly selected at the start of the game and should have an influence on the events that may take place during the game
R6	Random events may generate problems to be solved by the player so that the same problems do not occur at the same place and time during different executions of the game
R7	Random clues should be given to help the player
R8	The player should be able to take equipment from the base and carry out vehicle checks before the search
R9	The game should have a GUI which includes a map of the area and be capable of running independently of the developers terminal window
R10	A speech synthesizer should be used when the machine communicates with the user
R11	The software should be capable of accepting user commands as free-text input, identifying key words without understanding the meaning of all words used in the input
R12	The use of free-text should mean that the user interface is fairly flexible about the choice of words used and should not rely on the use of exact and exclusive command words
R13	It should not be possible to crash the program by using a free-text command that it does not understand
R14	The understanding of free-text commands should be case independent
R15	Good quality images and sounds should be used to maximise the quality of the "look and feel" of the game
R16	There should be a level of realism that would provide a useful learning experience of search and rescue for the user
R17	The elapsed time in "game hours" should be displayed together with a survivor count which should fall randomly with time
R18	The status of the game, including the player's position should be displayed for the user
R19	Clear instructions should be given at the start of the game
R20	There should be an option at the end of the game to play again
R21	There should be an option to leave the game during play
R22	The game should create an impression of intelligence and be capable of engaging in some level of general conversation

Table 3.1.3(a) Design Requirements

3.1.4 Developing the tuProlog Solution

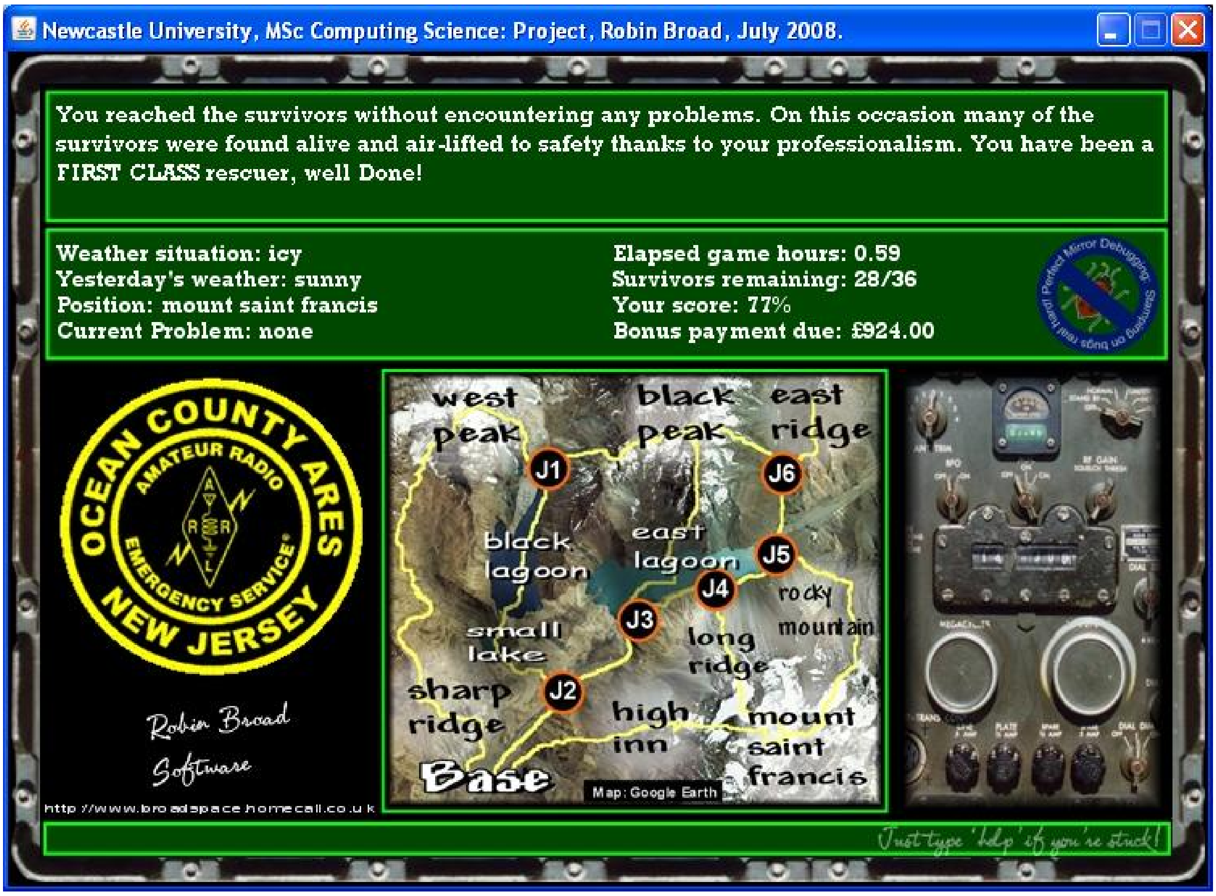
[not in LITE]

3.1.5 Developing the Speech Synthesis

[not in LITE]

3.1.6 Developing the GUI and Concurrency Solutions

[not in LITE]



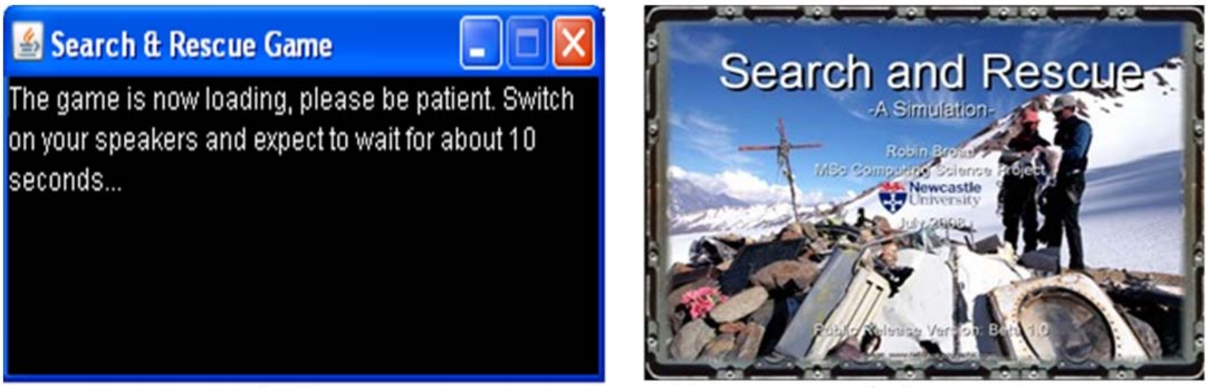
Newcastle University, MSc Computing Science: Project, Robin Broad, July 2008.

You reached the survivors without encountering any problems. On this occasion many of the survivors were found alive and air-lifted to safety thanks to your professionalism. You have been a FIRST CLASS rescuer, well Done!

Weather situation: icy
Yesterday's weather: sunny
Position: mount saint francis
Current Problem: none

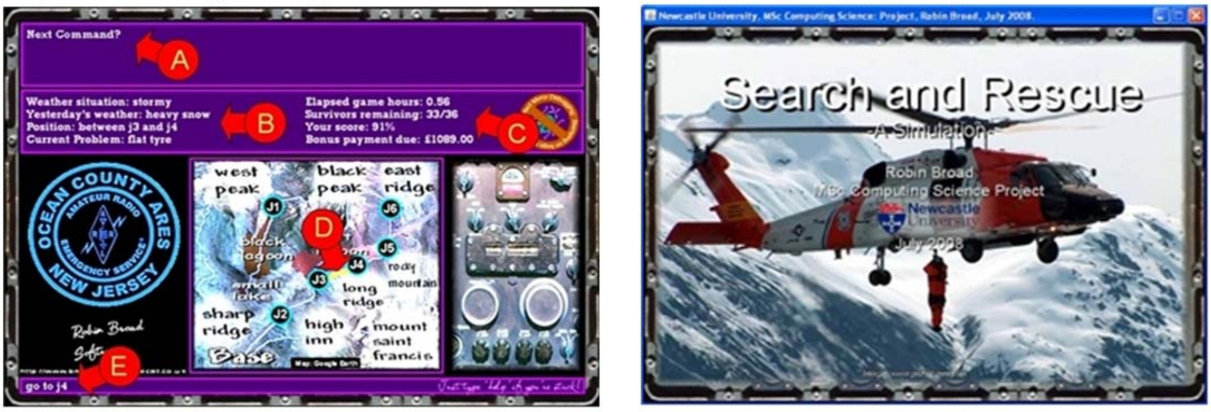
Elapsed game hours: 0.59
Survivors remaining: 28/36
Your score: 77%
Bonus payment due: £924.00

Figure 3.1.6(a) The Search & Rescue Final GUI



The game is now loading, please be patient. Switch on your speakers and expect to wait for about 10 seconds...

pre-loading screen loading screen



```
Next Command?

Weather situation: stormy
Yesterday's weather: heavy snow
Position: between j3 and j4
Current Problem: flat tyre

Elapsed game hours: 0.56
Survivors remaining: 33/36
Your score: 91%
Bonus payment due: £1089.00

go to j4
```

instructions closing credits

Figure 3.1.6(b) Intermediate Stage GUI Displays

3.1.7 Developing the Web Based Resources

[not in LITE]

3.2 High-level explanation

3.2.1 Prolog: RadioHam4

[not in LITE]

3.2.2 Java Classes

(i) Java Classes Overview

Detailed “Javadocs” outlining the purpose, constructor details and the methods of all of the Java classes used in this project have been included, together with commented Java code listings, in *section 7.4 Java Class Listings* in the appendix. It is not necessary therefore; to give a detailed explanation of the structure and operation of all of these classes here; however, an overview of the system and some reasons explaining why it was constructed in this manner will be given here.

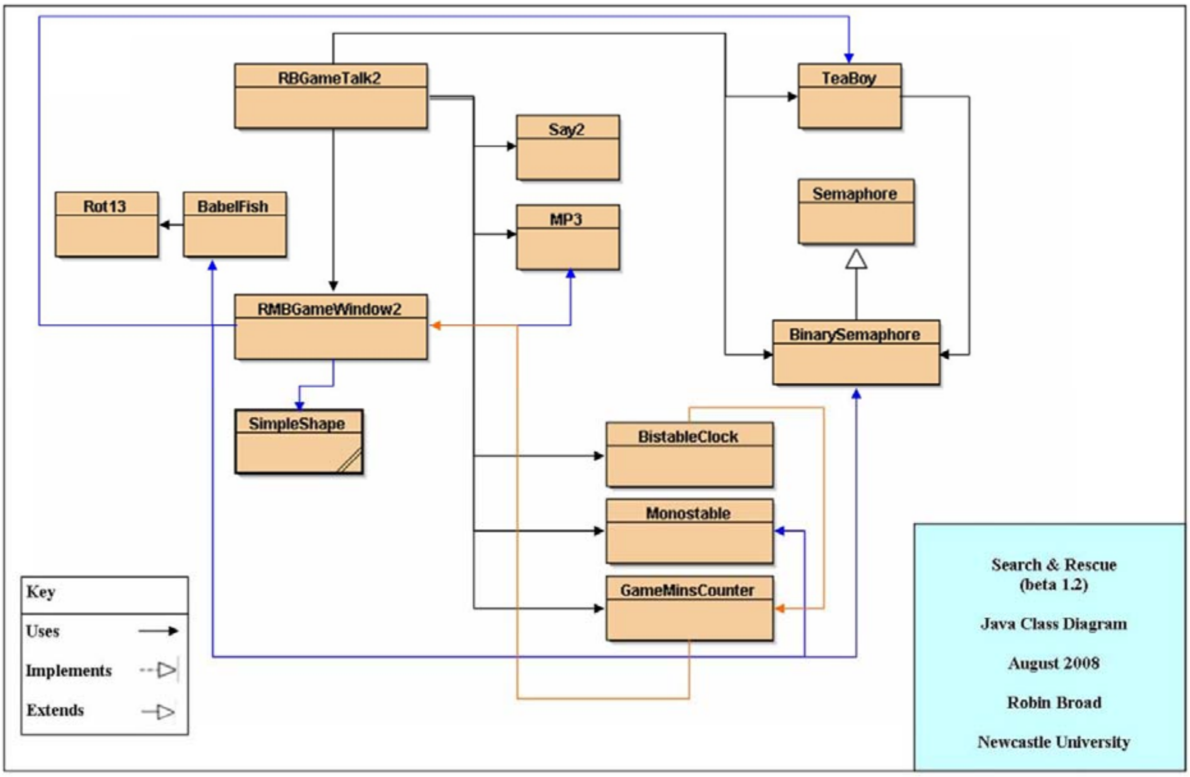


Figure 3.2.2(a) Search & Rescue (beta 1.2) Java Class Diagram

It is apparent from the Java class diagram that the structure of this project from the Java perspective is relatively straightforward. This is because much of the complexity of the game is contained within the Prolog program, which is not shown here. The Java classes are essentially performing the task of the mechanics of displaying the graphical window on the screen, coordinating and playing MP3 files, and synchronising communication between the various Java objects and the Prolog engine.

