

Proactive Persistent Agents:
Using Situational Intelligence to Create Support Characters in
Character-Centric Computer Games

A thesis submitted to the
University of Dublin, Trinity College
for the degree of
Doctor of Philosophy

Brian Mac Namee
Department of Computer Science,
University of Dublin,
Trinity College.

January, 2004

Declaration

The work presented in this thesis is, except where otherwise stated, entirely that of the author and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Brian Mac Namee

January, 2004.

Permission to Lend or Copy

I agree that the library of the University of Dublin, Trinity College, has my permission to lend or copy this thesis.

Signed:

Brian Mac Namee

January, 2004.

Acknowledgements

Firstly, I would like to thank Prof. Pádraig Cunningham for all of his help and guidance throughout my academic endeavours. He's always been there to rein in the *crazy schemes*, and this work could never have been done without him.

Secondly, I would like to express my gratitude to all of the members of the MLG (both past and present) - an exceptional group of people. In particular I would like to mention Michael and Kevin, many hours of discussion with whom were instrumental in forming the ideas behind this thesis.

The collaboration with the ISG was one of the best things that could have happened to this work. I would like to thank Simon, Chris and Carol for enabling this.

My parents, Liam and Roisín, have been incredibly supportive throughout my (long!) academic career, and in everything else I do. I can't say how much I owe them. Also, I have to thank my two brothers, Liam (great preface) and Colin (meticulous proof reading - nice!!), for their support.

Finally, I would like to thank Aoife. She has had to put up with an awful lot while I finished this, and I couldn't have done it without her. Thank you so much.

Abstract

Throughout the 1990s computer game development was dominated by the improvement of game graphics. However, graphical excellence is now the norm rather than the exception, and so other technologies are coming to the fore as developers strive to make their games stand out in an ever more crowded market. Sophisticated artificial intelligence (AI) is one such technology which is now receiving a large share of game development effort as game designers attempt to create more believable computer controlled characters.

Influenced by the notion of *situational intelligence*, this work presents the *proactive persistent agent* (PPA) architecture, an intelligent agent architecture that has been developed to drive the behaviours of *non-player support characters* in *character-centric* computer games. The key features of the architecture are that:

- PPA based characters display believable behaviour across a diverse range of situations, through the use of *role passing* and *fuzzy cognitive maps*
- PPA based characters are capable of sophisticated social behaviours based on psychological models of personality, mood and interpersonal relationships
- The architecture performs in real-time on machines of modest specifications
- It is relatively easy for non-programmers to author character behaviours using the architecture

The PPA architecture has been used in two demonstration applications (an adventure game and a system for animating sophisticated virtual humans) which will also be presented. These applications have been used to perform a rigorous evaluation of the architecture, which will also be described. Evaluation is notoriously difficult in systems used to simulate the behaviour of virtual characters, as the key notion of *believability* is troublingly subjective. However, in spite of the difficulties involved, an evaluation scheme has been developed and used. The evaluation shows that the PPA architecture is ideal for the control of non-player support characters in character-centric games.

Table Of Contents

Declaration	ii
Permission to Lend or Copy	iii
Acknowledgements	iv
Abstract	v
Table Of Contents	vi
List of Figures	x
List of Tables	xv
Related Publications	xvi
1 Thesis Introduction	1
1.1 Introduction	2
1.2 Thesis Contributions	3
1.3 Thesis Structure	4
2 Game-AI	6
2.1 Current Game Genres	7
2.1.1 Action Games	7
2.1.2 Adventure Games	9
2.1.3 Role Playing Games	11
2.1.4 Strategy Games	13
2.1.5 God Games	14
2.1.6 Sports Games	16
2.1.7 Defying Categorization	17
2.1.8 What About Deep Blue?	18
2.2 Roles for Game-AI	20
2.3 Game-AI – the Developer’s Perspective	22
2.3.1 The Current State of the Art	24
2.3.2 Path-Finding	24
2.3.3 Finite State Machines	25
2.3.4 Expert Systems	26
2.3.5 Artificial Life	31
2.3.6 Learning Systems	32
2.3.7 Cheating the Player	34
2.3.8 Game-AI Systems Development Kits	35
2.3.9 The MOD Community	35
	vi

2.4	Game-AI in Academia	36
2.4.1	Notable Academic Game-AI Projects	36
2.4.2	Problems Facing Game-AI in Academia	38
2.5	Summary & Conclusions	39
3	Intelligent Agents	41
3.1	What is an Intelligent Agent?	42
3.2	Agent Implementations	43
3.2.1	Reactive Agents	43
3.2.2	Deliberative Agents	44
3.2.3	Hybrid-Agents	45
3.3	Intelligent Agents as Virtual Humans	45
3.3.1	What is Required by Intelligent Agents as Virtual Humans?	45
3.3.2	The Spectrum of Agents	48
3.4	Virtual Fidelity	48
3.5	Requirements for Intelligent Agents in Computer Games	49
3.6	Situational Intelligence and the Proposed Agent Architecture	50
4	The Proactive Persistent Agent Architecture	52
4.1	Proactive Persistent Agents	53
4.2	The PPA Architecture	55
4.3	The Schedule Unit	56
4.4	The Role-Passing Unit	57
4.4.1	Motivations for Role-Passing	58
4.4.2	Fuzzy Cognitive Maps	59
4.4.3	The PPA Fuzzy Cognitive Control System	62
4.4.4	The Mechanics of Role-Passing	67
4.4.5	Benefits of Role-Passing	71
4.5	The μ-SIC System	72
4.5.1	Psychological Models of NPCs' Personae	73
4.5.2	Possible Interactions	76
4.5.3	Implementing the μ -SIC System	77
4.5.4	Benefits of the μ -SIC System	82
4.6	Knowledge Base	82
4.6.1	Events & Character Sighting Records	82
4.6.2	Facts	84
4.6.3	Benefits of the Knowledge Base	84
4.7	Summary & Conclusions	84
5	Games-As-Data and the Implementation of the PPA Architecture	86
5.1	Games-As-Data	87
5.2	Implementing the PPA Architecture	87
5.3	Languages and Tools	89

5.4	Smart Objects	90
5.5	Level-of-Detail AI	91
5.6	Berlin-End-Game	93
5.7	PPAs in the ALOHA System	94
5.8	Conclusions	95
6	Evaluation	97
6.1	Evaluation Problems	98
6.2	Evaluation Scheme	98
6.3	Believability	100
6.3.1	The PPA Believability Experiment	101
6.3.2	Experiment Results	104
6.4	Social Abilities	110
6.5	Application Domain	114
6.5.1	The Berlin-End-Game Evaluation Experiment	114
6.5.2	Experiment Results	115
6.6	Computational Issues	117
6.7	Production	121
6.8	Conclusions	121
7	Thesis Summary, Conclusions & Suggestions for Future Work	122
7.1	Thesis Summary & Conclusions	123
7.2	Suggestions for Future Work	126
7.2.1	Improvements to the Schedule Unit	127
7.2.2	Further Use of FCMs	127
7.2.3	Further Use of Psychological Models	128
7.2.4	Improvement of the μ -SIC system	128
7.2.5	Further Evaluation	129
7.2.6	Addition of a Goal Based Planning Unit	129
7.2.7	Implementation Optimization	130
7.2.8	Further Use of Level-of-Detail AI	130
7.2.9	Development of GUI based Design Tools	130
7.3	The Last Word	131
	References	132
	Game References	139
	Appendices	141
	Appendix A: Fuzzy Logic & Additive Fuzzy Systems	142
	Appendix B: The PPA Believability Experiment	147
B.1	Questionnaire	147
B.2	Experiment Results	148
	Appendix C: The Berlin-End-Game Evaluation Experiment Questionnaire	150
		viii

Appendix D: Implementation of the PPA Architecture	151
D.1 Implementing the Schedule Unit	152
D.2 Implementing the Role-Passing Unit	153
D.3 Implementing the Social Unit	156
D.4 Implementing the Memory Unit	157
D.5 Ancillary Technologies	157
Appendix E: Implementation of Berlin-End-Game	163
E.1 Implementing the Game World	163
E.2 World Designer Tool	164
E.3 The Player	165
Appendix F: Integration of the PPA Architecture into the ALOHA System	166
Appendix G: A Sample XML Character Listing for a Character in Berlin-End-Game	168

List of Figures

Figure 2-1: Fast paced action in (clockwise from the top left) Space Invaders^{G-2}, Doom^{G-4}, Half-Life 2^{G-8} and Grand Theft Auto Vice City^{G-7} 8

Figure 2-2: Adventure afoot in (clockwise from top left) Gabriel Knight – Sins of the Fathers^{G-16}, The Curse of Monkey Island^{G-15}, Indiana Jones and the Emperor’s Tomb^{G-20} and Tomb Raider – The Angel of Darkness^{G-19} 10

Figure 2-3: Role playing in (clockwise from top left) Baldur's Gate II: the Shadows of Amn^{G-21}, Neverwinter Nights: Hordes of the Underdark^{G-24}, Final Fantasy XII^{G-23} and Star Wars Knights of the Old Republic^{G-25} 12

Figure 2-4: Battles rage in (clockwise from top left) Age of Empires II: the Age of Kings^{G-28}, Command & Conquer Generals^{G-31}, Civilisation III^{G-29} and Warcraft III: Reign of Chaos^{G-32} 14

Figure 2-5: Example god games (clockwise from top left) Populous: The Beginning^{G-33}, Black & White^{G-34}, Republic the Revolution^{G-36} and Sim City 4^{G-35} 15

Figure 2-6: Highly realistic sports action in (clockwise from top left) Sega Bass Fishing^{G-38}, FIFA Soccer 2004^{G-37}, Colin McRae Rally 04^{G-52} and ESPN International Track & Field^{G-39} 16

Figure 2-7 Defying categorization, Diablo II^{G-41} and the Sims^{G-42} 18

Figure 2-8: An illustration of how a branching search (many of the paths of which have been omitted for clarity) would proceed from midway through a game of Xs and Os to a winning position for the player using X. The darker arrows indicate the paths that lead to a win for X 19

Figure 2-9: An illustration of a Go-Moku board 20

Figure 2-10: An illustration of the path-finding process in the Berlin-End-Game application which will be discussed in section 5.6 25

Figure 2-11: A simple finite state machine for a soldier NPC in a typical action game 26

Figure 2-12: A sample script from the combat behaviour of a warrior character in the RPG Baldur’s Gate 27

Figure 2-13: An expert system decision tree which determines whether a person should eat in a restaurant based on a number of properties 29

Figure 2-14: A simple Bayesian network used to allow a guard in a game such as Thief: The Dark Project ^{G-9} determine whether observations indicate the presence of an intruder, or harmless rats	30
Figure 2-15: The three rules used by Reynolds' original Boids system to simulate flocking behaviours.....	31
Figure 4-1: A toy game world containing four rooms - A, B, C and D - inhabited by four characters - Player, NPC-1, NPC-2 and NPC-3.....	54
Figure 4-2: A schematic of the proactive persistent agent architecture.....	55
Figure 4-3: A sample schedule used by a PPA based character	57
Figure 4-4: An FCM which shows how bad weather can affect the speed at which one drives on a California highway.....	60
Figure 4-5: Illustrations of the upward and downward motivation response curves used by a basic hunger motivation.....	63
Figure 4-6: A PPAFCM used to control a very basic agent	65
Figure 4-7: An illustration of a PPAFCM used to control the behaviour of a barman character.....	66
Figure 4-8: An illustration of the complete PPAFCM for an NPC that has assumed the role of a barman. For the purposes of clarity only concept nodes have been included in this illustration.....	68
Figure 4-9: An illustration of Maslow's Hierarchy of Needs.....	69
Figure 4-10: An illustration of the complete PPAFCM for an agent that has assumed the cinema-patron role. For clarity the non-concept nodes of the basic PPAFCM have been omitted.....	71
Figure 4-11: The Eysenck personality model which measures personality across the Introvert-Extrovert and Neuroticism-Stability axes. The positions of a number of personality types are shown.....	74
Figure 4-12: The Lang mood model which plots mood according to valance and arousal. Subjects reactions to different pictures are also shown	75
Figure 4-13: The relationship model used which plots a character's relationship to another character.....	76

Figure 4-14: The architecture of an artificial neuron.....	78
Figure 4-15: An illustration of the basic structure of an MLP. The figure shows an input layer, one hidden layer and an output layer. Networks can have any number of hidden layers.....	79
Figure 4-16: The structure of the ANN used within the μ -SIC system along, with a sample query case and result.....	80
Figure 4-17: The data structures used to allow NPCs store events in their memories	83
Figure 5-1: An schematic of a PPA character.....	88
Figure 5-2: How the μ -SIC system is incorporated into a virtual world	88
Figure 5-3: A screenshot of the world designer application developed to create virtual worlds for use in Berlin-End-Game.....	90
Figure 5-4: An illustration of a virtual model of a potted plant at various levels of detail .	92
Figure 5-5: An illustration of a path planned across a game location at (A) a low level of detail and (B) a high level of detail.....	92
Figure 5-6: A selection of screenshots from Berlin-End-Game, a game implemented using the PPA architecture. The screenshots show a hotel, a cinema, Checkpoint Charlie and a restaurant.....	93
Figure 5-7: A screenshot of the questioning system used in Berlin-End-Game.....	94
Figure 5-8: Screenshots of the simulation created using the ALOHA system and the PPA architecture.....	95
Figure 6-1: A screenshot of the bar environment used within the believability experiment	102
Figure 6-2: The full PPAFCM used for bar customer characters in the PPA believability experiment	103
Figure 6-3:A graph of the percentage of subjects who selected the simulation populated by proactive persistent agents as the most believable, against the percentage who selected the random agent populated simulation	105
Figure 6-4: A graph of subjects' responses to the statement "agents appear to behave under their own volition" for both the simulation using PPAs and that using random agents	107

Figure 6-5: A graph of subjects' responses to the statement "agents' behaviours appear to be goal driven" for both the simulation using PPAs and that using random agents..	108
Figure 6-6: A graph of subjects' responses to the statement "agents' behaviours appear to possess a random component" for both the simulation using PPAs and that using random agents	109
Figure 6-7: A range of screen-shots showing how some of the interactions possible within the μ -SIC system can arise	112
Figure 6-8: Graphs illustrating the responses of subjects in the Berlin-End-Game evaluation experiment.....	117
Figure 6-9: A graph of the memory required to simulate an increasing number of PPA based characters	120
Figure 7-1: Borrowed from [Parenthoën et al, 2001], an illustration of a low-level FCM for use to control a sheep dog as it tries to bring back a sheep	128
Figure 7-2: A schematic of an extended version of the PPA architecture which includes a goal-based planning unit.....	130
Figure A-7-3: An illustration of membership for the cool set as both a fuzzy and non-fuzzy set.....	143
Figure A-7-4: An illustration of how a set of fuzzy patches can be used to approximate a continuous function. (A) shows how a small set of large patches crudely approximates a function, while (B) shows how a larger set of small patches can more accurately approximate the same function.....	144
Figure A-7-5: An illustration of a set of if-then rules covering a function in fuzzy patches. A single rule (if X is A4 then Y is B4) has been highlighted	145
Figure A-7-6: An illustration of the fuzzy additive system used to control the speed of the fan in an air conditioning unit in order to maintain a constant room temperature	146
Figure D-7-7: An illustration of the division of the implementation of the PPA architecture into a Character and Brain class	151
Figure D-7-8: A portion of the class hierarchy used to implement the different PPAFCM objects used within the FCM class	154
Figure D-7-9: A portion of the class hierarchy used to implement behaviours within the role passing unit of the PPA architecture implementation	155

Figure D-7-10: An illustration of the simplest form of the A* based path-finding system used within the implementation of the PPA architecture. Crossable cells are shown in white, while uncrossable cells are shown as black. The green cell shows the starting position for the search, while the red cell shows the destination. The resultant path is shown in blue	158
Figure D-7-11: A more advanced version of the path-finding system in which a third path value (shown in grey) is used	159
Figure D-7-12: The final version of the path-finding system in which the virtual world is divided into segments of equal path values. The resultant path across the selected segments (shown in grey) is shown as the thick blue line	159
Figure D-7-13: An illustration of a smart bar object, showing its user slots and the sequence of events which characters must perform in using the object	161
Figure E-7-14: A schematic of the virtual world in which Berlin-End-Game was set.....	163
Figure E-7-15: A portion of the area hierarchy used within Berlin-End-Game. Parent areas contain all child areas	164
Figure E-16: A screenshot of the world designer tool created in order to aid the design of worlds used within the Berlin-End-Game application.....	165

List of Tables

Table 3-1: The unique set of requirements for support character NPCs in character-centric computer games, and how existing systems compare in terms of which requirements are fulfilled..... 50

Table 4-1: How characters’ positions in the toy game world change over time..... 54

Table 4-2: A sample script for a restaurant situation..... 59

Table 4-3: The interactions possible within the μ -SIC system, along with the effects they have upon the target character 76

Table 4-4: A confusion matrix showing the results of testing the MLP used in the μ -SIC system on the hand crafted data set 81

Table 6-1: The categories into which the evaluation of the PPA system has been divided and criteria for their success 99

Table 6-2: The results of an ANOVA performed to test the hypothesis that more subjects agreed with the statement that *agents appeared to act under their own volition* for the PPA based simulation, than for the simulation using random agents..... 108

Table 6-3: The results of an ANOVA performed to test the hypothesis that more subjects agreed with the statement that *agents’ behaviours appeared be goal driven* for the PPA based simulation, than for the simulation using random agents..... 109

Table 6-4: The results of an ANOVA performed to test the hypothesis that more subjects agreed with the statement that *agents’ behaviours appeared be possess a random component* for the random agent based simulation, than for the simulation using PPA based agents 110

Table 6-5: The results of the Berlin-End-Game evaluation experiment..... 116

Table 6-6: An evaluation of the processing requirements of agents created using the PPA architecture. Also shown are the theoretical maximum number of NPCs possible, if different percentages of the available processing power are used..... 119

Table 6-7: The amount of memory required to load a cast of a given number of characters 119

Table B-7-1: Subjects’ responses to questions 1 and 3 of the PPA believability experiment 148

Related Publications

“Simulating Virtual Humans Across Diverse Situations”, B. Mac Namee, S. Dobbyn, P. Cunningham & C. O’Sullivan, In *Proceedings of Intelligent Virtual Agents ’03*, pp 159-63. (2003)

“Enhancing Non Player Characters in Computer Games using Psychological Models”, B. Mac Namee & P. Cunningham, In *European Research Consortium for Informatics and Mathematics News*, Number 53. (2003)

“Creating Socially Interactive Non Player Characters: The μ -SIC System”, B. Mac Namee & P. Cunningham, *International Journal of Intelligent Games & Simulations*, Vol.2 No.1. (2003)

“Smart Objects for Attentive Agents”, C. Peters, S. Dobbyn, B. Mac Namee & C. O’Sullivan. *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, Plzen, Czech Republic. (2003)

“The μ -SIC System: A Connectionist Driven Simulation of Socially Interactive Agents”, B. Mac Namee & P. Cunningham, *Proceedings of Game-On 2002 the 3rd International Conference on Intelligent Games and Simulation*, pp. 129 – 133. (2002)

“Levels of Detail for Crowds and Groups”, C. O’Sullivan, J. Cassell, H. Vilhjálmsson, J. Dingliana, S. Dobbyn, B. Mac Namee, C. Peters & T. Giang, *Computer Graphics Forum*, 21(4) pp 733-742. (2002)

“Men Behaving Appropriately: Applying the Role Passing Technique to the ALOHA System”, B. Mac Namee, S. Dobbyn, P. Cunningham & C. O’Sullivan, In *Proceedings of Artificial Intelligence and the Simulation of Behaviour ’02*. (2002)

“Proposal for an Agent Architecture for Proactive Persistent Non Player Characters”, B. Mac Namee & P. Cunningham, *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*, pp. 221 – 232. (2001)

“Research Directions for AI in Computer Games”, C. Fairclough, M. Fagan, B. Mac Namee & P. Cunningham, *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*, pp. 333 – 344. (2001)

1 Thesis Introduction

This chapter will briefly introduce the concerns of this work, present its key contributions and outline the remainder of the thesis.

1.1 Introduction

One of 2002's most high profile computer games, Metal Gear Solid 2: Sons of Liberty^{G-17}, took a 70-strong team more than two years to develop, at a total cost of something in the region of ten million euro. The development team included talented programmers, scores of artists and animators, physicists, artificial intelligence experts, sound engineers, musicians, and voice actors, and by the end of 2002 the game had sold over 2.5 million copies worldwide. This example is given to show that computer game development is now a serious business, and not the unprofessional hobbyist endeavour that many think it to be. In fact the global games industry is now financially larger than the movie industry, and all indications suggest that this trend will continue [Poole, 2002].

Throughout the 1990s game developers strove to outdo one another in terms of the realism of their games' graphics, and this became the ultimate distinction between a game's commercial success and failure. However, as evidenced by the visually stunning virtual worlds in which games are now set, graphical excellence is now the norm rather than the exception, and is no longer enough to distinguish one game from another [Rubin, 2003]. This is not to say that there are no longer important and extremely difficult problems left to solve in the graphics space, but rather that solving these problems will lead to incremental improvements in the realism of game worlds, rather than the massive improvements seen previously.

For a brief period, realistic physics simulation was touted as the new leveller in game development. However, the release of a number of physics-based middleware solutions has led to its fast becoming a reality. This has been widely showcased in pre-release videos of the much anticipated Half-Life 2^{G-8}, which will be released later this year. And so, AI has come to the fore as the major challenge in game development.

The biggest challenge for AI in games (referred to as *game-AI*) lies in the creation of *Non-Player Characters* (NPCs), the computer controlled characters that populate a game's virtual world. Currently, the AI techniques used for this purpose in modern games are relatively crude, mainly due to the fact that AI has been largely overlooked by game developers in their fanatical pursuit of graphics technology. However, a number of recent, and hugely successful, titles (such as the Sims^{G-42} and Black & White^{G-34}) have shown that the inclusion of sophisticated AI can lend greatly to the success of a game.

This thesis will discuss an intelligent agent architecture which has been developed to drive the behaviour of NPCs in modern computer games. An attempt to develop a system that could be used to drive the behaviours of characters in all of the myriad different kinds of games currently available would not be realistic, and so this work focuses specifically on developing *support characters* for *character-centric* games. Support characters take background roles in games, playing parts such as shop-keepers, barmen and police patrol men, but are crucial to the creation of virtual game worlds in which players can become deeply immersed. Character-centric games are games in which game-play proceeds at a relatively slow pace, and focuses more upon character interactions, than fast-paced action. The fact that players must form lasting relationships with NPCs in character-centric games, also makes them uniquely challenging for the application of AI techniques.

The key notion driving the development of the architecture described by this work is that of *situational intelligence*. This idea suggests that, rather than being initially designed to cope with any situation which might arise over the course of a game, agents need only possess enough sophistication to allow them behave believably in their current situation. Over the course of a game, as different situations arise, an agent's internal composition can be adjusted to allow it cope with this. This agent architecture will allow the creation of agents that can behave believably across a range of diverse situations and perform sophisticated social interactions. Along with this, the architecture will be implemented in such a way as to make it suitable for practical use in modern computer games.

1.2 Thesis Contributions

This thesis makes a number of key contributions to the area of game-AI. These are as follows:

- The development of an intelligent agent architecture specifically designed for the creation of computer controlled support characters in character-centric games.
- The use of situational intelligence in games, which is unique to this work.
- The development of a system, based upon fuzzy cognitive maps, which can be used to drive the behaviour of characters in a simulation.
- The development of the role-passing process which allows NPCs behave believably in a wide range of diverse situations within the same simulation.

- The development of the μ -SIC system which uses psychological models to simulate sophisticated social interactions amongst NPCs.
- The use of an evaluation scheme, explicitly for the evaluation of intelligent agent systems used in games, and more generally in the simulation of virtual humans.

1.3 Thesis Structure

Chapter 2 opens this thesis with an overview of the field of game-AI. This begins by discussing computer games in general, in order to give an introduction to the subject for those who are unfamiliar with it, and also to serve as a grounding for the discussions in the remainder of the chapter. These later discussions will touch upon the problems to which AI techniques can be applied in modern games, some of the techniques which game developers use to this end, and the current state of the art in game-AI - both commercially and in academia.

Chapter 3 discusses the area of intelligent agents, an AI technology which strongly matches the requirements of modern games. After a general discussion of the area, this chapter will focus upon how intelligent agent techniques have been used in the simulation of virtual humans, a more heavily researched area than game-AI. Towards the end of this chapter, the requirements for an intelligent agent system for use in character-centric games will also be outlined.

Chapter 4 is where the major contributions of the thesis are discussed. The chapter describes the *proactive persistent agent* architecture which has been developed for the creation of support characters in character-centric computer games. Along with a discussion of the architecture, the various techniques which it uses (in particular fuzzy logic and neural networks) will be discussed, as will the notion of situational intelligence.

Chapter 5 discusses *games-as-data*, a design method used in game development, and explains how this approach has been taken in the implementation of the PPA architecture. This section also describes two applications in which the PPA architecture has been used.

Chapter 6 describes how the PPA architecture has been evaluated. The opening section of this chapter focuses on the evaluation scheme used, while the remainder of the chapter describes the evaluation performed, and presents its results.

Thesis Introduction

The seventh, and final, chapter of this thesis summarises what has come before, and draws conclusions on the ideas presented. It also suggests a number of promising areas for furthering the work described here.

2 Game-AI

This chapter will serve as an introduction to the field of game-AI. Before discussing the major AI techniques currently being used in game development, section 2.2 will enumerate the key genres into which games can be categorised. This will give readers unfamiliar with computer games a flavour of the field, and serve as a platform for the discussions given in the remainder of the chapter.

From this discussion of game genres, a list of the roles in which AI can be applied in games will emerge, and these will be discussed in section 2.3. Sections 2.4 and 2.5 will examine the major techniques currently being used, firstly by commercial game developers, and latterly in academia.

Finally, the focus of this work, support characters in character-centric games, will be discussed.

2.1 Current Game Genres

On the PC alone, 2003 saw the release of over 400 different computer games¹. Much like in the music and film industries, a small number of descriptive game genres are used in order to make sense of the large number of available titles. Although no division can hope to perfectly capture the nuances of all available games, arranging them by genre is useful in that it simplifies any discussion of the subject. This section (which loosely follows a similar discussion given in [Laird & van Lent, 2000]) will describe the most important of these genres - mentioning some of the key titles, and musing upon some of the roles for AI in these games. It will serve as both an introduction to the world of modern computer games, and a platform for the discussion of the application of AI techniques to follow.

2.1.1 Action Games

“Now, with aliens coming through the wall, a military death squad on the rampage and your panicking colleagues at the top of the endangered species list, you need to scramble to stay alive. Where do you go? Who can you trust? Can you survive?”

Half-Life^{G-1}

Although the challenge posed has not changed greatly since the 1978 release of Taito’s classic Space Invaders^{G-2}, *action games* have become by far the most popular game genre on the market today. Game-play consists of fast paced action, in which the player must defeat waves of demented foes, typically (for increasingly bizarre reasons) bent upon global destruction. The player’s part in action games can vary wildly, stretching from the pilot of a small craft lost in outer-space, to that of a bemused scientist attempting to escape from a top secret government experiment gone horribly wrong. However, the high adrenaline action is ever present. Figure 2-1 shows screenshots of some typical examples of the action game genre.

In recent years this genre has been dominated by the *first person shooter* (FPS). In an FPS a player controls a single character from a first person perspective, giving a strong sense of immersion into a virtual game world. Typically, players can move relatively freely around a game environment, interacting with objects such as doors, the ubiquitous crate and vast

¹ This figure is based upon the number of titles reviewed in 2003 at www.gamespot.com, a leading games web site.

arrays of weapons; and engaging with a range of opponents. Released in 1992, id Software's *Wolfenstein 3D*^{G-3} is recognised as the first FPS. This was quickly followed by the ground-breaking *Doom*^{G-4} (shown at the top right of figure 2-1) which gave players a sense of immersion into a game world which had not been seen before, and served as a benchmark for any action games released subsequently. Today the FPS torch is carried by titles such as *Half-Life*^{G-1}, *Deus Ex*^{G-5} and *Unreal Tournament*^{G-6}.



Figure 2-1: Fast paced action in (clockwise from the top left) *Space Invaders*^{G-2}, *Doom*^{G-4}, *Half-Life 2*^{G-8} and *Grand Theft Auto Vice City*^{G-7}

The AI related challenges in the development of action games come mainly from the control of opponents. In early attempts, enemies ran straight towards the player with all guns blazing, providing little challenge beyond target practice. These hapless foes were superseded by opponents that followed scripted behaviours, for example popping their heads out from behind a barrel to take a few shots at the player, before quickly returning to cover. Although this was certainly an improvement, players could simply memorise these patterns and easily learn to beat a game. The latest generation of virtual foes employ simple tactical behaviours to try to outsmart the player. Enemies will attempt to ambush

the player, run for cover when they run low on ammo or suffer serious injury and sometimes even call for back up when a battle is turning against them.

A number of recent games have taken the FPS genre in interesting new directions which have increased the development challenges. Half-Life^{G-1}, named *Best Game of all Time* by PC Gamer magazine in 1999 [PC Gamer, 1999], made strong use of narrative [Carless, 2003] and confronted players with highly effective opponents that appeared to work together in teams. Deus Ex^{G-5} and Thief^{G-9} have taken a more subtle approach in which the player must try not to alert opponents to his presence and so sneak around the game world. In such games opponents must possess relatively sophisticated sensing models allowing them to see and hear the player as he tries to avoid detection [Leonard, 2003]. Amongst others, Star Trek Voyager Elite Force^{G-10} and the Rainbow Six series^{G-11} (based upon the novels of Tom Clancy) put the player within a team of characters working towards a common goal. This has given rise to a number of considerable challenges as developers attempt to ensure that computer controlled team members do not prove more of a hindrance than a help to the player.

2.1.2 Adventure Games

“Something’s rotten in the land of the dead...and you’re being played for a sucker. Meet Manny Calavera, travel agent at the Department of Death. He sells luxury packages to souls on their four year journey to eternal rest. But there’s trouble in paradise. Help Manny untangle himself from a conspiracy that threatens his very salvation.”

Grim Fandango^{G-12}

In terms of appearance and game mechanics things have changed dramatically since the first *adventure game*, Adventure^{G-13} by Willie Crowther and Don Woods, was created in the early seventies. However, game-play in the genre has remained much the same. Players move through different locations, solving puzzles and interacting with characters in order to further a story-line.

While the earliest examples of the genre were text based (players were given written descriptions of where they were, and entered commands as simple sentences, such as “eat the peach”, “enter building” and “open door”), nowadays adventure games are graphically complex (the most recent examples have been set in stunning three dimensional virtual worlds) and input is given in a variety of novel ways – the most common being the use of the mouse to direct the player’s character (from which came the name *point and click*

adventure). Classic examples of this genre include the Zork^{G-14}, Monkey Island^{G-15} and Gabriel Knight^{G-16} series.

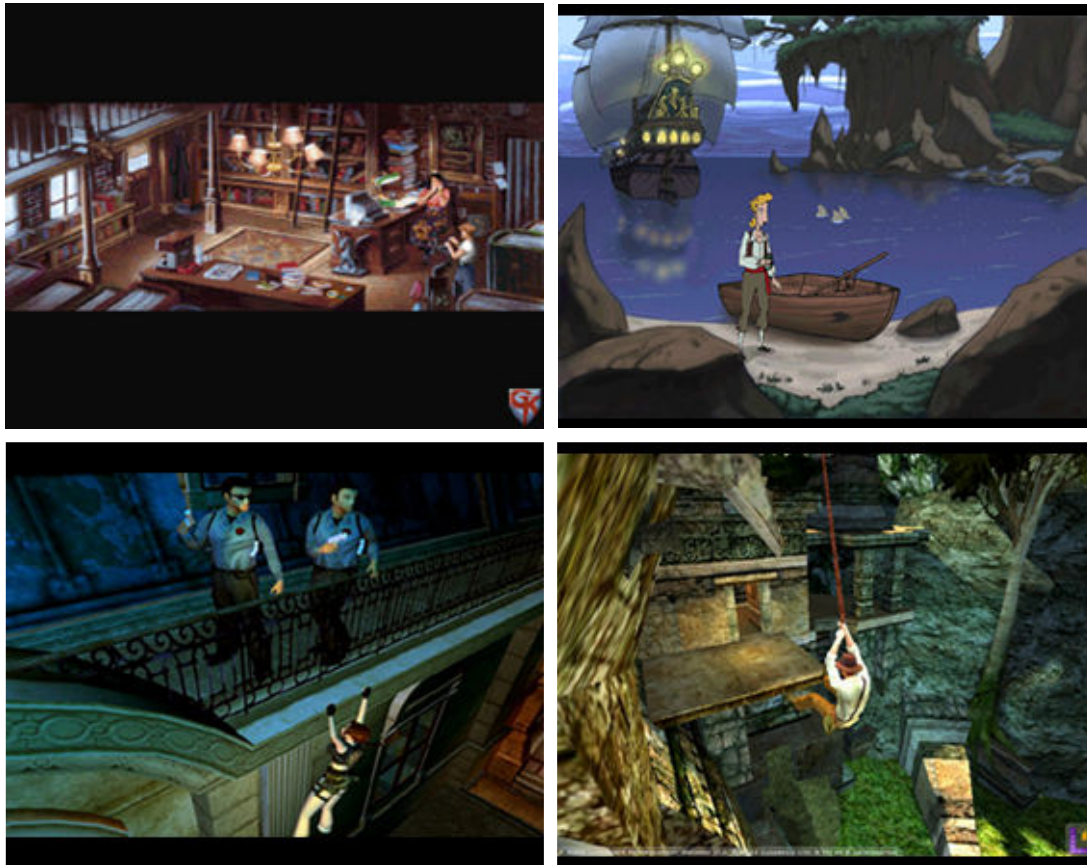


Figure 2-2: Adventure afoot in (clockwise from top left) Gabriel Knight – Sins of the Fathers^{G-16}, The Curse of Monkey Island^{G-15}, Indiana Jones and the Emperor's Tomb^{G-20} and Tomb Raider – The Angel of Darkness^{G-19}

Typically, adventure games have a linear storyline set in a limited environment. The urge to continue playing the game comes from the fact that the player can only advance the story by completing the challenges with which the game confronts them. Some attempts have been made to allow more dynamic story-lines (for example the 1997 hit *Bladerunner*^{G-18}) in which the player's actions affect the progression of the story. However, such efforts have been plagued with consistency problems.

