

# Massively Multi-Author Hybrid Artificial Intelligence

Oisín Mac Fhearaí, B.Sc. (Hons)

A Dissertation submitted in fulfilment of the requirements for the award of

Doctor of Philosophy (Ph.D.)

to the



Dublin City University

School of Computing

Supervisors:

Mark Humphrys

Ray Walshe

September 2014

## Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: \_\_\_\_\_ ID No.: \_\_\_\_\_  
Oisín Mac Fhearaí

Date: \_\_\_\_\_

# **Thesis title: Massively Multi-Author Hybrid Artificial Intelligence**

**Author: Oisín Mac Fhearaí**

## **Abstract:**

Biological intelligences often consist of many different, complex systems working together, rather than a single mechanism capable of solving every problem. It may be that artificial intelligence requires a composition of multiple, diverse systems; perhaps more than any one individual or one research group can create or even understand.

This dissertation presents an architecture for hosting and efficiently running user-submitted programs (“minds”) intended to solve particular problems (“worlds”), and to facilitate the assembly of these problem-solving minds into larger scale systems of “hybrid” minds which query an existing set of minds (which we call “subminds”) for their suggested actions. These subminds may be hosted on the same machine or remotely, across the Internet. They may have been written by many different authors, with each program intended either as a complete solution to the problem, or with an eye toward its re-use in a modular fashion. Even if a program is written and intended as a complete, independent solution to the problem, that need not preclude its inclusion in a larger, hierarchical hybrid program. In other words, the architecture presented forms a platform for building hybrid artificial intelligence systems using other people’s code.

The dissertation also presents and evaluates a method for automatically combining minds (which may have been written by different authors) for use in a hybrid mind, by performing a statistical analysis of the observed performance of each mind program in a collection of minds submitted by third parties competing to solve two different types of problem.



# Acknowledgements

When embarking upon the journey of a doctoral research plan, there are several ways to go about it. One approach is to run headlong into the task and to identify and attack the research questions with singular focus and steely determination. An alternative option is to take the scenic route, meandering about and engaging in many sidequests along the way. Having decidedly followed the latter path, I have many people of whom to be thankful, and accordingly more names to both remember and forget.

A great number of colleagues and friends provided invaluable help by reviewing drafts and by offering advice in many, many areas where my knowledge was lacking. Among them are Karthika Raghavan, Críostóir Mac Cárthaigh, Seán Mac an Bhaird, Graham Healy, Mark Hughes, Lijuan Zhou, Marija Bezbradica, Irina Roznovat, Declan McMullen, Cathal Gurrin, Aakash Ahmad, Pooyan Jamshidi, Kosala Yapa Bandara, Paul Clarke, Michael Dever, Jie Shi and Alan Smeaton. I am especially indebted to Martin Crane, Na Li and Yun Jin for guidance pertaining to statistics, and to Ciarán O’Leary, Brian Monks and John Pendlebury for brainstorming with me at various stages of the project.

The faculty and staff at DCU provided support in so many forms – Anne-Marie Caherty and Jonny Hobson in particular went out of their way to advise and assist regarding administrative and financial issues, such as forgetting to fill out forms or running out of money. Séamus Keating helped fix my bicycle after finishing his security shift on at least three occasions – thank you!

Time and again, Gary Conway helped set up and administer servers that were critical to achieving what I set out to do. Jim Doyle and Sean Haran also went above and beyond the call of duty, even at the eleventh hour when my hard drive (predictably) failed.

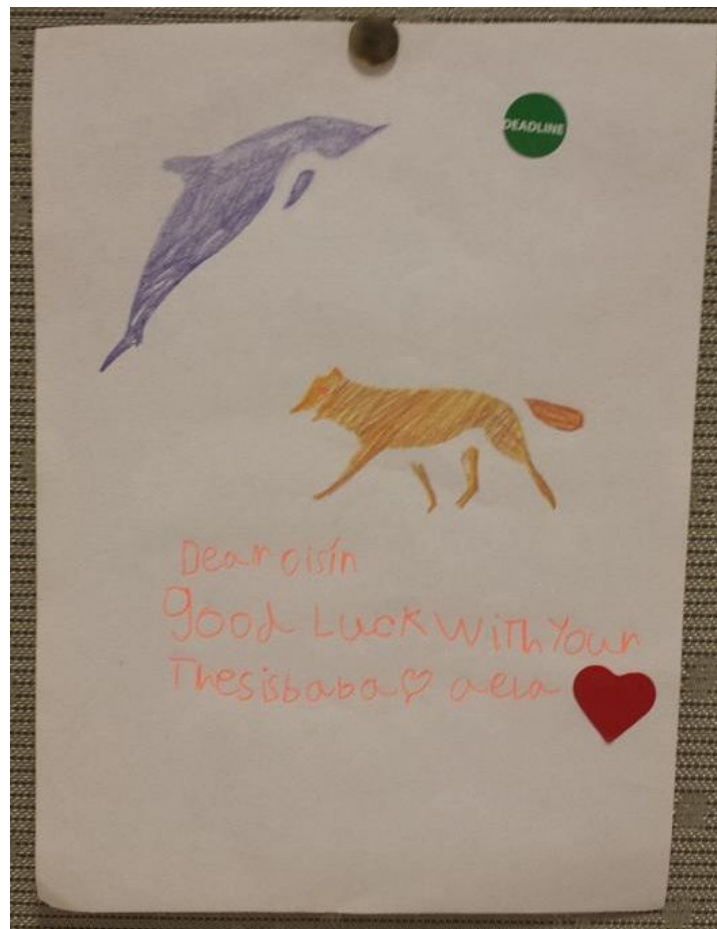
Emer Ní Bhrádaigh and Mairéad Nic Giolla Mhichíl gave excellent advice on practical matters towards the end of the writing process.

I am grateful to IRCSET for investing quite a lot of money to fund me for much of the way.

A large portion of thanks is also due to my supervisors Mark Humphrys and Ray Walshe for their advice and support, helpfully offered at different times so as to be evenly covered.

To my mother, Bríd, for several decades (and counting) of guidance, lifts and minding the kids when we were stuck.

And finally, my undying gratitude to Yunqing Zheng for being her wonderful self, and to Aela and Turloch for being a grand pair of kids whom I love very dearly.





# Contents

<b>1</b>	<b>Introduction</b>	<b>15</b>
1.1	Motivation – Decentralising A.I. . . . . .	15
1.2	Hybrid artificial intelligence . . . . .	16
1.3	The World-Wide Mind . . . . .	16
1.4	Research questions . . . . .	17
1.5	Contributions . . . . .	17
1.6	Dissertation structure . . . . .	18
1.6.1	List of chapters . . . . .	18
1.6.2	Roadmap . . . . .	19
<b>2</b>	<b>Terminology</b>	<b>20</b>
2.1	Introduction . . . . .	20
2.2	Worlds . . . . .	20
2.3	States . . . . .	21
2.4	Actions . . . . .	22
2.5	Scores . . . . .	23
2.6	Goals . . . . .	24
2.7	Minds . . . . .	25
2.8	The action-selection loop . . . . .	26
2.9	Hybrid minds . . . . .	26
2.10	Massively multi-author hybrid artificial intelligence . . . . .	27
2.11	Limitations of this framework . . . . .	28
2.12	Research gap: Automated hybrid building . . . . .	29
2.13	Research aims . . . . .	30



<b>3</b>	<b>Related Work</b>	<b>31</b>
3.1	Introduction . . . . .	31
3.2	Modular intelligence . . . . .	31
3.3	Cognitive architectures and distributed minds . . . . .	32
3.3.1	The Society of Mind . . . . .	32
3.3.2	CogAff . . . . .	33
3.3.3	Unified Theories of Cognition . . . . .	35
3.3.4	The Soar architecture . . . . .	35
3.3.5	CLARION . . . . .	36
3.3.6	Global workspace theory . . . . .	37
3.3.7	On knowledge representation and building interfaces . . . . .	37
3.4	Other work on building modular intelligent systems . . . . .	38
3.4.1	Hybrid minds created by O’Leary et al . . . . .	38
3.4.2	Behaviour-based A.I. and the subsumption architecture . . . . .	40
3.4.3	Behavior-oriented design . . . . .	41
3.4.4	Hierarchical Q-learning . . . . .	41
3.4.5	W-learning . . . . .	41
3.4.6	Constructionist Design Methodology . . . . .	42
3.4.7	Blackboard systems . . . . .	42
3.5	Competition and collaboration in mind-building . . . . .	43
3.5.1	Yet Another Robot Platform (YARP) . . . . .	44
3.5.2	Robot Operating System (ROS) . . . . .	45
3.6	Cloud computing and the semantic web . . . . .	45
3.7	Fast communication in networked computer games . . . . .	46
3.7.1	Packet transmission protocols . . . . .	46
3.8	Techniques for synchronising state over unreliable channels . . . . .	47
3.8.1	Latency and responsiveness . . . . .	47
3.8.2	Hiding the effects of latency . . . . .	48
3.8.3	Limitations of latency-hiding techniques . . . . .	50
3.9	Statistical methods . . . . .	50
3.10	Conclusion . . . . .	51

<b>4</b>	<b>Previous Work: the World-Wide Mind, version 1.0</b>	<b>52</b>
4.1	The World-Wide Mind . . . . .	52
4.2	Communication model . . . . .	53
4.3	Architecture of the World-Wide Mind v1.0 . . . . .	54
4.3.1	Software architecture . . . . .	54
4.4	Interaction of mind and world . . . . .	55
4.5	Society of Mind Markup Language (SOML) . . . . .	56
4.6	Client interface . . . . .	57
4.7	Software API for writing minds and worlds . . . . .	57
4.8	The W2M 1.0 service platform . . . . .	58
4.8.1	Message encoding, transmission and decoding . . . . .	59
4.9	Limitations . . . . .	61
4.9.1	Minds and worlds as web services . . . . .	62
4.10	Conclusion . . . . .	63
<b>5</b>	<b>W2M 2.0: An architecture to enable massively multi-author hybrid intelligence</b>	<b>64</b>
5.1	Introduction . . . . .	64
5.2	Purpose . . . . .	65
5.3	Requirements . . . . .	66
5.4	Architecture of the World-Wide Mind 2.0 . . . . .	67
5.4.1	The run logger . . . . .	67
5.4.1.1	The run logger XML log . . . . .	68
5.4.2	The user interface . . . . .	70
5.4.3	The W2MServer daemon . . . . .	70
5.5	Uploading and testing worlds and minds . . . . .	72
5.5.1	Uploading worlds . . . . .	72
5.5.2	Uploading minds . . . . .	73
5.5.3	Running and testing minds . . . . .	74
5.6	Improvements . . . . .	74
5.6.1	Moving from a peer-to-peer to a more centralised “islands” architecture . . . . .	74

5.6.2	Replacing the web application server . . . . .	76
5.6.3	Service classloader . . . . .	76
5.6.4	Custom network protocol . . . . .	77
5.6.5	Asynchronous runs . . . . .	79
5.6.6	Scalability of large hybrid minds . . . . .	81
5.6.7	Server-side world and mind database . . . . .	83
5.6.8	Robustness and security issues . . . . .	84
	5.6.8.1 Non-terminating programs . . . . .	84
	5.6.8.2 Security policy . . . . .	84
5.7	Writing and running hybrid minds . . . . .	85
	5.7.1 Proxy interfaces for accessing remote mind and world services	85
5.8	Implementation details . . . . .	88
	5.8.1 TcpConnectionHandler . . . . .	89
	5.8.2 ServiceProxy . . . . .	89
	5.8.3 Synchronous and asynchronous communication . . . . .	90
	5.8.4 The problem of non-terminating calls . . . . .	91
5.9	Evaluation of the W2M 2.0 platform . . . . .	94
	5.9.1 Performance benchmarks . . . . .	94
	5.9.2 A.I. teaching assignments . . . . .	95
	5.9.3 Usability . . . . .	97
5.10	Conclusion . . . . .	102
<b>6</b>	<b>Semi-automated hybrid building</b>	<b>104</b>
6.1	Introduction . . . . .	104
6.2	Problem environment: Tyrrell's animal world . . . . .	105
	6.2.1 Use in student assignments . . . . .	106
6.3	Experimental setup . . . . .	107
	6.3.1 Experimental hypothesis and procedure . . . . .	107
6.4	Selecting and ordering the differentiating score attributes . . . . .	108
	6.4.1 Experimental results . . . . .	109
6.5	Mining the information on a state-by-state basis . . . . .	109
6.6	The hybrid mind controller . . . . .	109

6.7	Experimental results . . . . .	110
6.7.1	Preliminary results . . . . .	110
6.7.2	Evaluation . . . . .	111
6.8	Conclusion . . . . .	113
6.8.1	Finding expert subminds . . . . .	114
6.8.2	Limitations of this approach . . . . .	114
<b>7</b>	<b>Towards more automated hybrid building</b>	<b>115</b>
7.1	Introduction . . . . .	115
7.1.1	Statistical methods used . . . . .	116
7.2	On the importance of context . . . . .	116
7.3	Method . . . . .	117
7.3.1	Rank the variables using correlation analysis . . . . .	117
7.3.2	Eliminate redundant variables . . . . .	118
7.3.3	Construct the hybrid mind . . . . .	118
7.4	Experimental setup . . . . .	118
7.4.1	Test scenarios . . . . .	119
7.4.2	Experimental hypothesis and procedure . . . . .	119
7.4.3	Collecting data . . . . .	120
7.4.4	Constructing a hybrid mind . . . . .	121
7.5	Correlation analysis . . . . .	121
7.6	Ranking the performance metrics automatically . . . . .	121
7.6.1	Measuring the correlations between performance metrics and overall success . . . . .	122
7.6.2	Ranking the correlations . . . . .	123
7.6.3	Examining the Tyrrell09 data . . . . .	126
7.7	Selecting subminds . . . . .	127
7.8	Principal Component Analysis (PCA) . . . . .	132
7.8.1	Principal component analysis of the Tyrrell world score data .	133
7.8.2	Selecting a subset of variables . . . . .	134
7.9	Using the correlation matrix to minimise redundancy in variable se- lection . . . . .	138

7.9.1	Reduced variable set for the Tyrrell09 world . . . . .	138
7.9.2	Reduced variable set for the ChessWorldG world . . . . .	140
7.10	Building the hybrid mind controller for <i>Tyrrell09</i> . . . . .	141
7.11	Building the hybrid mind controller for <i>ChessWorldG</i> . . . . .	142
7.11.1	Using reinforcement learning to produce a hybrid controller for ChessWorldG . . . . .	142
7.11.2	Choosing the reward function . . . . .	143
7.11.3	State input mapping . . . . .	143
7.11.4	Action definition . . . . .	144
7.11.5	Network architecture . . . . .	144
7.12	Experimental results . . . . .	145
7.12.1	Tyrrell09 results . . . . .	145
7.12.2	ChessWorldG results . . . . .	147
7.13	Conclusion . . . . .	149
7.13.1	Limitations . . . . .	150
<b>8</b>	<b>Future Work</b>	<b>152</b>
8.1	Extending the W2M 2.0 architecture . . . . .	152
8.1.1	Real-time environments . . . . .	152
8.1.2	Multi-mind environments . . . . .	154
8.2	Technical improvements to the W2M 2.0 platform . . . . .	154
8.2.1	Runlogger: single-threaded multi-process vs multi-threaded single- process . . . . .	154
8.2.2	Limitations of the single-threaded, multi-process design . . .	154
8.2.3	Experiments with a multi-threaded Runlogger . . . . .	155
8.2.4	Further optimisations: Bypassing the Runlogger . . . . .	156
8.3	Building hybrid minds . . . . .	156
8.3.1	Variable selection for hybrid building . . . . .	156
8.3.2	Overlap with existing cognitive architectures . . . . .	157
<b>9</b>	<b>Conclusion</b>	<b>159</b>
9.1	Research questions and outcomes . . . . .	159

9.1.1	How can a hierarchy of minds be supported and built from the programs of many authors? . . . . .	159
9.1.2	Is it useful to build these hierarchical hybrid minds? . . . . .	160
9.1.3	Can this process of building hybrid minds be automated in some way, and is it productive to do so? . . . . .	160
9.2	Closing . . . . .	160
<b>A</b>	<b>List of publications</b>	<b>176</b>
<b>B</b>	<b>Description of the “Tyrrell09” world</b>	<b>177</b>
B.1	Overview . . . . .	177
B.1.1	Perception . . . . .	178
B.1.2	Navigation . . . . .	180
B.1.3	Motor control . . . . .	180
B.2	Differences from Tyrrell’s original simulated environment . . . . .	180
B.3	World configuration file . . . . .	181
<b>C</b>	<b>Description of the “ChessWorldG” world</b>	<b>187</b>
C.1	Problem to be solved . . . . .	187
C.2	Rules . . . . .	187
C.2.1	Invalid moves . . . . .	187
C.2.2	Draws . . . . .	188
C.2.3	Move time limit . . . . .	188
C.2.4	Score . . . . .	188
C.3	State format . . . . .	188
C.4	Action format . . . . .	190
C.5	Graphics . . . . .	190
<b>D</b>	<b>Front-end interface</b>	<b>191</b>
D.1	Introduction . . . . .	191
D.2	Solutions chosen . . . . .	191
D.3	Automated mind and world uploading . . . . .	192
D.4	Browsing the available minds and worlds . . . . .	192
D.5	Initiating and controlling runs . . . . .	194

D.5.1	Starting a run . . . . .	194
D.5.2	Stepping through a run . . . . .	195
D.6	Conclusion . . . . .	196
<b>E</b>	<b>Hosting graphical worlds</b>	<b>197</b>
E.1	Introduction . . . . .	197
E.2	Graphical rendering in previous work . . . . .	198
E.3	A functional, generic graphical rendering system . . . . .	199
E.3.1	Multiple images per timestep . . . . .	199
E.3.2	Optional image generation and cleanup . . . . .	199
<b>F</b>	<b>W2M Server Usage</b>	<b>201</b>
F.1	Introduction . . . . .	201
F.2	Overview . . . . .	201
F.3	Locations . . . . .	201
F.3.1	Binaries . . . . .	201
F.3.2	Source code . . . . .	202
F.3.3	Building from source . . . . .	202
F.4	Usage . . . . .	202
F.4.1	Controlling the back-end daemon . . . . .	202
F.4.2	Cleaning up . . . . .	203
<b>G</b>	<b>Selected worlds written by third parties</b>	<b>205</b>
G.1	World type . . . . .	205
G.1.1	Models of puzzles . . . . .	205
G.1.2	Models based on video games . . . . .	206
G.1.3	Models of complex or real-world problems . . . . .	209
G.2	Conclusion . . . . .	211
<b>H</b>	<b>Code Listings</b>	<b>213</b>
H.1	Introduction . . . . .	213
H.2	A hybrid mind based on a condition list structure . . . . .	213

# Chapter 1

## Introduction

A primary task of intelligence is to provide a control system capable of performing the right actions at the right time. Although many successful control systems can be expressed in a simple way, it may be that complex biological intelligence is fundamentally hierarchical in nature. This dissertation suggests an approach to building intelligent hierarchical control systems, using a partially-automated method to analyse the activity of existing programs and determine which kinds of behaviour are important, and which existing programs to use as building blocks.

This chapter discusses the basic motivation behind this research before introducing some basic terminology. Then we will attend to the research questions which define the scope of this dissertation, before describing at a high-level the primary contributions made. Finally, an overview will be presented of the remaining chapters of this dissertation, along with a roadmap for readers interested in specific aspects of the work.

### **1.1 Motivation – Decentralising A.I.**

There is evidence that human and other biological intelligent systems are composed of many components working in concert, rather than one single mechanism capable of solving any problem [Samuels, 2006]. A number of authors have proposed that in the realm of artificial intelligence, as problems and solutions grow in scale and complexity, a decomposition and distribution of their solutions into component parts will follow [Maes, 1989; Brooks, 1990].



It is certainly possible to combine a set of individual programs together to create a more complex *hybrid* program. But how can this be done effectively, and which component programs should be selected to constitute important areas of expertise in solving the problems at hand?

## 1.2 Hybrid artificial intelligence

The idea of artificial intelligence as many different programs co-operating together is perhaps talked about more often than implemented. When such systems are constructed, the various subprograms are often linked in an ad-hoc way, often requiring the author of the hybrid program to understand or even modify the internal workings of other people's programs.

Sometimes the complexity and required learning curve for building upon or interfacing with other people's programs is so great that the only "hybrid" programs that arise are written by the same author.

In other cases, the roles of different modules are carefully designed, with each part or subprogram representing a well-defined aspect of the overall system's behaviour, and perhaps interfacing with each other in a restricted way which limits the interchangeability of programs.

This dissertation explores a different way of constructing large hybrid programs – by building them from a selection of third-party programs, which each attempt to solve the same overall problem in different ways.

## 1.3 The World-Wide Mind

This section introduces the basic terminology used throughout the dissertation. A much more detailed discussion will follow in chapter 2.

This work, and that which preceded it (see chapter 4 for details of the previous work), envisages that certain types of problem can be expressed as computer programs called *worlds*.

Once a world is made public, anybody can then write programs designed to solve the problem presented by the world. We call these programs *minds*, and hierarchical programs which build upon the expertise of several minds are termed *hybrid minds*.

To make this possible, an architecture and software platform – the World-Wide Mind (W2M) project – was created which allows world and mind programs to be submitted by anybody and made available as services on the Internet.

The fundamental ideas and terminology of the World-Wide Mind project are discussed in greater depth in chapter 2, and an outline of the research goals and the structure of this dissertation is presented.

## 1.4 Research questions

Let us define the scope of this dissertation by asking some questions. We shall return to these later.

1. How can a hierarchy of minds be supported and built from the programs of many authors?
2. Is it useful to build these hierarchical hybrid minds?
3. Can this process of building hybrid minds be automated in some way, and is it productive to do so?

This dissertation attempts to answer each of these questions by extending the W2M architecture and building a platform to support large hybrids, and by devising and evaluating a method for constructing these hybrids.

## 1.5 Contributions

The primary contributions presented in this dissertation include:

1. the World-Wide Mind 2.0 architecture,
2. a partially-automated method for building hybrid minds using minds written by many different authors, and
3. the collection of a large number of minds and several worlds, written by many authors.

The World-Wide Mind 2.0 architecture was designed and realised as a working platform which was used successfully by a large number of authors to develop and test

their mind programs. Some of these authors had very little experience in computer programming and yet produced functional and effective hybrid minds.

The method proposed in this dissertation of building hybrid minds was tested in two test environments: an animal behaviour simulation (Tyrrell09) and a chess problem (ChessWorldG). It is shown to succeed in the first, but not the second.

## 1.6 Dissertation structure

This dissertation consists of two distinct parts. The first part looks at the development of an architecture capable of supporting the development and testing of large hybrid minds, and the second part focuses on *how* these large hybrid minds might be created. To save time for readers with specific interests, this section provides a list of the remaining chapters in this dissertation as well as a roadmap which serves as a quick guide to important parts of the work.

### 1.6.1 List of chapters

The remainder of this dissertation is laid out as follows:

- Chapter 2 defines the core terms used throughout this dissertation, and defines the research gap this work intends to fill.
- A summary of related work is presented in chapter 3, addressing a series of topics relevant to this dissertation, ranging from cloud computing to modular intelligent systems to fast network communication and state synchronisation techniques.
- In chapter 4 the concepts, implementation details and technical limitations of previous work on the World-Wide Mind software platform are discussed.
- Chapter 5 presents some of the work done as part of the contribution of this research to produce a World-Wide Mind version 2.0, which would make the creation of large hybrid minds practical.
- In chapter 6, we look at the problem of constructing a larger-scale “hybrid” mind which delegates to other mind programs in an attempt to combine the individual expertise of each submind. A novel method is presented of selecting

subminds by ranking their performance against a set of metrics representing particular aspects of the problem environment. This method is experimentally validated and the results are discussed.

- These ideas are extended in chapter 7, which presents a partially-automated method of estimating the relative importance of each metric. This knowledge is used to select a small set of effective subminds which together represent expert performance in terms of these metrics. The method is tested in the same problem environment as in chapter 6 and experimental results are discussed in two different test environments.
- Chapter 8 identifies some possibilities for future research.
- Finally, in chapter 9 we will revisit the main contributions of this research.

### **1.6.2 Roadmap**

As a prerequisite, readers should begin with chapter 2 which defines the terminology used throughout the thesis.

For those interested in the architecture and technical implementation of the World-Wide Mind, which allows minds and worlds to be hosted as services on the Internet and provides a facility for minds to call each other, consult chapters 4 and 5. Those looking for more detail regarding the World-Wide Mind platform as it exists today might like to read appendices D, E, F and G.

Chapters 6 and 7 explore how best to construct a hybrid mind from a large set of minds written by other authors, either through manual or partially-automated means.

# Chapter 2

## Terminology

### 2.1 Introduction

This chapter attempts to clarify exactly what is meant by the core terms used in this dissertation, and then defines the research gap I seek to address, and how I intend to address it.

### 2.2 Worlds

A *world* represents a type of problem to be solved. Some problems are called *episodic environments*, where one task is to be performed which is independent of previously completed tasks [Russell and Norvig, 2003]. In these environments, the solution takes the form of a single answer; for example:

- a travelling salesman problem, whose solution will be an ordered sequence of nodes to visit,
- a 2D maze problem, where the solution is a complete sequence of moves leading from the start position to the goal, or
- a part-picking robot problem, where the input is an image of a part and the solution is a decision on whether to accept or reject the part.

More commonly however – at least in the types of problems I envisage being modelled and simulated on this platform – solutions will consist of a series of interactive steps, alternately sensing and acting. These problems are referred to as *non-episodic* or *sequential environments*. Some examples of this type of problem are:

- a game of poker, where the player has several turns where an action can be taken, such as raising the bet, calling or folding one's hand,
- a predator-prey simulation, where at each step, the predator attempts to find or capture the prey while the prey attempts to escape, and
- a game of chess, where two players take alternate turns to capture each other's pieces and ultimately force a checkmate, while minimising the capture of their own pieces and protecting their king.

A world is defined by the set of configurations or states which are possible in that world. Some or all of the state of the world may be perceivable by an agent at any given moment.

## 2.3 States

To effectively solve the problems presented by all but the simplest worlds, some form of sensory input must be available. The set of sensory percepts made available by the world at any given moment will be called the *state*. Note that the state visible to the agent may only be a subset (or even an erroneous subset) of the true and complete state of the world.

Some examples of simple world states are given in figure 2.1 on the following page. The first picture depicts a game of Tic-tac-toe, where the X player holds a favourable position.

The second picture represents a player's hand in a game of blackjack, in which the best move is probably to stick (i.e. not to ask for any more cards).

A final example shows a fictional, simulated foreign exchange price chart over some fixed interval, with the exchange rate recorded at discrete points in time.

The sensory inputs provided by the world at each timestep may be complete, as in the game of chess or Tic-tac-toe, where the position of every piece is known at all times – although the opponent's strategy is not necessarily known.

Other worlds may provide only a partial observation of the complete state, as in the game of poker, for example, where each player can only see a subset of the cards in play, including their own cards but not those held by other players [Russell and Norvig, 2003, p. 41].

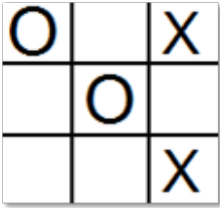

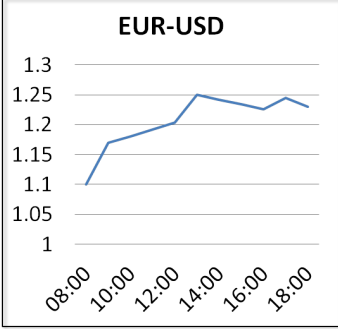
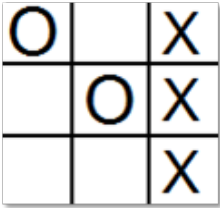

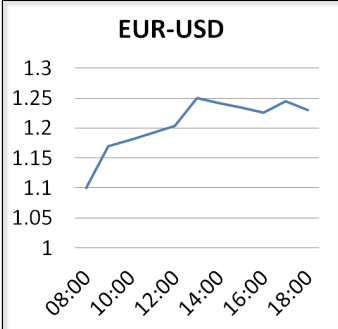
State			
Action	$[2,1]$	<i>Hit</i>	<i>Sell</i>
Afterstate			
Score	<i>Won in 3 moves</i>	<i>Bust: -\$10</i>	<i>Final profit: +155 pips</i>

Figure 2.1: Examples of the *state* information that might be presented by three different worlds, possible *actions* to take in each state, and a *score* summarising the overall outcome of each *run*. Also shown are the afterstates generated in the world by taking each suggested action (no effect is visible in the currency simulation).

## 2.4 Actions

By observing the current state of a world, an action may be chosen which will advance the world to a new state which is (hopefully) closer to the desired end goal than the previous state.

The action to be taken can often be expressed by a scalar value (for example, the symbols *hit* or *sell* in the blackjack game and currency exchange simulation – these could just as easily be replaced with a fixed range of integers; perhaps **+1** for *buy*, **-1** for *sell* and **0** for *do-nothing* in the currency simulation).

The action might also take the form of any integer quantity – for example, the currency simulation might interpret a positive integer  $N$  as a request to buy  $N$  units of an instrument, and negative value  $-N$  as a request to sell  $N$  units, and a value of zero to signify *do-nothing*.

Other worlds may require a multi-dimensional action (for example the player's next move **[2,1]** in Tic-tac-toe, although it might also be represented as a single

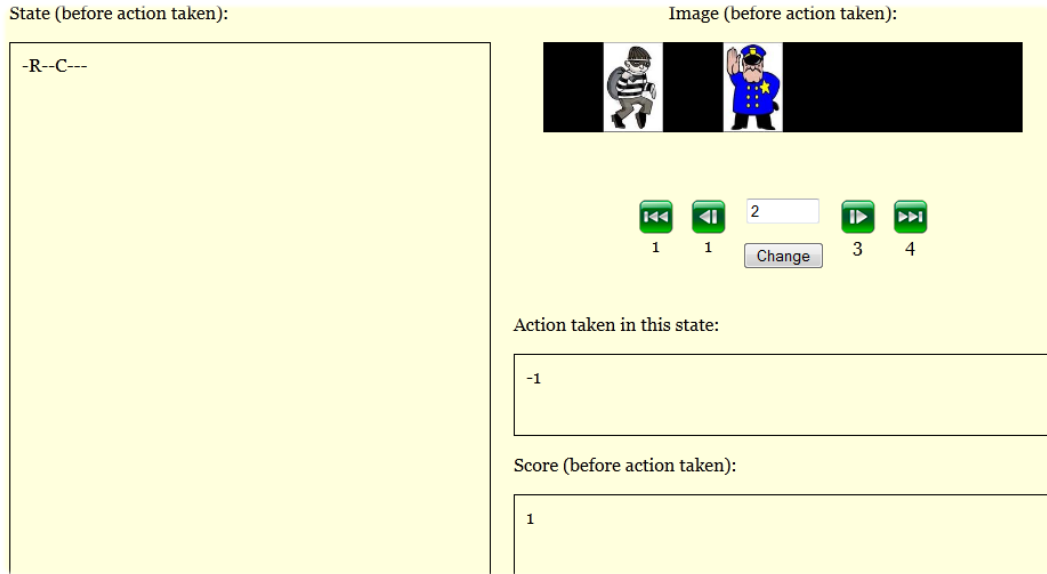


Figure 2.2: An image representing a single timestep in the Cops and Robbers world, a simple control problem. Users can freely create and upload new worlds to our server – this is one of many worlds provided and made available automatically. The mechanisms to facilitate this are described in chapter 5. This image shows a textual representation of the perceived state, as well as a dynamic graphical rendering of this state implemented by the world designer (this is the topic of appendix E). In the bottom right, the action taken from this state is shown (-1, which in this world signifies that the cop will move left and perhaps catch the robber if he happens to randomly move to the right), and beneath it a score of 1, which represents the number of actions the cop has taken so far.

integer  $x$  in the range  $1 \leq x \leq 9$ , or a selection from any nine discrete symbols).

And still other worlds might require actions in the form of a vector of continuous real numbers – perhaps representing applied forces or desired actuator rotations.

## 2.5 Scores

After each action is taken in the world, and at the end of a run, a scoring function is used by the world to generate a vector of score information, summarising various aspects of the run performed. Some example scores are given in figure 2.1, where a Tic-tac-toe game might end with “*X won in 3 moves*”, or perhaps simply “ $x,3$ ”. Similarly, a chess game might produce at the end a score vector of the form “*black,20*” indicating that black won in 20 moves.

On the other hand, the score information can be arbitrarily detailed since it is defined by the world author, and may include (in the chess example) details such as how many of the opponents pawns were captured, or how often each player was in



check.

It may be possible to gather or infer this meta-information by observing every state seen during a run and making a record of how often various events occur in the world. This requires that the important events are both observable by the mind and discrete enough to be easily measured and counted – for example, in a game of chess, if we know one of our pieces  $P$  was previously in position  $X$ , but on the next turn position  $X$  contains one of the opponent’s pieces  $Q$ , then we can deduce that our  $P$  was captured by the opponent’s  $Q$  in position  $X$ .

However, in a more complicated world, there may be many important events which are not recorded in the score vector and which are not trivial to detect from the observed state. For example, the observable state in a game of robot soccer might consist solely of raw data from unreliable ultrasonic sensors. The field of reinforcement learning (RL) is concerned with a related problem – learning an optimal control policy with only a scalar reinforcement signal for guidance<sup>1</sup>. This requires the ability to automatically discover which parts of the state are important and in which contexts, and to determine how what proportion of a reward or punishment should be assigned to recently-executed actions (known as the temporal credit assignment problem [Singh and Sutton, 1996; Sutton and Barto, 1998a]).

## 2.6 Goals

It is important to differentiate actions from goals. In some situations, multiple actions may satisfy the same goal – for example, when travelling from point A to point B, there may be many possible routes of equal length, so executing any sequence of actions that takes us to point B in the minimum number of steps could be said to satisfy the same goal.

Some problem worlds consist of multiple conflicting goals, where the problem-solving agent must prioritise the available options and act opportunistically where possible.

For example, consider a naïve approach to the game of chess, where we care about only two goals:

---

<sup>1</sup>In contrast to *supervised learning* methods, which learn from a series of known-correct input and output examples or *exemplars*.

1. *Protect my pieces.*
2. *Capture as many of the opponent's pieces as possible.*

It is not always clear when a particular goal should take priority when conflicts between them arise – it may be possible to capture more than one piece given the current state of the chessboard, and it may also be the case that capturing an opponent's piece results in one of the agent's own pieces being captured.

If the score information provided by a world is complete enough, it will provide enough data for the important goals to be inferred. By carefully examining the recorded score information, as well as the data about the states visited and actions taken, it is possible to estimate the relative importance of each goal and to identify the most successful mind programs for each goal. This is an important topic to which I will return in chapters 6 and 7.

## 2.7 Minds

A *mind* is an action-selecting program which interacts with an instance of a world in the manner described in section 2.8.

At each discrete timestep, an instance of a mind accepts a perceived state of the world, performs some computation and then returns a suggested action to be taken in the world.

It is important to note that a general representation of states and actions is not sought here – rather, it is left up to the author of a world to specify how states and actions should be described. Because the structure of the state and action representation is specified by each world, minds written for a particular world can pass the state information to another mind, possibly hosted on a remote machine, and receive a suggested action from that mind. Minds that call other minds to get suggested actions are called *hybrid* minds in this work, and the methods of their creation is the main focus of this thesis dissertation. When a hybrid mind **A** delegates to a mind **B** in this way, then mind **B** is called a *submind*.

## 2.8 The action-selection loop

An instance of a world will perform a loop, carrying out the following steps repeatedly:

1. At each discrete *timestep*, the world produces a set of sensory inputs, representing the subset of the current *state* visible to the virtual agent. This state is passed to the mind which controls the virtual agent in the world.
2. After passing the currently visible state to the mind, the world waits for the mind to present an *action* to be performed from the current state.
3. Carrying out the requested action takes the world to a new state, reflecting the changes caused by the mind's requested action, as well as by any simulated agents or phenomena modelled in the world. The process repeats in this fashion until a terminating state is reached, or some other arbitrary condition is met (for example, a predetermined limit of timesteps has been reached).

This type of interaction loop characterises the *action selection problem* [Maes, 1989].

In the real world, the world changes asynchronously and independent of any agent's action selection process. In *real-time* environments, time spent making decisions can have a significant impact on the situations that arise (inaction can be considered an action, in this sense). However, this need not be the case when working with simulated or artificial problems, although some limit must be set to avoid computational resources being wasted on requests which take excessively long or may never complete (for more details on the practical implications of this, see section 5.6.8.1).

In this work, real-time action selection is not considered – it may be an interesting possibility for future research due to its relevance in situated robot applications.

## 2.9 Hybrid minds

Because minds written for a particular world can query each other for suggested actions, it is possible to construct a large hybrid mind which consults several sub-minds, each embodying a particular expertise in solving the problem presented by the world.

In some cases, a hybrid mind might combine two subminds and perform better than each of its subminds might do individually.

For example, in the context of a chess world, mind **A** might be a master of the opening moves of a game, achieving and maintaining a strong position quickly, but might perform poorly in closing the game and achieving a checkmate.

On the other hand, mind **B** might perform poorly in the early stages of the game, but constitute an expert at achieving checkmate when the circumstances are suitable.

It stands to reason, then, that given this knowledge, one could create a hybrid mind **C** which has no logic or intelligence of its own, other than to determine whether the current state of the chess board is in the early stages of the game or in a situation where checkmate might be achieved. With some simple heuristics, the hybrid mind **C** might delegate to either submind **A** or **B** and achieve a success that neither submind could individually attain.

Although the subminds may not be written with the intent to be subminds, or to solve a specific subproblem of the world in a modular way, it may still be possible to analyse a collection of minds and determine which goals are most important, and which minds are best at fulfilling each of those goals.

## 2.10 Massively multi-author hybrid artificial intelligence

The primary focus of this dissertation is on the construction of large-scale artificial intelligence systems. This topic elicits several important questions, such as:

- How should such systems be built – all locally by one team in one research facility, or by a multitude of authors possibly worldwide?
- If such systems are to be built by geographically dispersed authors, then will they need to understand specific and detailed application programming interfaces (APIs) for every component of the system?
- Where and how should such a system be compiled or assembled?

This author proposes a minimal API where mind authors can utilise the expertise of minds written by other authors without needing to understand their inner workings

in great detail.

A centralised hosting of minds and worlds online is proposed, whereby anyone can add new worlds and minds and evaluate them at any time, and each mind is automatically made available for use in larger hybrid minds.

This dissertation describes a partially automated method of analysing a problem environment – referred to here as a world as defined above – so that an analysis and evaluation can be performed on mind programs which solve instances of the problem world.

With the help of this analysis, modular hybrid minds may be created that refer to multiple individual minds, taking advantage of their individual areas of expertise to better solve problems. These hybrid minds, assembled from the resulting set of *subminds*, are tested and demonstrated in this instance to perform better than the individual subminds.

To make it possible to gather a set of minds and worlds written by many different users, to evaluate their performance, and to use this collection of minds to create hierarchical hybrid minds, a supporting software system was developed, called the World-Wide Mind platform.

This platform extended and improved upon an earlier implementation (this will be described in section 4.1), automating the process of uploading new minds and worlds, significantly improving scalability and the speed at which runs can be carried out, and providing a means for worlds to generate graphical representations of each state which can be viewed in a web browser without requiring users to install software.

The technical challenges, design and implementation details of the World-Wide Mind platform are outlined in chapter 5.

## 2.11 Limitations of this framework

This high-level model represents problems and solutions as worlds and minds which interact in the ways described above, providing a powerful framework for expressing and solving many types of problems.

Can we fit *all* problems into this framework? No. Worlds which require human interaction at runtime (for example, an interactive chat bot) are not currently

supported by this framework, nor is it possible to run multiple minds together in a single instance of a world – something which would be an interesting possibility for future research into competitive game-playing strategies, for example. Similarly, the discrete timestep-based nature of the architecture makes it difficult or impossible to model continuous and/or real-time environments.

However, the framework attempts to make it possible to represent a broad range of problem types and solutions – for example, one student implemented and uploaded a 3D first-person shooter game as a world.

Chapter 8 identifies some of the technical limitations of the work and addresses some possibilities for further avenues of research.

## **2.12 Research gap: Automated hybrid building**

There is significant research into the development of models for cognitive processes, and of distributed, modular artificial intelligence systems, as will be discussed in chapter 3. However, little work appears to have been done on the subject of taking existing programs, built by many people to function individually, and assembling them together into larger “hybrid” programs. And more importantly, to achieve this hybrid-building through automated, or partially automated means.

To our knowledge, there are currently no other projects which attempt to build and evaluate large programs from many third-party subminds in the manner explored in this dissertation. This appears to represent a gap in previous research, where potentially fruitful results might be found.

To help address this gap, this dissertation attempts to develop a method of profiling existing programs written by many authors and selecting some of them to function as expert subminds in a hybrid mind program.

If the selection of subminds is performed adequately, then the set of chosen subminds will collectively represent the most important domain knowledge, and the resulting hybrid mind should be able to leverage the best characteristics of each submind and perform better than any of the individual programs (unless one of the programs is already an optimal solution to the problem – then the best the hybrid can do is to match that submind’s performance).

## 2.13 Research aims

In support of the research questions developed in section §1.4, we can divide the practical aims of this research into the following three areas:

- Firstly, an architecture and operational platform must be developed which is capable of hosting mind and world programs online as *services*, and which provides a framework for high-speed communication between these programs, either locally, on a single machine or remotely, across multiple machines on the Internet.
- Secondly, it seeks to establish *whether* hybrid minds can be built which consult existing minds and which can in some cases outperform those minds. Third-party mind programs submitted by users are gathered and their performance analysed to help answer this question.
- Finally, it asks *how* this building of hybrid minds can be done, and develops and evaluates a method of selecting a small number of task-specific experts from the many possible subminds which could be consulted for suggested actions. This is achieved through a statistical analysis of the performance of each mind in the world.

## Chapter 3

# Related Work

### 3.1 Introduction

This dissertation covers a range of topics. In this chapter, we will first look at similarities between this work and other research into cognitive architectures and modular intelligent systems, as well as the construction of programs as a composition of other programs. The idea of minds and worlds as services available on the Internet bears some similarity with the cloud computing and service-oriented architecture (SOA) domains.

As explained in chapter 5, the implementation of mind and world services as Java web services incurred a sizeable penalty on the throughput of messages. During the redesign of the messaging and communication architecture, several papers in the field of networked computer games provided some insight into the considerations involved, and some of these will be discussed here.

The development of a method for constructing hybrid minds – covered in chapter 6 and chapter 7 – requires an examination of the literature on statistical methods for determining the strength and direction of correlations between the recorded performance aspects of a mind program solving a problem and an overall success heuristic or quality function.

### 3.2 Modular intelligence

The concept of combining many weaker subprograms to create a strong hybrid mind bears some similarity with ensemble learning methods [Sollich and Krogh, 1995] in



machine learning. In ensemble learning, a set of weak learners are combined to make better predictions collectively. In this work, the subminds are programs written by humans which usually aim to solve the target problem on their own.

The learning element is a three-part process performed offline, consisting of:

1. an analysis of the individual performances of each mind,
2. ranking the minds using a variety of metrics, and
3. selecting a number of minds to use in the construction of higher-level hybrid minds.

### **3.3 Cognitive architectures and distributed minds**

A large body of work exists on the subject of computational cognitive architectures, spanning a variety of viewpoints from neuroethology to practical algorithmic methods, and several important contributions in the literature are discussed here.

Several cognitive architectures were proposed specifically to help answer questions in the analysis and modelling of human and animal behaviour. We do not propose a cognitive architecture in this sense, but rather a platform on which these cognitive architectures could be implemented.

However, there is some overlap in many of the ideas, and the architectures and research described in this section gave some insight into the basic principles underlying the project.

#### **3.3.1 The Society of Mind**

Minsky's Society of Mind [Minsky, 1986] presents and develops a wide range of ideas and theories of cognition. Among the many topics covered in Minsky's work are learning, language, short- and long-term memory storage and retrieval, and perception and classification of objects according to detected features, as well as often-debated philosophical issues, such as the nature of consciousness and whether it is an illusion.

Although the cognitive model developed in the Society of Mind is intended as a possible explanation for human intelligence, it provides some insight into the possibilities for future artificially intelligent systems. Most of the proposed mechanisms

are conceptual and abstract in nature, without being rooted in any particular model of animal neurobiology.

One idea introduced by the work is that while human memories may seem to use computer-like information storage and retrieval systems (albeit sometimes lossy and unreliable ones), the reality may be that recalling an experience or feeling involves replicating the “state of mind” we held at that moment.

As a possible theory of memory, Minsky suggests the *k-line*, or knowledge line. When a new event is experienced or a problem is solved, a new k-line is created to represent that experience. That k-line connects all of the mental agents which were active at that moment, and when re-activated at a later point in time, triggers those agents once more to recreate a similar mental state.