[not in LITE]

(iii) Speech Synthesis

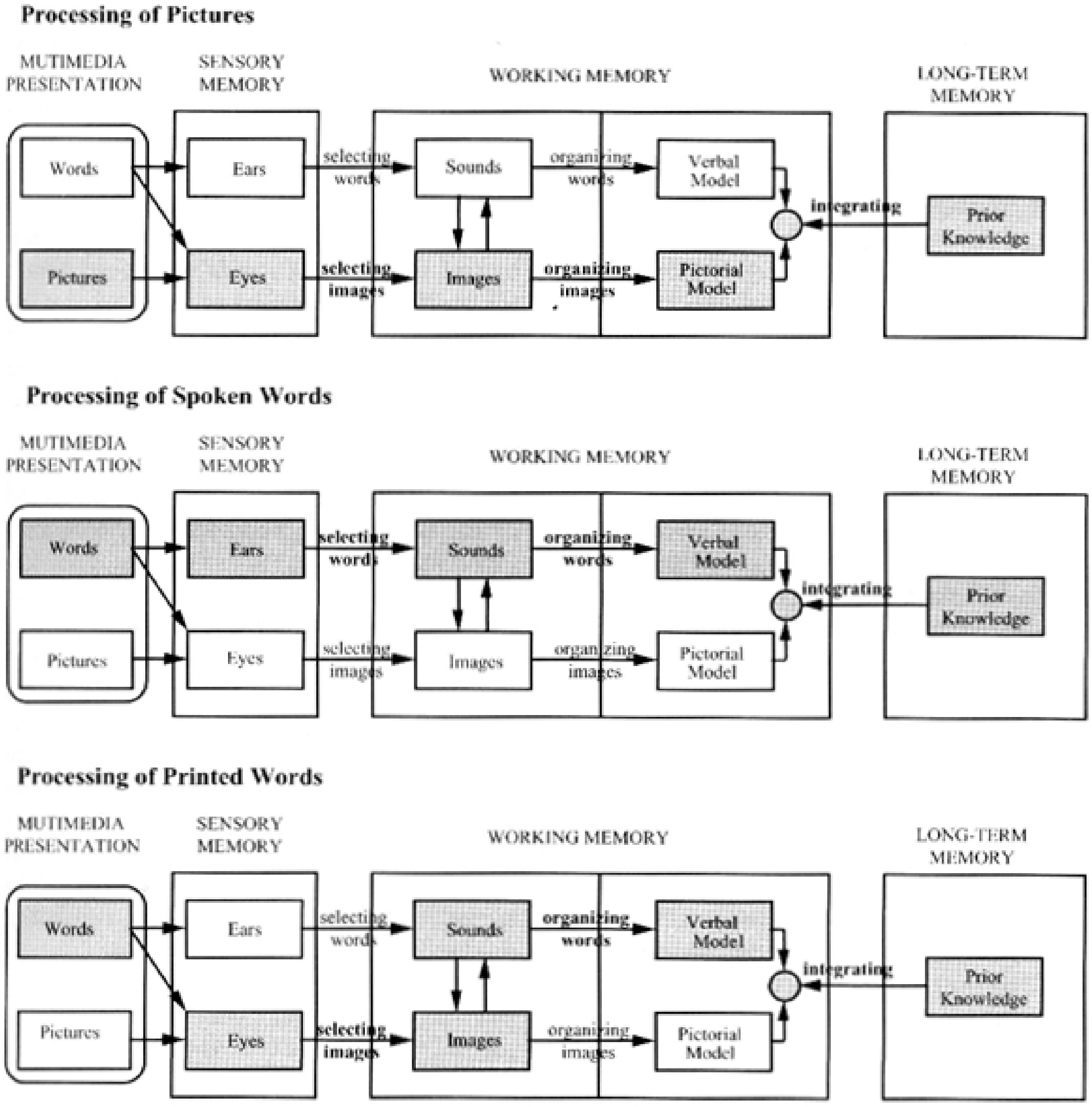
As described in section 3.1.5 *Developing the Speech Synthesis,* implementing the speech synthesizer proved difficult and it was later found to be difficult for users to install too. During the evaluation stage, the speech synthesizer was also found to be quite unpopular, although these problems and criticisms are more likely to be related to the quality of the synthesizer itself rather than the idea of speech synthesis in principle. Far better speech synthesis systems are available, for example that created by IBM but not at a cost that would be appropriate for use in this project. Had such a system been used, it is likely that it would have been easier to install and implement, and therefore be judged more favourably by the participants of the trial.

In their book *Designing the User Interface* Shneiderman & Plaisant refer to one advantage of using speech in educational software:

"Providing feedback and guidance in natural language, even in spoken form, can be helpful in encouraging students to stay engaged in the educational process (Di Eugenio, Glass and Trolio, 2002)." [Shneiderman & Plaisant05, p.338].

Not all users, or in the case of educational software; learners, find reading either easy or enjoyable and learners with a preferred auditory learning style may have good reasons for their choice. The processes taking place within the brain when interpreting printed words, as opposed to spoken words, are quite different. It involves an internal transfer of information from the visual channel to the auditory/verbal channel which is not required when simply listening to speech. This process is explained by Mayer in *The Cambridge Handbook of Multimedia Learning*

"For example, consider the case of the student who must read text and view an illustration. The words are presented visually so they must initially be processed through the eyes-as indicated by the arrow from words to eyes. Then the student may attend to some of the incoming words as indicated by the selecting images arrow and bring them into working memory as part of the images. Then by mentally pronouncing the images of the printed words, the student can get the words into the auditory/verbal channel as indicated by the arrow from the images to the sounds. Once the words are represented in the auditory/verbal channel, they are processed like the spoken words as described previously. This path is presented in the bottom frame of figure 3.2.2(d). As you can see, when verbal material must enter through the visual channel the words must take a complex route through the system, and must also compete for attention with the illustration that the student is also processing through the visual channel." [Mayer05, p.44].



Processing of Pictures

Processing of Spoken Words

Processing of Printed Words

Figure 3.2.2(d) Processing Images, Words and Speech, adapted from [Mayer05, p.43].

3.2.3 HTML and CSS

It is not necessary to give a detailed explanation of the html and CSS style sheets here since they are listed in the appendix; section 7.5 *HTML and CSS*. The web form survey collects information from an html form consisting of radio buttons and text boxes. Upon submission, this data is returned via the “post” method to a PHP script, which assigns the posted data to a series of variables, prints a confirmation report and saves the data in a text file on the server.

3.3 Testing

A great deal of time was spent on testing throughout the course of this project because an evolutionary prototyping software development model was employed. Each iteration in the development cycle needed to be tested and although it was not possible to produce a formal testing plan during the early stages of the project, appropriate tests were carried out at various stages to ensure that the program was behaving as expected at each stage before expanding its capability by adding one or more features or modules.

One of the problems in testing the simulation was that it was very time consuming to run a simulation from beginning to end (which took approximately 10 minutes), only to have to start again whenever a problem was encountered. In order to combat this, some programmer's shortcuts (cheats) such as "teleport(X)." which placed the player at any position on the map without following any logical checks or "set_weather." to enable the programmer to test different sections of the game under various weather conditions were created. The command "mail_order(X)." was created so that items could be 'delivered' from the base to a remote location. These shortcuts speeded up the testing process, enabling the programmer to set up certain situations for testing rather than having to wait for the correct situation to arise by chance.

During testing, around 60 different Prolog commands were tested by entering free-text commands into the command line. These Prolog commands are listed here:

PROLOG COMMANDS CHECKED DURING TESTING					
1	swearing.	21	move(mount_saint_francis).	41	quit.
2	prolog_on.	22	move(rocky_mountain).	42	walk.
3	prolog_off.	23	move(base).	43	swim.
4	no.	24	move(long_ridge).	44	wait_until_morning.
5	yes.	25	move(nonsense).	45	help_team.
6	negative_was_used.	26	take(boat).	46	help.
7	move(sharp_ridge).	27	take(radio).	47	continue_search.
8	move(west_peak).	28	take(map).	48	your_name.
9	move(j1).	29	take(spare_wheel).	49	your_age.
10	move(j2).	30	take(climbing_equipment).	50	your_maker.
11	move(j3).	31	take(binoculars).	51	favourite_colour.
12	move(j4).	32	take(dogs).	52	your_dreams.
13	move(j5).	33	take(food).	53	hello.
14	move(j6).	34	take(water).	54	your_intelligence.
15	move(black_peak).	35	take(compass).	55	your_address.
16	move(east_ridge).	36	take(nonsense).	56	your_emotion.
17	move(east_lagoon).	37	check(anti_freeze).	57	appreciated.
18	move(black_lagoon).	38	check(engine_oil).	58	unknown_question.
19	move(small_lake).	39	check(nonsense).	59	nonsense.
20	move(high_inn).	40	change_wheel.		

Table 3.3(a) Prolog Commands Checked During Testing

Besides testing these Prolog commands, a number of other checks were carried out which are listed here:

OTHER CHECKS CARRIED OUT DURING TESTING	
1	That the GUI would display correctly at different screen resolutions ranging from 800 x 600 pixels to 1200 x 800 pixels
2	That the MP3 files played correctly and that the associated time delays were appropriate for the duration of the file
3	That the map was readable to the user and that the spelling of the names of all locations were correct
4	That the feedback comments were given at the end of the game and that they were appropriate for the game-play that had just taken place
5	That the option to 'play again' behaved as expected
6	That the game ended appropriately when the user asked not to continue
7	That it was only possible to move to adjacent locations on the map
8	That it was not possible to move to a location not shown on the map
9	That progress would be blocked if a restriction was in place
10	That the random generation of clues worked as expected
11	That the random generation of certain feedback comments worked as expected
12	That in the data logging version of the software the text file was saved and appended to
13	That the game could be downloaded and executed on a Windows Vista PC, Windows XP PC and a Mac OS X computer.

Table 3.3(b) Further Checks Carried Out During Testing

Extensive care was taken to debug the Prolog program and to prepare the game to a high standard before the public release of version "beta 1.0". Tests were carried out to ensure that the software was compliant with the design requirements listed in Table 3.1.3(a).

One of the most difficult sections of Prolog code to get right was the 'play again’ option which involved retracting and asserting various facts in Prolog until the original state of play had been achieved. After many hours of patient and methodical work, the 'play again' procedures were repaired and tested until the game would restart in its initial state.

An assumption had been made prior to the publication of the software via the Internet that a Java application developed on a PC would run without a problem regardless of the platform used. This turned out not to be the case as it was soon discovered that the application created was not capable of running on an Apple Mac computer. This exposed a weakness in the testing program since the application clearly had not been tested on a Mac. This had been due to a lack of access to a Mac machine and arrangements were later made to gain access to a Mac-mini where the project was recompiled, tested and repaired until it was deemed to be Mac compatible. This meant that a separate Internet download needed to be made available and that a compromise had to be made by removing the speech synthesis component, which was not easy to install onto the Mac operating system.

3.4 User Manual

Since the software had been packaged for public release via the Internet, download and installation instructions were written in HTML for display on the website. The software was downloaded as a “zip” file, which contained around 30 component files, one of these being the instructions for installing and playing the game. It was discovered early on that the “jar” file created on a PC was incompatible with Mac computers and that the installation of the *speech.properties* file was too difficult for most PC users.

Because of this problem in installing the speech synthesiser, a simplified version of the software was created (as a modified version of the original) which had the speech components removed and was renamed "speechless". This program was then posted onto the website and marked as "Easy PC install" while the original version, which still employed a speech synthesiser was then labelled as” Expert PC install". The “speechless" version of the project was then compiled and tested on a Mac-mini and the resulting file placed on to the website and marked as "Easy Mac install".

The "Expert PC install" instructions are shown here as an example. The exact appearance of the original instructions is not fully replicated here, since the original instructions were designed to be displayed within a web browser.

SEARCH AND RESCUE

A computer simulation game based upon a search and rescue scenario.