Another major challenge facing developers of adventure games is the creation of believable and engaging NPCs, as games in this genre typically involve large amounts of player to NPC interaction. However, these interactions can become frustrating as NPCs are usually only capable of a small number of responses, which are endlessly repeated. On top of this, NPCs rarely give any appearance of having a purpose within a game world beyond their involvement with players. By achieving more realistic interactions and allowing

NPCs give the impression of having lives outside of their involvement with players, an extra degree of believability could be instilled into future adventure games.

In more recent times the *action adventure game* has emerged. These games blur the lines between the action and adventure genres, mixing the best aspects of both. Good examples include the Tomb Raider series^{G-19} and the more recent Indiana Jones and the Emperor's Tomb^{G-20}.

Figure 2-2 shows a selection of screenshots from a range of adventure and action-adventure games.

2.1.3 Role Playing Games

“Every world has a conflict. Good and evil. Friend and foe. Safety and Danger. In Baldur’s Gate II you’ll find yourself between these factions. This epic...will immerse you in a world of intrigue, adventure and fierce combat where your ability to discern between these sides - with the assistance of steel and spell – determines your fate.”

Baldur’s Gate II: The Shadows of Amn^{G-21}

Often seen as an extension of the adventure game genre, *role playing games* (RPGs) stem from the popular Dungeons & Dragons paper based games (more on which can be found at www.wizards.com) which originated in the 1970s. Over the past two decades the computer versions of these games have metamorphosed from being mostly text-based to the beautifully rendered 3-d games available today. Fine examples include Baldur’s Gate^{G-21}, Dungeon Siege^{G-22} and the Final Fantasy series^{G-23}. Game-play entails questing through fantasy locations, engaging in a mixture of puzzle solving and combat. Interactions with NPCs also form a major part of these games, as players seek to find information relating to a game’s plot.

The main differences between RPGs and adventure games are related to scale and depth. RPGs most often take place in extremely large worlds which players are free to explore essentially at their own pace. Also, RPGs are most often based on the original, and quite complex, Dungeons & Dragons rules. As well as controlling combat, and some parts of player/NPC interaction, these rules also introduce the idea of improving a player’s skills over the course of a game. At the beginning of an RPG, players create a character, giving it a range of attributes which, amongst other things, typically include intelligence, strength and charisma. As the game progresses, and players achieve tasks such as defeating foes or solving puzzles, these attributes change, allowing the player to develop new skills and

become more powerful. The drive to improve their character is one of the things which brings players back to RPGs again and again.



Figure 2-3: Role playing in (clockwise from top left) Baldur's Gate II: the Shadows of Amn^{G-21}, Neverwinter Nights: Hordes of the Underdark^{G-24}, Final Fantasy XII^{G-23} and Star Wars Knights of the Old Republic^{G-25}

The challenges facing AI developers working on RPGs are much the same as those in adventure games. NPCs must be made more believable, both by improving the believability of their behaviours, and ensuring that story-lines can both adapt to game events and remain consistent. One recent trend in RPG game-AI is the inclusion of clans and guilds. This has led to complex hierarchical social models where characters' attitudes towards players and other NPCs are coloured by membership of various groups [Alt & King, 2002].

Another interesting development in RPG gaming has come with the proliferation of broadband Internet connections. *Massively multi-player online RPGs* (MMORPGs) are online games in which huge numbers of players simultaneously inhabit the same virtual on-line world. Successful MMORPGs include Ultima Online^{G-26}, Neverwinter Nights^{G-24} and Star Wars Galaxies^{G-27}. These games follow much the same patterns as traditional

RPGs except that they are populated by so many players. Although this does somewhat eliminate the need for NPC opponents, it introduces an extra impetus for the creation of believable support characters to fill roles such as shop-keepers and barmen, as players are simply not interested in taking these parts. The behaviour of such characters is placed so closely alongside the behaviour of player characters that any inconsistencies are easily noticed. Issues related to maintaining story consistency also become more complicated in massively multi-player games.

Figure 2-3 shows a selection of screenshots of some of the more important RPGs.

2.1.4 Strategy Games

“...you have 1000 years to lead your people through the Middle Ages to greatness. Control one of the most powerful civilisations of the time. Decide whether to conquer the world through military might, rule through diplomacy and commerce, or seize power by means of intrigue and regicide. There are many paths to power but only one civilization will reign supreme”

Age of Empires II: The Age of Kings^{G-28}

Strategy games put the player in the role of a general marshalling her troops (known as *units*) into battle against computer or human controlled opponents. The games take place from a bird’s eye perspective, from which the player can issue orders and control her forces. Beyond simple combat, strategy games also require players to manage resources (used to support existing units and create new ones), and take charge of research and development, which allows access to new units and weaponry. Many games also feature diplomacy, offering players the opportunity to triumph through less violent means.

Since the earliest strategy games, the genre has split into two distinct classes – turn-based games, in which players give orders to their units one after another; and real-time games in which players order their units simultaneously. The *Civilisation* series^{G-29} is the definitive turn-based example of the genre, while the *Age of Empires*^{G-28} and *Command & Conquer*^{G-31} series stand out as fine real-time examples. Figure 2-4 shows screenshots of these and other strategy games.

From the developer’s perspective, the AI challenges arising from strategy games are split between controlling opponents’ high-level strategy, and controlling individual units’ behaviour. At the higher level opponents must be able to determine which units to create and how best to deploy them against the player, along with how best to manage resource

collection, and research and development. At unit level, issues such as path-finding, formation movement and tactical behaviours must be considered.



Figure 2-4: Battles rage in (clockwise from top left) Age of Empires II: the Age of Kings^{G-28}, Command & Conquer Generals^{G-31}, Civilization III^{G-29} and Warcraft III: Reign of Chaos^{G-32}

2.1.5 God Games

“Enter the world of Populous: The Beginning where you are god. Experience the only strategy game that puts the awesome powers of nature in your hands. Enforce your omnipotent influence over fantastic...worlds. Wield your divine power to convert wildmen to your cause or cast them down in a hail of fire and brimstone! Build your forces, annihilate the non-believers, and become ruler of the almighty universe...”

Populous: The Beginning^{G-33}

Taking fantasy to the extreme, *god games* allow the player to take on the role of protective deity to a tribe of hapless worshippers. Natural disasters often have to be overcome and conflict with rival deities, which takes place in much the same way as in strategy games, is a common feature of the genre. The main factor distinguishing god games from strategy

games is the player's ability to manipulate the environment – for example to raise mountains or to cause natural disasters – and the need to ensure that tribesmen continue to worship the player. Examples of the genre include the Populous^{G-33} series and Black & White^{G-34}. The SimCity series^{G-35} could also be considered to belong to the god game genre. These games are among those to feature in the screenshots shown in figure 2-5.



Figure 2-5: Example god games (clockwise from top left) Populous: The Beginning^{G-33}, Black & White^{G-34}, Republic the Revolution^{G-36} and Sim City 4^{G-35}

The challenges arising for the AI developer from this genre are similar to those arising from strategy games. Realistic strategic behaviour of rival deities is an important feature, as are tribes people that behave in a natural and believable manner.

Black & White brought a new level of sophistication to the god game with an advanced AI system which allowed the worshipping tribe to evolve based on the behaviour of the player. If the player is a cruel god, the tribe becomes harsh and mean, while if the player assumes the role of caring deity the tribe's world will become a paradise.

2.1.6 Sports Games

“The only game that lets you take your country to FIFA World Cup victory”

2002 FIFA World Cup^{G-37}

Albeit to varying levels of success, almost every sport imaginable has been converted into a computer game - Sega Bass Fishing^{G-38} being one of the more surprising hits. From the point of view of the AI researcher, however, there are a number of common threads running through the many sporting conversions. Principally, sports games can be divided into two sub-genres, namely team sports and individual sports, the challenges of which are quite different.



Figure 2-6: Highly realistic sports action in (clockwise from top left) Sega Bass Fishing^{G-38}, FIFA Soccer 2004^{G-37}, Colin McRae Rally 04^{G-52} and ESPN International Track & Field^{G-39}

Team sports games (such as soccer and basketball) involve the simulation, not only of individual players, but also - at a higher level - of team based tactics. At the lower level, the AI needed to control individual computer players is similar to the unit based AI required in strategy games; while at the higher level there is a requirement, again similar to strategy games, for a strategic opponent to run the entire team.

Individual sports (such as tennis and fencing) are more like action games in their AI requirements. Players should be confronted by challenging opponents that can be likened to the enemies required in action games, and so the challenges to the AI developer are much the same.

One very interesting role to emerge from the sports game genre is that of the sports commentator. A trend for recent sports games has been to create an experience much like that of watching the television coverage of a game. To achieve this, live commentary is recorded and AI systems are used to play back the right pieces of commentary at the right times.

A selection of screenshots from sports games are shown in figure 2-6.

2.1.7 Defying Categorization

Of course, just like any attempt at categorization, not all computer games fit neatly into one of the genres listed above. Firstly, there is a large amount of overlap between the different categories – 2002's Diablo II^{G-41} (shown in figure 2-7) is considered an RPG, but a huge amount of the game-play is made up of combat sequences, so could it not also be considered an action game?

Secondly, from time to time, a completely original title is released that simply defies categorization. One such example is the Sims^{G-42} (a screenshot of which is shown in figure 2-7) which was one of the shock successes of 1999, and (with its expansion packs) has gone on to become the best selling PC game of all time. The Sims puts players in control of a virtual family which they must guide through life, instructing them to perform tasks ranging from eating and washing to making new friends. Although it may sound mundane, the game is highly addictive and has been hugely successful. The Sims was also one of the first games to defy the typical game-playing demographic of males aged between 12 and 30 and attract a significant number of female players.

However, in spite of the difficulties in doing it, forming a categorization such as that above, is still useful in beginning a discussion of the basic requirements of game-AI.



Figure 2-7 Defying categorization, *Diablo II*^{G-41} and *the Sims*^{G-42}

2.1.8 What About Deep Blue?

Before finishing the discussion of the different game genres which concern this work, an obvious question must be answered – *what about Deep Blue?* Deep Blue [Feng-Hsiung, 2002] is IBM's chess playing computer system, and has ranked higher than any human chess player. So what is left to do in terms of research into game playing if computers can already beat the best human players at one of our most difficult games?

To answer this question the distinction must be made between classical games (such as chess) and modern computer games. Classical games for which computer players have been successfully created include chess, draughts, and backgammon. However, all of these games share the fact that they are set in worlds which greatly restrict the range of events that can take place within any game (i.e. on a board).

Because of their closed world nature, AI systems for playing classical games rely largely on sophisticated searching techniques, which allow the computer player to search through a multitude of possible future situations, considering what moves it might make and what moves its opponent might make in response. Based on this search, and some clever heuristics that indicate what constitutes a positive game position, the best sequence of moves can be chosen. Figure 2-8 illustrates how such a search might proceed in a game of Xs and Os from midway through the game to a winning position for the player playing as X. Xs and Os is used because of its simplicity, but a branching search in any closed world game would proceed in a similar fashion.

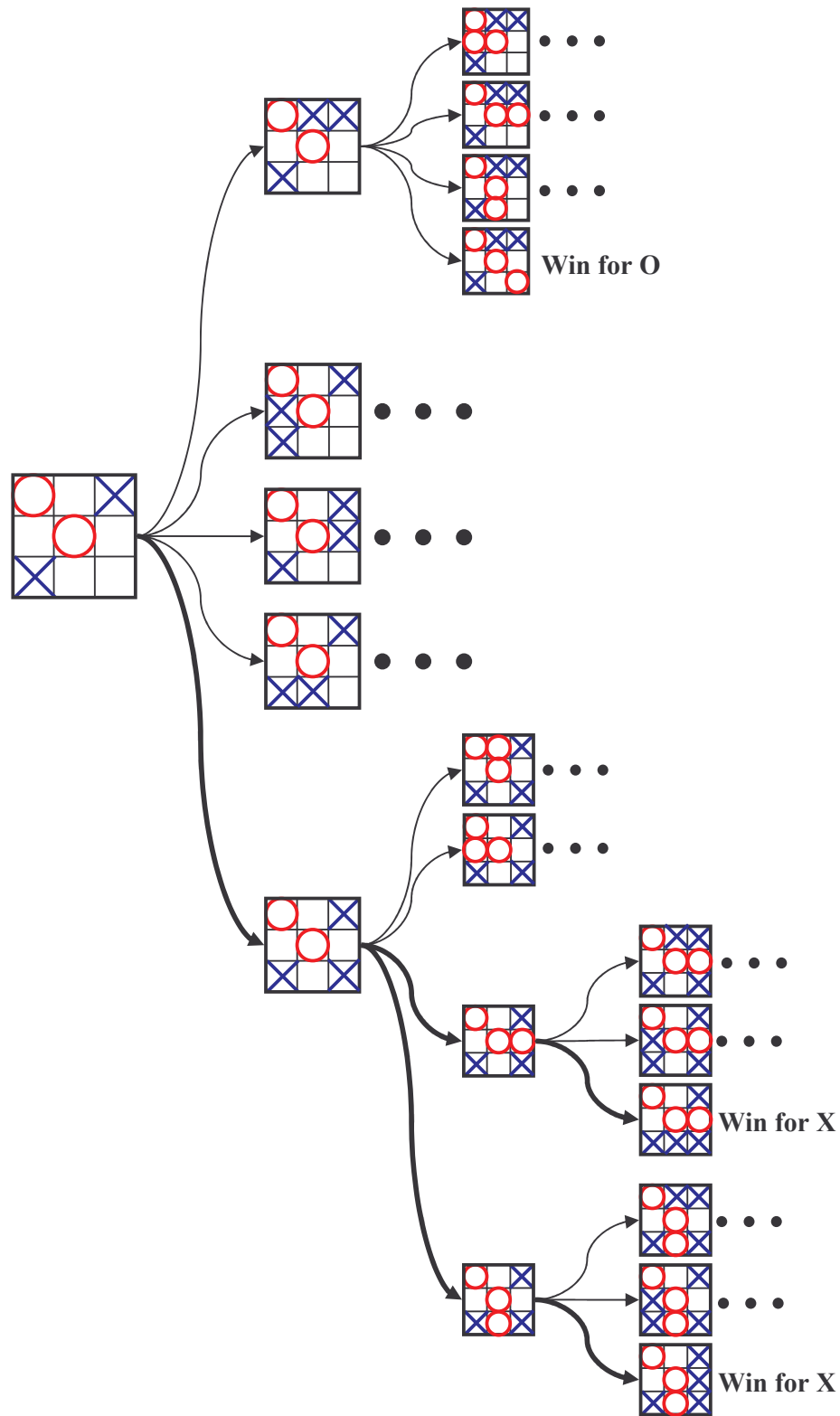


Figure 2-8: An illustration of how a branching search (many of the paths of which have been omitted for clarity) would proceed from midway through a game of Xs and Os to a winning position for the player using X. The darker arrows indicate the paths that lead to a win for X

This searching technique relies on the fact that there are a relatively small number of moves that a player can make at any given time. However, games in which this is not the case cannot be mastered by such search techniques. Tellingly, the ancient Chinese game of Go-Moku (shown in figure 2-9) has not, to date, been mastered by computer players [van der Werf et al, 2002]. The reason for this is due to the simplicity of the game's rules which make it much more freeform than a game such as chess. At any one time in a game, players have the option of making a huge number of different moves, which makes a branching style search impossible.



Figure 2-9: An illustration of a Go-Moku board

Modern computer games cannot be solved using searching techniques for the same reasons that Go-Moku cannot. Modern computer games are now set in highly realistic, dynamic worlds and take place in real-time. Players are also allowed huge amounts of freedom in terms of what they can do at any given time within a game. For these reasons, the searching techniques which have been so successful in classical games do not typically transfer to modern computer games.

Another interesting way in which research into modern computer games contrasts classical game research, is that the ultimate goals are not entirely the same. Typically, the driving force in the creation of computer systems for playing classical games is to better the performance of the best human players. For modern video games this is not necessarily the case. Video games are created to entertain, and so the goal of computer opponents is not to always beat the player, but rather to entertain her. This point will be elaborated upon in a later section.

2.2 Roles for Game-AI

Based on the different genres listed above, [Laird & van Lent, 2000] suggests a list of the different roles for AI in video games. These roles are as follows:

- **Tactical Opponents:** Tactical opponents are the kind of NPC that confront players in typical action, or individual sports games. The NPCs must attempt to tactically outwit the player, for example by selecting better weapons or performing the right moves in response to the player's actions.
- **Strategic Opponents:** As mentioned in the previous section, most team sports games, strategy games and god games require opponents which take a high-level view of a game's proceedings. In strategy and god games opponents must manage resources in the same way as players, must decide what kind of units to create, and must decide in what way these units might best be deployed against the player. For sports games the requirements are similar - what formations will be used, what tactics will be employed and what players will make the final team.
- **Partners:** Games which place the player within a team of NPCs have given rise to the need to create partner characters [Reynolds, 2003]. The challenges involved in creating these are much like those involved in the creation of tactical opponents, as they must decide how best to challenge enemy NPCs. However, there are extra restrictions placed on partner characters as they must attempt to determine a player's intentions and complement them. Also, it must be ensured that partner characters do not hinder the player's progress, thus becoming a frustrating annoyance.
- **Support Characters:** In order to make the virtual worlds in which role playing and adventure games are set really immersive, they must be populated by a wide range of characters. Many of these characters may not play a key role in a game's plot, but it is important that they appear to be more than just eye candy. These support characters, which are used to play roles such as shop-keepers, barmen and police-officers, must appear to pursue existences beyond their interactions with players, be capable of holding a player's attention and give the impression of being aware of the events taking place within the game world.
- **Semi-Autonomous Units:** The units which players can create in strategy and god games do not require a player to control their every move. Rather, players give high level orders such as instructing units to move to a certain location, to attack a group of enemies or to defend a particular area. Exactly how these orders are carried out is not controlled by the player. To enable this, units must be at least semi-

autonomous and capable of making decisions regarding issues such as path-finding or weapon selection.

- **Commentators and Camera Systems:** As game worlds have become more sophisticated many less obvious roles for the application of AI techniques have emerged. For example, sports games require commentators capable of passing suitable comment on the happenings in a game. In recent games (such as the FIFA football series^{G-37}) the sophistication of the commentary, and the inclusion of multiple commentators, has made the game experience remarkably close to the televised sports experience. Another surprising area in which AI techniques have been applied is the control of virtual cameras. As most games are now set in rich 3-d worlds, positioning the player's viewpoint to best capture game events is a non-trivial task. Intelligent cameras are beginning to find use in some games [Charles et al, 2002] in order to overcome this problem.
- **Story Directors:** Coherent narrative is extremely important in almost all game genres. However, as more sophisticated game worlds give players the freedom to explore at their own pace, maintaining a coherent story has become difficult. Players can upset a narrative structure, by performing actions out of sequence or not exploring locations crucial for a story's advancement. Intelligent story directors are being used to ensure that consistency within a story is maintained by adapting the narrative to the player's actions [Fairclough & Cunningham, 2003].

This list covers the major areas to which AI might be applied in modern computer games. The next section will discuss the AI techniques that game developers are using to solve the problems presented by these roles.

2.3 Game-AI – the Developer's Perspective

Although the AI techniques used by games developers have often been put to great effect, they tend to be rather simplistic [Woodcock, 2000b]. The main reasons for this lack of sophistication are as follows:

- **The lack of CPU resources available for AI.** Until relatively recently, the driving force behind game development has been improving the realism of game graphics. The result of this is that rendering has used most of the processing power available on typical users' machines, leaving little or no resources for anything else – for example AI. However, the power of modern processors has reached such heights

that this is becoming less of a concern. Also, modern graphics cards have become so powerful [Paschedag, 2003] that they have taken a large part of the processing burden away from a machine's CPU, leaving it free for other things – mostly realistic physics simulation and AI.

- **The lack of development time available for AI.** Related to the previous point, most of the development time in creating games has been used for improving graphics systems. This means that AI is typically added to a game towards the end of the development cycle, leaving little time to try new techniques or fine tune existing ones. Again this is currently changing as development teams are beginning to employ full time AI developers from the very beginning of the game development cycle [Woodcock, 2000b].
- **Graphics research has overshadowed everything else.** As game companies rushed to outdo each other in terms of graphical realism, most other areas have been ignored. For this reason AI research and development in the game development community has been extremely low-key.
- **A suspicion amongst game developers of non-deterministic techniques.** Before games are released to the public they are extensively tested to ensure that they are free of bugs, and that the game-play is as finely tuned as possible. For this reason, game developers are suspicious of non-deterministic AI techniques (such as learning systems) as these cannot be exhaustively tested.
- **A lack of understanding of sophisticated AI techniques within the game industry.** Many developers within the game industry are not familiar with some of the sophisticated AI and machine learning techniques that are widely used within other industries and academia. The result of this is that developers are slow to apply sophisticated techniques, and often don't recognise areas in which such techniques could be used. However, as game development is becoming more mainstream, people from more traditional computer science backgrounds are becoming involved in the industry and bringing an understanding of such techniques with them.

Although all of the above reasons are becoming less pertinent, they are still hindering the use of sophisticated AI techniques in game development. The following section will paint, in broad strokes, an overview of the current state of the art of game-AI.

2.3.1 The Current State of the Art

With regard to AI, the current game-playing experience could be summed up with a single word - frustration. Too often, games which are set in beautifully rendered 3-d worlds are populated by a cast of characters that, although they look fantastic and move wonderfully realistically, behave in obviously non-intelligent ways.

For example, units in strategy games that cannot find their way safely across a game environment (Age of Empires^{G-28}), police cars that display no regard for the safety of civilian drivers while chasing the player in a driving game (Grand Theft Auto III^{G-43}), partner characters that get in the way of the player (Hidden & Dangerous^{G-44}) or squads of enemy soldiers that follow one another into the line of fire (Medal of Honour^{G-45}). All of this (and there are countless other examples) ruins the sense of immersion created by all of the other aspects of a game. It is the responsibility of future game-AI developers to ensure that this kind of obvious non-intelligent behaviour is avoided.

However, in spite of the many examples of bad game-AI, there are a growing number of examples to the contrary. The following sections will describe the most important and interesting AI techniques being used in games today.

2.3.2 Path-Finding

The first AI-related area to enjoy transfer of techniques from academia to game development was path-finding. For many years games had suffered terribly from NPCs incapable of manoeuvring themselves successfully or realistically around a game world. Accurate path-finding is a requirement of almost every game genre, and so achieving it successfully is crucial. For this reason, path-finding has received more attention from game developers than any other area of game-AI.

The first step in performing successful path-finding is to populate a game world with path-finding information, across which a search can be performed. This is usually achieved by adding an invisible layer of nodes on top of the game terrain which indicates which areas are accessible from which other areas. Once this has been achieved, a search across these nodes can be used to determine a path from one part of the game world to another. Nodes, or the links between them, can be annotated with information regarding the difficulty of crossing particular terrain types, the danger of particular areas, or anything else which might be taken into account in forming a believable path.

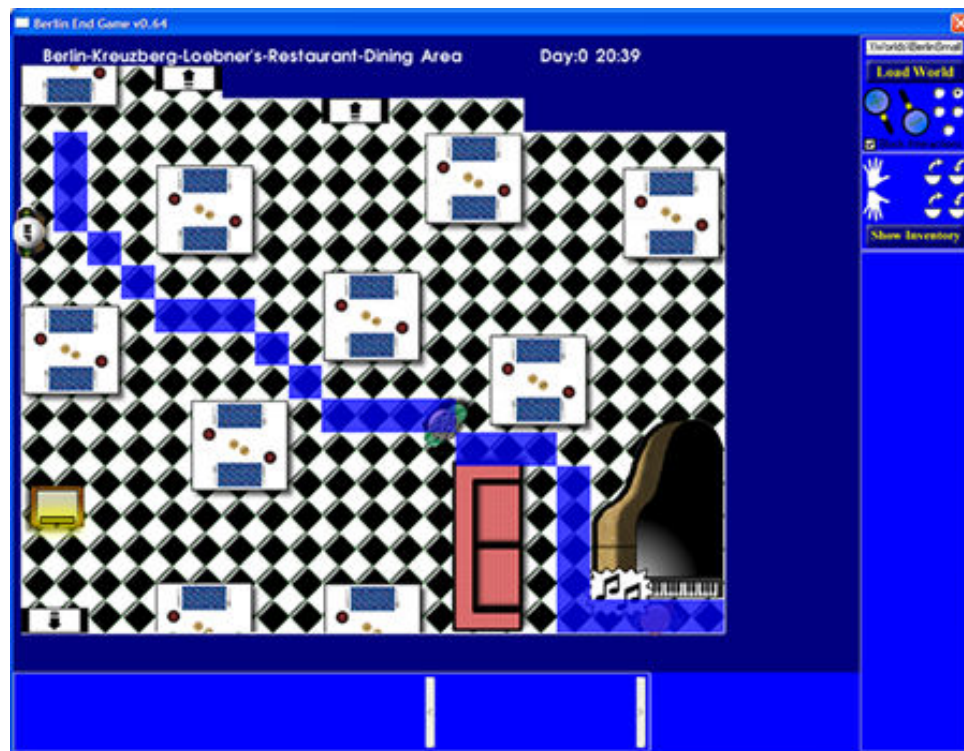


Figure 2-10: An illustration of the path-finding process in the Berlin-End-Game application which will be discussed in section 5.6

Game developers have borrowed a range of search algorithms from academia, and in particular robotics. Such algorithms have included simple breadth or depth first searches, bi-directional searches, Dijkstra's algorithm and the use of influence maps. However, the A* path-finding algorithm has become the de-facto standard in modern game development. The reasons for this are that the A* algorithm is relatively simple, robust, flexible and by now fairly well understood amongst the game development community. For a detailed discussion of path-finding in general, and the A* algorithm see [Smith, 2002, Stout, 1996].

2.3.3 Finite State Machines

For the control of NPCs the *finite state machine* (FSM) [Houlette & Fu, 2003] has become the weapon of choice amongst game developers. An FSM is a simple system in which a finite number of *states* are connected by a graph of directed *transitions*. When used for the control of NPCs, the nodes of an FSM indicate the possible states (manifested as actions within the game world) NPCs can assume. Transitions indicate how changes in the state of the game world or the character's attributes can move the NPC from one state to the next. Figure 2-11 shows a sample FSM for the control of an NPC in a typical action game.

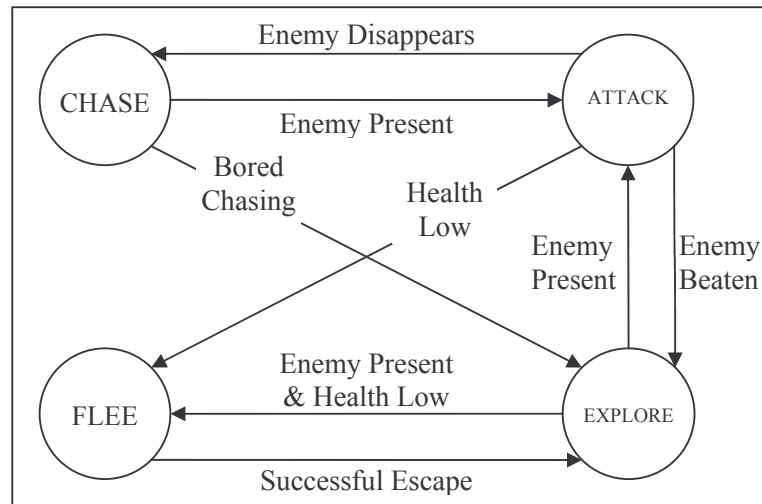


Figure 2-11: A simple finite state machine for a soldier NPC in a typical action game

FSMs are widely used because they are so simple, well understood and extremely efficient both in terms of processing and memory usage. However, FSMs are not without their drawbacks. When designing state machines developers must envisage every possible situation that might confront an NPC over the course of a game. While this is quite possible for many games, for NPCs that are required to move between many different situations this task can become overwhelming. Similarly, as more and more states are added to an FSM, designing the links between these states can become a mammoth undertaking.

2.3.4 Expert Systems

Expert systems are knowledge based systems that attempt to rival the performance of human experts. An expert system is architected by acquiring expert domain knowledge from human experts (typically through interview) and encoding this knowledge using a representation scheme which allows it to be queried in order to solve domain problems. Typically, a query to an expert system takes the form of a set of feature values which describe that query. Based on these values an expert system makes a recommendation. For example, a system in the medical domain might be created in order to diagnose patients. The inputs to such a system might be a set of test results and presented symptoms, while the output would be a positive or negative diagnosis.

In game-AI, expert systems can be used to control the behaviour of NPCs, where the expert is considered to be a human player. The remainder of this section will discuss a number of different expert system based techniques used in game-AI. Each technique uses

a different scheme for encoding and querying domain knowledge, but they can still all be considered expert systems.

Rule-Based Systems

From [IGDA, 2003] the definition of a rule-based system states that “...[they are] comprised of a database of associated rules. Rules are conditional program statements with consequent actions that are performed if the specified conditions are satisfied”. Rule-based systems have been used extensively in game-AI [Christian, 2002], in particular for the control of NPCs in role playing games. The behaviours of NPCs are scripted using a set of rules which typically indicate how an NPC should respond to particular events within a game world. Borrowed from [Woodcock, 2000a], figure 2-12 shows a snippet of the rules used to control a warrior character in the RPG Baldur’s Gate^G.

21

```

IF
    // If my nearest enemy is not within 3
    !Range(NearestEnemyOf(Myself),3)
    // and is within 8
    Range(NearestEnemyOf(Myself),8)
THEN
    // 1/3 of the time
    RESPONSE #40
    // Equip my best melee weapon
    EquipMostDamagingMelee()
    // and attack my nearest enemy, checking every 60
    // ticks to make sure he is still the nearest
    AttackReevalutate(NearestEnemyOf (Myself),60)
    // 2/3 of the time
    RESPONSE #80
    // Equip a ranged weapon
    EquipRanged()
    // and attack my nearest enemy, checking every 30
    // ticks to make sure he is still the nearest

```

Figure 2-12: A sample script from the combat behaviour of a warrior character in the RPG Baldur’s Gate

Rule-based systems are favoured by game developers as they are relatively simple to use and can be exhaustively tested. They also have the advantage that rule sets can be written using simple proprietary scripting languages [Berger, 2002], rather than full programming languages. This makes it easier for game designers, rather than programmers, to author rule sets. Development companies have also gone so far as to make these scripting languages available to the general public, enabling them to author their own rule sets.

Rule-based systems, however, are not without their drawbacks. Authoring extensive rule sets is a non-trivial task, and so they are usually restricted to simple situations. Also, rule-based systems can be restrictive in that they do not allow sophisticated interplay between

NPCs' motivations, and require that authors foresee every situation that an NPC might find itself in.

Some of the disadvantages of simple rule-based systems can be alleviated by basing them on more sophisticated underlying theories. One such example [Laramée, 2002a] uses Dempster-Schafer theory which allows rules to be evaluated by combining multiple sources of (often incomplete) evidence. This goes some way towards supporting the use of rule-based systems in situations where complete knowledge is not available.