By way of distinguishing human intelligence from the intelligence exhibited by most other animals, Minsky suggests that *reflective* thinking – that is, a meta-cognitive process, or thinking about one’s *own thoughts* – is a capability possessed by humans which allows them to consider and modify their behaviour in ways not demonstrated by other animals.

Some elements of the work were inspired by observations and experiments in child behavioural psychology. It was also stimulated and informed by a successful effort to construct a physical robot capable of solving problems in the Blocks world (a world previously demonstrated in a symbolic context by Winograd’s famous SHRDLU natural language system [Winograd, 1972]), using machine vision to interpret images from an installed camera.

### 3.3.2 CogAff

One ambitious project called CogAff [Sloman, 2002] aims to understand and explain how cognitive architectures can work at many different levels of abstraction.

Cognitive processes are viewed as a spectrum of virtual machines in a *layered architecture*, operating in a hierarchical fashion. This layered view is a general theme shared by much of the literature on the subject of animal and machine cognition.

In CogAff, a high-level distinction is drawn, distinguishing different types of mental processes:

- **Reactive processes**, which are directly triggered when certain situations are

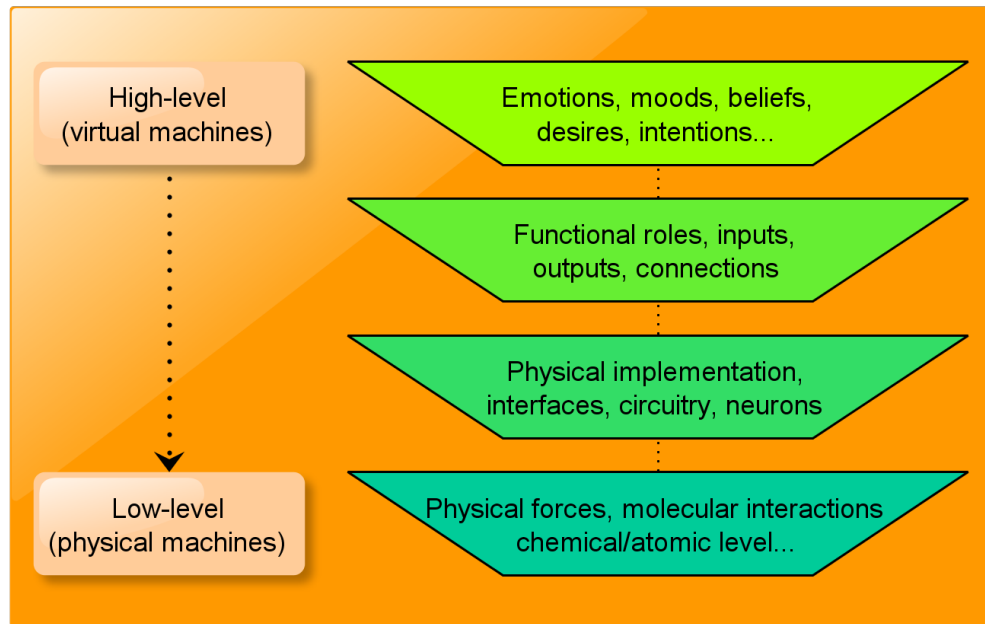


Figure 3.1: A layered view from the “virtual” level down to the physical machine level. The CogAff project tries to address processes which operate at different levels of abstraction.

perceived. In biological minds, these are frequently associated with ancient mechanisms developed by some evolutionary pressure (for example, a fear of snakes or spiders operating at the reflex level). Some animals – for example, insects or mold – might operate solely via reactive processes, without employing any higher-level cognitive mechanisms [Sloman and Chrisley, 2003]. However, many creatures following simple reactive behaviours can collectively produce complex behaviours, for example, when insects carry out very large group tasks such as nest-building.

CogAff describes an example “alarm” mechanism which provides a reactive capability.

- **Deliberative processes**, able to hypothesise and consider possible future events or actions. The costs, risks, possible gains and their likelihoods are estimated so that effective decisions can be made. By their nature, deliberative processes can be expensive in terms of time or computational resources. To help mitigate the possibility of a deliberative mechanism being interrupted repeatedly by perceptual inputs and alarms, CogAff proposes an attention filter mechanism with a dynamically varying threshold.

- **Reflective or meta-management processes**, which can understand and describe other internal states and processes. These processes can help detect and modify one’s faulty behaviours, noticing when one is “thinking in circles”. They might also provide an insight into whether the actions taken recently have served the desired goal, or indeed whether the desired goals need to be re-evaluated.

The general notion of these categories of mental processes has much in common with the ideas proposed in the Society of Mind and other works. Using these divisions, the CogAff generic schema is represented as a 3x3 grid, allowing mechanisms and mental states to be further categorised by their role in perceiving, producing actions or “central processing”.

Following on from CogAff, the H-CogAff project attempts to translate these insights into a model based on human behaviour and psychology. The CoSy and Cogx projects implemented the architecture in a physical robot.

The resulting model attempts to provide an implementation of affective states (emotions and attitudes) as well as both high-level goal selection (in H-CogAff, goals can be generated anywhere, from reactive processes such as hunger management to reflective processes like an ethical self-monitoring mechanism) and perception, and low-level action selection and motor control.

### **3.3.3 Unified Theories of Cognition**

Newell argues in [Newell, 1990] that a unified theory of cognition is of critical importance to large-scale A.I. systems integration, and that such a theory must provide a complete explanation of how intelligent creatures can react to sensory inputs (perception), how they select and fulfill goals – down to the level of motor control – and how they can learn and store knowledge about their world.

This last consideration follows on from earlier work on the physical symbol system hypothesis, an issue which has spawned considerable debate and opposing viewpoints.

### **3.3.4 The Soar architecture**

As an example model of cognition, Newell presents a symbolic cognitive architecture called Soar [Laird et al., 1987]. Underlying the Soar architecture is an assumption

that structured, symbolic representations of knowledge in various forms, as well as methods of reasoning with that knowledge are necessary for a complete spectrum of cognition and behaviour or “general intelligence”. For this reason, Soar provides several models of memory – working memory, as well as procedural, episodic and semantic long-term memories.

Helgason notes in [Helgason, 2013] that Soar is not designed for real-time operation, as it does not contain a central mechanism for focusing attention, yet it relies on synchronous processing of events.

More recent extensions to the Soar architecture allow for a form of sub-symbolic processing, intended for visual perception of object features [Lathrop and Laird, 2007].

### 3.3.5 CLARION

Another cognitive architecture, CLARION (Connectionist Learning with Adaptive Rule Induction ON-line) [Sun and Zhang, 2006] is designed primarily to explain human behaviour and psychological phenomena. CLARION draws a distinction between *implicit* and *explicit* processes. This dichotomy bears some resemblance to the fundamental division between subsymbolic and symbolic reasoning in artificial intelligence.

An example given of an explicit reasoning process is rule-based decision-making, an explicit process which is straightforward to implement in software.

Examples of implicit processes include similarity-based reasoning, which is presented as a fundamental way of thinking in human behaviour, and an associative memory which stores knowledge in a distributed way as a network of “microfeatures”, which link to symbolic or explicit “chunk” nodes representing higher-level concepts in another architectural layer. The notion of an associative memory composed of many distributed features is reminiscent of the “k-lines” proposed in the Society of Mind.

This architecture attempts to model other interesting types of mental phenomena, such as “discovery” tasks, where sudden insights are produced through gradual exposure to information. Modelling of this kind of effect can give an insight into the nature and role of intuition in human behaviour.

### 3.3.6 Global workspace theory

In cognitive science, the *frame problem* refers to the problem of determining what effects one's actions will have on the world without needing to explicitly model the effects those actions will *not* have [McCarthy and Hayes, 1969]. Shanahan proposes a cognitive architecture based on global workspace information flow theory, in part as a practical answer to the frame problem of cognitive science [Shanahan, 2006]. The architecture attempts to provide a working model of consciousness, emotion and imagination, and distinguishes between conscious and non-conscious information processing, having many cognitive processes running in parallel and competing to influence a constantly-running serial thread. Internal simulation is proposed as a mechanism for anticipation and planning processes, and affect plays a role in the selection of actions, both in reality and during internally simulated interactions with the world.

The architecture is implemented from a connectionist neuronal viewpoint, as opposed to some of the other cognitive architectures, such as Soar, which rely mainly on symbolic reasoning. The analogical function of neuron maps allows for more straightforward spatial reasoning than was afforded by traditional logic-based approaches.

Much of the architecture is inspired by an understanding of biological brains, with models of components such as the basal ganglia (for action selection) and amygdala (as an affect generator).

### 3.3.7 On knowledge representation and building interfaces

The cognitive architectures described above make assumptions about how information should be represented and communicated, and specify ways in which systems should be organised and what their responsibilities are.

However, it may be the case that with fewer requirements and less restrictions placed on mind authors, a greater level of participation and contribution from many authors will be facilitated, and this assumption underlies much of the work outlined in this dissertation and in the previous work discussed in chapter 4.

The approach taken in the World-Wide Mind is largely architecture- and implementation agnostic. One important goal of the W2M project is to minimise the technology barrier to entry for researchers, students and casual programmers to create

and test their own worlds and minds freely [Walshe et al., 2004] and to evaluate and reason about their programs. Consideration for the inclusion of complex mechanisms present in existing modular architectures (e.g. k-lines in Society of Mind [?], alarms and perceptual filters in CogAff [Sloman, 2002]) and decisions about knowledge representation is reserved for possible future work, if at all, as they would constrain the solutions provided by mind authors. Rather, world and mind designers are free to define the world state and action representations, as well as the interactions between subminds, in almost any way they see fit.

### **3.4 Other work on building modular intelligent systems**

Apart from the cognitive architectures described above, of which there are many more, there is a large body of research into other methods of building modular intelligent systems. A small selection of these are discussed here in brief.

#### **3.4.1 Hybrid minds created by O’Leary et al**

The work described by O’Leary et al in [O’Leary et al., 2004] was carried out using the World-Wide Mind v1.0 platform described in chapter 4, running a large set of subminds in an implementation of Tyrrell’s SE. This large pool of 505 potential subminds (submitted by 234 different authors) was narrowed down to a small set of candidates using several strategies.

First, a manual analysis of the top-scoring mind on the scoreboard was performed, creating a profile of what the authors believed were the most important aspects of that mind’s performance in the world (both positive and negative aspects were listed) as follows. According to the authors, the exemplar mind:

- Survived for an average of 3,500 timesteps
- Often slept to extend its lifespan (which may cause it to miss potential mating opportunities)
- Favoured drinking over eating (it doesn’t eat enough)
- Cleaned itself efficiently
- Did not suffer injury from animals or dangerous places

- Never used its “move fast” directional actions, which may imply that it didn’t need to escape predators.

Based on this profile of the top-ranking mind, a series of ten hybrid minds were created in a manner similar to the method proposed in chapter 6, but as a complement to what the authors considered weaknesses in the exemplar mind. In one hybrid mind, for example, they called the exemplar mind in all cases except when an “eater” mind suggested an eating action – where the eater mind was chosen by finding a mind which ate non-toxic food every time it selected an eating action.

For nine of the ten hybrid minds proposed, the hybrid mind controller is simple – it presents both the “main” submind and the specialist submind with the current state at each step, and if the specialist submind suggests an action directly related to its perceived expertise (for example, eating for the “eater” mind) then its action is taken, otherwise the main submind’s action is taken. The tenth hybrid was different, and instead implemented the “Drives” ASM, a simple algorithm which calculates the drive or motivation level for each of several systems [Hull, 1943; Tyrrell, 1993]. The system with the highest drive is selected at each moment, although as noted by Tyrrell in [Tyrrell, 1993], the method of combining multiple stimuli into a single drive variable for each system is not specified. This implementation of the Drives ASM did not produce a successful hybrid mind.

Two possible issues with this type of modular mind composition were identified:

1. Firstly, that unless all subminds are asked for an action in every state, some subminds will have an incomplete picture of what happened before they were called. This could have negative consequences in non-Markov worlds.
2. Second, if all minds *are* asked for an action at each timestep, then they may presume that their action was followed, when only one mind’s action can be taken (except when several minds suggest the same action). This can also lead to an incorrect understanding of the true world state.

The work by O’Leary et al differs from the methods presented here in several respects. First, most of the hybrid minds created consisted of a “default” submind and exactly one other mind which is called when certain conditions are met (for example, when the second mind suggests an eating action). However, some of the hybrid minds



they created were used as part of a larger, hierarchical hybrid mind – something not done in this research. Also, the methods used by O’Leary to select subminds were based on an ad hoc analysis of an exemplar mind, which may be subject to human bias. This potential for bias is also present in the method which will be suggested in chapter 6 of this dissertation, and is something which chapter 7 will attempt to address in a more systematic way.

### **3.4.2 Behaviour-based A.I. and the subsumption architecture**

At another extreme, compared to the general intelligence systems of Newell et al, is Brooks’ *subsumption architecture* [Brooks, 1990, 1991]. The work presents a case for modular behaviour-based artificial intelligence, built in a layered fashion of sorts from simple behaviour-generating components, but without using a structured, explicit form of knowledge representation.

In Brooks’ view of behaviour-based AI, raw sensor data directly represents the world state, without an explicit conversion into a series of symbols. This contrasts with the physical symbol system hypothesis espoused by [Newell and Simon, 1976].

The subsumption architecture was designed specifically with the intent that it be used in real robots, at a time when much of the existing work into building intelligent control systems was based on software implementations which operated on (perhaps unrealistically convenient) symbolic descriptions of the simulated environment. In fact, the robots produced from this research – starting with the wandering three-layered, sonar-sensing “Allen” – were some of the first successful robots capable of operating in the real world, reacting and moving more fluently than those which had come before.

It is possible for world designers, when creating worlds in the World-Wide Mind, to encode their states in a symbolic form (for example, in the Blocks world) or as a large vector of perceived sensor data, and minds are free to interpret or translate these percepts into other forms.

The fundamental idea explored in Brooks’ subsumption architecture – of many simple, modular agents seeking to control a single body – is one inspiration for this work, and it may be possible and fruitful to use a subsumption-type approach in designing the hybrid mind controller.

### 3.4.3 Behavior-oriented design

In behavior-oriented design (BOD) [Bryson, 2002] (related, but not to be confused with an earlier use of the term for a co-operative multi-agent arbitration system [Steels, 1994]), action-generating behaviour modules are combined with a dynamic planning system called POSH.

The planner arbitrates between the individual behaviours in the same sense that a hybrid program does in the World-Wide Mind. The dynamic plans offer a structured way to perform this run-time high-level behaviour selection.

As with Brooks' subsumption architecture, this type of approach to generating a modular problem-solving agent may be compatible with and complementary to the techniques described in this dissertation. A methodology like behavior-oriented design could be applied to develop the control system for a hybrid mind, after the individual behaviours which lend themselves to overall success have been identified and ranked by performing the analysis described in chapter 7.

### 3.4.4 Hierarchical Q-learning

Reinforcement learning is concerned with the problem of learning an action-selection policy from scratch [Russell and Norvig, 2003], with only a reward (or punishment) signal for guidance. A powerful reinforcement learning method called Q-learning [Watkins, 1989] was extended by Lin [1993] by training a set of learners, rather than one monolithic structure. In HQL, each learner focuses on a different area of the state-action space, thus modularising the problem and its solution.

The learning process happens online – that is, through trial and error in the world by suggesting actions and receiving rewards or punishments based on the immediate outcome and its expected future value. In contrast, the learning aspect of the work described in this dissertation happens offline when the performance of a collection of minds is analysed.

### 3.4.5 W-learning

In W-learning [Humphrys, 1995, 1997], the hybrid mind controller learns a policy for selecting among subminds (called *agents* in W-learning) rather than among possible actions. W-learning draws on ideas from Brooks' subsumption architecture and

combines the Q-learning [Watkins, 1989] online reinforcement learning algorithm with a series of strategies for agent selection and competition.

The agents in W-learning are a set of Q-Learners, each with its own reward function intended to facilitate learning a different desired expertise. At each timestep in the world, each agent produces a scalar value indicating the degree to which it wishes to take control of the creature. The controller arbitrates between the agents, and selects the agent which would be most unhappy if it were not selected, and the winning agent's suggested action is then accepted.

### **3.4.6 Constructionist Design Methodology**

Another approach to building large A.I. systems is the Constructionist Design Methodology (CDM) [Thórisson et al., 2004], which was implemented in the form of a virtual agent in an augmented-reality room equipped with wireless tracking sensors. In CDM, the emphasis is on the modular design of the complete system. The functionality of the system is broken into distinct functional modules with clearly-separated responsibilities, just as problems and solutions are tackled by the divide-and-conquer approach in software engineering. As a result, the methodology can be used to construct large systems which incorporate complex hierarchies of control and data flow.

Although a methodology like CDM could be applied to the construction of complex hybrid minds in the World-Wide Mind, an important issue we address is the ability to construct a hybrid mind from subminds which were not designed explicitly to co-operate or to embody separate, orthogonal functions within a larger solution.

### **3.4.7 Blackboard systems**

An early attempt at building modular intelligent systems was the blackboard system [Erman et al., 1980; Corkill, 1991], which was inspired by the idea of a group of human experts brainstorming together. The blackboard represents a shared workspace for partial solutions.

This high-level idea has been revisited in various systems since then, perhaps most relevant in the multi-agent setting described by Corkill [2003].

One advantage of the blackboard system approach is that it provides a richer, more strongly-defined structure for information sharing between the co-operating

modules known as *knowledge sources*. Also, the task of adding a new module to an existing blackboard system – perhaps dynamically – is more straightforward than with directly connected modules.

As with CDM however, it is necessary that each collaborating module in a traditional blackboard system be designed in such a way as to play a part in the collaborative activity. Furthermore, the system does not lend itself to an explicitly hierarchical organisation of behaviour-driven minds which is a fundamental aspect of this work, and it is unlikely to be of use where the knowledge sources – minds – return only actions, without supplementary meta-data which can be used by other minds and by the *control shell* (the hybrid controller).

However, it should be possible to implement a blackboard system as a hybrid mind, where modules are represented as named subminds with specially-designed `getaction` methods, and the blackboard is maintained by the hybrid controller and passed to the subminds in place of the world state (the blackboard contains the input state, as well as the shared repository of information created by knowledge sources).

Several cognitive architectures include blackboard systems as part of their design – for example, Ymir, an architecture which integrates multimodal perception and action, uses a blackboard as part of its distributed planning and control system [Thórisson, 1999].

### 3.5 Competition and collaboration in mind-building

Cristianini discusses the problem that research in artificial intelligence has become increasingly fragmented, with more work being done on narrow, focused topics and less integration of ideas into larger systems [Cristianini, 2010].

Some collaboration and sharing exists within artificial intelligence research; for example, a number of websites serve as repositories for machine learning code and training datasets [Kantrowitz, 2009; Asuncion and Newman, 2009]. These repositories are useful, but the steps required to install or adapt an existing solution differ in each instance and there is little consistency in the types of programs and interfaces provided.

Potential users must download the code and may need to modify it to compile on

their own machine. The program will generally require adaptation to suit the interface and/or problem structure they wish to solve, if indeed the program is suitable for addressing the chosen problem.

One project titled RL-Glue [Tanner and White, 2009] attempts to alleviate this problem by providing a common API, mainly focused on the reinforcement learning community. The interfaces allow agents and problem environments to interoperate, even if they are implemented in different programming languages. RL-Glue is explicitly focused towards reinforcement learning research, but could be applied in a more general sense.

Competitive environments such as RoboCup [Visser and Burkhard, 2007] and the DARPA Grand Challenge [Buehler et al., 2009] are interesting in terms of encouraging the creation of intelligent programs, but the problem domains are specific and there is no defined infrastructure for building and sharing hybrid minds.

### **3.5.1 Yet Another Robot Platform (YARP)**

YARP is a platform for building modular robot control systems, aimed at reducing the amount of infrastructure-level work that needs to be done in the process [Metta et al., 2006]. It supports the software development process by providing an application programming interface (API) oriented toward the building of humanoid robots, including a library of image processing functions and interprocess communication (IPC) methods for distributing processes across multiple compute resources.

The platform has been used successfully by several situated robots, including COG and Kismet [Fitzpatrick et al., 2008].

A strength of YARP is that it facilitates an abstraction between control software and the hardware on which it runs. This abstraction is one of the core themes of the World-Wide Mind, which not only aims for hardware independence but also imposes minimal constraints on state and action representation, and encourages a behaviour-based design of minds. All of these factors promote modularity and re-usability of programs.

YARP is capable of supporting “soft real-time” applications, something not considered in the design of the W2M.

One downside to robot-oriented platforms like YARP is the initial complexity

involved in writing one's first programs. YARP is written in C++, presumably for performance reasons, and does not provide simple high-level interfaces for getting started on a solution to a particular problem, for building hierarchical programs like the hybrid minds in W2M, or methods for easily evaluating and comparing the effectiveness of one's programs at solving that problem.

### **3.5.2 Robot Operating System (ROS)**

Another popular robot programming platform is ROS, which like YARP focuses on the creation of abstraction layers between software and robot hardware, although unlike YARP, ROS is not a realtime system [Quigley et al., 2009]. The ROS project is more community-oriented than YARP, and supports robot builders by providing a package management system for a variety of different tasks, such as planning, gesture recognition and simultaneous localisation and mapping (SLAM) [Engelhard et al., 2011].

ROS is very much focused on the development of real robots, and is not directly concerned with hierarchical control. However, it attempts to be architecture-agnostic, using high-level abstractions such as topics and services for information exchange between modules.

## **3.6 Cloud computing and the semantic web**

From a software engineering perspective, the encapsulation of minds and worlds as services on the Internet brings this work into the area of cloud computing and, more specifically, service-oriented architecture (SOA).

SOA includes a notion of composing services [Rao and Su, 2005], either statically or at runtime, based on various considerations.

In the original proposal for the World-Wide Mind [Humphrys, 2001b], these services were intended to be implemented as web services, communicating with each other through HTTP requests and responses. In this work, the web service-based communications scheme has been replaced with a simple messaging protocol over TCP, intended to reduce the latency of message transmissions between services (see section 5.6.4).

SOA generally relates to web services in particular, but the focus on interoperability, and the potentially distributed nature of mind and world services has not changed.

The idea of complex and intelligent agents that interact automatically and autonomously with services (and perhaps each other) over the Web [Berners-Lee et al., 2001] is not new. One possibility to achieve this is to have agents attempt to parse existing webpages which are written for humans. This involves complicated parsing and is prone to error, particularly when understanding of implicit context is required. The alternative is to augment the existing web with semantically meaningful markup intended for machines [Bryson et al., 2002]. This goal has perhaps not been achieved in a widespread sense to date, but the possibility for a hierarchical composition of agents on the Web is interesting and certainly overlaps some of the central themes of the World-Wide Mind.

### **3.7 Fast communication in networked computer games**

The development of multiplayer networked computer games spawned considerable research into the problem of synchronising the simulated state between a remote server and local client, both of which maintain an internal model of the world. The goal is for state changes to be transmitted rapidly and for the remote and local models of the state to be in close agreement, even in the presence of noise, unpredictable transmission latency, or data packets which arrive out-of-order or go missing entirely.

#### **3.7.1 Packet transmission protocols**

Distributed interactive simulations and real-time multi-player network games frequently use the user datagram protocol (UDP) [Gautier and Diot, 1998; Roehl, 1995] for network data transmission in preference to the more commonplace transmission control protocol (TCP).

The reason for this is that TCP provides a number of features which are intended to maximise reliability of message delivery, at the cost of message throughput – for example:

- automatically retransmitting packets which may have been lost

- generating and checking sequence numbers to ensure messages are delivered in order
- generating and testing checksums to protect against data corruption.

These features make TCP the protocol of choice for most purposes, but increase the transmission latency and reduce the rate at which messages can be transmitted. Because of this, some client-server implementations eschew the use of TCP in favour of simple UDP transmission schemes, either re-implementing the required features provided by TCP.

Other implementations allow packets to simply be lost or delivered out of order, moving the responsibility of re-requesting missing or corrupted data to the next level – the application layer. In some scenarios missing or corrupted data is ignored completely. This is particularly common in the delivery of streaming video or audio, where UDP is typically used as the transport protocol [Wu et al., 2001].

## **3.8 Techniques for synchronising state over unreliable channels**

Networked multiplayer games inevitably produce a measurable latency (or *lag*) between issuing a command or input and seeing that action applied in the game world. This latency must be minimised – at least in perceptive terms – for the player, who expects a game to respond to inputs in the same fashion and speed during multiplayer online play as when playing in single-player mode.

### **3.8.1 Latency and responsiveness**

For the player, then, there is a limited responsiveness threshold beyond which the game becomes frustrating or uncomfortable to play – particularly in fast-paced “twitch” action games like 3D first-person shooter games. Studies estimate this responsiveness threshold [Ferretti and Rocchetti, 2005; Beznosyk et al., 2011] to be between 100 and 200 milliseconds.

This level of responsiveness can be difficult to meet, and may be unachievable using TCP under certain conditions. The World-Wide Mind was not intended to



address this problem, since the player is an artificial mind which may itself be hosted on a remote server, or even composed of many subminds on different remote servers. However, overall throughput and completion time for runs is of concern, especially when large, distributed hybrid minds are involved.

### 3.8.2 Hiding the effects of latency

Since the end-to-end connection latency between server and client cannot be completely eliminated, or reduced beyond what the laws of physics allow, a number of techniques have been devised for hiding some of that delay from the player, by allowing the local client to modify its internal state representation in accordance with the game model. Errors – that is, divergences from the server’s canonical model of the simulation state – are corrected through the use of synchronisation algorithms which reconcile those differences in client and server state based on newly-received data.

In 1983, a DARPA research project named “SIMNET” [Kanarick, 1991] was initiated, which described and implemented an architecture for distributed interactive simulations (DIS), to be used in collective training exercises for U.S. Army soldiers, as well as in testing new combat vehicle designs.

Due to the focus of team training in SIMNET, the distributed aspect of simulations was a primary concern, and this led to the development of various techniques for improving networked performance and hiding latency from the human user. An important observation for the purposes of network performance was that, for most simulations, the changes in state from one moment to the next are generally minute, compared to the complete state description. Because of this, it is more bandwidth-efficient to avoid transmitting redundant data, by sending only the *differences* between the old and new state.

Another technique used in SIMNET was *dead reckoning* [Calvin et al., 1993], which allows local clients to extrapolate new states between authoritative state updates. To enable this, simulated world objects must carry a model of their behaviour which can be used on the client side to simulate new states, and which is consistent with the world model used by the server.

Information in the state updates, such as position, velocity and orientation are

used to inform the local models so that objects can simulate their movement and behaviour within a certain degree of error. When this error threshold is exceeded, the server sends an update packet with an authoritative definition of the object's current and true attributes.

Although the use of dead reckoning is limited by the fact that it must be designed into each world model, it nevertheless achieves very good responsiveness, and has become a widely-used technique in multiplayer online games.

A simple scheme called *local lag* [Mauve et al., 2002, 2004] strikes a balance between responsiveness and consistency of state, by deliberately inserting a delay after each game event or user command is received. This provides some time for incoming events to be re-ordered such that fewer inconsistencies occur.

However, since it is still possible for a game event to be received after the delay period, this method must still apply a rollback and re-apply scheme to correct errors when they do occur.

Some methods are designed to execute events without being sure that earlier messages are not due to arrive, and in the event that out-of-order messages are detected, the inconsistencies in state are repaired. These algorithms are called optimistic synchronisation algorithms.

One well-known technique arising from distributed systems research is the *Time Warp* mechanism [Jefferson, 1985], which saves a copy of the current state after each simulation event is received. When it is discovered that an event has been processed before an earlier event is received, the state is rolled back to the latest copy before the delayed event, and then all events received since then are re-applied in order to generate a consistent state. Additionally, “anti-messages” are broadcast to cancel any events that might have been issued while the simulation was in an inconsistent state.

The problem with Time Warp is that the state snapshots are recorded for every game event received – which might be tens of times per second, for every connected client. This makes it unsuitable for massively-multiplayer real-time online games.

To address this problem, a method named *trailing-state synchronisation* was presented in [Cronin et al., 2001], which requires that each game state server (of which there could be several, in a redundant, mirrored configuration) keeps multiple copies

of the game state, each one corresponding to a different point in time. The leading state (events occurring right now) is compared with these older state snapshots to identify inconsistencies. If an error is detected, the state can be rolled back to an earlier, consistent copy and events and user inputs re-applied to bring the old state up-to-date.

### **3.8.3 Limitations of latency-hiding techniques**

These techniques are powerful and can make the difference between a simulation being perceived by the human participant as responsive or completely unplayable. While artificial minds have no such responsiveness demands, it may be the case that some of these optimisations could improve the overall execution time of a run across the network.

However, each of the techniques requires that the world model be modified (for example, to provide the basic facility to roll the world back to an earlier state and re-apply a stream of updates) and for dedicated client code to be introduced which provides a limited, local model of elements in the simulation, so that intermediate states can be extrapolated while waiting for an authoritative state update from the world, as in dead reckoning.

Furthermore, since these techniques rely on client-side approximations of the world, it would be difficult or perhaps impossible to provide a deterministic interface between minds and worlds. For this reason, none of these techniques has been applied to the World-Wide Mind architecture.

However, future work might benefit from a further examination of these techniques, for certain classes of world which could declare themselves to be tolerant of “lossy” interactions.

## **3.9 Statistical methods**

The field of computational statistics has grown so large and broad that no treatment here can give more than a superficial examination of a small number of methods among many techniques which might be useful for our purposes.

Chapters 6 and 7 discuss methods for determining which of the attributes making

up a score vector gives the strongest contribution to the overall, aggregate score.

One way to model and understand this relationship is to perform correlation analysis on the collected data. Correlation is essentially the problem of determining the degree to which two observed variables seem to be associated with one another [Lowry, 2010], and it can be performed very quickly using several different methods.

Principal component analysis [Jolliffe, 2002] has been used for dimensionality reduction in fields such as image compression [Du and Fowler, 2007], as well as for variable selection [Westad et al., 2003].

### **3.10 Conclusion**

This chapter has touched on a wide variety of subjects, encompassing different areas of research from distributed artificially intelligent systems and their construction, to techniques for fast updates of shared state models over unreliable network connections in multiplayer computer games and military simulations.

## Chapter 4

# Previous Work: the World-Wide Mind, version 1.0

To help place some context on the work carried out during this period of research, this chapter introduces and describes the previous work on the World-Wide Mind project. First, the initial motivations behind the project are discussed, before the basic communication model and high-level system architecture are introduced.

After this, a technical overview is presented of the important system components, focusing on the local user interface for controlling runs in any world, and the server-side mind and world hosting mechanisms. Finally, the chapter finishes with a look at some of the limitations to this previous work which the contributions described in chapter 5 attempt to address.

### 4.1 The World-Wide Mind

Although there has been a great deal of research into specific techniques for problem-solving and decision-making in the broad field of artificial intelligence, there has been less integration of diverse methods together, and a certain degree of Balkanisation in research focus.

The World-Wide Mind (W2M) project aims to encourage greater cross-pollination and diversity of problem-solving programs, while keeping the entry barrier for participation low. The project is an attempt to scale up artificial intelligence by distributing problem-solving programs (which we refer to as “minds”) and problem en-

vironments (which we call “worlds”) on the Internet, and by allowing minds to call other minds freely and thus facilitate building *hybrid minds* from multiple minds which may have been written by multiple authors, perhaps with no specific intent for collaboration or reuse.

The project was started in 2001 to facilitate the integration of many diverse components of agent minds into whole minds [Humphrys, 2001a], evolving out of previous work in multi-mind intelligence [Humphrys, 1997].

## 4.2 Communication model

In this “1.0” design of the World-Wide Mind platform, minds and worlds are represented as web services, available on the Internet [O’Leary and Humphrys, 2003]. The interactions described in chapter 2 were accomplished by sending messages embedded in web requests to mind or world services, and receiving reply messages in response [Walshe et al., 2004].

For example:

- The **getstate** request message, when sent to an instance of a world service, prompts the world to return the state (or an observable subset of the complete state) in its response.
- Similarly, the **getaction** request, when sent along with an observed state **s** to an instance of a mind service, prompts the mind to return an appropriate action to be taken from that state.

In this sense, the aim was to make possible a genuinely “world-wide” mind system, where anybody could set up a server for their minds and/or worlds, assigning to each one a unique URL.

Once a mind had been created and hosted as a mind server somewhere on the Internet, it could then be used by a hybrid mind through this URL. This opened up the possibility of building distributed hierarchical hybrid minds online, each composed of a multitude of mind programs written by multiple authors.

This notion of having large hybrid minds with components located all around the world was one of the motivating ideas behind the work.

## 4.3 Architecture of the World-Wide Mind v1.0

### 4.3.1 Software architecture

At a high level, the system architecture of the World-Wide Mind v1.0 consists of two main components::

- the user interface (client),
- the world and mind services, hosted by a web application server (Apache Tomcat, in our case).

#### **The user interface**

The local client program, shown and described in section §4.6, allows users to initiate and manage runs. The interface allows runs to be carried out and stepped through, displaying the state and action taken at each timestep in the world.

#### **The mind and world services**

It was originally envisaged that minds and worlds would be implemented as stand-alone web services, hosted on many different physical server machines by their authors.

An obvious use of the architecture was for teaching rather than research. To require that students deal with the issues involved in bringing online and maintaining their own web servers before they could even start writing their minds seemed needlessly onerous.

A simple web application server was therefore set up to host minds submitted by those students who did not wish or know how to implement their own servers. Minds were submitted by users manually, via email to the server administrator. A series of batch scripts were then called by the server administrator to produce a web service from the submitted mind class. This process is discussed in further detail in section §4.8. Scores from each run were collected manually and a HTML scoreboard was updated using other batch scripts by the server administrator.

Although this reference server was implemented using Java and servlets, the architecture itself is language-independent, since messages are sent between services

over the Internet as XML. Minds and worlds can be hosted by any technology which allows the creation of web services capable of constructing XML messages and sending/receiving them as parameters to HTTP requests.

## 4.4 Interaction of mind and world

When an instance of a world and mind are created, a series of interactions is performed in a continuous loop, until the *run* terminates (for example, when a chess game is won, drawn or lost, or when a simulated animal dies or achieves its goal). At the highest, most simplified level, this action selection loop consists of three steps:

- **getstate**: The mind observes the currently visible state **s** of the world.
- **getaction(s)**: The mind selects an action **a** to be taken in response to the current state and presents it to the world.
- **takeaction(a)** : The world applies the selected action **a** and updates its internal state.

From the perspective of the controlling entity which initiates the run, which we call the *client* (for example, the Java AWT program depicted in figure 4.3), the basic series of interactions is as follows, with the main loop (steps 3-5) shown in figure 4.1.

1. The client sends a **newrun** message to the world, which creates an instance of the problem world unique to this run.
2. The client sends a **newrun** message to the mind, which creates a mind instance unique to this run.
3. The client sends a **getstate** message to the world instance, which responds by returning the current observed state of the problem world. The observed state may be partial (e.g. a simulated animal's sensory percepts, or the visible cards in a simulated poker game) or complete, as in a game of chess.
4. This observed state is passed to a mind in a **getaction** message, which returns a suggested action **a** to be performed in the world (for example, "*0.8157*", "*Bxc2*", "*turn-right*" or "*place the red block on top of the leftmost green block*").



5. Action  $\mathbf{a}$  is passed to the world in a `takeaction` message, which returns the new state generated by applying the given action to the current state.
6. If the run has completed, then the world returns an `endrun` response instead, including the score information which summarises the mind's performance on this world instance.
7. If the client received an `endrun` response from the world, then it sends an `endrun` request to the mind and terminates the loop. Otherwise, it received the new state generated by taking action  $\mathbf{a}$ , and continues the loop from step 4.

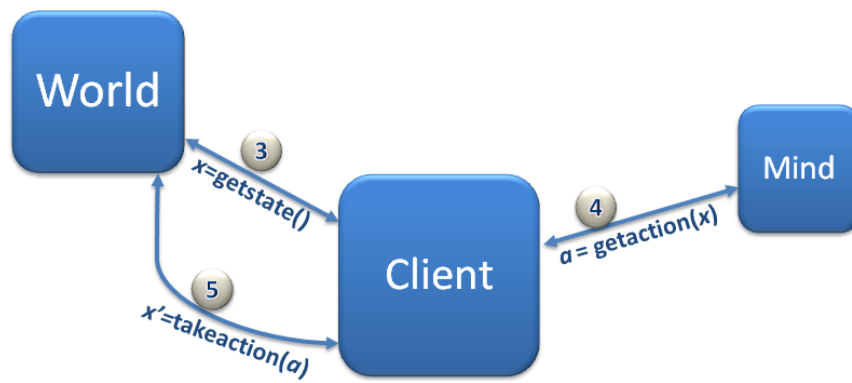


Figure 4.1: The basic interactions between the client, mind and world at each timestep. Once the run has been started in the run, the client interface – a program running on the user's machine – issues requests to and receives responses from one world service and one or more mind services. Each service may be hosted on a different physical machine on the Internet. The numbers correspond to the interactions discussed in section 4.4.

## 4.5 Society of Mind Markup Language (SOML)

In order to communicate with a mind or world service – possibly hosted on a remote server – a message protocol is required. An XML-based description language was developed to represent the messages which would be sent from the user's machine (perhaps running the client program described in section §4.6) to the intended world or mind services, and the messages received in response, as well as messages which might be sent from one service to another. This description language was called Society of Mind Markup Language, or SOML.

An example SOML excerpt encoding a **getaction** request message is displayed in figure 4.2.

It was envisioned that users could create web forms for the purpose of generating custom messages, since they were to be transmitted as HTTP requests and responses. These custom messages could be used to represent knowledge or actions in a mind-specific way, but the same information could be attached to a standard message such as **getaction**.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <soml version="1.0" xmlns="http://w2mind.org/soml"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation=
5     "http://w2mind.org/soml_1_0.xsd" http://w2mind.computing.dcu.ie
6     <getaction type="request" runid="1354563031043">
7     <state>
8     12345,empty,empty
9     </state>
10 </getaction>
11 </soml>
```

Figure 4.2: An SOML message sent to a mind service. This message represents a *getaction* request, presenting the mind with an observed state of “12345, empty, empty”. The “runid” parameter to the *getaction* tag represents a unique identifier, allowing the user to communicate with an individual instance of a mind which may have its own internal state.

## 4.6 Client interface

To help initiate and control runs, a graphical client program was provided and executed on the user’s local machine, acting as a middleman which sends messages to the world and mind services and managed the responses. The client program is shown in figure 4.3.

## 4.7 Software API for writing minds and worlds

A simple application programming interface (API) defined several interfaces and base classes which new minds and worlds extend to provide their own functionality.

Algorithm 4.1 shows a trivial mind which takes one of two possible discrete actions based on the visible state and an internal counter initialised at the beginning of the

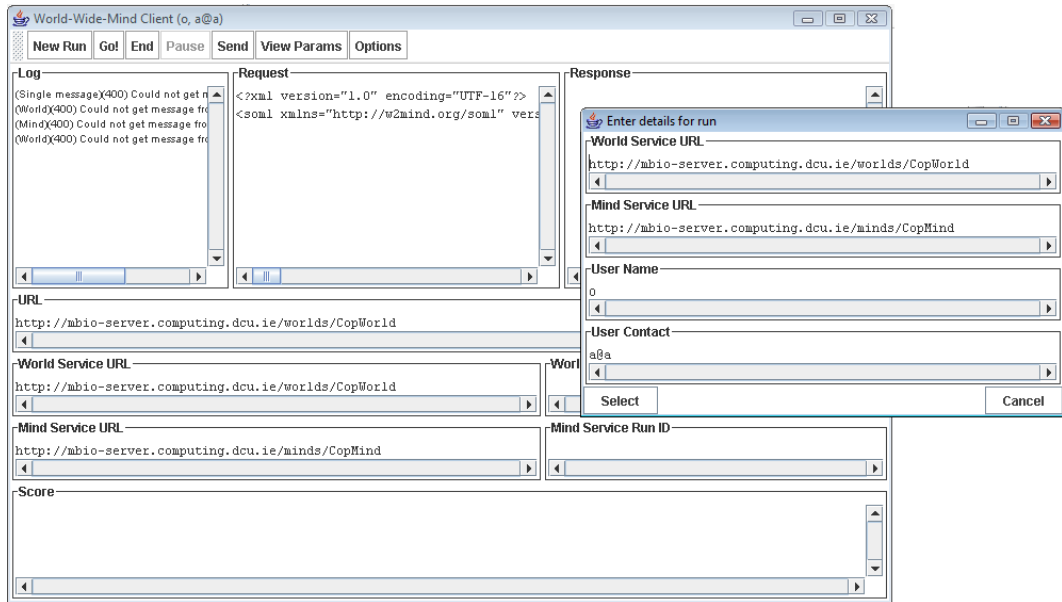


Figure 4.3: The W2M 1.0 client interface, a Java AWT program written to initiate and manage runs of mind services in a world service. The services are specified by URL, since a central theme of the project was to make possible the building of hybrid minds from various submind services, possibly hosted on different machines on the Internet.

run.

## 4.8 The W2M 1.0 service platform

A software platform (which is referred to here as W2M 1.0, so as to distinguish it from the newer W2M 2.0 platform described in chapter 5) was developed that would host Java servlets representing mind and world services. Users submitted mind classes by email to the server administrator, who then called a series of batch scripts which performed the following functions:

- compile the Java source code for the submitted mind, producing bytecode class files, and
- generate servlet wrapper classes, derived from the mind name. The servlet wrapper classes implement HTTP request methods, and install the newly-generated servlet in a directory visible to the Apache Tomcat web application server.

Once the servlets were hosted by the application server, communication between minds and worlds was then possible by sending HTTP requests; effectively converting

---

**Algorithm 4.1** An example of a simple mind.

---