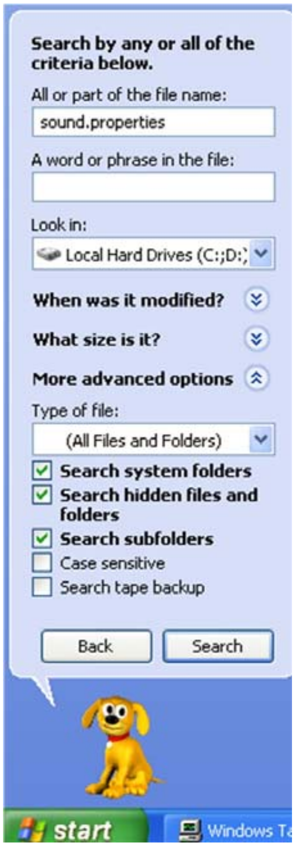
Robin Broad July 2008
Newcastle University, MSc Computing Science Project
Executable Public Prototype Beta 1.2
(5 August 2008)

Instructions

STEP 1

Search for the file "sound.properties" on your machine by going to the XP start menu -> search ->All files and folders.
Type: sound.properties
Click: "More advanced options" and check the boxes shown in Screenshot [1]
Click "Search"

If the search DOES find it then, go to STEP 3
OTHERWISE go to STEP 2



All or part of the file name: sound.properties
A word or phrase in the file:
Look in: Local Hard Drives (C:;D:;
Type of file: (All Files and Folders)
☑ Search system folders
☑ Search hidden files and folders
☑ Search subfolders
☐ Case sensitive
☐ Search tape backup

Screenshot [1]

STEP 2
You don't have a suitable version of Java installed on your machine.
You can get it from here:
(Download the OFFLINE version, it's easier to install)

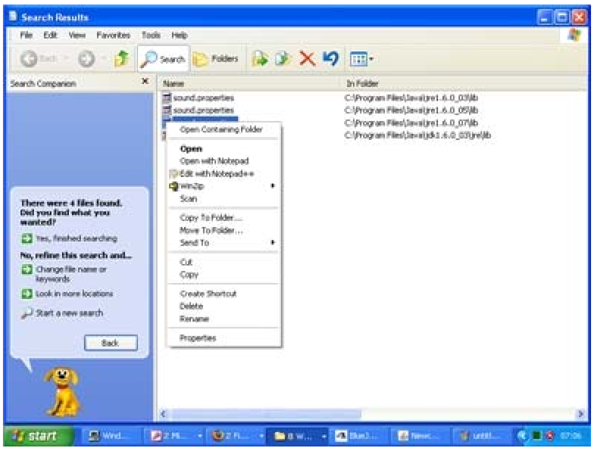
http://www.java.com/en/download/manual.jsp

You will get a file called something like "jre-6u7-windows-i566-p-s.exe" (just make sure that you direct it to download to your desktop so that you can actually find it when you need it!). Double click on it, once it's downloaded, to install the latest version of Java.

Now, GO BACK to STEP 1

STEP 3

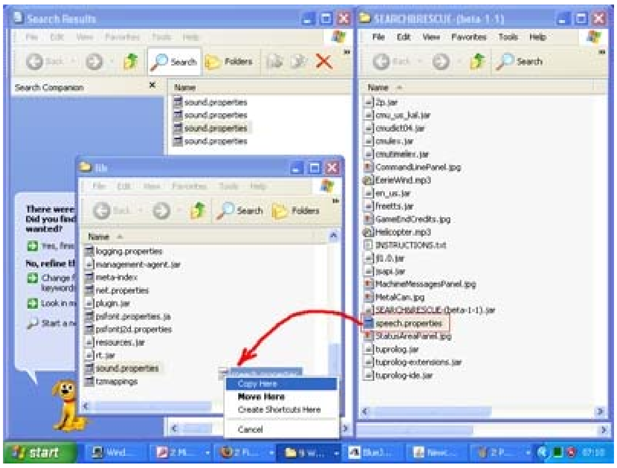
You should now be able to see at least one "sound.properties" file in your search results window. Choose one and right-click on it. Select: "Open containing folder" (see Screenshot [2])



Screenshot [2]

STEP 4

Open the Game folder "SEARCH&RESCUE-Beta-1-2" and look for a file called "speech.properties" RIGHT-CLICK (MUST right-click!) on it and drag it into the "lib" folder which you opened in STEP 3. (See screenshot [3]). Make sure that you choose "COPY HERE" and not "move here" (see screen shot[3]).



Screenshot [3]

STEP 5

If you found more than one "sound.properties" file, then repeat steps 3 and 4 FOR EACH ONE.

STEP 6

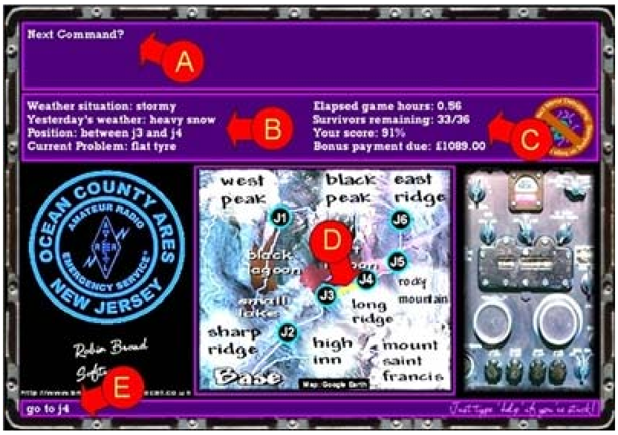
Open the Game folder and double-click on the file "SEARCH&RESCUE-Beta-1-2.jar" to launch the game. {If still won't run, it may be that you skipped STEP 2 and that, although you found a "sound.properties" file, it's from an *old* version of Java. Go back to STEP 2 if you missed it out.}

STEP 7

Game Instructions

The aim of this game is to try and locate the survivors after an air-crash in the mountains. If you reach them, the computer will tell you.

The screen is made up of 5 main parts.
(See the "Game Instructions Screenshot")



Next Command?

Weather situation: stormy
Yesterday's weather: heavy snow
Position: between j3 and j4
Current Problem: flat tyre

Elapsed game hours: 0.56
Survivors remaining: 33/36
Your score: 91%
Bonus payment due: £1089.00

go to j4

Part A is where the machine gives you written messages.

In part B you will see information about today's weather, yesterday's weather, your current position and whether you have any problems to deal with.

Part C shows you how many game hours have passed, how many survivors are left and some other things about your score.

Area D is the map, you can only move along the yellow paths shown.

In part E you type in your commands for the computer.

You can start off by giving commands like "take the food", "take the water", "take the compass" and go to j2 ("j" means junction).

One other thing to mention is that it is possible to "chat" with the AI in the game. You can ask things like "What is your name?" and "Where do you live?", it is programmed with a basic awareness of itself!

Good Luck!

Robin Broad

4 Results and Evaluation

4.1Presenting the results

In his article *Motivational beliefs and perceptions of instructional quality: predicting satisfaction with online training* published in the Journal of Computer Assisted Learning, Tony Artino of the Cognition and Instruction Program of the Department of Educational Psychology at the University of Connecticut, describes how he evaluated an online course on aviation survival training which was developed by the U.S. Navy, where he used the *Likert scale*: [Artino08].

The Likert scale is a psychometric scale, which is widely used in survey research, and this approach was used in 9 out of the 15 questions used in this survey. The reason that this method was chosen was to allow for the numerical processing of the results obtained. The questions used, focused on the following aspects:

- Following instructions and installing the game
- Opinions About the speech synthesizer
- Opinions about learning the rules through discovery
- Rating the stability of the software
- Rating the visual appearance of the game
- The age suitability of the game
- The use of language in the game
- The best and worst things about the game
- What was learned

The actual survey questions used are included here:

Search and Rescue Game Survey
Robin Broad MSc Computing Science
There are only 15 questions.

Q1 How easy was it to download, install and run the game?
(Answer with a number in the range 1 to 5, where 1=very difficult, 5=easy)

1 2 3 4 5
I didn't install the game

Q2 How easy was it to follow the instructions from the website and in the game
folder?
(Answer with a number in the range 1 to 5, where 1=very difficult, 5=easy)

1 2 3 4 5
I didn't install the game

Q3 The game uses a speech synthesizer. How did you find it?
(Answer with a number in the range 1 to 5, where 1=annoying, 5=good fun)

1 2 3 4 5
I couldn't hear the speech

Q4 The game doesn't tell you all of the rules at the start. How did you find
that?
(Answer with a number in the range 1 to 5, where 1=annoying, 5=good fun)

1 2 3 4 5

Q5 Do you think that the game was stable (meaning that it didn't "crash")?

```
(Answer with a number in the range 1 to 5, where 1=not stable at all,
5=very stable)

1 2 3 4 5

Q6 How did you rate the visual appearance of the game?
(Answer with a number in the range 1 to 5, where 1=awful, 5=very good)

1 2 3 4 5
I couldn't see the game

Q7 The game uses humour and sometimes sarcasm. How did you find that?
(Answer with a number in the range 1 to 5, where 1=not good, 5=engaging)

1 2 3 4 5

Q8 Sometimes the game uses informal & friendly language. How did you find
that?
(Answer with a number in the range 1 to 5, where 1=not good, 5=very good)

1 2 3 4 5

Q9 Which age group do you think would like this game the most?
(Answer as a number)

Q10 Was the language easy to understand?
(Answer with a number in the range 1 to 5, where 1=very difficult, 5=easy)

1 2 3 4 5

Q11 Briefly list 3 things that you learned about search and rescue from
playing
this game.

Q12 What were the worst 3 things about this game?

Q13 What were the best 3 things about this game?

Q14 How do you think that this game could be improved?

Q15 Please write any other comments that you would like to add here:

Thank you for filling in and sending this form.
You opinions are valued.
Robin Broad
July 2008
```

Following the Alpha testing of both the Java classes and the Prolog program, which tested for compliance against expected behaviour, a public beta 1.0 version of the software was published online at www.broadspace.homecall.co.uk. This version of the software was accompanied with download and installation instructions, together with user instructions and a link to an online survey, which was placed on the homepages of Newcastle University's server. This survey use the *post* method to return results to a PHP script, which then stored the responses in a text file on the server (the scripts of the web form and PHP scripts are listed in section 7.5 of the appendix).

Arrangements were then made to trial the game with the cadets of the 346 (Tynemouth) Squadron air cadets. The software was installed onto two laptops and four cadets at a time (two on each machine) spent approximately half an hour trying the software. This was repeated three times, giving a total of twelve cadets, who were able to provide feedback to be used in the evaluation. The software used was also slightly modified so that the dialogue between the user and the machine was logged in a text file for later analysis.

During each half hour session, approximately 10 minutes was spent in explaining the aim of the game and the basic instructions for using it. Approximately 10 minutes were then spent playing the first game, from leaving the base to finding the survivors and then, in the last 10 minutes the cadets were given a second attempt at playing the game. This was done because the feedback from the first game and the experience of playing it should have helped them to improve their performance during the second attempt.

A number of emails were sent to several contacts encouraging them to download and install the online release of the game. They were also asked to complete the online survey but a number of people found the installation quite difficult, the biggest problem being the installation of the *speech.properties* file into the JRE folder. In fact, only one person was successful so, it was decided to only take into account the responses provided by the cadets themselves on the evening of the test. The answers to the Likert questions are shown here (questions 1 and 2 have been omitted since the software had already been installed). A simplified statement describing the question is shown in column 2 and the cadets' responses, which show a level of agreement on a scale of 1-5, are shown in the central white cells with the average value being shown to 1.d.p.

It can be seen that some questions, for example question 4, have generated some quite varied responses whereas other questions, for example question 7, produced more consistent scores. In the worst-case situation where the results were totally polarised between 1 and 5, then the mean would be 3 and the standard deviation (which is a measure of the spread of the data from the mean) would be 2. It was decided to display a number on the bar graph (which showed the average score for each question) which would be indicative of the level of disagreement on a scale of 0-5. Since the maximum possible standard deviation would be 2, then this number was multiplied by 2.5 to give a "disagreement" score out of 5. These results are shown in table 4.1 and figure 4.1(a) below:

Question Number		Score											Average	Standard Deviation	Disagreement
1															
2															
3	Like speech synth	3	2	4	3	3	3	4	4	4	4	5	3.5	0.8	2.1
4	Like not knowing rules	1	4	2	2	4	3	2	4	4	4	4	3.1	1.1	2.8
5	Game is stable	1	5	5	4	5	4	5	3	5	5	4	4.2	1.3	3.1
6	Liked visual appearance	2	4	3	2	3	3	3	5	3	5	4	3.4	1.0	2.6
7	Liked humour & sarcasm	3	5	4	4	5	3	3	5	5	4	5	4.2	0.9	2.2
8	Liked informal & friendly language	5	4	4	1	3	3	4	5	4	3	5	3.7	1.2	3
10	Language was easy to understand	2	4	3	3	4	3	5	4	3	3	4	3.5	0.8	2.1