Decision Trees

Decision trees model the process of making a composite decision as a series of more simple decisions, each one of which is based on a single variable. The decision process is modelled as a tree structure in which each node is labelled with a variable. The connections emanating from a node to its children represent possible values for the variable at that node. When a query is presented to a decision tree, a decision is made by following the links which best match the current value of each variable until a leaf node, corresponding to a decision, is reached. Taken from [Russell & Norvig, 1995] figure 2-13 shows a decision tree which can be used to make a decision about whether or not to eat at a particular restaurant.

Decision trees are used extensively in industry, medicine and finance and are amongst the most widely used machine learning techniques around. The main reason decision trees are so widely used is that their decision making process is so easy to explain. Explanation is crucial in many application domains, particularly within the medical field where patients are not comfortable with blindly following a machine's diagnoses. Decision trees have also been used for the control of NPCs in a number of games, including the aforementioned Black & White [Evans, 2002].

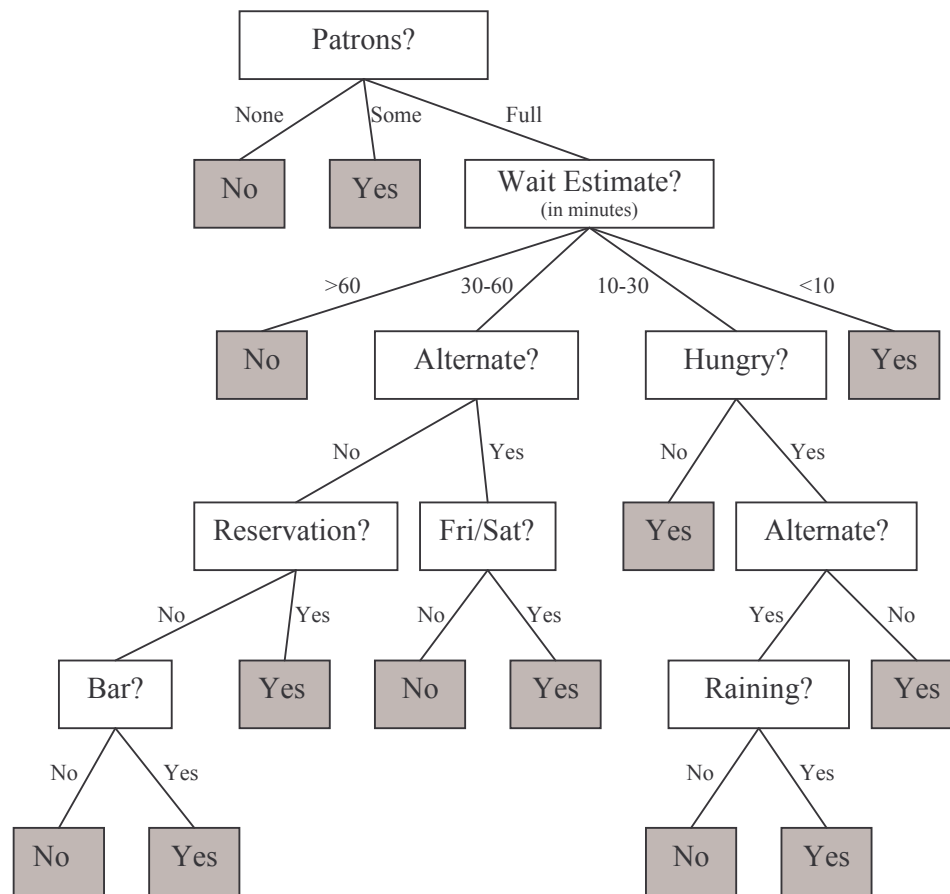


Figure 2-13: An expert system decision tree which determines whether a person should eat in a restaurant based on a number of properties

Bayesian Networks

Many situations in which expert systems must make decisions are plagued by the problem of uncertain knowledge. In such cases simple rule-based techniques cannot be used, as not enough data is known. *Bayesian networks*, or *belief networks*, can be used to make valid decisions based on uncertain knowledge. [Russell & Norvig, 1995] define a Bayesian network as a graph in which the following holds:

1. A set of random variables makes up the nodes of the network.
2. A set of directed links connects pairs of nodes, indicating that a source node has influence over a destination node.
3. For each node, there exists a conditional probability table that quantifies the effects that the parents have on the node.
4. The graph is a-cyclical.

Based on such a graph, and the conditional probabilities which it contains, inferences can be made in the presence of uncertain evidence. For instance, figure 2-14 shows an

illustration of a simple Bayesian network based on an example given in [Tozour, 2002]. This network could be used to allow a guard NPC, in a game such as Thief: The Dark Project^{G-9}, determine whether a set of observations (such as hearing a noise or spotting some movement) indicate the presence of an intruder, or just some rats. This diagram shows how all observations influence these two possibilities. Not shown are the probability tables which quantify these influences, and allow inferences to be made.

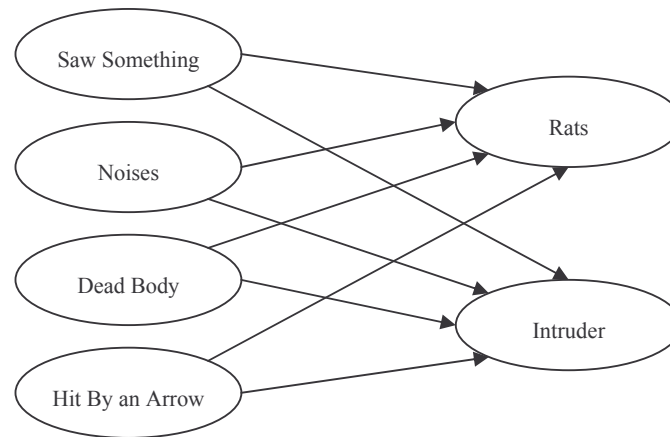


Figure 2-14: A simple Bayesian network used to allow a guard in a game such as Thief: The Dark Project^{G-9} determine whether observations indicate the presence of an intruder, or harmless rats

Fuzzy Logic

Fuzzy logic is a branch of mathematical logic which attempts to deal with incomplete information. Rather than dealing with strict true or false values, as is the case in conventional logic, fuzzy logic deals with degrees of truth. Since its first appearance in the late 1960s fuzzy logic has been used successfully in a wide range of applications ranging from the control of washing machines, to subway trains.

Fuzzy logic is an ideal control mechanism for computer games as it is “subtle...complex...[and] lightning fast at run-time” [O’Brien, 1996] and enables game characters “to perform some remarkable human factoring” [Morris, 1999]. However, game developers have been slow to use the full power of fuzzy logic, due to the complexity of authoring such systems. One attempt to overcome these difficulties is the Free Fuzzy Logic Library [Zarozinski, 2002] which is an open-source fuzzy logic API for use in video games. Fuzzy logic is discussed again in section 4.4.2, and in detail in appendix A.

2.3.5 Artificial Life

Artificial life, or *a-Life* is a branch of AI which attempts to combine such diverse fields as computer science, biology and cognitive science, in an attempt to create artificial creatures which display characteristics common in nature, e.g. learning, evolution and co-operation. More of a design philosophy than a strict methodology, by taking its inspiration from nature, a-Life attempts to bypass some of the complications associated with using more traditional AI techniques to simulate complex systems.

One of the seminal works in a-Life is Craig Reynolds' Boids system [Reynolds, 1987], which simulates the flocking behaviours exhibited in nature by schools of fish, or flocks of birds. The essence of Reynolds' approach is the use of a small number of simple rules which each creature within a flock obeys. From these simple rules, complex flocking behaviours emerge, and this *emergent behaviour* is one of the key features of any a-Life system. In the original Boids system, each creature used just three rules (illustrated in figure 2-15), to control its movement.

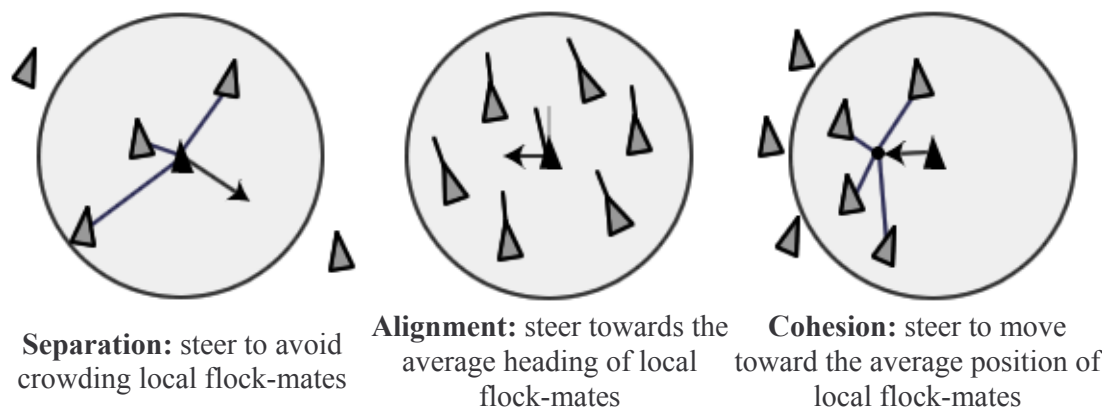


Figure 2-15: The three rules used by Reynolds' original Boids system to simulate flocking behaviours

The Boids system and its successors have been used to imbue virtual creatures with the ability to flock in countless games (including *Half-Life*^{G-1}) and a number of mainstream movies, such as *Batman Returns* and *Cliffhanger*. In fact, in 1997 Reynolds received an academy award for his work.

Breve [Klein, 2002] is another interesting system making use of artificial life techniques. Breve is a 3d open source, freely available simulation environment designed to allow easy experimentation with a-Life techniques. So far the system has not been used directly in game applications, but a number of example applications have been created. These include

a flocking based on Reynolds work described above, and a system which displays the evolution of walking behaviours based on work by Sims [1991].

The a-Life approach at first seems like an ideal candidate to satisfy many of the needs of game-AI developers. However, beyond flocking it has not been widely used in commercial games. One possible explanation for this is that the very fact of emergence makes a-Life systems extremely difficult to test, thus making them anathema to game development testing departments. Interesting titles which do make use of a-Life techniques include the Sims^{G-42}, Creatures^{G-46} and Mindrover^{G-47}.

2.3.6 Learning Systems

It is anticipated that learning will be one of the major advances in game-AI in the future [Manslow, 2002]. The expectation is, that imbuing NPCs with the ability to genuinely adapt to players' behaviour will fundamentally change the way that games are played. However, even though there are a set of extremely powerful learning techniques which have been used widely in industry and academia, these have not crossed over to any great extent to game-AI. The most pertinent reasons for this are firstly, that in general game developers do not have a great understanding of learning techniques, and secondly that developers are wary of any techniques which could lead to game characters exhibiting unexpected behaviour, fearing that this could make a game unplayable. Also, it is difficult to frame a learning problem in the dynamic worlds in which games are set. In spite of this wariness, a smattering of games have put learning techniques to use and the remainder of this section will discuss these, and the particular techniques they used.

Artificial Neural Networks

Artificial neural networks (ANNs) [Bishop, 1995] are a class of machine learning technique based upon the manner in which neurons in biological brains operate. ANNs can be used to perform classification tasks in which a set of inputs describing a particular problem case are presented to a network, which then outputs its class. Before this classification takes place, an ANN must be trained by presenting to it a set of patterns for which the correct classifications are known, and adjusting the internal configuration of the network in order to allow it to recognise these patterns. This process, and the overall area of ANNs, will be discussed in detail in section 4.5.6.

The rally car racing game, Colin McRae Rally 2.0^{G-48}, is one of the only commercially successful games which has made well known use of ANNs [Generation5, 2001].

Modelling the styles of rally drivers across unstable terrain such as gravel and mud is extremely difficult to do with a set of hand crafted rules, and for this reason the developers of the game used an ANN to learn the way in which a car could be most successfully steered around a rally track. Although this was a successful use of ANNs in a commercial game, it is important to note that learning did not take place within the game in order to adapt to players' behaviour. Rather, the ANN was used as a development tool and its results were fixed within the final game.

When *Black & White*^{G-34} was released it delighted players with the ability of its main characters (large animal representations of the player's god character) to learn how to behave within the game world [Evans, 2002]. One of the ways in which this took place was through the use of perceptrons, a very simple form of ANN. Perceptrons were used to model the desires (such as the desire to eat something) of these animal characters, and simply sum a set of weighted desire source inputs (such as the animal's hunger or the tastiness of an object that the animal is considering eating) to give the value of a particular desire. ANN based techniques were used to learn the weights applied to these inputs.

A final example of ANNs being put to use is in the *Creatures* series^{G-46}, although these titles are sometimes considered more toys than games. *Creatures* simulates the behaviour of a population of virtual pets (known as Norns) that inhabit a 2-d virtual world. Users can observe Norns' behaviour and indirectly control them using reward and punishment techniques, this way changing the patterns of their behaviour. The behaviour of each Norn is controlled by a series of ANNs which attempt to model biological systems. However, although the use of learning within the *Creatures* games is undoubtedly impressive, it must be noted that it is not particularly suitable for use in more general games. Norns do not evolve towards any particular goal, but rather their purpose is to entertain the user. Also, Norns are not capable of the kind of sophisticated goal-directed behaviour required by more mainstream games.

Genetic Algorithms

Genetic algorithms (GAs) were formally introduced in the 1970s by John Holland [1973]. GAs mimic the mechanisms of the natural evolution of species to stochastically solve search problems, where other means are too inefficient. Mechanisms like mutation, survival of the fittest, and crossover (breeding) have analogues in GAs. Although they have been much touted as being ideally suited to the requirements of game-AI, GAs have not been widely used commercially. However, there are a few applications at the fringes of

game-AI which make use of the technique. [Laramée, 2002b] discusses how GAs can be used to evolve the behaviour of characters in a fantasy role playing game. Also, the Norns which populate the aforementioned Creatures games are capable of breeding, which uses standard GA techniques to create new creatures based on their parents' attributes.

Reinforcement Learning

Reinforcement learning [Mitchell, 1997] is another AI technique which at first glance appears to suit the requirements of game-AI. Reinforcement learning is an unsupervised learning technique which adjusts the decision making process based on the outcome of actions. Thus, reasoning processes which lead to successful actions are reinforced, while those leading to unsuccessful actions are degraded. Reinforcement learning was used in Black & White^{G-34}, in which the player could interact with their creature character by petting, or striking it. Petting the creature encouraged the its recent actions, while striking it discouraged them [Evans, 2002].

2.3.7 Cheating the Player

The final major technique utilised by game developers, and worthy of discussion, is simply to cheat. This is most often used in action and strategy games. As NPC opponents are part of the simulated game world, they can be given access to the full description of this world. In practical terms, this means that NPCs can have the ability to see through walls, always have perfect aim, know exactly what their human opponent is doing with his armies or what units he is developing, and so on. On top of this NPCs can be given limitless ammunition or, in strategy games, endless resources to build new units.

Although initially this sounds like an appalling situation, it can in fact be used to great effect. As long as the player does not know that their opponent is cheating, it is perfectly acceptable, as players tend to assume that their opponents are simply better at the game than they are. Anecdotal evidence shows that players attribute great strategic capabilities to AI opponents that place units wherever they are needed, simply by cheating. However, a player's discovery that their opponent is cheating is disastrous. Any illusion that the opponent is playing an intelligent game is instantly shattered, and players tend to switch off as soon as they discover that this is the case.

2.3.8 Game-AI Systems Development Kits

Systems development kits (SDKs) have had huge success in computer game development, particularly in the areas of graphics (Maya, available at www.maya.com; and 3ds Max, available at www.discrete.com) and physics modelling (Havok, available at www.havok.com). In a poll at gameAI.com, a highly respected game-AI web site, it was shown that many developers feel that the release of a sophisticated SDK would be just the impetus required for game-AI to make the massive jump expected by gamers². However, many developers feel an aversion to the use of SDKs. The first reason for this is that developers feel that an SDK that does too much will, by its inclusion in a game-engine, restrict developers' freedom. Secondly, developers feel that an SDK that does not do enough is not worth the expense, as they could code the same capabilities themselves.

In spite of this reluctance, a number of commercial game-AI SDKs are currently available. Some of the more notable offerings are Character Studio (www.discreet.com) which controls agents in crowds; AI-Implant (www.ai-implant.com) which enables rule-based control of game characters; Renderware A.I. (www.renderware.com) which enables path-finding and a number of useful tactical behaviours; and DirectIA (www.directia.com) which claims to be a total AI solution for games. However, none of these SDKs has been widely accepted by the game development community. The key to the creation of a successful SDK will be in striking the balance between including enough functionality so as to make the system non-trivial, while at the same time not over-complicating the system, and thus allowing it to become restrictive. The successes and failures of available AI SDKs are discussed in [Dybsand, 2003].

2.3.9 The MOD Community

In recent times many game developers have begun to make portions of their game code freely available so that the general public can author their own game content. These modified versions of original games (or *mods*) are typically released freely back into the public domain to give gamers extra content for their most loved games. Typical mods overhaul the appearance of original games, often creating levels and character models to mimic historical events, or films and television shows. Well known mods include a Star

² The results of this poll are available at www.gameai.com

Trek themed version of Homeworld^{G-49} and a World War II themed version of Half-Life^{G-1}.

However, mod developers can change almost every aspect of a game to create something far removed from the original. Counter Strike^{G-50}, a Half-Life^{G-1} based mod, is an excellent example of this, and has gone on to become a successful commercial game in its own right.

Game modifications also present excellent opportunities for game-AI research, as the AI in original games can be experimented with. This is on-going in the creation of computer controlled characters (or *bots*) in multi-player action games. A number of academic projects are also using mods as the basis for their work, and some of these will be discussed in the next section which focuses on game-AI in academia.

2.4 Game-AI in Academia

Until recently academic research into game-AI had been rare. However, a considerable amount of research into the area is now beginning to take place. A number of high profile research groups have been formed, and some universities are even offering game development courses which include game-AI modules³. Much of the new game-AI research has emerged from work conducted with military institutions, and from existing research efforts in the areas of interactive drama and interactive narrative. The remainder of this section will describe some of the more interesting academic game-AI projects⁴, and conclude with an exploration of some of the pitfalls of pursuing research into the area.

2.4.1 Notable Academic Game-AI Projects

One of the first major academic research projects into the area of game-AI was led by John Laird at the University of Michigan, in the United States. Laird's work uses the Soar architecture [Rosenbloom et al, 1993], which was developed in the early eighties in an attempt to "*develop and apply a unified theory of human and artificial intelligence*". Soar is essentially a rule-based inference system, which takes the current state of a problem and

³ For example, both located in England, Bolton University (www.bolton.ac.uk) offers a degree in *Computer Games and Software Development* and the University of Wolverhampton (www.wlv.ac.uk) includes game development options in its computer science program.

⁴ This section will only cover projects which are directly concerned with game-AI. Some seemingly obvious omissions will be described within §3.4 which covers academic projects looking into the more general area of intelligent agents for the control of virtual humans.

matches this to production rules which lead to actions. After initial applications into the kind of simple puzzle worlds which characterised early AI research [Laird & Newell, 1983; Laird et al, 1984], the SOAR architecture was applied to the task of controlling computer generated forces [Hill et al, 1997; Jones et al, 1999]. This work led to an obvious transfer to the new research area of game-AI [Laird, 2000].

Laird's group initially applied the Soar architecture to the task of controlling opponents in the multiplayer version of Quake^{G-51} (a first person shooter in the mould of Doom^{G-4}) [Laird, 2000]. This proved quite successful leading to opponents that could play against human players, and even display some sophisticated behaviour such as planning based on anticipation of what the player was about to do [Laird, 2000]. The main drawback to this research is that the Soar system is enormously resource hungry, with NPC controllers typically running on separate machines to the actual game.

In recent times Laird's group have begun to focus upon more character-centric games [Laird et al, 2002]. Although the Soar architecture is still being used, it is being applied to a game in which the player takes the role of a ghost attempting to influence the emotions and behaviours of computer controlled characters.

At Northwestern University in Chicago, the Interactive Entertainment group has also applied approaches from more established research areas to the problems facing game-AI. In [Forbus et al, 2001] a team lead by Ken Forbus have extended research previously undertaken in conjunction with the military [Forbus et al, 1991], and applied it to the problem of terrain analysis in modern strategy games. Their goal is to create strategic opponents which are capable of performing sophisticated reasoning about the terrain in a game world, and using this knowledge to identify complex features such as ambush points. This kind of high level reasoning would allow AI opponents play a much more realistic game, and even surprise human players from time to time - something that is sorely missing from current strategy games.

Within the same research group, Ian Horswill leads a team which are attempting to use architectures traditionally associated with robotics for the control of NPCs. In [Horswill & Zubek, 1999] it is considered how ideally matched the behaviour based architectures often used in robotics are with the requirements of NPC control architectures. This work has led to the creation of a test-bed environment, FlexBot (details of which can be found at flexbot.cs.northwestern.edu/about.htm), which is an add on to Half-Life^{G-1}. Using this

environment, a number of NPC implementations [Khoo et al, 2002; Khoo & Zubeck, 2002] have been created which use reactive architectures to control NPC behaviour.

As well as the work which has spring-boarded from existing applications, a number of projects have been started purposely to tackle problems in game-AI. Three which particularly stand out are the Excalibur Project, led by Alexander Narayek [2001], work by John Funge [1999] and the CogAff project [Hawes, 2002]. All three of these projects have attempted to apply sophisticated planning techniques to the control of game characters.

Narayek uses constraint based planning to allow game agents reason about their world. By using techniques such as local search, Narayek has attempted to allow these sophisticated agents perform resource intensive planning within the constraints of a typical computer game. The term *anytime agent* is used to describe the process by which agents actively refine existing plans based on changing world conditions.

Funge uses the situational calculus to allow agents reason about their world. Similarly to Narayek, he has addressed the problems of dynamic worlds, plan refinement and incomplete knowledge. Funge's work uses an extension to the situational calculus which allows the expression of uncertainty.

While the approaches of both of these projects have shown promise within the constrained environments to which they have been applied, it remains to be seen whether they can be successfully applied to a commercial game environment, with all of the resource constraints that this entails.

The CogAff project also makes use of the concept of anytime agents. This project uses interruptible planning techniques to control game agents in Unreal Tournament^{G-6}, a multiplayer first person shooter. However, although this has been a relatively successful experiment, as the researchers themselves admit, since the planning requirements in a game such as Unreal Tournament "*favour physical qualities over cerebral ones*" [Hawes, 2002] more work will be required in order to definitively prove the usefulness of such an architecture.

2.4.2 Problems Facing Game-AI in Academia

Unfortunately, since game-AI is such a new research area, working in it is not without its pitfalls. The first problem is that there is a lack of formal structure to the field. No dedicated traditional academic journals exist (although a number of online journals such as

IJIGS⁵ are becoming established), and only now are a few conferences dedicated to the subject beginning to gain respect (for example, the annual industry-driven Game Developers Conference, the Game-On conference held every year in the UK and the US based AAAI spring symposia on Artificial Intelligence and Computer Games). This makes it difficult to find avenues for publishing, and leads to complications in keeping abreast of emerging work.

To further complicate things, the world of commercial game development is one of suspicion, distrust and high secrecy. Games companies that have developed new technologies are not willing to share these with other companies or academic institutions, as new technologies are the key to new products, and business success. Until the lack of contact between commercial game developers and researchers is amended, it will remain difficult for advances to be made in the area.

However, in spite of these difficulties research into game-AI can be extremely rewarding. In fact [Laird & van Lent, 2000] goes so far as to suggest that computer games are the perfect platform upon which to pursue research into the elusive human-level AI. Games take place in dynamic worlds in which complex decisions, often based on partial knowledge, must be made. This reads like a list of the conditions required to formulate the really difficult AI problems which have troubled researchers since the birth of the field.

2.5 Summary & Conclusions

Games will no longer be sold based on their graphics alone [Rubin, 2003]. Rather, other features are expected to take the place of graphics as the major selling point for future games. As realistic physics simulation in games is fast becoming a reality, sophisticated AI is now seen as the forerunner to do this.

Currently, game-AI is relatively primitive, mainly due to game developers' blinkered focus on graphics technologies, resource restrictions and a lack of understanding of sophisticated AI techniques amongst the game development community. However, this situation is changing and a number of games have made impressive use of interesting AI techniques. In parallel, game-AI research is becoming more established in academia and a number of projects are now well underway in highly respected institutions.

⁵ IJIGS is available at www.scit.wlv.ac.uk/~cm1822/ijigs.htm

The ways in which AI can be used in games are broad, but by far the most promising is the control of NPCs. The different game genres currently on the market offer a range of different challenges to doing this. For this reason, an attempt to develop a system capable of being used to drive the behaviour of NPCs in games in all genres would not be realistic. Instead, this work focuses on a system to drive the behaviours of NPCs in character-centric games, particularly those in the adventure game and role playing game genres. These genres have been selected as offering the most scope for creating immersive game experiences, set in worlds that are populated by NPCs with whom players can become more deeply involved than in other game genres. Also, in a poll at the highly respected web site gameAI.com, it has been suggested that RPGs will be the area in which the next major advance in game-AI will be made⁶. Finally, little work has been done in this area, as most research into game-AI has focused on the creation of tactical opponent NPCs for action games. Within the area of character-centric games, the support character role has been chosen as offering the most promising area for the application of AI techniques. These characters will populate game worlds offering more immersive game experiences than those in current games.

In order to create sophisticated support characters for highly immersive character-centric games, this work has developed an intelligent agent architecture. The next chapter will discuss the area of intelligent agents in general, and in particular how intelligent agent techniques have been used for the control of virtual humans in virtual reality applications, including games.

⁶ The results of this poll are available at www.gameai.com

3 Intelligent Agents

The major application area for AI in computer games is in the control of NPCs. The most obvious way to implement sophisticated NPCs in games is as *intelligent agents*, to which this chapter will serve as an introduction.

The chapter's first section will focus on the question of what exactly an intelligent agent is. Following this, the ways in which agents for interactive entertainment applications (such as games) differ from those used in other application areas will be discussed. This section will also include an overview of the existing intelligent agent systems for the control of *virtual humans*. Research into the field of virtual humans is more extensive than research into game-AI. As well as being used in games, virtual human technologies have been applied to areas such as film-making, human computer interaction and industrial design. As this is quite a vibrant research area, and is extremely close to the game-AI field, it is worth exploring - even if some of its concerns are somewhat different.

To justify the creation of an intelligent agent architecture for the express purpose of controlling game NPCs, the notion of *virtual fidelity* will be introduced. Following, this the requirements for intelligent agents in computer games will be enumerated, after which the proposed architecture will be introduced.

3.1 What is an Intelligent Agent?

The term intelligent agent is one of the most misused in modern computer science. Much like when the term artificial intelligence itself was first introduced, the *cutting edge* flavour of the phrase intelligent agent has led to its being attached to a huge variety of software applications. This ubiquity is exacerbated by the fact that definitions for the term are somewhat nebulous. In [Wooldridge & Jennings, 1995], the search for a definition of the term intelligent agent is considered similar to the search for an answer to the question *what is intelligence?* Some attempts to define the term are so broad that they include almost all computer programs, while others are so narrow that they only include a small collection of niche systems (typically those created by the originators of a particular definition). For example, if a broad enough definition of sensors is assumed, the definition from [Russell & Norvig, 1995] (*"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors"*) can be stretched to cover almost any piece of software, while the definition from [Coen, 1995] (*"Software agents are programs that engage in dialogs [and] negotiate and coordinate transfer of information"*) appears much too narrow.

The following are two of the more useful definitions. Firstly, Wooldridge and Jennings [1995] describe an agent as *"... a hardware or (more usually) software-based computer system that enjoys the following properties:*

- ***autonomy:*** *agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state*
- ***social ability:*** *agents interact with other agents (and possibly humans) via some kind of agent-communication language*
- ***reactivity:*** *agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it*
- ***pro-activeness:*** *agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative"*

Secondly, [Franklin & Grasser, 1996] states that “*An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future*”.

Between them, these two give a definition of agency which is both exclusive and inclusive enough to be useful. NPCs in games suit a very strong notion of agency, in that they should display all of the characteristics described by Wooldridge and Jennings, and should also fit the definition of Franklin and Grasser.

3.2 Agent Implementations

Along with categorising agents based on their capabilities (as is the approach of the above definitions) it is also useful to categorise intelligent agent systems based upon their implementations. Research to date has resulted in three distinct agent implementation architectures, which the remainder of this section will describe.

3.2.1 Reactive Agents

Reactive, or behaviour based, agents are the simplest form of intelligent agent architecture, with Brooks’ subsumption architecture [Brooks, 1985] being the most widely acclaimed implementation. Reactive agents operate in a hard-wired, stimulus-response driven manner. Certain sensor information always results in a specific action being taken. This can be most simply implemented as a rule-based system, or using finite state machines, and can be summarised as follows:

$$\textit{Current World State} \rightarrow \textit{Action}$$

The use of reactive agents has a number of compelling advantages. These include the fact that they can be implemented extremely efficiently both in terms of memory usage and processing power requirements, and that they require very little support infrastructure, such as the maintenance of a knowledge base. Finally, reactive architectures are completely deterministic, making comprehensive testing straightforward.

Unfortunately, reactive architectures also have a number of severe drawbacks. Firstly, every possible situation an agent might find itself in must be encoded within a system’s rules. This allows for no learning or adaptability, and places a huge burden of responsibility on agent designers, as they must allow for every eventuality. Also, for complex environments the range of possible situations can be vast, making the design of a reactive system extremely difficult, if not impossible. For example, a behaviour based

dialog system presented in [Hasegawa et al, 1997] required a separate behaviour for every possible utterance by every possible speaker.

Finally, reactive systems are not capable of any kind of long term planning. Reactive agents have no internal model of their world and so are incapable of reasoning about it in any abstract way. This greatly limits reactive agents' ability to pursue goals that stretch for any length of time.

3.2.2 Deliberative Agents

Built upon symbolic AI techniques, deliberative agents build internal models of their world, which they then use to formulate plans to achieve goal states. This can be summarised as:

$$\textit{Current World State} + \textit{Goal State} \rightarrow \textit{Plan}$$

Rao and Georgeff's Belief, Desire, Intention (BDI) architecture [Rao & Georgeff, 1991; Rao & Georgeff, 1992] is considered the definitive deliberative agent implementation. The ability to form long term plans is the main advantage of deliberative systems. They do, however suffer from the complexity involved.

As plans are typically formed using computationally expensive logic based inference, a deliberative system cannot make the real-time guarantees required by many application areas. Another serious drawback to the use of deliberative agents, is that they require constant maintenance of a knowledge base. In fast moving, dynamic environments, this can be a major issue as the consistency of the system must be maintained, often involving updating existing inferences based on newly acquired knowledge.

A number of efforts to overcome the difficulties associated with using deliberative agents in dynamic environments have been undertaken. One such piece of work is the Excalibur project [Narayek , 2001], in which the notion of *anytime agents* is introduced. These are deliberative agents capable of forming plans iteratively. For each processing slice given to a particular agent, as much of the current plan as possible is inferred. When this time slice expires at least some action is taken by the agent. For subsequent processing time slices allotted to the agent, plans are re-evaluated and improved upon. Anytime agents attempt to overcome some of the drawbacks of purely deliberative agents such as the processing time issue, and the problem of poor performance in dynamic environments.

3.2.3 Hybrid-Agents

Hybrid-agents combine aspects of both the reactive and deliberative approaches, in an effort to benefit from the best features of both. For example, a reactive system can be used to deal with time critical behaviours such as collision avoidance, while long term planning can be achieved using a deliberative system. The synergy of two sub-systems, however, leads to the introduction of another problem, that of mediating between them. *being-in-the-world* [DePristo & Zubek, 2001] is an excellent example of a hybrid-agent based system which was created for use in games.

3.3 Intelligent Agents as Virtual Humans

The remainder of this chapter will focus on how intelligent agent techniques have been applied to the control of virtual humans. Virtual humans are used in a wide range of application areas including education, engineering, military training, architecture and, the focus of this work, computer games. The simulation of virtual humans is a more established research area than that of game-AI, and so it is useful to take this more broad focus in this section.

Unfortunately, like the definition of agency itself, there is an amount of confusion as to what properties agents should display when being used to control virtual humans, mainly due to the wide range of application areas to which they are applied. To illustrate this, the next section will explore some of the requirements made by system developers and outline the issues unique to using an intelligent agent based approach for this purpose. Following this, the *spectrum of agents* will be introduced as a means to arrange the various virtual human control systems which exist. Finally, the notion of *virtual fidelity* will be considered and a defence of the decision to create a bespoke agent architecture for the control of game NPCs will be made.