```
import org.w2mind.net.*;
import simpleworld.*;

public class SimpleMind implements Mind {
    int counter;
    public void newrun() {
        counter = 0;
    }
    public Action getaction(State s) {
        TrafficLightState ts = new TrafficLightState(s);
        if(ts.isLightGreen() || counter++ > 30)
            return Actions.CROSS;
        else
            return Actions.WAIT;
    }
}
```

---

minds and worlds into web services.

This scheme was flexible and convenient, since Tomcat takes care of network I/O and automatically loads the appropriate classes when a servlet receives a request, and fits the motivation that difficult network programming should be unnecessary for a user to construct mind or world services and to make them available online.

#### 4.8.1 Message encoding, transmission and decoding

When a message is sent from the user to a mind or world service (or from one instance of a mind to another), these steps are followed. We will call the sender of the message request the *originator*, for clarity:

1. The message from the originator is encoded into an XML document containing:
  - (a) The message type (for example, *newrun* or *getaction*).
  - (b) A flag indicating that the document represents a *request* message.
  - (c) All the relevant data fields – for example, the perceived world state as a string of text.
  - (d) The *run ID*, or run identifier, which was returned from the remote service after it was initialised by a *newrun* message. The run identifier is a token which allows the appropriate instance of the remote service to be retrieved, since the communication channel is based upon stateless web requests.

- i. If this message is a *newrun* request, the remote service will generate a new run identifier and return it embedded in the response message.
2. The resulting XML document is passed as a parameter in a HTTP request to the web service specified by the target service URL.
3. The remote service handler – in this case a Java web servlet wrapper class – retrieves the XML document from the HTTP request parameter.
4. The remote service handler parses the received XML document and extracts all the previously-specified data fields, constructing a new message object.
5. If a run identifier was included with the message, the remote service handler uses it to retrieve the intended instance of the remote service (a mind or world) which was previously created and which should receive the message. In this case, every instance is retrieved from disk when the message is sent to it, and written to disk after the message has been handled.

To do this, an operation called *serialisation* is performed, which recursively “flattens” an object or data structure in memory and produces a string which can be sent across a network connection or stored to a file or database.

This string, when later deserialised, should produce an object identical to the one which was originally serialised.

- (a) If no run identifier was included, and the message is a *newrun* request, then a new instance of the remote service is created and assigned a new run identifier, which will be returned embedded in the response message for use in future requests.
6. The message type is used to determine the appropriate method of the remote service instance to call. If there are any relevant data fields (for example, containing the perceived state if the message is a *getstate* request), then they are passed as parameters to the selected method.
7. The remote service instance – an instance of a mind or world – carries out the required processing, and may or may not produce a result. For example, as can

be seen in algorithm 4.1, handling a *newrun* message does not return a value, but a *getaction* message does.

8. The remote service handler adds the returned value, if any, to a new message object. This message is once again serialised to an XML document containing:
  - (a) The message type (for example, *newrun* or *getaction*).
  - (b) A flag indicating that the document represents a *response* message.
  - (c) The value (if any) that was returned by the remote service instance.
  - (d) The newly-generated run identifier, if the message was a *newrun* request.
9. The remote service handler serialises the service instance to disk, using the run identifier as a key, so that it can be retrieved when more messages are sent to it in the future.
10. The resulting XML document is returned as a HTTP response to the originator's request, completing the exchange.

It should be clear that this process adds a significant overhead to the time taken for every exchange of messages. This is especially noticeable over long runs, which may result in many thousands of messages being sent and received.

## 4.9 Limitations

The first implementation of the World-Wide Mind platform was capable of interfacing minds with worlds and with other minds over the Internet, but performed very slowly. Because of this, when the system was used for undergraduate assignments in artificial intelligence courses, the majority of submitted minds were monolithic programs which did not seek the advice of other minds to select actions.

This section describes some of the limitations in the previous design and implementation of the World-Wide Mind 1.0 platform which motivated the work described in chapter 5.

To make the building of large hybrid minds a more plausible exercise, improving the latency and throughput of messages was of major concern. Section §5.6 describes what was done to minimise message sending overheads and increase the speed of runs by several orders of magnitude.

### 4.9.1 Minds and worlds as web services

In the initial W2M implementation, minds and worlds were represented as web services, so that messages could be sent to any service using a simple HTTP request. This afforded some simplicity and transparency in distributing minds and worlds on the Internet, and left the door open for interacting with mind and world services manually by constructing HTTP requests in a web browser without needing special software, although it was more practical to use a dedicated client program.

However, there is a time penalty to be paid for this flexibility, in terms of increased latency between sending a message and receiving a response back, even if minimal processing is carried out by the mind or world service.

The following factors contribute to this latency:

- The use of a web application server (Apache Tomcat), which introduces computational and I/O overheads for features beyond those necessary for our purposes, for example, authentication checking, filter chains and logging.
- Every time a message is sent, it is encoded into a new XML document. This could include very large data structures, such as the current state of the world which might consist of a large number of recorded sensory perceptions.
- Similarly, upon receipt of an XML document it must be decoded to reconstruct the original message. This repeated encoding and decoding of messages into XML is computationally expensive.
- Using HTTP to wrap messages adds extra message headers and causes additional network transmissions.
- Each run consists of a unique mind and world instance interacting together for a series of timesteps. Because HTTP requests are inherently stateless, it was necessary to persist the mind and world instances to and from disk storage, using a unique identifier to retrieve and store the correct instances at every timestep.

This process of serialising and recreating objects is expensive in terms of disk access and computation time, since it happens so frequently. It would be faster to store the world and mind instances in a hash table, but this could lead to

excessive physical memory use – a potentially unbounded amount of memory if runs do not terminate correctly and are not removed from the table.

Together, these factors add up to a significant latency overhead for every message sent and every response returned. However, some of these sources of latency are avoidable in certain situations, and some can be eliminated entirely. Perhaps most importantly, the serialisation of world and minds at every timestep which is very costly.

## 4.10 Conclusion

Previous work on the World-Wide Mind project produced a server architecture which represented worlds and minds as web servlets, each associated with a run by a run identifier.

This system worked, but due to the design of the Tomcat application server, which offered many capabilities which are unnecessary for our purposes (such as filter chains, authentication etc), and inherently due to the wrapping of all messages in XML and then transmitting them over stateless HTTP, the performance of the system was rather slow – on the order of one second of overhead for every timestep.

These limitations and some resolutions applied in this work are addressed in section §5.6, which describes the improvements made to the World-Wide Mind architecture and implementation that enabled the research into hybrid mind creation presented in chapter 6 and chapter 7.



## Chapter 5

# W2M 2.0: An architecture to enable massively multi-author hybrid intelligence

### 5.1 Introduction

Chapter 4 discussed a software architecture and platform – which we call the World-Wide Mind 1.0, to distinguish from this work – which was intended to make possible the building of mind programs from a widely distributed set of submind services. The 1.0 platform was implemented mainly by Ciarán O’Leary [O’Leary et al., 2004].

To address some of the limitations which were identified in section §4.9, this author designed a “World-Wide Mind 2.0” platform, building on the earlier work and integrating with a front-end interface created by Mark Humphrys and Brian Monks [Monks, 2010]. The work described in this chapter is an original contribution by this author, except for the front-end interface by Humphrys and graphical rendering system by Monks, a brief description of which can be found in appendix D and appendix E. Together, these contributions form an entirely new system called W2M 2.0.

This chapter outlines the technical challenges and design of the W2M 2.0 software platform and architecture and evaluates the contributions made by this work.

## 5.2 Purpose

The World-Wide Mind project was developed with the intent of hosting minds and worlds as network services on a potentially large scale, and allowing these services to communicate across the Internet.

In the W2M 1.0 model, it was assumed that users would generally host their mind or world web services on their own servers, although a reference server was developed capable of hosting a limited number of minds and worlds as Java servlets. The Java programming language was chosen for several reasons: most importantly, it is the language with which most of the immediately available participants (undergraduate students) were familiar. It also enjoys considerable cross-platform independence and is well-supported and mature.

The W2M 2.0 model moves away from the model of individually distributed minds and worlds, and towards a model whereby automated World-Wide Mind servers each host a large number of mind and world services. These services are uploaded to a server on the Internet by untrusted users, and must be managed carefully to minimise the risk of damage from poorly-written or malicious programs. Considerations regarding security issues are treated in section 5.6.8. The move toward a more centralised model is discussed in section 5.6.1.

It is also important that the communications architecture and implementation allow messages to be transmitted between mind and world services with as little delay as possible, to allow problems and solutions to be tested and evaluated quickly, over many runs. This latency was one practical limitation in previous research which is addressed in this work.

Increasing the speed at which messages can be sent and received will make it possible and practical to build hybrid minds which communicate directly with other minds for suggested actions, as described in section §2.10. Details of the communications implementation are discussed in section §5.6.

To encourage more contributions and collaboration, a front-end web interface was developed with certain requirements chosen such that only a minimal level of commitment was required of users to write, submit and test their minds. Firstly, it must be possible to upload and test minds and worlds automatically, without admin-

istrator intervention as was necessary in the earlier proof-of-concept implementation [O’Leary and Humphrys, 2003]. The front-end interface was developed by Mark Humphrys with contributions by Brian Monks [Monks, 2010], and its features will be discussed in the context of this work in appendix D, with a description of the graphical rendering of world states in appendix E.

This facility is supplemented with an automated scoreboard system which ranks the best performance scores of the uploaded minds for each hosted world. The web interface provides a facility to explore all of the minds and worlds hosted on a server, to upload new minds or worlds and delete old ones (provided the user has appropriate ownership or permissions), to view the performance ranking of minds for a particular world and to carry out runs of any mind in its associated world environment.

Where the world author has implemented the appropriate method, graphics can be generated during a run, and displayed as the user steps through the list of states seen and actions taken. To view the scoreboard and carry out runs of hosted minds, all that is required is that the user has a working web browser.

### 5.3 Requirements

Following on from the discussion of the W2M 2.0’s purpose, the following list summarises the primary requirements of the W2M 2.0 which are relevant to this work (front-end elements are excluded as they were contributed externally):

1. A server daemon should be created which can automatically load a Java archive (JAR) file containing classes constituting either a world or mind.
2. The server should make loaded worlds and minds available as a service on the Internet.
3. Requests destined for any hosted world or mind service should be passed to the appropriate service and responses returned, creating a new instance of a service if necessary.
4. It should be possible to perform a run of a mind in a world rapidly online, with minimal overhead in message decoding or processing.

5. A carefully designed security model should prevent broken or malevolent services from damaging the system or other services.
6. A generic scoreboard system should be designed which can persist to a database each mind's best score in a world. This is intended to encourage competition between mind authors.
7. It should be easy to create a new hybrid mind which calls other minds, possibly written by multiple authors. This should be possible when the hybrid is running on a W2M server or when run locally; otherwise it would be difficult to test and debug programs.

By fulfilling each of these requirements, we can then answer the first research question posed in section §1.4. The remainder of this chapter describes the general architecture decisions made to support the W2M 2.0 platform, and how each of these requirements is satisfied. As far as was practicable, the behaviour of every programmed feature or optimisation was verified through the use of unit tests, and the performance gains (or losses) were checked through the use of targeted benchmarks.

## 5.4 Architecture of the World-Wide Mind 2.0

At a high level, the World-Wide Mind architecture consists of three components whose interactions are summarised in figure 5.2:

- the run logger,
- the user interface (client), and
- the W2MServer daemon program.

### 5.4.1 The run logger

A Java program called *Runlogger* manages the communication between the user and a mind. An instance of this program is created when the user starts a run from the web client, receiving as parameters the URLs for the selected mind and world services.

The program then starts to communicate with the mind and world via TCP network sockets (although communication with the world now happens indirectly as described in section 5.6.5).

As the run progresses, updates containing the states seen and actions taken by the mind are collected and added to an XML logfile, which is then parsed and presented by the web client upon completion of the run.

#### 5.4.1.1 The run logger XML log

The structure of this XML document is explained with the following example. The entire document is enclosed within a `<soml>` tag:

```
<?xml version="1.0"?>
<soml xml:lang="EN">
  ...
</soml>
```

Contained within the soml tags are an ordered series of `<asynccrun>` tags, describing the run. The first of these `<asynccrun>` tags contains metadata describing the parameters of the run:

```
<asynccrun recipient="ImageMind">
  <imagesdesired value="true"/>
  <otherparticipant value="ImageWorld"/>
  <runid value="1347293678374"/>
  <steps value="0"/>
  <world value="ImageWorld"/>
  <worldHost>som://mbio-server.computing.dcu.ie</worldHost>
</asynccrun>
```

The meaning of these parameters is as follows:

**recipient** The name of the mind we wish to carry out the run.

**otherparticipant, world** The name of the world in which the mind will perform the run.

**worldHost** The URL of the server which hosts the desired world service.

**runid** An automatically generated run identifier returned by the mind service, associating this run with a particular instance of the mind.

**steps** The maximum number of timesteps we wish the mind to take in the world – once this number is reached, the run will be terminated. A value of zero signifies no specified limit.

**imagesdesired** A flag indicating whether or not the world should generate graphical depictions of each state encountered by the mind.

After this, the remaining `<asynctrun>` tags describe the events seen during the run, with each entry containing the timestep number (beginning with 1), and a world-generated textual description of both the state seen by the mind and the score achieved by the mind at that timestep, as well as the action taken by the mind in response to that state, which will frequently be an integer value.

```
<asynctrun>
  <action value="0"/>
  <score value="0,0"/>
  <state value="7,6"/>
  <timestep value="1"/>
</asynctrun>
```

In this example from a world where a cop must catch a robber by moving left or right on a one-dimensional looping path, the action “0” is taken which signifies “move one space left” (conversely, an action of “1” represents moving one space to the right). The robber’s movements are randomly generated, with the cop’s movements controlled by the mind.

The state value (7,6) describes the positions of the cop and robber respectively on the track as an index, where position 0 is the leftmost space and position 7 is the rightmost. Moves beyond the leftmost or rightmost space causes the character to wrap around to the opposite end.

The score value (0,0) indicates the number of times that the robber was caught, and the number of times it was due to the cop’s action rather than by the robber

entering our space, respectively -the robber's move is applied first. Thus, a score of "10,8" would show that the robber had been caught 10 times, but only 8 times due to the cop's action, having wandered into the cop's space twice.

### 5.4.2 The user interface

The local client program described in section §4.6 was designed to communicate with remote mind and world services by means of HTTP web requests, which was identified as a significant performance limitation in section §4.9. A new client was created using the Java Swing graphical user interface toolkit, capable of communicating over a much faster communications protocol. This protocol will be described in section 5.6.2 and section 5.6.4.

In section 5.6.1 some benefits of moving to a more centralised architecture will be explained, perhaps most importantly the ability to automate processes such as uploading worlds and minds to the server and make them available as world or mind services. To take advantage of these possibilities, this local client was again replaced, this time by a web interface on the World-Wide Mind server machine. This front-end allows users to initiate and manage runs, and provides a listable directory of worlds and minds, as well as an automatically-generated scoreboard for each world.

The interface allows runs to be carried out and stepped through, displaying the state and action taken at each timestep in the world, as well as a graphical view of the current state, if the world provides one.

When a run completes without an error, an entry is added to the world's scoreboard which is sorted according to the score definition provided by the world author.

The score attributes are treated as a sequential list in descending order of importance, for the purposes of sorting and tie-breaking partially equivalent scores.

Worlds and minds can also be submitted by logged-in users through an upload form, and deleted by the users who uploaded them. This user interface is presented in greater depth in appendix D.

### 5.4.3 The W2MServer daemon

The server component is *W2MServer*, a backend program which hosts the mind and world services. This daemon waits for incoming TCP connections (from an instance

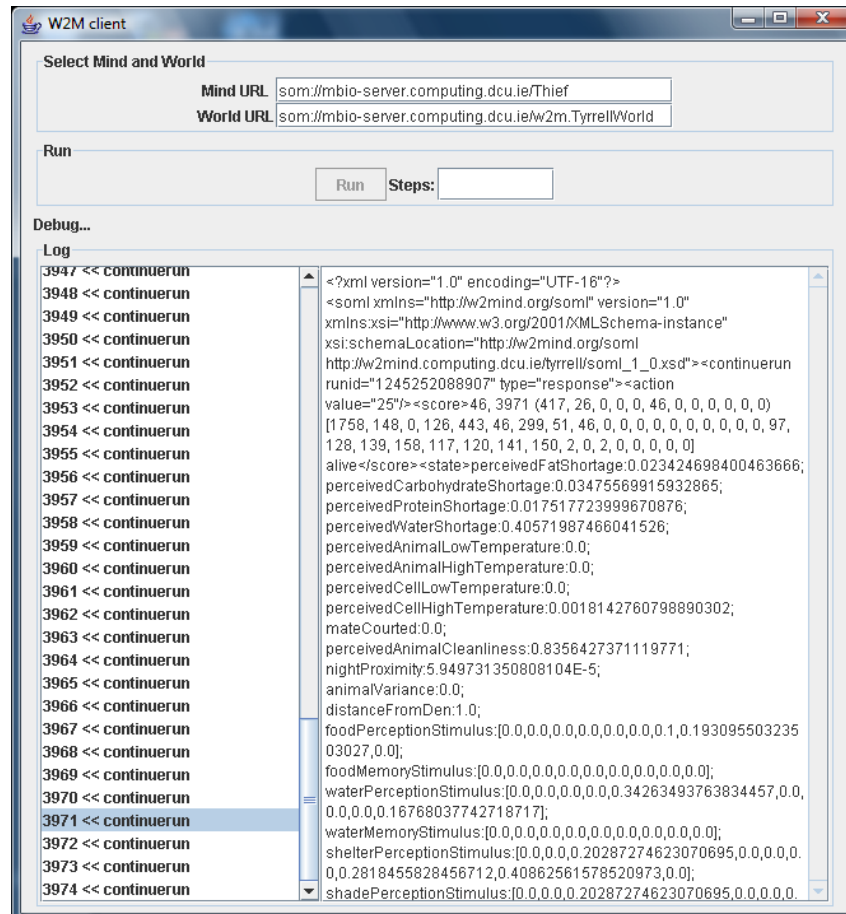


Figure 5.1: The “2.0” local client program for initiating runs of mind services in a world. Both mind and world services may be on remote machines, so a URL is used to locate each one and make a network connection. The protocol string “som:” is used to denote World-Wide Mind service addresses, as the communication method uses a custom network protocol (see section 5.6.4). This local client program was later replaced with a centralised, web-based interface which facilitates a wider range of interaction and collaboration. The server-side web interface is described in greater detail in appendix D.



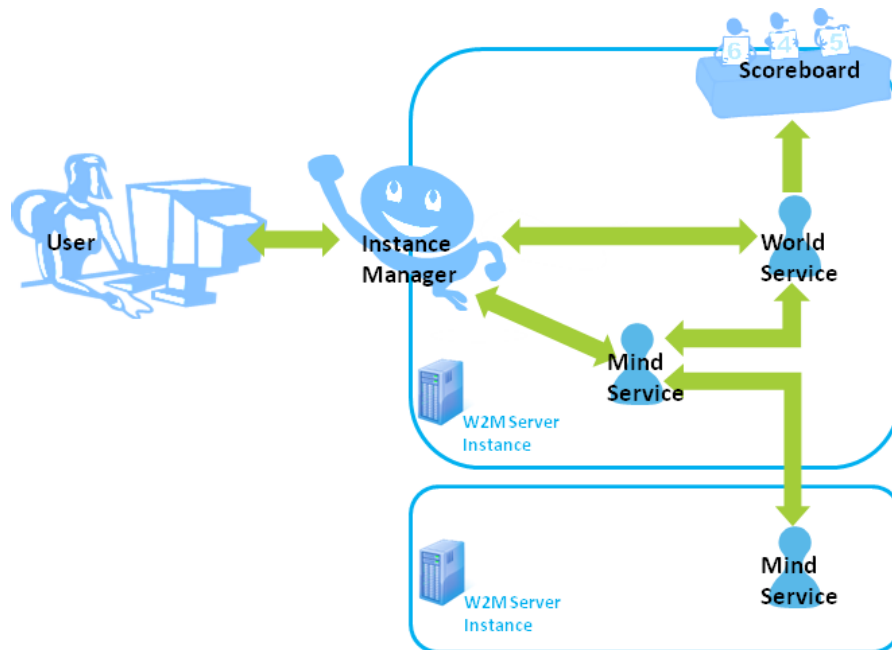


Figure 5.2: Architecture diagram showing the main interactions between components. The instance manager represents the web interface which allows the user to interact with and upload new mind and world services. In this example, the user starts a run with a hybrid mind which communicates with a submind located on a separate server.

of Runlogger or from a remote machine) and passes them to the appropriate mind or world service. Section 5.6.3 explains how mind and world service classes are found and loaded by the service classloader.

## 5.5 Uploading and testing worlds and minds

The W2M 2.0 server and the provided API for writing worlds and minds are implemented in Java for now. However, the communication protocol is designed so that other languages could be used in the future to create and host worlds and minds. Skeleton world and mind classes are provided which make the process of writing a new mind or world easier.

### 5.5.1 Uploading worlds

To create a world and make it available on a W2M server, the user follows these steps:

- Download the framework library “w2m.jar”, which contains classes that encapsulate network operations, as well as the `Mind` and `World` base classes.

- Create a Java class which extends the `World` base class, whose classname will be the world name as seen on the server.
- Implement the required methods: `newrun`, `endrun`, `getstate`, `takeaction`, corresponding to the messages sent and received as described in section 4.4, as well as the `getscore` method which will provide feedback on the performance of the mind in this world instance at each timestep and at the end of the run.
- Assemble the compiled world class, and any external classes it may need, into a JAR archive with a filename of the format `<worldname>.jar`.
- Upload the world's JAR file to the server via the web interface's world upload form.

The world will appear in the list of available worlds, and a scoreboard will be generated after a mind completes a run in the new world.

### 5.5.2 Uploading minds

To create a mind and make it available online, the process followed by a user is to:

- Download the framework library archive (`w2m.jar`), which contains classes that implement the required network communication functions, as well as the `Mind` and `World` base classes.
- Download the library archive containing the world-specific classes needed to describe the state and actions, if necessary.
- Create a Java class which implements the `Mind` interface, whose fully-specified class name will be used as the mind name when the server produces a scoreboard or list of minds available for the target world.
- Implement the required methods: `newrun`, `endrun` and `getaction`, corresponding to the messages sent and received as described in section 4.4 (see also 5.1 for an example of writing a hybrid mind which queries other minds for actions). The mind can be tested offline before being uploaded to the server.
- Assemble the compiled mind class, and any external classes it may need, into a JAR archive with a filename of the format `<mindname>.jar`.

- Upload the mind’s JAR file to the server via the problem world’s upload form for new minds. The mind will appear in the list of minds for that particular world and can be run immediately, generating a new entry on the world’s scoreboard.

### 5.5.3 Running and testing minds

For the purposes of running and testing minds, the web interface allows users to step forwards and backwards through the run, examining the states seen and actions taken at each timestep, as well as viewing the raw XML message content for debugging purposes.

The interface will display a graphical rendering of the current state of the world if the author has implemented the requisite `getimage()` method. These and other features of the web front-end interface are described in detail in appendix D and appendix E.

## 5.6 Improvements

A number of limitations of the previous work were identified in section §4.9, and these are addressed here.

### 5.6.1 Moving from a peer-to-peer to a more centralised “islands” architecture

The early Web was a model where data and services were largely hosted on authors’ individual sites. What became called “Web 2.0” was distinguished by, among other things, centralised sites to host people’s data. These sites also provided a range of services for interacting with that data, either through web interfaces or programmatically through well-defined public application programming interfaces (APIs).

For example, before the arrival of YouTube and other such portals for user-submitted video, the standard model for producers was to host their video files on an HTTP or FTP server to which they had access and sufficient storage space. Unless the producer had set up an embedded video player (which happened to work in that user’s browser), viewers downloaded videos as large files, and hoped that they had

the correct codecs installed to render the video and audio content.

When YouTube was established in early 2005, a new model of video production, hosting and consumption became available and was successful, with the number of videos viewed globally per day growing from 700 million in 2007 to 4 billion in 2012 [Hofsfeld et al., 2013]. Moving from the older ad hoc distributed model to a more centralised<sup>1</sup> model offered a host of benefits – among them:

- Viewers only needed a web browser and one browser plugin (Adobe Flash Player) to play any video uploaded to the site. No more downloading and installing of proprietary codecs.
- Perhaps most importantly for the viewer, videos were streamed rather than downloaded as single files. This means that one can start watching a video almost instantly after clicking a link, rather than waiting for a large file to download and opening it in a standalone player.
- Videos were automatically indexed and could be searched for by keywords and category with date ranges and other constraints.
- A recommender system automatically provided links (and thumbnails) to videos which may be of interest to the viewer.

In an attempt to gain similar benefits with the World-Wide Mind platform, and recognising that network latency impacts heavily on the speed of runs, it seemed appropriate to optimise the case where minds or worlds reside on the same machine such that no network access or construction and parsing of XML need occur.

This encoding, sending and parsing overhead is significant when a mind queries a set of other minds to assist in selecting an action which is to be taken in a world. Because the queries are executed in series, encoding messages to XML and back and passing them through the network stack would add a noticeable latency to the process and effectively penalise the use of many subminds, which is contrary to the research objective of creating large scale hybrid minds.

---

<sup>1</sup>Centralised in the sense that YouTube serves as a global repository of sorts, with a uniform interface where everyone can search for, watch and upload videos. To cope with the enormous demands on bandwidth and to provide a reasonable quality of service (QoS), YouTube's internal architecture consists of a network of caching servers and content delivery networks (CDNs) located across the world [Gill et al., 2007; Hofsfeld et al., 2013].

### 5.6.2 Replacing the web application server

To remedy these performance issues, the web application server was replaced with a daemon program, *W2MServer*. This daemon listens on a TCP port for incoming network connections. When a connection is made, the daemon spawns a thread to handle the incoming connection to a world or mind service hosted by *W2MServer*.

Connections can be made locally from an instance of the Runlogger program (part of the client web interface which will be explained later), or from a remote machine, which could represent a hybrid mind seeking to connect to a submind hosted on this server.

### 5.6.3 Service classloader

To make a mind or world available as a service, two issues must be addressed:

1. Messages destined for the service must be received and processed accordingly.
2. There must be some method of packaging the mind or world's Java class along with any other classes or file resources it depends on.
3. There must be a way of loading the class(es) from the resulting package.

The first issue has already been addressed in section 5.4.3. The standard solution to the second problem in Java is with Java archives (JARs), which are a compressed archive file format designed for storing Java classes and other required resources. To solve the last problem, a custom classloader [Gong, 1998] is implemented which attempts to load the class(es) pertaining to a world or mind in the appropriate JAR file, by looking for the file `<mindname>.jar` or `<worldname>.jar` in the mind and world service directories.

The classloader also attempts to make available the world API classes for mind programs, to simplify the uploading and sharing of code. When a user starts a run of a mind in a world, the world name is passed as a parameter in the *newrun* message sent to the mind instance's *ServiceProxy* message-handling object, which attempts to add the world's JAR file to the mind's classpath, and to thus make accessible to the mind all of the classes and interface definitions provided by the world author. This is necessary when the *State* and *Action* classes are subclassed by the world

author to provide an interface encapsulating the perceptions available and the types of actions which can be performed by the mind.

Taken together with the communication daemon introduced in section 5.4.3, this fulfills the first three requirements described in section §5.3.

#### 5.6.4 Custom network protocol

Once a thread has been created to handle an incoming connection, W2MServer must use some scheme for marking the end of each message. Otherwise, the receiving party will not know when a message has been completely received, and will block indefinitely when no more data needs to be read.

There are three ways of specifying the end of a message:

- Fixed size messages. By enforcing an exact size on all messages, and truncating or padding any messages which do not fit, the recipient knows exactly how many bytes to read from the network socket. This method is often used when it is known in advance that messages will be small and/or all follow a structure with static fields and fixed field lengths. This is not suitable for our purposes, since the contents and layout of data in each message depends for the most part on the world author's definition of possible states and actions.
- Terminating messages by appending a *delimiter symbol* – that is, a sequence of one or more characters which represents the end of a message. For example, in textual configuration files, a newline symbol might represent the end of a specified rule or setting.
- Prefixing each message with a fixed header section which includes a number that specifies the length of the ensuing message.

An advantage of delimited strings is that they are easy for humans to read, especially in complicated message structures with multiple delimiters. For example, program configuration files do not need to be parsed quickly, but would be inconvenient for humans to read and write if it required a complex binary format, rather than a syntax which relies on newlines and quote characters.

Some of the disadvantages of using symbols to delimit messages are:

- The recipient of the message must search the message contents for the delimiter symbol, one byte at a time, to know when enough data has been read from the network socket or file, and
- If the delimiter appears inside the message, then it must be *escaped* to prevent the message from being incorrectly truncated at that point by the recipient. Escaping the delimiter requires that the message be searched before sending, and any occurrences of the delimiter replaced, usually by doubling the delimiter. For example, the backslash symbol is used as an escape character inside strings in the C language. If the string should actually contain a backslash symbol, then specifying it in the form "\" would cause it to be parsed as an unterminated string – instead, the symbol must be written twice: "\\". This means that the recipient of the message must also search for these sequences and de-escape the received string by replacing doubled delimiters with their single counterparts.

For these reasons, it can be significantly faster to use a length prefix rather than delimiter symbols when sending strings which do not need to be read directly by humans.

W2MServer therefore uses a simple protocol for sending and receiving XML messages over the network as length-prefixed strings. When a message is ready for transmission, its length in bytes plus four is calculated and written as a four-byte integer header.

When reading a message from the network socket, the first four bytes are used to determine the number of bytes to be read in total (for more technical detail, see section §5.8).

00	00	00	4B	<soml><getaction_type="request"><state>x:147</state></getaction ↔ ></soml>
----	----	----	----	---

Figure 5.3: A sample message (omitting some boilerplate headers and attributes in the XML document) encoded for transmission over a network TCP stream. The first four bytes signify the length of the message, giving a maximum message size of  $2^{32}$  bytes, or 4 gigabytes. In this example, the message itself is 71 bytes in length, and the total transmission length of 75 (including the header) is encoded as a four-byte value, expressed in hexadecimal as 0000004B.

By using stateful TCP connections, it is no longer necessary to serialise world

and mind instances to files on disk, and retrieve them when messages are received, as explained in 4.9.1. Instead, when a message is sent to an instance of a world or mind, a *ServiceProxy* object is created which lasts for the duration of the run, and which passes messages to the appropriate world or mind instance, creating the instance first if need be.

An alternative possibility would be to use one of a number of existing message queue libraries, such as ZeroMQ or RabbitMQ [Hintjens, 2014; SpringSource, 2012], which can provide a facility to send and receive messages between processes, via shared memory systems or on top of TCP or UDP. Although these libraries are well-designed and perform well in the general case, writing a simple, custom protocol for our purposes afforded greater control over the low level transmission mechanisms.

### 5.6.5 Asynchronous runs

The initial design of the World-Wide Mind envisioned all minds and worlds as web services distributed on the Internet and communicating via XML requests and responses. However, there is an unavoidable latency involved in network communication, especially across the Internet.

To mitigate this, and because the use of a dedicated communication protocol allows it, messages between services now avoid the network stack when the two services are hosted on the same machine by the same server process [Mac Fhearai et al., 2011].

An *asyncrun* message<sup>2</sup> was devised which, when received by a mind instance, causes it to carry out a run directly with a world instance, sending asynchronous updates of which states were seen and which actions were taken. This two-way conversation between mind and world speeds up the communication loop significantly, since the user now acts as an observer, receiving a stream of updates without blocking the progress of the run, rather than as a middleman between the mind and world instances. Updates to be returned to the user are added to a queue and sent asynchronously, and should not impact significantly the progress of a run.

The basic series of interactions for an asynchronous run is as follows, with the

---

<sup>2</sup>The *asyncrun* message was originally introduced as *continuerun*, and was renamed *asyncrun* by Brian Monks, a collaborator on the World-Wide Mind project. This term may express more precisely the intent of the message.



main loop (steps 4-6) visible in figure 5.4:

1. The client sends an **asynrun** message to the mind, providing the name of the intended world service and the hostname of the world server.
2. The mind initialises itself, and sends a **newrun** message to the world, which creates a world instance unique to this run. Note that this scheme was optimised and simplified somewhat in Brian Monks' research described in [Monks, 2010]. Before his design contributions, the client sent a **newrun** message to both the mind and world before sending an **asynrun** (or **continuerun**) message to the mind.
3. The mind sends a **getstate** message to the world instance. The world instance will respond to any **getstate** message by returning the current observed state **x** of the world.
4. This observed state is passed to the mind in a **getaction** message, which returns a suggested action **a**.
5. Action **a** is passed to the world in a **takeaction** message, which returns the new state generated by applying the given action to the current state.
6. The mind appends the pair **(x, a)** representing the state seen and the action taken to a queue where it will be sent asynchronously to the client, without blocking the progress of the run.
7. If the run has completed, then the world returns an **endrun** response to the mind instead, including the score information which summarises the mind's performance on this world instance.
8. If the mind received an **endrun** response from the world, then it terminates the loop. Otherwise, it received the new state generated by taking action **a**, and continues the loop from step 4.

It should be noted that because of this asynchronous design, it is possible for a very long and rapid run to exhaust the available space in Java's heap memory area by adding large messages to the sending queue more quickly than they can be sent over the network and removed from the queue. To mitigate this possibility, it may

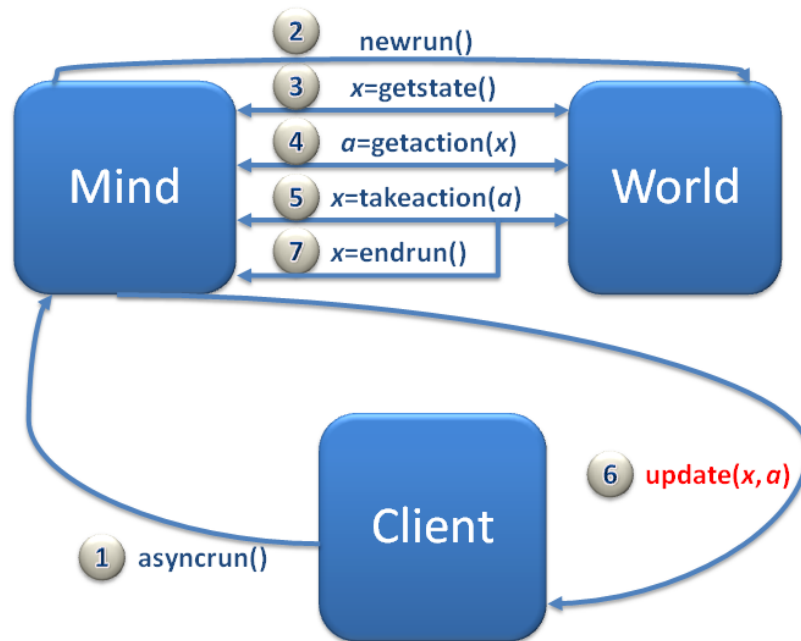


Figure 5.4: The interactions between client, mind and world in an *asyncrun* loop. The mind and world communicate directly and the client receives an asynchronous stream of updates containing state/action pairs, instead of serving as a middleman between mind and world.

be wise to enforce a preset limit on the number of steps that can be executed in a run. This can be done by making the runlogger count the number of timesteps that have been processed and end the run if the count is greater than the defined limit.

Allowing a mind to communicate directly with the world suits the more centralised “distributed islands” approach taken in constructing a web interface to the minds and worlds hosted by a server, and makes it possible to provide a browsable directory of all submitted minds and worlds hosted on the server, with a scoreboard for each world generated by user-defined score attributes.

These improvements greatly reduce the time taken required to perform a run of a mind in a world, often by several orders of magnitude, especially when a mind and world are hosted on the same server.

Taken together, the performance improvements described in this section and section 5.6.4 satisfy the fourth requirement stated in section §5.3 (p. 66).

### 5.6.6 Scalability of large hybrid minds

While these measures are intended to greatly optimise speed in many cases, I expect that it will sometimes be necessary to rely on mind services which do not exist on

the same server.

Indeed this may be inevitable as hybrid minds are scaled up, since if we consider a hybrid mind as a hierarchy where the mind calls  $n$  subminds, which themselves call  $n$  other subminds, and so on up to some fixed depth  $d$ , then the total number  $M$  of minds in the tree structure (a perfect  $n$ -ary tree) will be  $\frac{n^{d+1} - 1}{n - 1}$  [Sawada, 2008], and therefore exponential in the depth of the tree. For example, if a hybrid mind  $M$  is composed of three subminds, each of which can itself call three subminds, then we have  $M = \frac{3^{2+1} - 1}{3 - 1} = \frac{27 - 1}{2} = 13$  total minds. With  $n = 5$  and  $d = 5$  we have 3,906 minds, and for  $n = 6$  and  $d = 6$  we have a hierarchy of 55,987 total minds, even if this situation seems rather unlikely. The situation is more complicated if graph-like structures exist and several minds refer to the same submind, but nonetheless it is clear that hybrid minds can quickly grow too large for one server alone to cope with.

The exponential problem of hierarchies is not new. The human brain has enormous low-level parallel connectivity, with the fan-in (connections to) and fan-out (connections from) of neurons in the cortex reaching 10,000 [Furber et al., 2006]. While neurons operate in the scale of milliseconds – a factor of  $10^6$  slower than the nanosecond scale of current day microprocessors [Thagard, 1998, p. 209] – this sheer connectivity more than makes up for the difference on many tasks. However, in terms of hierarchical structure, the cortex yields only 6 layers [Furber et al., 2006], which tells us that an extended hierarchy takes us only so far. Although we do not expect to see hybrid minds anywhere near as interconnected as any biological brain, the possibility of having many (perhaps hundreds of) diverse specialties is exciting and could certainly be of use in building large-scale complex minds.

The W2M 2.0 design operates efficiently across the network, but transmission latencies and encoding/decoding of XML messages will add an unavoidable time penalty for every exchange of messages. This is cumulative and noticeable on long runs.

In future, this problem may be partially addressed by using a more compact message encoding. For example, the MessagePack binary serialisation format produces significantly less space overhead than XML, and can be encoded and decoded very quickly [Furuhashi, 2012].

Some methods for synchronising the state of a distributed simulation over unre-

liable communication channels are described in section §3.8, but they do not provide a general method that would automatically work for all worlds. Another technique which may help to minimise the effects of network latency is delta encoding [Mogul et al., 1997], where rather than complete snapshots of the world state, only the *differences* in state from one timestep to the next are transmitted.

### 5.6.7 Server-side world and mind database

As the W2M 1.0, was not designed with an intent for centralisation, there was no structured storage system for metadata about the hosted worlds and minds. A significant improvement in the W2M 2.0 was simply to automate the processes of uploading worlds and minds to a server and hosting them as network services; this afforded an opportunity to capture and maintain important metadata relating to the mind and world services.

To make this possible, a server-side database was added which stores information about all of the installed minds and worlds, including the following metadata:

1. Mind or world name (this also serves as the *fully-specified class name* used to create instances of the mind or world service).
2. The author of the mind or world.
3. The total number of runs performed by that world or mind.
4. For minds:
  - (a) The mind type – `Mind` for self-contained minds, `Mind_M` for hybrid minds which may request actions from other minds. This is purely for descriptive purposes, however, and does not influence the behaviour of the server.
  - (b) The fully qualified class name [Lindholm et al., 2013] of the *world* in which the mind is designed to solve problems. This is required so that the `w2mServer` daemon can load and make available the classes provided in the appropriate world JAR file. Otherwise, useful classes pertaining to the world (for example, those which extend the `State` class with world-specific accessors such as `getPerceivedThirst()` or `isKingInCheck()` which facilitate minds to programmatically examine the current state) would be

either have to be duplicated in the mind's JAR file by its author, or the mind would have to parse and query a generic string-based data structure such as XML or JSON.

This database is operated on by both the front-end web interface and the back-end server programs, and another database table records the scores achieved by each mind during world runs, when they improve upon the best score so far for that mind. This fulfills the sixth requirement listed in section §5.3 (p. 66).

### **5.6.8 Robustness and security issues**

The mind and world programs executed during each run consist entirely of code submitted by users on the Internet. The source code may not be provided, since the services are uploaded to the server – without administrative oversight – in the form of Java archives that need only contain compiled bytecode classes, which are directly executed by the W2MServer daemon.

The quality and intention of this third-party code cannot therefore be guaranteed or audited, which leads to problems if programs are submitted which either fail to terminate or attempt to cause damage to the server.

#### **5.6.8.1 Non-terminating programs**

Detection of infinite loops, deadlocks and good or bad intentions of these user-submitted programs is difficult and cannot be guaranteed in all instances, since it would require that the halting problem be decidable [Turing, 1936; Sipser, 2012].

Instead, the server daemon monitors and kills any service instance which takes more than some fixed amount of time to respond to a message.

This heuristic will occasionally terminate runs engaged in lengthy but legitimate computation, and must therefore be set at an appropriate threshold. Details of this problem, and the heuristic solution implemented, will be discussed in section 5.8.4.

#### **5.6.8.2 Security policy**

It is impossible to know in advance or guarantee what sort of actions will be attempted by the mind and world programs, which can be submitted to the server

by anybody on the Internet. Like the problem of non-terminating programs, this problem is not new, and affects any server which hosts third-party programs.

One solution to the problem is to simply disallow all calls to functions or methods which might perform dangerous operations – for example, deleting or renaming files in the filesystem. However, this would be a considerable limitation if a program needed to perform those operations for genuine reasons.

Another less extreme solution is to define a robust environment, called a *sandbox*, in which the program's actions can be safely performed such that any accidental or purposeful damage is limited to the contents of the sandbox.

In the W2M 2.0, a combination of sandboxing and restriction of API calls is used. To help guard against the actions of malicious (or dangerously careless) programs, Java's security manager is invoked with a policy that prevents minds and worlds from carrying out certain activities, such as:

- reading and writing files outside of certain temporary directories,
- making network connections directly, rather than through the provided interfaces and classes for accessing remote services, or
- accessing environment variables or Java's system properties.

The non-terminating program monitor, and the implementation of a robust security policy together satisfy the fifth requirement specified in section §5.3 (p. 66).

## 5.7 Writing and running hybrid minds

A fundamental idea motivating the World-Wide Mind is the possibility of creating hybrid minds which consult other subminds at each timestep when deciding which action to take, given the currently observable state.

### 5.7.1 Proxy interfaces for accessing remote mind and world services

In the W2M 1.0, if an author wanted their mind to call another mind hosted on the same server (or a remote server), they were forced to write code to create and decode XML messages in the SOML format (see section §4.5) and communicate them over the Internet using HTTP. As a result of this complexity, many authors

were discouraged and very few attempted to create hybrid minds. Those that were created were extremely slow due to the design and implementation issues identified in section §4.9.

To facilitate the construction of hybrid minds in W2M 2.0, a simple API was developed with the *RemoteMind* and *RemoteWorld* classes serving as proxy interfaces to remote minds and worlds. These wrapper classes implement the same methods specified in the Mind and World interfaces, so using them is almost identical to using a locally accessible mind or world, except that the user specifies the desired service name and the address of its hosting server.

The design and provision of these proxy interfaces are intended to make it easy to re-use minds and worlds written by many authors, and in particular to make the process of creating hybrid minds straightforward, rather than requiring users to write complicated networking and parsing code as was necessary under the W2M 1.0. This is intended to fulfill the seventh requirement listed in section §5.3.

The important methods provided by the *RemoteMind* class are:

**newrun** instantiates a connection to a remote mind and calls its `newrun` method, triggering any required initialisation (for example, a hybrid mind may construct *RemoteMind* objects to access its own set of remote minds) therein.

**endrun** sends an end run request to the remote mind and terminates the connection.

**getaction(s)** passes the state `s` to the remote mind instance and receives its suggested action.

**asynrun** populates an asynchronous run request with the world service details and passes it to the remote mind, which creates its own connection to the world and carries out a run, passing back updates via an asynchronous queue.

And the important methods provided by *RemoteWorld* are:

**newrun** instantiates a connection to a remote world and calls its `newrun` method, triggering any required initialisation therein.

**endrun** sends an end run request to the remote mind and terminates the connection.

**getstate** retrieves the current state of the remote world.

---

**Algorithm 5.1** A mind calling another mind. MindM delegates to w2m.LookupMind for selection of each action.

---

```
import org.w2mind.net.*;
import org.w2mind.tyrrell.*;

public class MindM implements Mind {
    String OTHER_MIND_NAME = "w2m.LookupMind";
    String WORLD_NAME = "w2m.TyrrellWorld";

    // specify the host server for the remote mind and world services
    String REMOTE_SERVER = "som://w2mind.computing.dcu.ie";
    // the remote mind proxy object
    RemoteMind rm;

    // when called at the beginning of a run, initialise mind proxy
    public void newrun() {
        rm = new RemoteMind(REMOTE_SERVER, OTHER_MIND_NAME, WORLD_NAME);
        rm.newrun();
    }

    // perform any needed clean up at the end of a run
    public void endrun() {
        rm.endrun();
    }