Table 4.1(a) Cadet Responses to Likert Questions on a scale of 1-5

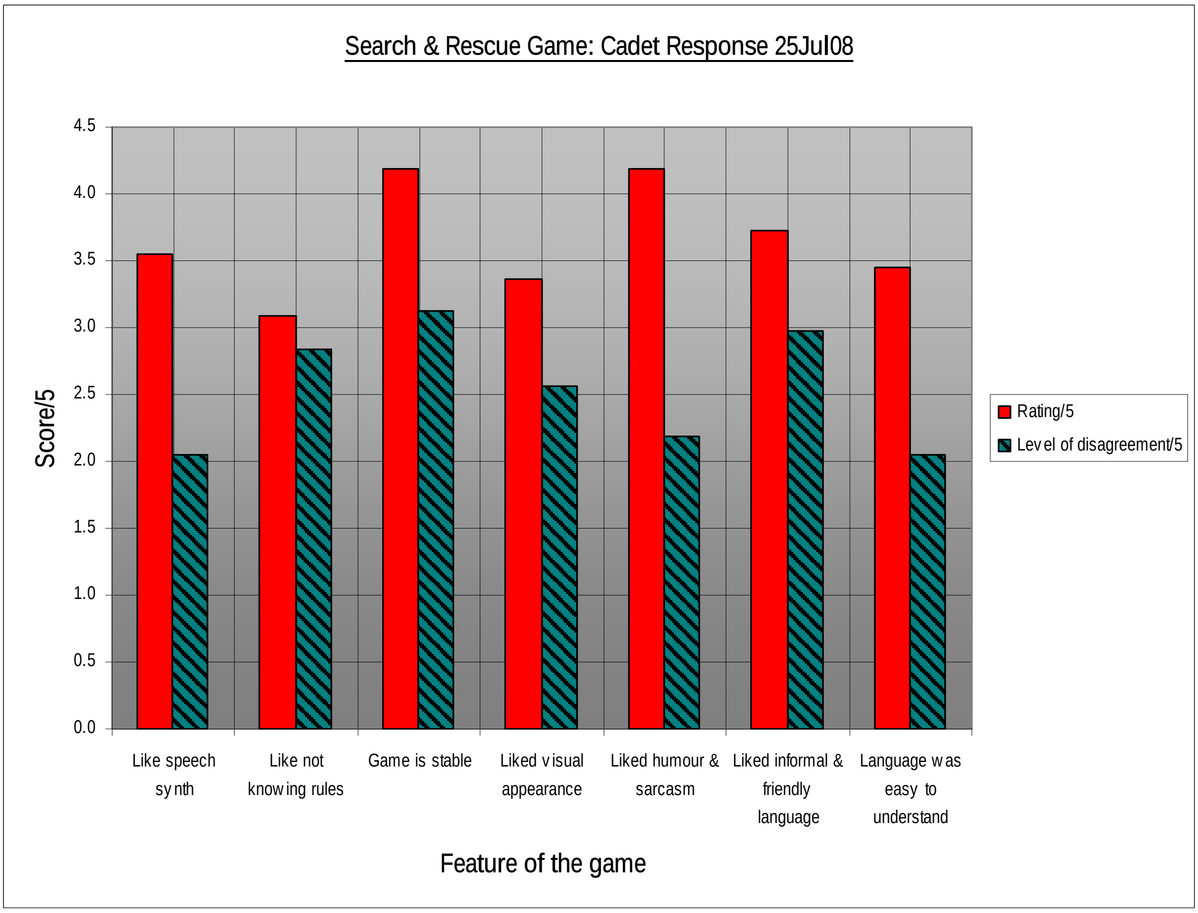


Figure 4.1(a) Cadet Responses to Likert Questions Together with a Disagreement Indicator

These results will be discussed in detail in the next section but it is worth mentioning a few initial observations here. "Liked humour and sarcasm" achieved one of the highest scores and the lowest levels of disagreement, meaning that this aspect of the game was seen to be a good thing, fairly universally. Although “game is stable” also achieved a high score, the level of disagreement was noticeably higher. This was because the software was stable in the experience of 10 out of 12 participants, but for two of them (sharing the same machine), their operating system crashed after clicking too many times on the ".jar" file to try and open it (this was later improved by including a pre-loading message). So, although it achieved a high score, this single very low score increase the spread (standard deviation) of the results, hence raising the "level of disagreement" score.

The questionnaire contained a number of open-ended response questions, a summary of these are given here. These results will be discussed in the next section.

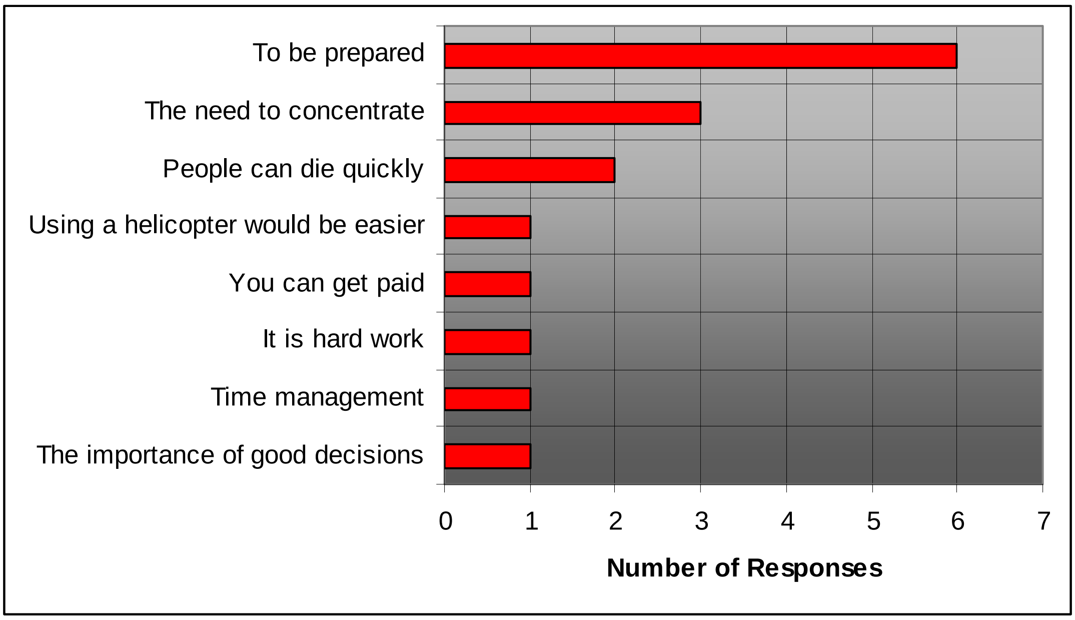


Figure 4.1(b) Cadet Responses to Q11 What did you learn?

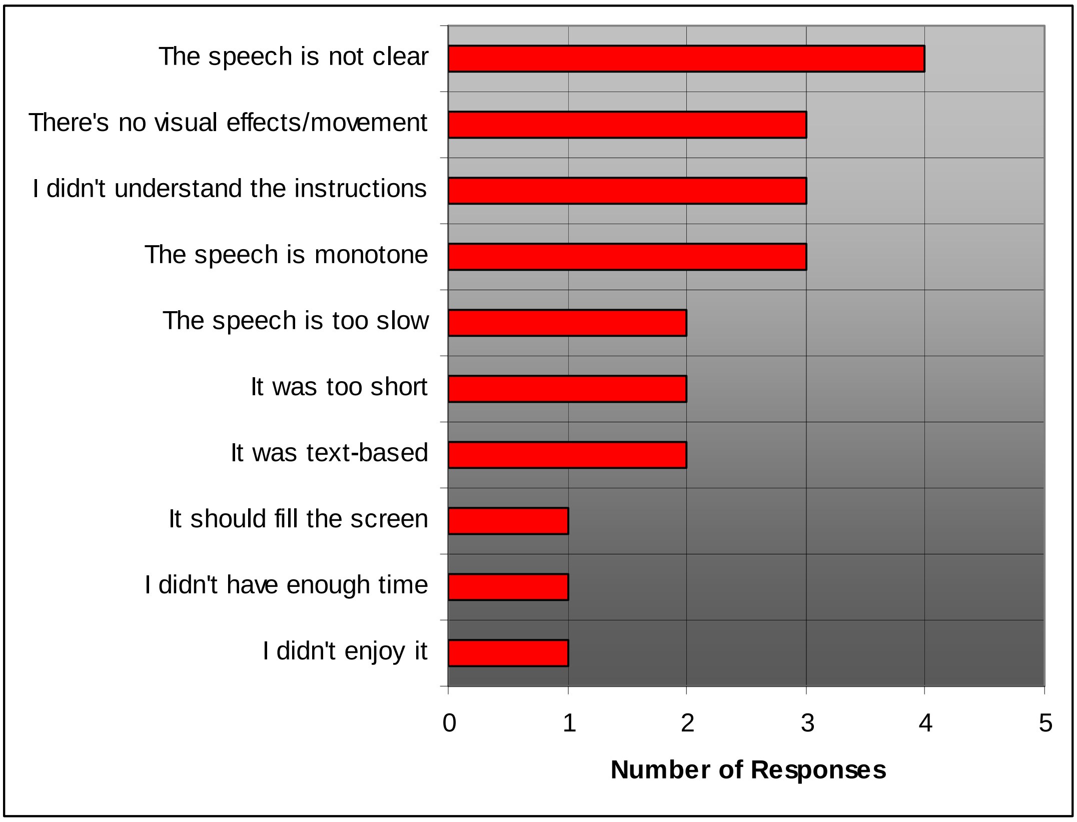


Figure 4.1(c) Cadet Responses to Q12 What were the worst things about the game?

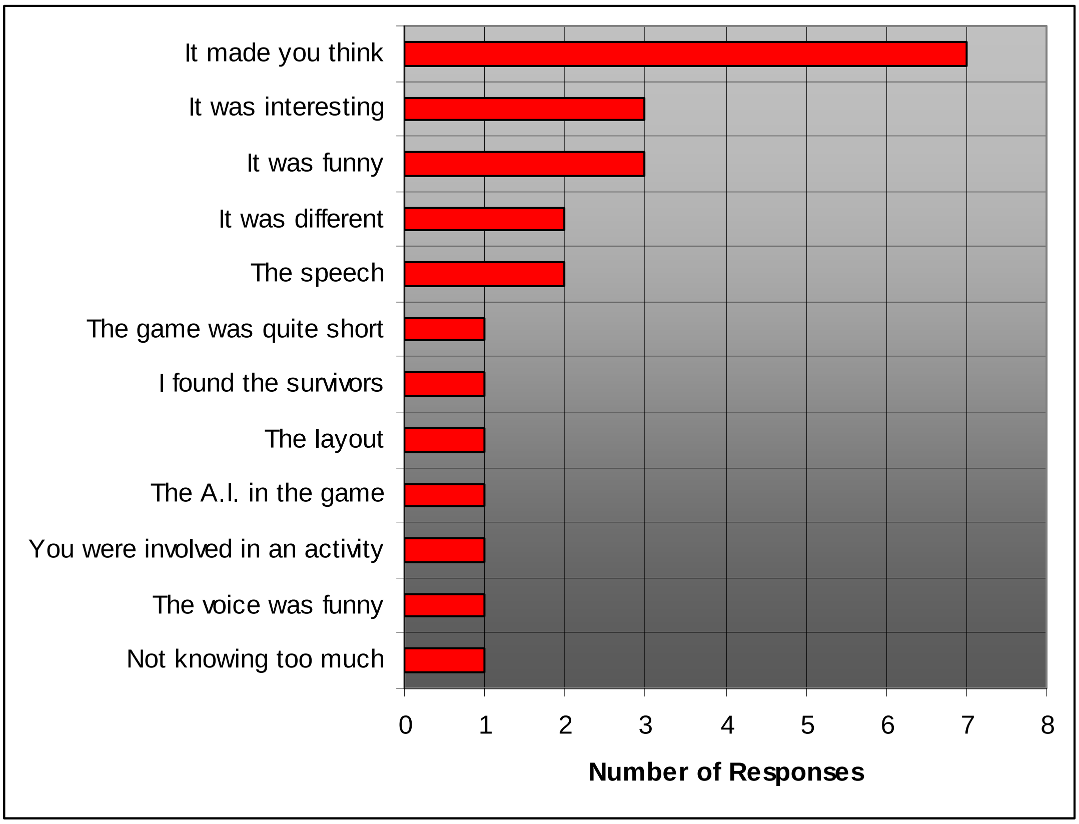


Figure 4.1(d) Cadet Responses to Q13 What were the best things about the game?