3.3.1 What is Required by Intelligent Agents as Virtual Humans?

Using intelligent agents to control virtual humans gives rise to a range of unique requirements. The key to understanding these is to realise that the goal in designing agents for this purpose is typically not to design the most efficient or effective agent, but rather to design the most interesting character. When creating virtual humans, designers are concerned with maintaining the *illusion of believability*. This refers to the fact that the user of a system must be able to believe that virtual humans are living characters with goals, beliefs, desires and, essentially, lives of their own. Thus, it is not so important for a virtual

human to always choose the most efficient or cost effective option available to it, but rather to always choose reasonable actions and respond realistically to the success or failure of these actions. With this in mind, some of the forerunners in virtual human research have the following to say about the requirements of intelligent agents as virtual humans.

Loyall [1997] writes: “*Believable agents are personality-rich autonomous agents with the powerful properties of characters from the arts.*” Coming from a dramatic background it is not surprising that Loyall’s requirements reflect this. Agents should have strong personality and be capable of showing emotion and engaging in meaningful social relationships. To be more succinct (and borrow from Disney [Thomas & Johnston, 1995], who are experts in endowing artificial characters with realistic characteristics) agents should present the *illusion of life*.

According to Blumberg [1996], “*...an autonomous animated creature is an animated object capable of goal-directed and time-varying behavior.*” The work of Blumberg and his group is concerned with virtual creatures rather than humans in particular, and his requirements reflect this. Creatures must appear to make choices which improve their situation and display sophisticated, and individualistic movements.

Hayes-Roth and Doyle [1998] focus on the differences between “*animate characters*” and traditional agents. With this in mind they indicate that agents’ behaviours must be variable rather than reliable, idiosyncratic instead of predictable, appropriate rather than correct, effective instead of complete, interesting rather than efficient, and distinctively individual as opposed to optimal.

Perlin and Goldberg [1996] concern themselves with building believable characters “*that respond to users and to each other in real-time, with consistent personalities, properly changing moods and without mechanical repetition, while always maintaining an author’s goals and intentions.*”

In characterising believable agents, Bates [1992] is quite forgiving requiring “*only that they not be clearly stupid or unreal.*” Such broad, shallow agents must “*exhibit some signs of internal goals, reactivity, emotion, natural language ability, and knowledge of agents...as well as of the...micro-world.*”

Finally, Foner [1993] makes the point that it is important to choose a setting in which users’ expectations for an agent can be met. An interesting corollary to this is that it has been shown that when agents are given a graphical representation, users expectations of the agents’ behaviours alter in relation to the complexity of this representation. Thus, human-

like agents are expected to display human-like intelligence, while users are much more forgiving towards agents given more stylistic representations [Ruttkey et al, 2002].

Considering these definitions [Isbister & Doyle, 2002] identifies the fact that the consistent themes which run through all of these definitions match the general goals of agency. Virtual humans must display autonomy, reactivity, goal driven behaviour and social ability.

It is important to note that when considering the development of virtual humans, the concept of social ability is quite different from what is implied by this property in most other spheres of agent research. In most agent research social ability refers to the ability of agents to communicate with other agents - usually in an attempt to share knowledge or to create multi-agent plans. However, although this can also be the case for virtual humans, usually the property of social ability is more closely aligned with what is considered social ability in the real world. Virtual humans must be able to engage users in interesting and entertaining interactions, and to give the appearance of engaging in such interactions with each other. Also, virtual humans must be capable of maintaining relationships with both the user and other agents, for example is the user a close friend or stranger? This last requirement is very well illustrated by the following quote from [Coco, 1997]:

"The fact is, in most computer role-playing games, the [agents] don't give any semblance of intelligence," he continues. "Say you're in a bar and you throw your beer at the bartender one day. The next day you go back, and he's just as happy to see you. That shouldn't happen."

On top of these behavioural constraints, virtual human systems must also adhere to (often severe) practical restrictions. Often it is required that a simulation contain multiple characters and run in real-time on a machine of modest specifications. This is particularly true for NPCs in computer games, which are expected to contain whole populations of virtual humans and run on the average home PC or games console. Typically, AI processing in a computer game receives something in the region of 10% to 20% of the processor cycles available [Woodcock, 2000a], greatly limiting the reasoning powers with which NPCs can be endowed. One way in which this problem can be overcome is the use of *level-of-detail* AI [Wright & Marshall, 2000], a technique which arose from similar work conducted in graphics research. This will be discussed in detail in chapter 5.

Finally, systems developed for the creation of virtual humans must consider the issue of authoring. Most of the systems in which virtual humans are used are created by technical

developers and then handed over to creative talent for content creation. This means that virtual humans are ultimately created by non-technical authors, and so systems should be designed with this in mind.

3.3.2 The Spectrum of Agents

The differences between all of the virtual human systems mentioned previously can be captured on the *spectrum of agents* presented by Aylett and Luck [2000]. This positions systems on a linear spectrum based on their capabilities, and serves as a useful tool in differentiating between the various systems available.

One end of this spectrum focuses on *physical agents* which are mainly concerned with simulation of believable physical behaviour, including sophisticated physiological models of muscle and skeletal systems, and of sensory systems. Interesting work at this end of the spectrum includes Terzopoulos' highly realistic simulation of fish [Terzopoulos et al, 1994] and the virtual stuntman project [Faloutsos et al, 2001], which creates virtual actors capable of realistically synthesizing a broad repertoire of lifelike motor skills.

Cognitive agents inhabit the other end of the agent spectrum and are mainly concerned with issues such as reasoning, decision making, planning and learning. Systems at this end of the spectrum include Funge's cognitive modelling approach [Funge, 1999], and Narayek's work on planning agents for simulation [Narayek, 2001].

Many of the most effective intelligent agent systems inhabit the middle ground of this spectrum. Amongst these are *c4* [Burke et al, 2001], used to great effect to simulate a virtual sheep dog with the ability to learn new behaviours; *Improv* [Perlin & Goldberg, 1996], which augments sophisticated physical human animation with scripted behaviours; and the *Intelligent Virtual Agent* system [Caicedo & Thalmann, 2000], which sits on top of a realistic virtual human animation system and uses planning to control agents' behaviour.

3.4 Virtual Fidelity

The existence of so many agent based systems, all for the control of virtual humans, can be justifiably questioned. The answer, however, lies in the notion of *virtual fidelity*, as described by Badler [1999]. Virtual fidelity refers to the fact that virtual reality systems need only remain true to actual reality in so much as this is required by, and improves, their application area.

[Määttä, 2002] illustrates this point extremely effectively. This article explains that when architecting the environments in which typical action games are set, game designers do not keep the scale to which these environments are created true to reality. Rather, to ease players' movement in these worlds, areas are designed to a much larger scale, relative to character size, than in the real world. However, game players do not notice this digression from reality, and in fact have a negative response to environments that are designed to be more true to life, finding them overly restrictive. This is a perfect example of how, although designers stay true to reality for many aspects of environment design, the particular blend of virtual fidelity required by an application can dictate that certain real world restrictions can be ignored in virtual worlds.

With regard to virtual humans, virtual fidelity dictates that the set of capabilities which agents should display is determined by the application which they are to inhabit. As this work is focused upon the creation of NPCs for character-centric computer games, the capabilities of any intelligent agent system used to drive these characters' behaviours must reflect this. This set of requirements positions the system required firmly towards the cognitive end of the spectrum of agents, with the key requirements being that agents can behave believably in a wide range of diverse situations and display sophisticated social abilities.

3.5 Requirements for Intelligent Agents in Computer Games

Based on the set of virtual fidelity requirements outlined above, and a number of practical restrictions, a unique set of requirements for an intelligent agent system to drive the behaviours of support characters in character-centric games can be drawn up. These unique requirements are as follows:

- Agents should display believable behaviour
- Agents should display believable behaviour across a diverse range of situations, all within a single game
- Agents should be capable of sophisticated social behaviours
- The agent system should perform in real-time
- The system should not have unrealistic processor or memory requirements
- It should be relatively easy for non-programmers to author agent behaviours within the system

It is argued that no system exists which satisfies all of these requirements. This is shown in table 3-1 which lists how these requirements are, and are not, satisfied by some of the most highly regarded intelligent agent systems available for the control of virtual humans. For this reason this work focuses on the creation of a bespoke agent architecture for the control of support characters in character-centric games.

Table 3-1: The unique set of requirements for support character NPCs in character-centric computer games, and how existing systems compare in terms of which requirements are fulfilled

	Game Agent Requirements					
	Believable Behaviour	Behaviour In Diverse Situations	Sophisticated Social Abilities	Real-Time Performance	Resource Inexpensive	Easy Authoring
Soar	✓	✗	✗	✓	✗	✗
Improv	✓	✗	✗	✓	✗	✓
Oz	✓	✗	✓	✗	✗	✗
Intelligent Virtual Agent System	✓	✗	✓	✓	✗	✓

3.6 Situational Intelligence and the Proposed Agent Architecture

The agent architecture proposed in this work is based on the notion of *situational intelligence*. This concept, the use of which in game-AI is unique to this work, takes the stance that at any time characters need possess only enough intelligence to deal with their current situation. So, if a character is found in a restaurant situation it should possess the capability of displaying believable behaviour within that situation, and no more. If the character’s situation changes (for example if they leave the restaurant to catch a bus home) their capabilities should change to reflect this new situation. Designing the system in this way offers compelling advantages in terms of the amount of processing agents must do at any time, and a reduction of the authoring burden on game designers.

The agent architecture can be implemented essentially as a reactive system, as by using the notions of situational intelligence many of the drawbacks associated with this kind of architecture are overcome. Also, the fact that the system is to be used to drive the behaviours of support characters means that the ability to form long term plans is not required, and so further suggests a reactive architecture. Finally, game worlds should be populated by large numbers of support characters, and so the use of overly resource

intensive techniques (such as deliberative inference) is not realistic. The next chapter will describe this architecture in detail.

4 The Proactive Persistent Agent Architecture

This chapter will describe the *proactive persistent agent* (PPA) architecture which has been developed for the control of support NPCs in character-centric games. The chapter will firstly define the terms proactive and persistent, and explain how agents displaying these properties improve the game-playing experience.

Following this introduction, the architecture will be broadly introduced, and the ways in which it satisfies the unique set of requirements outlined in section 3.4.3 will be explained. The remainder of the chapter will then describe each component of the architecture in detail.

4.1 Proactive Persistent Agents

“don’t let monsters outside of the player’s PVS [Potentially Visible Set] act up, or most of the interesting things will happen before the player gets there”

This quote comes from a code comment in the Half-Life systems development kit⁷, and represents an attitude behind most current game design. The essence of this is that games are designed only to do interesting things in the areas of the game environment in which players are currently active. Although this is acceptable for many games, for character-centric games it is not. If the high levels of player immersion to which character-centric games aspire are to be achieved, they must be set in environments in which NPCs appear to live virtual lives even when not directly involved with a player.

One might well ask, do today’s games not already offer this kind of environment? Although current RPG and adventure game worlds are populated by large numbers of NPCs (in fact the number of NPCs with which a player can interact is often a major game selling point), these characters are typically not modelled within the game world until a player arrives at their location. Once a player arrives at an NPC’s location, the character will either wait for the player to involve them in some interaction, or play through a scripted sequence. This work seeks to overcome these limitations through the use of game characters that are *proactive* and *persistent*.

These characters are proactive in the sense that they act under their own volition. NPCs pursue actions based upon personal goals and motivations. As they do this irrespective of their involvement with player characters, proactive NPCs lend an extra degree of believability and immersion to a game world, furthering the illusion that it is populated by characters with lives of their own.

Out of proactivity comes the need for persistence. This means that NPCs are modelled within a game world at all times - at least to some extent - regardless of their locations relative to players. To illustrate the need for persistence, a simple game world made up of four rooms, and inhabited by three NPCs and a player character can be considered. Figure 4-1 illustrates this world in its initial state. As the simulation of the game world progresses the player moves between its various rooms, as do some of the NPCs. These movements are tabulated in table 4-1.

⁷ The Half-Life systems development kit is available at www.planethalflife.com/half-life/files/

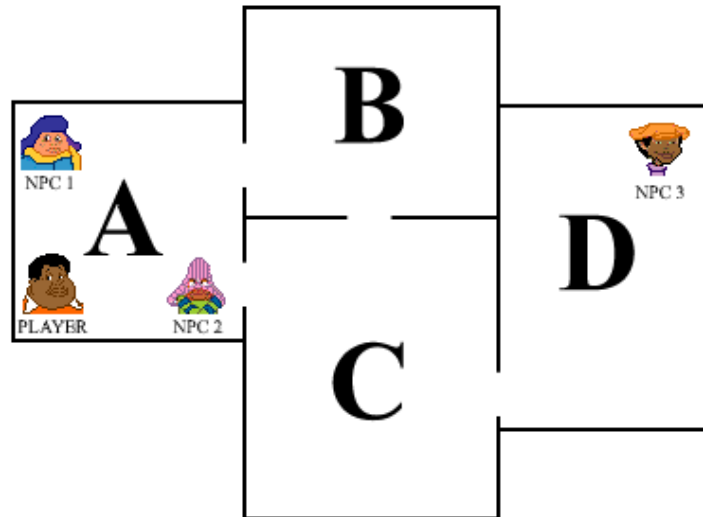


Figure 4-1: A toy game world containing four rooms - A, B, C and D - inhabited by four characters - Player, NPC-1, NPC-2 and NPC-3

If the NPCs in this world were implemented without persistence, then only NPCs that are in the same room as the player would be modelled at any time. In table 4-1 the time periods during which particular NPCs would not be modelled are shown un-shaded. However, a difficulty arises with this scheme when NPC-1 is considered. During the first time period NPC-1 is in the same room as the player, room A, and so his behaviour would be modelled. During the second and third time periods though, the player moves on to other rooms while NPC-1 remains in room A. Without persistence, the behaviours of NPC-1 would not be modelled in any way during these time periods.

Table 4-1: How characters' positions in the toy game world change over time

		Character Location			
		Player	NPC-1	NPC-2	NPC-3
Time Step	1	A	A	A	D
	2	B	A	B	D
	3	C	A	C	D
	4	A	A	A	D

X	Must be modelled
X	Need not be modelled

In the fourth time period the player returns to room A, and so, the behaviours of NPC-1 would be modelled again. It is here that the lack of persistence becomes apparent, as there would be no notion of what NPC-1 was doing during time periods two and three. Similarly, even though in the situation shown the player never encounters NPC-3, this may occur in the future, and so a model of what NPC-3 has been doing during this time must be maintained.

The Proactive Persistent Agent Architecture

Theatre and film script-writers consider the moments before and after a character appears in a scene as crucial, as they establish the motivations behind the character's appearance, and give them a reason for leaving. Traditional game agent techniques ignore these two notions completely. Using characters that are proactive and persistent, on the other hand, will allow for continuous modelling of NPCs and so address this problem.

The use of persistent characters could also change the dynamics of role-playing and adventure games. Players could be faced by opponents pursuing the same, or contrary goals to them. As a player explores a game world they might hear of their opponents actions by questioning other NPCs. A game would then become akin to a race in which the player attempts to achieve their goals before they are foiled by their opponents.

As an interesting aside, the question of persistence can be thought of as a virtual take on the ancient philosophical question: does a tree falling in a forest make any noise if there is nobody there to hear it? Although thousands of years of navel gazing have not been able to answer this question in the real world, in computer games there is no doubt. The tree would make no noise as, if there is nobody there to hear it, a tree would never even fall!

4.2 The PPA Architecture

To create proactive and persistent game characters, an intelligent agent architecture has been developed - the *proactive persistent agent* (PPA) architecture. A schematic of this is shown in figure 4-2.

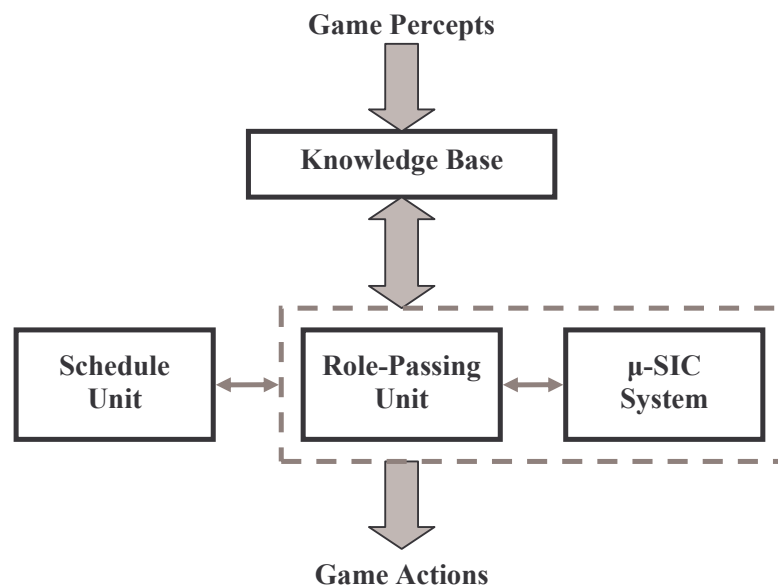


Figure 4-2: A schematic of the proactive persistent agent architecture

The architecture consists of a knowledge base and three key components - the *schedule unit*, the *role-passing unit* and the *μ -SIC system*⁸. The remainder of this chapter will explain each of these components in detail, illustrating how they fit together to create the full architecture. As this discussion continues, it will be shown how the architecture enables the creation of NPCs which satisfy the unique set of requirements outlined in section 3.4.3. During the discussion of each individual component, the manner in which that component fulfils these needs will be highlighted.

4.3 The Schedule Unit

Within a game world, NPCs created using the PPA architecture must give the impression of having a life beyond their association with players. For example, a character might begin her day at home in the morning, then go out to work, visit a bar after work and, finally, return home again. Players should notice that NPCs perform these different tasks, and in this way the illusion that characters have lives beyond their involvement with players is created.

In order to achieve this, each NPC is given a *schedule* which divides a simulation day into a set of time periods, for each of which it is indicated where the character should be and how they should behave. A character's behaviour is determined by an adopted *role*, and these will be discussed in section 4.4. The great advantage of using a schedule is that it endows agents' behaviour with a large degree of believability, without requiring any sophisticated computation. It is also reasonably realistic as, in their day-to-day lives, it is fair to say that people tend to follow similar schedules.

A typical example of a schedule, written in XML (the use of which will be discussed in section 5.3), is shown in figure 4-3. In this example an NPC begins his day at home, assuming the AT-HOME role. This would instruct him to sleep in his bed if he is tired, go to his kitchen to make a sandwich if he gets hungry, or, if he gets bored, go to his living room and watch a little TV. At about 11:00 it is time for the character to go to work, where he assumes the CHEF role, up to about 23:00. The CHEF role places the character in a restaurant's kitchen where he cooks meals as required by his waiting staff. Finally, approaching 23:00 the NPC adjourns to his local bar for a few well earned drinks before retiring home and repeating the process on each subsequent day.

⁸ The name *μ -SIC* does not hold any particular meaning

```

<schedule>
  <scheduleEntry>
    <startTime>02:01</startTime>
    <endTime>11:00</endTime>
    <location>Berlin-Schoneberg-Guest House 0-Living Room</location>
    <role>
      <name>At-Home</name>
      ... <!--Role details omitted to conserve space-->
    </role>
  </scheduleEntry>
  <scheduleEntry>
    <startTime>11:01</startTime>
    <endTime>23:00</endTime>
    <location>Berlin-Kreuzberg-Loebner's-Restaurant-Kitchen</location>
    <role>
      <name>Chef</name>
      ... <!--Role details omitted to conserve space -->
    </role>
  </scheduleEntry>
  <scheduleEntry>
    <startTime>23:01</startTime>
    <endTime>02:00</endTime>
    <location>Berlin-Kreuzberg-Loebner's-Bar</location>
    <role>
      <name>Bar-Patron</name>
      ... <!--Role details omitted to conserve space -->
    </role>
  </scheduleEntry>
</schedule>

```

Figure 4-3: A sample schedule used by a PPA based character

4.4 The Role-Passing Unit

This section will describe the *role-passing unit* of the PPA architecture. The purpose of this unit is, not only to allow NPCs display believable behaviour, but also, within a single simulation, to allow them behave believably in a range of diverse situations. The description of this unit will begin with an explanation of the motivations behind its inclusion, and move on to a discussion of how it operates. The latter will include a discussion of *fuzzy cognitive maps* (FCMs) which are used to control an agent's behaviour

once a particular role has been adopted. Finally, some illustrative examples will be given to illustrate the mechanics of the role-passing process.

4.4.1 Motivations for Role-Passing

In order to satisfy the need for persistence, NPCs controlled using the PPA architecture should be capable of behaving believably in a range of diverse situations. If the schedule shown in figure 4-3 is considered, the character must be seen at home, working in his kitchen and relaxing in a bar, all within the same game. The behavioural requirements of the NPC in each of these situations are extremely different, and the challenge of creating a single control system capable of displaying all of these different behaviours is considered beyond the capabilities of current AI techniques. This is particularly so under the extremely limiting resource usage requirements imposed on game-AI.

Inspired by [Horswill & Zubek, 1999], the technique of *role-passing* begins with a very simple agent capable of assuming different *roles* as it moves between different situations. By assuming a new role, an agent becomes capable of behaving believably in a new situation.

A role can be considered an agent control system suitable to a given situation, throughout which it dictates the agent's behaviour. By assuming and discarding roles as appropriate it becomes possible to create agents which can behave believably in any number of situations. This sits comfortably with the notion of situational intelligence, as agents possess only enough intelligence to behave believably in their given situation.

It is worth noting that role-passing is closely related to the "old AI" idea of Schank's scripts [Schank & Abelson, 1977], a technique used for discourse understanding. This technique involves writing scripts to describe the processes involved in different situations. For example, a restaurant script would indicate that the important actors in a restaurant situation are waiters, customers and chefs; that the important props include tables, menus, and food and that the sequence of events upon a customer entering a restaurant proceeds with the customer choosing a table, the waiter taking their order, the food arriving and so on. Discourse understanding is made easier by the fact that, once the current position in a script is determined, a context for language understanding and the resolution of ambiguity is created. Table 4-2 shows a sample script for a restaurant situation. The role-passing process is related to Schank's scripts in that the behaviour of each agent within a simulation is contextualised by the situation in which they find themselves.

Table 4-2: A sample script for a restaurant situation

Name: Restaurant	Scene 1: Entering
Props: tables, chairs, food, bill, money, tip	Customer enters restaurant
Roles: waiter, waitress, customer, cook, cashier, owner	Customer looks for table
Requisites: customer hungry, customer has money	Customer decides where to sit
	Customer sits down

4.4.2 Fuzzy Cognitive Maps

Once a PPA has adopted a particular role, that agent’s behaviour is controlled using a *fuzzy cognitive map* (FCM). FCMs are based on *fuzzy logic* which, first introduced in the 1960s [Zadeh, 1965], is a superset of conventional logic, extended to handle the concept of partial truth – thus allowing truth values between *completely true* and *completely false*. Although originally developed as a means to model the uncertainty of natural language, fuzzy logic has been applied to a wide range of application areas - finding particular success in the field of systems control, where applications have been as wide ranging as controlling a cement kiln [Holmblad & Ostergaard, 1982] to piloting subway trains in the Japanese city of Sendai [Yasunobu et al, 1983]. A more in depth discussion of fuzzy logic, which broadly follows a similar discussion in [Kosko, 1997], is given in appendix A.

FCMs have their roots in the psychological technique of *cognitive maps*, first described by Tolman [1948] in the 1940s. Tolman used his cognitive maps to describe complex topological memorising behaviours in rats. Axelrod later extended this work, describing cognitive maps as “*directed, inter-connected, bivalent graphs*” [Axelrod, 1976] and putting them to use in decision theory in the politico-economic domain. Finally, in 1986, Kosko introduced the notions of fuzzy logic to Axelrod’s cognitive maps to create FCMs [Kosko, 1986].

An FCM is a fuzzy, signed digraph with feedback. The nodes of an FCM represent labelled causal concepts. These concepts are considered fuzzy sets, and at all times each node in an FCM has an activation in the interval [0, 1]. The arcs of an FCM represent fuzzy rules, or the causal flow between concepts. These arcs are weighted with values in the interval [-1, 1] indicating the degree of causal flow. Figure 4-4 shows one of the earliest FCMs developed by Kosko [1993], which captures the manner in which weather conditions affect a driver’s speed on a California highway.

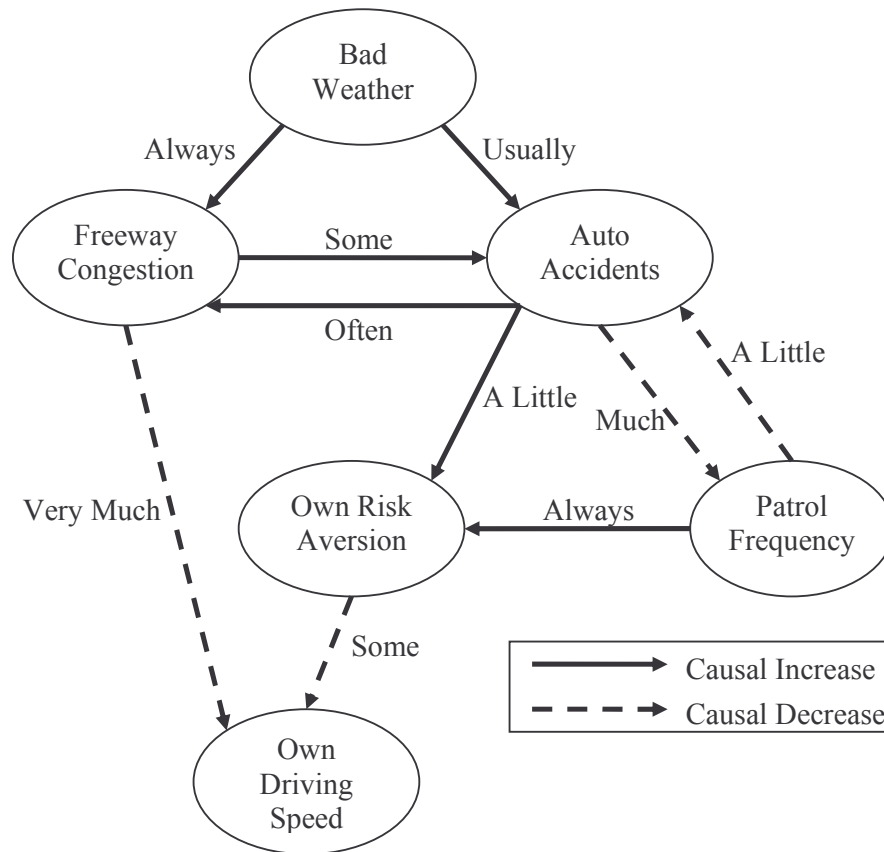


Figure 4-4: An FCM which shows how bad weather can affect the speed at which one drives on a California highway

In this FCM, nodes represent the current weather conditions, the amount of congestion currently on the freeway, the incidence of motor accidents, the frequency of police patrols, one’s personal aversion to risk and one’s current speed. The links in the FCM in figure 4-4 are separated into those that result in causal increase (i.e. increase the activation of the destination node), and those that result in causal decrease (i.e. decrease the activation of the destination node). Furthermore, each link is labelled with a fuzzy weight indicating the degree of this causal effect. One of the advantages of using fuzzy links is that FCM authors can use commonplace terms (such as *always* or *usually*) to describe the degree of causal effect between nodes. These terms will later be converted into values in the range [-1, 1]. In fact, link weights can also be learned, using techniques similar to those used in artificial neural networks. [Buche et al, 2002] showed how this could be done in order to create virtual sheep and a virtual sheepdog.

The links between the nodes in figure 4-4 show how each concept affects the others within the graph. For example, the link between the BAD WEATHER node and the FREEWAY CONGESTION node indicates that when the weather is bad, freeway congestion always increases. Similarly, the link between the AUTO ACCIDENTS node and the PATROL

FREQUENCY node indicates that a high incidence of motor accidents decreases the number of highway police patrols by a significant amount.

An FCM functions as a *fuzzy additive system* in which the inputs to each node are summed to give their activation. The activation of a node gives the system's membership in the *fuzzy set* represented by that node. However, because FCMs allow feedback, activation of nodes is not a single step process, but rather an iterative one. Activation flows through a network until it either reaches *equilibrium* or enters a *chaotic attractor*. When an FCM enters a chaotic attractor the set of concept activations moves chaotically through a small portion of the possible fuzzy set membership space. However, this is relatively uncommon.

An FCM can reach equilibrium either by settling at a *fixed point* within the fuzzy set membership space, or entering a *limit cycle*. When equilibrium occurs at a fixed point the flow of activation completes to give a stable set of node activations. A limit cycle occurs when, rather than the system settling on a single set of node activations, the system continually cycles between a small number of node activation sets.

The purpose of FCMs, as described by Kosko, is to answer “what-if” questions. An FCM can be given a set of initial activations and then observed as it reaches equilibrium. Kosko suggests that FCMs can be drawn for any editorial type article to enable readers to test the theses put forward by the author. This has been demonstrated through FCMs created to describe articles written about the conflict in the Middle East and the drug problem in the USA [Kosko, 1993] amongst others. The FCM in figure 4-4 can be used to test how varying weather conditions affect a driver's speed. By changing the activation of the BAD WEATHER node the manner in which causal activation flows through the graph can be observed, and the resulting activation of the OWN DRIVING SPEED node can be noted.

It is worth noting that an FCM can be considered to be an artificial neural network, the only significant difference being that while most of the nodes in a neural network are given no semantic meaning, in an FCM each node is labelled, thus giving it a strong sense of semantics. In spite of this difference, nodes in both kinds of systems exhibit essentially the same behaviour.

FCMs and Game-AI

Although FCMs have not been used directly in games before, the fact that they are capable of capturing complex systems so well, makes them ideally suited to use in virtual worlds in general. In one example of this, a large FCM was used to successfully simulate an entire

undersea virtual world populated by sharks, dolphins and fish [Dickerson & Kosko, 1996]. However, this approach is not ideally suited to use in games as it does not promote a strong sense of agency, the behaviours of all of the actors within the world being controlled by the same FCM.

An approach which more closely suits the needs of game-AI is that used in [Parentoën et al, 2001]. This work uses FCMs to control the behaviour of individual creatures within a virtual world. Although this is close to what is required by the game-AI field the application areas which have been chosen do not match the complexity of those required by modern games.

A further compelling reason for the use of FCMs in games is that they are extremely easy to author, with most people being capable of drawing an FCM for an area with which they are familiar [Kosko, 1993]. This is ideal for game-AI as it is required that game designers, rather than programmers, populate game worlds with NPCs.

4.4.3 The PPA Fuzzy Cognitive Control System

For the purposes of the PPA architecture, an FCM based system - the *proactive persistent agent fuzzy cognitive control system* (PPAFCCS) - has been created by the author. The PPAFCCS augments the original FCM system with a number of node types which allow the input of activation to the system from agents' personal motivations and perceptions of events in the game world, and for the extraction of agents' chosen behaviours. Along with the original *concept* nodes, the extra nodes used within the PPAFCCS represent *motivations, event occurrences, rules, persistent flags* and *actions*. When these extra nodes are included in graphs they are referred to as *proactive persistent agent fuzzy cognitive maps* (PPAFCMs), rather than as FCMs.

Concepts are at the core of the PPAFCCS, and are the glue that holds PPAFCMs together. Motivations, rules and persistent flags always link exclusively to concept nodes and never amongst each other. Concepts output activation only to other concept nodes, and action nodes. Similarly, action nodes take input only from concept nodes, and very rarely persistent flag objects.

Concept nodes within the PPAFCCS behave in broadly the same manner as those discussed in the previous section. Each concept is labelled and has a set of input links (coming both from other concepts and nodes of the other possible types) the activations of which are summed and then passed onto a set of output links emanating from the node.

Motivation nodes allow the simulation of an agent's internal motivations. A motivation node maintains two response curves [Alexander, 2002a], one which stores the manner in which a motivation rises, and another for the manner in which a motivation falls. These motivations can rise and fall regularly over time (motivations such as hunger behave in this way), or be affected by external events within a virtual world (such as an increase in a fear motivation due to a threatening occurrence nearby). The motivation curves for a basic hunger motivation are shown in figure 4-5.