    // called once per timestep. In this case, delegate to the submind.
    public Action getaction(State s) {
        return rm.getaction(s);
    }
}
```

---

**takeaction(a)** causes the remote world to apply the chosen action **a** to its current state and move to the next state. To cut down on unnecessary network transmissions, the remote world also returns the new state.

These methods are equivalent to the messages described in section 4.4. A remote mind instance can be created and accessed with code such as that shown in algorithm 5.1.

If the world author has implemented the *getimage* method which renders a graphical display of the world, then this will also be displayed at each timestep, at the user's request. The user may not wish to do so, since rendering an image at every timestep will inevitably slow down the execution of a run, often significantly. For example, running a simple mind "mater" in a modified implementation of Tyrrell's animal simulation world (introduced in 5.9.2 and described more completely in ap-



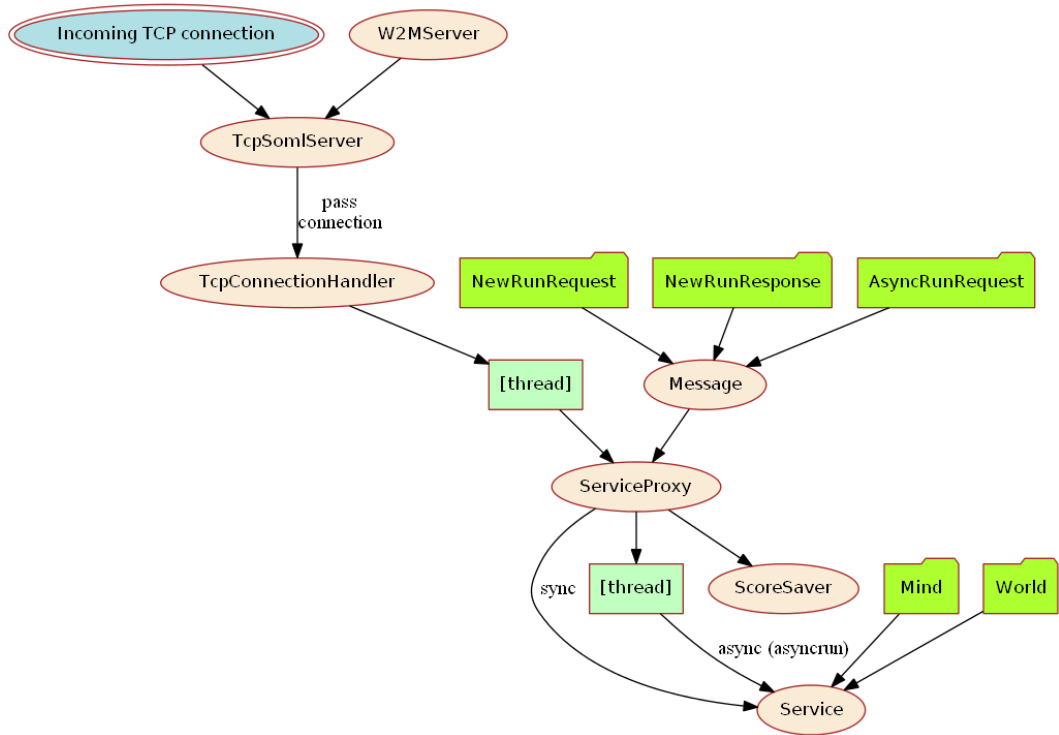


Figure 5.5: A diagram showing the interactions between some of the important classes used by the backend *w2mServer* program.

pendix B) without generating graphics took 0.77 seconds to complete 178 timesteps in the world. Running the same mind while generating graphics at each timestep required 14.6 seconds to complete 179 timesteps – a rate almost 20 times slower than the no-graphics condition.

## 5.8 Implementation details

This section provides a brief explanation of the system design from a software engineering perspective.

The diagram in figure 5.5 shows some of the primary interactions between the important classes used by the *w2mServer* daemon program.

When the *w2mServer* program is started, it invokes an instance of the *TcpSomlServer* class, which listens on a TCP port for incoming network connections. When an incoming connection is made, the connection object is passed to a new thread of execution, managed by the *TcpConnectionHandler* class.

### 5.8.1 `TcpConnectionHandler`

Each instance of *TcpConnectionHandler* pertains to a separate conversation with a mind or world, ultimately represented by an implementation of either the *Mind* or *World* interfaces, which each extend the base *Service* interface.

*TcpConnectionHandler* will listen on its connection's TCP socket for any data. When any bytes are available, it reads as much as possible into a buffer, and then extracts the first four bytes which are used to specify the size *N* of an incoming message. The socket is then read from until *N-4* more bytes have been received. This may require waiting for several network packets to arrive and appending bytes from the buffer to the message string.

The received message string is expected to be a document in SOML format, representing a message sent to or between mind and world services (for more information on SOML, see section §4.5). When the message string has been successfully obtained from the socket, *TcpConnectionHandler* passes it to an instance of the *ForwardingListener* class via its `getResponseForMessage(s)` method.

This method converts the received message string – a SOML document – to a *Message* object, which encapsulates all the data relevant to a message being sent to or from a mind or world service. The *Message* class is a map-type data structure, with named keys (or fields) and values.

For the most part, subclasses are used which extend the *Message* class with convenience setter methods which populate fields specific to the type of message – for example, a *NewRunRequest* instance provides a `setImagesDesired()` method which sets the `imagesdesired` field to contain a boolean value, but a *NewRunRequest* will not, and a *GetActionResponse* will contain an “action” field and value.

Before a message reaches the mind or world instance, however, it is passed from *TcpConnectionHandler* to the `handleMessage(m)` method of an instance of the *ServiceProxy* class.

### 5.8.2 `ServiceProxy`

The *ServiceProxy* class processes incoming *Message* objects, initialising and calling the appropriate methods on an instance of a world or mind service. When a message arrives, its `type` property is queried and its value used to select the appropriate

handler method by use of reflection. For example, if the message type is “`newrun`”, then the `newrun()` method is called.

The handler methods extract further parameters from the incoming message as required – for example, the contents of the `action` field in the `takeaction()` method – and call the actual methods implemented by the target service, which is created and initialised first if necessary. This is where the mind or world service actually performs its task – which is usually either suggesting an action or updating the world model with a selected action.

When the call to the world or mind service’s method returns, any return values (for example, the suggested action, in response to a `getaction` request) are added to a new response *Message* and returned to the *ForwardingListener* instance, which transforms the message object back to a flattened XML representation, ready for sending back by *TcpConnectionHandler*.

### 5.8.3 Synchronous and asynchronous communication

At the bottom of figure 5.5, there are two arcs leading from the *ServiceProxy* class to an abstract *Service* interface (which will be a concrete instance of either a world or mind). One arc is labelled “sync”, and refers to normal, synchronous invocation of the service’s methods. The other arc, labelled “async (asynrun)”, refers to the asynchronous run process depicted in figure 5.4 and described in detail in section 5.6.5.

When an `asynrun` request message is received and passed to a mind service’s *ServiceProxy* instance, it creates a new thread which initialises both the mind service and the intended world service. The world name and the Internet address of its hosting server are defined by the `world` and `worldHost` fields, respectively, of the *AsyncRunRequest* message class.

The new thread spawned by *ServiceProxy* then carries out a complete run of the mind in that world, adding at each timestep  $t$  the world state, score and the action taken at timestep  $t-1$  to an *AsyncRunResponse* message object.

These messages are added to a concurrent blocking queue structure, waited on by the *TcpConnectionHandler* in its own execution thread. When it successfully removes a message from the queue, it immediately converts it back to a string of XML before sending it back over its TCP socket.

After the final action has been taken, the response message from the world will be an *endrun* response, rather than a *takeaction* response. When this happens, the last state and score seen and the action taken by the mind will be included in the *AsyncRunResponse* message, and a flag will be set in the *ServiceProxy* indicating that there will be no more messages to read.

When the *TcpConnectionHandler* tests this flag and finds that there are no more messages to send, it sends a null message (a 4-byte header representing the integer value 4), which is interpreted by the other side as a signal that the asynchronous run has completed.

#### 5.8.4 The problem of non-terminating calls

It is possible to submit minds and worlds which enter an infinite loop when their *getaction()*, *takeaction()* or other methods are called by a *ServiceProxy* object. This will cause a significant portion of processor time to be wasted, which slows down other “genuine” computation.

Even if these methods call other methods which block, rather than “spinning” and wasting CPU time, the objects and data structures created to manage the run (including message transmission buffers as well as the entire world state and related objects) will occupy memory, a limited resource.

When a separate Java interpreter process is started to handle a run, a virtual machine is created and allocated a private heap space. If a run hangs – that is, never completes – then its operating system process and the virtual machine’s allocated heap area will remain active and occupy memory wastefully.

As discussed in section 5.6.8.1, there is no general way to detect whether a program will hang, or even whether a program is *currently* stuck in an infinite loop. This is not a new problem in computer science, and must be dealt with by web service hosting providers, for example, whose servers execute third-party code under less restrictive conditions than are imposed upon mind and world services hosted on a W2M server.

Some hardware solutions exist which attempt to detect process crashes through the use of instruction count heartbeats, and infinite loop hang detectors (several types are proposed and discussed in [Nakka et al., 2005]), but application of these

techniques is either difficult or impossible in a high-level software platform, and where a software implementation is possible the instrumentation would likely be costly in compute time and memory overhead.

Another software-based approach attempts to tackle the problem by attaching a monitor program named Jolt to a program which one suspects to be caught in an infinite loop [Carbin et al., 2011]. At the beginning of a loop iteration, Jolt records the program’s current state. On the next iteration, the new program state is compared with the saved snapshot. If they are the same, then the program is deemed to be stuck in an infinite loop, and two options are provided: terminate the program, or “escape” the loop by transferring control to the next instruction after the looping instruction. To do this, Jolt requires a static instrumentation of the target program’s source code (adding special runtime calls which indicate the entry and exit of every loop in the program’s control flow graph), before the monitor program can be dynamically attached to the running target [Carbin et al., 2011]. However, the system fails to detect infinite loops where the program state changes between iterations, and is intended only for single-threaded applications (a multi-threaded program waiting in a spinlock, for example, can be mischaracterised as in an infinite loop).

Another limitation common shared by most of these techniques is that it’s unclear what happens when a mind or world is blocked while waiting on a response from a remote submind, for example. We cannot instrument the remote service and detect whether it is itself stuck in an infinite loop.

The rise of virtualisation in server computing has provided one solution to the problem of runaway processes using excessive CPU time: each user’s virtual account is allocated a fixed computation allowance. Rather than running directly on the server’s operating system level, the users’ programs execute inside a virtual machine which is isolated to some degree from other virtual machines running on the same hardware [Oikawa et al., 2004]. This mitigates, but does not eliminate, the negative effect caused to other users’ programs. It’s unclear how the user account specific nature of the intervention might apply on the W2M; should limits be applied to individual world and mind services, or to all programs uploaded by a particular user, or to each instance of a world or mind service?

Another solution is to set a fixed deadline after which a process or method call is automatically terminated. This solves the problem of both CPU-hogging infinite loops and wasted memory used by non-terminating processes, but at the expense of possibly terminating a process which *would* complete given enough time. This solution is inappropriate for web service providers, where well-behaved daemon programs often need to run indefinitely. However, this approach is taken in the W2M 2.0, since it seems preferable to mistakenly terminate a slow run than to ignore a run stuck in an infinite loop and risk wasting a large amount of compute time and memory space – an amount that would increase as more runs are initiated, eventually leading to system failure.

Alternately, runs suspected of having entered a “hung” state might be added to a “freeable” list to be terminated if and when a new job is ready to start and a large number of runs are already in progress. This approach is akin to a garbage collection system triggered only by a memory allocation request in a low-memory state (rather than continuously or at periodic intervals), and may prevent the needless termination of some (but not all) slow runs. The disadvantage of this sort of solution is that each run may use a different (and unpredictable) amount of computing resources, and without artificially and carefully limiting each thread’s compute capability, it is possible for a single non-terminating run to significantly impact performance by hogging the CPU and available memory and by constantly accessing disk or network resources.

To prevent an accumulation of these “hung” threads and processes, wasting both CPU cycles and memory resources, the `handleMessage()` method of *ServiceProxy* creates a *Future* object to encapsulate the computation, before submitting it to an *executor service* which executes it in a separate thread. This is shown in algorithm 5.2.

This allows the computation to be interrupted and terminated with an error message if a fixed completion deadline expires. Currently, this timeout – defined in the `TIME_OUT` constant – is preset to 180 seconds, to minimise the risk of terminating method calls which take a long time but do complete.

---

**Algorithm 5.2** The *handleMessage()* method, which schedules the real processing of a message as a job to be executed with a timeout.

---

```
public Message handleMessage(final Message message) {
    Callable<Message> job = new Callable<Message>() {
        public Message call() throws Exception {
            return handleMessageAsync(message);
        }
    };
    Message response = null;
    try {
        Future<Message> future = executorService.submit(job);
        response = future.get(TIME_OUT, TimeUnit.SECONDS);
    } catch (Exception e) {
        System.out.println("Timed out waiting for response to message: ["
            + message.toSOML() + "]");
        e.printStackTrace();
        response = message;
        response.setAsResponse();
        response.setStatus("400");
        response.setStatusText(e.getStackTrace()[0].toString());
    }
    return response;
}
```

---

## 5.9 Evaluation of the W2M 2.0 platform

As far as the backend architecture and implementation of the W2M 2.0 platform are concerned, each of the requirements listed in section §5.3 has been met. However, it is important to evaluate the platform as a whole and ask whether it sufficiently addresses its intended goals. Accordingly, this evaluation process includes:

- A comparative performance benchmark of the W2M 1.0 versus the W2M 2.0 systems,
- An analysis of the use in A.I. teaching assignments, where the W2M 2.0 platform was used by a large number of authors to write minds (some of which are hybrids, calling subminds written by other authors), and
- Analysis of a usability survey completed by some of those authors.

### 5.9.1 Performance benchmarks

The W2M 1.0 server design executed online runs of a mind in a simple world at a speed of approximately 1 timestep per second, where the mind was hosted on a

separate machine to the world, connected by a LAN. Under the same conditions, the W2M 2.0 server implementation was able to complete runs at a speed of over 100 timesteps per second.

In the optimal performance scenario, running a mind in a simple world which was hosted on the same machine resulted in an execution speed of 21,008 timesteps per second on the same hardware, because the network stack and XML encoding/decoding steps could be avoided entirely for communication between mind and world.

### 5.9.2 A.I. teaching assignments

The W2M platform has been used to host several worlds and a large number of minds submitted by undergraduate computer science students taking artificial intelligence modules.

The problem environment was a modified implementation of *Tyrrell's simulated environment* [Tyrrell, 1993], a simulated animal behaviour problem where the mind controls a creature living in a two-dimensional grid world with a rich set of features. In the world, the simulated animal must satisfy a number of often conflicting goals, such as foraging for food, finding a mate, maintaining cleanliness and temperature. A description of both Tyrrell's original world and the modified version used in this research can be found in appendix B. An instance of the world during a run is depicted visually in figure 5.6 on the following page.

The sensory input for this world is a large series of real numbers representing internal perceptions such as perceived carbohydrate, fat and protein shortages, body temperature or thirst, as well as external stimuli such as outside temperature and perceived closeness of food sources or mates in each direction and in the current cell. There is a degree of noise in the senses and a limited distance to external perceptions, as well as a degree of randomness which makes the problem non-deterministic.

The world scoreboard provided a competitive measure of success by which students could improve their submissions and search for suitable candidate subminds for use in creating their own hybrid minds. For one assignment, an extra requirement was added that each student must submit at least one hybrid mind, even if it relied only upon a "dummy" submind which did nothing useful. Some submissions indeed consisted of a hybrid mind which carried its own action-selection logic, but



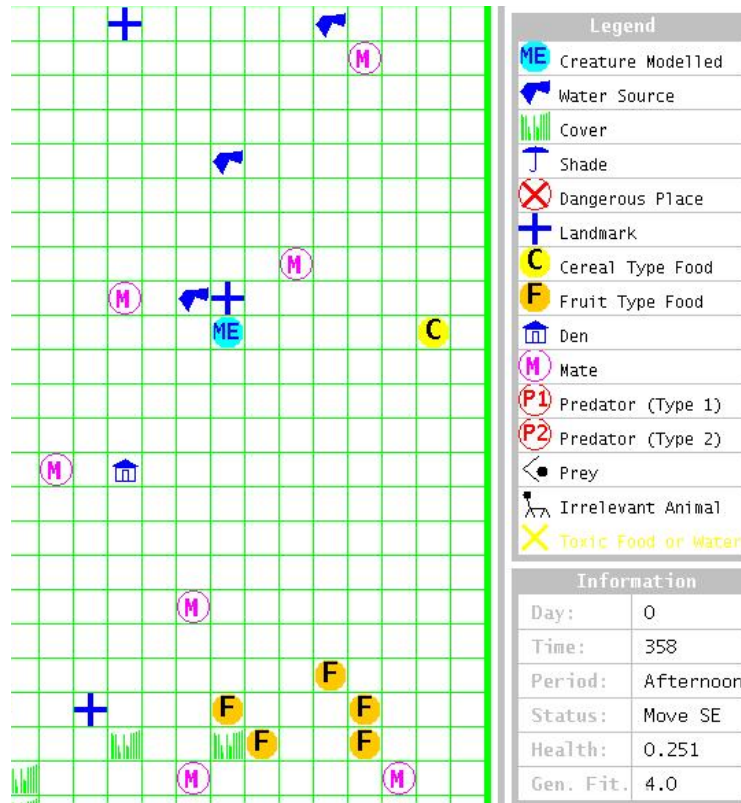


Figure 5.6: A section of the visual rendering of a single state in a modified implementation of Tyrrell’s simulated environment (see appendix B). In the vicinity of the simulated agent (“ME” in the image) controlled by the mind, several features (potential mates, sources of drinking water, cereal food, and the animal’s den) are easily visible. Additionally, a large cluster of features in the bottom section is visible here, but are not seen by the mind, whose sensory percepts only extend a fixed number of grid cells.

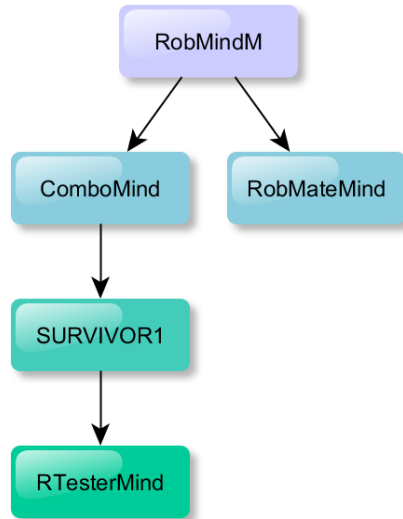


Figure 5.7: A hierarchy diagram of the top-scoring mind submitted by a user, *RobMindM*. This mind combines *ComboMind* (created by another author) with *RobMateMind* (created by *RobMindM*'s author).

delegated to a “sleep” mind when necessary – this submind would always return the “sleep” action and was therefore superfluous.

However, many other hybrids took advantage of the collective expertise of multiple subminds, and at the end of the assignment period, nine of the top ten best-scoring minds submitted during the assignment were hybrids. To help characterise the behaviour of minds, a feature was added which records the names of any subminds called by a hybrid mind. This allows us to examine the hierarchy tree for any hybrid mind, although it does not capture the context in which those subminds were called. The hierarchy tree for the top two minds on the scoreboard, *RobMindM* and *CowardlyMindFinal*, are shown in figure 5.7 and figure 5.8 respectively.

The deepest hierarchy tree of the hybrids is that of *CowardlyMindFinal*, at four layers deep. This suggests that the deeper hierarchies may have presented diminishing returns in performance, or were more difficult to create, or more error-prone for hybrid mind authors (recall the discussion of hierarchical scalability from both a computational and biological standpoint in section 5.6.6).

### 5.9.3 Usability

This research intends to address the possibility of constructing hybrid minds by using minds written by a potentially large number of people. If writing and testing minds

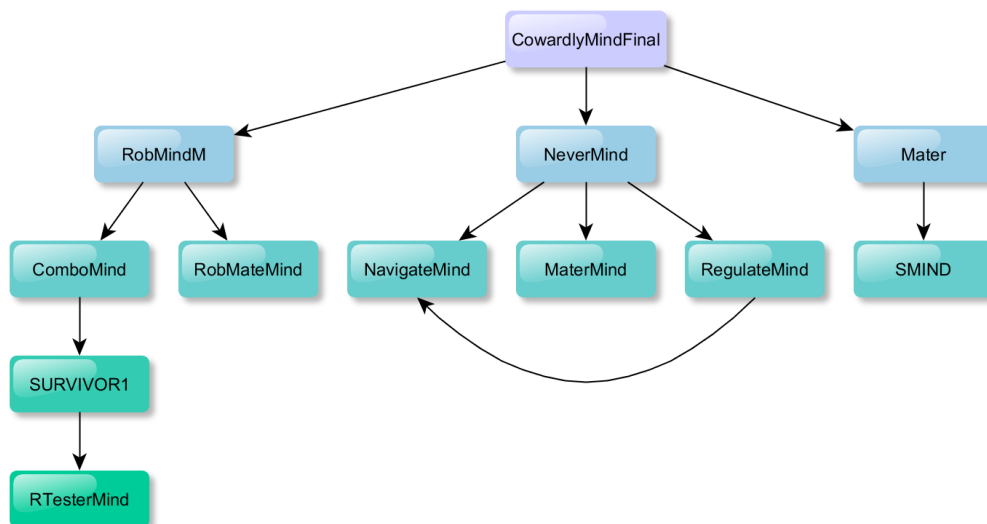


Figure 5.8: A hierarchy diagram of the second-highest scoring mind submitted by a user, *CowardlyMindFinal*. Note that this mind called the top-scoring mind, *RobMindM*, at least once, but did not outperform that mind.

is unnecessarily difficult, then users may be less inclined to participate by developing and contributing their minds and worlds and by building upon the minds submitted by others.

In an attempt to evaluate the overall usability of the W2M 2.0 platform, a survey was put to the students who recently submitted minds in a competitive assignment who wrote minds for a chess world (a description of this world can be found in appendix C on page 187). Participation in the survey was anonymous and not mandatory.

The questions were as follows:

1. How long did it take you from first reading the documentation to uploading your first Mind?
2. What could we have done to make that time shorter?
3. How easy did you find it to run a Mind?  
*(1 - Very easy, 2 - Easy, 3 - Medium, 4 - Hard, 5 - Very hard)*
4. How easy did you find it to upload a Mind?  
*(1 - Very easy, 2 - Easy, 3 - Medium, 4 - Hard, 5 - Very hard)*
5. (If applicable) How easy did you find it to call other people's Minds from your code?  
*(1 - Very easy, 2 - Easy, 3 - Medium, 4 - Hard, 5 - Very hard)*
6. (If applicable) How useful did you find it to call other people's Minds in order to actually succeed at the problem?  
*(1 - Very useful, 2 - Useful, 3 - Medium, 4 - Not so useful, 5 - Not at all useful)*
7. If you have ever participated in a collaborative project (e.g. at work, or an open source project) how would you compare the ease of calling other people's Minds with the ease of calling other people's methods/functions in those projects?
8. If you noticed any bugs or problems with the system, please describe them.
9. Please describe any important features you would like to see added, if any.
10. What did you think of the chess problem? Can you suggest other interesting problems and challenges that it would be good to have on the system?

Questions 3, 4, 5 and 6 use a five-point semantic differential scale – a rating scale shown to be effective for measuring sentiment and attitude [Heise, 2010, p. 35] [Babbie, 2012, p. 178] – and their responses are summarised in figure 5.9. The remaining questions are free-form and their responses will be grouped and summarised below.

A total of 23 responses to the survey were collected. The semantic differential questions suggest that respondents generally found the process of running and uploading minds to be easy, although 22% of respondents considered these tasks to be

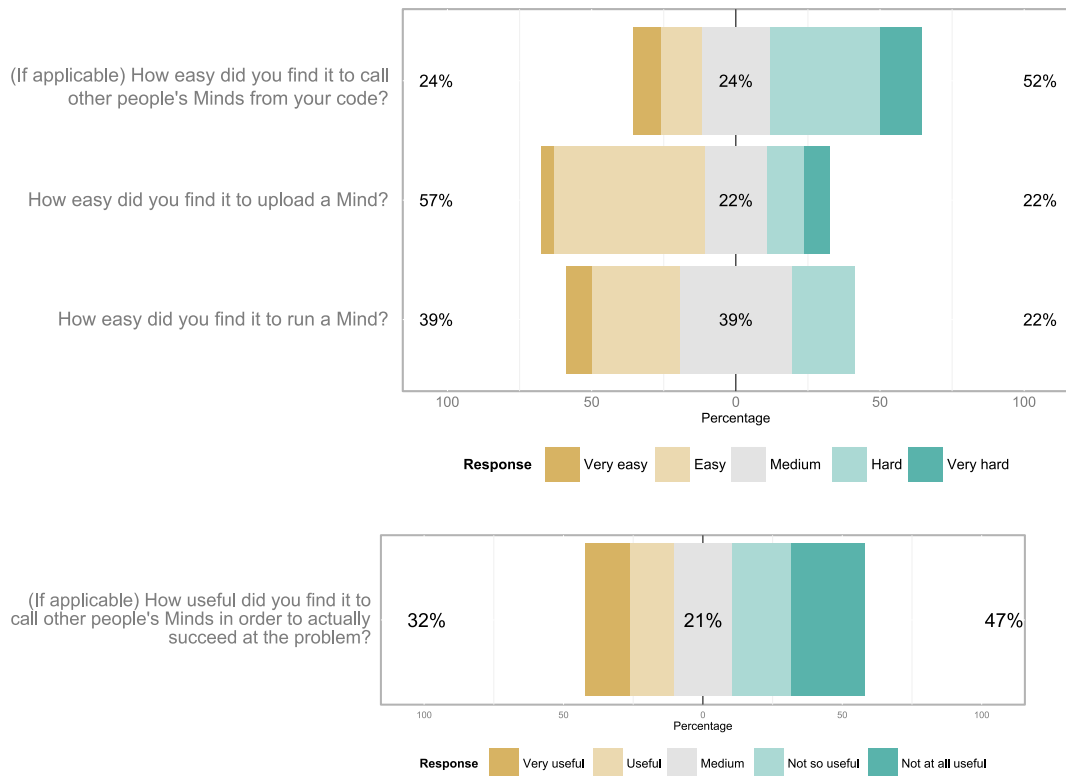


Figure 5.9: Summarised survey results, in stacked bar format.

either “hard” or “very hard”. The free-form responses help to shed some light on this, with several participants stating that the server had gone down close to a submission deadline.

A total of 52% of 21 respondents considered the process of calling another mind to be “hard” or “very hard”, while 32% of 19 respondents found this either “useful” or “very useful” in practice.

Breaking down the free-form responses by frequency:

### Q1 — How long did it take you from first reading the documentation to uploading your first Mind?

Of 20 respondents to this question, eight gave a figure of two weeks or more (excluding one answer which was more precise yet perhaps less informative: “The time between you showing us it in class for the assignment and the day before the assignment was due”).

**Q2 — What could we have done to make that time shorter?**

Of the 20 responses to this question, 11 focused on better documentation and instructions, while three cited a compatibility issue regarding Java class versions.

**Q7 — If you have ever participated in a collaborative project (e.g. at work, or an open source project) how would you compare the ease of calling other people’s Minds with the ease of calling other people’s methods/functions in those projects?**

Of the seven responses, three regarded it positively, one cited server problems, one “just wanted to write my own original mind” and one took issue with this type of behaviour-based re-use, stating:

“For our project the ability to call other people’s minds wasn’t particularly useful as we didn’t have access to individual methods such as their evaluation functions. While accessing their single action method was simple enough there wasn’t a way to discern what their mind was doing.”

**Q8 — If you noticed any bugs or problems with the system, please describe them.**

Of the 16 responses to this question, ten highlighted the server problems close to the submission deadline. One respondent pointed out that the security model prevented him from running his mind online when it had worked offline (where the security manager was not active).

**Q9 — Please describe any important features you would like to see added, if any.**

Of seven responses, three suggested a redesign of the W2M server’s web interface and two asked for the ability to call other minds in an offline setting, for testing purposes. While it is already possible to call remote minds, it may be that the added latency caused the participants’ hybrid minds to miss the 700ms time limit for each move in the *ChessWorldG* world. Another possibility is to allow the JAR files for submitted minds to be downloaded and tested offline with some additional wrapper code.

**Q10 — What did you think of the chess problem? Can you suggest other interesting problems and challenges that it would be good to have on the system?**

Seven of the 14 respondents to this question found the chess problem and the opponent A.I. to be too challenging. Others suggested alternative problems including 2D Pong, Tetris, Checkers and Noughts and Crosses.

### **Summary**

Due to the conflation of several issues – in particular the server downtime and perceived difficulty of the chess problem – it is difficult to draw firm conclusions from this qualitative analysis. It is apparent from the comments regarding the chess problem that this usability evaluation is partially task-dependent. Another approach which may produce a more robust outcome is to perform a comparative study against a similar system or methodology. This form of evaluation was applied by Gemrot et al. [2012] to compare two approaches to the design of game-playing agents in the same test environment: a traditional programming approach using the Java programming language, and a reactive planner using a graphical editor to construct the agents.

It seems clear however that the process of uploading and running minds was generally considered easy, although participants were less positive about the difficulty and utility of building hybrid minds.

## **5.10 Conclusion**

This chapter introduced the problems the World-Wide Mind 2.0 architecture attempts to address and the solutions it provides, describing in brief the back-end server implementation and the front-end user interface, which together provide not only a starting point for exploration by mind and world designers, but also a framework for the experiments described later in chapters 6 and 7.

Some of the technical challenges and methods used to reduce the latency, and therefore improve the runtime performance of the server implementation were discussed. Limitations and possibilities for future work will be outlined in chapter 8.

Section §5.9 evaluated the system as a whole and validates the work as an answer

to the first research question posed in section §1.4.



## Chapter 6

# Semi-automated hybrid building

### 6.1 Introduction

The core novel application this architecture enables is the manual or automated or semi-automated building of hybrid A.I. systems out of the code of multiple authors.

Chapter 5 described the software architecture and platform which hosts a number of mind and world services, and which allows a mind to query other mind services for suggested actions. While the subject of creating larger-scale hybrid minds using the work of others has been discussed, the question of how this might be done has not yet been addressed .

Let us recall the idea of a hybrid chess-playing mind, introduced in section §2.9 (page 26), which when asked for an action produces no move of its own, and delegates instead to one of two expert subminds which specialise in different tasks. The aim in this chapter is to build a hybrid mind for a modified version of Tyrrell's world which outperforms any individual mind without directly suggesting its own actions. Tyrrell's world will be discussed briefly in section §6.2 and described in further detail in appendix B.

This chapter describes a method which was developed for ranking existing minds according to their performance in the world on several criteria. This method provides an answer to the second research question posed in section §1.4.

By ranking the minds based on their performance on the most important subgoals (for example, in the Tyrrell world environment, these goals might be mating and minimising thirst), a set of subminds can be selected which constitute experts for

each goal. The goals were selected and ordered based on human insight into their relative importance to solving this particular problem. This may not be the best approach, and can be vulnerable to bias – chapter 7 will introduce a method which attempts to automate this goal selection and ordering step.

A series of experiments were performed with varying numbers of subminds, producing a hybrid mind which outperformed the best student-submitted mind by a margin of 10%.

This outcome indicates that even a simple approach to building hybrid minds can yield useful results. A more in-depth statistical analysis may determine the importance of each subgoal in a more objective and automated manner, and this is the topic of chapter 7, which presents such a method and evaluates it in two different worlds.

## **6.2 Problem environment: Tyrrell’s animal world**

The most challenging problem environment currently installed as a world service on the reference World-Wide Mind server is Tyrrell’s simulated animal world, introduced earlier in section 5.9.2. This simulation models a small animal in a dynamic and hostile two-dimensional grid world, and attempts to serve as a testbed for examining and evaluating different action-selection mechanisms [Tyrrell, 1993].

The primary objective for the simulated animal is to mate as many times as possible, but a large number of subproblems also require attending to if the animal is to live long enough to achieve good genetic fitness. To this end, the animal’s lifespan (the number of simulated timesteps where the mind perceives the world and selects actions, until the animal dies) is considered a secondary goal here, and is used as a tie-breaker in the world scoreboard.

The set of sensory percepts available to the mind program in this world consist of numbers representing levels of many homeostatic and external stimuli, and the actions are chosen from a set of 35 high-level activities including drinking, cleaning, sleeping, courting, mating and moving in one of the compass directions

Performing well in the world requires not only selection of appropriate actions, but at a higher level, selection of appropriate goals among competing options.

Rank	Mind	Mated	Steps	Runs	Subminds	Author
1	RobMindM	74	4567	10	2	rblestr
2	CowardlyMindFinal	63	3840	4	3	hands3
3	Mater	55	3380	2	1	murpha74
4	Bavaria	54	4306	1	1	dan
5	RTesterMind2	54	3836	0	1	rosshaugh
6	NeverMind	53	3294	6	3	rosshaugh
7	SFINALMIND2	52	4256	0	1	lawa3
8	CraigMind	52	3625	1	1	craig1928
9	TimiMind1	51	4086	0	None	milansatata

Table 6.1: The top ten highest-scoring minds for a modified Tyrrell animal world. The score components specified by the world author are “Mated” and “Steps”, and the scoreboard automatically sorts entries in descending order. In the web interface, clicking the numbers in the “Minds called” column will display the set of subminds called (if any) by a mind.

Because of the complex nature of the problem and the requirement for goal selection as well as action selection, this world was selected for use in an undergraduate A.I. course and implemented in Java by Ciarán O’Leary as part of his research [O’Leary et al., 2004].

### 6.2.1 Use in student assignments

In this work, a modified version of Tyrrell’s simulated environment – called Tyrrell09 – is used which attempts to simplify the problem and reduce noise in the perceptual and motor control systems. A more detailed description of Tyrrell’s world and the modified environment can be found in appendix B.

A collection of over 100 minds was created and uploaded by undergraduate students and their performances automatically evaluated. This set of minds was used to perform the experiments in submind selection which are described in this chapter.

For one assignment, a requirement was added that every student must submit at least one hybrid mind which delegates to one or more subminds.

A call graph feature was implemented to keep track of calls between minds, so that the number of unique subminds called by a mind is saved and displayed alongside its performance on the scoreboard for that world.

At the end of the assignment period, the scoreboard (see table 6.1) showed that almost all of the top ten minds called at least one other submind during a run.

## 6.3 Experimental setup

Following the use of the World-Wide Mind reference server as a hosting platform for minds submitted by undergraduate students to compete against one another at solving the Tyrrell world problem for several weeks, a large set of possible subminds was gathered from those students who agreed to make their work available.

A set of programs and scripts was developed which runs every mind in this world and records some of the output for later analysis. This recorded performance data contained the scores achieved and states observed over three runs by each mind, and this data was analysed for the purpose of creating a new hybrid mind which calls the existing minds to achieve better scores without directly contributing any actions of its own.

### 6.3.1 Experimental hypothesis and procedure

The best mind submitted to the system serves as a control benchmark against which to evaluate the experimental hybrids. For a new hybrid to be considered successful, it must perform better than the best existing mind – otherwise, it could just call the best mind directly and contribute nothing of value.

To define what it means to perform better than another mind, it must be noted that the *Tyrrell09* world is stochastic and there is a large degree of variability in results – two runs of the same mind can produce significantly different outcomes. To limit the impact of the stochastic nature of the world on the evaluation process, each mind must be tested a large number of times and its performance treated as an aggregate. A median and mean score were established for the control mind – *RobMindM* – and for the most promising hybrid mind by running each in the world for a large number of trials ( $N = 1000$ ) and recording the results.

This enables us to define the *experimental hypothesis*:

The mean score of the hybrid mind created in this chapter is significantly greater (better) than the mean score of the control mind.

To test this hypothesis, the following *null hypothesis* is defined:

There is no significant difference between the mean scores of the hybrid mind in this chapter and of the control mind.

The chosen significance criterion is that the probability of such a result arising by chance must be less than 5% ( $p < 0.05$ ). A t-test of the recorded samples from both minds must return a p-value of less than 0.05 in order to reject the null hypothesis.

## 6.4 Selecting and ordering the differentiating score attributes

At the end of a run in Tyrrell's world, a score vector is generated, summarising the mind's performance at solving the problem. The attributes of this score vector include the number of times the animal mated and the number of steps survived, which are used by the world scoreboard to rank scores. Additionally, information is produced regarding significant events which happened during the run, such as how many times the simulated animal drank water or slept.

Some of this data can be used to formulate an evaluation metric for classifying the behaviour of individual minds. For example, the mind that eats the most food could be considered to be an expert food finder/eater, and the mind that mates more than any other might be considered an expert mater.

If an ordering could be established of how each metric contributes to overall success in solving the world problem, then a set of expert subminds would emerge which might form part of an effective hybrid mind.

To test whether this is the case, a series of experiments was devised. The metrics were ordered based on human insight into their significance in solving the problem – in chapter 7, a method of analysing the performance data to infer this importance ranking in a more rigorous and automatic way will be demonstrated.

For each of the top  $N$  most important metrics (where the value of  $N$  varies depending on the experiment, from 2 to 6), the mind which achieves the best score for that particular attribute was found by parsing and sorting the complete set of performance data.

### 6.4.1 Experimental results

The results of the first experiments (shown in table 6.2) in selecting subminds by data mining the score information alone were mixed.

This could indicate that some of the score information does not completely capture the simulated animal's condition and the subgoals needed for effective survival. For example, a mind could score highly in eating food by simply finding a food source and engorging upon it and then sleeping until the run ends, but the simulated animal may be hungry for 90% of the run. However, in the next chapter, a more automated method of analysing the score data was successful in producing a hybrid mind which contributes no direct actions of its own, yet outperforms all of the individual subminds.

The next section discusses mining the data on a state-by-state basis to make more informed decisions.

## 6.5 Mining the information on a state-by-state basis

Using the summary information contained in the score vector can be an effective way of classifying and ranking the set of available minds. However, the performance of the hybrid mind created by profiling the score data was not particularly impressive. To improve on this, further experiments were performed, this time mining the data captured on individual states encountered by each mind. This was more computationally intensive, but allows us to make more fine-grained distinctions. For example, instead of looking solely at the total amount of food eaten in the score summary, we can take the animal's perceived hunger senses at each timestep and calculate the sum. Weighting or squaring of the deviation from a healthy value (usually 0.5, in this world) may help to produce a more effective fitness function, where maintaining a consistently healthy level of hunger or thirst is rewarded more highly than quickly reaching a very high level and then neglecting it.

## 6.6 The hybrid mind controller

Having created a list of expert subminds by establishing rankings for each important score or state metric, a hybrid mind was created in each experiment which defers

to one of the subminds based on a simple decision list. At each timestep in a run, the hybrid mind's *getaction* method is called, which consists of a sequence of condition-action rules making up the decision list. These rules correspond directly to the metrics which are considered most important to the overall performance of a mind and are executed in descending order of perceived importance. When a rule is matched, the submind responsible for that metric is sent a *getaction* request and will reply with a suggested action which can be taken by the hybrid mind without processing any further rules, until the next *getaction* request is received and the rules are tested again from the top.

Some example pseudocode for a hybrid mind based on a condition list structure is shown in algorithm 6.1, and a concrete example in Java is included in section §H.2 on page 213.

---

**Algorithm 6.1** Example pseudocode for a simple hybrid mind controller based on a condition-action rule table.

---