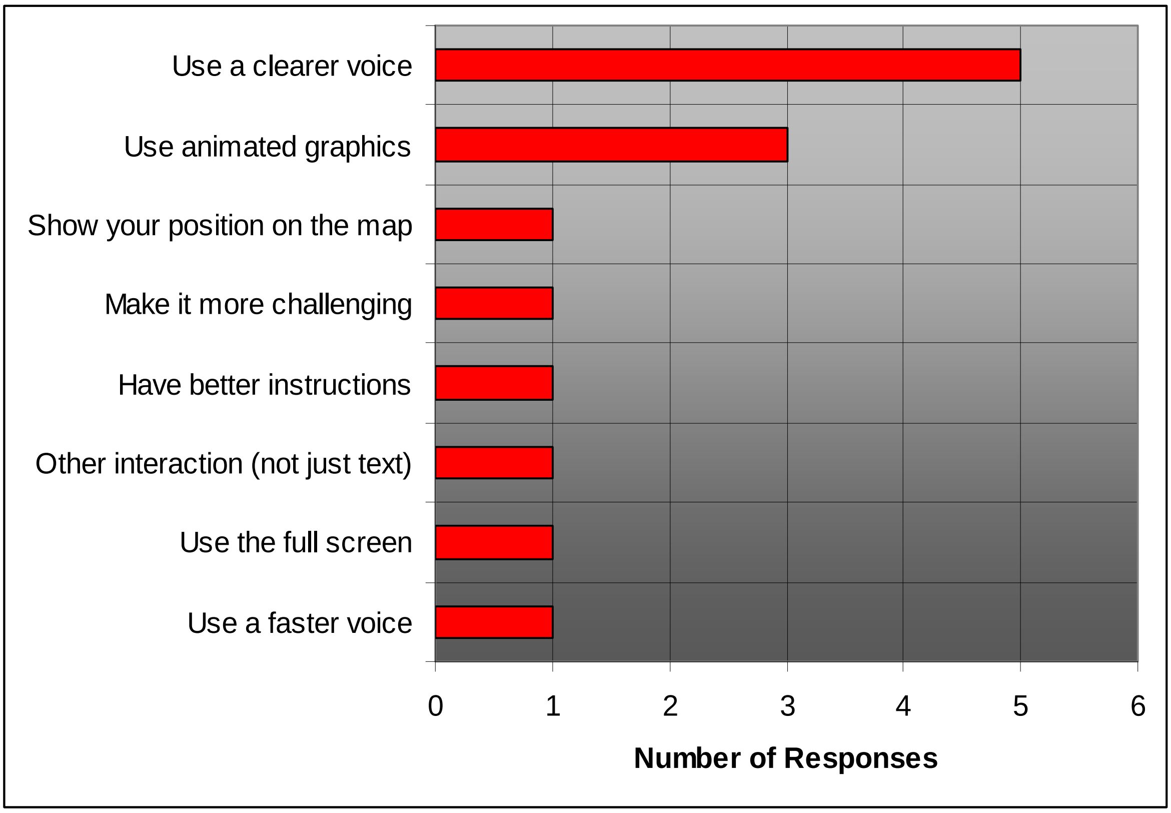


Figure 4.1(e) Cadet Responses to Q14 How could the game be improved?

Q15 Other comments:

- Good!
- I liked the game as it was different to any other games
- It was fun to play
- It was too short
- I enjoyed it
- I will play it at home

4.2Discussion

Although it would have been desirable to have had a greater number of participants for this evaluation, the best use of a limited amount of contact time with the cadets was made and the results obtained lend themselves to making a meaningful analysis of the usability of the software. In addition to the comments already made regarding the responses to the Likert questions, it can be seen that the participants were fairly ambivalent about the use of the speech synthesiser and there was very little disagreement about its average rating. Not knowing the rules of the game caused quite a polarised response with either a fairly high, or a fairly low score and the visual appearance received a similar mixed response. The use of informal and friendly language scored relatively highly, but again, opinion on this was mixed. Finally, there was strong agreement on the language being easy to understand.

Figure 4.1(b), in a sense provides support for the claim that the game has educational value. Since this response was open-ended, it is interesting to note the comments made, in particular those made on more than one occasion, i.e. by more than one participant. “Being prepared” was cited by 6 different participants and the need to concentrate by 3. It is interesting to note that two participants stated that they had noticed that people can die quickly. In the early stages of development of the software some consideration had been given to the notion that people should die at all during a game, it was only after discussion with William Close-Ash that it was decided that this aspect of reality should be represented.

Figure 4.1(c) exposes the perceived weaknesses of the game. The poor speech synthesis is a predominant feature, which demonstrates that, to a modern audience, expectations associated with speech synthesis are high. Criticism for the lack of visual effects and movement is no surprise since this was not one of the key aims of the project. What was of greater concern was the comment from a quarter of the participants stating that they did not understand the instructions. Because of this, in the beta 1.2 release of the software, this was improved with clearer spoken instructions at the start of the game, which were this time a recorded human voice in an attempt to offset some of the criticism about the speech synthesis.

One of the most rewarding aspects of doing this project, besides observing the cadets enjoying using the software, was in reading their responses to question 13 where 7 of them stated that the game had “made them think”. This is shown in Figure 4.1(d). Their responses also included comments about it being “interesting”, “funny” and “different”. Many of the cadets had never used a text-based game before, perhaps because this style of interaction was more fashionable and necessary in the 1970s, i.e. before they were born. It could be argued that this style of interaction is perhaps more intellectually challenging than “point and click” tools with their possibly over used multiple-choice responses.

The opinions about improvements shown in Figure 4.1 are fairly unanimous in their indication that the speech synthesis and the use of animation needs to be improved. The final question, question 15, gave the cadets the opportunity to add any other comments they wished. These were all positive and can be found in section 4.1.

4.3 Evaluation of the approach

Having decided at an early stage of the software development to opt for the potentially more challenging text-based interface rather than the multiple choice selection option, it is worth reflecting upon the outcome of this decision. The results have been impressive if you consider the simplicity and compactness of the code used to process natural language in this game. BabelFish, for example, has only 120 lines of Java code and the Prolog program itself dedicates around 730 lines of code to dealing with what may be considered natural language processing. A rough calculation shows that this is only equal to about 50 Kb of code. Teuscher describes here the complexity of PARRY, one of the earliest chatbots:

"PARRY, like all the large computer programs, is dramatically bound by limitations of cost effectiveness. What was important to Colby and his crew was simulating his model of paranoia. This was a massive effort. PARRY had a thesaurus or dictionary of about 4500 words and 700 idioms and the grammatical competence to use it -a parser in the jargon of computational linguistics. The entire PARRY program takes up about 200,000 words of computer memory all laboriously installed by the programming team." [Teuscher04, p.305].

This can be contrasted with the simplicity of Robbie (as the search and rescue program is known to the player) which occupies an estimated 10 per cent of the memory space occupied by PARRY. This is illustrated in Figure 4.3(a)

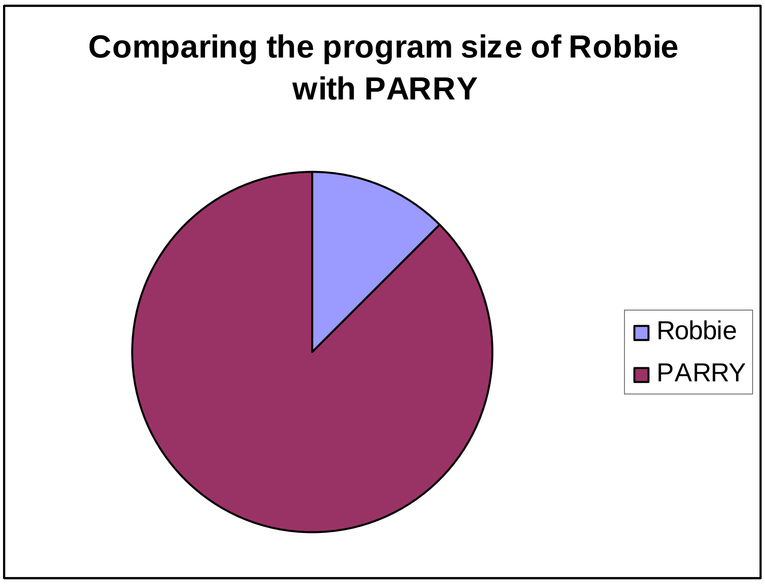


Figure 4.3(a) Comparing the program size of Robbie with PARRY

In the experience of the cadets, using the software was clearly of some educational value and it presented a challenge, which made them think in new ways. When the simplicity and the prototype nature of this software are taken into consideration, then the positive responses described in the previous section are perhaps a little surprising. This is clearly not a highly sophisticated program but it was capable of creating an impression of a level of intelligence that was appealing particularly to younger users. Gazdar & Mellish support the view that although the building of highly intelligent machines, i.e. the goal of "hard AI", is a difficult challenge, applications such as this one, that operate within limited domains still have their place:

"We cannot construct a robust understanding computer without providing it with an encyclopaedic knowledge of the world. These are rather pessimistic conclusions, but they need not prevent us from continuing with the theoretical study of language or indeed from constructing useful computer programs that operate in limited domains." [Gazdar & Mellish89, p.12].

Giarratano & Riley also describe the *knowledge acquisition bottleneck* which refers to the problem of transferring large amounts of human knowledge into a machine. [Giarratano & Riley05, p.11].

Despite the effort that has been put into the evaluation of this software, there is some evidence to suggest that the real value for the participants taking part in such a simulation cannot be easily measured. This was described by Ellington et al. in *Games and Simulations in Science Education* and suggests that using the simulation could have had more benefits than simply those elicited from the survey:

"Other workers have suggested that the real educational value of games, simulations and case studies simply does not lend itself to measurement using formal evaluation procedures. This is echoed by Bloomer, who points out that such teaching methods very often involve complex human interactions of the type that formal input/output evaluation measurements ignore." [Ellington, Addinall & Percival81, p.116].

5 Conclusions

5.1 Meeting the original objectives

Essentially the aims and objectives of this project that were stated in the project proposal and were included in sections 1.5 and 1.6 of this dissertation were all met. This is supported by the feedback from the cadets that took part in the trial. They certainly felt that the program made them think and that they had learned some new things in using it.

There are perhaps two objectives, which, it could be argued have not been fully met. One of these relates to the objective of creating a suitably robust and intuitive program. One pair of participants did manage to crash the program by repeatedly double-clicking on the ".jar" icon used to execute the Java program. This problem came about because there was quite a long time delay, around 10 seconds, for the Java program to initialise and start both the GUI and the introductory MP3 file. During this time, nothing appeared to be happening on screen. This was later improved by creating a small, and fast to initiate, message window which appeared almost immediately after double-clicking on the ".jar" icon, explaining that there would be a short delay. Hopefully, this would reduce the likelihood of this happening again.

It has already been explained that the class BabelFish was quite a lightweight attempt at handling free-text processing and this was a necessary consequence of having to handle many other development priorities. This included facing a deadline to complete both the development and testing of the software, before meeting with the participants for the evaluation trial on a set date and at a set time. Notwithstanding this, it, together with the Prolog rules, which were programmed to respond to its 63 different outputs, performed surprisingly well but when you consider what it cannot do and what it fails to understand, then it could be argued that it is not robust at all.

On the other hand, it is not possible to crash the program by typing in a free-text command that it does not understand. It will always explain that it does not understand rather than trying to do something unpredictable. In this sense it is robust. Although not having your commands understood may be frustrating, the program always offers a way out by giving a tailored response to the user's request for help. It even prompts the user to ask for help whenever it has not been able to make any sense out of a free-text command.

The other objective that was only partially met was that of explaining the reasoning process that was used to reach a solution. Although the program did issue feedback comments at the end of the game which related directly to errors of judgment made by the player, they did not relate to the solving of any particular problem. This would suggest that the machine was capable of looking at the situation and trying a number of different approaches to find out which approach gave the best outcome and then explaining to the player why this approach was the best. Although it would have been possible to achieve this, it was not done since the feedback comments provided were felt to be sufficient. In addition, the time-scale of the project offered only limited opportunities for improvement. Instead, time was spent on improving the processing of free-text commands and making the application easier to download and install on a range of different machines including the Apple Mac.

5.2 Positive and negative aspects

The positive aspects of doing this project have outweighed the negative ones by far. It was very interesting to study Prolog and even more rewarding to create such a large application from it. It was valuable to discover how the power of Java programming can be expanded by incorporating an additional language such as Prolog and that the use of Prolog is still an area of interest and development for computer scientists.

It has been exciting to be able to make so much use of open source code in this project, thereby utilising the work of others to accelerate and enhance the overall progress made in a limited time.

Reading about AI, game playing and simulations, together with the success of this project and the positive feedback given by the cadets who trialed the game, has brought about a realization of the potential for this type of software in education.

One of the most frustrating aspects of developing this software has been in dealing with the way in which the software behaves on different machines. It was found early on, after developing the GUI on a machine with a screen resolution of 1200 x 800 pixels and then transferring it later to an older machine with a screen resolution of 800 x 600 pixels, that the GUI of the program was “over-spilling” the screen area. As a result, the whole GUI had to be rebuilt in order to make it compatible with older machines. There may be away for Java to resize all of the jpeg images within a window to fit the current screen resolution but at the time no way was found of doing this. Building a GUI around a screen resolution of 800 x 600 pixels created the reverse problem: when the application was transferred to a machine of higher resolution it looked “lost” on the screen; a small window on a large desktop.

Not only were there problems associated with varying screen resolutions but there were some minor problems encountered when transferring the program from a machine running one operating system to a machine running another. For example, the title bar in Windows Vista appears to be a few pixels narrower than that of Windows XP, which created annoying little gaps between the components of the panel used in the GUI and some changes had to be made to the background colours etc to minimize the visual impact of this. In the end, the program was tested and modified to behave as expected on three different operating systems; Windows Vista, Windows XP and Mac OS X but this meant that a range of different downloads and installation instructions had to be created.

The use of the command “sleep (seconds*1000)” in class *Monostable* was understood to create a delay in milliseconds equal to the value of the integer passed to the *sleep* method. It now seems that this is not the case since the time delays on the Apple Mac appeared to be much shorter. There was not enough time to investigate the cause of this but in these days of multi core processors, it may be that the unit of time associated with this method is not milliseconds but perhaps executed clock cycles (or something along those lines).