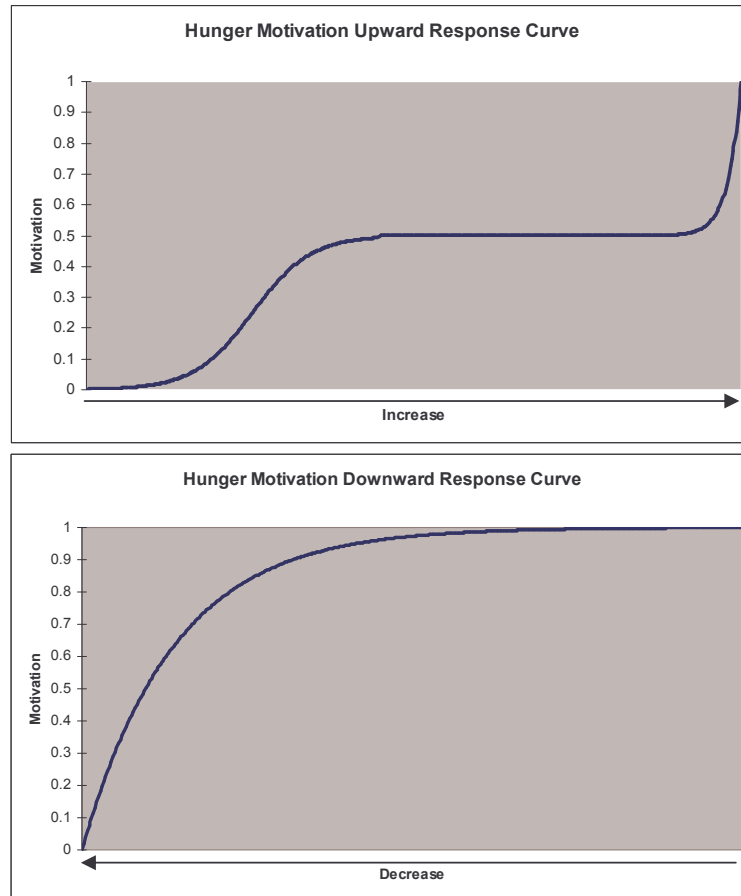


Figure 4-5: Illustrations of the upward and downward motivation response curves used by a basic hunger motivation

Although most motivations simply rise and fall with time, some change in response to external events. Such motivation nodes are closely related to the PPAFCCS's event occurrence nodes. Throughout a simulation, whenever actions take place events are generated, and nearby agents are notified of these. An event occurrence object is created to respond to a particular kind of event, and when such an event occurs an excitation burst is passed to a related motivation object. This excitation burst increases, or decreases a motivation node's value by a specified amount, often related to the event which occurred.

The Proactive Persistent Agent Architecture

For example, the distance from a character to the source of an event could affect the amount by which a motivation is increased or decreased.

PPAFCCS rule nodes allow agents check for the occurrence of particular states within a game world. Each rule node contains an associated rule which is evaluated regularly to determine the node's activation - a function of the rule's degree of satisfaction. Rules can be either Boolean, in which case activation is zero when the rule is not satisfied and one when it is; or continuous, in which case a rule can be partially satisfied, resulting in partial activation of a node. Rules are typically used to check if objects are in particular states, or for the presence of certain characters or character types.

Often it is useful for an agent to maintain knowledge of particular states within a game world which may or may not have been satisfied. These can relate to factors such as whether important tasks have been completed (such as a door being unlocked), or indicate that an agent is in the process of performing some job (for example serving a customer). Persistent flag objects are used for this purpose. Persistent flag nodes can be switched to an on or off state, usually at the beginning or completion of an action, and continually output an activation value of zero when switched off, and one when switched on.

The final node type used within the PPAFCCS is used to represent actions which an agent is capable of performing. Typical actions include using a particular object, moving to a new location and mingling freely with other agents within a character's current locale. Action nodes accumulate the values at their inputs to determine their current activation. At all times within a simulation, all of the action nodes within an agent's PPAFCM are activated, at least to some extent. In order to determine which action the agent should currently pursue, the action nodes are polled and the one with the highest activation is selected.

Within a PPAFCM, links show the causal influence that nodes exert upon each other. Links possess a signed weight which indicates whether a node positively or negatively influences another node, and to what extent. Weights can possess any value within the range [-1, 1]. One extension to the original FCM model is that links also have the ability to threshold activation, and thus only allow activations above a certain level propagate through a network. Finally, the PPAFCCS allows the use of composite links which perform functions such as AND-based or OR-based combination of inputs.

Control links are used together with action nodes within the PPAFCCS. Rather than affect node activations control links are used to carry information around a PPAFCM. For

The Proactive Persistent Agent Architecture

example, an action node could indicate that a particular object is to be used, based on the activation of a rule node. This node must possess a control link to the rule node in question, so that the object to be used can be determined. Similarly, action nodes can affect the state of persistent flag nodes. Flags can be turned on or off at the beginning or completion of an action, and control links are used to indicate this.

Figure 4-6 shows an example of a PPAFCM used to control the behaviour of a very basic agent. In this example, the concepts embodied by the basic agent are FEAR, FATIGUE, BLADDER, HUNGER and SOCIALISE. All of these concepts are driven by motivations, bearing the same names. Except for the FEAR motivation, these motivations increase regularly over time. The FEAR motivation takes burst input from the THREAT EVENT OCCURRENCE node, leading to an agent becoming afraid when ominous events take place.

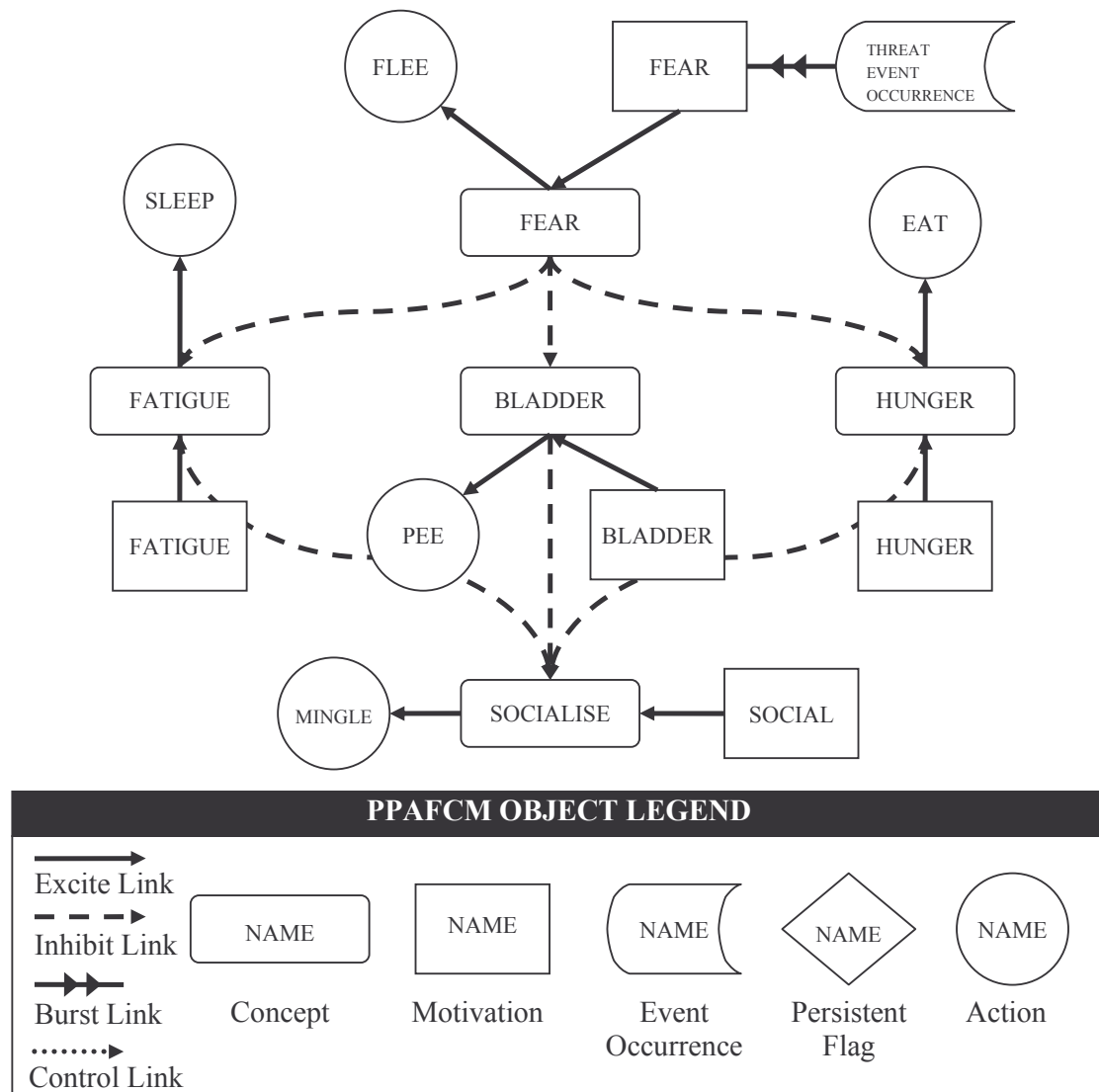


Figure 4-6: A PPAFCM used to control a very basic agent

The Proactive Persistent Agent Architecture

The important concepts in this PPAFCM are SERVE, LOCK BAR, OPEN BAR and NOT IN BAR. The SERVE concept object is activated by a rule object (PUNTER AT BAR) which indicates that there is a customer waiting at the bar. If this is the case, the link to the concept is fully activated, which in turn activates the SERVE action node. The control link between the SERVE action node and the PUNTER AT BAR rule node is important as it allows the behaviour created to use the particular bar object which fired the PUNTER AT BAR rule.

Control links are used throughout this PPAFCM. Both the OPEN BAR and LOCK BAR action nodes are connected by control links to the BAR LOCKED and BAR OPEN flags respectively. The purpose of these control links is to turn on or off the flag on the completion of the action spawned by the action node. In this way, once the bar is locked or opened, the relevant flag adjusts to indicate this.

PPAFCMs differ slightly from the original FCM scheme in the way in which activation moves through a network. Originally, once a node passed activation onto another node, the original node's activation returned to zero. This enables the calculation of node activations to be determined highly efficiently using matrix multiplications. However, in a PPAFCM nodes should not lose all of their activation after affecting other nodes. Rather, nodes within a PPAFCM maintain their activation but use this to affect other nodes. This means that activations are not calculated using matrix maths, but rather using simple summations at each node object.

4.4.4 The Mechanics of Role-Passing

Based on the previous discussions of the schedule unit and the PPAFCCS, this section will describe in detail the mechanics of the role-passing process. When a simulation begins, agents' behaviours are driven only by a very simple PPAFCM. Typically, this is similar to the one shown in figure 4-6, simulating basic drives such as hunger, fatigue and the need to socialise; and allowing agents perform simple actions to satisfy these needs, such as interacting with other agents to sate their social desire. The set of basic concepts can change between simulations, and even between characters within the same simulation. For example, if an NPC smokes this would be captured within its basic PPAFCM.

As well as these basic nodes an agent's PPAFCM also includes a persistent flag node, TIME TO CHANGE ROLES, and a concept object, NOT READY TO CHANGE ROLE. These two nodes are shown in figure 4-8 along with the rest of the basic PPAFCM concept nodes, and their

use will be explained presently. Note that all of the non-concept nodes have been omitted for clarity.

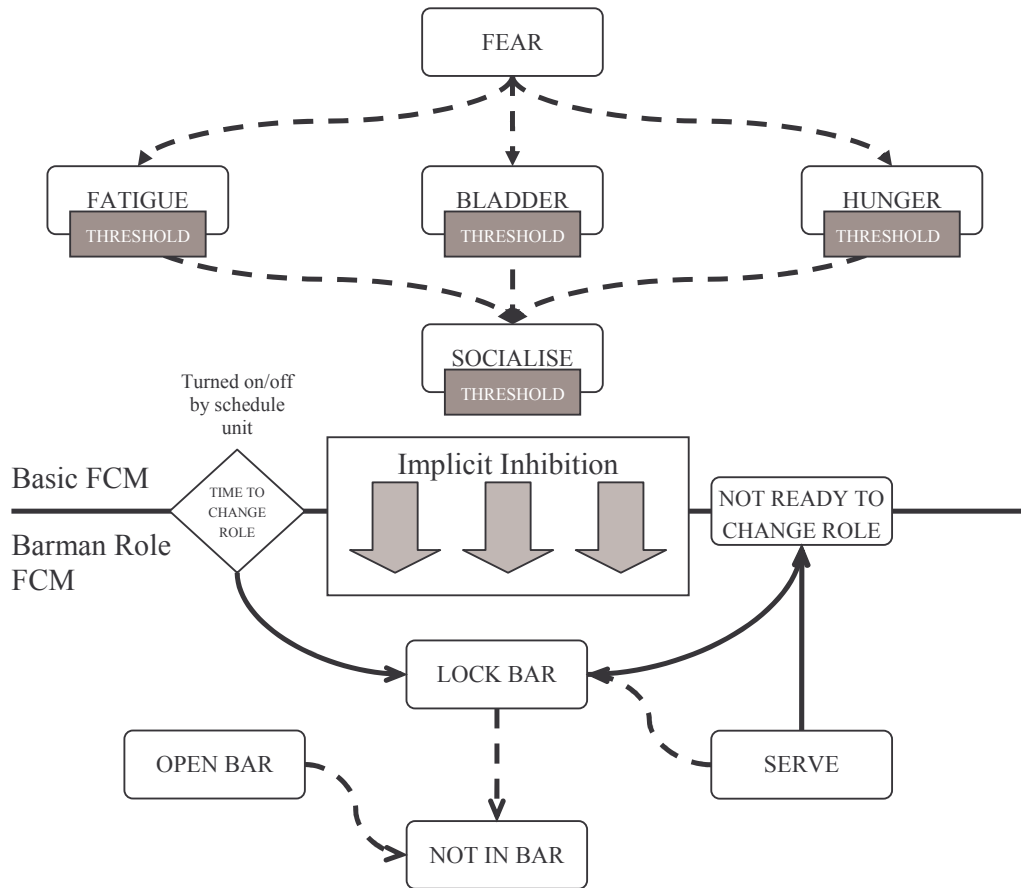


Figure 4-8: An illustration of the complete PPAFCM for an NPC that has assumed the role of a barman. For the purposes of clarity only concept nodes have been included in this illustration

As a simulation progresses, an agent’s schedule unit instructs it to adopt different roles depending on the agent’s changing situation. Adopting a role simply involves adding extra nodes on top of the agent’s basic PPAFCM. When nodes are added to the basic PPAFCM inhibition links are automatically created from all of the concept nodes in the basic map, to all of the concept nodes in the map for the new role. The purpose of these links is to create a simple hierarchy of needs in which the basic needs of an agent are more important than those related to any particular role.

Although it is not done in an rigorous fashion, the inclusion of these links goes some way towards implementing *Maslow’s Hierarchy of Needs* [Maslow, 1970]. This is a psychological theory which suggests that human behaviours are motivated by unsatisfied needs which can be organised in a hierarchy, dictating their relative importance. Figure 4-9 illustrates this hierarchy. Motivations are divided into five groups: *physiological*, *safety*,

love, esteem and self-actualisation. The importance of these motivations decreases as they move up the hierarchy. For example, a physiological motivation such as hunger is given more importance than a self-actualisation motivation such as self improvement.

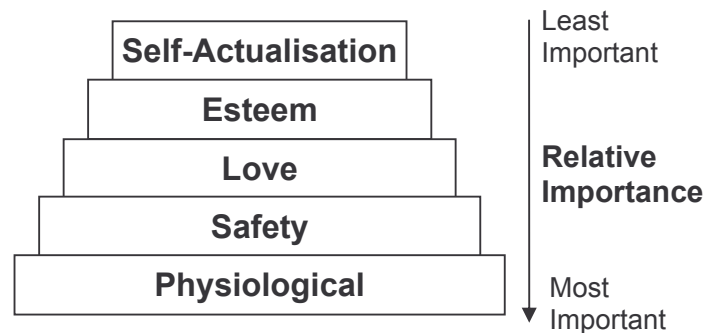


Figure 4-9: An illustration of Maslow's Hierarchy of Needs

Returning to figure 4-8, these implicit inhibitions between basic concept nodes and role related concept nodes are shown as the three large arrows which cross the strong line in the middle of the graph. This line separates the basic agent nodes (shown above the line) from the nodes which are included through the adoption of a role (shown below the line). In this case the agent has adopted the barman role discussed previously. Again, non-concept nodes have been omitted.

The first important feature of this graph is the presence of threshold objects. When a role is adopted it is possible for it to impose thresholds upon links in a basic agent's PPAFCM, thus ensuring that these links do not pass on activation through the map until their own activations reach a sufficient level. This enables realistic behaviour whereby agents are able to ignore basic motivations with low activations in favour of role related concepts which are more important. For example, in the case of the barman role the agent will not abandon the bar every time he feels the first pangs of hunger. Rather, he will wait until he becomes extremely hungry and then briefly step out for something to eat. The inclusion of threshold objects does not run contrary to the ideas behind Maslow's Hierarchy of Needs as eventually the base physiological needs win out. Threshold objects remain valid for the entire duration for which a role is adopted, and so are only used when this is the desired behaviour.

The second key feature of figure 4-8 is the inclusion of the TIME TO CHANGE ROLE and NOT READY TO CHANGE ROLE nodes. These nodes are used to ensure smooth transitions from one role to the next. It would not be believable for agents to change roles on the basis of time alone. If this was the case an agent with an adopted barman role, could be observed midway through pouring a drink suddenly dropping everything and moving away to

The Proactive Persistent Agent Architecture

assume some other role. To avoid this, the NOT READY TO CHANGE ROLE concept node stops the schedule unit from changing an agent's role if it is unsuitable to do so. The schedule unit will not change an agent's role while this node has an activation above zero. Concept nodes associated with a role can activate the NOT READY TO CHANGE ROLE node to ensure that the role is not removed abruptly. In the example in figure 4-8, links exist from the LOCK BAR and SERVE nodes to the NOT READY TO CHANGE ROLE node. These ensure that the agent does not drop the barman role while in the middle of serving a customer, or before he has locked his bar.

The TIME TO CHANGE ROLE persistent flag object has a complimentary purpose to the NOT READY TO CHANGE ROLE concept. This flag object is turned on by the schedule unit when it determines that it is time for the agent to change roles. This allows the agent to perform any tasks which must be completed before dropping the role. For example, a factory worker might clock out, or – as is the case in figure 4-8 – a barman might lock up his bar.

The TIME TO CHANGE ROLE and NOT READY TO CHANGE ROLE nodes are positioned straddling the dividing line between the basic agent nodes, and those of an adopted role. The reason for this is that, although all agents' PPAFCMs possess these nodes at all times, they are never used as part of the basic PPAFCM. Rather, all links to these nodes come from nodes associated with an adopted role.

Eventually, the schedule unit will dictate that each role is swapped out and a new one layered on top of the basic agent PPAFCM. When a role is removed, all of its related nodes are removed, as are all of its links and threshold objects. Figure 4-10 shows the full PPAFCM after the barman role has been swapped out and replaced by a cinema-patron role. This role has just two concepts: OCCUPY SEAT and GET TICKET. The GET TICKET concept takes input from the rule, DOESN'T HAVE A TICKET, which is fired when the agent is not in possession of a cinema ticket, and gives output to an action node (GET TICKET) which causes the agent to use a ticket desk object in order to obtain a ticket. The OCCUPY SEAT concept receives input from the SIT DOWN motivation, and outputs activation to the SIT DOWN action node. GET TICKET negatively influences OCCUPY SEAT, so that the agent does not sit down until he has a ticket.

This PPAFCM also includes links from concepts that are part of the cinema-patron role to those in the basic agent PPAFCM. These links are used for the same reasons that threshold objects are used - to enable role related concepts to override more basic concepts. In this case the desire to sit in the agent's cinema seat should, to a large extent, override any

Role-passing also makes populating a virtual world with agents a straightforward process. Placing agents within novel situations involves simply defining new roles, easing some of the complications involved in attempting to design very general agents capable of behaving realistically in many situations. To design a new role, game designers just draw a new PPAFCM, which as Kosko pointed out, most people should be able to do.

Finally, role-passing moves some way towards creating agents capable of being transferred between different applications. This is a major research area in intelligent agent technology [Aylett et al, 2000].

4.5 The μ -SIC System

One of the key requirements of the PPA architecture is that NPCs be able to engage in believable and interesting social interactions with each other and with players. The success of a number of recent games, in particular the Sims, has shown that meaningful interactions amongst game characters add greatly to the sense of immersion created for players. Social interactions should be driven by characters' moods and personalities, which must be realistically modelled. Based on their interactions, NPCs should also establish and maintain relationships with each other and with players. All of these aspects of a character's persona should become apparent through their interactions and change accordingly, as interactions take place. To these ends the *μ -SIC system* has been created by the author.

The purpose of the μ -SIC system is to choose which social interactions NPCs should engage in when placed within a virtual environment with other characters (both players and other NPCs). When a moment within a simulation arises in which a character is free to interact, the μ -SIC system is queried as to which interaction the character should perform and with whom.

In order to model the chosen aspects of characters' personae (mood, personality and relationships), psychology offers a number of solutions. The next section will discuss which of these have been chosen, and why.

To use these models to drive characters' social behaviours, connectionist AI techniques, and in particular *artificial neural networks* (ANNs), can be used. An ANN can be trained to take a set of values for a character's mood, personality and relationship to another NPC, and from these, determine which interaction, if any, the character should perform. ANNs will be discussed in section 4.5.5, along with their use in the μ -SIC system.

4.5.1 Psychological Models of NPCs' Personae

This section will describe the quantitative models taken from psychology which are used to model NPCs' personalities, moods and relationships. However, before discussing the models themselves, it is worth mentioning the criteria used for their selection. Firstly, the models chosen need not necessarily represent the current state of the art in psychology and cognitive science. The goal of this work is to create characters which behave plausibly at all times within a simulation, and so models which achieve this are enough.

The second important criterion for model selection is that the models used should be as simple as possible. Game designers must be able to use the PPA architecture to add NPCs to their games. If this is to be achieved, models must be simple enough so that designers can understand how they work, and more importantly, how changing a model's parameters might affect a character's behaviour. Related to this concern, the system must be efficient both in terms of memory usage and the amount of computation required, and so the models chosen must reflect this concern.

Personality Model

The first component of an NPC's persona which is modelled by the μ -SIC system is personality. From the myriad schemes which psychologists have developed to model personality, Eysenck's two dimensional classification [Eysenck & Rachmann, 1965] has been chosen. This allows the creation of characters with broad personality types, such as aggressive, sociable and moody.

The Eysenck model plots personality across two orthogonal axes, *introversion-extroversion* and *neuroticism-stability*. From [Lloyd et al, 1984], the extrovert is said to be “*sociable, impulsive and open to new experiences*”, while the introvert is “*quiet, serious and prefers solitary experiences*”. The neurotic is contrasted with a stable person by suffering from tension and interpersonal difficulties. An illustration of this model, which shows the positions of a number of the possible personality types, is shown in figure 4-11.

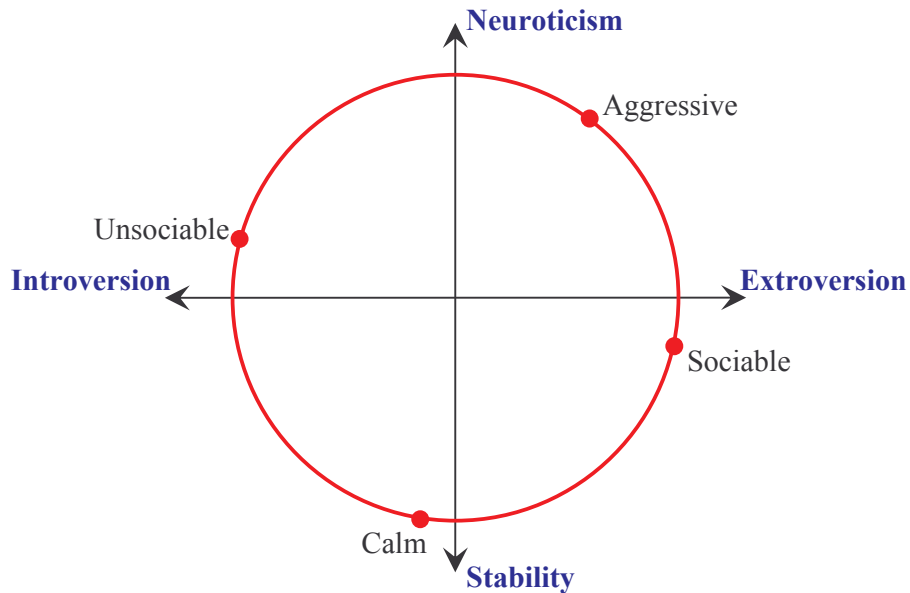


Figure 4-11: The Eysenck personality model which measures personality across the Introvert-Extrovert and Neuroticism-Stability axes. The positions of a number of personality types are shown

It is worth noting that psychologists generally accept that two axes are not enough to accurately model the whole gamut of human personality types. Currently, the most sophisticated models, such as OCEAN [McCrae & Costa, 1996], use five axes. However, the use of more axes is deemed overly complex for the purposes of game-AI, and the Eysenck model is used as it remains one of the most respected and well established personality models in psychological theory [Lloyd et al, 1984].

Mood Model

The second psychological model used in the μ -SIC system simulates a character's mood as it changes over time, through interactions with other NPCs or players. Again simplicity is key, and a model (shown in figure 4-12) which works across two axes has been chosen. An agent's mood is measured according to *valance* and *arousal*, where valance refers to whether the mood is positive or negative, and arousal refers to the intensity of the mood.

This model has been used in computing applications before [Picard, 1995], and is originally due to Lang [1995]. Over the course of Lang's work, this model was used in experiments wherein subjects were shown a number of pictures and their reactions to these pictures plotted according to the two axes. Some of these reactions are shown in figure 4-12.

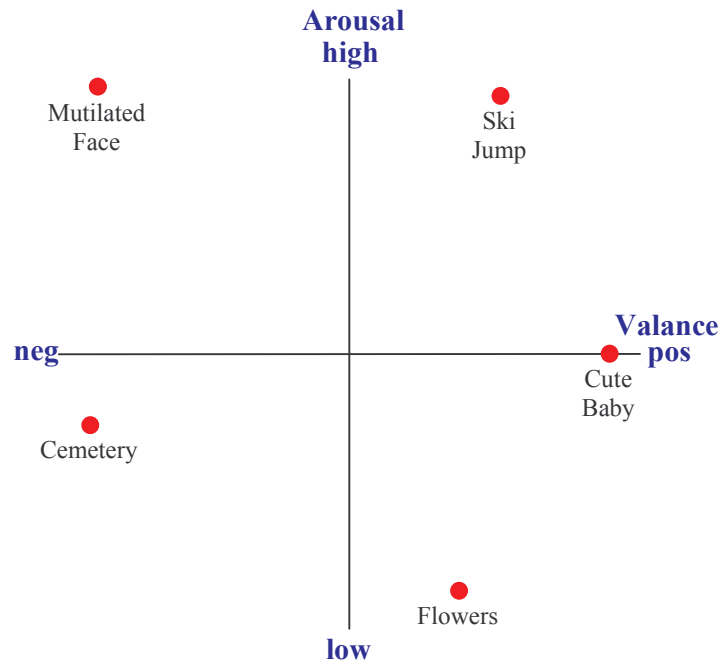


Figure 4-12: The Lang mood model which plots mood according to valance and arousal. Subjects reactions to different pictures are also shown

Relationship Model

The final psychological model used in the μ -SIC system is shown in figure 4-13, and is used to represent NPCs' relationships with each other and with players. The model has been used in a number of other entertainment projects, namely the Oz Project [Scott Neal Reilly, 1996], TALE SPIN [Meehan, 1976], and UNIVERSE [Lebowitz, 1985], and has its psychological basis in [Wish et al, 1976]. Traditionally, four values are used to characterise the relationship of one character to another. These are: the amount that a particular character likes another character, how physically attracted they are to the other character, whether they are dominant or submissive towards the other character and how intimate the first character is with the second.

To facilitate conversation, this model has been augmented with a value indicating how interested one character is in another. Conversation within the μ -SIC system is based on a very simple model in which each character possesses a list of subjects in which they are interested. When characters engage in a conversation they simply pass these subjects back and forth, thus simulating the kind of everyday conversations that people engage in all of the time, for instance in the pub, or over the garden fence. Characters are interested in one another if they share a common interest in a number of subjects, which leads to their becoming more likely to converse in the future.

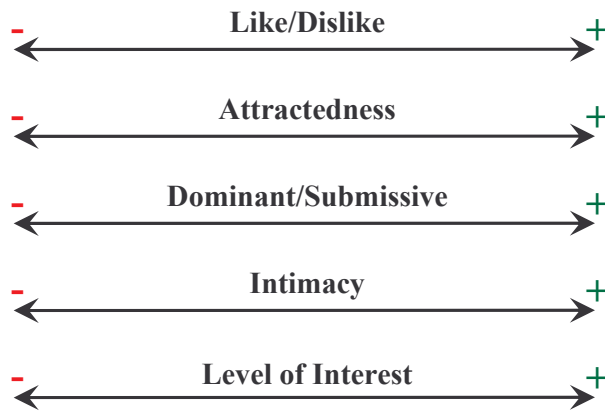


Figure 4-13: The relationship model used which plots a character's relationship to another character

4.5.2 Possible Interactions

Within the μ -SIC system agents are capable of performing a broad set of social interactions. This set was settled upon through experimentation and examination of the social capabilities in existing games, in particular the Sims. This set of interactions and their effects on character moods and relationships are not rigorously based upon psychological theory. The complete set of interactions used within μ -SIC is shown in table 4-3, along with short descriptions of what an interaction entails, and the effect it has upon the character that is the target of the interaction, and their relationship to the interacting character.

Table 4-3: The interactions possible within the μ -SIC system, along with the effects they have upon the target character

Interaction	Description	Effects on Target Character
Kiss	The interacting character kisses the target character.	<i>If attraction is high:</i> Valance+++ Arousal+++ Like/Dislike+++ Intimacy+++ <i>If attraction is low:</i> Valance--- Arousal--- Like/Dislike--- Intimacy+
Flirt	The interacting character flirts with the target character.	Intimacy+ <i>If attraction is high:</i> Valance++ Like/Dislike++ <i>If attraction is low:</i> Valance-- Like/Dislike--
Joke	The interacting character jokes with the target character.	Intimacy+ Valance+ Like/Dislike+

Compliment	The interacting character compliments the target character.	Intimacy+ Valance+ <i>If target is submissive to interacting character:</i> Like/Dislike++
Chat	This simulates general every day talk, or <i>shooting the breeze</i> , and is implemented through a set of subjects of interest which characters pass back and forth. Characters also trade information through the chat interaction.	Intimacy+ <i>If subject is common:</i> Like/Dislike+ LevelOfInterest++ Valance+ <i>If subject is not common:</i> Like/Dislike- LevelOfInterest-- Valance-
Insult	The interacting character insults the target character.	Like/Dislike-- Dominance- Valance-- Arousal+
Assault	The interacting character verbally assaults the target character	Like/Dislike--- Dominance- Valance--- Arousal++

4.5.3 Implementing the μ -SIC System

In order to use this set of psychological models to drive social behaviour, a technique is required which can take the current values of the models, and from these determine whether an interaction should be performed, and if so which one. An ANN was chosen to perform this task. This section will begin with a brief overview of ANNs and then describe how such a network has been put to use within the μ -SIC system.

Artificial Neural Networks and Multi-Layer Perceptrons

ANNs [Russell & Norvig, 1995] are a class of machine learning technique based on the manner in which neurons in biological brains function. An ANN is composed of a large number of highly interconnected processing elements, known as *artificial neurons*, which work together to operate as a function approximator. ANNs, are trained by example and can be used to perform classification tasks in which a set of inputs describing a particular problem case are presented to a network, which then outputs the class of the problem case.

The basic building block of any ANN is the *artificial neuron*, developed in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pitts [1943]. A neuron is composed of a number of *weighted inputs*, a *bias*, a *summation unit* and an *activation unit*. An illustration of a typical artificial neuron is shown in figure 4-14. When a neuron is activated, its weighted inputs and the bias are summed by the summation unit. This sum is then passed to the activation unit and altered according to a *transfer function* (typically a sigmoid function) to determine the activation level of the node.

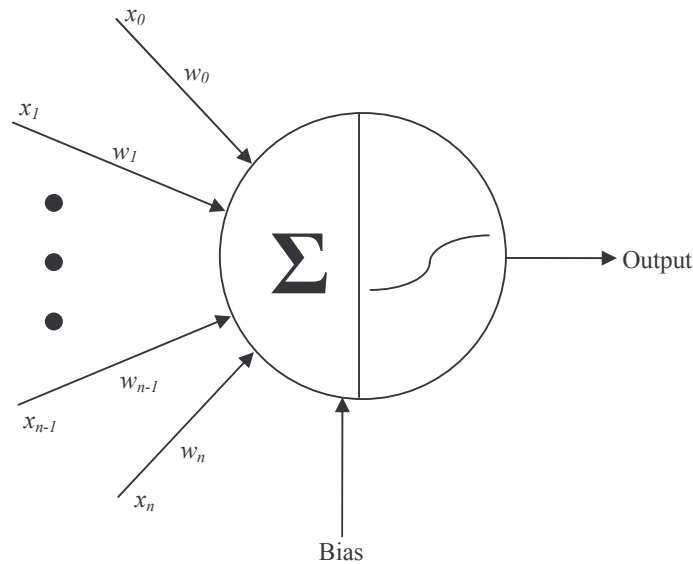


Figure 4-14: The architecture of an artificial neuron

A group of artificial neurons are linked together to create an artificial neural network. Many different ANN architectures have been developed, but this work will focus only on *feed-forward networks*. In a feed-forward network neurons are connected using unidirectional links and cycles are not allowed.