```

function getaction()
  if mate is nearby then return mating_mind.getaction()
  if thirst > thirst_threshold then return drinking_mind.getaction()
  if health < health_threshold then return survivor_mind.getaction()
  else return mate_finder.getaction()
end

```

---

## 6.7 Experimental results

In these experiments,  $n$  subminds were selected (where  $n = 1.4$ ) by walking through the collected state information for each mind and classifying it according to several simple fitness functions which maximise or minimise values, or which minimise the deviation from an optimum level (often 0.5 in this world, but not always – for example, cleanliness should be as close to 1 as possible).

### 6.7.1 Preliminary results

Some of the hybrids built using these subminds performed well, as can be seen in the initial results in table 6.2, where a hybrid mind constructed from four existing subminds achieved results superior to those of the existing minds.

The successful hybrid was capable of outperforming the subminds without needing to know how to contribute its own actions in the world, although it is of course

	Subminds	Analysis	Max mates	Max life	Median mates	Median life
1	4	State+score avg.	16	724	6	419
2	4	State (avg./step)	36	3269	23	2512
3	4	Score	52	3762	20	2264
4	2	State	70	4840	48	4358
5	4	State+score	72	4873	49	4213
6	2	Best existing mind	74	4567	-	-
7	3	State	74	4877	48	4212
8	4	State	82	4797	49	4226

Table 6.2: Results of the preliminary experiments in constructing hybrids, ordered by the maximum value of the “mated” primary score measure over several hundred runs of each mind. The mind in row 6 was the existing best mind on the scoreboard, included as a control. The mind in row 8 consisted of four subminds, selected by evaluating the states seen to identify minds which constitute experts in four separate behaviours relevant to the Tyrrell world.

possible for a hybrid mind to both delegate to subminds and issue actions directly during a run.

On the basis of these results, the hybrid mind shown in row 8 of table 6.2 was selected for a more thorough evaluation over  $N = 1000$  runs.

### 6.7.2 Evaluation

The hybrid mind created in this chapter was run in the same world with the same parameters as the subminds. The best, median and mean scores achieved by both minds after  $N = 1000$  runs are displayed in table 6.3, along with the standard deviation in each score variable. The score data are summarised with boxplots of the *mated* and *lifespan* score samples for both minds in figure 6.1.

The performance of the hybrid mind created in this chapter is shown on column 3 of table 6.3, having outperformed all of the subminds submitted by users, without ever needing to know how to act directly in the world.

To determine whether these results are significant, Welch’s t-test was performed<sup>1</sup>, comparing the means of the existing best mind and the hybrid built by mining the manually-selected state data. The results of these tests are as presented in table 6.4.

The outcome of the t-test allows us to reject the null hypothesis – that the

---

<sup>1</sup>Welch’s t-test is used when the two samples may have unequal variances [Welch, 1947]. In any case, the results for Student’s t-test are almost identical to those of Welch’s test for these samples.



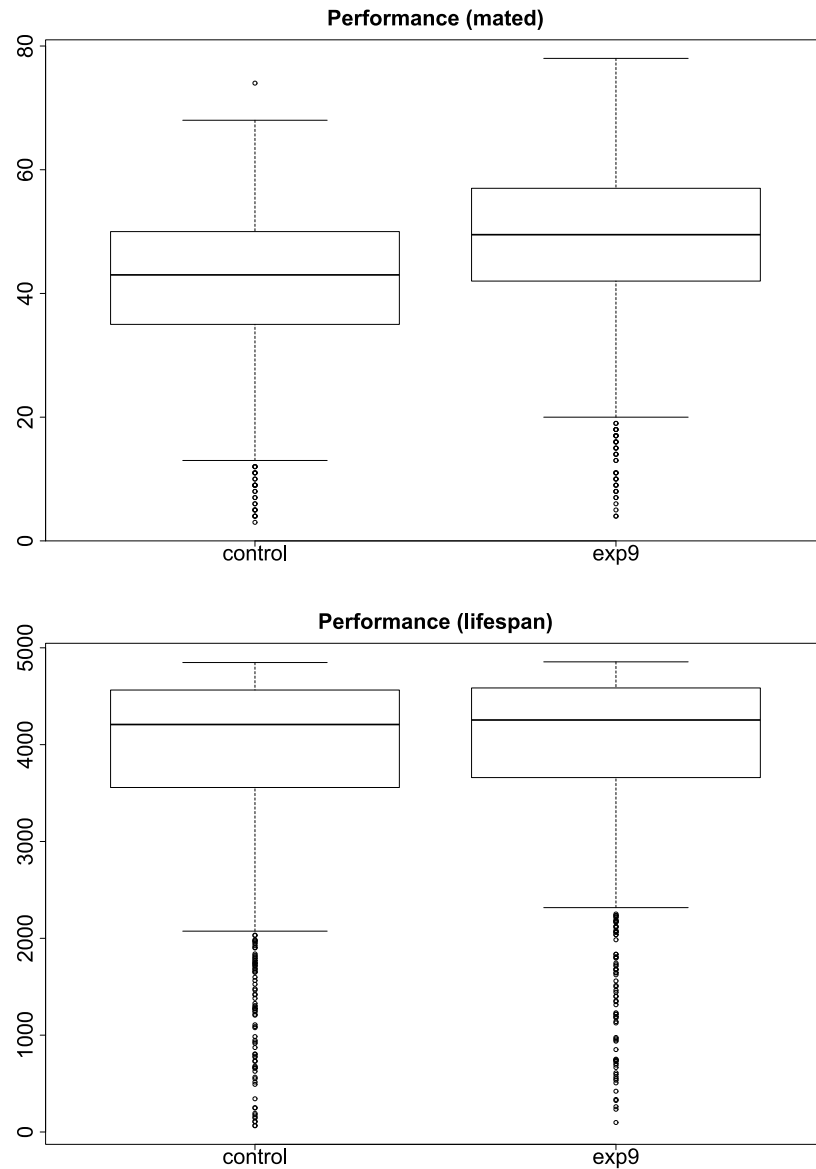


Figure 6.1: Boxplots of the performance in *Tyrrell09* of the control mind and the hybrid mind for the *mated* and *lifespan* score variables. The label *control* indicates the best mind used as a benchmark against which the new hybrid mind is tested. The label *exp9* refers to the hybrid mind created in this chapter. This mind exhibits better performance in the *mated* score variable than the control mind.

	Existing best mind	State analysis
<b>Subminds</b>	2	4
<b>max(mates)</b>	74	78
<b>median(mates)</b>	43	49.5
<b>mean(mates)</b>	41.533	47.95
$\sigma(\text{mates})$	12.53204	12.63523
<b>max(life)</b>	4848	4797
<b>median(life)</b>	4208	4254
<b>mean(life)</b>	3849	3942
$\sigma(\text{life})$	1019.069	934.176

Table 6.3: The hybrid mind in the rightmost column was constructed by selecting subminds according to a ranking of important variables from the world state. The middle column shows the performance of the existing best mind submitted to the server by a student.

( $N = 1000$ )	Manual state-mined hybrid
Mean mates	47.95
Existing best	41.533
p-value	$< 2.2 \times 10^{-16}$
Mean lifespan	3942.08
Existing best	3848.696
p-value	0.03279

Table 6.4: Results of t-tests for the hybrid mind. The mean for the “mated” score variable is significantly greater (better) than the mean recorded for the best existing mind. For completeness, the “lifespan” variable is also included in the analysis and is also significantly greater, albeit with a lesser degree of confidence.

difference in means between the hybrid mind and the control occurred by chance – and therefore conclude that the hybrid mind is superior to the best existing third-party mind.

## 6.8 Conclusion

This chapter dealt with the idea of constructing a hybrid mind by selecting from a set of “subminds”, programs possibly written by third parties intended to solve the same world problem. These results support the idea that a hierarchical hybrid mind constructed in this way can be superior to the subminds from which it was constructed, which was the second research question in section §1.4.

### 6.8.1 Finding expert subminds

In choosing appropriate subminds for use in a hybrid mind, we can look for behaviours carried out by successful minds which can be regarded as expert behaviours.

From all of the summary variables presented at the end of each run – the score – and from the variables observed in each world state reached during a number of runs, sets of these variables were chosen as metrics by which minds could be evaluated for inclusion as an expert submind in a larger hybrid mind.

Once the appropriate metrics have been selected, and a ranking of minds established for each one, a new hybrid mind can then be built, perhaps using the condition-action rule template shown in algorithm 6.1.

### 6.8.2 Limitations of this approach

Defining what constitutes expert behaviour, and ranking the relative worth of multiple behaviours, is critical to the construction of an effective hybrid mind. In the experiments above, the metrics chosen, and the order in which the desired behaviours are prioritised were based on human judgement. To produce a hybrid mind required a lengthy period of trial and error experimentation, in finding the right set of subminds and in developing a successful controller.

This process requires significant experience with and understanding of the problem world, and can easily result in the selection of behaviours which might intuitively seem important, but may in practice be ineffective. A more objective, mathematical method for making these decisions would be more generally useful, and this is the topic of the next chapter.

Another possible limitation with any approach of the sort (this is discussed on page 39 of this dissertation and by O’Leary et al. [2004]) is that a submind may implement a memory that tracks parts of the observed state and/or actions taken – relying on the implicit assumption that the submind is always in control, when in fact it is not. Violation of this assumption may lead to a faulty understanding of the world and therefore produce faulty behaviour.

## Chapter 7

# Towards more automated hybrid building

### 7.1 Introduction

The previous chapter presented a method for selecting appropriate subminds during the construction of hybrid minds. This method requires an understanding of which score and state attributes should be maximised or minimised, and in which order.

While useful results have been demonstrated using this method, it would be better if the relative importances of these ranking metrics could be discovered algorithmically.

The novel contribution of this chapter is an extension of the method introduced in chapter 6 with a statistical approach to evaluating and ranking the various performance metrics provided by the score vector in a more objective and automatic way. The recorded state information was not used for the experiments described in this chapter.

First, the primary techniques used to perform the analysis and selection of minds are introduced. Then the experimental setup will be discussed and the experimental and null hypotheses defined, before going into a more detailed treatment of the experiments and the results, which show that the hybrid mind created using this method is superior to the best of a number of minds submitted by multiple competing authors. Finally, the conclusion section will discuss some limitations and general observations.

### 7.1.1 Statistical methods used

Several analytical methods are discussed in chapter 3 which may be of use in solving this problem. For this work, two techniques are used to analyse the same dataset used in chapter 6:

- **Correlation analysis**, which allows us to rank the relative importance of each metric, relative to an objective measure of overall performance on the problem at hand, and
- **Principal Component Analysis (PCA)**, which allows us to look at the dataset and reduce it to a smaller set of dimensions which explain a large proportion of the total variance within the dataset.

Both of these techniques are briefly explained, along with a discussion of how they inform the selection of subminds for the purposes of building a larger hybrid mind.

## 7.2 On the importance of context

For an action-selection mechanism to perform adequately in a non-trivial environment, the appropriate exploitation of context is key. This is what distinguishes a strategy of “always attempt to drink” from “attempt to drink if there is water present and you are thirsty”. The first strategy is blind while the second demonstrates an understanding of the context in which those actions are appropriate.

In the method proposed here, there are three places where context might be considered:

- The subminds,
- The hybrid control mind, and
- The ranking and selection system.

Both the subminds and the hybrid controller directly sense the state of the world, and therefore have access to the information needed to select context-appropriate actions.

The ranking and selection system in this particular experimental setup can only observe the performance of each mind after a run has been completed, and does

not play a direct role in the selection of actions. Furthermore, to automatically distinguish between these two types of activity – between inappropriate and context-guided – is a difficult task. The only attempt to do so in the ranking and selection system is to weight the perceived importance of each behaviour with the tested mind’s overall performance measure, to help weed out minds which perform an “important” action repeatedly and at inappropriate times.

How the subminds respond to context is up to the mind authors, and therefore cannot be controlled by the method proposed here. This leaves the hybrid control mind with the responsibility of selecting the appropriate submind at the right moments. There are numerous methods which could be used to design the hybrid controller, and only two are used here – an ad-hoc condition-action list, and a reinforcement learning agent.

### **7.3 Method**

The problem we are faced with is how to construct a hybrid mind which uses the most successful behaviours represented among a large pool of possible subminds. How can we classify behaviours as successful, and how can we select a small number of subminds which constitute expertise in those behaviours?

This section summarises the approach taken in an attempt to answer these questions and produce a method for constructing a useful hybrid mind in a more automated way.

#### **7.3.1 Rank the variables using correlation analysis**

First, we attempt to rank the variables which are used to summarise the minds’ performance in the world score vector. These variables provide information about how often certain conditions and events occurred, and how often each action was taken during a run.

The ranking will be based not on how much variance is accounted for, but rather to what degree each variable affects a scalar measure of success in the world. We will use correlation analysis for this purpose in section 7.6.2.

Returning to the problem world used for these experiments – the Tyrrell simu-

lated animal world – we define our measure of success simply as the number of times the simulated animal successfully mated. For other worlds, several important output variables may be composed into a scalar value by means of a weighted sum function, but in our case the value can be used directly.

### **7.3.2 Eliminate redundant variables**

After the variables have been ranked in descending order by the strength of their correlation with the success measure, there may be multiple variables present which account for the same behaviour. To reduce the number of variables (and subminds), two methods are examined and evaluated: principal component analysis in section §7.8, and a method which re-uses the correlation matrix to eliminate redundant variables in section §7.9.

### **7.3.3 Construct the hybrid mind**

After a small set of ranked, relatively independent variables has been created, each variable is then used for two purposes:

1. To select a submind which represents expertise in the behaviour associated with that variable. This is done by choosing the mind which maximises that particular variable and designating it as the submind responsible for that behaviour.
2. The event measured by that variable is then used to devise a simple conditional test, for use in the hybrid mind controller, which when triggered will request and return a suggested action for the submind chosen in step 1.

The conditions are tested, where possible, in order of the underlying variable’s ranked correlation with the success measure.

## **7.4 Experimental setup**

The experiments described in chapter 6 used a dataset gathered by running each of the uploaded minds several times in the modified Tyrrell world and saving a record of the summary score vector produced after each run, which contains counts of important events that occurred and actions which were taken in the world.

### 7.4.1 Test scenarios

To evaluate the method proposed in this chapter from a more general perspective, it will be tested in two worlds which are quite different from each other:

**Tyrrell09** — An animal behaviour simulation which models a small animal in a complex two-dimensional grid world, where the goal is to increase the animal’s genetic fitness by finding and mating with receptive partners. This world is used to evaluate the method presented in section §7.3, by constructing and testing a hybrid mind against the best of a large number of minds submitted by many authors in a competitive assignment set in the *Tyrrell09* world. A more complete definition of this world can be found in appendix B.

**ChessWorldG** — A world which models the game of chess<sup>1</sup>. The mind plays as white in each game, while the A.I. opponent provided by the world plays as black. The fitness of each mind will be measured by the “survival moves” variable; a count of the total moves made by the mind over five games. This variable was used as a measure of fitness because only one of the student-submitted minds was able to defeat the A.I. opponent, and the one successful mind simply reused the world’s A.I., a specialised open-source chess engine called CuckooChess. For this reason, that mind is excluded from the analysis and not used as part of the hybrid built here. A more complete description of the world can be found in appendix C on page 187.

### 7.4.2 Experimental hypothesis and procedure

As in chapter 6, the best mind submitted to the system serves as a control benchmark against which to gauge the effectiveness of the hybrid creation method proposed here. A median and mean score were established for this mind – in the Tyrrell09 world, *RobMindM*, and in the *ChessWorldG* world, – and for the new hybrid mind by running each in the world for a large number of trials ( $N = 1000$  for Tyrrell09 and  $N = 100$  for ChessWorldG) and recording the results.

---

<sup>1</sup>The motivation here was to test these methods in a world substantially different to Tyrrell’s world, and ideally a world not created by myself. My initial choice was a third-party world implementing the game of Minesweeper, but it was not appropriate due to a lack of subminds and the fact that one submind appeared to provide an optimal solution, meaning there was no benefit to be gained by creating a hybrid mind for that world. At this point, chess was selected as the target problem, using an open-source chess engine to implement the opponent.



This enables us to define the *experimental hypothesis*:

The mean score of the hybrid mind created through the method described in this chapter is significantly greater (better) than the mean score of the control mind.

To test this hypothesis, the following *null hypothesis* is defined:

There is no significant difference between the mean scores of the hybrid mind created through the method described in this chapter and of the control mind.

The chosen significance criterion is that the probability of such a result arising by chance must be less than 5% ( $p < 0.05$ ). A t-test of the recorded samples from both minds must return a p-value of less than 0.05 in order to reject the null hypothesis.

### 7.4.3 Collecting data

As discussed in chapter 6, over 100 mind programs were created for each world and uploaded by undergraduate students during the assignment period. Each of these minds was run and evaluated, and their best performances were recorded on a scoreboard.

A small number of obviously broken minds were excluded for the purposes of this analysis, leaving 120 mind programs for *Tyrrell09* and 111 for *ChessWorldG*. These were evaluated by running each mind in the *Tyrrell09* world three times and recording the resulting score. The *ChessWorldG* minds were run ten times each.

Several minds still failed to complete some or all of their runs before a preset timeout in the evaluation script, and at the end of the data collection process, scores for *Tyrrell09* and *ChessWorldG* were gathered for 301 and 1093 runs respectively.

Note that due to the non-deterministic nature of *Tyrrell09*, the sample size for each mind of  $N = 3$  is likely to give an incomplete picture of their performance. This was done for practical reasons, since a number of the minds are themselves hybrids which take longer to complete runs, and several minds timed out during the data collection process. However, for the experimental evaluation described later, a larger number of trials ( $N = 1000$  for *Tyrrell09*,  $N = 100$  for *ChessWorldG*) was performed to ensure statistically meaningful measurements.

#### 7.4.4 Constructing a hybrid mind

Based on a correlation analysis of the collected data, the 5 metrics most strongly associated with success in the world were chosen. Then, the same dataset is used to select the minds which scored highest under each of these metrics.

A hybrid mind is then constructed, using each of these subminds, based on a simple condition-action list controller in the case of the *Tyrrell09* world (see algorithm 6.1 on page 110), or a  $Q(\lambda)$  reinforcement learning agent in the case of *ChessWorldG* (this will be described in section §7.11).

### 7.5 Correlation analysis

The statistical analysis of correlation allows us to look for signs of a dependence between two variables. Correlation is generally expressed in the form of a scalar coefficient in the range -1 to +1, where +1 indicates a perfect positive correlation, -1 indicates a perfectly negative correlation (anticorrelation) and values in between denote the strength of the association.

We will use correlation analysis to infer relationships between various aspects of behaviour and overall success in the world. Correlation does not necessarily imply a causal nature in these relationships, but it may serve as a useful heuristic for the purpose of constructing hybrid minds.

### 7.6 Ranking the performance metrics automatically

In chapter 6, the performance metrics provided by the score vector and the recorded states were manually reduced to a small set of important variables and ranked. Although this produced good results, it would be useful to perform the reduction and ranking processes automatically.

This section describes how correlation analysis is used to determine the statistical dependence between each variable in the score vector and the overall success measure. With that information, the variables are ranked in descending order and subminds are selected which perform well at the behaviour represented by each variable.

### 7.6.1 Measuring the correlations between performance metrics and overall success

The Tyrrell world offers two important score measures by which to evaluate the performance of a mind during a particular run:

1. the reproductive fitness of the simulated animal – that is, the number of times the animal successfully mated, and
2. the lifespan of the animal – the total number of discrete timesteps for which the animal survived.

For our purposes, we shall focus on the number of times the animal mated. As we shall see, this is influenced somewhat by the animal’s lifespan. In *ChessWorldG*, the measure of fitness is the “survival moves” variable which counts the number of moves made across each of the five games played by the mind.

Although in this instance we use only a scalar value as our success metric, in other worlds, it may be useful to define a *fitness function* which provides a heuristic measure of overall success in solving the problem. A fitness function of this type might be expressed as a weighted sum of important elements in the score vector, or of some of the values observed in the states experienced during the run.

The R statistical analysis package [R Development Core Team, 2012] was used to inspect the recorded performance data of each run, and to measure and rank the correlations found between each score value and the fitness measure for that run – for this world, the number of times the simulated animal successfully mated.

To perform this analysis, the data was prepared as a text file in comma-separated values (CSV) format. The file can then be loaded into R:

```
scores = read.csv(file="scores.csv", head=TRUE, sep=",")
```

A matrix of correlations can then be calculated. R provides methods to derive the Spearman, Kendall and Pearson coefficients. In this case, we shall use the Spearman rank correlation coefficient, a statistical measure of the dependence between two variables.

The Spearman coefficient is a non-parametric measure which does not make an assumption of linearity in the variables’ relationship (as Pearson’s correlation does) – rather, it assumes that there is a *monotonic relationship* [Lowry, 2010].

A monotonic relationship between two variables  $x$  and  $y$  means that either:

1. as the value of  $x$  increases, the value of  $y$  always *increases*, or
2. as the value of  $x$  increases, the value of  $y$  always *decreases*.

The magnitudes of the changes in  $x$  and  $y$  need not be related.

The robustness of Spearman’s correlation under weaker assumptions is important given that the behaviour of a given world is often unknown and complex. However, it should be noted that all of these methods have difficulties measuring non-linear relationships, and in these conditions will underestimate the degree of association [Lieberson, 1964].

```
corrs = cor(scores, method="spearman")
```

A diagram summarising the correlations for *Tyrrell09* is shown in figure 7.1 on the following page, with the *ChessWorldG* correlation matrix displayed in figure 7.2.

### 7.6.2 Ranking the correlations

We wish to identify the elements within the score vector which have the greatest positive impact on the overall performance of the mind. To do this, we can sort the score attributes in descending order of the correlation strengths derived above.

These correlations can be ranked as follows, where `-fitness` represents the value of the “mated” score variable in the *Tyrrell09* world, or the “survival moves” variable in *ChessWorldG*:

```
cs = data.frame(corrs)
sorted_corrs = cs[with(cs, order(-fitness)), ]
```

The five strongest and five weakest correlations obtained via this method are listed for *Tyrrell09* in table 7.1 and for *ChessWorldG* in table 7.2, excluding observations with missing values.

As one would expect, correlating a variable against itself produces a result of 1. Therefore, a sensible first step in constructing an effective hybrid mind might be to select one submind solely on the basis of achieving the best mating score.

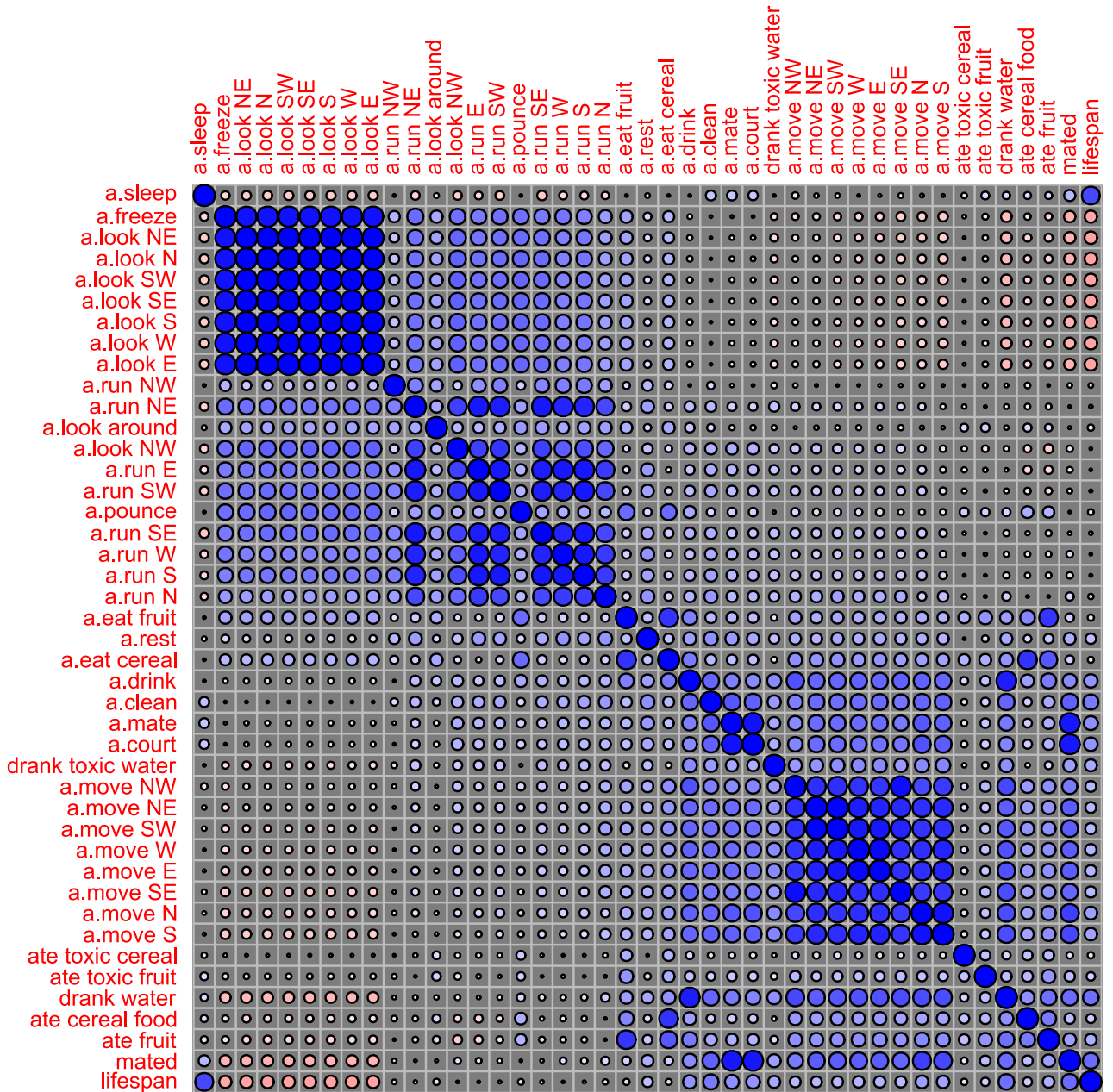


Figure 7.1: A graph summarising the correlation matrix for the Tyrrell world score data. The size of each circle denotes the magnitude of the correlation between the variables corresponding to that row and column. Blue circles denote positive correlation and pink circles denote negative correlation.

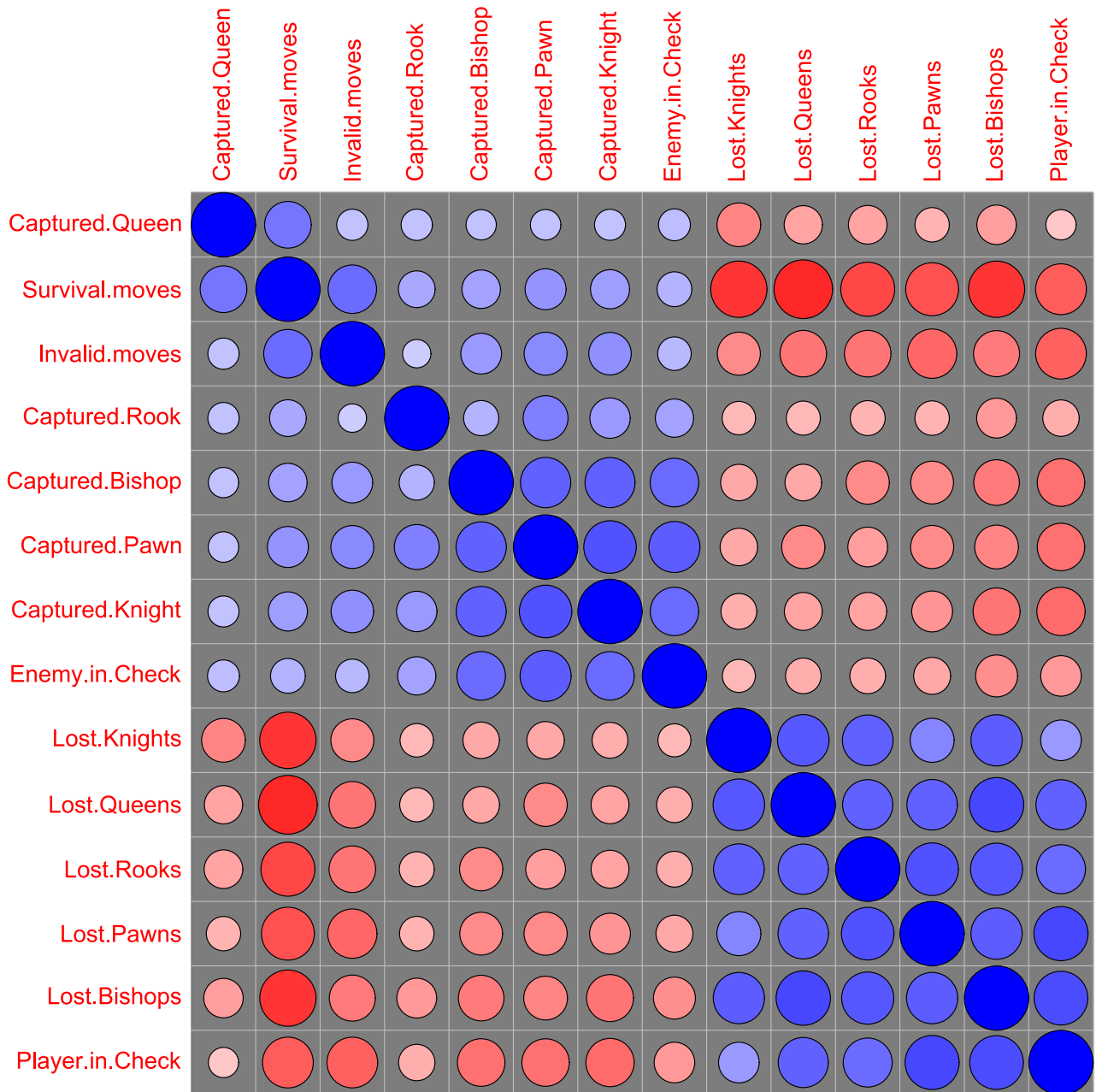


Figure 7.2: A graph of the correlation matrix for the *ChessWorldG* score data. The size of each circle denotes the magnitude of the correlation between the variables corresponding to that row and column. Blue circles denote positive correlation and pink circles denote negative correlation.

-	mated
mated	1
court-mate	0.896437454
attempt-mate	0.887892958
move-north	0.730114876
move-south	0.706036127
...	...
look-southwest	-0.283351808
look-southeast	-0.283444079
look-north	-0.284615951
freeze-in-place	-0.287978008
look-northeast	-0.29064326

Table 7.1: The five most positive and five most negative correlations obtained by the Spearman method, using the dataset gathered from the minds submitted to solve the *Tyrrell09* problem.

-	Survival.moves
Survival.moves	1.0000000
Invalid.moves	0.5651416
Captured.Queen	0.5250827
Captured.Pawn	0.4044723
Captured.Knight	0.3677592
...	...
Lost.Pawns	-0.6742261
Lost.Rooks	-0.7010840
Lost.Bishops	-0.7624508
Lost.Knights	-0.7632312
Lost.Queens	-0.8289098

Table 7.2: The five most positive and five most negative correlations obtained by the Spearman method, using the dataset gathered from the minds submitted to solve the *ChessWorldG* problem.

From there, we can see that the “court mate” and “attempt mate” actions produce strong correlations with overall reproductive success, as observed in this data.

### 7.6.3 Examining the Tyrrell09 data

At the bottom of table 7.1, we can see that the *look-[direction]* actions appear to exhibit an inverse correlation with successful mating behaviour. A closer examination of the dataset shows that the *look-northeast* action was performed only 248 times in total across all runs of all the minds. In contrast, the action for attempting to mate was performed 3,945 times.

It is possible, then, that only a small handful of mind programs performed the *look-northeast* action, and that the overall effectiveness of those minds was poor. In

fact, the *look-north* action was performed in only 39 of the 301 runs (13%), and in those 39 observations the average reproductive fitness was a score of 2.1 successful mating attempts, compared with 10.3 in the minds that did not perform the *look-north* action.

This does not necessarily mean that the *look-[direction]* actions are inherently counterproductive – rather, there is not sufficient evidence to confirm that they are useful, compared to other actions which were performed more frequently by successful minds. These examples demonstrate that having a small number of observations of an attribute across the total dataset can lead to faulty conclusions. This potential bias may be mitigated by collecting more minds, or by constructing confidence measures for each variable.

Several actions and events proved to have a weaker than anticipated effect on the performance of minds. For example, the relatively weak value of eating cereal food is demonstrated and discussed briefly in figure 7.3.

Similarly, many minds are capable of achieving a long lifespan without mating at all, which slightly weakens the correlation between lifespan and mating (see figure 7.4), although a longer lifespan enables more mating attempts to be made.

Other actions have a stronger influence on the performance of a mind. Shown in figure 7.5 is a graph of reproductive success against the number of times the mind selected the “*move north*” action.

This analysis does not perfectly capture the relationship between the input and output variables. In particular, many of the score attributes have a curvilinear relationship with the output variable, where too little or too much damages the simulated animal’s performance.

For example, not drinking any water reduces the lifespan of an animal, but conversely drinking too much can also cause injury which will impact the animal’s survival prospects. This non-linear relationship is explored in figure 7.6.

## 7.7 Selecting subminds

At this point, an ordering has been established by which we can relate each score metric to the overall success of a mind. Using this information, we can rank the



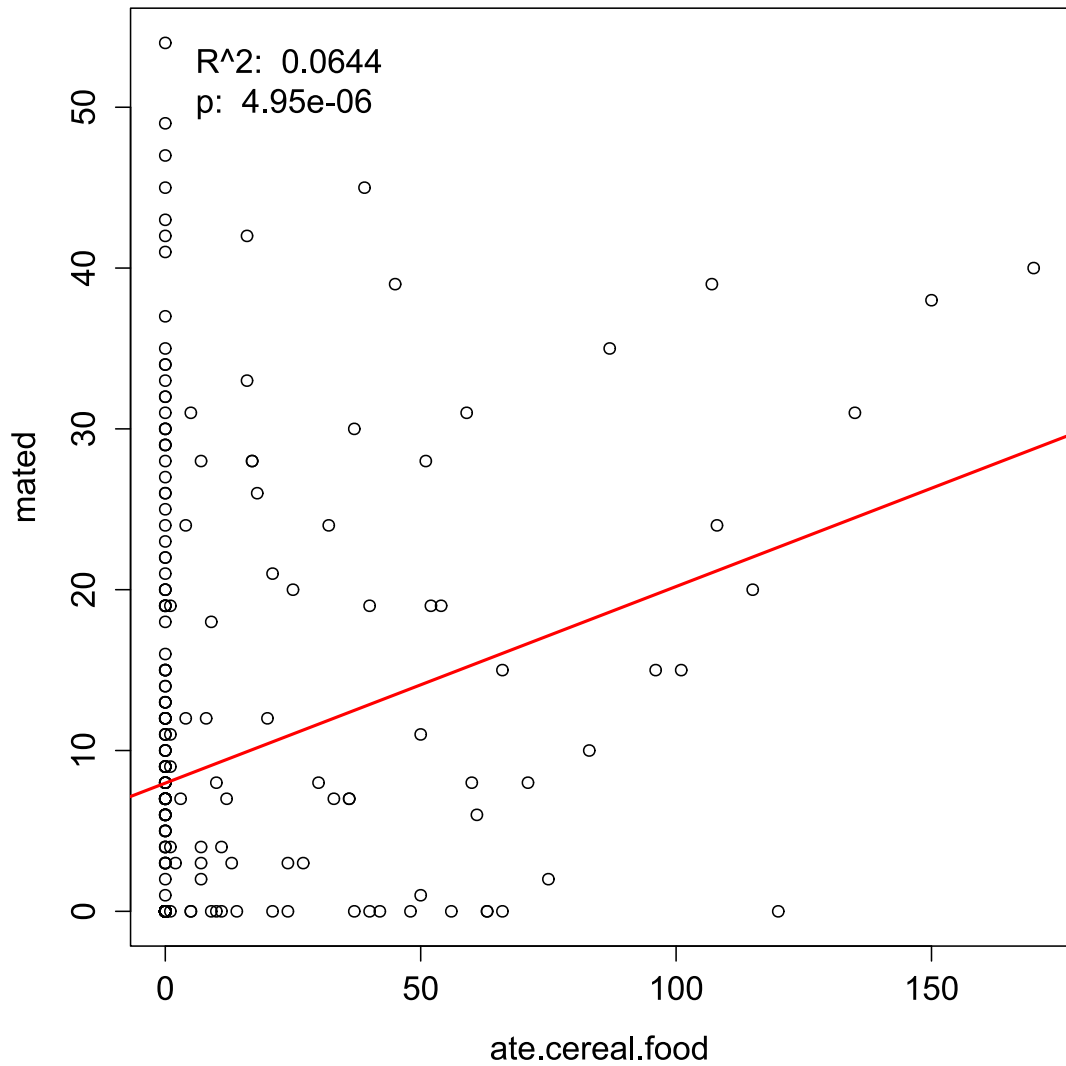


Figure 7.3: This graph links the amount of cereal food eaten with the reproductive success of the simulated animal. The prevalence of noise in the graph, and the presence of minds which scored across the entire range of the “mated” variable, suggest that eating cereal food has only a small effect on the success of a mind – at least in the mind programs collected and examined in this analysis. The estimated probability of error ( $p$ ) is low, due to the number of samples taken.

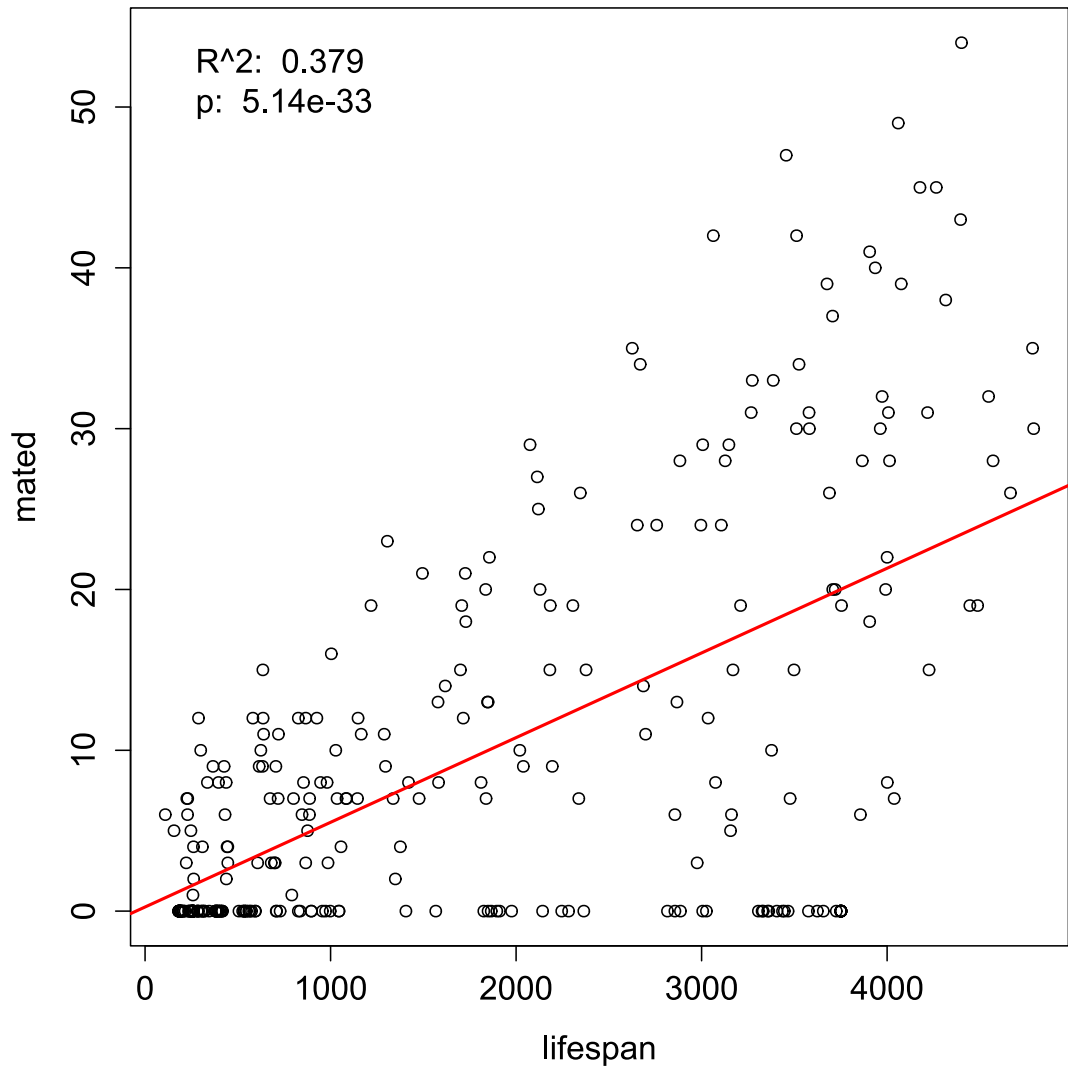


Figure 7.4: A positive correlation is observed between the simulated animal's reproductive success and the lifespan of the animal. The link is made weaker by the presence of many minds which achieve a long lifespan without mating, since moving around and mating causes significant energy expenditure which must be replenished by finding food and water.

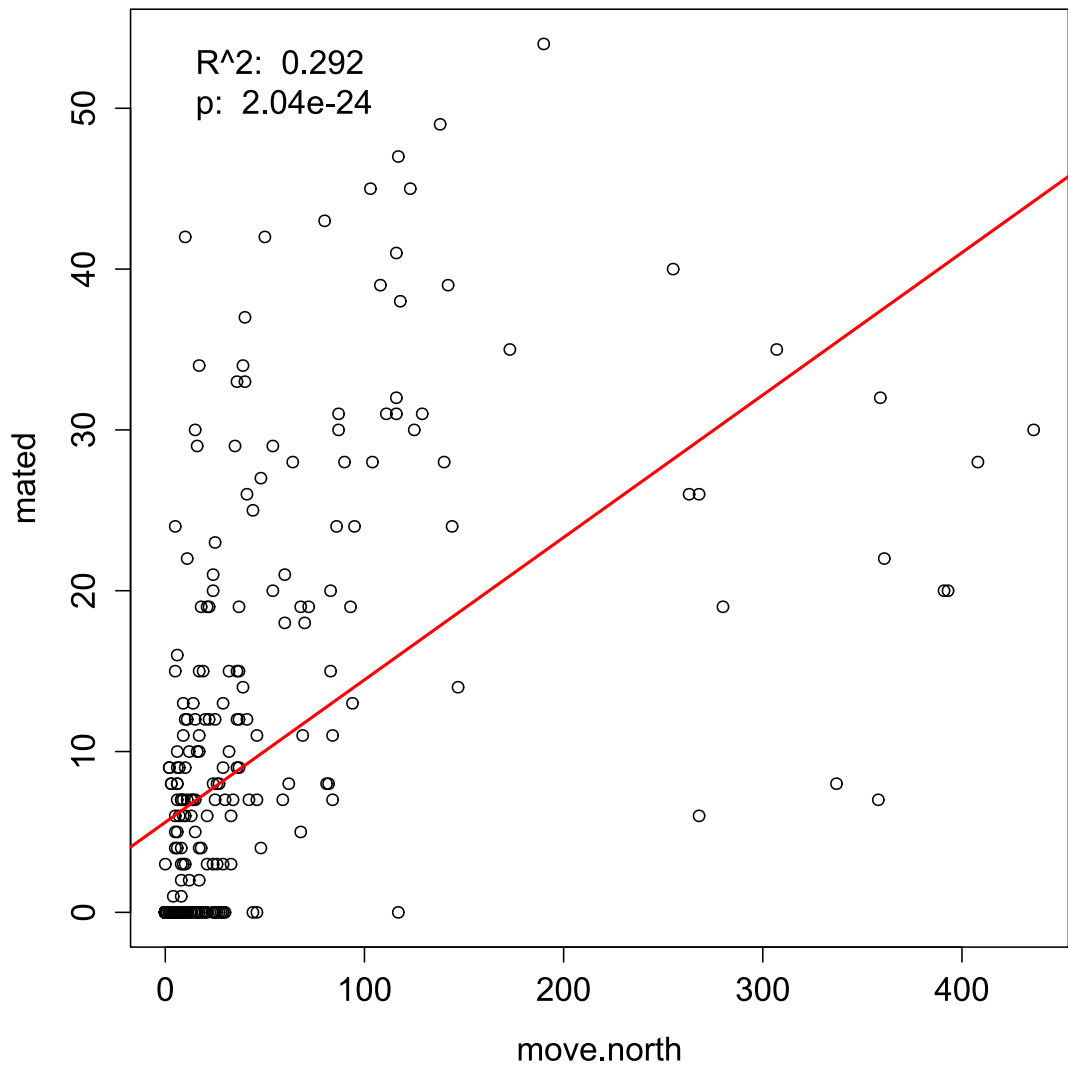


Figure 7.5: In this graph, a positive correlation between the “move north” action and reproductive success is seen. This may simply indicate that the more active animals tend to find mates more frequently.

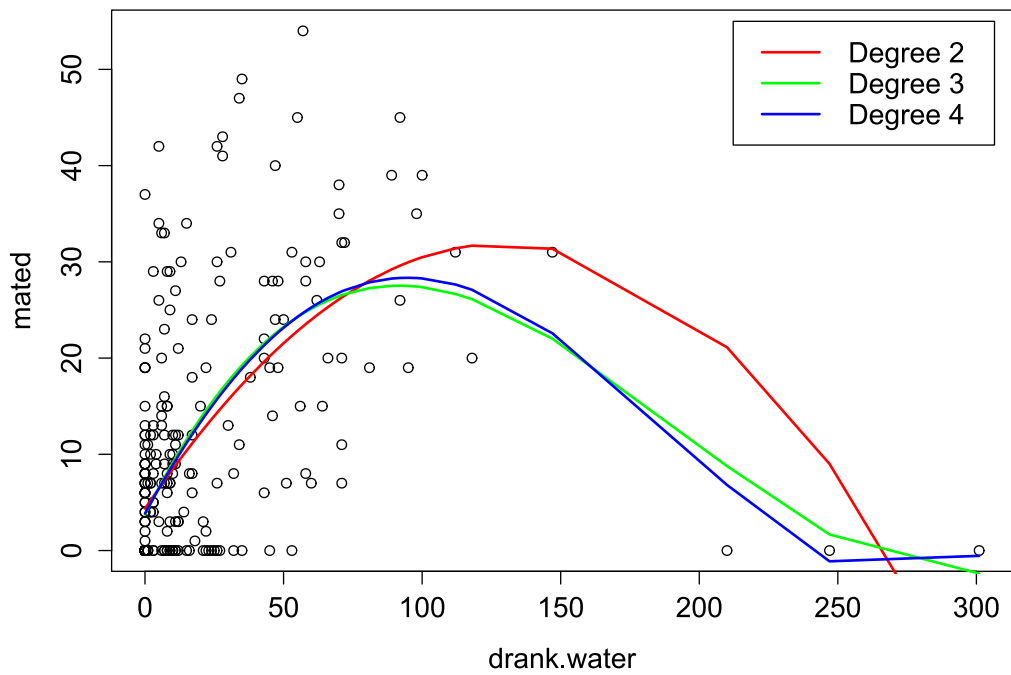


Figure 7.6: This graph examines the relationship between drinking water and mating score. Although the graph is quite noisy, it is apparent that drinking either too little or too much water leads to a lower reproductive fitness in this world. Three polynomial regression models fitted by the least squares method produced curves which highlight the non-linear relationship between these score attributes. Each curve in this diagram represents a polynomial of a different degree.

available subminds for their performance against the metrics which correlate strongly with success in the world.

A slight problem is evident with this analysis. Referring again to the ranked correlations for *Tyrrell09* listed in table 7.1, there is a clear overlap between the number of times the simulated animal successfully mated, and the number of times it attempted both the *court-mate* and *attempt-mate* actions. Similarly, seven of the next eight strongest correlations represent actions which move the animal in one of eight possible directions by one cell on the two-dimensional world grid.

If we were to select five subminds based directly on the most influential five metrics as listed in the table, for example, then only two behaviours would really be addressed: mating and moving.

It is clear that within these score attributes which count events in the world and actions taken, there exists a degree of redundancy. Perhaps by only accounting for actions and events which appear to be relatively independent of each other, then the highest ranked correlations may represent a wider spectrum of behaviour needed by minds to solve the problems presented in this world.

Next, we will investigate whether principal component analysis can help to separate the variables and minimise this redundancy.

## 7.8 Principal Component Analysis (PCA)

A tool often used in exploratory data analysis is principal component analysis (PCA). In PCA, a series of measurements over several dimensions is transformed into a set of orthogonal and linearly uncorrelated components which attempt to represent most of the variance within the dataset. This section explores the use of PCA to obtain a reduced set of variables by which to evaluate and rank subminds for inclusion in a hybrid mind such that holistic behaviour in *Tyrrell09* world might be produced by those subminds.

PCA is frequently used in certain types of “lossy” data compression, where it is acceptable to lose a small amount of detail in exchange for a large saving in the space required to represent the data. This technique of dimensionality reduction is employed in, for example, the JPEG2000 image compression format [Du and

Fowler, 2007]. In this context, the technique is often referred to as a Hotelling or Karhunen–Loève transformation.

The technique is also commonly employed in exploratory analysis of a dataset, where the relationships between variables, and the relative significance of those relationships is unknown.

When the analysis is performed, the result is a series of components (or dimensions) sorted such that the first component accounts for the largest part of the variance observed in the dataset, the second component accounts for the second largest portion of variance, and so forth.

Summing the variance represented by each component provides us with a cumulative measure of variance, which allows us to set a stopping criterion – for example, when 95% of the variance has been explained.

### **7.8.1 Principal component analysis of the Tyrrell world score data**

In this step, we will perform a principal component analysis of the collected score data in the Tyrrell world – the same dataset used in chapter 6.

When performing PCA, there are several methods which can be used depending on the type of data being examined. In this case, a correlation matrix is used, rather than a covariance matrix, and the data are mean-centred and scaled to unit variance. This is so that variables measured on different scales and taken from different distributions do not distort the calculations.

Additionally, variables which contain constant values (that is, where every measurement of the variable produces the same result; typically zero in this instance) are removed prior to performing the scaling and analysis operations. This reduces the dimensionality of our dataset from 49 to 43 variables.

Performing the analysis on the score data produces a set of components whose relative importances are summarised in table 7.3 on page 135.

The majority of the variance is accounted for by a small number of components – the first 3 components represent 54.13% of the variance, the first 16 components represent 90.67%, and the first 27 components represent 99.15%, respectively. This pattern is highlighted by the scree plot in figure 7.7.

When using PCA for dimensionality reduction, some criteria must be used to

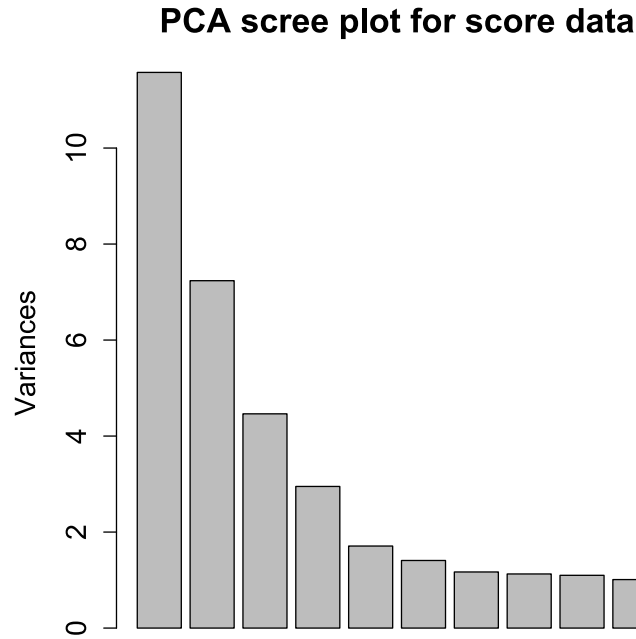


Figure 7.7: A scree plot of the first ten principal components derived from the score data for *Tyrrell09*, showing that most of the variance is explained by the first few components.

decide how many principal components should be taken. One common rule of thumb is to take all the components whose eigenvalues are greater than 1 [Kaiser, 1960], when the PCA is performed with a correlation matrix, as in our case.

The standard deviations listed in table 7.3 represent the square roots of the eigenvalues, and show that the first ten components satisfy this condition.

### 7.8.2 Selecting a subset of variables

There is some redundant information contained within the variables of our dataset. For example, the “court mate” and “attempt mate” actions are positively correlated to each other and to the actual “mated” score value.

It is conceivable that some worlds might produce score and state vectors which contain greater redundancy than this – for example, by completely duplicating certain score elements.

This raises the question: Now that we have established the principal components

	Standard deviation	Proportion of Variance	Cumulative Proportion
PC1	3.402321737	0.2692	0.2692
PC2	2.690120056	0.1683	0.4375
PC3	2.112627929	0.1038	0.5413
PC4	1.717686254	0.06862	0.60991
PC5	1.307308458	0.03975	0.64966
PC6	1.18663635	0.03275	0.6824
PC7	1.081106091	0.02718	0.70958
PC8	1.061777678	0.02622	0.7358
PC9	1.048370096	0.02556	0.76136
PC10	1.005172408	0.0235	0.78486
PC11	0.958171286	0.02135	0.80621
PC12	0.954520536	0.02119	0.8274
PC13	0.938438055	0.02048	0.84788
PC14	0.934439945	0.02031	0.86819
PC15	0.916750642	0.01954	0.88773
PC16	0.902045181	0.01892	0.90665
PC17	0.852216271	0.01689	0.92354
PC18	0.826033804	0.01587	0.93941
PC19	0.745020755	0.01291	0.95232
PC20	0.657591494	0.01006	0.96238
PC21	0.545807682	0.00693	0.9693
PC22	0.519198629	0.00627	0.97557
PC23	0.489900048	0.00558	0.98116
PC24	0.407530071	0.00386	0.98502
PC25	0.323009025	0.00243	0.98744
PC26	0.299246551	0.00208	0.98953
PC27	0.292100946	0.00198	0.99151
PC28	0.272918462	0.00173	0.99324
PC29	0.242446387	0.00137	0.99461
PC30	0.206409072	0.00099	0.9956
PC31	0.178276841	0.00074	0.99634
PC32	0.169777801	0.00067	0.99701
PC33	0.155280923	0.00056	0.99757
PC34	0.144839377	0.00049	0.99806
PC35	0.138836955	0.00045	0.99851
PC36	0.125266759	0.00036	0.99887
PC37	0.112928107	3.00E-04	0.99917
PC38	0.103047906	0.00025	0.99942
PC39	0.093464392	2.00E-04	0.99962
PC40	0.083541865	0.00016	0.99978
PC41	0.074575328	0.00013	0.99991
PC42	0.062035551	9.00E-05	1
PC43	1.51E-15	0	1

Table 7.3: A summary of the relative importance of each of the 43 principal components representing the score dataset.



in the data, how do we map the strongest of these components back to the *original variables* which characterise our data?

One method is to look at the *loadings* or *rotations* for each component. In principal component analysis, each component represents a linear transformation of the original variables into a new co-ordinate system, expressed as a weighted linear sum of the form:

$$PC_n = r_{n,1}V_1 + r_{n,2}V_2 + \dots + r_{n,N}V_N$$

Here,  $r_{n,k}$  represents the loading coefficient applied to the original variable  $V_k$  as part of the contribution to the principal component  $PC_n$ .

Having obtained the components and loadings from PCA, we then select for each component the variable with the highest absolute value which has not already been chosen. This strategy is called method *B4* by Jolliffe [2002]).

The desired outcome was a small number of variables which would be relatively independent of one another, which could then be sorted by the strength of their correlation with the overall fitness measure (the “mated” score value) and then used to select appropriate subminds.

The loadings observed by running PCA against the dataset, however, do not seem to be helpful in this regard, however. For example, the loadings for the first three principal components are shown in table 7.4 on the following page.

By applying the B4 method with these values, the variables selected to represent the first three components were action 26 (move northwest) for PC1, action 15 (look south) for PC2 and action 3 (eat cereal food) for PC3. Following a similar process for the next seven principal components gives us more variables, shown in table 7.5.

While the selected variables may account for a large proportion of the variance in the dataset, they do not seem to be associated with successful behaviour in the world. For example, one might expect to see either the “mated” variable, or the actions associated with mating (the *court* and *mate* actions) in this subset of variables, on the basis that our selected measure of success in the world is the number of times that the simulated animal mated.

However, what is considered important in terms of successfully solving the prob-

	PC1	PC2	PC3
<i>e.mated</i>	-0.18032999	0.048726304	-0.108211554
<i>e.lifespan</i>	-0.184201693	0.075923672	-0.043815159
<i>e.ate fruit</i>	-0.052100384	0.088486302	-0.309170297
<i>e.ate toxic fruit</i>	-0.033501923	0.048782699	-0.171867524
<i>e.ate cereal food</i>	-0.047863705	0.079506941	-0.258443161
<i>e.ate toxic cereal</i>	-0.021909162	0.044480412	-0.138083732
<i>e.drunk water</i>	-0.170604864	0.023023315	-0.172802301
<i>e.drunk toxic</i>	-0.125221877	0.007234402	-0.124085452
<i>a.sleep</i>	0.013472972	0.113430966	0.056609683
<i>a.rest</i>	-0.003176053	0.029368695	0.035962487
<i>a.freeze</i>	0.037129048	-0.097405162	-0.047075606
<i>a.eat cereal</i>	-0.049590127	0.077649074	<b>-0.338860327</b>
<i>a.eat fruit</i>	-0.052281697	0.090549266	-0.325985041
<i>a.drink</i>	-0.155250917	0.005056609	-0.172288866
<i>a.clean</i>	-0.149270878	-0.026322986	0.00704298
<i>a.court</i>	-0.055526292	0.015677608	-0.02736446
<i>a.mate</i>	-0.057632433	0.011869894	-0.035578131
<i>a.pounce</i>	-0.015324684	0.020149622	-0.139381539
<i>a.look around</i>	-0.019296913	0.053668579	-0.077083523
<i>a.look N</i>	0.105088958	-0.322093956	-0.132295199
<i>a.look NE</i>	0.105057003	-0.318991056	-0.127754423
<i>a.look E</i>	0.10580077	-0.322742046	-0.134789799
<i>a.look SE</i>	0.105611028	-0.321936813	-0.134403935
<i>a.look S</i>	0.105514741	<b>-0.3235404</b>	-0.136255115
<i>a.look SW</i>	0.106848317	-0.323028951	-0.133151461
<i>a.look W</i>	0.10569415	-0.321556265	-0.13461841
<i>a.look NW</i>	-0.092732616	-0.028673339	0.072296007
<i>a.move N</i>	-0.244935261	-0.022468664	-0.133176412
<i>a.move NE</i>	-0.250291476	-0.035673587	-0.118912362
<i>a.move E</i>	-0.241667587	-0.023741451	-0.136829854
<i>a.move SE</i>	-0.237536573	-0.029705524	-0.071612001
<i>a.move S</i>	-0.084669049	0.016394893	-0.042898005
<i>a.move SW</i>	-0.254312776	-0.035544793	-0.119912409
<i>a.move W</i>	-0.248995383	-0.030174852	-0.117975375
<i>a.move NW</i>	<b>-0.260388627</b>	-0.059657058	-0.044148528
<i>a.run N</i>	-0.208269629	-0.143788123	0.144083005
<i>a.run NE</i>	-0.209010646	-0.169637221	0.181985854
<i>a.run E</i>	-0.208349782	-0.169060853	0.187976977
<i>a.run SE</i>	-0.209683492	-0.167315797	0.187857884
<i>a.run S</i>	-0.210666418	-0.168512138	0.18297351
<i>a.run SW</i>	-0.178037462	-0.142586077	0.177066125
<i>a.run W</i>	-0.210181523	-0.164257761	0.181745995
<i>a.run NW</i>	0.038018808	-0.115842284	-0.072963862

Table 7.4: The loadings determined by principal components analysis of the Tyrrell world score dataset, for the first three components. Variables prefixed by “e.” indicate events recorded in the score vector, while variables prefixed with “a.” indicate actions taken. The absolute maximum loading for each component is highlighted in bold, giving the *move northwest* action for PC1, the *look south* action for PC2 and the *eat cereal food* action for PC3.

Component	Variable
PC4	<i>a.eat fruit</i>
PC5	<i>a.sleep</i>
PC6	<i>a.look NW</i>
PC7	<i>e.ate toxic cereal</i>
PC8	<i>a.look around</i>
PC9	<i>a.run NW</i>
PC10	<i>a.freeze</i>

Table 7.5: The variables corresponding to the remaining principal components PC4-PC10, selected without replacement by greatest absolute loading value.

lem presented by the world may not have a large impact on the statistical variance present in the rest of the dataset.

A better approach for selecting important variables to be used in ranking subminds while avoiding redundancy might be to examine the variable correlation matrix and discard variables which are highly correlated with a variable which has already been selected.

## 7.9 Using the correlation matrix to minimise redundancy in variable selection

Another method to minimise the selection of redundant variables in the score data might be to ignore variables which exhibit a greater statistical dependence with already-selected variables than with the desired fitness measure.

That is to say, before a variable  $v$  is added to the “independent” set  $s$  (initially empty), to test that the correlation strength  $R^2$  between  $v$  and every variable in  $s$  is less than the correlation between  $v$  and the fitness measure (the value of the *mated* score variable in the *Tyrrell09* world).

An implementation of this method in the R language is shown in algorithm 7.1.

### 7.9.1 Reduced variable set for the Tyrrell09 world

Running this algorithm on the collected scores in our *Tyrrell09* dataset produces a list with the following score attributes:

- *e.mated* – The number of times the simulated animal successfully mated during a run.

---

**Algorithm 7.1** A variable selection algorithm which attempts to avoid selecting redundant variables by testing the correlation strength between a new variable and the set of already-chosen variables against the correlation strength between the new variable and the primary fitness measure. A small tolerance threshold *corr\_t* is added to avoid removing too many variables.

---