5.3 Future work

One of the most interesting problems that presented itself during the course of this project was that of natural language processing. This subject in itself could easily form the basis of a project that could look at the potential for developing the existing system by increasing its usability from a natural language processing perspective. This includes the problem of synonyms explained in section 3.2.1.

Further research could be done in to the type of questions that the user may ask of such a system and the type of commands that may arise from the user during free-text input. Data logging of the type used already during this project would be away for word in gathering information for analysis.

Another idea which arises from Turing’s “imitation game” [Teuscher04, p.168] is the idea of creating a piece of software which gave the impression to the user that they were interacting with a program, similar to the one written for this project, whereas in fact they were communicating with a human placed in another room. The human would respond intelligently to the user's commands and control the game-play while the dialogue was logged to a file. If this interaction were repeated for a variety of different users, then the log files generated would provide a useful source of information in developing the natural language processing capability of the program. The communication that would be observed during this interaction would be representative of the type of commands and responses that would be reasonably expected during the running of such a simulation. Attempting then, to create software that was capable of dealing with the user requests encountered during this trial would be a good starting point in trying to develop a more intelligent and robust natural language processing system.

One of the things that the program developed during this project does not do is to build up some sort of user profile that could be used to determine how much of a particular training program had been completed. It would be especially important to be able to save this information between sessions so that the user could carry on from a point in the training cycle where they had previously left off and the machine could ensure that they had completed all aspects of the training program.

Such a system, GUIDION, was developed for the MYCIN Medical expert system, which was described by Sleeman & Brown [Sleeman & Brown82, p.204]. Not only was GUIDION capable of tracking student progress, but it was able to tailor its feedback depending upon the student’s ability. This is not something, which was taken into account in the search and rescue simulation and again is an idea that could be developed during further work.

Another aspect of training simulation, outlined by Bloom & Bowen-Loftin in their description of the NASA training simulator, is that of scenario generation. There were some random elements in the generation of the scenario for each game played in the simulations generated for this project, but there was neither variation in the level of difficulty nor any attempt to match the complexity of a simulation with the ability of the user. This is something that can be learned from the NASA system and developed in relation to this project. Bloom & Bowen-Loftin’s description of it is given here:

"The training scenario generator (for Space Shuttle Flight Dynamics Officers) examines the trainee model and creates a unique scenario for each trainee whenever a new session begins. This scenario is built from a database containing a range of typical parameters describing the training context and problems of graded difficulty. Scenarios evolve to greater difficulty as the trainee demonstrates the acquisition of greater skills in solving the training problems. (Refers to the "PD/ICAT" -Payload Assist Module/ Intelligent Computer Aided Training System)" [Bloom & Bowen-Loftin98, p.219].

As mentioned in the proposal, an interesting extension would be to explore the possibility of stripping out the rules, and replacing them with a rule set for an entirely different scenario, while still maintaining the functionality of the original program, therefore making some of the software re-useable.

Finally, it must not be forgotten what the cadets felt were areas for future improvement, namely the improvement of the quality of the speech synthesis component and making better use of interactive graphics during the simulation.

6 References

6.1 Books

[Adams79] Adams, D., *The Hitchhiker's Guide to the Galaxy* Pan Books 1979, ISBN: ISBN 0-330-25864-8.
[Alessi & Trollip85] Alessi, S.M. & Trollip, S.R., *Computer-Based Instruction*, Prentice-Hall 1985, ISBN: 0-13-164161-1.
[Beetham & Sharpe07] Beetham, H. & Sharpe, S., *Rethinking Pedagogy for a Digital Age*, Routledge 2007, ISBN: 0-415-40873-3.
[Bergmann, Moor & Nelson98] Bergmann, M., Moor, J. & Nelson, J., *The Logic Book*, McGraw-Hill 1998, ISBN: 0-07-006059-2.
[Bloom & Bowen-Loftin98] Bloom, C.P. & Bowen-Loftin, R., *Facilitating the Development of Interactive Learning Environments*, Lawrence Erlbaum Associates 1998, ISBN: 0-8058-1851-0.
[Brighton & Selina03] Brighton, H. & Selina, H., *Introducing Artificial Intelligence*, Icon Books 2003, ISBN: 1-84046-463-1.
[Clocksin & Mellish94] Clocksin, W.F. & Mellish, C.S., *Programming in Prolog*, Springer-Verlag 1994, ISBN: 3-540-58350-5.
[Cunningham & Andersson99] Cunningham, U. & Andersson, S., *Teachers, Pupils and the Internet*, Stanley Thornes 1999, ISBN: 0-7487-4307-3.
[Eck95] Eck, D.J., *The Most Complex Machine*, A.K.Peters 1995, ISBN: 1-56881-054-7.
[Ellington, Addinall & Percival81] Ellington, H., Addinall, E. & Percival, F., *Games and Simulations in Science Education*, Kogan Page 1981, ISBN: 0-850388-338-2.
[Gazdar & Mellish89] Gazdar, G. & Mellish, C., *Natural Language Processing in Prolog*, Addison-Wesley 1989, ISBN: 0-201-18053-7.
[Giarratano & Riley05] Giarratano, J.C. & Riley, G.D., *Expert Systems*, Thomson 2005, ISBN: 0-534-38447-1.
[King83] King, M., *Parsing Natural Language*, Academic Press 1983, ISBN: 0-12-408280-7.
[Lowe & Burd07] *Java for Dummies*, Wiley Publishing Inc., ISBN 978-0-470-12451-2
[Luger05] Luger, G.F., Artificial Intelligence, Addison-Wesley 2005, ISBN: 0-321-26318-9.
[Mayer05] Mayer, R.E., *The Cambridge Handbook of Multimedia Learning*, Cambridge University Press 2005, ISBN: 0-521-83873-8.
[Russel & Norvig03] Russel, S. & Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall 2003, ISBN: 0-13-080302-2.
[Saunders, Coote & Crookall87] Saunders, D., Coote, A. & Crookall, D., *Learning from Experience through Games and Simulations*, SAGSET 1987, ISBN: 0-9504682-6-6.
[Shneiderman & Plaisant05] Shneiderman, B. & Plaisant, C., *Designing the User Interface*, Pearson 2005, ISBN: 0-321-26978-0.
[Sleeman & Brown82] Sleeman, D. & Brown, J.S., *Intelligent Tutoring Systems*, Academic Press 1982, ISBN: 0-12-648680-8.
[Teuscher04] Teuscher, C., *Alan Turing: Life and Legacy of a Great Thinker*, Springer 2004, ISBN: 3-540-20020-7.

6.2 Papers and Journals

[Artino08]. Artino, A.R., *Motivational beliefs and perceptions of instructional quality: predicting satisfaction with online training* Journal of Computer Assisted Learning (2008), **24**, 260–270
[Blake & Scanlon07] Blake,C. & Scanlon, E., *Reconsidering simulations in science education at a distance: features of effective use* Journal of Computer Assisted Learning (2007), 23, 491–502
[de Freitas et al.08]. de Freitas, S., M. Oliver,M., Mee.A. & Mayes,T. *The practitioner perspective on the modeling of pedagogy and practice* Journal of Computer Assisted Learning (2008), 24, 26–38
[Filippoupolitis08] Filippoupolitis, A.et al., *Emergency Response Simulation Using Wireless Sensor Networks,* Ambi-sys ICST 978-963-9799-16-5
[Klügl08]. Klügl, F., *A Validation Methodology for AgentBased Simulations* SAC'08 March 1620,2008, Fortaleza, Cear´a, Brazil ACM 9781595937537/08/0003
[Quinn & Neal08] Quinn, C. & Neal, L., *Opinion: Serious games for serious topics* eLearn Magazine, Volume 2008, Issue 3 (March 2008), Page 5.
[Ravenscroft07] Ravenscroft, A., *Promoting thinking and conceptual change with digital dialogue games* Journal of Computer Assisted Learning (2007), **23**, 453–465

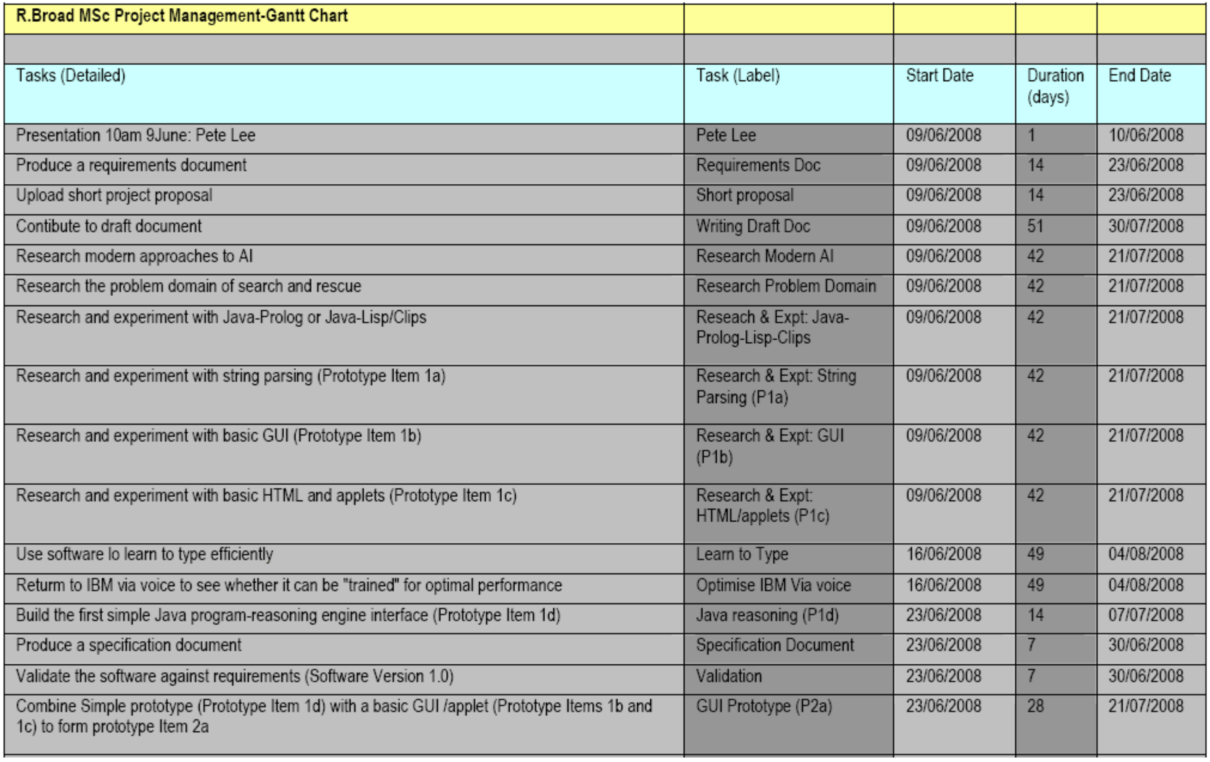
6.3 Websites

[URL1] *Fox-Duck Adventure Game*
http://www.cs.auckland.ac.nz/~jhamer/07.363/explore.html, June 2008.
[URL2] *Synonyms for the word "take"* http://www.yourdictionary.com/take, August 2008.
[URL3] *Variations on the word "Carabiner"* http://en.wikipedia.org/wiki/Carabiner, August 2008.
[URL4] *Visual thesaurus representation of the word "take"* http://www.visualthesaurus.com/, August 2008.
[URL5] *'freetts' installation*
http://freetts.sourceforge.net/docs/index.php#download_and_install, August 2008.
[URL6] *Using 'freetts' with BlueJ*
http://telscenter.org/confluence/display/UCWH/MontyHall+BlueJ, August 2008.
[URL7] *Agents in tuProlog* http://lia.deis.unibo.it/corsi/2007-2008/SMA-LS/slides/7-SMA2007-tuprolog-h.pdf, August 2008.
[URL8] *Unleash Your Creativity with Swing and the Java 2D API!*
http://java.sun.com/products/jfc/tsc/articles/swing2d/index.html, August 2008