The most well-known form of feed-forward network is the *perceptron*, originally presented in [Rosenblatt, 1957]. The earliest, perceptrons were built with a single layer of artificial neurons, each of which received connections from every input and represented an output class. However these networks could only approximate linearly-separable functions, a drawback famously presented in [Minsky & Papert, 1969], after which research into the area slowed dramatically. This situation was reversed when [Werbos, 1974] presented the *backpropagation of error* algorithm which allowed *multi-layer perceptrons* (MLPs) to be trained to approximate linearly-inseparable functions.

The topology of an MLP is such that it is composed of a layer of input units (the same number as there are features in the data set to be classified), any number of hidden layers of any number of artificial neurons, and an output layer composed of an artificial neuron for each output class. Each of these layers is fully interconnected with the next, but there are no connections between units in the same layer, and no connections skip a layer. Figure 4-15 shows the architecture of an MLP.

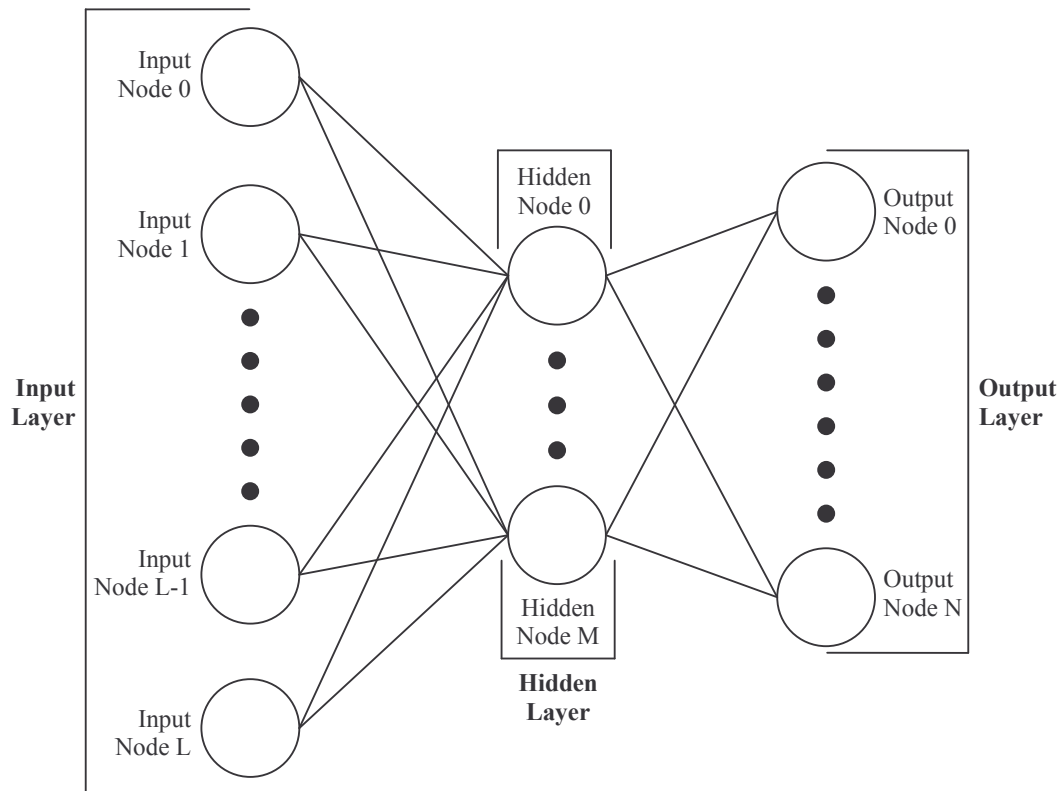


Figure 4-15: An illustration of the basic structure of an MLP. The figure shows an input layer, one hidden layer and an output layer. Networks can have any number of hidden layers

The operation of an MLP is quite simple in concept. When a pattern is presented at the input layer, its feature values are passed through to the hidden layer, the computed outputs of which are passed on to the output layer. The most highly activated neuron in the output layer indicates the class of the input pattern.

As was mentioned previously, MLPs must be trained to perform classification. Training an MLP involves repeatedly presenting a set of known training examples of the classes in question to the network and adjusting the weight and bias values to ensure that the network makes correct classifications. The aforementioned backpropagation of error (or more succinctly Backprop) algorithm [Bishop, 1995] is the most popular choice for training MLPs, and is used in this work.

MLPs in the μ -SIC System

Within the μ -SIC system an MLP is used to determine what interaction, if any, an NPC should perform with another given NPC. The structure of this network is shown in figure 4-16. The network's input layer has nodes for the personality and mood of the character who is attempting to instigate an interaction, and their relationship to a character with

whom they are considering an interaction. The output layer has nodes for each of the possible interactions which the characters engage in. The network shown uses just a single hidden layer, composed of six neurons.

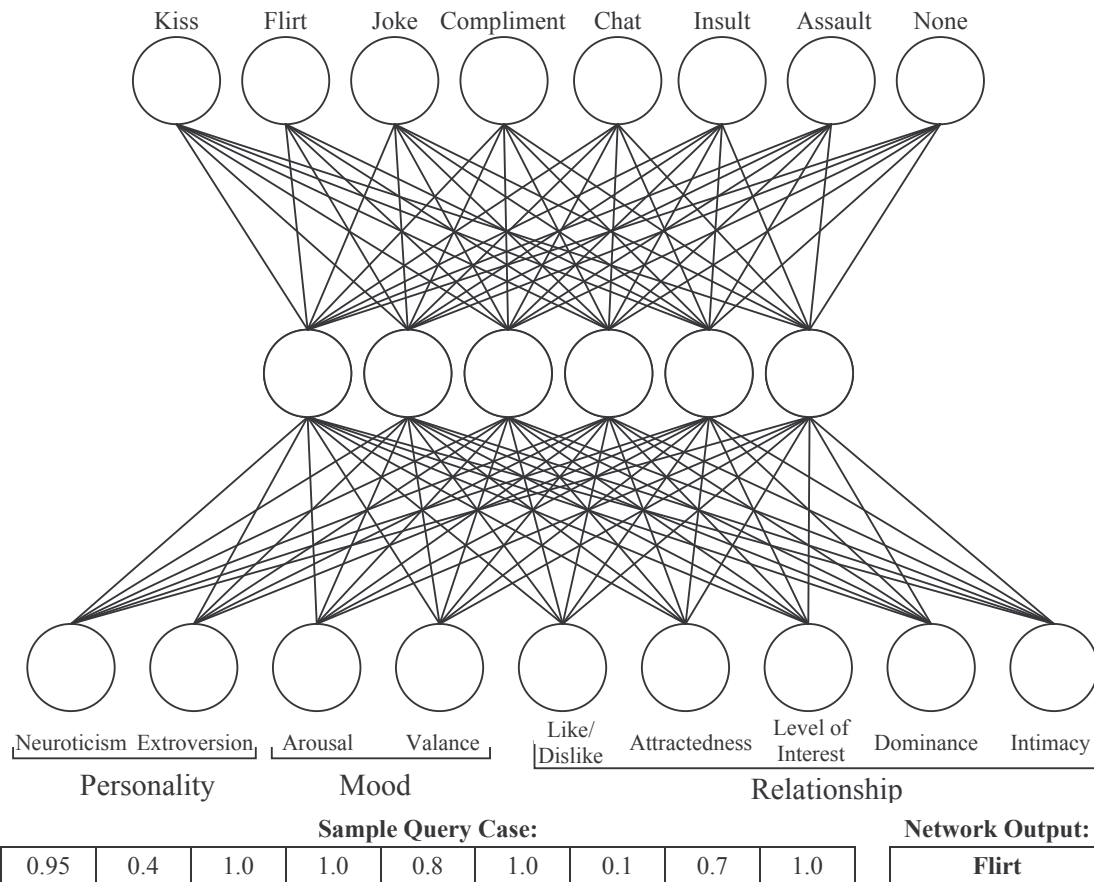


Figure 4-16: The structure of the ANN used within the μ -SIC system along, with a sample query case and result

In order to train any ANN, a data set describing the problem space must be acquired. Data acquisition is often a difficult problem in machine learning tasks, and this is particularly so for the μ -SIC system, as there are no databases available containing information on how people interact. Because of this, an artificial data set was created with which to train the μ -SIC system. For this purpose, a number of simulation situations were created and populated with characters whose personalities were set using the Eysenck model. Relationships between these characters were initialised and a group of people determined which interactions they would engage in as their moods changed over time. This resulted in a hand-crafted data set consisting of approximately 300 data elements.

If an ANN were trained using just 300 data elements it is very likely that the network would *over-fit* to these examples. Over-fitting [Bishop, 1995] is a problem suffered by classifiers in which they too-closely learn the patterns used during training. If a classifier

The Proactive Persistent Agent Architecture

over-fits it will not generalise well to unseen query cases, and so perform poorly. So that this would not occur, the original data set was transformed by adding Gaussian noise to each of the samples to perturb the hand crafted elements, and thus create a set of new data elements. This new data set was composed of approximately 2000 training elements.

After training the network using this data set, its accuracy was determined through a *five-fold cross validation*. Cross validation takes a system's initial training data set and splits it into a number of folds, each time taking approximately 90% of the data for training the network and reserving 10% to test the network's accuracy after training. This is repeated for each fold and the average accuracy across the folds is computed. This way a realistic idea of the network's performance can be determined.

Over the course of cross validation, the network achieved an average accuracy of 85%, indicating that its output was both consistent and coherent. In addition to the cross validation test, the network was also tested on the original hand crafted data set, which was not used during training. The network achieved an accuracy of 84% in this test.

Table 4-4: A confusion matrix showing the results of testing the MLP used in the μ -SIC system on the hand crafted data set

	Kiss	Flirt	Joke	Compliment	Chat	Insult	Assault	None
Kiss	9	0	0	0	0	0	0	0
Flirt	3	2	0	0	0	0	0	1
Joke	1	0	16	0	5	0	0	0
Compliment	0	1	1	2	5	0	0	0
Chat	1	0	1	0	56	0	0	6
Insult	0	0	0	0	0	10	1	2
Assault	0	0	0	0	0	0	6	0
None	0	0	2	0	5	0	0	58

In addition to these high accuracies it is worth noting that when the system produces incorrect predictions, these are rarely significantly wrong. For example, the system may produce a CHAT interaction rather than a FLIRT interaction, but will never produce an ASSAULT interaction instead of a KISS interaction. Table 4-4 shows a confusion matrix for the results produced by the network when tested on the original hand-crafted data set. Each row of this matrix shows the number of predictions of each possible type which were

actually made by the network when the expected type was that of the row's label. For example, the third row shows that in the twenty-two test cases where the expected output interaction was a JOKE, the system output a KISS interaction once, a JOKE sixteen times and a CHAT on five occasions.

4.5.4 Benefits of the μ -SIC System

The ability to perform sophisticated and believable social interactions is one of the key requirements of the PPA architecture. The μ -SIC system satisfies this requirement by driving characters' social behaviours using psychological models of personality, mood and character relationships. Characters are capable of a comprehensive range of interesting social interactions beyond those available in typical commercial games. Characters also possess the ability to build relationships based upon the interactions in which they are involved, thus making future interactions more consistent, and allowing the emergence of groups within the characters in a game world. Finally, through the CHAT interaction, it is made possible for information to flow through a game world as characters interact together, thus allowing characters to appear aware of events which take place around them.

4.6 Knowledge Base

The final unit of the PPA architecture is the *knowledge base*. This allows agents store information about *events* which they have witnessed within the game world, or that they have been told about by other NPCs. The knowledge base also stores a list *facts* of which the NPC is aware. The remainder of this section will discuss the manner in which knowledge of events and facts are maintained.

4.6.1 Events & Character Sighting Records

NPCs are made aware of occurrences within a game world through an event based system. Events are generated every time an action (such as a character entering a room, using an object or interacting with another character) takes place, and all nearby agents are notified. Characters add details of events of which they are notified to their knowledge base. As well as passively responding to events, agents also proactively look around their locations at regular intervals in order to ascertain which other agents are present. A sighting of another character leads to a *character sighting event*.

In order to manage a character's memory of events, the raw events themselves are processed into *character sighting records*. A character sighting record is intended to

The Proactive Persistent Agent Architecture

capture the fact the an NPC will first notice another character on entering a location themselves, when the other character enters the location or when the first NPC notices an event in which the other character is involved (assuming they did not notice the other character enter the location). This character will then be the source of a number of events until either the original NPC leaves the location or the observed character does. This period of time defines a character sighting record which is composed of the name of the character sighted, a start time, an end time, a location and a list of events in which the observed character took part.

In order to facilitate the creation of character sighting records all events of which an NPC is notified are first stored in their memory in a list called the *new event list*. Periodically, this list is processed to create character sighting records which are placed in the *character sighting record list*. Events which are used to create character sighting records are moved to the *processed event list*. It is to this list that links are created to events associated with particular character sighting records. This scheme is illustrated in figure 4-17.

It is possible for NPCs to tell one another (or players) about an event which they have witnessed. This is achieved through the CHAT interactions generated by the μ -SIC system. Periodically, if characters have established a suitable relationship, rather than simply mentioning a subject during a CHAT interaction, a character will instead pass on information about an event. This allows the flow of information through a game world.

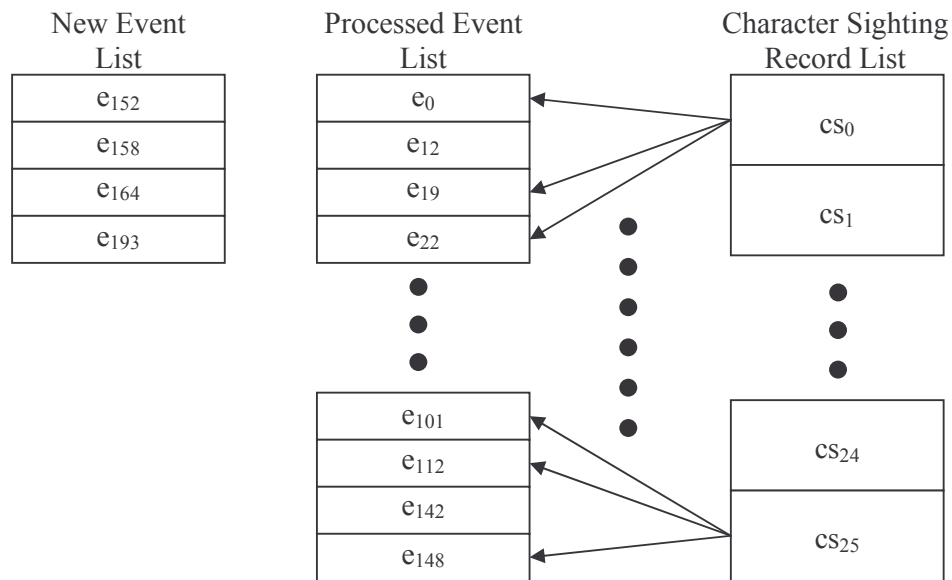


Figure 4-17: The data structures used to allow NPCs store events in their memories

4.6.2 Facts

It is possible for game designers to place information for players into a game world through the introduction of *facts*. A fact is a simple piece of knowledge typically used to indicate information such as where a particular character lives or works, or where an object might be found. Facts are generated by game designers and no new facts are created over the course of a game. Facts are simply stored in a list within characters' memories. Similarly to events, NPCs can inform one another of facts through the CHAT interactions generated by the μ -SIC system.

4.6.3 Benefits of the Knowledge Base

Although the knowledge base in the PPA architecture is extremely simple, it allows agents to store knowledge about the events taking place in the world around them. This imbues agents with an extra degree of believability as it gives the appearance of their being interested in their environment and, as will be shown in section 5.6, the knowledge base can be used to allow players question NPCs about events in a game world, or other NPCs that inhabit the same world. The inclusion of facts also gives game designers a method through which to plant information within a game world in order to offer hints, or distractions, to players. Although this is a satisfactory solution to the problem of allowing agents store knowledge of their world, the system could benefit from a more sophisticated model such as that developed by [Peters & O'Sullivan, 2002].

4.7 Summary & Conclusions

The PPA architecture has been developed to enable the creation of sophisticated, believable and engaging non-player support characters for use in character-centric computer games. These characters must be capable of:

- Displaying believable behaviour
- Behaving believably in a range of diverse situations
- Sophisticated social interactions

The PPA architecture allows the creation of characters which satisfy these needs through its three key components - the schedule unit, the role-passing unit and the μ -SIC system.

Within the role-passing unit the use of FCMs allows agents to behave believably within a given situation, with their behaviour governed by the interplay of motivations, rules and

The Proactive Persistent Agent Architecture

event occurrences. By swapping roles as dictated by their schedule, characters are also capable of behaving in a range of diverse situations within a single game. This promotes the ideas of situational intelligence which are a the core of this work, as characters are only given enough intelligence to behave believably in their current situation.

The μ -SIC system satisfies the need for sophisticated social behaviours. Characters are capable of a comprehensive range of social interactions which remain consistent due to the fact that they are driven by psychological models of personality, mood and characters' relationships. Characters also form believable relationships based on these social interactions, which colour their future behaviours. Finally, the use of the knowledge base allows information to propagate through game worlds as a result of character interactions.

These benefits of the PPA architecture, and related resource implications, will be evaluated in chapter 6. Before this, the next chapter will describe how the architecture has been implemented and used within two test applications.

5 Games-As-Data and the Implementation of the PPA Architecture

Many game development houses use the development method of *games-as-data*. This chapter will explain this idea and discuss how it has been used in the implementation of the PPA architecture. The discussion will also touch upon the use of *level-of-detail AI* and *smart objects*. Towards the end of the chapter, two applications which have been developed in order to demonstrate the capabilities of the PPA architecture will be described. One of these is an adventure game loosely based on the novels of Len Deighton, and the other is the integration of the PPA architecture into a 3-d human animation system.

5.1 Games-As-Data

Within most game development houses, programmers no longer develop the majority of game content. Rather, programmers develop core technologies which are used by game designers in the creation of actual game content. One way in which this division of labour is achieved is through conceptualising *games-as-data*. When games are developed in this way, programmers only work on the creation of the underlying technologies used within a game-engine, while game designers use bespoke tools to define all game content (such as object and character models, sounds, NPCs, locations, objects, animations and game objective data) in data files, which are loaded into a game-engine at run time. This completely separates a game's content from its code-base. The advantages of this approach (which are discussed in [Fermier, 2002]) include:

- That it completely separates the game creation process into those tasks undertaken by programmers and those that are the responsibility of game designers
- That it makes more obvious the set of game creation tools which are required
- That it enables the easy creation of mods by game players
- That it makes updating a game more straightforward, as there is generally no need to write new code, or compile a new code-base

The PPA architecture has been written based upon the ideas of games-as-data. A core set of technologies which drive characters' behaviour has been created, into which details of particular characters, including their schedules and PPAFCMs for each of their roles, are loaded at run time. All character details are stored in data files which are either written by hand or created using custom tools.

5.2 Implementing the PPA Architecture

The PPA architecture has been implemented in such a way as to make its integration into existing game-engines as straightforward as possible. To achieve this, an attempt has been made to separate application specific details (such as characters' appearance and properties used by other technologies such as an animation system) from the implementation of the PPA architecture. Based on this goal the design shown schematically in figure 5-1 was reached. In this scheme the properties embodied in a character object are split into those used by a particular application and those used by the PPA architecture - the latter being referred to as the character's *brain*.

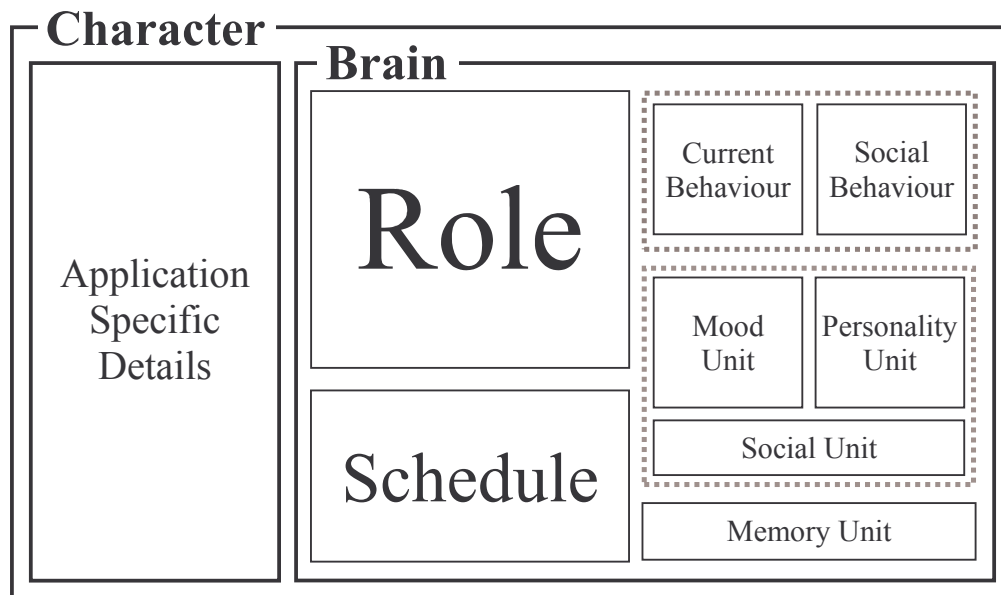


Figure 5-1: An schematic of a PPA character

A character’s brain object primarily stores a character’s schedule, and a role object storing its basic PPAFCM and those of any adopted roles. This role object dictates the behaviour that a character should pursue at any given time, which is stored in the *current behaviour* object within the character’s brain. While performing many such behaviours (in particular those that involve the use of objects) characters can simultaneously mingle with any other characters nearby. For this reason, all agents maintain a *social behaviour* object which can be enabled or disabled as dictated by the role object, and the current behaviour object.

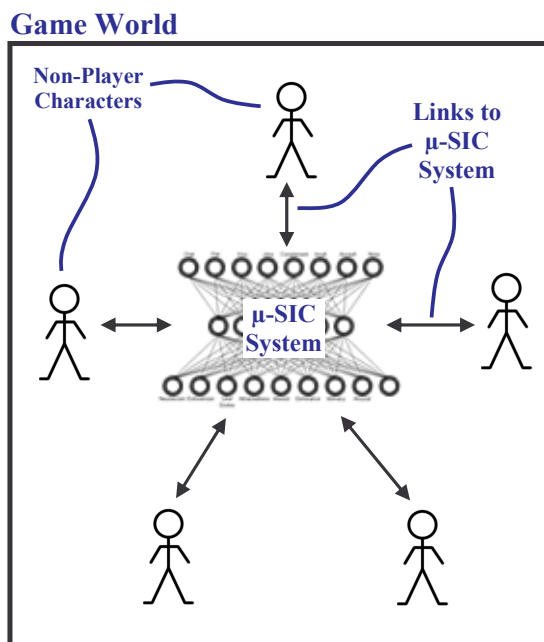


Figure 5-2: How the μ -SIC system is incorporated into a virtual world

Social behaviours are controlled by the μ -SIC system, which is shown in figure 5-1 as the *social unit*. Also shown are the *mood unit* and *personality unit* which implement these aspects of a character’s persona. As well as implementing the μ -SIC system, this social unit also stores details of a character’s relationships with other NPCs and players. Each character’s social unit does not contain a separate copy of the ANN used by the μ -SIC system. Rather, as shown in figure 5-2, each character possesses a link to a single copy of the ANN, which is queried for

every social interaction. Thus, μ -SIC can be considered an oracle advising characters on how to behave. The system is implemented in this way so as to minimise storage requirements, as an individual copy of this ANN for each game character would escalate these beyond what is practical. However, this implementation scheme does not in any way affect the behaviours of the characters implemented using the system.

The final component of the brain object is a character's memory unit, which implements the memory scheme discussed previously. This unit manages a character's recollection of the events and facts of which it becomes aware.

5.3 Languages and Tools

The PPA implementation was written entirely in C⁺⁺, which is the standard language for game development. As the system was written to promote the ideas of games-as-data, the implementation makes significant use of data files. The details of characters' schedules, roles, personalities, initial moods and relationships, and seeded facts and events are stored in a *cast file*, which is loaded into the game-engine at run-time. These data files are written in XML⁹, which is fast becoming the de-facto standard for authoring data files in all areas of computing, including game development (2002's Age of Mythology^{G-28} used XML files for all of its data storage needs [Fermier, 2002]). XML is so compelling as a solution for data storage because, amongst other things, it is highly portable, easy to understand and there exists a wide range of reliable and useful tools to ease the coding burden on the developer.

In the game developed as part of this work, which will be discussed shortly, details of the entire game world were stored in an XML file and loaded into the game-engine at run-time. To enable the creation of these world files a GUI based application, a screenshot of which is shown in figure 5-5, was developed. Game designers could use this tool to graphically design game worlds and populate them with smart objects. This is a typical example of the games-as-data development approach as all game content was developed outside of the game's code-base.

⁹ For more details see www.xml.org

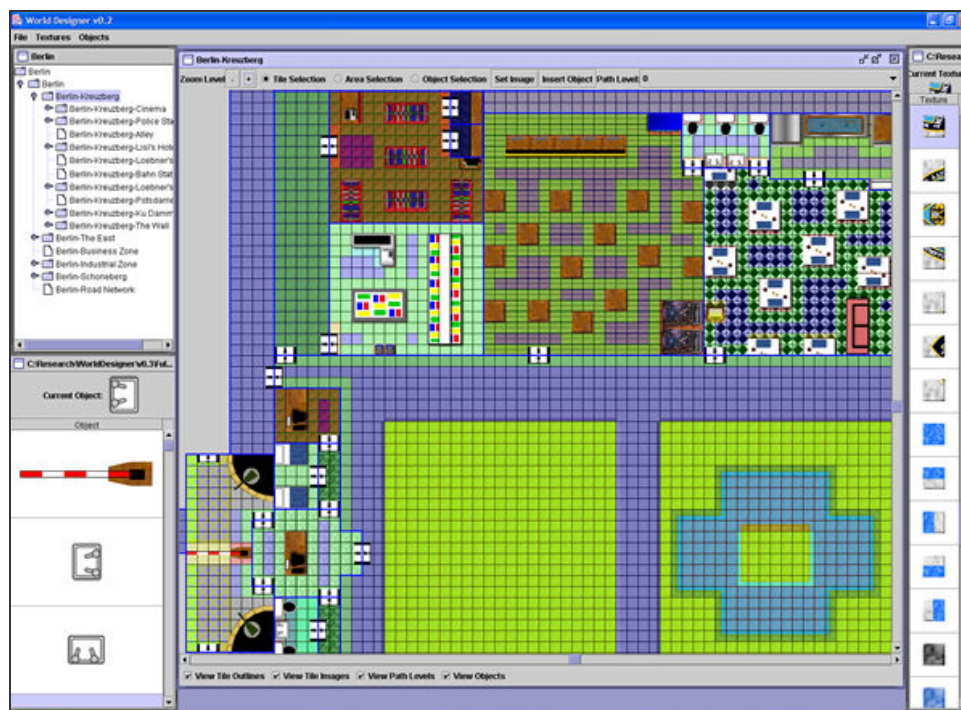


Figure 5-3: A screenshot of the world designer application developed to create virtual worlds for use in Berlin-End-Game

5.4 Smart Objects

Many of the behaviours that PPAs perform involve the use of virtual objects, such as doors, vending machines, chairs, tables and bar-counters. In allowing characters to conduct interactions with such objects, this work uses the approach of *smart objects*, based on accomplished work by Kallman and Thalmann [1998]. The original smart object system has been extended to make objects central to interactions between characters [Peters et al, 2003], thus further promoting the ideas of situational intelligence.

Smart objects are virtual objects that contain more information than just intrinsic object properties, such as position, mass and appearance [Levison, 1996]. This extra information is referred to as an object's *interaction features*, and may vary from what part of a door should be grasped when it is opened, to the fact that an agent's thirst should decrease after using a drinking fountain. Within the smart object model used in this work, objects also store information about the sequences of events which allow multiple users perform interactions based on the use of a single object. For example, a virtual concierge desk in a restaurant environment should indicate the sequence of interactions through which a customer requests a table and has one allocated to them. The smart object should also govern the transfer of information involved in this interaction.

Games-As-Data and the Implementation of the PPA Architecture

The use of smart objects has a number of compelling advantages over more commonplace approaches. Firstly, smart objects comply with the idea of *affordances*, which is an important theory in ergonomics, and in the field of human computer interactions [Preece, 1994]. The theory of affordances states that objects, by their appearance, suggest how they should be used. For example, a door handle by its shape suggests that it should be turned, and is said to offer this affordance. Smart objects behave in essentially the same way, as they inform agents about how they can be used. Secondly, smart objects conform with the idea of games-as-data since objects can be entirely defined in data files, outside of an application's code-base. And finally, the use of smart objects sits comfortably with the notion of situational intelligence, as agents are not burdened with extra information until it is required.

5.5 Level-of-Detail AI

The use of persistence in a game world leads to a resource burden, considerably beyond that created by more typical approaches. This comes from the fact that every NPC within the world is modelled at all times, whereas in more typical approaches this is only the case for those agents in the same portion of the game world as a player. In order to make possible the use of persistence, and still meet the resource restrictions imposed by games, the technique of *level-of-detail AI* is used in the implementation of the PPA architecture.

Level-of-detail AI is based on level-of-detail techniques used in more general simulation, through which areas not currently under the focus of a user are simulated to a lesser degree of detail than those under the user's scrutiny. This approach has been applied to physics simulation [O'Sullivan & Dingliana, 2001], model fidelity [Määttä, 2002] and animation [Carlson & Hodgins, 1997]. As a simple illustration, virtual objects not directly under the focus of a simulation user (for example those that are far away) can be modelled in a less sophisticated way than those to which a user is currently attending, without affecting the overall realism of a simulation. This is the case in figure 5-4 (borrowed from [Schaufler & W. Stürzlinger, 1995]) which shows an illustration of a virtual model of a potted plant at various levels of detail.

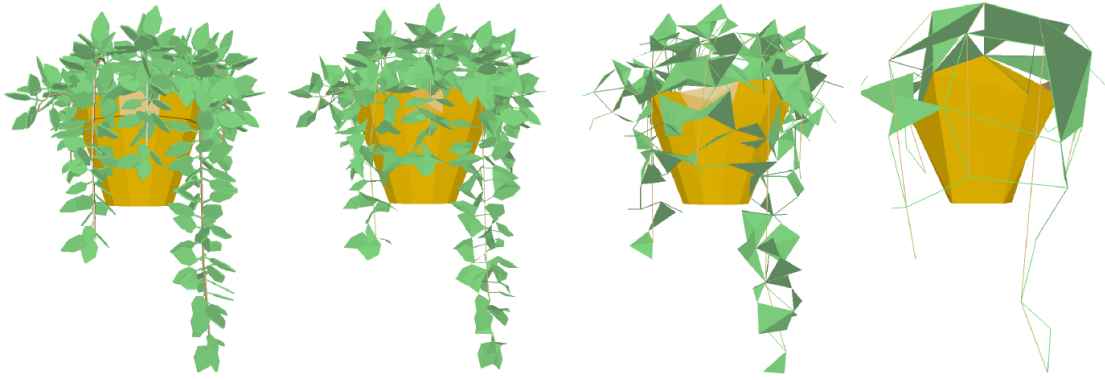


Figure 5-4: An illustration of a virtual model of a potted plant at various levels of detail

This same process can be applied to game-AI. When this is done, characters close to the location of a player, or important to a game's plot, can be simulated to a higher level of detail than those that are not so. This can be easily applied to path-finding, a particularly resource intensive task. The processing burden of a game can be lessened by requiring that only those NPCs which a player can see perform sophisticated path-finding, while those that a player cannot see are free to walk through objects and so on. This is illustrated in figure 5-5.