```
build_independent_varlist <- function(corrs) {
  # first sort by decreasing correlation with primary fitness
  measure
  corrs_sorted <- corrs[order(corrs[,"mated"], decreasing=TRUE),]
  s = c()
  corr_t = 0.025

  for(nextVariable in rownames(corrs_sorted)) {
    corrOther = FALSE
    for(existing in s) {
      if(corrs[existing, nextVariable] > corrs["mated",
        nextVariable]+corr_t) {
        corrOther = TRUE
        break
      }
    }

    if(length(s) == 0 || !corrOther) {
      s = c(s, nextVariable)
    }
  }
  return(s)
}
```

---

- *a.court* – The number of times the “court mate” action was taken.
- *a.move N* – The number of times the “move north” action was taken.
- *a.clean* – The number of times the “clean” action was taken.
- *e.lifespan* – The number of timesteps survived in the world.
- *e.ate toxic fruit* – The number of times the animal ate toxic fruit.

It is perhaps surprising that *e.ate toxic fruit* was selected, and not *e.ate fruit*. Looking at the correlation matrix shows that the association between the *e.ate fruit* and *a.move N* attributes was stronger ( $R = 0.453$ ) than the association between *e.ate fruit* and *e.mated* ( $R = 0.364$ ). The correlation between *e.ate toxic fruit* and *e.mated* was lower ( $R = 0.243$ ) but still positive.

### 7.9.2 Reduced variable set for the ChessWorldG world

The same algorithm was run on the set of scores observed in *ChessWorldG*, with a slight modification to use the absolute correlation strength (regardless of direction). This produced the following list of variables:

- *Survival.moves* – The total number of moves the mind made during the five games played.
- *Lost.Queens* – The *negative* of the number of queens captured by the opponent. The negative value is used so that the world scoreboard can rank minds in the correct order – that is to say that, all other things being equal, a mind with a score of zero for the *Lost.Queens* variable will be ranked higher than a mind with a score of -10. However, this does not mean that a mind with a score of zero in this metric is better – it might be zero because the mind made an invalid move early in each game and forfeited immediately, and in fact this is the case for two of the top five minds ranked by this metric.
- *Invalid.moves* – The *negative* of the count of illegal moves made by the mind.
- *Captured.Queen* – The total number of queens captured by the mind.
- *Captured.Rook* – The total number of rooks captured by the mind.

As demonstrated in the *Lost.Queens* example above, ranking subminds directly on the maximisation of these variables can sometimes select subminds which are poorly designed or even broken. To help compensate for this effect, a weighted ranking is performed, taking into account a proportion of each submind’s overall success. In this case, the effective score for a mind was given by the formula  $v + \frac{f}{50}$ , where  $v$  is the value of the ranking variable and  $f$  is the value of the *Survival.moves* variable for that mind..

The minds are selected without replacement, yielding the following minds in order of the variables described above: *Mind\_M*, *AlphaBeta*, *Mindcall2*, *MiniMax*, and *AggEvalBoard*. After some testing however, the *AggEvalBoard* mind was found to fail intermittently. Rather than replace it with the next best mind, the “*Captured.Rook*” variable was dropped and four subminds were used.

## 7.10 Building the hybrid mind controller for *Tyrrell09*

As in section §6.6, a controller based on a decision list was used to select which of the subminds to obey at each timestep.

The conditions were tested and ranked such that the first condition tested corresponds to the score element (in the reduced set produced in section §7.9) which correlated most strongly with the “mated” score, and so forth in descending order of correlation strength.

For example, in our hybrid, the conditions are evaluated and the appropriate submind called as shown in pseudocode in algorithm 7.2.

In some cases, there is no obvious way to determine whether a particular behaviour is appropriate because there is no condition to test. An example of this is the “move north” action and related movement behaviours. It may be possible to devise an indirect condition which triggers the behaviour – for example, if the animal has not moved for several timesteps in a row – but this would be somewhat subjective.

Another option is simply to move that behaviour to the default case, when no other conditions have been met, as can be seen in the last line of algorithm 7.2. If there were several such behaviours, a random selection might be performed to choose one at any given timestep. Random selection is also used to arbitrate between

---

**Algorithm 7.2** A pseudocode implementation of the hybrid controller for the *Tyrrell09* world, which implements a simple condition list structure to determine which submind to obey at any timestep.

---

```
if state.getMateCourted() > 0 then return mater.getAction(state)
elseif state.getMatePerceptionStimulus[0] > 0 then
    return courter.getAction(state)
elseif sum(state.getMatePerceptionStimulus()) > 0 then
    return mater.getAction(state)
elseif state.getPerceivedAnimalCleanliness() < 0.5 then
    return cleaner.getAction(state)
elseif state.getAnimalHealth() < 0.9 then
    return survivor.getAction(state)
elseif state.getPerceivedFatShortage() > 0.9 or
    state.getPerceivedProteinShortage() > 0.9 or
    state.getPerceivedCarbohydrateShortage() > 0.9 then
    return eater.getAction(state)
// default to best submind if no other conditions match
else return mater.getAction(state)
```

---

multiple behaviours which are triggered by the same condition – as in the third line of algorithm 7.2, between the mating and courting subminds.

It is important to note that although the hybrid mind examines the world state to decide which submind should be obeyed at any given moment, it does not directly issue any actions of its own. This need not be the case, of course – the hybrid mind could select some actions directly and only defer to subminds when certain circumstances are observed in the world state.

## 7.11 Building the hybrid mind controller for *ChessWorldG*

The process of building a hybrid mind controller for this world was less straightforward than for Tyrrell’s world, where the task is to control a single actor with several well-defined responsibilities, and there is no specific opponent which must be faced. In chess, the player begins with 16 pieces which can each capture or be captured by the opponent’s pieces. Each piece has different strengths and weaknesses, and a correct policy is not obvious.

### 7.11.1 Using reinforcement learning to produce a hybrid controller for *ChessWorldG*

Rather than attempt to hand-design the hybrid chess controller, a reinforcement learning agent was constructed using the *Elsy* connectionist Q-learning framework

Input representation	Piece type
-1	Empty space
-5/6	White king
-4/6	White queen
-3/6	White rook
-2/6	White bishop
-1/6	White knight
0	White pawn
1/6	Black king
2/6	Black queen
3/6	Black rook
4/6	Black bishop
5/6	Black knight
1	Black pawn

Table 7.6: The mapping between neural network input representation and the corresponding piece (or space) on the chessboard.

[Kapusta, 2011], which uses feed-forward neural networks with eligibility traces to learn an approximation to the Q-value function [Sutton and Barto, 1998b]. The function of reinforcement learning is to learn an action-selection policy in response to numeric rewards [Russell and Norvig, 2003], where a positive-valued reward is issued for correct behaviour, a negative reward for poor behaviour and zero otherwise.

### 7.11.2 Choosing the reward function

The impact of the reward function is critical; a poorly-designed one can result in a learner which never converges on a successful policy. In particular, rewards should exceed a value of 1.0, as this could cause a failure of the learning element to converge on optimal policy [Kapusta, 2011]. In this case, a reward of +1 is assigned for a win, +0.5 for a draw, -1 for a loss, and zero in all other cases.

### 7.11.3 State input mapping

The convergence rate of a neural network for any particular function can be sensitive to the description of the inputs, so some form of normalisation is usually performed to help this process along [Bishop, 1995, p. 298]. For the *ChessWorldG* controller, the set of inputs is a set of 64 numbers (one for each square of the board) in the range  $[-1, +1]$ . The assignment of these numbers to chess pieces is defined in table 7.6.



#### 7.11.4 Action definition

Instead of learning to select moves directly, the reinforcement learning controller is tasked with learning which of the four subminds it should obey at any particular moment. Each of the four actions specifies one of the subminds selected according to their performance in the variables identified in section 7.9.2.

#### 7.11.5 Network architecture

The 64 input nodes connect to two hidden layers with 32 and 24 neurons respectively. These feed into an output layer with four neurons each representing the expected value of taking one of the available actions in the current state. Several variations were tested, with more or fewer hidden layers and different amounts of neurons in each, before arriving at this configuration.

Once the Q-values are estimated, the selection of actions is done using softmax activation according to a Boltzmann distribution [Sutton and Barto, 1998b], which selects actions probabilistically so that the “greedy” (best Q-value) action is most likely to be chosen, and the apparent worst action is least likely to be chosen. This is performed so that an effective balance can be struck between exploration and exploitation while learning. At timestep  $t$ , action  $a$  will be selected with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^n e^{Q_t(b)/\tau}}$$

where  $\tau$  is a *temperature* parameter, such that a high temperature produces more uniform selection of actions and lower temperatures approach greedy selection. Sometimes, a high temperature is used to begin with to encourage more exploration of the action space, and gradually reduced towards greedy selection over time [Guo et al., 2004]. Here, the initial temperature is set to 1.0 and decayed by a factor of 0.9 until it reaches a minimum value of 0.002.

Other important parameters for the learner are  $\gamma = 0.95$ , the discount factor which represents the importance of future states in Q-learning,  $\alpha = 0.1$ , which represents the learning rate of the neural network which learns to approximate the Q-value function, and  $\lambda = 0.5$ , which signifies the eligibility trace forgetting rate.

## 7.12 Experimental results

An analysis of the experimental outcomes in both the TyrrellWorld09 and ChessWorldG worlds follows, showing that the method proposed in this chapter produced a successful hybrid mind in the first world but not in the latter.

### 7.12.1 Tyrrell09 results

The hybrid mind created through this process was run in the same world with the same parameters as the subminds and the hybrid mind developed in chapter 6. The best, median and mean scores achieved by each mind after  $N = 1000$  runs are displayed in table 7.7, along with the standard deviation in each score. The score data are summarised with boxplots of the *mated* and *lifespan* score samples for each mind in figure 7.8.

	Existing best mind	Manual state analysis	Score (auto)
<b>Subminds</b>	2	4	5
<b>max(mates)</b>	74	78	84
<b>median(mates)</b>	43	49.5	47
<b>mean(mates)</b>	41.533	47.95	46.478
<b><math>\sigma</math>(mates)</b>	12.53204	12.63523	12.74472
<b>max(life)</b>	4848	4797	4840
<b>median(life)</b>	4208	4254	4205
<b>mean(life)</b>	3849	3942	3934
<b><math>\sigma</math>(life)</b>	1019.069	934.176	868.9804

Table 7.7: The hybrid mind in column 4 was constructed by automatically selecting and ranking important variables relevant to the world state in the *Tyrrell09* world. For comparison, the mind produced in chapter 6 by manually selecting important state attributes is included in column 3. Column 2 shows the performance of the existing best mind submitted to the server by a student.

The performance of the hybrid mind created in this chapter is shown on column 4 of the table, having outperformed all of the subminds submitted by users, without ever needing to know how to act directly in the world.

These results are comparable with, but slightly inferior to those of the hybrid mind built in chapter 6, and less human judgement is required. Note that the hybrid mind in column 3 was created by mining the manually-selected state data, which contains richer information than is provided by the world's score vector.

To determine whether these results are significant, Welch's t-test was again per-

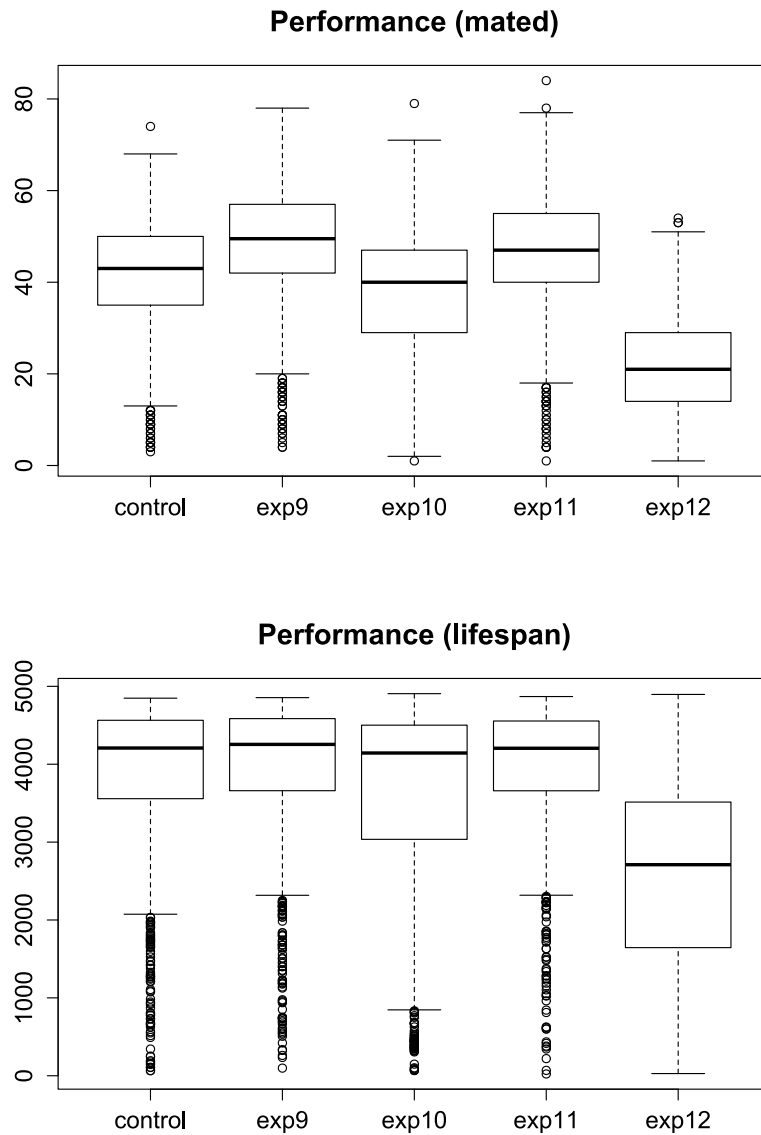


Figure 7.8: Boxplots of the performance in *Tyrrell09* of several minds for the *mated* and *lifespan* score variables. The label *control* indicates the best mind used as a benchmark against which the new hybrid minds are tested. The label *exp9* refers to the hybrid mind created in chapter 6, and *exp11* refers to the hybrid created in this chapter. Both of these minds exhibit better performance in the *mated* score variable than the control mind. The other two minds, *exp10* and *exp12*, were created using the same submind selection method as *exp11*, but with different controller designs – neither of which is successful. The *exp12* mind in particular demonstrates much worse performance, with a lower median in both *mated* and *lifespan* scores, and a much higher variance in lifespan compared to the other minds.

$(N = 1000)$	Manual state-mined hybrid	Semi-auto score-mined hybrid
Mean mates	47.95	46.478
Existing best	41.533	41.533
p-value	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$
Mean lifespan	3942.08	3934.424
Existing best	3848.696	3848.696
p-value	0.03279	0.04308

Table 7.8: Results of t-tests for the hybrid minds produced for the *Tyrrell09* world in this and the preceding chapter. In both cases, the means for the “mated” score variable are significantly greater (better) than the means recorded for the best existing mind. For completeness, the “lifespan” variable is also included in the analysis and is also significantly greater, albeit with a lesser degree of confidence.

$(N=10)$	Existing best	Hybrid with neural $Q(\lambda)$ controller
Median survival moves	211	200.5
Mean survival moves	205.8	198.9

Table 7.9: Results of evaluating the hybrid mind created for the *ChessWorldG* world.

formed, comparing the means of the existing best mind and the hybrid built using automatic variable ranking. The results of these tests are as presented in table 7.8. Additionally, the level of deviation from statistical normality of each set of samples was checked with a Q-Q plot, shown in figure 7.9.

The outcome of the t-test allows us to reject the null hypothesis – that the difference in means between the hybrid mind and the control occurred by chance – and therefore conclude that the hybrid mind is superior to the best existing third-party mind.

### 7.12.2 ChessWorldG results

The experimental outcomes recorded for the ChessWorldG world are shown in table 7.9.

Possible explanations for a weaker result in ChessWorldG (compared to Tyrrell09) include:

- The process of breaking chess into a large number of distinct behaviours was not as obvious as the various behaviours and subproblems modelled in Tyrrell’s world.
- It was not as straightforward to detect “interesting” events in the state. Tyrrell’s world provides a rich set of sensory inputs as arrays of real-valued numbers which can be tracked.

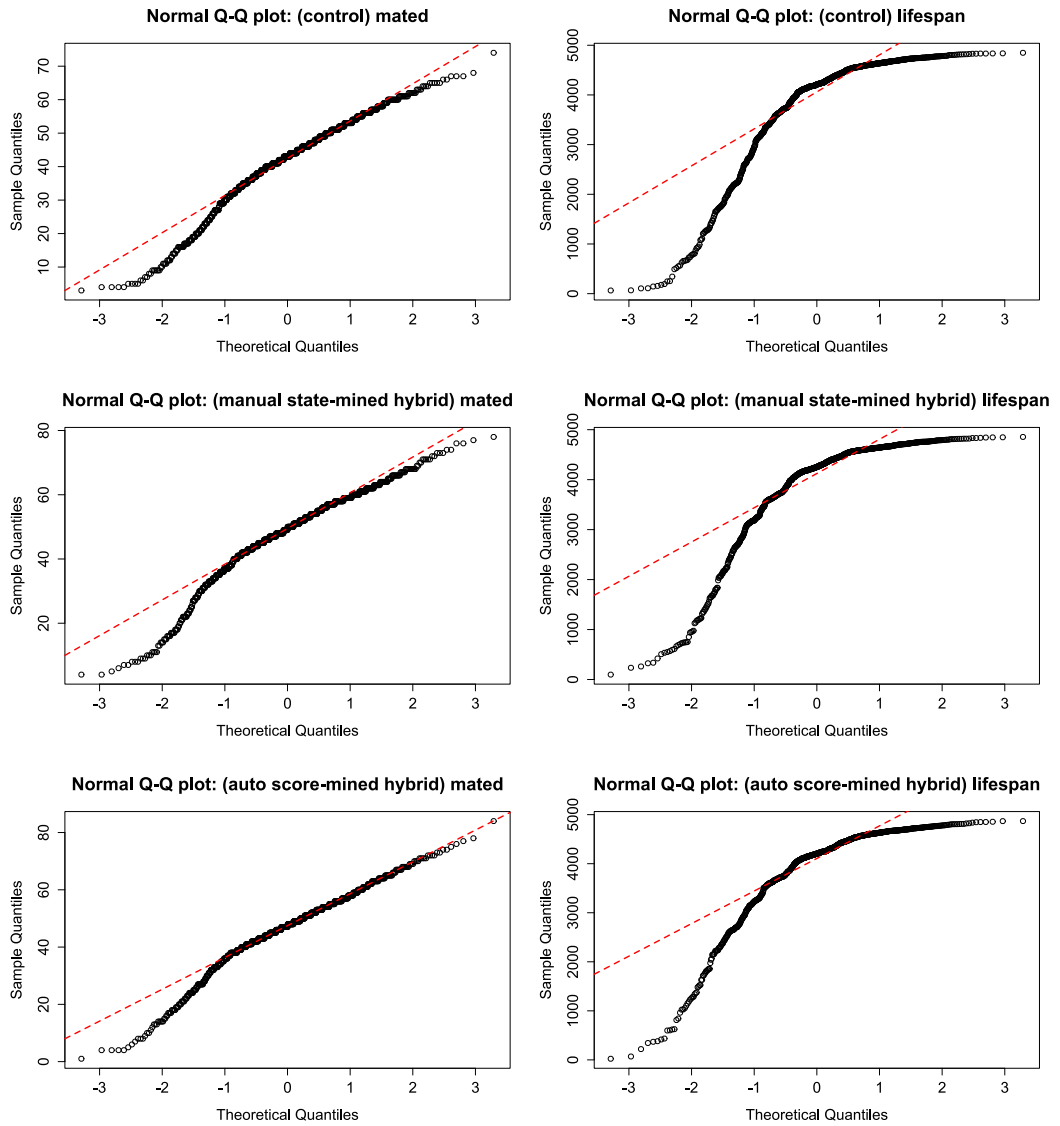


Figure 7.9: Quantile-quantile (Q-Q) normality plots [Wilk and Gnanadesikan, 1968, p. 5] of the score samples for the three tested minds in the *Tyrrell09* world. For the most part, the distributions of the *mated* variable samples are close to a normal distribution, with a small degree of right-skew. The distributions of the *lifespan* samples exhibit much stronger right-skew, mostly due to the large concentration of samples achieving more than  $\approx 4,000$  timesteps (the maximum theoretical lifespan is 5,000 but none reach this).

- Subminds did not specialise in particular behaviours – such as extreme defensiveness, or aggression, or attempting to achieve pawn promotion – instead, they could be separated only into good and bad chess players. The reinforcement learning agent’s policy converged to rely mostly on two “strong” subminds from the available four.
- The reinforcement learning algorithm may have become “stuck” in a local optimum. The application of domain-specific knowledge regarding chess may help to redesign the controller to better avoid these optima and produce a more successful policy.
- The input encoding (described in section 7.11.3) tried to cram too much information into too few input nodes in the neural network, resulting in interference which may have prevented the network from specialising on observed input. A better approach might be to group the inputs into a *1-of-C* encoding [Bishop, 1995], with each of the 64 chessboard squares taking an ordered series of 13 binary category variables corresponding to the possible piece types (or nothing) that might inhabit that square. Each variable is given the value zero, except for the one representing the piece located in that square, which is given a value of one. This type of input mapping would result in a neural network with 832 inputs and would likely need a much larger number of hidden units than the one used for these experiments.

As a result, the creation of a control program for the hybrid mind was non-trivial. Rather than observing that a “trigger” sensory input has dropped below (or exceeded) a simple threshold value as in Tyrrell’s world, a more complicated analysis of the state must be performed in order to select an appropriate submind for the current state – a process which requires more domain knowledge.

### 7.13 Conclusion

This chapter built on the ideas presented in chapter 6, extending the approach by using principal component analysis and correlation methods in an attempt to determine important and relatively orthogonal metrics, and to help select expert subminds

for inclusion in a hybrid mind. The subset of variables produced using the PCA-based method, although helpful for explaining variance in the dataset, was not useful for the purposes of selecting subminds for use in a new hybrid mind. However, the method based on correlation analysis and ranking yielded a series of plausible metrics with which subminds could be ranked and selected.

The method has proved to be useful in the *Tyrrell09* world – a complex problem environment with a large number of variables. The method was also tested for a different world, *ChessWorldG*, where it did not perform well. This suggests that the proposed method of constructing hybrids is task-sensitive, and that it is not suited to certain classes of problems; perhaps, where it is difficult to decompose optimal policy for the world into several distinct behaviours.

Taken together, these results help elucidate the third research question asked in section §1.4.

### 7.13.1 Limitations

The results presented for *Tyrrell09* are competitive with the hybrid minds created through human insight. However, the design of the hybrid mind controller was neither obvious nor trivial. Small changes in the controller which seemed like a logical choice that would improve overall performance often resulted in poor results, and it was necessary to go through several revisions and test them separately before finding one which appropriately harnessed the individual strengths of each submind. It may be interesting to integrate the submind selection approach described here with one of the systematic design methods for constructing modular control systems described in chapter 3 such as POSH [Bryson, 2002].

A potential weakness of this method is that a variable which correlates strongly with success may be dependent on a context which is not understood by the algorithm. This is highlighted in section 7.9.2, where the *Lost.Queens* variable was discovered to have a strong relationship with the fitness value for *ChessWorldG* – the value of the *Survival.moves* variable – but the most highly-ranked minds according to this metric were quite ineffective. One approach to address this problem might be to factor the primary fitness measure into every other score measure, so that ineffective minds will be weighted down and effective minds will be weighted up.

This would affect the correlations of variables and accordingly should be performed after the correlation analysis, when ranking the subminds, as is done here.

In a more general sense however, the problem of context-sensitivity of metrics remains, and must be passed onto the hybrid controller. In effect, the method presented here can help to discover *what* behaviours are important, while the controller must specify *when* they are important, be it through a separate learning element or the author's domain knowledge.



## Chapter 8

# Future Work

This chapter discusses some possibilities for future work, extending from the research described in earlier chapters. Section 8.1 looks at the possibility of extending the W2M 2.0 architecture with several new features. Then, a series of lower-level engineering changes is considered which could greatly improve performance and help scalability. Finally, we look at work that either builds on the tools for hybrid building introduced in chapters 6 and 7, or attempts to bridge the gap with some of the existing cognitive architectures described in chapter 3.

### 8.1 Extending the W2M 2.0 architecture

#### 8.1.1 Real-time environments

The architecture currently operates in a synchronous and non-realtime way. One interesting research possibility would be to extend the W2M 2.0 architecture to support real-time environments and thereby make possible tasks such as interacting with a human – for example, a chatbot – and controlling a physical robot in the real world.

How robot control with the W2M could be accomplished depends on where the robot is situated:

1. A robot or other device that interacts with the real world is made available for others to control by uploading minds either to a connected server machine, or directly to the robot if it is running an onboard W2M server. Note that the overhead of calling subminds – more specifically, remote subminds hosted

elsewhere on the Internet – could pose issues regarding latency. If such a mind is relied upon to issue low-level motor commands, then there’s a chance that a walking robot might fall over mid-stride thanks to gravity’s immediacy, or a driving robot may hit an obstacle before the mind even receives a proximity warning from the robot’s sensors. Also, allowing minds written by unknown third parties to control your robot could lead to incorrect behaviour resulting in damage to equipment or even injury in the real world, either by accident or by intention. For obvious reasons, it would only be possible to have one run executing in that “world” at a time. However, there have been controllable robots online for some time [Taylor and Dalton, 1997; Stein, 2003], so it seems unlikely that these are insurmountable problems.

2. Alternately, everybody who wishes to use a physical robot “world” could have their own identical robot – or at least one with a compatible feature set and physical characteristics which are close enough to allow reasonable portability of minds between compatible robots. Authors could design and test their minds on their own robot before uploading them to a central repository, or download other authors’ minds into their local robot. This presents issues for hybrid minds – must one download the entire tree of subminds? – and does not solve the problems of malicious or faulty behaviour in the third-party minds, but it allows multiple “runs” at once since each lab has their own robot.

Having one’s own robot does not solve the latency problem when the controller is a hybrid mind which uses remote subminds. Some approaches to solve or mitigate this problem might include:

- Having the hybrid mind take care of low-level control (akin to a biological lifeform’s nervous system), handling basic motor control and reflexes, while subminds provide higher-level guidance and goal-directed behaviour.
- Having the hybrid mind provide a “default” behaviour as above, which the subminds can override or interrupt with their own commands if they can be provided to the hybrid controller before a hard deadline is passed (for example, 100 milliseconds after a state is observed). This approach loosely resembles that taken by the subsumption architecture [Brooks, 1991].

- Having subminds declare which parts of the sensory input they are interested in – this might reduce the amount of information to be encoded, sent across the network and parsed.

Each of these solutions has its own drawbacks, so it may be the case that hybrid authors will only choose to work with subminds installed on the same server (or robot).

### **8.1.2 Multi-mind environments**

The current design allows only a single agent to interact directly with an instance of a world. It might be interesting to allow multiple minds to interact with the same world instance, since it would enable both collaborative and competitive behaviours to be studied.

## **8.2 Technical improvements to the W2M 2.0 platform**

The W2M 2.0 platform is an improvement in many ways over the previous work discussed in chapter 4. However, some limitations are identified here which could be investigated and addressed in future.

### **8.2.1 Runlogger: single-threaded multi-process vs multi-threaded single-process**

In the current design, a *Runlogger* process is started every time a user initiates a run via the web interface. This has important scalability implications: the number of concurrent users starting runs is equal to the number of operating system processes spawned.

### **8.2.2 Limitations of the single-threaded, multi-process design**

There is a cost in terms of run-time (due to computation and disk I/O latency), and of memory usage and latency, associated with spawning a process and initialising a new Java virtual machine (JVM) instance, and allocating a new Java heap memory area.

Furthermore, load testing carried out on the Linux machine used to host the World-Wide Mind server has demonstrated that the operating system scheduler can enter a state where too many concurrent processes either slow the system to a crawl or crash it entirely.

This phenomenon is known as *swap death* or *thrashing* [Denning, 1968], where the memory paging system runs so low on available physical RAM that it must constantly swap sections of allocated memory to disk, so as to make room for another piece which needs to be loaded back into memory from disk, and the cycle is repeated when the evicted block is then sought by a program.

### 8.2.3 Experiments with a multi-threaded Runlogger

An alternative implementation of the Runlogger program was created and tested by Monks [Monks, 2010] which runs as a single instance daemon, spawning a new thread to handle each run, rather than creating a completely new process every time, which would cause the creation and destruction of a new Java virtual machine every time as described above.

This modified Runlogger demonstrated far better scalability as the number of simulated users increased, and did not render the system inoperative as the process-based model did.

It would seem appropriate to use this thread-based model to replace the current Runlogger implementation, but it would require some major changes to the user interface, which starts runs by passing the appropriate arguments (including the URLs of the selected mind and world services) to a new instance of Runlogger. If instead only a single instance of Runlogger was running as a background process, then a different method of passing arguments from the web interface would be required; perhaps by adding requests to an interprocess message queue, with the Runlogger daemon listening on the queue and starting threads to service requests as they arrive.

Several free, open-source implementations of message queues exist, such as ZeroMQ and ActiveMQ, which are designed to provide high performance under heavy loads, and could perhaps suit this purpose.

## 8.2.4 Further optimisations: Bypassing the Runlogger

### Avoidable indirection between Runlogger and W2MServer

The W2MServer and Runlogger programs exist as separate processes, and are therefore each associated with separate JVMs and cannot share memory directly, even though they are running on the same host machine.

When an instance of Runlogger executes a run on a local instance of the W2MServer daemon, it communicates by sending and receiving messages over the network, in the same way a mind service would communicate with a W2MServer daemon running on a remote machine.

Although network communication through a local TCP network socket is fast, it is much slower than direct method invocation.

### Merging a multi-threaded Runlogger and W2MServer

If Runlogger could be redesigned to run as a single process with multiple threads as described in section 8.2.1, then it should be possible to remove Runlogger entirely and merge its functionality completely into the W2MServer daemon.

The daemon would then have multiple listener threads:

- one to listen on TCP for incoming connections from remote services, and
- one to wait on a local message queue (or other efficient and fast interprocess communication (IPC) mechanism) for new run requests coming from the web interface.

This would reduce I/O and memory load, and would in turn improve message throughput and scalability, allowing more runs to be performed simultaneously, and at higher speeds.

## 8.3 Building hybrid minds

### 8.3.1 Variable selection for hybrid building

An alternative to the algorithm for minimising the selection of redundant variables proposed in section §7.9 may be to use unsupervised clustering methods such as

Voronoi iteration [Lloyd, 1982] to group together similar variables into distinct categories.

It is noted in section 7.6.2 on page 123 that non-linear relationships may not be treated properly by this type of correlation analysis, which tends to underestimate the association between variables with a non-monotonic curvilinear relationship [Lieberson, 1964]. Future work might examine the use of non-linear regression and prediction methods, such as locally-weighted smoothing [Cleveland and Devlin, 1988], to explore these relationships more deeply.

Finally, it may be fruitful to try local search (for example, genetic algorithms) or a reinforcement learning approach to discover and optimise a mind selection policy.

### **8.3.2 Overlap with existing cognitive architectures**

Some aspects of existing cognitive architectures and modular control systems (discussed in section §3.3) share similarities with the World-Wide Mind.

The Soar architecture provides a complete action-selection implementation [Laird et al., 1987], and therefore would be well suited to exist as a self-contained mind in the W2M. However, due to its reliance on several models of memory it may not be straightforward to integrate Soar as one part of a hybrid mind, where its proposed actions may not be followed and it may not even be allowed to perceive the environment at every timestep.

In Soar, production rules propose, evaluate and select operators that are appropriate for the current situation. Since operators can generate external actions as well as changing internal state, the function of Soar's rules is analogous to the role of the hybrid controller in W2M. This opens up the possibility of implementing a set of subminds either as action-generating operators or as higher-level actions in Soar to produce a new type of hybrid controller which exploits Soar's well-developed procedural knowledge, decision-making and meta-reasoning facilities. It may also be possible to apply either Soar's chunking mechanism or the more recent reinforcement learning extensions [Laird, 2008], or both together, to learn about the world and the available subminds.

The World-Wide Mind also shares some fundamental ideas with Minsky's Society of Mind [1986]. Since the Society of Mind is more of a loose collection of ideas regard-

ing cognitive processes than a fully-specified working system, there is the possibility of taking subsystems from the Society of Mind and implementing them piecemeal into a W2M system, perhaps to enable large-scale hierarchical learning – for example, Minsky’s concept of *K-lines* might be used to re-activate subminds relevant to a task which has been performed before.

## Chapter 9

# Conclusion

The aim of this research has been to improve collaboration and re-use in the building of large hybrid A.I. systems from programs written by many different authors – a subject which has been neglected in current research.

The next section will examine the individual outcomes in the context of the three key research questions asked in chapter 1.

### 9.1 Research questions and outcomes

In section §1.4, three research questions were proposed, which serve the basis for all of the work and experiments performed and discussed in this dissertation. We will now consider how we have succeeded in addressing each of these questions.

#### 9.1.1 How can a hierarchy of minds be supported and built from the programs of many authors?

To develop an answer to this question, an existing architecture – the World-Wide Mind – was modified and extended as described in chapter 5, and a functional server implemented capable of hosting minds and worlds written by many different authors. The new architecture, called the World-Wide Mind 2.0, was validated from a technical and usability perspective.



### **9.1.2 Is it useful to build these hierarchical hybrid minds?**

Chapter 6 proposed a method of assembling large hybrids from a set of possible subminds manually selecting the important attributes by which subminds should be ranked. This method was validated in a complex environment – the Tyrrell09 world – and was able to significantly improve upon the solutions of a large group of authors who worked for several weeks to construct their mind programs. The hybrid mind was able to do this without directly suggesting any actions of its own. This opens up the possibility of a division of labour, where specialists create solutions to tasks, and hybrid builders create hybrids from collections of specialist solutions.

### **9.1.3 Can this process of building hybrid minds be automated in some way, and is it productive to do so?**

The method described in chapter 6 was extended in chapter 7 to automatically select the set of variables by which subminds should be ranked, through the use of correlation analysis. This method of building complex, modular hybrid minds was experimentally validated and shown to be successful in one test environment (Tyrrell09) – achieving significantly better scores in the test world than the existing hand-built subminds – and unsuccessful in another test environment (ChessWorldG). We suspect that chess requires more domain specific knowledge in order to build a better hybrid. This could be integrated with the semi-automated approach in future work.

## **9.2 Closing**

This research has provided three contributions to the field of modular intelligence:

1. the World-Wide Mind 2.0 architecture, and
2. a partially-automated method for building hybrid minds using minds written by many different authors, and
3. a collection of worlds, minds and hybrid minds written by multiple authors.

The research described in this dissertation has provided evidence that the first contribution – the World-Wide Mind2.0 architecture – enables the construction of multi-

author hybrids in a way not easily possible under other existing systems. The benefits of this will be fully seen as it scales to larger numbers of users and authors.

The second contribution presented a novel way of semi-automatically building hybrid A.I. systems which also opens up a future vista of hybrid building which until now was not possible, or at least very difficult. Of course, not all automated approaches will work on all problems, but this method has been successful in one complex problem with multiple goals.

# Glossary

- API Application Programming Interface, a set of symbols, functions and classes which encapsulate the parts of a software library which can be used within other programs
- ASM Action-selection mechanism – an algorithm which receives a set of sensory percepts pertaining to a simulated agent’s current situation and selects an appropriate action to be taken. Directly analogous to a Mind.
- fully-specified class name The fully-specified name of a Java class, formed by appending the class name to the package specification, if any.  
For example, *org.w2mind.net.ServiceProxy* is the fully-specified name for the *ServiceProxy* class in the *org.w2mind.net* package.
- IPC Inter-Process Communication, a set of mechanisms for exchanging data between concurrently running execution threads.
- RL Reinforcement Learning – A branch of artificial intelligence concerned with learning an optimal control policy with only a reinforcement signal (a reward or punishment for recent behaviour) for guidance.
- SOA Service-oriented architecture: A high-level set of principles for building interoperable services
- TCP Transmission Control Protocol, an Internet protocol for reliably delivering an ordered stream of bytes across a network.
- URL Uniform Resource Locator: A string which specifies the address of a resource on the Internet.

# Bibliography

- A. Asuncion and D.J. Newman. UCI Machine Learning Repository. <http://archive.ics.uci.edu/ml/>, 2009.
- Earl R. Babbie. *The Practice of Social Research*. Cengage Learning, Belmont, CA, 2012. ISBN 1133049796.
- Don F Beal. A generalised quiescence search algorithm. *Artificial Intelligence*, 43(1):85–98, 1990.
- Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific American*, 284(5):28–37, 2001.
- Anastasiia Beznosyk, Peter Quax, Karin Coninx, and Wim Lamotte. Influence of network delay and jitter on cooperation in multiplayer games. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and its Applications in Industry*, VRCAI '11, pages 351–354, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1060-4. doi: 10.1145/2087756.2087812. URL <http://doi.acm.org/10.1145/2087756.2087812>.
- Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995. ISBN 0198538642.
- Rodney A. Brooks. Elephants don't play chess. In P. Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 3–15. The MIT Press: Cambridge, MA, USA, 1990. URL <http://www.ai.mit.edu/people/brooks/papers/elephants.ps.Z>.
- Rodney A. Brooks. Intelligence without reason. In *Computers and Thought, IJCAI-91*, pages 569–595. Morgan Kaufmann, 1991.

- Joanna Bryson. The behavior-oriented design of modular agent intelligence. In Ryszard Kowalczyk, Jörg P. Müller, Huaglory Tianfield, and Rainer Unland, editors, *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, volume 2592 of *Lecture Notes in Computer Science*, pages 61–76. Springer, 2002. ISBN 3-540-00742-3. URL [http://dx.doi.org/10.1007/3-540-36559-1\\_7](http://dx.doi.org/10.1007/3-540-36559-1_7).
- Joanna J Bryson, David Martin, Sheila A McIlraith, and Lynn Andrea Stein. Semantic web services as behavior-oriented agents. *IEEE Computer*, 35(11):48–54, 2002.
- Martin Buehler, Karl Iagnemma, and Sanjiv Singh, editors. *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, George Air Force Base, Victorville, California, USA*, volume 56 of *Springer Tracts in Advanced Robotics*, 2009. Springer. ISBN 978-3-642-03990-4. URL <http://dx.doi.org/10.1007/978-3-642-03991-1>.
- J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen. The SIMNET virtual world architecture. In *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*, pages 450–455, sep 1993. doi: 10.1109/VRAIS.1993.380745.
- Michael Carbin, Sasa Misailovic, Michael Kling, and Martin C. Rinard. Detecting and escaping infinite loops with Jolt. In Mira Mezini, editor, *ECOOP 2011 – Object-Oriented Programming*, volume 6813 of *Lecture Notes in Computer Science*, pages 609–633. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-22654-0. doi: 10.1007/978-3-642-22655-7\_28. URL [http://dx.doi.org/10.1007/978-3-642-22655-7\\_28](http://dx.doi.org/10.1007/978-3-642-22655-7_28).
- William S Cleveland and Susan J Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.
- Daniel D Corkill. Blackboard systems. *AI expert*, 6(9):40–47, 1991.
- Daniel D Corkill. Collaborating software: Blackboard and multi-agent systems & the future. In *Proceedings of the International Lisp Conference*, volume 10, New York, New York, October 2003. URL <http://mas.cs.umass.edu/paper/265>.

- N. Cristianini. Are we there yet? *Neural Networks*, 23(4):466–470, 2010.  
URL <http://www.scopus.com/inward/record.url?eid=2-s2.0-77950298664&partnerID=40&md5=667b055d1c9b369488e787d696d9a0dd>.
- Eric Cronin, Burton Filstrup, and Anthony Kurc. A distributed multiplayer game server system. In *University of Michigan*, June 25 2001. URL <http://citeseer.ist.psu.edu/476074.html>; <http://www.eecs.umich.edu/~bfilstru/quakefinal.pdf>.
- Peter J. Denning. Thrashing: its causes and prevention. In *Proceedings of the December 9-11, 1968, Fall joint computer conference, part I*, AFIPS '68 (Fall, part I), pages 915–922, New York, NY, USA, 1968. ACM. doi: <http://doi.acm.org/10.1145/1476589.1476705>. URL <http://doi.acm.org/10.1145/1476589.1476705>.
- Q. Du and J.E. Fowler. Hyperspectral image compression using JPEG2000 and principal component analysis. *Geoscience and Remote Sensing Letters, IEEE*, 4(2):201–205, 2007.
- Nikolas Engelhard, Felix Endres, Jürgen Hess, Jürgen Sturm, and Wolfram Burgard. Real-time 3D visual SLAM with a hand-held RGB-D camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, volume 2011, 2011.
- Lee D Erman, Frederick Hayes-Roth, Victor R Lesser, and D Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys (CSUR)*, 12(2):213–253, 1980.
- Stefano Ferretti and Marco Roccetti. Fast delivery of game events with an optimistic synchronization mechanism in massive multiplayer online games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACE '05, pages 405–412, New York, NY, USA, 2005. ACM. ISBN 1-59593-110-4. doi: 10.1145/1178477.1178570. URL <http://doi.acm.org/10.1145/1178477.1178570>.
- Paul Fitzpatrick, Giorgio Metta, and Lorenzo Natale. Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29 – 45, 2008. ISSN 0921-

8890. doi: <http://dx.doi.org/10.1016/j.robot.2007.09.014>. URL <http://www.sciencedirect.com/science/article/pii/S0921889007001364>.

SB Furber, Steve Temple, and AD Brown. High-performance computing for systems of spiking neurons. In *AISB'06 workshop on GC5: Architecture of Brain and Mind*, volume 2, pages 29–36, Bristol, 2006.

Sadayuki Furuhashi. MessagePack. <http://msgpack.org/>, 2012.

L. Gautier and C. Diot. Design and evaluation of MiMaze a multi-player game on the Internet. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, ICMCS '98, pages 233–236, Washington, DC, USA, jun-1 jul 1998. IEEE Computer Society. ISBN 0-8186-8557-3. doi: 10.1109/MMCS.1998.693647. URL <http://dx.doi.org/10.1109/MMCS.1998.693647>.

Jakub Gemrot, Cyril Brom, Joanna Bryson, and Michal Bída. How to compare usability of techniques for the specification of virtual agents' behavior? an experimental pilot study with human subjects. In Martin Beer, Cyril Brom, Frank Dignum, and Von-Wun Soo, editors, *Agents for Educational Games and Simulations*, volume 7471 of *Lecture Notes in Computer Science*, pages 38–62. Springer, Berlin, 2012. ISBN 978-3-642-32325-6. doi: 10.1007/978-3-642-32326-3\_3. URL <http://opus.bath.ac.uk/31432/>.

Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. YouTube traffic characterization: A view from the edge. In Constantine Dovrolis and Matthew Roughan, editors, *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 15–28, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-908-1. doi: 10.1145/1298306.1298310. URL <http://doi.acm.org/10.1145/1298306.1298310>.

Li Gong. Secure Java class loading. *IEEE Internet Computing*, 2(6):56–61, November 1998. ISSN 1089-7801. doi: 10.1109/4236.735987. URL <http://dx.doi.org/10.1109/4236.735987>.

Maozu Guo, Yang Liu, and Jacek Malec. A new q-learning algorithm based on the metropolis criterion. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(5):2140–2143, 2004.

- E. A. Heinz. Extended futility pruning. *ICCA Journal*, (21):75–83, 1998.
- David R Heise. *Surveying cultures: Discovering shared conceptions and sentiments*. John Wiley & Sons, 2010.
- Helgi Páll Helgason. *General Attention Mechanism for Artificial Intelligence Systems*. PhD thesis, Reykjavik University, May 2013.
- Pieter Hintjens. ØMQ – the guide. <http://zguide.zeromq.org/>, 2014.
- Tobias Hoffeld, Raimund Schatz, Ernst Biersack, and Louis Plissonneau. Internet video delivery in YouTube: From traffic measurements to quality of experience. In Ernst Biersack, Christian Callegari, and Maja Matijasevic, editors, *Data Traffic Monitoring and Analysis*, volume 7754 of *Lecture Notes in Computer Science*, pages 264–301. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36783-0. doi: 10.1007/978-3-642-36784-7\_11. URL [http://dx.doi.org/10.1007/978-3-642-36784-7\\_11](http://dx.doi.org/10.1007/978-3-642-36784-7_11).
- C.L. Hull. *Principles of Behavior: An Introduction to Behavior Theory*. The Century Psychology Series. D. Appleton-Century Company, Incorporated, 1943. URL <http://books.google.ie/books?id=6WB9AAAAMAAJ>.
- Mark Humphrys. W-learning: competition among selfish Q-learners. Technical Report UCAM-CL-TR-362, University of Cambridge, Computer Laboratory, April 1995. URL <http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-362.ps.gz>.
- Mark Humphrys. *Action selection methods using reinforcement learning*. PhD thesis, University of Cambridge, 1997.
- Mark Humphrys. Distributing a Mind on the Internet: The World-Wide-Mind. In Jozef Kelemen and Petr Sosik, editors, *Advances in Artificial Life, 6th European Conference, ECAL 2001, Prague, Czech Republic, September 10-14, 2001, Proceedings*, volume 2159 of *Lecture Notes in Computer Science*, pages 669–680. Springer, 2001a. ISBN 3-540-42567-5. URL <http://link.springer.de/link/service/series/0558/bibs/2159/21590669.htm>.
- Mark Humphrys. The World-Wide-Mind: Draft proposal. Technical Report CA-0301, Dublin City University, School of Computer Applications, Decem-



ber 2001b. URL <http://citeseer.ist.psu.edu/499760.html>; <http://www.compapp.dcu.ie/~humphrys/Publications/01.wwm.TR.ps>.

David R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425, July 1985. ISSN 0164-0925. doi: 10.1145/3916.3988. URL <http://doi.acm.org/10.1145/3916.3988>.

I. T. Jolliffe. *Principal Component Analysis*. Series in Statistics. Springer Verlag, 2002.

Henry F. Kaiser. The Application of Electronic Computers to Factor Analysis. *Educational and Psychological Measurement*, 20(1):141–151, April 1960. doi: 10.1177/001316446002000116. URL <http://dx.doi.org/10.1177/001316446002000116>.

Craig M. Kanarick. A technical overview and history of the SIMNET project. In Vijay Madisetti, David Nicol, and Richard Fujimoto, editors, *Advances in Parallel and Distributed Simulation*, volume 23 of *Simulation*, pages 104–111. Society for Computer Simulation (SCS), San Diego, CA, January 1991.

Mark Kantrowitz. CMU Artificial Intelligence Repository. <http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/0.html>, 2009.

D Kapusta. Connectionist Q-learning Java framework. <http://elsy.gdan.pl>, 2011.

John E. Laird. Extending the Soar cognitive architecture. In Pei Wang, Ben Goertzel, and Stan Franklin, editors, *Artificial General Intelligence*, volume 171 of *Frontiers in Artificial Intelligence and Applications*, pages 224–235. IOS Press, 2008. ISBN 978-1-58603-833-5. URL <http://dblp.uni-trier.de/db/conf/agi/agi2008.html#Laird08>.

John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987. ISSN 0004-3702. doi: [http://dx.doi.org/10.1016/0004-3702\(87\)90050-6](http://dx.doi.org/10.1016/0004-3702(87)90050-6). URL <http://www.sciencedirect.com/science/article/pii/0004370287900506>.

Scott D. Lathrop and John E. Laird. Towards incorporating visual imagery into a cognitive architecture. In Richard L. Lewis, Thad A. Polk, and John E. Laird,

- editors, *Proceedings of the Eighth International Conference on Cognitive Modeling*, pages 25–30. Taylor and Francis / Psychology Press, 2007.
- David Levy, David Broughton, and Mark Taylor. The SEX algorithm in computer chess. *ICCA Journal*, 12(1):10–21, 1989.
- Stanley Lieberman. Limitations in the application of non-parametric coefficients of correlation. *American Sociological Review*, 29(5):744–746, 1964.
- Long-Ji Lin. Hierarchical learning of robot skills by reinforcement. In *Proceedings of 1993 IEEE International Conference on Neural Networks (Joint FUZZ-IEEE'93 and ICNN'93 [IJCNN93])*, volume I, pages 181–186, San Francisco, California, March-April 1993. IEEE/INNS. CMU.
- Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley. *The Java virtual machine specification, Java SE 7 Edition*. Addison-Wesley, Upper Saddle River, NJ, 2013. ISBN 978-0133260441.
- Lowell Lindstrom and Ron Jeffries. Extreme programming and agile software development methodologies. *Information Systems Management*, 21(3):41–52, 2004. doi: 10.1201/1078/44432.21.3.20040601/82476.7. URL <http://www.tandfonline.com/doi/abs/10.1201/1078/44432.21.3.20040601/82476.7>.
- Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, IT-28(2):129–137, March 1982.
- Richard Lowry. VassarStats. <http://faculty.vassar.edu/lowry/VassarStats.html>, October 2010. URL <http://faculty.vassar.edu/lowry/VassarStats.html>.
- Oisín Mac Fhearaí, Mark Humphrys, and Ray Walshe. A high-speed architecture for building hybrid minds. In Joaquim Filipe and Ana L. N. Fred, editors, *ICAART 2011 – Proceedings of the 3rd International Conference on Agents and Artificial Intelligence, Volume 1 – Artificial Intelligence*, pages 659–663, Rome, Italy, 2011. SciTePress. ISBN 978-989-8425-40-9.
- Pattie Maes. The dynamics of action selection. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence – Volume 2, IJCAI'89*, pages

991–997, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. URL <http://dl.acm.org/citation.cfm?id=1623891.1623914>.

Martin Mauve, Stefan Fischer, and Jörg Widmer. A generic proxy system for networked computer games. In Lars C. Wolf, editor, *NETGAMES*, pages 25–28. ACM, 2002. ISBN 1-58113-493-2. URL <http://doi.acm.org/10.1145/566500.566504>.

Martin Mauve, Jürgen Vogel, Volker Hilt, and Wolfgang Effelsberg. Local-lag and timewarp: providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia*, 6(1):47–57, 2004. URL <http://doi.ieeecomputersociety.org/10.1109/TMM.2003.819751>.

John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence*, number 4, pages 463–502. Edinburgh University Press, 1969.

Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. YARP: Yet Another Robot Platform. *International Journal of Advanced Robotic Systems*, 3(1):43–48, 2006.

Marvin Minsky. *The Society of Mind*. Simon & Schuster, Inc., New York, NY, USA, 1986. ISBN 0-671-60740-5.

J.C. Mogul, F. Douglass, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for http. In *ACM SIGCOMM Computer Communication Review*, volume 27, pages 181–194. ACM, 1997.

Brian Monks. Master’s thesis (mof). Master’s thesis, Dublin City University, School of Computing, 2010.

Nithin Nakka, Giacinto Paolo Saggese, Zbigniew Kalbarczyk, and Ravishankar K Iyer. An architectural framework for detecting process hangs/crashes. In Mario Cin, Mohamed Kaâniche, and András Pataricza, editors, *Dependable Computing – EDCC 5*, volume 3463 of *Lecture Notes in Computer Science*, pages 103–121. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-25723-3. doi: 10.1007/11408901\_8. URL [http://dx.doi.org/10.1007/11408901\\_8](http://dx.doi.org/10.1007/11408901_8).

Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

- Allen Newell and Herbert A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, March 1976. ISSN 0001-0782. doi: <http://doi.acm.org/10.1145/360018.360022>.
- Patrick Niemeyer and Joshua Peck. *Exploring Java*. O’Reilly & Associates, Inc., 1998.
- S. Oikawa, M. Sugaya, M. Iwasaki, and T. Nakajima. Using virtualized operating systems as a ubiquitous computing infrastructure. In *Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, 2004. Proceedings*, pages 109–113. IEEE Computer Society, 2004. ISBN 0-7695-2123-1. doi: 10.1109/WSTFES.2004.1300424.
- Ciarán O’Leary and Mark Humphrys. Building a hybrid society of mind using components from ten different authors. In Wolfgang Banzhaf, Thomas Christaller, Peter Dittrich, Jan T. Kim, and Jens Ziegler, editors, *ECAL*, volume 2801 of *Lecture Notes in Computer Science*, pages 839–846. Springer, 2003. ISBN 3-540-20057-6. URL <http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=2801&spage=839>.
- Ciarán O’Leary, Mark Humphrys, and Ray Walshe. Constructing an animat mind using 505 sub-minds from 234 different authors. In S. Schaal, A.J. Ijspeert, A. Billard, S. Vijayakumar, J. Hallam, and J.A. Meyer, editors, *From Animals to Animats 8. Proceedings of the Eighth International Conference on the Simulation of Adaptive Behavior (SAB’04)*, pages 39–48. MIT Press, 2004.
- Peter Österlund. CuckooChess. <http://web.comhem.se/petero2home/javachess/>, January 2014. URL <http://web.comhem.se/petero2home/javachess/>.
- Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.
- R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL <http://www.R-project.org/>. ISBN 3-900051-07-0.

- J. Rao and X. Su. A survey of automated web service composition methods. *Semantic Web Services and Web Process Composition*, pages 43–54, 2005.
- Bernie Roehl. Distributed virtual reality: an overview. In *Proceedings of the first symposium on Virtual reality modeling language, VRML '95*, pages 39–43, New York, NY, USA, 1995. ACM. ISBN 0-89791-818-5. doi: 10.1145/217306.217312. URL <http://doi.acm.org/10.1145/217306.217312>.
- Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 2 edition, 2003. ISBN 0137903952.
- Richard Samuels. Is the human mind massively modular? In *Contemporary Debates in Cognitive Science*. Blackwell, 2006.
- Kiyoshi Sawada. Adding relations in the same level of a linking pin type organization structure. *IAENG International Journal of Applied Mathematics*, 38(1):20–25, 2008.
- Murray Shanahan. A cognitive architecture that combines internal simulation with a global workspace. *Consciousness and Cognition*, 15(2):433–449, 2006. ISSN 1053-8100. doi: <http://dx.doi.org/10.1016/j.concog.2005.11.005>. URL <http://www.sciencedirect.com/science/article/pii/S1053810005001510>.
- Satinder Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. In Leslie P. Kaelbling, editor, *Machine Learning*, volume 22, pages 123–158, Boston, USA, 1996. Kluwer Academic Publishers.
- Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, Boston, MA, 2012. ISBN 113318779X. URL <http://www.amazon.com/Introduction-Theory-Computation-Michael-Sipser/dp/113318779X>.
- Aaron Sloman. How many separately evolved emotional beasts live within us? In *Emotions in Humans and Artifacts*, pages 35–114. MIT Press, May 27 2002.
- Aaron Sloman and Ron Chrisley. Virtual machines and consciousness. *Journal of Consciousness Studies*, 10(4–5):133–172, 2003. URL <http://www.ingentaconnect.com/content/imp/jcs/2003/00000010/F0020004/1350>.

P Sollich and A Krogh. Learning with ensembles: how over-fitting can be useful. volume 8, pages 190–196, San Francisco, CA, 1995. Morgan Kaufmann Publishers. ISBN 9780262201049.

SpringSource. RabbitMQ. <http://www.rabbitmq.com/>, 2012.

L. Steels. A case study in the behavior-oriented design of autonomous agents. In Dave Cliff, Ph. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior, SAB'94*, Complex Adaptive Systems, pages 445–452, MIT Press, 1994. MIT Press. ISBN 0-262-53122-4. URL <http://arti.vub.ac.be/steels/sab94.ps>.

Matthew R. Stein. The pumapaint project. *Auton. Robots*, 15(3):255–265, November 2003. ISSN 0929-5593. doi: 10.1023/A:1026216520523. URL <http://dx.doi.org/10.1023/A:1026216520523>.

Ron Sun and Xi Zhang. Accounting for a variety of reasoning data within a cognitive architecture. *J. Exp. Theor. Artif. Intell.*, 18(2):169–191, 2006. URL <http://dx.doi.org/10.1080/09528130600557713>.

Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998a. URL <http://doi.ieeecomputersociety.org/10.1109/TNN.1998.712192>.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. A Bradford book. Bradford Book, 1998b. ISBN 9780262193986. URL <http://books.google.ie/books?id=CAFR6IBF4xYC>.

Brian Tanner and Adam White. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10: 2133–2136, September 2009.

Kenneth Taylor and Barney Dalton. Issues in Internet telerobotics. In *International Conference on Field and Service Robotics (FSR 97)*, pages 151–157, Canberra, Australia, December 1997. The Australian National University.

- P. Thagard. *Mind Readings: Introductory Selections on Cognitive Science*. A Bradford book. MIT Press, 1998. ISBN 9780262700672. URL [http://books.google.ie/books?id=7\\_\\_\\_V7sUVfgC](http://books.google.ie/books?id=7___V7sUVfgC).
- Kristinn R. Thórisson. Mind model for multimodal communicative creatures and humanoids. *Applied Artificial Intelligence*, 13(4-5):449–486, 1999.
- Kristinn R. Thórisson, Hrvoje Benko, Denis Abramov, Andrew Arnold, Sameer Maskey, and Aruchunan Vaseekaran. Constructionist design methodology for interactive intelligences. *AI Magazine*, 25(4):77–90, 2004. URL <http://www.aaai.org/ojs/index.php/aimagazine/article/view/1786>.
- A. M. Turing. On Computable Numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 2(42):230–265, 1936.
- Toby Tyrrell. *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh, 1993.
- Ubbo Visser and Hans-Dieter Burkhard. RoboCup: 10 years of achievements and future challenges. *The AI Magazine*, 28(2):115–132, 2007.
- Ray Walshe, Mark Humphrys, and Ciarán O’Leary. Constructing complex brains: Building minds using sub-minds from biotechnology authors. In *1st IFIP Conference on Artificial Intelligence Applications and Innovations (AIAI-04), Toulouse, France, August 22-27, 2004, Proceedings*, 2004.
- Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.
- B. L. Welch. The generalization of ‘Student’s’ problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35, 1947. ISSN 00063444. URL <http://www.jstor.org/stable/2332510>.
- Frank Westad, Margrethe Hersletha, Per Lea, and Harald Martens. Variable selection in PCA in sensory descriptive and consumer data. *Food Quality and Preference*, 14(5-6):463–472, 2003. ISSN 0950-3293. doi: 10.1016/S0950-3293(03)00015-6. URL <http://www.sciencedirect.com/science/article/pii/S0950329303000156>. The Sixth Sense – 6th Sensometrics Meeting.

M. B. Wilk and R. Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55(1):pp. 1–17, 1968. ISSN 00063444. URL <http://www.jstor.org/stable/2334448>.

Stewart W. Wilson. The animat path to AI. In Jean-Arcady Meyer and Stewart W. Wilson, editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 15–21, Cambridge, MA, 1991. MIT Press.

Terry Winograd. *Understanding Natural Language*. Academic Press, New York, NY, 1972.

Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya-Qin Zhang, and Jon M Peha. Streaming video over the Internet: approaches and directions. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(3):282–300, 2001.



# Appendix A

## List of publications

Various aspects of the research described in this thesis were previously published in:

- Oisín Mac Fhearaí and Mark Humphrys and Ray Walshe, “A High-Speed Architecture for Building Hybrid Minds”, in Joaquim Filipe and Ana L. N. Fred, editors, *3rd International Conference on Agents and Artificial Intelligence (ICAART 2011)* (INSTICC, 2011), pp. 659–663.
- Oisín Mac Fhearaí and Mark Humphrys and Ray Walshe, "A Framework for Scaling Up Distributed Minds", in Darina Dicheva, Zdravko Markov, Eliza Stefanova, editors, *Advances in Intelligent and Soft Computing, 2011, Volume 101, Third International Conference on Software, Services and Semantic Technologies (S3T 2011)* (Springer, 2011), pp. 211–212.
- Oisín Mac Fhearaí and Ray Walshe and Mark Humphrys, “Assembling Hybrid Minds”, in Martin McGinnity, editor, *Proceedings of the 22nd Annual Irish conference on Artificial Intelligence and Cognitive Science, Belfast, Ireland (AICS 2011)*, pp. 76–85.

## Appendix B

# Description of the “Tyrrell09” world

Many of the experiments performed in the course of this research consisted of running a mind in the Tyrrell09 world, an implementation of Tyrrell’s simulated environment (SE), a complex animal behaviour problem designed to serve as a useful testbed for evaluating different mechanisms of action-selection [Tyrrell, 1993]. O’Leary et al. [2004] implemented Tyrrell’s SE in the Java programming language, attempting to stay as truthful to the original version as possible, complete with a graphical representation of the world drawn with Java’s Abstract Windowing Toolkit (AWT). The Tyrrell09 world is a branch by this author of O’Leary’s implementation, with a small number of bugfixes and some changes to the world configuration which attempt to reduce noise and variability for the purpose of the experiments laid out in this dissertation.

This appendix will give a brief overview and background of Tyrrell’s world before discussing the specific modifications made to the world for this research.

### B.1 Overview

In Tyrrell’s SE, a two-dimensional grid-based environment is described. This grid is randomly populated with multiple features, including hazardous places and animals, food and water sources, shelter and potential mates. A small omnivorous animal is controlled by the action-selection mechanism (ASM), with one primary objective: to

maximise the simulated animal's genetic fitness (specified by the number of times it successfully mates with another animal of its species).

The SE was designed to serve as a testbed for evaluating action-selection mechanisms in a complex and dynamic environment [Tyrrell, 1993]. To attain a sufficient degree of difficulty for this purpose, the primary objective of increasing genetic fitness is dependent on a number of subgoals, many of which represent a significant challenge by themselves. For example, the animal must find, court and attempt to mate with a potentially receptive partner (attempting to mate with an unreceptive partner can result in serious injury to the animal). Similarly, to survive for more than a short duration (and therefore gain opportunities to mate), the animal must eat non-toxic food, drink water, and avoid hazards such as toxic food, heatstroke, dangerous places and predators.

To further increase the challenge for an ASM, the sensory perception of the animal is restricted to a few grid squares in each direction and obscured with a degree of random noise, and its ability to obey selected actions is made fallible.

The SE attempts to separate the problems of perception, navigation and motor control from the task of action selection.

### **B.1.1 Perception**

Sensory input is presented not as raw signals from nerve receptors, but as a set of labelled homeostatic and external stimuli, generally expressed in the form of real-valued numbers of the range  $[0 - 1.0]$ . For example, *perceived cleanliness* is a homeostatic variable defined such that a value of 1.0 means perfectly clean and 0 means maximally dirty. The variable's level is reduced over time by a pseudo-random process (not in response to specific events in the world), and is increased when a cleaning action is performed by the animal, at the cost of a decrement to the animal's health in proportion with the degree of dirtiness (this damage can be understood as due to parasite infestation or infection of wounds).

To further add to the environment's complexity, many features in the world are dynamic. Light levels change, animals move about, and resources in the world (such as a pool of water or a food source) change in quantity.

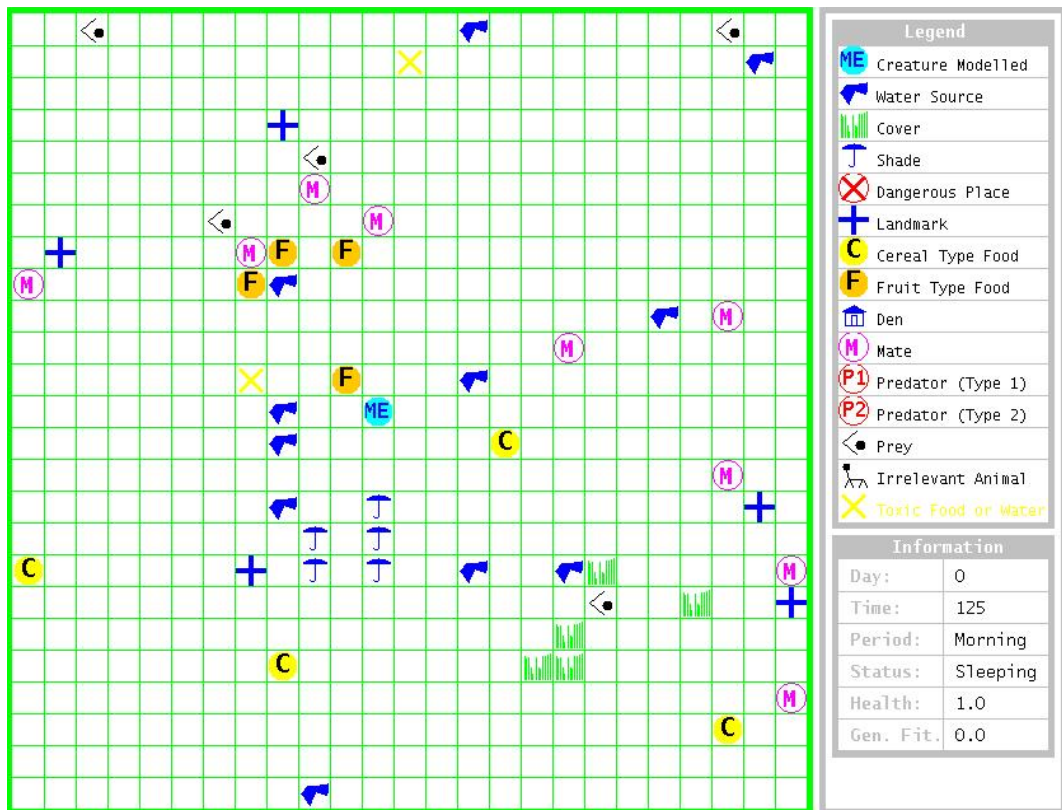


Figure B.1: An image representing a single timestep in the Tyrrell world. The world is rendered as a two-dimensional grid with coloured icons denoting various features in the environment which are labelled in the legend on the top-right of picture. This version of the world was tweaked to make it slightly more sparse and less dangerous than the default configuration – note the lack of dangerous places and predators on the map.

### **B.1.2 Navigation**

A navigation mechanism is built into the simulated animal and presented to the ASM as an extra set of sensory inputs – for example, in addition to a food *perception* stimulus, there is also a food *memory* stimulus which reflects the degree to which the animal (fallibly) remembers food in its own square and in eight directions.

### **B.1.3 Motor control**

The SE models a transformation from a labelled action (a number representing one of 35 possible high-level actions the animal can perform) into a low-level system of muscle contraction or motor impulses which execute the desired action. These actions are chosen from a set of 35 possible behaviours including drinking, cleaning, sleeping, courting, mating and moving in one of eight directions on the two-dimensional grid.

To increase realism and apparent non-determinism of the environment, actions are error-prone, with a probability that the animal will not successfully execute the action.

## **B.2 Differences from Tyrrell’s original simulated environment**

In the original implementation of Tyrrell’s SE and (by definition, since it attempted to be a faithful re-implementation) O’Leary’s implementation as a World in the World-Wide Mind 1.0 system, the SE was a very noisy environment. This was an intentional design feature, as the goal of the SE was to provide a challenging test of the various action-selection methods it would be used to evaluate. In his thesis [Tyrrell, 1993], Tyrrell introduced a number of previous simulated environments that were lacking the degree of complexity which he felt was necessary to properly exercise an ASM. Citing one taxonomy for simulated environments by Wilson [1991], Tyrrell designed his SE such that it presented a complex problem in terms of each of the following concerns:

1. Average latency between a correct action being performed and its reward being received,

2. Consistency of rewards after performing the same action in response to the same stimuli,
3. Regularity of features in the environment,
4. Degree of perceptual noise, and
5. Degree of noise in the reward function.

For the most part, Tyrrell's SE attempts to maximise the difficulty of each of these five aspects in Wilson's taxonomy. However, the degree of noise presented and the variance in performance by minds in Tyrrell's world was very high, which made it more difficult to reason about the performance of individual minds. For this reason, in creating the Tyrrell09 world, I simplified the environment by minimising or removing entirely some features and increased the maximum view distance (from 3 squares in each direction to 6), as well as reducing to zero the likelihood of the simulated animal performing an action other than the one specified by the controlling mind.

The creature's maximum lifespan is unchanged, but the likelihood of predators, irrelevant animals and dangerous places appearing was reduced to near-zero. Noise in sensory percepts was reduced as far as possible.

### B.3 World configuration file

Many of the parameters underlying the world model can be altered through a configuration file. This encompasses all the changes made to the Tyrrell09 world, and can be seen below.