7 Appendices

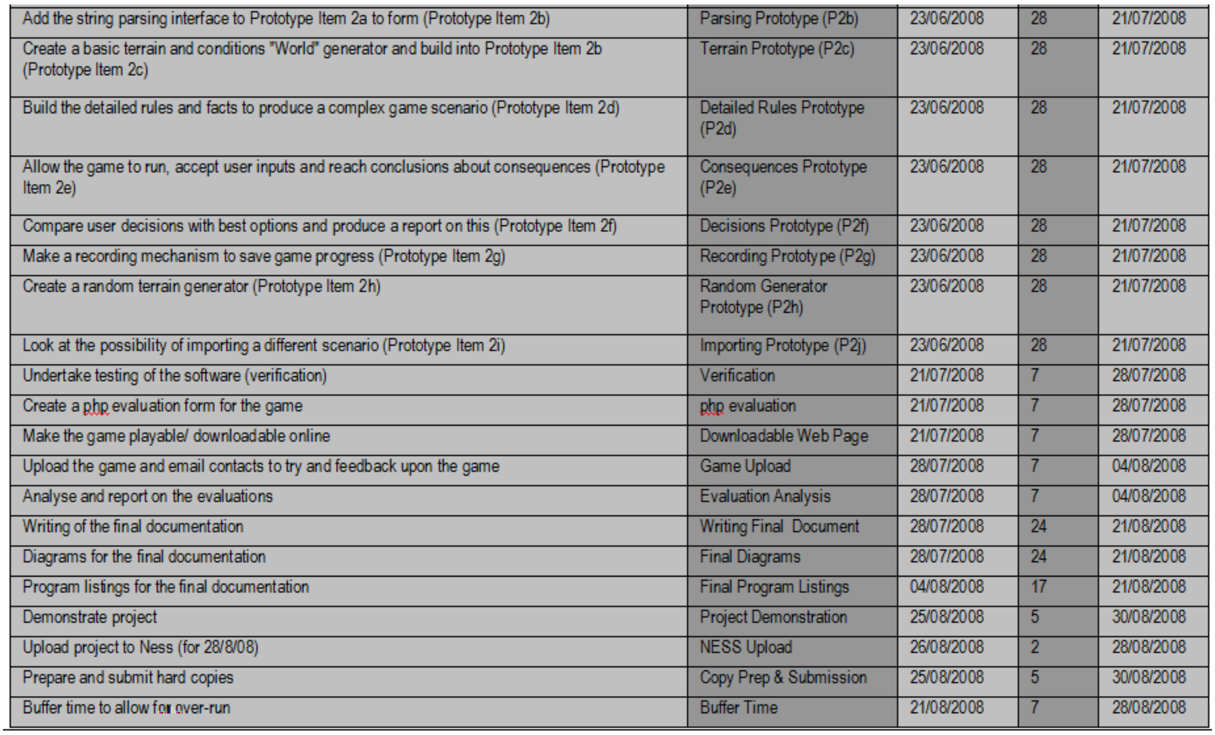
7.1 Planning Gantt Chart

(i) Spreadsheet data used for the Gantt Chart



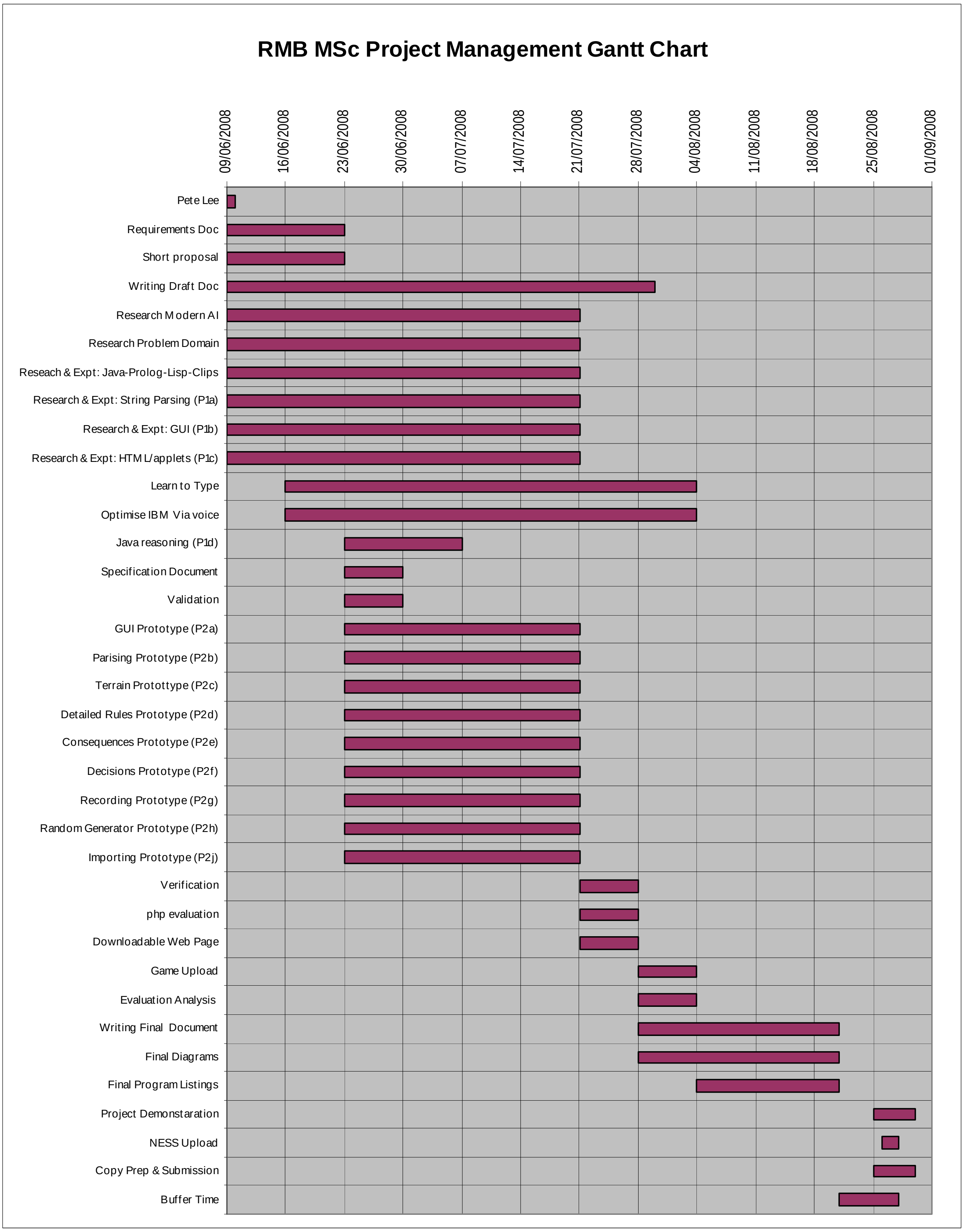
R.Broad MSc Project Management-Gantt Chart

Tasks (Detailed)	Task (Label)	Start Date	Duration (days)	End Date
Presentation 10am 9June: Pete Lee	Pete Lee	09/06/2008	1	10/06/2008
Produce a requirements document	Requirements Doc	09/06/2008	14	23/06/2008
Upload short project proposal	Short proposal	09/06/2008	14	23/06/2008
Contibute to draft document	Writing Draft Doc	09/06/2008	51	30/07/2008
Research modern approaches to AI	Research Modern AI	09/06/2008	42	21/07/2008
Research the problem domain of search and rescue	Research Problem Domain	09/06/2008	42	21/07/2008
Research and experiment with Java-Prolog or Java-Lisp/Clips	Reseach & Expt: Java-Prolog-Lisp-Clips	09/06/2008	42	21/07/2008
Research and experiment with string parsing (Prototype Item 1a)	Research & Expt: String Parsing (P1a)	09/06/2008	42	21/07/2008
Research and experiment with basic GUI (Prototype Item 1b)	Research & Expt: GUI (P1b)	09/06/2008	42	21/07/2008
Research and experiment with basic HTML and applets (Prototype Item 1c)	Research & Expt: HTML/applets (P1c)	09/06/2008	42	21/07/2008
Use software lo learn to type efficiently	Learn to Type	16/06/2008	49	04/08/2008
Returm to IBM via voice to see whether it can be "trained" for optimal performance	Optimise IBM Via voice	16/06/2008	49	04/08/2008
Build the first simple Java program-reasoning engine interface (Prototype Item 1d)	Java reasoning (P1d)	23/06/2008	14	07/07/2008
Produce a specification document	Specification Document	23/06/2008	7	30/06/2008
Validate the software against requirements (Software Version 1.0)	Validation	23/06/2008	7	30/06/2008
Combine Simple prototype (Prototype Item 1d) with a basic GUI /applet (Prototype Items 1b and 1c) to form prototype Item 2a	GUI Prototype (P2a)	23/06/2008	28	21/07/2008



Add the string parsing interface to Prototype Item 2a to form (Prototype Item 2b)	Parsing Prototype (P2b)	23/06/2008	28	21/07/2008
Create a basic terrain and conditions "World" generator and build into Prototype Item 2b (Prototype Item 2c)	Terrain Prototype (P2c)	23/06/2008	28	21/07/2008
Build the detailed rules and facts to produce a complex game scenario (Prototype Item 2d)	Detailed Rules Prototype (P2d)	23/06/2008	28	21/07/2008
Allow the game to run, accept user inputs and reach conclusions about consequences (Prototype Item 2e)	Consequences Prototype (P2e)	23/06/2008	28	21/07/2008
Compare user decisions with best options and produce a report on this (Prototype Item 2f)	Decisions Prototype (P2f)	23/06/2008	28	21/07/2008
Make a recording mechanism to save game progress (Prototype Item 2g)	Recording Prototype (P2g)	23/06/2008	28	21/07/2008
Create a random terrain generator (Prototype Item 2h)	Random Generator Prototype (P2h)	23/06/2008	28	21/07/2008
Look at the possibility of importing a different scenario (Prototype Item 2i)	Importing Prototype (P2j)	23/06/2008	28	21/07/2008
Undertake testing of the software (verification)	Verification	21/07/2008	7	28/07/2008
Create a php evaluation form for the game	php evaluation	21/07/2008	7	28/07/2008
Make the game playable/ downloadable online	Downloadable Web Page	21/07/2008	7	28/07/2008
Upload the game and email contacts to try and feedback upon the game	Game Upload	28/07/2008	7	04/08/2008
Analyse and report on the evaluations	Evaluation Analysis	28/07/2008	7	04/08/2008
Writing of the final documentation	Writing Final Document	28/07/2008	24	21/08/2008
Diagrams for the final documentation	Final Diagrams	28/07/2008	24	21/08/2008
Program listings for the final documentation	Final Program Listings	04/08/2008	17	21/08/2008
Demonstrate project	Project Demonstration	25/08/2008	5	30/08/2008
Upload project to Ness (for 28/8/08)	NESS Upload	26/08/2008	2	28/08/2008
Prepare and submit hard copies	Copy Prep & Submission	25/08/2008	5	30/08/2008
Buffer time to allow for over-run	Buffer Time	21/08/2008	7	28/08/2008

(ii) Gantt Chart



7.3.2 Fox-Duck Adventure Game

```
/* Fox-Ducks adventure game */
/* from http://www.cs.auckland.ac.nz/~j-hamer/07.363/explore.html */
/* and http://www.amzi.com/articles/prolog_fun.htm */
/* modified (and de-bugged !) for gnu prolog by Robin Broad */
/* as part of a MSc Computing Science Project, Newcastle University 12/6/08 */

:-dynamic(location/2).
:-dynamic(nextto/3).
:-dynamic(you_have/1).

location(egg, duck-pen).
location(ducks, duck-pen).
location(fox, woods).
location(you, house).
you_have(nothing).

nextto(duck-pen, yard, closed).
nextto(yard, house, open).
nextto(yard, woods, open).

connect(X, Y) :- nextto(X, Y, open).
connect(X, Y) :- nextto(Y, X, open).

goto(X) :-
  location(you, L),
  connect(L, X),
  retract( location(you, L) ),
  asserta( location(you, X) ),
  write('You are in the '), write(X), nl.
goto(X) :-
  write('You can not get there from here. '), nl.

open(gate) :-
  retract( nextto(duck-pen, yard, _) ),
  asserta( nextto(duck-pen, yard, open ) ).
shut(gate) :-
  retract( nextto(duck-pen, yard, _) ),
  asserta( nextto(duck-pen, yard, closed) ).

chase(ducks) :-
  location(ducks, yard),
  location(you, yard),
  retract( location(ducks, yard) ),
  asserta( location(ducks, duck-pen) ),
  write('The ducks are back in their pen.'), nl.
chase(ducks) :-
  write('No ducks here.'), nl.

take(X) :-
  location(you, L),
  retract( location(X, L) ),
  asserta( you_have(X) ),
  write('You now have the '), write(X), nl.
take(X) :-
  write('There is no '), write(X), write(' here.'), nl.

ducks :-
  location(ducks, duck-pen),
  location(you, duck-pen),
  connect(duck-pen, yard),
  retract( location(ducks, duck-pen) ),
  asserta( location(ducks, yard) ),
  write('The ducks have run into the yard.'), nl.
ducks.

fox :-
  location(ducks, yard),
  location(you, house),
  write('The fox has taken a duck.'), nl.
  fox.

go :- done.
go :-
  write('>> '),
  read(X),
  call(X),
  ducks,
  fox,
  go.

done :-
  location(you, house),
  you_have(egg),
  write('Thanks for getting the egg.'), nl.
```