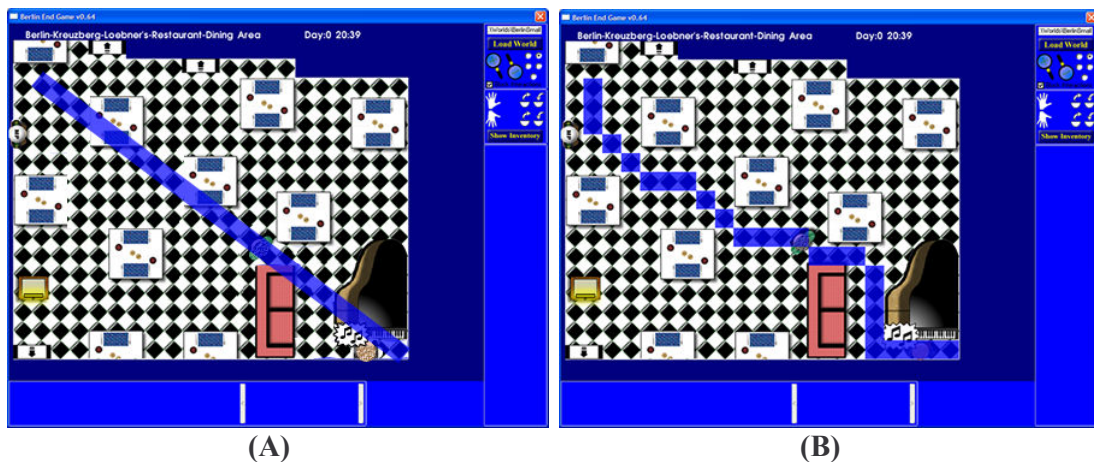


Figure 5-5: An illustration of a path planned across a game location at (A) a low level of detail and (B) a high level of detail

Level-of-detail AI has been used in game applications before [Wright & Marshall, 2000; Brockington, 2002] and is used in the implementation of the PPA architecture. Currently just two levels of detail are used, and at present these only affect path-planning. Characters in the same location as a player are modelled to the highest level of detail and plan sophisticated and realistic paths, while others do not. Level-of-detail AI has only been used to this limited extent due to the size of the environments created thus far, which have not

warranted its further use. It is expected that as the size of game worlds increases, more areas in which this technique can be applied will emerge.

5.6 Berlin-End-Game

In order to demonstrate the capabilities of the PPA architecture, *Berlin-End-Game*, a character-centric adventure game loosely based upon Len Deighton's Bernard Samson novels [Deighton, 1984], has been developed. These novels follow the dealings of British field-agent Bernard Samson in, amongst other places, 1980's Berlin. In the game created, the player takes the role of Samson in a virtual recreation of Berlin, as he tries to acquire a code-book from the East German Stasi agent: Eric Stinnes. The game takes the form of a typical point and click adventure, viewed from above through a cartoon style graphical interface (the creation of which was aided by the use of ClanLib, an open source game development library available at www.clanlib.org). Players are able to navigate the game world, interact with NPCs, use smart objects and collect important pieces of evidence. A selection of screenshots from the game are shown in figure 5-6.



Figure 5-6: A selection of screenshots from *Berlin-End-Game*, a game implemented using the PPA architecture. The screenshots show a hotel, a cinema, Checkpoint Charlie and a restaurant

Games-As-Data and the Implementation of the PPA Architecture

The key to playing Berlin-End-Game is to successfully interact with the game's population of characters, and through these interactions determine how best to locate Stinnes, from whom the code book can be obtained. NPCs can be questioned using a simple series of dialog boxes in order to plumb their knowledge. A screenshot of this system is shown in figure 5-7.

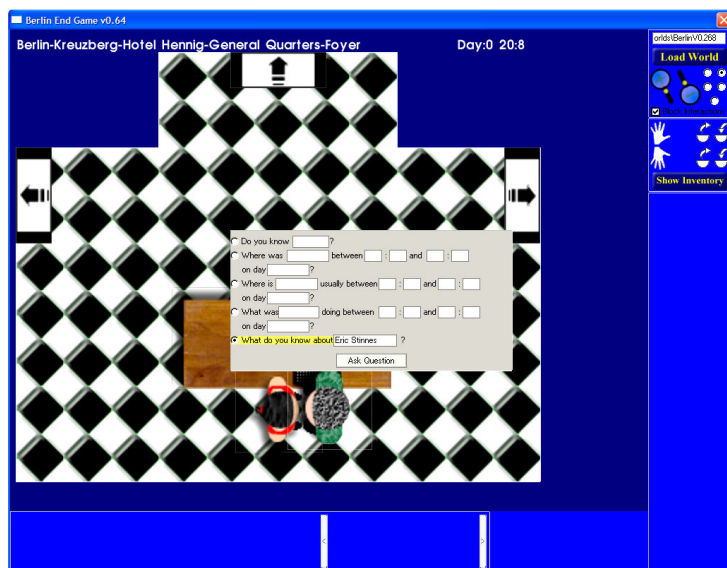


Figure 5-7: A screenshot of the questioning system used in Berlin-End-Game

Berlin-End-Game successfully demonstrated how the PPA architecture can be used within an actual character-centric game. The game was made available on-line within the Machine Learning Group at Trinity College, many of the members of which played it. Berlin-End-Game will be discussed further in chapter 6, during which evaluation experiments that used the game will be described.

5.7 PPAs in the ALOHA System

The *Adaptive Level of Detail Human Animation* (ALOHA) system [Giang et al, 2000] was developed by members of the Image Synthesis Group at Trinity College, Dublin in order to apply level-of-detail techniques to the animation of virtual humans. Previously, the behaviours of characters in simulations developed using this system were entirely scripted, which was time consuming and led to repetitive behaviours. By incorporating the PPA architecture into the ALOHA system these problems were overcome [Mac Namee et al, 2002], as agents could be placed within a situation in which simulations would develop dynamically. Using the two systems together, a simulation of a small portion of a university campus was created. Simulated virtual humans could be observed attending and presenting lectures, relaxing in the college bar and moving between these environments. A

Games-As-Data and the Implementation of the PPA Architecture

number of screenshots of this simulation are shown in figure 5-8. The integration of the PPA architecture into the ALOHA system demonstrated how it could be used in very different applications, and in particular showed how it worked together with a sophisticated animation system. The simulations which were created using the ALOHA system will also be further discussed in chapter 6.



Figure 5-8: Screenshots of the simulation created using the ALOHA system and the PPA architecture

5.8 Conclusions

Many game developers now treat game development as two separate tasks: the creation of technologies to drive a game-engine, and the authoring of actual game content. One way in which to achieve this split is by conceptualising games-as-data. When this approach is taken, game developers code a game-engine, while game designers author game content in data files, which are loaded into the game-engine at run-time. The PPA architecture has been implemented to promote this idea, and also to make its integration into existing game-engines as painless as possible. The use of data files and XML achieves the first goal, while the latter is achieved through the separation of engine specific details from those related to the PPA architecture. To demonstrate its capabilities, the architecture has been

Games-As-Data and the Implementation of the PPA Architecture

put to use in two applications, one an adventure game, and the other a system for the simulation of 3-d virtual humans. Along with showcasing the architecture, these applications served as a test environment for evaluation experiments which will be discussed in the next chapter.

More details of the implementation of the PPA architecture and the applications that use it are given in appendices D to G.

6 Evaluation

Evaluation is a problematic issue in game-AI, and also within the more general field of virtual human research, which has led to its often simply being over-looked. This chapter will begin by considering some of the problems associated with performing rigorous evaluations in these fields. Following this, a scheme which has been used to evaluate the capabilities of the PPA architecture will be described, and this will lead to a discussion of the evaluation process itself, including the results which arose from it.

6.1 Evaluation Problems

Rigorous evaluation is something that has been largely overlooked in research into virtual humans and game-AI. In the first case, the most obvious reason for this is that the simulation of sophisticated and autonomous virtual humans is a relatively new achievement. This means that any system that manages to do it is inherently novel, and as such, can be considered a success without rigorous evaluation. However, as the field has now become quite well established this is no longer acceptable.

In the latter case, the commercial nature of computer games serves as an implicit form of evaluation. If a game is commercially successful, and receives favourable reviews, then its various components (including its AI) can be considered a success. However, if game-AI techniques are developed outside of a commercial release, this form of implicit evaluation is not possible. This is the case with techniques developed for inclusion in SDKs, and is also the case with this work.

Finally, rigorous evaluation of virtual human based systems is extremely difficult and so is often simply ignored. The major question which must be answered is whether the behaviours of the virtual humans created are believable. Unfortunately, by itself this is too subjective a question upon which to base an evaluation. If subjects are simply asked to rate the believability of characters' behaviours in a simulation, their answers would be overly influenced by their familiarity (or lack of same) with the field, and this would skew any results.

However, if advances are to be made in these fields, rigorous evaluations must be performed and new techniques must be compared to existing systems. The remainder of this chapter will describe an evaluation scheme for use in the general field of virtual human research, and more specifically game-AI, and how this scheme has been used to evaluate the PPA architecture.

6.2 Evaluation Scheme

In spite of the difficulties outlined in the previous section, a scheme for the evaluation of the PPA architecture has been developed. This is based on work by Isbister and Doyle [2002] on the evaluation of embodied conversational pedagogical agents. This field has actually been the focus of a number of efforts towards rigorous evaluation [Abbattista et al, 2002; Höök, 2002; Ruttkay et al, 2002], and serves as a model for the more general area of

virtual human research. The key to the Isbister and Doyle scheme is that it splits the evaluation problem into distinct categories, each of which can be evaluated separately. These categories have been slightly adjusted to better suit evaluations in game-AI, and are shown in table 6-1. Alongside each evaluation category, criteria for their success are also shown.

Table 6-1: The categories into which the evaluation of the PPA system has been divided and criteria for their success

Category	Criteria for Success
Believability	Do characters convey the “illusion of life” to players?
Social Abilities	Are NPCs capable of consistent, believable and interesting social interactions?
Application Domain	Does the use of PPAs create a more satisfying and immersive computer game playing experience?
Computational Issues	Does the system perform successfully according to criteria of speed and efficiency?
Production	How easy is it for designers to use the system to create agents for use in their games?

The foremost concern of this scheme is the *believability* of the behaviour of virtual humans, and this is the first evaluation category considered. A system is deemed successful if its characters continuously behave in a consistent and believable manner. This is often referred to as characters portraying the *illusion of life*.

Related to the question of believability, characters should also possess sophisticated *social abilities*. Whether or not this capability is displayed in a believable, consistent and interesting way is the criterion for the second evaluation category.

The scheme’s third evaluation category considers how the use of a virtual human system improves the *application domain* in which it is used. In this case the question asked is whether the use of PPAs in character-centric computer games offers players a more satisfying and immersive game experience.

In this evaluation scheme practical issues such as resource requirements are referred to as *computational issues*. Technologies for use in games must run under particularly severe restrictions. In the case of the PPA architecture games in which it is used must run on typical home PCs or games consoles, and the system’s resource requirements must reflect this.

Finally, comes the issue of *production* which considers the ease with which a virtual human control system can be used in authoring virtual worlds. The criterion for success in this category is the level of ease with which game designers can use the PPA architecture to create NPCs for inclusion in their games.

The remainder of this chapter will consider each of these categories in turn, examining the relevant evaluations performed. By considering each individual category, an overall evaluation of the PPA architecture will emerge.

A number of the evaluations performed took the form of user trials, in which users observed or used a system, and filled out a questionnaire in order to quantify their experiences. However, due to the difficulties associated with performing such trials - both in terms of the time required and the complications of finding suitable subjects willing to take part - this is not possible for every category. For those categories in which user trials could not be performed, illustrative simulation examples will be described in order to show the capabilities of the system, or the performance of the PPA architecture will be compared against that of other similar systems.

6.3 Believability

The central question in the evaluation of the PPA architecture is that of believability. Do agents created using the PPA architecture behave in a believable fashion when they are placed within a simulation? Unfortunately, there is an inherent danger in attempting to measure such a subjective concept as believability, as is outlined in [Slater, 2003] which considers the issues associated with using questionnaires to measure *presence* (a slightly more encompassing notion than believability) in virtual environments. Slater argues that such a subjective measure cannot be successfully measured in this way. The point is illustrated through an experiment in which it is shown that the invented concept of *the colourfulness of somebody's day* can be measured using a questionnaire, and statistically linked to whether somebody got out of bed later that day than usual, or not! Slater suggests that implicit measures, such as physical reactions to a virtual experience or subjects' performance of a given task, would be a less subjective measure.

However, it is not clear how such measures might be used to determine the merit of populating a virtual world with PPA based characters. There are no obvious physical attributes which would indicate an increase in believability, and since the purpose of using PPA based characters is to create better game-playing experiences, rather than allow

players more easily complete a game, there is no obvious task completion which can be measured. For these reasons, the questionnaire is considered the only available way in which believability can be measured.

However, the experiment can still be formulated in as objective a manner as possible. In this case, rather than requiring that subjects view a simulation and comment on the believability of its characters' behaviours, subjects were required to simultaneously view two versions of the same simulation (only one of which used the PPA architecture) and compare the behaviours of the characters in both. In this way some of the dangers in using questionnaires to quantify subjects' experiences were overcome, as an objective comparison can be made between their reactions to the two versions of the simulation. The remainder of this section will describe the experiment performed and present its results.

6.3.1 The PPA Believability Experiment

The *PPA believability experiment* is based upon a simulation of a bar environment, implemented using the ALOHA system. The ALOHA system was used as it offers a much more visually appealing simulation of a virtual environment than that of Berlin-End-Game. The fact that users cannot take an active role within an ALOHA based simulation, was considered less important than the effect appearance has on user expectations of virtual human behaviour. The effect of the aesthetics of a simulation on users' expectations has been explored in [Ruttkey et al, 2002] and, in considerable depth, in [Hodgins et al, 1998] which describes a study of viewers' sensitivities to differences in motions displayed using models of differing complexity. The study showed that subjects were significantly more sensitive to variations in motion performed by a polygonal model of a human than those performed by a human stick figure. Although the authors do not claim that subjects will always be more sensitive to more complex models, their work shows that model selection is an important issue in setting up experiments.

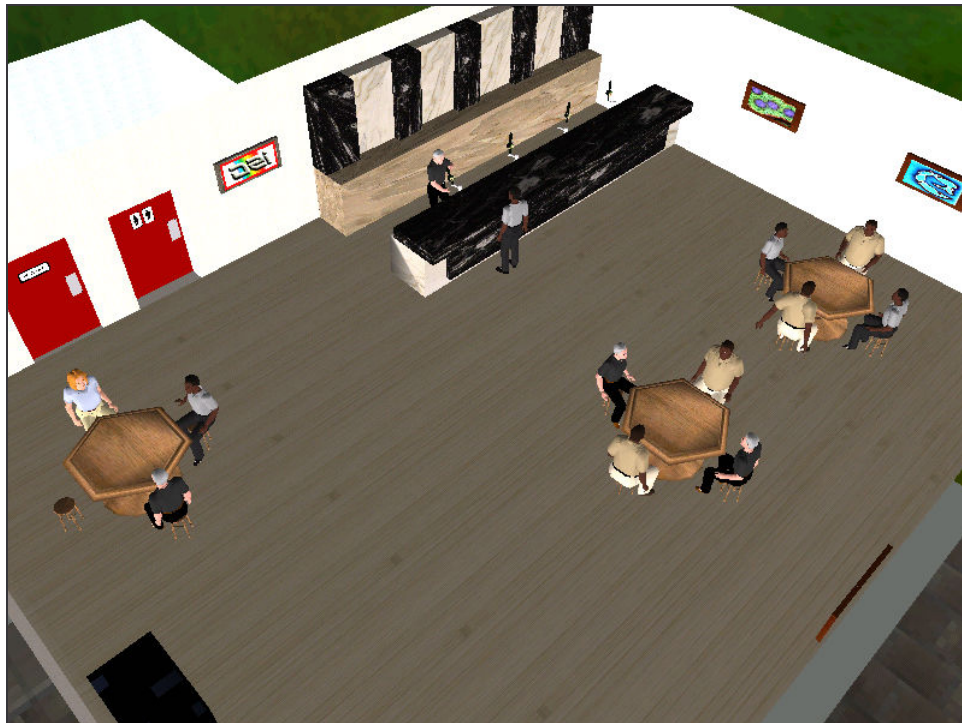


Figure 6-1: A screenshot of the bar environment used within the believability experiment

The bar environment, shown in figure 6-1, was populated with a barman character and a number of customer characters. In the experiment, subjects were simultaneously shown two versions of this simulation, which ran on two separate PCs.

The first of these simulations was populated by customer characters implemented using the full PPA architecture. Each character's behaviours were driven by a PPAFCM possessing a set of basic concepts, and an adopted BAR-CUSTOMER role, which remained in place for the duration of the simulation. The PPAFCM used is shown in figure 6-2. The basic concepts embodied by customer characters were HUNGER, BLADDER and SOCIALISE. Each of these received input from a simple motivation object of the same name, which allowed these three basic needs to adjust based on time and agents' interactions with smart objects, or other virtual humans. Each concept object positively influenced the activation of an action object which caused virtual humans to satisfy these particular needs: use a vending machine to satisfy hunger, use one of the bar's toilets to empty the bladder and interact with other characters in order to satisfy the desire to socialise.

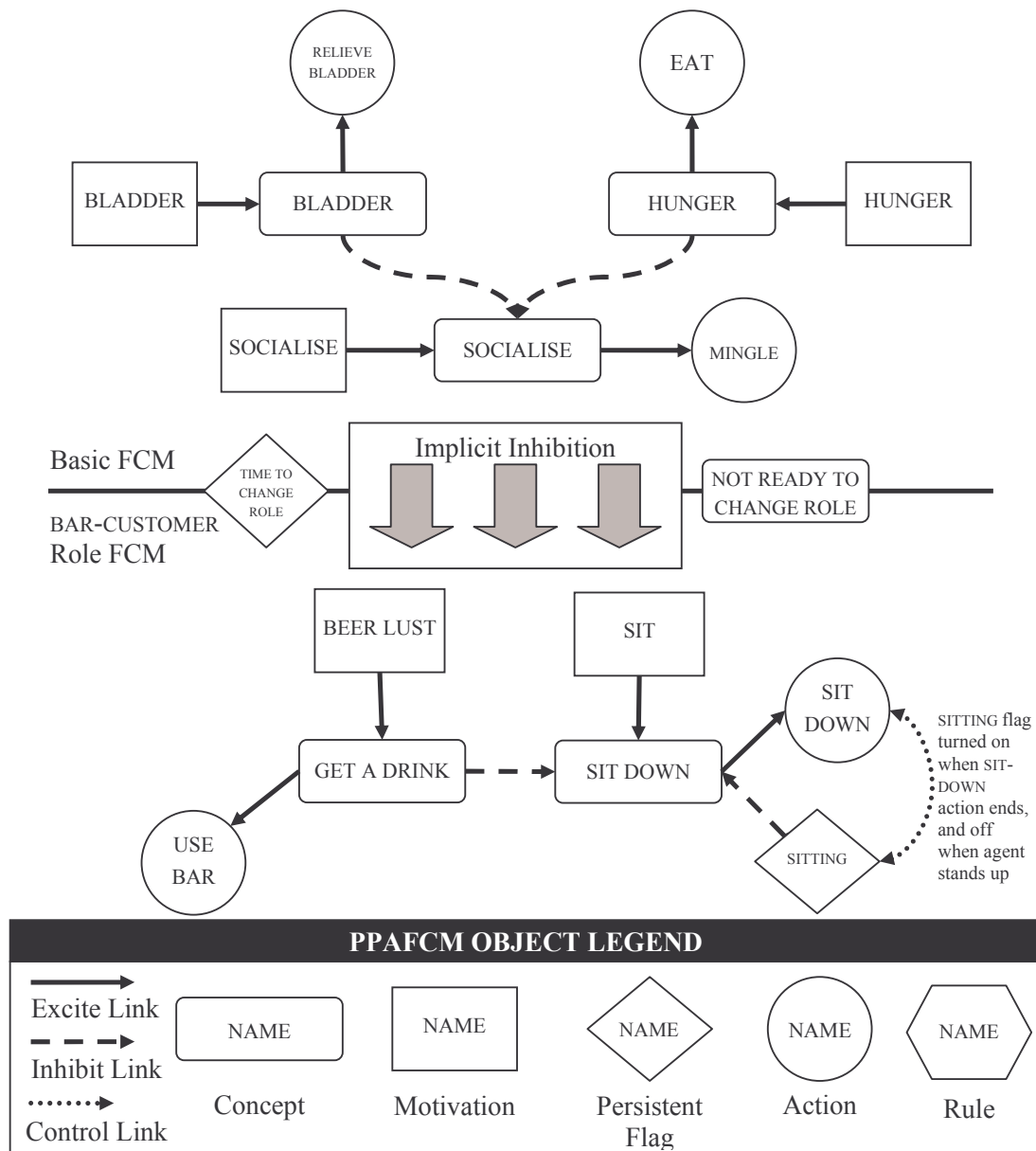


Figure 6-2: The full PPAFCM used for bar customer characters in the PPA believability experiment

The adopted BAR-CUSTOMER role was composed of a GET A DRINK concept and a SIT DOWN concept. By taking input from the BEER LUST motivation, the GET A DRINK concept caused activation of the USE BAR node, which allowed agents use one of the environment’s bar objects to get a drink. Taking a drink caused an agent’s BEER LUST motivation to decrease and, conversely, their BLADDER motivation to increase.

The SIT DOWN concept took input from a motivation object (SIT) simulating an agent’s desire to sit down, and output to an action object (SIT DOWN) causing the agent to sit at one of the bar’s tables. Characters chose which table to sit at based on their relationships with

the other characters already at each table with a free chair. When agents were seated, they were free to interact with the other agents seated at the same table.

The second simulation was populated by customer characters that chose their behaviours randomly from a small set suitable to the bar environment. Agents could sit down at one of the bar's tables and interact with other agents, use the bar to get a drink, go to the toilet or use the vending machine present in the bar to get some food. Periodically agents made a decision to perform a new action. The agents used the bar, went to the toilet or used the vending machine, each with a frequency of one decision in every fifty, and sat at a random free table and interacted with other characters at all other times. The frequency of $1/50$ was reached through experimentation, in order to avoid agents' behaviours appearing chaotic, while at the same time ensuring that they did not appear lethargic. The use of random characters as a benchmark is justified by the fact that support characters in adventure and role playing games are often implemented this way.

In both simulations the behaviour of the barman character was driven using the PPAFCM shown previously in figure 4-8. This allowed the barman serve characters that he noticed waiting at the bar.

Subjects taking part in the experiment were given a short explanation of what was involved (which did not indicate the experiment's goals or include any discussion of what drove agents' behaviours in either simulation), and were then allowed as much time as they required to view both simulations, which ran simultaneously. After this viewing, subjects were required to fill out a questionnaire indicating their views on the simulations shown. This questionnaire, along with a detailed breakdown of subjects' responses, is given in appendix B. A total of 13 subjects were involved in the PPA believability experiment. These had a mixture of backgrounds, with some being familiar with the areas of game-AI and virtual humans, while others were not. The next section will present this experiment's result.

6.3.2 Experiment Results

On the PPA believability experiment questionnaire, subjects were first asked to select the simulation "*in which [they] thought agents' behaviour was most believable*". Just under 77% of subjects found that characters implemented using the PPA architecture were more believable than those implemented to pursue random behaviours. Although this result, shown in figure 6-3, is compelling by itself, it can also be supported statistically. Subjects'

answers to this question can be considered a set of 13 Bernoulli trials in which the PPA based simulation was chosen 10 times, while the random agent based simulation was chosen on just 3 occasions. A hypothesis test was performed to determine whether or not this data statistically supported the hypothesis that the majority of people would find the PPA based simulation the most believable. This test supported the hypothesis with a confidence of over 95%.

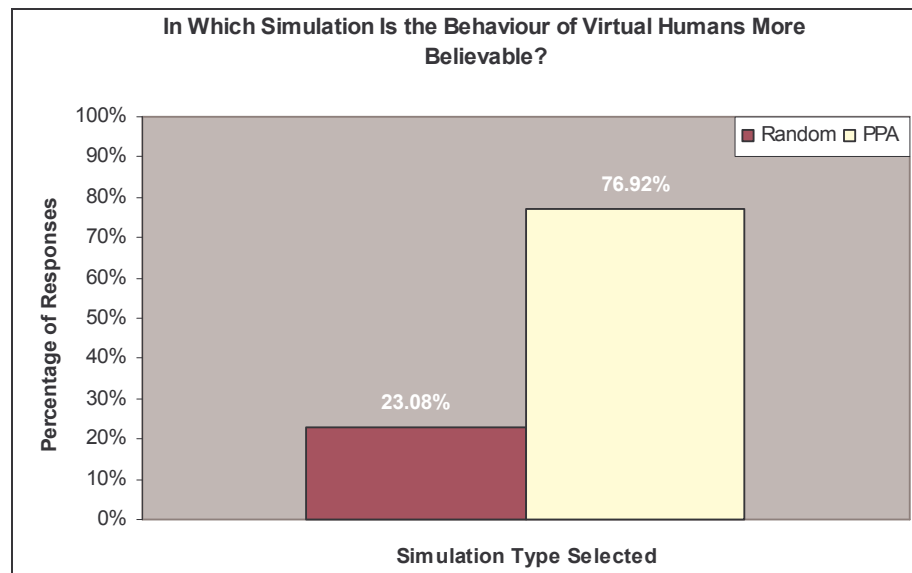


Figure 6-3: A graph of the percentage of subjects who selected the simulation populated by proactive persistent agents as the most believable, against the percentage who selected the random agent populated simulation

This is the key result of the believability experiment as it indicates that subjects found the behaviour of the PPA based characters significantly more believable than that of the characters implemented to pursue random behaviours. Although this is not the same as saying that subjects found the behaviours of these characters entirely believable, it is a more objective result, and still indicates that this is, in fact, the case.

In question 2, subjects were asked to “*describe the behaviour of the bar patron characters*” in the simulation they chose as most believable, focussing on “*what is driving the characters’ behaviour and what actions they choose to perform*”. Some of the answers given to this question are illustrative of why subjects found the behaviour of the PPA based characters more believable.

One of the key features of the PPA based simulation which made people choose it as more believable was the fact that agents’ behaviours displayed a high degree of consistency. In particular, subjects found the cause and effect nature of the way in which agents would go to the bar a couple of times and then use the toilet before returning to the bar highly

believable. This behaviour comes from the use of motivations and the interplay between them allowed by the use of FCMs.

Similarly, subjects found the emergence of groups within the PPA based simulation added greatly to believability. This emergent property came from the fact that agents chose whether or not to sit at a particular table, based upon their relationships with the other characters already seated there. These relationships were managed by the μ -SIC system, and grew from the interactions in which agents were involved. The result of this was that characters would continually return to the same table, and only occasionally move to interact with a new group of agents in another part of the bar. In the random simulation agents chose randomly which table to sit at, which gave no impression of group formation.

Some unexpected results emerged from the answers to question 2 given by subjects that chose, as more believable, the simulation populated by random agents. The first of these arose from an Italian subject (all of the other people involved in the experiment were Irish) who did not find agents that returned to sit at the same table over and over believable, but was much more comfortable with the random simulation in which characters were more inclined to move between the different tables in the bar. One possible explanation for this is that believability has a cultural component, and in fact this is an existing idea in research into the gestures displayed by virtual humans [Ruttkay et al, 2003].

Related to the previous point, some of the answers to question 2 suggested that a bar environment was not the ideal choice for evaluation, as some subjects had extremely diverse expectations as to what kind of behaviour to expect. For example, one subject found the fact that agents did not appear to get drunk highly unbelievable, while another expected some extremely sophisticated mingling between the male and female characters in the bar, the type of which might be expected in a singles bar. Perhaps a slightly more sedate environment such as a café, or an office would be more suitable for future evaluations.

In order to further delve into the reasons behind subjects' choice of the more believable simulation, part 3 of the questionnaire asked them to agree or disagree with a number of statements regarding both simulations. These statements were:

- *Agents appear to act under their own volition*
- *Agents' behaviours appear to be goal driven*
- *Agent's behaviours possess a random component*

Subjects' responses to these statements were placed on a five point scale, the values of which ranged from *strongly disagree* - through *disagree*, *no opinion* and *agree* - to *strongly agree*. Based upon these responses it was determined whether or not subjects found the statements to be more telling about the PPA based simulation than the simulation populated by random agents. These comparisons were achieved by performing paired T-tests and *analysis of variance* (ANOVA) on the data which emerged from this part of the questionnaire. For the purposes of discussion, and in the statistical tests performed, the results have been reduced to just a three point scale showing *agree*, *no opinion* and *disagree*.

The first statement, which referred to agents behaving under their own volition, was intended to show in which simulation it was most strongly apparent that agents displayed a strong sense of agency, appeared to make their own decisions and behaved proactively. Subjects' responses to this statement are shown in figure 6-4, in which it can be seen that more people agreed with this statement for the PPA based simulation, than the random agent based simulation. This hypothesis is statistically supported with a significance of over 99%. The hypothesis is also statistically supported to the same significance by an ANOVA, the details of which are shown in table 6-2.

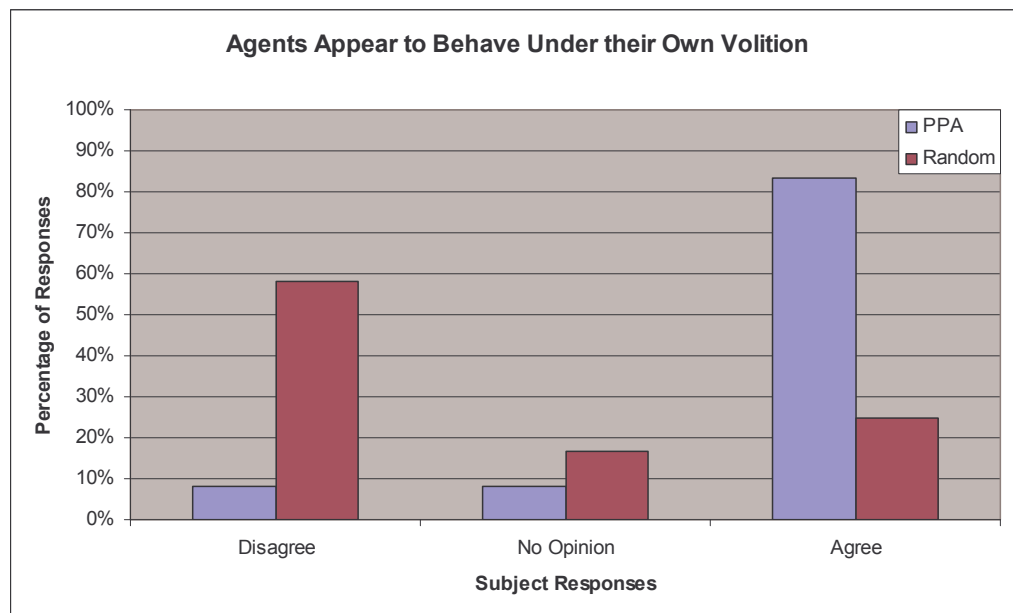


Figure 6-4: A graph of subjects' responses to the statement "agents appear to behave under their own volition" for both the simulation using PPAs and that using random agents

Table 6-2: The results of an ANOVA performed to test the hypothesis that more subjects agreed with the statement that *agents appeared to act under their own volition* for the PPA based simulation, than for the simulation using random agents

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	7.041667	1	7.041667	11.99355	0.002211	4.300944
Within Groups	12.91667	22	0.587121			
Total	19.95833	23				

The second and third statements were expected to act essentially as controls in this experiment. Statement 2 asked subjects whether or not agents’ behaviours appeared to be goal driven. Although it should be slightly more so where PPAs are used, this should be the case for both simulations, as agents pursue the goal of using particular objects. As is shown in figure 6-5, this pattern is seen in subjects’ responses, with only marginally more of them agreeing that agents in the PPA based simulation behaved in a more goal directed manner, than those in the random agent based simulation. A paired T-test does not support the notion that more subjects agree with this statement in the case of the PPA based simulation than in the case of the random agent based simulation, to a statistically significant level. Similarly, an ANOVA on the same data (the details of which are shown in table 6-3) does not support the hypothesis to a significant level

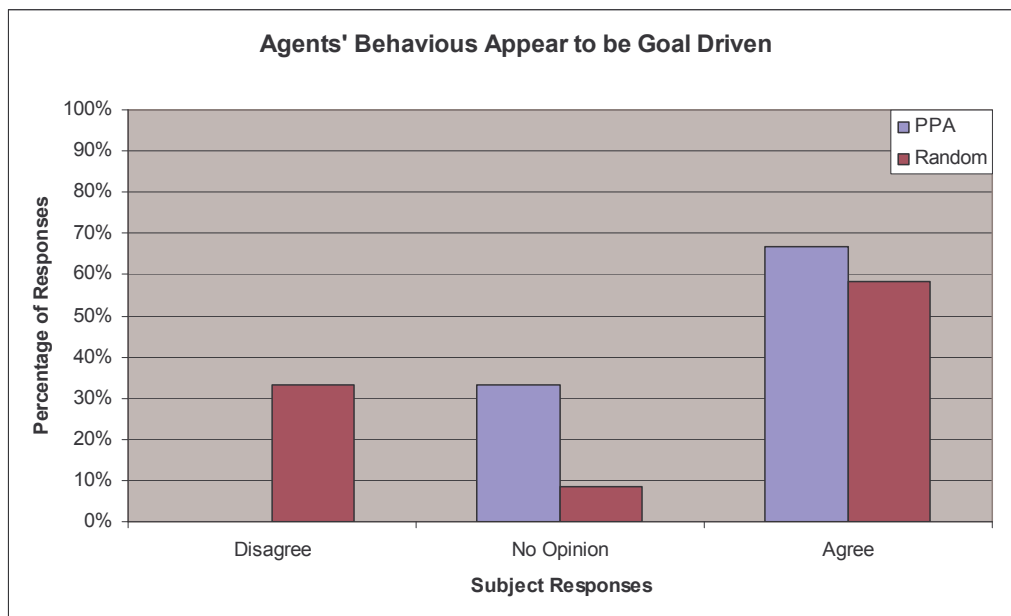


Figure 6-5: A graph of subjects’ responses to the statement “agents’ behaviours appear to be goal driven” for both the simulation using PPAs and that using random agents

Table 6-3: The results of an ANOVA performed to test the hypothesis that more subjects agreed with the statement that *agents’ behaviours appeared be goal driven* for the PPA based simulation, than for the simulation using random agents

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1.041667	1	1.041667	1.99355	0.196499	4.300944
Within Groups	12.91667	22	0.587121			
Total	13.95833	23				

The final statement in this part of the evaluation questionnaire asked users to state how much they felt that agents appeared to behave in a random fashion. It was obviously expected that subjects should notice that the characters in the random agent based simulation behaved more randomly. This was supported by subjects’ responses, shown in figure 6-6, and was also supported statistically in a paired T-test which showed that subjects agreed with the statement more in the case of the random based simulation than for the PPA based simulation, to a significance of approximately 98%. Again, this hypothesis is supported to the same level of significance by an ANOVA performed on the same data. The details of this analysis are shown in table 6-4.