```
# number of columns
org.w2mind.tyrrell.environment.ENV_ARR_COLS = 25
# number of rows
org.w2mind.tyrrell.environment.ENV_ARR_ROWS = 25
# number of squares distance that can be perceived
org.w2mind.tyrrell.environment.M_A_P = 6
org.w2mind.tyrrell.environment.MAX_ANIMAL_LIFESPAN = 10
org.w2mind.tyrrell.environment.RECOVERY_RATE = 0.003
# good perception
org.w2mind.tyrrell.environment.NIGHT_PERCEPTION_FACTOR = 1
```

```

org.w2mind.tyrrell.environment.DUSK_PERCEPTION_FACTOR = 1
org.w2mind.tyrrell.environment.DAY_PERCEPTION_FACTOR = 1
org.w2mind.tyrrell.environment.LOOKING_DIRECTION_PERC_FACTOR = 1
# noise in actions
org.w2mind.tyrrell.environment.FAULTY_ACTION_PROB = 0
# can store features
org.w2mind.tyrrell.environment.MAP_SIZE = 100
# perfect memory
org.w2mind.tyrrell.environment.MEMORY_DECAY_FACTOR = 1
# navigation error
org.w2mind.tyrrell.environment.NAV_ERROR = 0
# for many of these:
# even if set to 0, they do not go all the way to 0, they just get
    very sparse
org.w2mind.tyrrell.environment.INIT_NUM_CEREAL_FOOD = 5
org.w2mind.tyrrell.environment.INIT_NUM_COVER = 5
org.w2mind.tyrrell.environment.INIT_NUM_DANGEROUS_PLACE = 0
org.w2mind.tyrrell.environment.INIT_NUM_DEN = 1
org.w2mind.tyrrell.environment.INIT_NUM_FRUIT_FOOD = 5
org.w2mind.tyrrell.environment.INIT_NUM_IRRELEVANT = 0
org.w2mind.tyrrell.environment.INIT_NUM_LANDMARK = 5
org.w2mind.tyrrell.environment.INIT_NUM_MATE = 15
org.w2mind.tyrrell.environment.INIT_NUM_PREDATOR_1 = 0
org.w2mind.tyrrell.environment.INIT_NUM_PREDATOR_2 = 0
org.w2mind.tyrrell.environment.INIT_NUM_PREY = 5
org.w2mind.tyrrell.environment.INIT_NUM_SHADE = 5
org.w2mind.tyrrell.environment.INIT_NUM_WATER = 18
org.w2mind.tyrrell.environment.MAX_NUM_CEREAL_FOOD = 10
org.w2mind.tyrrell.environment.MAX_NUM_COVER = 10
org.w2mind.tyrrell.environment.MAX_NUM_DANGEROUS_PLACE = 0
org.w2mind.tyrrell.environment.MAX_NUM_DEN = 1
org.w2mind.tyrrell.environment.MAX_NUM_FRUIT_FOOD = 10
org.w2mind.tyrrell.environment.MAX_NUM_IRRELEVANT = 0
org.w2mind.tyrrell.environment.MAX_NUM_LANDMARK = 10
org.w2mind.tyrrell.environment.MAX_NUM_MATE = 20
org.w2mind.tyrrell.environment.MAX_NUM_PREDATOR_1 = 0
org.w2mind.tyrrell.environment.MAX_NUM_PREDATOR_2 = 0
org.w2mind.tyrrell.environment.MAX_NUM_PREY = 20
org.w2mind.tyrrell.environment.MAX_NUM_SHADE = 10

```

```

org.w2mind.tyrrell.environment.MAX_NUM_WATER = 22
org.w2mind.tyrrell.environment.MAX_NUM_TOXIC = 0
# the maximum number of any individual type of feature that is likely
  to occur at any one time in the field of perception for the animal
org.w2mind.tyrrell.environment.MAX_NUM_SINGLE_PERC_FEATURE = 50
# minimum value of food or water, below which it is not worth the
  animal eating or drinking from that source
org.w2mind.tyrrell.environment.MIN_WORTHWHILE_VALUE = 0.01
# maximum value of toxicity in food or water, above which it is not
  worth the animal eating or drinking from that source
org.w2mind.tyrrell.environment.MAX_ACCEPTABLE_TOXICITY = 0.40
# time for which the animal should ignore water or a food type if it
  has been visited recently with unsuccessful results
org.w2mind.tyrrell.environment.IGNORE_USELESS_TIME = 10
# minimum thickness of cover, below which it is not worth the animal
  using the cover to hide in
org.w2mind.tyrrell.environment.MIN_WORTHWHILE_COVER = 0.30
org.w2mind.tyrrell.environment.DAY_LENGTH = 500
org.w2mind.tyrrell.environment.OPT_DRYNESS = 0.825
org.w2mind.tyrrell.environment.COVER_CLUSTER_SIZE = 12
org.w2mind.tyrrell.environment.DANGEROUS_PLACE_CLUSTER_SIZE = 3
org.w2mind.tyrrell.environment.FRUIT_FOOD_CLUSTER_SIZE = 5
org.w2mind.tyrrell.environment.IRRELEVANT_CLUSTER_SIZE = 8
org.w2mind.tyrrell.environment.PREDATOR_1_CLUSTER_SIZE = 1+r2.5
org.w2mind.tyrrell.environment.SHADE_CLUSTER_SIZE = 5
org.w2mind.tyrrell.environment.COVER_CLUSTER_WIDTH = 3
org.w2mind.tyrrell.environment.DANGEROUS_PLACE_CLUSTER_WIDTH = 1
org.w2mind.tyrrell.environment.FRUIT_FOOD_CLUSTER_WIDTH = 2
org.w2mind.tyrrell.environment.IRRELEVANT_CLUSTER_WIDTH = 0
org.w2mind.tyrrell.environment.PREDATOR_1_CLUSTER_WIDTH = 2
org.w2mind.tyrrell.environment.SHADE_CLUSTER_WIDTH = 2
org.w2mind.tyrrell.environment.C_F_RECUP_RATE = 0.2
org.w2mind.tyrrell.environment.F_F_RECUP_RATE = 0.3
org.w2mind.tyrrell.environment.WATER_RECUP_RATE = 0.03
# number of squares distance that predator 1 can perceive
org.w2mind.tyrrell.environment.M_P1_P = 2.5
# number of squares distance that predator 2 can perceive
org.w2mind.tyrrell.environment.M_P2_P = 3
org.w2mind.tyrrell.environment.MAX_MOVING_DISTANCE = 1.5

```



```

org.w2mind.tyrrell.environment.MAX_MOVING_FAST_DISTANCE = 2.9
org.w2mind.tyrrell.environment.IRRELEVANT_DAY_APPEARANCE_PROB = 0.003
org.w2mind.tyrrell.environment.MATE_DAY_APPEARANCE_PROB = 0.07
org.w2mind.tyrrell.environment.PREDATOR_1_DAY_APPEARANCE_PROB = 0.003
org.w2mind.tyrrell.environment.PREDATOR_2_DAY_APPEARANCE_PROB = 0.0003
org.w2mind.tyrrell.environment.PREY_DAY_APPEARANCE_PROB = 0.008
org.w2mind.tyrrell.environment.IRRELEVANT_DUSK_APPEARANCE_PROB = 0.003
org.w2mind.tyrrell.environment.MATE_DUSK_APPEARANCE_PROB = 0.03
org.w2mind.tyrrell.environment.PREDATOR_1_DUSK_APPEARANCE_PROB = 0.004
org.w2mind.tyrrell.environment.PREDATOR_2_DUSK_APPEARANCE_PROB = 0.006
org.w2mind.tyrrell.environment.PREY_DUSK_APPEARANCE_PROB = 0.01
org.w2mind.tyrrell.environment.IRRELEVANT_NIGHT_APPEARANCE_PROB = 0
org.w2mind.tyrrell.environment.MATE_NIGHT_APPEARANCE_PROB = 0
org.w2mind.tyrrell.environment.PREDATOR_1_NIGHT_APPEARANCE_PROB =
0.0025
org.w2mind.tyrrell.environment.PREDATOR_2_NIGHT_APPEARANCE_PROB =
0.012
org.w2mind.tyrrell.environment.PREY_NIGHT_APPEARANCE_PROB = 0
org.w2mind.tyrrell.environment.IRRELEVANT_MOVE_PROB = 0.5
org.w2mind.tyrrell.environment.MATE_MOVE_PROB = 0.35
org.w2mind.tyrrell.environment.PREDATOR_1_MOVE_PROB = 0.5
org.w2mind.tyrrell.environment.PREDATOR_2_MOVE_PROB = 0.75
org.w2mind.tyrrell.environment.PREY_MOVE_PROB = 0.30
org.w2mind.tyrrell.environment.CEREAL_FOOD_NEAR_WATER_PROB = 0.2
org.w2mind.tyrrell.environment.COVER_NEAR_WATER_PROB = 0.8
org.w2mind.tyrrell.environment.DEN_NEAR_WATER_PROB = 0.8
org.w2mind.tyrrell.environment.FRUIT_FOOD_NEAR_WATER_PROB = 0.75
org.w2mind.tyrrell.environment.SHADE_NEAR_WATER_PROB = 0.4
org.w2mind.tyrrell.environment.CEREAL_FOOD_TO_WATER_DIST = 2
org.w2mind.tyrrell.environment.COVER_TO_WATER_DIST = 1
org.w2mind.tyrrell.environment.DEN_TO_WATER_DIST = 2
org.w2mind.tyrrell.environment.FRUIT_FOOD_TO_WATER_DIST = 2
org.w2mind.tyrrell.environment.SHADE_TO_WATER_DIST = 2
org.w2mind.tyrrell.environment.CEREAL_FOOD_REMOVE_CREATE_PROB =
0.00005
org.w2mind.tyrrell.environment.CEREAL_FOOD_CHANGE_CONDITIONS_PROB =
0.00002
org.w2mind.tyrrell.environment.CEREAL_FOOD_CHANGE_TOXICITY_PROB =
0.00005

```

org.w2mind.tyrrell.environment.COVER\_REMOVE\_CREATE\_PROB = 0.00002  
org.w2mind.tyrrell.environment.COVER\_CHANGE\_THICKNESS\_PROB = 0.00002  
org.w2mind.tyrrell.environment.FRUIT\_FOOD\_REMOVE\_CREATE\_PROB = 0.00002  
org.w2mind.tyrrell.environment.FRUIT\_FOOD\_CHANGE\_CONDITIONS\_PROB =  
0.00002  
org.w2mind.tyrrell.environment.FRUIT\_FOOD\_CHANGE\_TOXICITY\_PROB =  
0.00005  
org.w2mind.tyrrell.environment.SHADE\_REMOVE\_CREATE\_PROB = 0.00001  
org.w2mind.tyrrell.environment.SHADE\_CHANGE\_THICKNESS\_PROB = 0.00005  
org.w2mind.tyrrell.environment.WATER\_REMOVE\_CREATE\_PROB = 0.0001  
org.w2mind.tyrrell.environment.WATER\_CHANGE\_SIZE\_PROB = 0.00002  
org.w2mind.tyrrell.environment.WATER\_CHANGE\_TOXICITY\_PROB = 0.00005  
org.w2mind.tyrrell.environment.CEREAL\_FOOD\_EATEN\_PER\_TIMESTEP = 0.02  
org.w2mind.tyrrell.environment.FRUIT\_FOOD\_EATEN\_PER\_TIMESTEP = 0.04  
org.w2mind.tyrrell.environment.WATER\_DRUNK\_PER\_TIMESTEP = 0.06  
org.w2mind.tyrrell.environment.INIT\_TIMESTEP\_VALUE = 125  
org.w2mind.tyrrell.environment.CEREAL\_FOOD\_TOXICITY\_PROB = 0.05  
org.w2mind.tyrrell.environment.FRUIT\_FOOD\_TOXICITY\_PROB = 0.1  
org.w2mind.tyrrell.environment.WATER\_TOXICITY\_PROB = 0.05  
org.w2mind.tyrrell.environment.RAIN\_TO\_CLOUD\_PROB = 0.240  
org.w2mind.tyrrell.environment.RAIN\_TO\_SUN\_PROB = 0.045  
org.w2mind.tyrrell.environment.CLOUD\_TO\_RAIN\_PROB = 0.180  
org.w2mind.tyrrell.environment.CLOUD\_TO\_SUN\_PROB = 0.180  
org.w2mind.tyrrell.environment.SUN\_TO\_CLOUD\_PROB = 0.240  
org.w2mind.tyrrell.environment.SUN\_TO\_RAIN\_PROB = 0.045  
org.w2mind.tyrrell.environment.MIN\_RAIN\_TEMP = -1  
org.w2mind.tyrrell.environment.MAX\_RAIN\_TEMP = 0.35  
org.w2mind.tyrrell.environment.MIN\_CLOUD\_TEMP = -0.7  
org.w2mind.tyrrell.environment.MAX\_CLOUD\_TEMP = 0.7  
org.w2mind.tyrrell.environment.MIN\_SUN\_TEMP = -0.35  
org.w2mind.tyrrell.environment.MAX\_SUN\_TEMP = 1  
org.w2mind.tyrrell.environment.NIGHT\_TEMP\_INCR = -0.1  
org.w2mind.tyrrell.environment.DUSK\_TEMP\_INCR = 0  
org.w2mind.tyrrell.environment.DAY\_TEMP\_INCR = 0.1  
org.w2mind.tyrrell.environment.FOOD\_OPT\_VALUE = 0.5  
org.w2mind.tyrrell.environment.WATER\_OPT\_VALUE = 0.5  
org.w2mind.tyrrell.environment.TEMPERATURE\_OPT\_VALUE = 0.5  
org.w2mind.tyrrell.environment.MIN\_OK\_FAT = 0.25  
org.w2mind.tyrrell.environment.MAX\_OK\_FAT = 0.75

org.w2mind.tyrrell.environment.MIN\_OK\_CARBO = 0.25  
org.w2mind.tyrrell.environment.MAX\_OK\_CARBO = 0.75  
org.w2mind.tyrrell.environment.MIN\_OK\_PROTEIN = 0.25  
org.w2mind.tyrrell.environment.MAX\_OK\_PROTEIN = 0.75  
org.w2mind.tyrrell.environment.MIN\_OK\_WATER = 0.25  
org.w2mind.tyrrell.environment.MAX\_OK\_WATER = 0.75  
org.w2mind.tyrrell.environment.MIN\_OK\_TEMP = 0.25  
org.w2mind.tyrrell.environment.MAX\_OK\_TEMP = 0.75  
org.w2mind.tyrrell.environment.MAX\_FAT\_DECREMENT = 0.00135  
org.w2mind.tyrrell.environment.MAX\_CARBO\_DECREMENT = 0.0018  
org.w2mind.tyrrell.environment.MAX\_PROTEIN\_DECREMENT = 0.0009  
org.w2mind.tyrrell.environment.MAX\_WATER\_DECREMENT = 0.0020  
org.w2mind.tyrrell.environment.T\_H\_LENGTH = 5  
org.w2mind.tyrrell.environment.R\_H\_LENGTH = 5  
org.w2mind.tyrrell.environment.IRRELEVANT\_PACK\_CLOSENESS = 10  
org.w2mind.tyrrell.environment.PREDATOR\_1\_PACK\_CLOSENESS = 15  
org.w2mind.tyrrell.environment.MAX\_NUM\_IRRELEVANT\_PACKS = 30  
org.w2mind.tyrrell.environment.MAX\_NUM\_PREDATOR\_1\_PACKS = 30  
org.w2mind.tyrrell.environment.CEREAL\_FOOD\_FAT\_RATIO = 0.27  
org.w2mind.tyrrell.environment.CEREAL\_FOOD\_CARBO\_RATIO = 0.55  
org.w2mind.tyrrell.environment.CEREAL\_FOOD\_PROTEIN\_RATIO = 0.18  
org.w2mind.tyrrell.environment.FRUIT\_FOOD\_FAT\_RATIO = 0.35  
org.w2mind.tyrrell.environment.FRUIT\_FOOD\_CARBO\_RATIO = 0.47  
org.w2mind.tyrrell.environment.FRUIT\_FOOD\_PROTEIN\_RATIO = 0.18  
org.w2mind.tyrrell.environment.PREY\_FAT\_RATIO = 0.40  
org.w2mind.tyrrell.environment.PREY\_CARBO\_RATIO = 0.3  
org.w2mind.tyrrell.environment.PREY\_PROTEIN\_RATIO = 0.3  
org.w2mind.tyrrell.environment.MAX\_AN\_TEMP\_INCREASE = 0.1  
org.w2mind.tyrrell.environment.MAX\_AN\_TEMP\_DECREASE = 0.1

## Appendix C

# Description of the “ChessWorldG” world

As discussed in section 5.9.3 (p. 97), as part of an undergraduate artificial intelligence course, students were tasked with writing minds to solve the chess problem presented by the *ChessWorldG* world. This appendix describes that world in brief.

### C.1 Problem to be solved

In *ChessWorldG*, the game of chess with standard rules is modelled, with a challenging A.I. opponent. The game engine and opponent are both implemented by CuckooChess [Österlund, 2014], an open-source library which provides a powerful A.I. player using techniques such as quiescence search [Beal, 1990], futility pruning [Heinz, 1998] and late move reductions [Levy et al., 1989].

### C.2 Rules

#### C.2.1 Invalid moves

Invalid moves are treated as a forfeit – the current game will be counted as a loss and the “Invalid moves” score will be decremented by one. This score starts at zero and grows downwards so that, all other things being equal, a mind which made no invalid moves will be ranked higher on the world scoreboard than a mind which made one invalid move.

### C.2.2 Draws

There is no facility for the mind to request a draw, even after 40 moves, after which in real chess either player can insist upon a draw.

### C.2.3 Move time limit

To discourage solutions which perform extremely deep tree searches, a move time limit of 700 milliseconds is enforced, after which the game is considered forfeit. For the sake of fairness, the opponent A.I. player is restricted to 500 milliseconds' thinking time. As a result, it is possible to exploit this fairness by having a mind re-use the opponent A.I. code (included in the world JAR file) with a time limit between 500 and 700 milliseconds.

### C.2.4 Score

The score vector for ChessWorldG consists of the fields described in table C.1.

## C.3 State format

The board state is presented to the mind in a compact textual representation known as Forsyth-Edwards Notation (FEN) . The starting position looks like this:

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
```

Each row of the board is separated by a forward slash, and lower-case letters represent the black pieces with the mapping: r  $\Rightarrow$  rook, n  $\Rightarrow$  knight, b  $\Rightarrow$  bishop, q  $\Rightarrow$  queen, k  $\Rightarrow$  king, p  $\Rightarrow$  pawn. White's pieces follow the same mapping but in upper-case. Digits represent consecutive empty cells, so an empty row is indicated by an 8. The board layout is followed by a space then either a "w" or "b" which indicates whether white or black is to move. Then the availability of castling moves is shown (in this case, both white and black can castle on kingside and queenside) followed finally by a turn counter.