7.6 A beginner’s two game dialogue

```
Weather situation: sunny
Yesterday's weather: sunny
Displayed Machine message: Hello. Starting from the base you are searching the mountains for
the crash site of a missing aircraft. Use the command 'move' to get from one place to another.
You can use the command 'help' at any point in the game, GOOD LUCK!
Displayed Machine message: Next Command?
User free text command: move to j2
BabelFish translated command: move(j2).
New position Status: j2
Displayed Machine message: Your current position is: j2
Displayed Machine message: Next Command?
User free text command: move to small lake
BabelFish translated command: move(small_lake).
Displayed Machine message: It is not possible to reach the small lake because you did not
bring the boat
Displayed Machine message: Next Command?
User free text command: go back to base
BabelFish translated command: move(base).
New position Status: base
Displayed Machine message: Your current position is: base
Displayed Machine message: Next Command?
User free text command: get the boat
BabelFish translated command: take(boat).
Displayed Machine message: You have taken the boat
Displayed Machine message: Next Command?
User free text command: go to j2
BabelFish translated command: move(j2).
New position Status: j2
Displayed Machine message: Your current position is: j2
Displayed Machine message: Next Command?
User free text command: go to small lake
BabelFish translated command: move(small_lake).
New position Status: small lake
Displayed Machine message: Your current position is: small lake
Displayed Machine message: Next Command?
User free text command: go to black lagoon
BabelFish translated command: move(black_lagoon).
New position Status: black lagoon
Displayed Machine message: Your current position is: black lagoon
Displayed Machine message: Next Command?
User free text command: go to j1
BabelFish translated command: move(j1).
New position Status: j1
Displayed Machine message: Your current position is: j1
Displayed Machine message: Next Command?
User free text command: go to west peak
BabelFish translated command: move(west_peak).
Displayed Machine message: It is not possible to reach the west peak because you did not bring
the climbing equipment
Displayed Machine message: Next Command?
User free text command: go to to base
BabelFish translated command: move(base).
Displayed Machine message: You will need to reach some other locations first. Look at your
position in the status window below.
Displayed Machine message: Next Command?
User free text command: go to base
BabelFish translated command: move(base).
Displayed Machine message: You will need to reach some other locations first. Look at your
position in the status window below.
User free text command: go to j1
BabelFish translated command: move(j1).
Displayed Machine message: Next Command?
Displayed Machine message: You are already there, daftie!
Displayed Machine message: Next Command?
User free text command: go to black lagoon
BabelFish translated command: move(black_lagoon).
New position Status: black lagoon
Displayed Machine message: Your current position is: black lagoon
Displayed Machine message: Next Command?
User free text command: go to small lake
BabelFish translated command: move(small_lake).
New position Status: small lake
Displayed Machine message: Your current position is: small lake
User free text command: go to j2
BabelFish translated command: move(j2).
Displayed Machine message: Next Command?
New position Status: j2
Displayed Machine message: Your current position is: j2
Displayed Machine message: Next Command?
User free text command: go back to base
BabelFish translated command: move(base).
New position Status: base
Displayed Machine message: Your current position is: base
Displayed Machine message: Next Command?
User free text command: get climbing equipment
BabelFish translated command: take(climbing_equipment).
Displayed Machine message: You have taken the climbing equipment
Displayed Machine message: Next Command?
```

```
User free text command: get chocolate
BabelFish translated command: take(nonsense).
Displayed Machine message: That isn't in the store.
Displayed Machine message: Next Command?
User free text command: get food
BabelFish translated command: take(nonsense).
Displayed Machine message: That isn't in the store.
Displayed Machine message: Next Command?
User free text command: get water
BabelFish translated command: take(nonsense).
Displayed Machine message: That isn't in the store.
Displayed Machine message: Next Command?
User free text command: get first aid kit
BabelFish translated command: take(nonsense).
Displayed Machine message: That isn't in the store.
Displayed Machine message: Next Command?
User free text command: go to j2
BabelFish translated command: move(j2).
New position Status: j2
Displayed Machine message: Your current position is: j2
Displayed Machine message: Next Command?
User free text command: go to small lake
BabelFish translated command: move(small_lake).
New position Status: small lake
Displayed Machine message: Your current position is: small lake
Displayed Machine message: Next Command?
User free text command: go to black lagoon
BabelFish translated command: move(black_lagoon).
New position Status: black lagoon
Displayed Machine message: Your current position is: black lagoon
User free text command: go to j1
BabelFish translated command: move(j1).
Displayed Machine message: Next Command?
New position Status: j1
Displayed Machine message: Your current position is: j1
Displayed Machine message: Next Command?
User free text command: go to west peak
BabelFish translated command: move(west_peak).
New position Status: west peak
Displayed Machine message: Your current position is: west peak
Displayed Machine message: Next Command?
User free text command: go to sharp ridge
BabelFish translated command: move(sharp_ridge).
New position Status: sharp ridge
Displayed Machine message: Your current position is: sharp ridge
Displayed Machine message: Next Command?
User free text command: go to high inn
BabelFish translated command: move(high_inn).
Displayed Machine message: You will need to reach some other locations first. Look at your
position in the status window below.
User free text command: go to base
BabelFish translated command: move(base).
Displayed Machine message: Next Command?
New position Status: base
Displayed Machine message: Your current position is: base
Displayed Machine message: Next Command?
User free text command: go to high inn
BabelFish translated command: move(high_inn).
New position Status: high inn
Displayed Machine message: Your current position is: high inn
Displayed Machine message: Next Command?
User free text command: go to mount saint francis
BabelFish translated command: move(mount_saint_francis).
New position Status: mount saint francis
Displayed Machine message: Your current position is: mount saint francis
Displayed Machine message: Next Command?
User free text command: go to rocky mountain
BabelFish translated command: move(rocky_mountain).
New position Status: rocky mountain
Displayed Machine message: Your current position is: rocky mountain
Displayed Machine message: You have reached the survivors, well done! GAME OVER.
Displayed Machine message: You did not think to bring the  climbing equipment, which hindered
your progress in finding the survivors. You will need to be better prepared in future.
Displayed Machine message: You did not think to bring the  boat, which hindered your progress
in finding the survivors. You will need to be better prepared in future.
Displayed Machine message: Shall we play again?
User free text command: no
BabelFish translated command: no.
Displayed Machine message: OK then, have a nice day.
Weather situation: sunny
Yesterday's weather: sunny
Displayed Machine message: Hello. Starting from the base you are searching the mountains for
the crash site of a missing aircraft. Use the command 'move' to get from one place to another.
You can use the command 'help' at any point in the game, GOOD LUCK!

Displayed Machine message: Next Command?
User free text command: get boat
BabelFish translated command: take(boat).
Displayed Machine message: You have taken the boat
Displayed Machine message: Next Command?
User free text command: get climbing equipment
```

```
BabelFish translated command: take(climbing_equipment).
Displayed Machine message: You have taken the climbing equipment
Displayed Machine message: Next Command?
User free text command: go to small lake
BabelFish translated command: move(small_lake).
Displayed Machine message: You will need to reach some other locations first. Look at your
position in the status window below.
User free text command: go to j2
BabelFish translated command: move(j2).
Displayed Machine message: Next Command?
New position Status: j2
Displayed Machine message: Your current position is: j2
Displayed Machine message: There are some walkers here. They are quite flustered and are
saying  that they saw an aircraft, which looked as though it was in trouble, going down near
the ridge just to the east of here.
User free text command: go to small lake
BabelFish translated command: move(small_lake).
Displayed Machine message: Next Command?
New position Status: small lake
Displayed Machine message: Your current position is: small lake
Displayed Machine message: Next Command?
User free text command: go to j2
BabelFish translated command: move(j2).
New position Status: j2
Displayed Machine message: Your current position is: j2
Displayed Machine message: Next Command?
User free text command: go to j3
BabelFish translated command: move(j3).
New position Status: between j2 and j3
New current Problem: flat tyre
Displayed Machine message: You tried turning the radio up to drown out the funny noises coming
from one of the wheels. That did the trick for a while but now the jeep is leaning to one side
and not gripping. You get out and look. The tyre is hanging off the rim, the funny noise must
have been a flat tyre.
Displayed Machine message: Next Command?
User free text command: go to long idge
BabelFish translated command: move(long_ridge).
Displayed Machine message: You will need to reach some other locations first. Look at your
position in the status window below.
User free text command: go to to j4
BabelFish translated command: move(j4).
Displayed Machine message: Next Command?
Displayed Machine message: You will need to reach some other locations first. Look at your
position in the status window below.
Displayed Machine message: Next Command?
User free text command: go to j3
BabelFish translated command: move(j3).
Displayed Machine message: It is not possible to reach the j3 due to an unresolved problem.
Displayed Machine message: Next Command?
User free text command: go to j2
BabelFish translated command: move(j2).
Displayed Machine message: It is not possible to reach the j2 due to an unresolved problem.
Displayed Machine message: Next Command?
User free text command: fix tyre
BabelFish translated command: change_wheel.
Displayed Machine message: You did not bring a spare tyre.
Displayed Machine message: Next Command?
User free text command: go away
BabelFish translated command: move(nonsense).
Displayed Machine message: It is not possible to reach the nonsense due to an unresolved
problem.
Displayed Machine message: Next Command?
User free text command: argh
BabelFish translated command: nonsense.
Displayed Machine message: I am sorry but you're not making sense. Try again, this time
checking your spelling, otherwise, try typing 'help' to get an idea of what you can do next.
User free text command: help
BabelFish translated command: help.
Displayed Machine message: Next Command?
Displayed Machine message: I reckon that changing the wheel should do it. Failing that, you
could always walk!
Displayed Machine message: Next Command?
User free text command: change wheel
BabelFish translated command: change_wheel.
Displayed Machine message: You did not bring a spare tyre.
Displayed Machine message: Next Command?
User free text command: walk to base
BabelFish translated command: walk.
New current Problem: None
Displayed Machine message: You are now travelling on foot.
Displayed Machine message: Next Command?
User free text command: run
BabelFish translated command: walk.
Displayed Machine message: You are already walking.
Displayed Machine message: Next Command?
User free text command: walk faster
BabelFish translated command: walk.
Displayed Machine message: You are already walking.
Displayed Machine message: Next Command?
User free text command: go to base
BabelFish translated command: move(base).
```

```
Displayed Machine message: You will need to reach some other locations first. Look at your
position in the status window below.
User free text command: go to j2
BabelFish translated command: move(j2).
Displayed Machine message: Next Command?
New position Status: j2
Displayed Machine message: Your current position is: j2
Displayed Machine message: Next Command?
User free text command: go to base
BabelFish translated command: move(base).
New position Status: base
Displayed Machine message: Your current position is: base
Displayed Machine message: Next Command?
User free text command: get spare tyre
BabelFish translated command: take(spare_wheel).
Displayed Machine message: You have taken the spare wheel
Displayed Machine message: Next Command?
User free text command: go to j2
BabelFish translated command: move(j2).
New position Status: j2
Displayed Machine message: Your current position is: j2
Displayed Machine message: Next Command?
User free text command: go to jeep
BabelFish translated command: move(nonsense).
Displayed Machine message: I'm sorry we don't do trips off the map, nice try!
Displayed Machine message: Next Command?
User free text command: drive
BabelFish translated command: nonsense.
Displayed Machine message: I am sorry but you're not making sense. Try again, this time
checking your spelling, otherwise, try typing 'help' to get an idea of what you can do next.
User free text command: drive
BabelFish translated command: nonsense.
Displayed Machine message: Next Command?
Displayed Machine message: I am sorry but you're not making sense. Try again, this time
checking your spelling, otherwise, try typing 'help' to get an idea of what you can do next.
User free text command: help
BabelFish translated command: help.
Displayed Machine message: Next Command?
Displayed Machine message: If a problem occurs, type 'help' for specific clues. At the base
you can take things and check things. You will learn about these things from performance
feedback as you play this computer simulation. Some useful commands are move, take, check,
change wheel, walk, drive, abandon vehicle, help team, continue search, wait until morning,
swim, quit.
User free text command: go
BabelFish translated command: nonsense.
User free text command: go to j4
BabelFish translated command: move(j4).
Displayed Machine message: Next Command?
Displayed Machine message: You will need to reach some other locations first. Look at your
position in the status window below.
Displayed Machine message: Next Command?
User free text command: go to j3
BabelFish translated command: move(j3).
New position Status: j3
Displayed Machine message: Your current position is: j3
Displayed Machine message: Next Command?
User free text command: go to j4
BabelFish translated command: move(j4).
New position Status: j4
Displayed Machine message: Your current position is: j4
Displayed Machine message: Next Command?
User free text command: end
BabelFish translated command: quit.
Displayed Machine message: Your vehicle had a flat tyre which you were unable to change
because you didn't bring a spare tyre. You should have taken a spare wheel from the base
before setting off.
Displayed Machine message: Your vehicle had a flat tyre which you were unable to change
because you didn't bring a spare tyre. You should have taken a spare wheel from the base
before setting off.
User free text command: end
BabelFish translated command: quit.
Displayed Machine message: You opted, at one point, to walk instead of using the jeep. This
may have been because you didn't take a spare wheel from the base and were then unable to
change it. This in fact cost you two hours, during which time two survivors were lost as a
direct result of your disorganisation.
Displayed Machine message: Sorry you had to go. Have a nice day.
```