Field	Description
Won	The number of wins achieved by the mind.
Drew	The number of draws achieved by the mind.
Avg. moves to win	The negative of the total number of moves made in games won by the mind.
Survival moves	The total number of moves made in games lost by the mind.
Invalid moves	The negative of the total number of invalid moves made by the mind.
Captured Queen	The number of queens captured by the mind.
Captured Bishop	The number of bishops captured by the mind.
Captured Knight	The number of knights captured by the mind.
Captured Rook	The number of rooks captured by the mind.
Captured Pawn	The number of pawns captured by the mind.
Promoted Pawn	The number of pawns promoted by the mind to queen – no choice is given.
Castled	The number of times the mind performed a castling move.
Enemy in Check	The number of times the mind put the enemy in check.
Player in Check	The negative of the number of times the enemy put the mind in check.
Lost Queens	The negative of the number of queens captured by the enemy.
Lost Bishops	The negative of the number of bishops captured by the enemy.
Lost Knights	The negative of the number of knights captured by the enemy.
Lost Rooks	The negative of the number of rooks captured by the enemy.
Lost Pawns	The negative of the number of pawns captured by the enemy.

Table C.1: A description of each field included in the score vector for ChessWorldG. Most of these values are calculated by parsing the state at each timestep in the world.

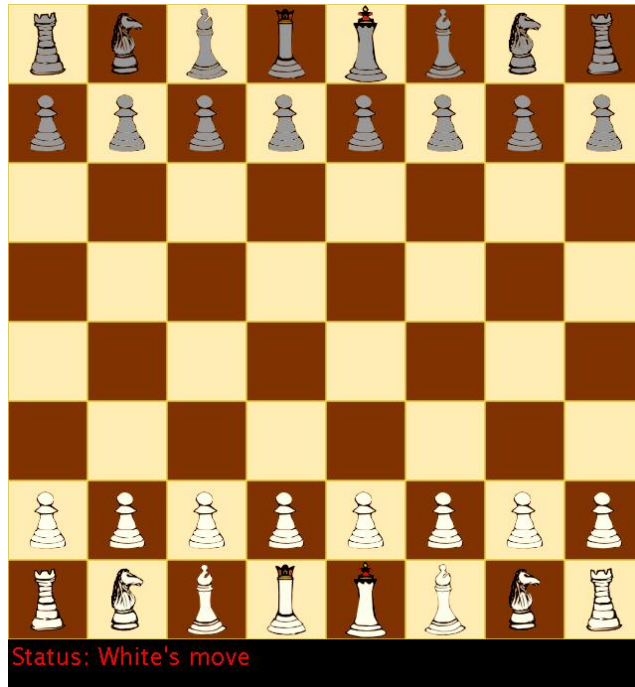


Figure C.1: The state of the board at the beginning of a game.

## C.4 Action format

The chessboard is numbered by cell, from zero in the bottom left (white rook) to 63 in the top right (black rook). Each move is specified simply as the numeric value of the source and destination indices. For example, moving a piece from the bottom left corner of the board forward by one row is specified by the string “0,8”.

## C.5 Graphics

To reduce the learning curve for mind authors slightly, a graphical representation of the board state was implemented with hand-drawn images for the chess pieces. The starting state of the board is shown in figure C.1.

## Appendix D

# Front-end interface

### D.1 Introduction

This appendix provides a brief description of the front-end interface and its primary features, in the context of the architectural requirements which were the topic of section §5.3 (p. 66).

A fundamental feature of the World-Wide Mind project is the ability to perform online runs. This could be achieved by installing some software on the user's local machine, for example, the client program shown in figure 5.1 on page 71.

Another option is to implement all the functionality in a web-based interface – this has the advantage of not requiring users to download any extra software to explore the worlds and minds.

The front-end implementation was performed by Mark Humphrys, with some collaboration from Brian Monks and myself, and accordingly is included as an appendix to the rest of the work in this dissertation.

### D.2 Solutions chosen

Rather than expect users to download client-side software before being able to run any mind in a world, we chose to implement a web-based interface, where runs are carried out entirely on the server side and all user interface mediated through the user's web browser.

This interface replaced the local Java Swing client interface and provides a number of capabilities that take advantage of the centralised server design discussed



previously.

The move to a more centralised architecture makes it possible to automate certain important functions which were implemented in an ad-hoc or manual fashion in the previous work (for example, section §4.8 explains how users were required to email their minds to an administrator who would then run batch conversion and execution scripts).

### **D.3 Automated mind and world uploading**

The manual workflow for uploading minds, as described in section §4.8, was sustainable for a small number of users for a short period of time. It did, however, lead to occasional delays while waiting for one’s submitted mind to be processed and made available on the server as a new web service.

One fundamental tenet espoused by contemporary “Agile” software engineering methodologies is that the cycle of coding and testing should be short in duration [Lindstrom and Jeffries, 2004], so that it can be performed frequently, or even continuously, with the programmer receiving almost instant feedback on the logical errors that inevitably occur.

Therefore, it makes sense to minimise this code-to-test-output latency. In the context of the World-Wide Mind, then, a major reduction in this latency was gained by automating and simplifying the process of uploading minds and making them ready for testing.

This was achieved by adding a form to the web interface (depicted in figure D.1) which allows users to describe and upload their mind and world JAR files. When these JAR files are uploaded, records are added to the database containing the information needed to support those minds and worlds as executable services, as well as other metadata describing those services.

### **D.4 Browsing the available minds and worlds**

The addition of a database makes it straightforward to enumerate the set of uploaded mind and world services on a World-Wide Mind server. The front-end allows users to browse through the list of worlds, providing information about each world, as well

## Upload a new World

See [How to write a World](#).

Required fields are in green.

World JAR file	<input type="text"/> <input type="button" value="Browse_"/>	Must be of the form <code>MyWorld.jar</code>
Description (home page)	<input type="text"/> <input type="button" value="Browse_"/>	Home page to describe this World. Should explain the state, action and score format. Must be called <code>index.html</code>
Sample skeleton Mind for this World	<input type="text"/> <input type="button" value="Browse_"/>	Should show how to parse the state and generate actions for this World. Must be of the form <code>MyMind.java</code>
Description (supporting files)	<input type="text"/> <input type="button" value="Browse_"/> <input type="text"/> <input type="button" value="Browse_"/> <input type="text"/> <input type="button" value="Browse_"/> <input type="text"/> <input type="button" value="Browse_"/> <input type="text"/> <input type="button" value="Browse_"/> <input type="text"/> <input type="button" value="Browse_"/>	You can upload up to 6 supporting files that the home page can embed or link to. (Images, further web pages, etc.) These will all go in the same directory (so use relative links). Allowed file extensions: <code>java, txt, html, htm, jpg, gif, png, pdf</code>

Figure D.1: The process of uploading a mind or world to a World-Wide Mind server is now controlled by the end user. Once a working JAR file is built, containing the compiled mind or world classes, it can be submitted to the server via a web form which places the JAR file in the appropriate location and updates the server's database of services.

## Scoreboard for World: [w2m.TyrrellWorld](#)

Rank	Mind	Mated	Steps	Runs	Minds called	Author	Date created	Description	Run	Run (images)
1	<a href="#">SeekAdvice2</a>	60	4832	9	<a href="#">1</a>	<a href="#">danielm</a>	5 Oct 2010	<a href="#">Description</a>		
2	<a href="#">SleepWalker8</a>	58	4046	5	<a href="#">1</a>	<a href="#">eoincos</a>	5 Oct 2010	<a href="#">Description</a>		
3	<a href="#">MacMind</a>	57	4671	7	<a href="#">2</a>	<a href="#">Mac</a>	5 Oct 2010			
4	<a href="#">callMyMinds7</a>	57	4009	3	<a href="#">3</a>	<a href="#">zhangj5</a>	5 Oct 2010			
5	<a href="#">MacReturnsSeeksAdvice</a>	55	4593	4	<a href="#">1</a>	<a href="#">Mac</a>	5 Oct 2010	<a href="#">Description</a>		

Figure D.2: Part of the scoreboard for the Tyrrell world (a simulated animal behaviour problem, described more completely in appendix B), showing the best scores of the four highest-scoring minds submitted, sorted by their performance on two metrics defined by the world author: the number of times the simulated animal mated, and the number of timesteps survived. The “minds called” link for each entry shows the total count of the subminds called by each mind, if any. Clicking the numbers will produce a listing which enumerates those subminds, allowing a recursive traversal of their subminds in turn.

as links to that world’s automatically-generated scoreboard, and to the set of minds uploaded with the intention of solving the problems posed by the world.

Similarly, for every mind, the interface provides a list of every submind it has called (allowing one to explore the *call graph* of minds<sup>1</sup>), and links to a user-supplied description page if one was supplied.

## D.5 Initiating and controlling runs

### D.5.1 Starting a run

The two “play” buttons shown on the right of a world scoreboard page (for example, the one shown in figure D.2) allow the user to initiate a run of a mind in that world. These buttons serve as HTML links to the “run.php” page, taking three parameters:

- **mind** – The canonical name of the mind class,
- **world** – The canonical name of the world class, and
- **images** – A boolean flag indicating whether the user wants images to be generated.

So, for example, clicking the button shown in the top right of figure D.2 might cause the browser to request the URL:

```
http://w2mind/sys/run.php?mind=JonnyMind1&world=w2m.Tyrrell10&images=true
```

By manually constructing an URL of this format, one can attempt to start a run of a mind in the wrong world, which will most likely result in an error since the world classes for states and actions will almost certainly be incompatible.

When browsing the list of minds or scoreboard for any world, the only minds which are displayed are those specified by the author as compatible with that world, or which have successfully completed a run in that world in the past.

Once either one of the “play” buttons has been clicked, the web server initiates a run by calling the run logger program (detailed in section 5.4.1), containing the

---

<sup>1</sup>In general, of course, one would expect this structure to be a tree rather than a graph, since a cyclic dependency between two hybrid minds could potentially result in infinite recursion.

parameters described above: the canonical names of the mind and world, and a flag specifying whether images should be generated by the world or not.

The call to the run logger blocks until the process is completed and the run is completed, at which point either an error message will be returned, or a numeric run identifier is produced. This run identifier is used to retrieve the XML log file (described in 5.4.1.1) containing all of the states seen by the mind during the run, as well as the actions it took and the score at every timestep.

### D.5.2 Stepping through a run

Once a run has completed and the XML log file produced, the web front-end provides an interface for examining and stepping through the states seen and corresponding actions taken at every timestep.

Looking at the example XML output for one timestep of a run in *ImageWorld*:

```
<asynchruncrun>
  <action value="1"/>
  <score value="3,2"/>
  <state value="6,7"/>
  <timestep value="5"/>
</asynchruncrun>
```

At the completion of a run, the entire XML document is made available for download. This can help in debugging some errors, but in general it is easier to work with the data if the web interface parses the XML and presents the important fields (action, score and state) separately.

Shown in figure D.3 is the presentation of the run at this timestep by the web front-end. The value of each attribute is extracted from the XML document by the web browser using asynchronous Javascript and XML (AJAX) methods, and presented by the interface in textual form, with the world state rendered visually if the world implements the required drawing method and the user requested a graphical run (for more detail on the graphical rendering system, see appendix E).

Beneath the rendered image are the controls for moving forward and backward in time, either by single-stepping, or by jumping directly to the first, last or any

## Analyse this run: Timesteps 1 to 21

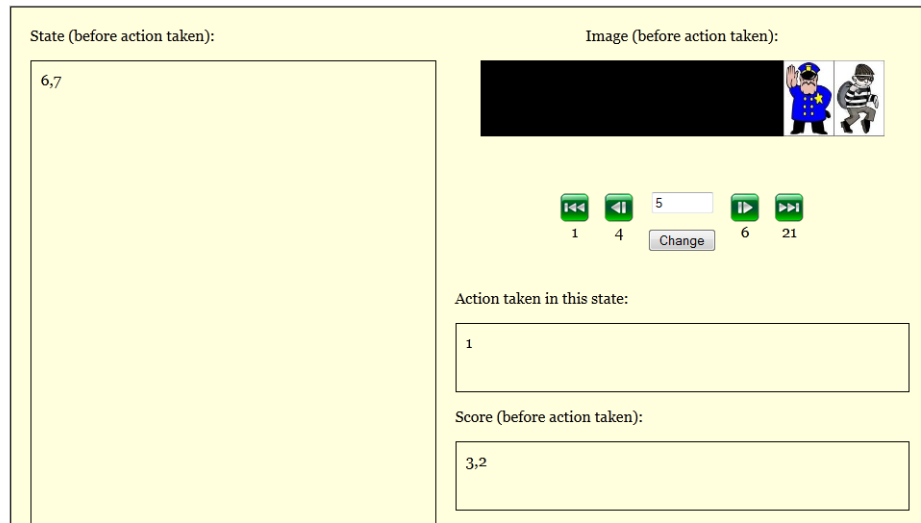


Figure D.3: A run in *ImageWorld*, showing the extracted values of the score and state at the fifth timestep in the world, and the action taken by the mind in response to that state. The state description “6,7” describes the position of the cop and robber agents, respectively, on a one-dimensional path. The action “1” in this world signifies “move right”.

arbitrary timestep.

## D.6 Conclusion

In the previous work (chapter 4), minds and worlds were envisaged as web services distributed on the Internet. Even if a mind and world were both hosted by the same server, the mode of interaction when carrying out a run was for the user to act as a middleman, receiving and forwarding on messages between the mind and world.

Chapter 5 discusses a switch in focus from these distributed web services toward an “islands” type architecture, with centralised mind and world servers which can start a run locally and issue updates to the remote user asynchronously. This allowed runs to be performed much more quickly, and opened up possibilities to automate important processes such as world and mind uploading and scoreboards for each world.

This chapter outlined the main features of the web-based interface to the World-Wide Mind server, which takes advantage of this switch in focus to provide a usable, efficient way of uploading and evaluating minds and worlds.

## Appendix E

# Hosting graphical worlds

### E.1 Introduction

An important facility for world designers and mind creators is the ability to generate images which represent the state of the world at any given moment. Without a graphical rendering of the world, mind writers are forced to make do with the textual descriptions of state provided by the world, which for some worlds may consist of an overwhelming volume of impenetrable numeric data.

In other worlds, the state visible to minds may only represent a partial snapshot of the true world state, and in some worlds may purposely contain false information, in an attempt to model environments where sensor noise is a real problem.

To illustrate both of these problems, consider a small fragment of the textual description of a single observed state at one timestep in the Tyrrell world (a description of the world can be found in appendix B):

```

...
perceivedAnimalCleanliness:0.2640972144091428;
nightProximity:0.5632035465923408;
animalVariance:0.0;
distanceFromDen:1.0;
foodPerceptionStimulus:[0.0,0.0,0.0,0.04733347679939349,0.0,0.0,0.0,0.0,0.0];
foodMemoryStimulus:[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0];
waterPerceptionStimulus:[0.0,0.37982594722763063,
                          0.0,0.0,0.0,0.0,0.0,0.0,0.5713010581724843];
waterMemoryStimulus:[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0];
shelterPerceptionStimulus:[0.0,0.0,0.0,0.0,0.0,0.0,0.8055555555555556,0.0,0.0];
matePerceptionStimulus:[0.0,0.0,0.0,0.0,0.0,0.0,0.3076923076923077,0.0,0.0];
denPerceptionStimulus:[0.0,0.0,0.0,0.0,0.0,0.0,0.46153846153846156,0.0,0.0];
...

```

This extract contains 11 out of a total of 35 sensory inputs, which includes arrays of numeric data. Some of the salient items, highlighted above in red, correspond to the features displayed in the visual representation depicted in figure 5.6 on page 96.

Note that many of the features in the bottom part of the image are not visible in the textual state description given above. Apart from making clear some of the most important features in the world, the visual rendering also shows features which are currently invisible to the mind because of the limited range of the senses.

In both of these cases, a well-crafted visual representation can be easier to work with, making the most important information about the running mind's situation in the world intuitively clear at a glance. This could serve as an important feature for debugging errors and trying to gain a better understanding of the world model and of the unexpected behaviours sometimes exhibited by minds.

## E.2 Graphical rendering in previous work

In the earlier work on the World-Wide Mind software platform (see chapter 4), the Tyrrell world was implemented with a built-in renderer which produced an image at each timestep which represents the current state of the world. This implementation was hardcoded, with no generic way in which worlds could provide graphics.

## E.3 A functional, generic graphical rendering system

In Monks' work [Monks, 2010], he added a method `getImage()` to the `World` class (the abstract base class which all worlds extend). The `getImage()` method returns, rather than one single image, a list which may contain one or more `BufferedImage` objects, to facilitate some worlds where multiple images are generated at every timestep.

### E.3.1 Multiple images per timestep

Recall that a *timestep* represents all of the activity in the world between the mind issuing an action and being ready to issue the next action. During this time, other simulated agents and phenomena may cause changes to the world state. There could be many of these “in-between” state updates, depending on the complexity of the simulated environment.

Generating more than one image per timestep may be useful, then, in worlds with multiple simulated opponents or entities, so that these “in-between” changes to the state can be viewed one at a time rather than interleaved all at once.

This is important, for example, if one of several simulated entities reverts or alters the result of an action performed by another entity earlier, but within the same timestep.

The default implementation of this method is simply to return null (in other words, to do nothing). This means that worlds which do not generate graphics do not require the `getImage()` method to be re-implemented by the designer.

If a world does wish to produce graphics, then it must construct and draw its own `BufferedImage` objects using the standard Java Advanced Windowing Toolkit (AWT) classes [Niemeyer and Peck, 1998], and return them in a new list in response to a `getImage()` method call.

### E.3.2 Optional image generation and cleanup

As one can imagine, there is a computational cost to the generation, storage and eventual transmission of these images, which will be generated once for each of potentially thousands of timesteps in a run. To avoid wastefully generating images when



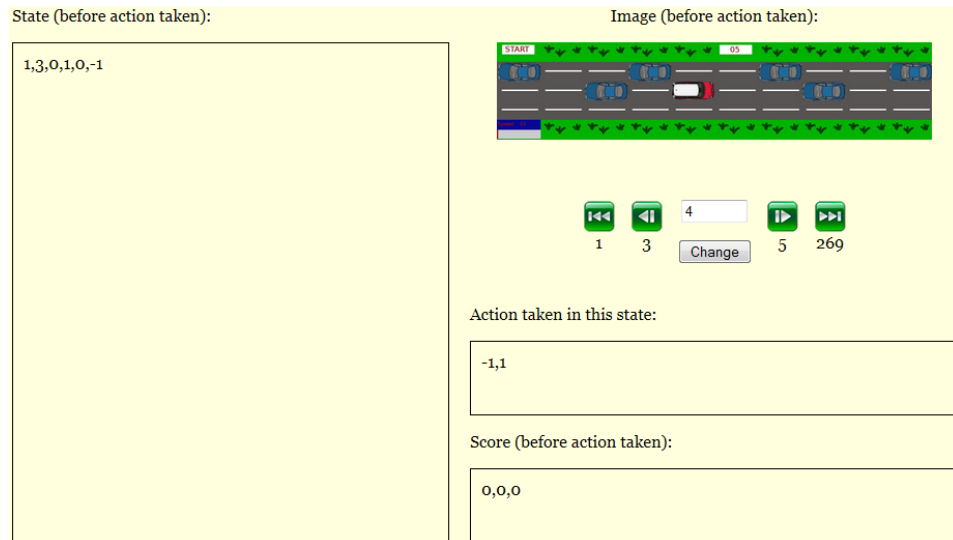


Figure E.1: A user-submitted world simply entitled “Speed” demonstrates a graphical rendering of the world state. The buttons beneath the image allow the user to step forward and backward between states and, if applicable to the world, between intermediate state representations within a single timestep.

none are desired by the user when performing a run, an `imagesDesired` parameter flag was added to the `RemoteWorld` class and to the run logger program.

As discussed in section F.4.2, the generation of large XML files to record the states seen and actions taken by minds can quickly consume a large amount of disk space, particularly under heavy load from multiple users. The situation is exacerbated by the creation of a large number of image files per run. For this reason, generated image files are deleted one hour after their creation.

# Appendix F

## W2M Server Usage

### F.1 Introduction

The World Wide Mind server-side system consists of front-end the (implemented with PHP and Javascript/Ajax) and back-end (Java) components. This appendix briefly describes the W2M back-end setup, the locations of the source code, the program and library JAR files and how to operate them.

### F.2 Overview

The components of the W2M server back-end are:

**runlogger.jar**, which is called by the web interface to initiate a run given appropriate parameters (the mind name, world name, URLs of the mind and world server machines and a flag indicating whether the user wishes graphics to be generated or not).

**w2mServer.jar**, which starts the back-end server daemon running (on port 8228 unless specified otherwise with the *-tcp\_port* command line option).

### F.3 Locations

#### F.3.1 Binaries

In the current configuration, *runlogger.jar* is placed in the */data/development/* directory and *w2mServer.jar* in the */data/development/w2mtest* directory. (There is

no particular reason for them to be in two different directories).

### F.3.2 Source code

The relevant source code is currently available via SVN on the server machine at <http://mbio-server.computing.dcu.ie/repos/w2mind> in projects named: w2mLib (the generic library code), w2mServer, and w2mRunLogger. (The *TyrrellWorld* projects are also hosted in this repository).

### F.3.3 Building from source

In Eclipse, a “jardesc” file is used to automate the process of constructing a JAR from files in the workspace. Right-clicking a JAR description file (such as “runlogger.jardesc” in the w2mRunLogger project) and selecting the “*Create JAR*” menu option will build an executable JAR file which can be placed in the appropriate directory on the server machine.

## F.4 Usage

### F.4.1 Controlling the back-end daemon

The W2M server and run logger are expected to run under a user account which allows access to mind and world files in the following directories, respectively:

- `/data/minds` (unless changed with the `-mind_path otherPath` command line option to w2mServer)
- `/data/worlds` (unless changed with the `-world_path otherPath` command line option to w2mServer)

The TCP listener port used by the server daemon is by default set to port 8228. This can be changed by passing the command line option `-tcp_port portNum` to w2mServer.

Runlogger is called directly by the httpd process when a run is started. W2M is controlled via the command `sudo /etc/init.d/w2m-server {stop/start}`. This script issues the following commands to start or stop the backend server, which inherits the permissions of the `w2m-daemon` user:

```

#!/bin/sh

case "$1" in
    start)
        su -w2m-daemon -c '/data/development/java/bin/java
-jar /data/development/w2mtest/w2mServer.jar &'
        echo "Starting W2M Server"
        ;;
    stop)
        W2M_PID='ps -aef|grep java|grep w2mServer|awk '{print $2}''
        kill -9 $W2M_PID
        echo "Killed W2M Server"
        ;;
    *)
        echo "Usage: $0 {start|stop}"
        exit 1
        ;;
esac

```

Minds and worlds are placed in the */data/minds* and */data/worlds* directories respectively, with the scoreboard entries being saved to an SQLite3 database at */data/scores.db*.

#### F.4.2 Cleaning up

When a run is executed by the run logger, an XML logfile is generated which contains a record of every state seen and every action taken by the mind in that run. These files can become quite large, since hundreds or thousands of timesteps may be executed per second - for example, a typical run in the Tyrrell world can take up to 10 megabytes of storage.

With multiple users testing their minds and competing with each other to achieve the highest ranks on the scoreboard, these accumulated size of these logs can add up quickly. And if the users ask that the world generate images representing the state at every timestep, there is a danger of running out of storage space in a short length

of time. If the available storage space is exhausted, then the server will come to a complete halt, preventing any more runs from being completed.

To mitigate this danger, a recurring job can be specified, to be executed periodically by the “cron” daemon. The following example cron job specifies that once every hour (*@hourly*), any files in the `/data/development/logged-runs` directory or subdirectories should be deleted, if they were last modified over an hour ago (*-type f -mmin +60*).

This ensures that stale images and logs do not accumulate more than an hour’s worth of files. However, it means that if a noteworthy run is observed by a user, it will not be preserved. However, it is possible for the user to save the XML log generated by the run logger, as well as the generated images, and perhaps more conveniently a movie file created by appending all of the images as individual frames.

```
@hourly find /data/development/logged-runs/  
↔      -type f -mmin +60  
↔      -exec rm {} 2>/dev/null \;
```

## Appendix G

# Selected worlds written by third parties

In late 2011 an assignment was set for undergraduate computer science students taking an artificial intelligence module. As part of the assignment, the students were required to build and demonstrate a problem world.

This appendix presents a brief description of a selection of those worlds, illustrating the range of problems that can be expressed as worlds, and the variety of graphics that can be used to explain the states and events which occur.

### G.1 World type

#### G.1.1 Models of puzzles

##### **Mazeworld**

*Author:* John Pendlebury

*URL:* <http://w2mind.computing.dcu.ie/sys/world.php?world=MazeWorld>

This world produces a randomly-generated two-dimensional maze to be solved during each run. Solving the maze requires that the player moves to an adjacent blank cell, in one of the four compass directions, until the cell containing a randomly-placed ruby is reached.

The mind is allowed to make 500 moves before the run is terminated. The state passed to the mind consists solely of a vector of four ordinals, symbolising the type of block sensed on that side: free, wall, goal, etc.

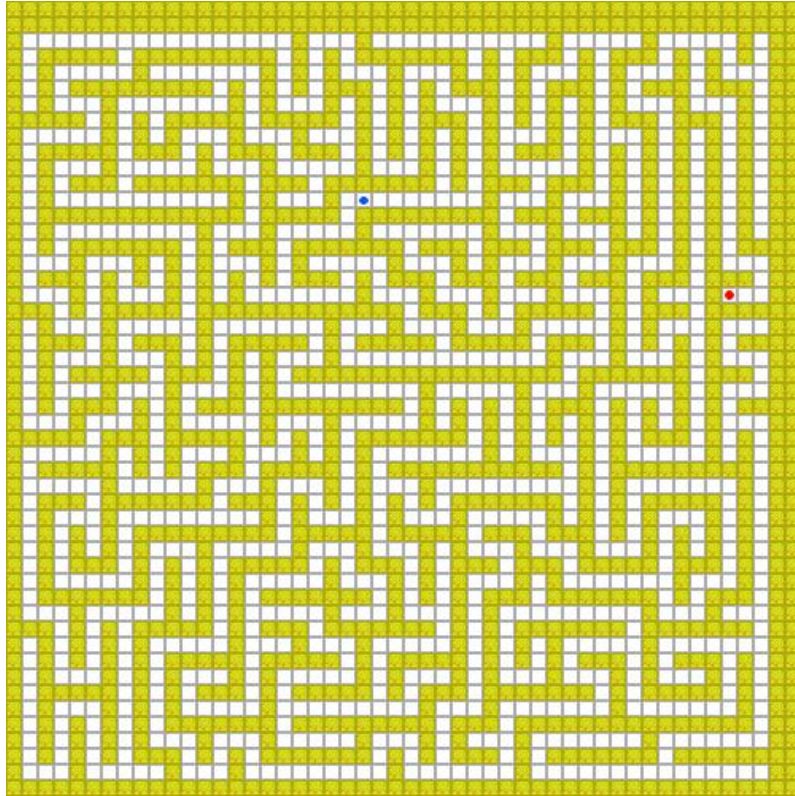


Figure G.1: An instance of MazeWorld, with the player (the blue dot in the upper middle) about to run out of time without having found the ruby (near the right edge).

A sample mind was also uploaded which implements a simple strategy of always following the left-hand wall. If the maze is simply connected – i.e. all walls are connected together or to the outer boundary – then the controller is guaranteed to eventually discover the exit.

If the maze is not simply connected, then the mind might encounter a loop and circle it forever.

## G.1.2 Models based on video games

### Minesweeper

*Author:* Shane Stacey

*URL:* <http://w2mind.computing.dcu.ie/sys/world.php?world=MinesweeperWorld>

The classic computer game *Minesweeper* was implemented on a two-dimensional grid, with the state encoded as a grid of integers, where -2 signifies a blank (unknown) cell, -1 signifies a cell previously flagged by the player, and values in the range 0-8 indicate the number of mines which surround that cell in the grid.

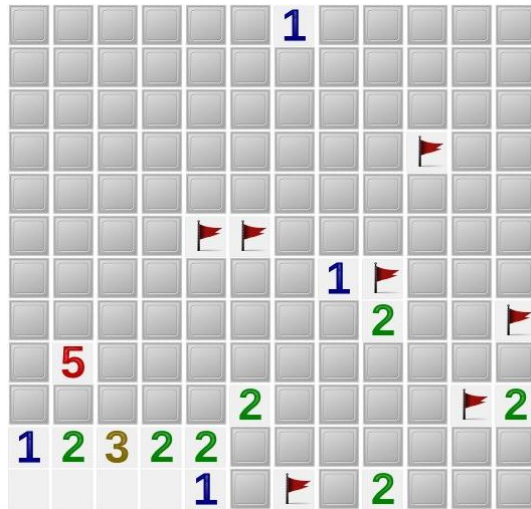


Figure G.2: The Minesweeper world, showing several mines flagged and detected in adjacent cells.

Actions consist of a string in the form  $x,y,type$  where  $x$  and  $y$  specify a position on the grid and  $type$  enumerates the following actions as integer values:

- flag cell as safe  $\rightarrow 0$ ,
- flag cell as a mine  $\rightarrow 1$ ,
- remove flag  $\rightarrow 2$ .

The score is derived by awarding or deducting points for correct or incorrect flagging of mines, and for finishing the game. If all the mines have been correctly flagged, then the game ends with a positive score bonus.

If the player flags as safe a cell which actually contains a mine, then the game is terminated immediately and a penalty is applied.

### Crimson Land

*Author:* Xiaodong Yu

*URL:* <http://w2mind.computing.dcu.ie/sys/world.php?world=CrimsonLand6>

Inspired by a PC video game of the same name, this world pits a hero against successive waves of zombies in a two-dimensional environment. The state provided to minds includes the  $x$  and  $y$  co-ordinates of the hero's position, as well as the type (each class of zombie has its own radius and speed) and position of every zombie on the playing field.





Figure G.3: The Crimsonland world, showing various types of enemy surrounding the player, as well as several dead zombies (marked by a red X).

The action submitted by a mind allows the hero to turn to a specified angle and take a step forward, and to fire one of three weapons – a handgun, shotgun or sniper rifle, each which different characteristics.

An example mind, *XDMind6*, was uploaded which moves randomly and fires the handgun at every timestep.

## Miner World

*Author:* Dmitri Lerko

*URL:* <http://w2mind.computing.dcu.ie/sys/world.php?world=MinerWorldUpdated>

In MinerWorld, the mind controls a drill which can move around a two-dimensional underground world in search of valuable minerals which can be sold later in exchange for fuel (which drops at a rate dependent on the types of actions performed), repairs and drill upgrades.

A graphical view of the world (shown in figure G.4) follows the drill as it descends and moves left or right.

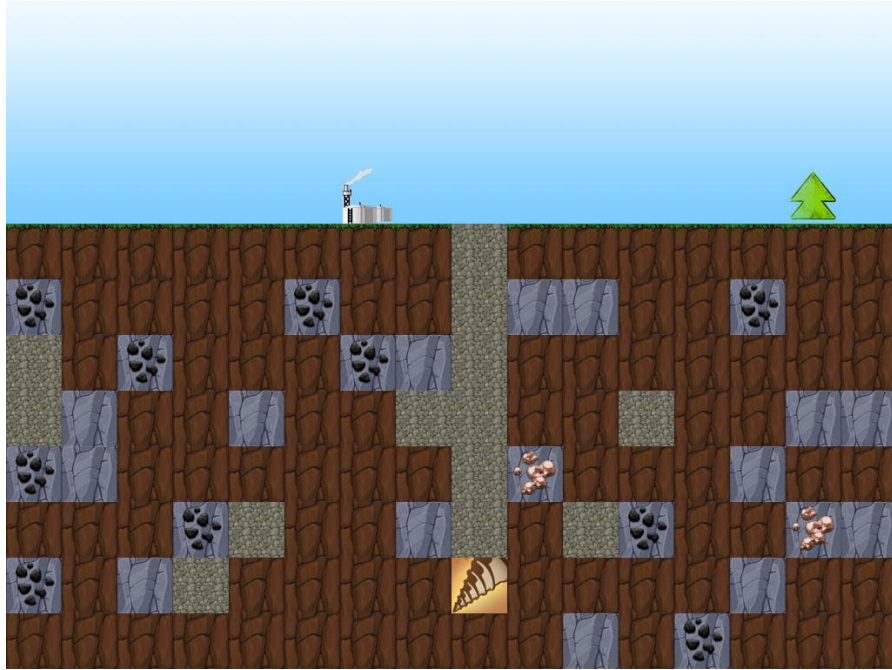


Figure G.4: A state in the procedurally-generated MinerWorld, close to the surface.

### G.1.3 Models of complex or real-world problems

#### SoccerWorld2

*Author:* Chris Courtney

*URL:* <http://w2mind.computing.dcu.ie/sys/world.php?world=SoccerWorld2>

This world simulates a restricted model of football where each team has two outfield players and one goalkeeper. The players can jog or sprint in one of eight cardinal directions, as well as perform a standing or sliding tackle, shoot at the goal, pass to a teammate or, interestingly, call for a pass from the teammate. Each of these actions can affect the player's stamina to varying degrees, and may prevent the player from carrying out other actions for up to three timesteps.

Among the information passed to the mind in each observed state are:

- the positions and orientation of all players,
- the ball position,
- which player currently has the ball,
- distance to the goal,
- player fatigue, and

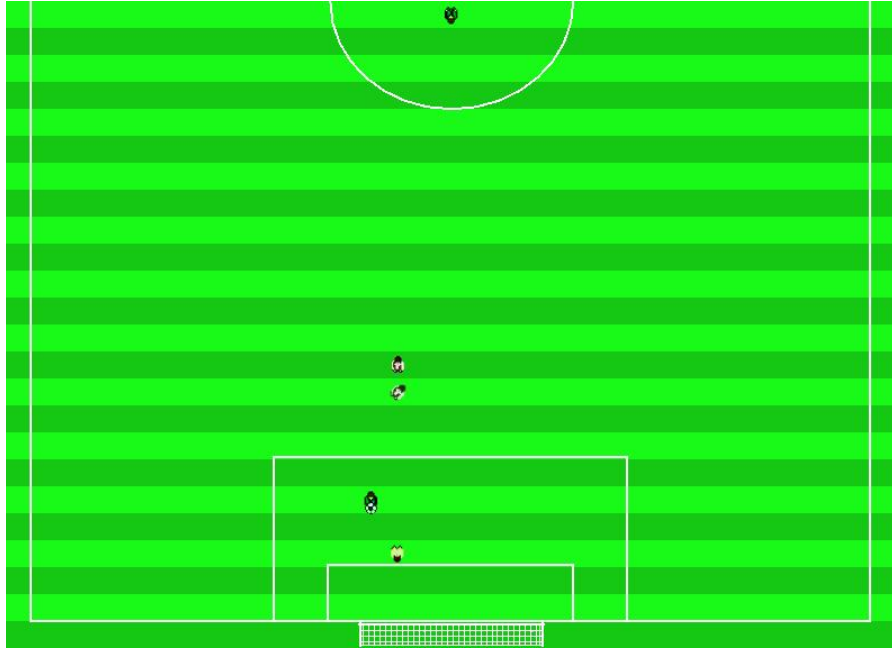


Figure G.5: A possible scoring situation in SoccerWorld2, where both opposing outfield players have failed to tackle the attacking player.

- a factor representing how much pressure the player is under if they attempt to shoot (this is derived from proximity to the opponents) and several other variables.

The score for a mind is constructed from five measurements:

1. the number of goals scored by the mind's team,
2. goals conceded,
3. goals scored by the mind-controlled player (the teammate is not controlled by the mind),
4. tackles won by the mind-controlled player, and
5. ball concessions (when the mind-controlled player is tackled and loses possession of the ball).

Chapters 6 and 7 describe methods to use this type of score (and state) information to construct a hybrid mind from many minds which constitute experts in individual elements of the score – for example, one mind might specialise in minimising the number of goals conceded, while another mind may specialise in maximising the number of tackles won.



Figure G.6: The player's ship navigates the grid, nearing three pawn orbs.

### 3D Shooter AI World

*Author:* Oisin St John Kelly

*URL:* <http://w2mind.computing.dcu.ie/sys/world.php?world=ShooterAI3D>

The 3D Shooter AI world (figure G.6) models a bounded grid terrain with a player ship controlled by the mind. The goal is to find and shoot pawns while avoiding aggressive hunter robots.

Additionally, to maximise performance in the world, the mind must also discover and collect health and ammo, represented graphically by the small red and blue cubes scattered around the grid.

This was the first submitted world to produce 3D graphics, opening up a range of interesting possibilities for world builders.

## G.2 Conclusion

A wide variety of worlds was submitted to the server by third-parties. Some of these worlds provide no graphical rendering of the states encountered, and others provide a graphical display of two- or three-dimensional scenes.

The range of worlds submitted to the server was quite varied, and the effort that was spent in making a graphical display for many of the worlds is encouraging.

# Appendix H

## Code Listings

### H.1 Introduction

This appendix contains any lengthy code examples referenced in the main text.

### H.2 A hybrid mind based on a condition list structure

```
1 import org.w2mind.net.Action;
2 import org.w2mind.net.RemoteMind;
3 import org.w2mind.net.RunError;
4 import org.w2mind.net.State;
5 import org.w2mind.tyrrell.TyrrellState;
6
7 public class Exp2Mind extends HybridExperiment {
8     RemoteMind mater, eater, drinker, survivor;
9     String materName, eaterName, drinkerName, survivorName;
10
11     public void setMaterName(String materName) {
12         this.materName = materName;
13     }
14
15     public void setEaterName(String eaterName) {
16         this.eaterName = eaterName;
```

```

17     }
18
19     public void setDrinkerName(String drinkerName) {
20         this.drinkerName = drinkerName;
21     }
22
23     public void setSurvivorName(String survivorName) {
24         this.survivorName = survivorName;
25     }
26
27     /**
28      * This method is called at the beginning of a run,
29      * so we initialise each of our subminds.
30      */
31     public void newrun() throws RunError {
32         mater = createAndPrepareSubmind(materName);
33         eater = createAndPrepareSubmind(eaterName);
34         drinker = createAndPrepareSubmind(drinkerName);
35         survivor = createAndPrepareSubmind(survivorName);
36     }
37
38     /**
39      * This method is called at the end of the run, so we
40      * clean up by calling the endrun() method in each of
41      * our subminds.
42      */
43     public void endrun() throws RunError {
44         try {
45             mater.endrun();
46             eater.endrun();
47             drinker.endrun();
48             survivor.endrun();
49         } catch (Exception e) {
50             e.printStackTrace();
51             System.err.println(this.getClass().getSimpleName())

```

```

52         + ":_Error_trying_to_end_the_run.");
53     }
54 }
55
56 /**
57  * This method is called on every timestep in the world,
58  * so we perform some simple threshold-based tests to
59  * decide which of our subminds should take control in
60  * response to this state.
61  */
62 public Action getaction(State state) throws RunError {
63     try {
64         final TyrrellState ts = new TyrrellState(state);
65         final double mateNearby =
66             sumVec(ts.getMatePerceptionStimulus());
67
68         if (mateNearby > 0)
69             return mater.getaction(state);
70
71         final double maxHunger = 0.75;
72         if (ts.getPerceivedWaterShortage() > maxHunger)
73             return drinker.getaction(state);
74
75         if (ts.getPerceivedFatShortage() > maxHunger ||
76             ts.getPerceivedProteinShortage() > maxHunger ||
77             ts.getPerceivedCarbohydrateShortage() > maxHunger)
78         {
79             return eater.getaction(state);
80         }
81
82         if (ts.getAnimalHealth() < 0.8 ||
83             ts.getNightProximity() > 0.6)
84         {
85             return survivor.getaction(state);
86         } else {

```



```
87     return mater.getAction(state);
88     }
89     } catch (Exception e) {
90         System.err.println(this.getClass().getSimpleName()
91             + ": Failed to get an action.");
92         e.printStackTrace();
93     }
94     return null;
95 }
96 }
```