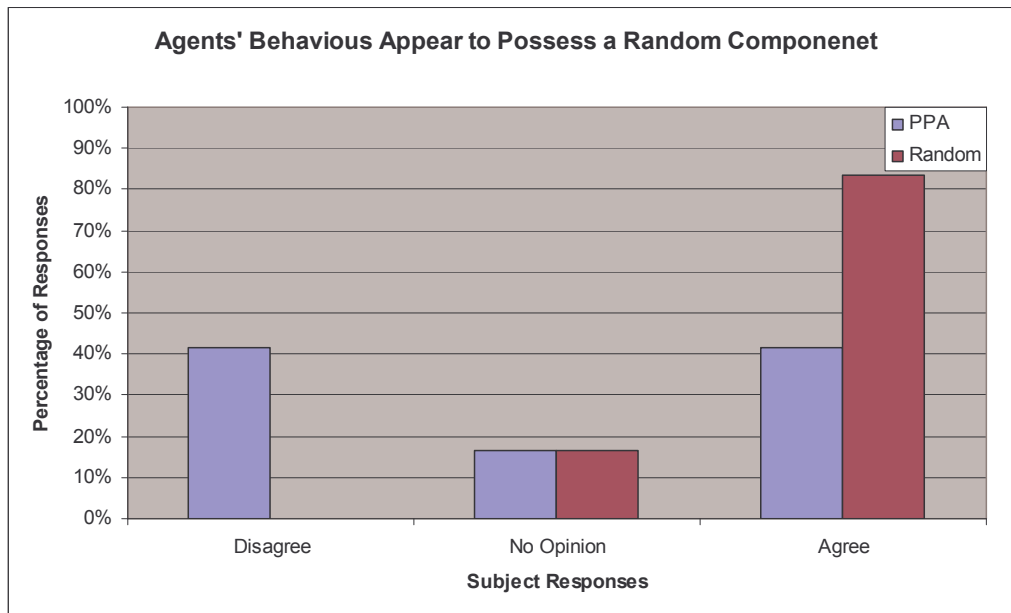


Figure 6-6: A graph of subjects’ responses to the statement “agents’ behaviours appear to possess a random component” for both the simulation using PPAs and that using random agents

Table 6-4: The results of an ANOVA performed to test the hypothesis that more subjects agreed with the statement that *agents' behaviours appeared to possess a random component* for the random agent based simulation, than for the simulation using PPA based agents

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	4.166667	1	4.166667	7.857143	0.010363	4.300944
Within Groups	11.66667	22	0.530303			
Total	15.833333	23				

6.4 Social Abilities

NPCs in character-centric games should be capable of consistent, believable and interesting social interactions. Not only should players be able to engage NPCs in such interactions, but it should also be possible to observe NPCs as they engage in interactions amongst themselves, leading to changing relationships and the propagation of information through a game world. An evaluation of these capabilities within the PPA architecture takes the form of an evaluation of the μ -SIC system.

The purpose of the μ -SIC system is to choose which interactions characters engage in when given the opportunity to do so. Section 4.5.3 described how the system was trained to do this and presented a short evaluation of its performance based on this training. Although this showed that the system learned what was intended, it said nothing about the believability or consistency of the interactions which characters perform. In order to evaluate these issues, a number of simulation snap-shots which illustrate how a particular interaction arises (based on an NPC's personality, current mood and relationship to the character under consideration for interaction) will be considered. These simulation snap-shots come from Berlin-End-Game, the game implemented using the PPA architecture.



The simulation snap-shots are shown in figure 6-7. Each screenshot illustrates how a different interaction can arise between two characters. The characters involved in the interaction are highlighted, and their details are listed below each image. The details shown are the personality of the character instigating the interaction, his current mood and his relationship with the character with whom he is interacting. In each image the character instigating the interaction is further highlighted.

To illustrate how a particular interaction arises, situation A, which shows how the character Moe chooses to perform a general CHAT interaction with the character Sean, can be considered. The information below the image shows that, although Moe is an aggressive character, his mood is good and he likes Sean, whom he does not know particularly well. Added to these facts, Moe's mild submissive attitude towards Sean and the fact that he is

quite interested in him result in Moe choosing the general CHAT interaction. A CHAT interaction causes the instigating character to randomly select one of the subjects in which he is interested (or details of a fact or event which he knows about) and announce this to the other character. This is intended to simulate very general conversations, or more colloquially *shooting the breeze*. In the image, the musical note icon indicates that Moe chooses to chat about music.

A further example, situation F, shows the character Allison performing a FLIRT interaction with the player character. The reasons for this interaction arising are, firstly, that Allison has a sociable personality and is in a relatively good mood. On top of this she considers herself neither dominant nor submissive to the player, likes him and is highly attracted to him. All of these factors add together to cause her to flirt with the player.

The rest of the interactions shown in figure 6-7 can be explained in a manner similar to the two examples considered here. Taken together, all of these examples show that the interactions chosen by NPCs are consistent in that they are based upon their personalities, moods and relationships. Although figure 6-7 shows only a small number of examples, this consistency is apparent in observed interactions throughout test simulations. It is also important that in the interactions shown, there is a mixture between interactions which NPCs perform with other NPCs, and interactions which are performed with players.

<p>Situation A Moe performs a CHAT interaction with Sean</p>	<p>Situation B Moe performs an ASSAULT interaction with George</p>
	
<p>Personality: Aggressive Mood: Fine</p> <p>Relationship</p> <p> Like/Dislike: Okay Intimacy: Low</p> <p> Interest: Okay Attractedness: None</p> <p> Dom/Sub: Low Submissive</p> <p>Interaction: Chat</p>	<p>Personality: Aggressive Mood: Angry</p> <p>Relationship</p> <p> Like/Dislike: Dislike Intimacy: Low</p> <p> Interest: Okay Attractedness: None</p> <p> Dom/Sub: Low Submissive</p> <p>Interaction: Assault</p>

<p style="text-align: center;">Situation C The Boss performs a COMPLIMENT interaction with Mary</p>  <p>Personality: Leader Mood: Good Relationship Like/Dislike: Okay Intimacy: Low Interest: Okay Attractedness: High Dom/Sub: Submissive Interaction: Compliment</p>	<p style="text-align: center;">Situation D Niamh performs an INSULT interaction with the Player</p>  <p>Personality: Thoughtful Mood: Bad Relationship Like/Dislike: Dislikes Intimacy: Medium Interest: Quite High Attractedness: Low Dom/Sub: Low Submissive Interaction: Insult</p>
<p style="text-align: center;">Situation E Bob performs a JOKE interaction with the Chief</p>  <p>Personality: Active Mood: Very Happy Relationship Like/Dislike: Likes Intimacy: Low Interest: Okay Attractedness: None Dom/Sub: Equal Interaction: Joke</p>	<p style="text-align: center;">Situation F Allison performs an FLIRT interaction with the Player</p>  <p>Personality: Social Mood: Okay Relationship Like/Dislike: Likes Intimacy: Low Interest: Okay Attractedness: High Dom/Sub: Equal Interaction: Flirt</p>

Figure 6-7: A range of screen-shots showing how some of the interactions possible within the μ -SIC system can arise

Although it cannot be seen in the snap-shots shown, test simulations also demonstrate how relationships between characters evolve over time. The result of this is that, as NPCs

become more familiar with each other, the kinds of interactions in which they engage change. NPCs that form positive relationships are more likely to interact with each other, and when they do, perform positive interactions. Those that form more negative relationships will do the opposite, performing interactions such as INSULT and ASSAULT, or simply ignoring one another.

An interesting emergent property of the system is that, as a simulation progresses and relationships between NPCs become established, characters begin to form groups. NPCs are more likely to interact with characters with whom they have formed favourable relationships, and less likely to interact with those with whom they have not done so. The result of this is that in situations in which group interactions are possible (such as around a table in a bar, or around the water cooler in an office) groups of agents with strong relationships emerge. This emergent property was observed by a number of the subjects in the PPA believability experiment.

As interactions are performed, information also moves through the game world. When characters perform the general CHAT interaction, periodically, rather than choosing a random subject to announce, they tell the other character about an event which they have witnessed, or a fact that they know. The choice to do this is based on a mixture of relationship-based and random factors. This allows the flow of information between groups of characters that have formed relationships.

Taken together the facts that interactions are consistent, lead to observable effects such as the formation of groups, and allow the flow of information through a game world lead to the conclusion that interactions are also believable.

The most subjective aspect of the evaluation of PPAs' social abilities, is whether or not the interactions which NPCs perform are interesting. The best answer to this question is that PPA based NPCs are capable of a much more broad range of interactions than NPCs in most current commercial games. Typically, these NPCs are only capable of a predefined number of canned responses to a limited set of questions which players can ask. The capabilities of the characters in the Sims^{GR-42} most closely resemble those of PPA based characters. NPCs in the Sims were capable of sophisticated interactions based on their personalities and moods. The fact that the Sims is now the best selling PC game of all time, would suggest that players find such a range of interactions interesting, and so suggests that they would find the kinds of social interactions which PPAs are capable of performing equally so.

6.5 Application Domain

The purpose of the application domain evaluation category is to determine the extent to which the use of the PPA architecture makes players' game experiences more satisfying and immersive. It has already been established that the behaviours of characters within a given situation are suitably believable, and that NPCs are capable of the sophisticated social interactions required in character-centric games. However, the effect that *persistence* has on players' game experiences has not yet been evaluated. Persistence is manifested in a game world by the fact that characters can be observed pursuing their daily activities over the course of a game, irrespective of the actions of players. For example, within the same game an NPC might first be encountered leaving home for work, later on in their office and then again at the cinema that night. This behaviour is enabled by the use of the schedule unit, and the fact that characters are modelled at all times within a game engine. The question that this part of the evaluation asks is whether or not players notice that NPCs are persistent, and if so, does this add to the sense of immersion felt while playing a game. A more immersive game is considered a more satisfying game.

A user trial using the Berlin-End-Game application was conducted to perform this part of the evaluation. The following section will describe this trial, and then its results will be presented.

6.5.1 The Berlin-End-Game Evaluation Experiment

The *Berlin-End-Game evaluation experiment* required that subjects play two versions of Berlin-End-Game. In the first version, NPCs were implemented using the full PPA architecture and so displayed all of the behaviours associated with that. In the second version of the game, NPCs did not adhere to a schedule, and so were only ever encountered within a single location, in which they had adopted a single role. For example, a character that was first encountered in the restaurant location, would not be seen anywhere else. These agents were not considered to display persistence (although they were modelled within the game world at all times, they did not display the key attributes of persistent agents) and so were more akin to the characters in current games.

After brief instruction in how to play the game, and having their part in the experiment described to them (which did not include the goals of the experiment or any information regarding the differences between the two versions of the game), subjects were given approximately twenty-five minutes to play each version of the game. After doing this,

subjects were required to fill out a questionnaire (given in appendix C) detailing their experiences. The order in which subjects played the two versions of the game was random, and this was found to have no effect on subjects' responses. Nine subjects took part in this experiment, seven of whom regularly played computer games, and two of whom did not. The next section will detail their responses.

6.5.2 Experiment Results

After playing both versions of the game, subjects were first asked to indicate if there was “*a noticeable difference between version one and version two*”. Out of the nine people that performed the experiment, just one indicated that they had noticed no differences. That subjects would not be able to distinguish between the two versions of the game was of great concern in conducting this experiment, since the brief given (to notice any differences between the two versions of the game) was extremely broad. However, that this happened in only one case showed that this concern was unjustified.

As an addendum to question 1, those subjects that had noticed differences between the two versions of the game were asked to “*describe the differences that [they] noticed*”. Out of the eight subjects that answered this question, six indicated that the most obvious difference was that NPCs could be observed moving between different situations which matched the time of day in the game, the key manifestation of persistence. The two subjects that did not notice this were the two that did not regularly play computer games. This was not a surprise, as for someone who was not familiar with games, so much of what was involved in this experiment was novel. Thus, isolating a subtle feature such as the use of persistence was expected to prove difficult.

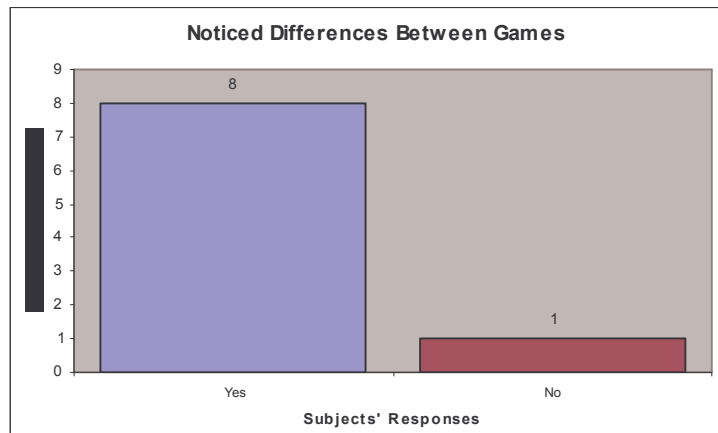
The final part of the questionnaire asked subjects to select in “*which version of the game did [they] find the game-playing experience most immersive*”, and “*to explain [their] choice*”. Of the six subjects that had noticed the inclusion of persistence, five selected the version of the game which used it as most immersive, and indicated its inclusion as the reason why. Only one subject selected the other version of the game as being more immersive. The reason given for this choice was that in this version particular characters were much easier to find. Although this was certainly true (characters always remained in a single location), and may have made the game easier, perhaps it does not relate directly to the question of immersion. However, it is still an interesting result and raises the issue of balancing the difficulty of a game with its realism. However, that question is outside the scope of this evaluation.

The responses of all of the subjects that took part in the Berlin-End-Game evaluation experiment are given in table 6-5, and graphed in figure 6-8. Figure 6-8 (A) shows the number of subjects that noticed a difference between the two versions of the game, along side the number that did not. Of those subjects that noticed a difference between the two versions of Berlin-End-Game, figure 6-8 (B) shows which of them noticed the inclusion of persistence as the major difference. Finally, figure 6-8 (C) shows which of the subjects that noticed the inclusion of persistence felt that it added to the sense of immersion arising from the game.

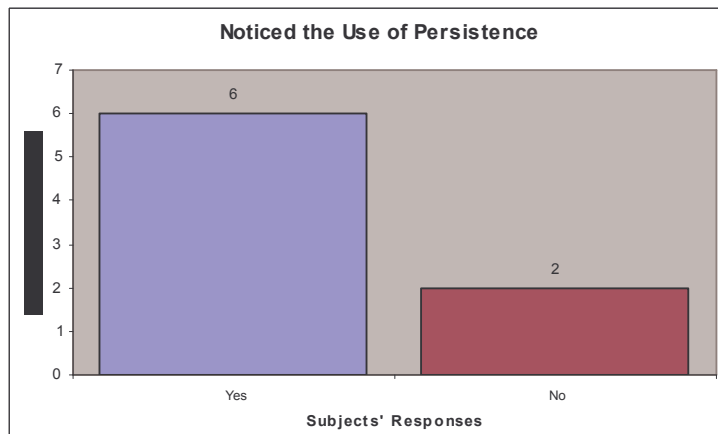
From the results shown it is clear that of those subjects that regularly played games, the majority of them found the version of Berlin-End-Game in which characters adhered to a schedule more immersive than the version in which they did not. This shows that the manifestations of persistence add significantly to the sense of immersion created while playing a game. If this result is considered along with the results of the PPA believability experiment and the evaluation of the social abilities of PPA based characters given in section 6.4, then it is clear that the use of the PPA architecture adds greatly to the sense of immersion felt while playing a game and so, also makes the game experience more satisfying. This satisfies the criteria of the application domain evaluation category.

Table 6-5: The results of the Berlin-End-Game evaluation experiment

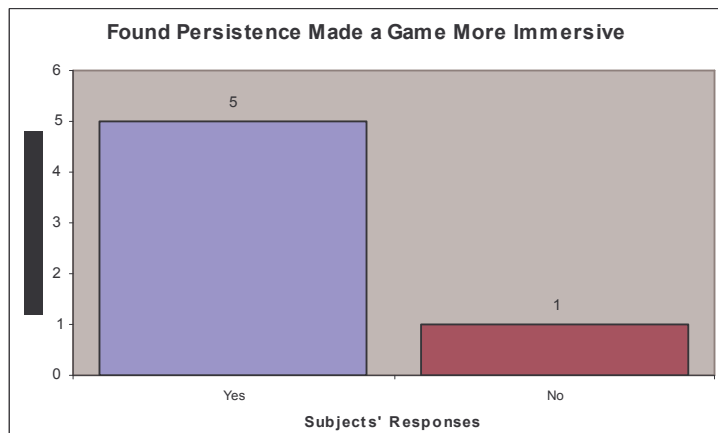
	Noticed a difference between the two games	Noticed the use of persistence	Found persistence made a game more immersive
Subject A	✓	✓	✓
Subject B	✓	✓	✓
Subject C	✓	✓	✓
Subject D	✓	✓	x
Subject E	x	-	-
Subject F	✓	x	-
Subject G	✓	x	-
Subject H	✓	✓	✓
Subject I	✓	✓	✓



(A)



(B)



(C)

Figure 6-8: Graphs illustrating the responses of subjects in the Berlin-End-Game evaluation experiment

6.6 Computational Issues

Although they are important in all applications, computational issues are particularly so in technologies for use in game development. Games must run on typical home PCs or games consoles, and the processing power available on such machines is fairly restrictive.

Furthermore, a large amount of these available resources are already used for other tasks, such as rendering. For these reasons, game-AI techniques must be extremely efficient both in terms of memory usage and their processing requirements. This portion of the evaluation will show that this is so in the case of the PPA architecture.

In a typical modern RPG or adventure game, which must run at a frame rate of approximately 40 frames per second, game-AI processing receives 10% - 20% of the power available on a console or PC [Woodcock, 2000b]. Agents implemented using the PPA architecture must satisfy this constraint. In order to show that this is the case, a version of Berlin-End-Game was created which does not graphically display the game world. This system was run, as a simulation without players, for 50,000 game update cycles which was timed to take approximately 25 seconds. This implies a time of 0.5 milliseconds to update all of the NPCs within the game world. The game world in question was populated by 40 characters, which leads to an average time of 12.5 microseconds per individual character AI update. These timings were made on a PC running a Pentium 4 2GHz processor with 512 MB of RAM, which, for the purposes of this evaluation, are considered the specifications of an average home PC at the time of writing.

If the constraints of a frame rate of 40 frames per second and a total of 10% - 20% of the available processing power are considered, a theoretical maximum time of between 0.025 and 0.05 seconds is available in a typical game for each world AI update. If this limit is considered along with the timings just described, it suggests that a game world could be populated by between 200 and 400 NPCs implemented using the PPA architecture. This would allow games to be set in large environments, densely populated by sophisticated support characters. These timings are detailed in table 6-6, and show that the processing requirements of the PPA architecture make it suitable for use in games.

Table 6-6: An evaluation of the processing requirements of agents created using the PPA architecture. Also shown are the theoretical maximum number of NPCs possible, if different percentages of the available processing power are used

Game Results			
Number of World Updates	50,000		
Total Time Elapsed (seconds)	25		
Time Per World AI Update (seconds)	0.0005		
Number of NPCs	40		
Time Per NPC Update (seconds)	0.0000125		
Documented Ideal			
Frames Per Second	40		
Time Per Frame (seconds)	0.025		
Percentage of CPU for AI	10%	15%	20%
Time Per World AI Update (seconds)	0.0025	0.00375	0.005
Number of Possible NPCs			
Theoretical Maximum Number of NPCs	200	300	400

In order to gauge the memory required to simulate a PPA based character, a test application was developed which simply loaded into memory a population of characters, without actually loading a virtual world, or performing any world updates. This application was run repeatedly using different numbers of agents each time. The total memory used by this application was measured, and from this an approximation of the total memory required by a single character was reached. The amount of memory used by each run of this application, along with the number of characters loaded, is shown in table 6-7. To show how this total is affected by increasing numbers of characters, this is also graphed in figure 6-10.

Table 6-7: The amount of memory required to load a cast of a given number of characters

Number of Characters	Memory Required (Kb)
0	2,320
20	11,772
40	20,404
60	29,696
80	38,228
100	47,520
120	56,060

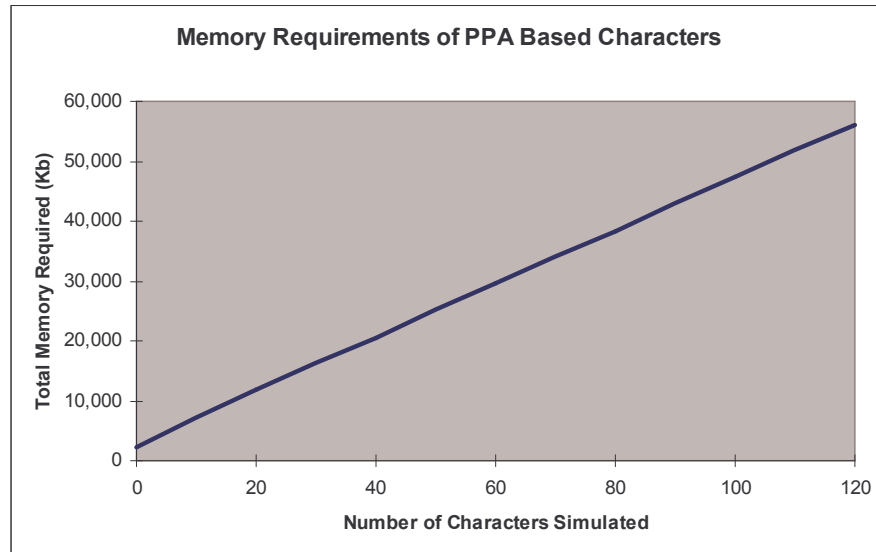


Figure 6-9: A graph of the memory required to simulate an increasing number of PPA based characters

Based on the results of this experiment it can be seen that the memory required to simulate a population of PPA based characters increases linearly with the number of NPCs, and that the memory required for a single character is approximately 0.5 megabytes. No documented guidelines for the memory usage split between the different aspects in a game could be found to use in this experiment. However, if 512MB is assumed as the memory present on an average home PC, then it is reasonable to suggest that $\frac{1}{4}$ of this would be given over to the AI requirements of the characters within a game. This suggests 128Mb for character AI, which in turn leads to a theoretical maximum of 250 game characters. It is worth noting that this evaluation only considers physical memory, and if virtual memory and page swapping techniques are taken into account, then it should be possible to simulate a much larger cast of characters.

Although the results presented in this section are fairly satisfying by themselves, it is extremely important to note that the code used in these experiments has not been highly optimised, especially not to the high levels required in game development. There are many places already identified where memory could be conserved, in particular the extensive use of character strings rather than other smaller data types which would be just as effective. Similarly, by profiling the game code it is expected that many optimizations could be made in terms of the processing time required for each agent update.

6.7 Production

The final evaluation category with which this work is concerned is production, which refers to the ease with which game designers can use the PPA architecture to populate their games with NPCs. The ease with which designers can author schedules and roles for use in their games is the main concern of this segment of the evaluation. Authoring schedules is by itself a straightforward task, and the use of XML makes it even more so. Although it has not yet been implemented, this task could be made even easier through the creation of a GUI based schedule authoring tool, which would distance designers from raw XML files.

Authoring roles involves drawing PPAFCMs to suit a particular situation. Kosko states that based upon reading any editorial-style article an average person ought to be capable of drawing an FCM to explain that article [Kosko, 1993]. There is no reason to suggest that this should not also be the case for PPAFCMs for use in games. Once a game designer is sure of the details of a particular role, they should have little difficulty in architecting a PPAFCM to represent it. Although it has not yet been tested, it is expected that many designers would find using PPAFCMs easier than writing scripts, or architecting the kind of rule-based systems which are in wide use today.

Similarly to schedules, architecting PPAFCMs could be made easier by the creation of a GUI based tool, and the process of translating a hand-drawn PPAFCM into XML (which although trivial to perform by hand is time consuming) could easily be automated.

6.8 Conclusions

Although evaluation of virtual human simulation systems is notoriously difficult, this section has presented an evaluation scheme which can be used for the purpose. By dividing the evaluation task into different categories and considering these individually, an evaluation of the entire system emerges. The evaluation categories used are: believability, social abilities, application domain, computational issues and production. Through a mixture of user trials and descriptive examples it has been shown how each of these is satisfied by the PPA architecture. Characters can be created which are capable of behaving believably in a range of diverse situations and displaying sophisticated social behaviours. These abilities have been shown to add to the sense of immersion created while playing a game. The resource requirements of the system are also suitable for use in games, and the production task for game designers is made easier through the use of schedules and FCMs.

7 Thesis Summary, Conclusions & Suggestions for Future Work

This final chapter will summarise the major contributions of this thesis, draw conclusions from the work presented and suggest directions in which it might be taken in the future.

7.1 Thesis Summary & Conclusions

Game development has become a major industry - now considered financially on a par with the film industry. Commercial games are no longer developed by interested individuals, but rather by large teams of talented artists and developers, with budgets of millions of euro. Until relatively recently, these teams have focused their development effort upon improving the graphical realism of games. However, the graphics in games have now reached such a high standard that, game developers are concentrating their efforts on other technologies important in creating the kinds of immersive experiences that game-players demand. AI in games (*game-AI*) is one of these, and is expected by many to be the next major growth area in game development.

Game-AI presents a number of unique challenges to developers, but the most important of these is the simulation of believable and interesting computer controlled non-player characters (NPCs). In particular, *support characters* - a unique kind of NPC that take parts such as shop-keepers, barmen and patrol guards - are crucial to the creation of believable virtual worlds, and have been identified as offering unique challenges to the application of AI techniques. This area has also been largely overlooked in what little game-AI research has gone on. Support characters are most important in *character-centric* games. These are games in which, rather than concentrating upon action, game-play moves at a slower pace and focuses on players' interactions with NPCs. The established adventure and role-playing game genres fit most comfortably into this category.

Intelligent agent technologies offer a compelling solution for controlling the behaviours of NPCs. The key requirements of an intelligent agent based system for the control of support characters in character-centric games are as follows:

- Agents should display believable behaviour
- Agents should display believable behaviour across a range of diverse situations, all within a single game
- Agents should be capable of sophisticated social behaviours
- The agent system should perform in real-time
- The system should not have unrealistic processor or memory requirements
- It should be relatively easy for non-programmers to author agent behaviours using the system

Thesis Summary, Conclusions & Suggestions for Future Work

It is argued that no existing agent architecture satisfies this unique set of requirements, and so, the *proactive persistent agent* (PPA) architecture, has been developed. Characters implemented using this architecture are proactive in the sense that they pursue their own goals over the course of a game, irrespective of the actions of players; and are persistent in the sense that they are at all times modelled - at least to some extent - within a game-engine. Neither of these properties is common in the typical NPC implementations currently used by commercial game developers.

The notion of *situational intelligence* is at the core of the PPA architecture. Rather than creating very general agents capable of dealing with any occurrence which might arise over the course of a game, the idea of situational intelligence suggests that it is better to create a very basic agent and to this add just enough sophistication to allow a character behave believably in its current situation. The constant adjustment of an agent's internal composition to match its current situation enables it to behave believably across any range of situations. The use of situational intelligence in game-AI is unique to this work, and one of its major contributions.

The PPA architecture is composed of three key components:

- The **schedule unit** which manages the way in which agents move between different situations
- The **role-passing unit** which controls an agent's behaviour in a given situation and allows it adjust its internal composition to match new situations
- The **μ-SIC system** which uses psychological models to endow agents with sophisticated social capabilities

The schedule is the simplest component of the architecture, but still lends greatly to the believability of the characters that use it. A character's schedule defines where the character should be, and broadly, what it should be doing over a set of time periods which divide up a simulation day. In this way, characters can give the impression of pursuing lives beyond their involvements with players, as they are observed going to work, going out to socialise, relaxing at home and so forth.

As characters move between different situations, as dictated by the schedule unit, the way in which they behave should change dramatically. The manner in which a character behaves in a particular situation is referred to as a *role*. The purpose of the role-passing unit is to control a character's behaviour when a particular role is adopted, and to manage

Thesis Summary, Conclusions & Suggestions for Future Work

the addition and removal of roles, as a character's situation changes. When a particular role is adopted, a character's behaviour is controlled using an approach based on *fuzzy cognitive maps* (FCMs). FCMs are based on *fuzzy logic* and are extremely useful in situations in which the complex interplay between concepts must be modelled, as is the case in controlling the behaviour of NPCs. An FCM designed to control an agent in a particular situation, allows extremely believable behaviour, as was shown by the believability experiment described in section 6.3. This satisfies the first of the requirements outlined above.

When a character's situation changes the role-passing unit swaps out the FCM controlling the character's behaviour in the current situation, and swaps in a new one. This is where the ideas of situational intelligence are most apparent. The ability to change roles, therefore adjusting an agent's behaviour to match its changing situation, satisfies the second requirement outlined above - that characters can behave believably across a range of diverse situations.

The third requirement of intelligent agents used in character-centric games is that they are capable of sophisticated social behaviours. The μ -SIC system has been created for this purpose. When a character's role-passing unit dictates that the character is free to socially engage with the other characters around it, the μ -SIC system determines which interactions the character should perform, and with whom. Interactions are driven by psychological models of an NPC's personality, mood and relationships with other characters. As was discussed in the evaluation chapter, the μ -SIC system allows NPCs perform believable, consistent and interesting social interactions both with each other, and with players. This ability is crucial in character-centric games as much of their game-play involves players interacting with NPCs.

These units, at the core of the architecture, satisfy the most important requirements outlined for an intelligent agent system for the control of NPCs. Characters implemented using the architecture can behave believably in any number of diverse situations and are capable of sophisticated social behaviours.

The remaining requirements focus on more practical issues, and the PPA architecture has been implemented in order to satisfy these. The use of role-passing and *level-of-detail AI* help in satisfying the requirements that games must run in real-time, and have resource requirements not beyond the capabilities of a typical home PC, or games console. That the PPA architecture satisfies these requirements was shown in the evaluation chapter, and is

Thesis Summary, Conclusions & Suggestions for Future Work

particularly evident in the two demonstration applications which use it, both of which run in real-time on machines of modest specifications. The first of these, Berlin-End-Game, is a character-centric adventure game loosely based on the Bernard Samson novels by Len Deighton, while the second is the integration of the PPA architecture into the ALOHA system, a 3-d engine for the animation of virtual humans.

The implementation of the architecture promotes the idea of *games-as-data*, a game design method that explicitly divides the game development process into those tasks that are performed by developers, and those that are performed by designers. When games are created in this way, developers create the core game-engine technologies, while game designers define all of the actual game content in data files, which are then loaded into the game-engine at run time. These go some way towards satisfying the final requirement of the architecture - that it can be easily used by game designers, who are typically non-programmers, in authoring virtual game worlds. This is further satisfied by the fact that the authoring task essentially reduces to writing schedules and illustrating the FCMs required to drive a character's behaviour in particular situations. Writing schedules is a trivial task and, as was argued in section 6.7, illustrating an FCM is something that most people should be able to do without too much difficulty. This makes the PPA architecture ideally suited to the purposes of game design.

That the PPA architecture satisfies the requirements of an intelligent agent system to drive the behaviours of NPCs was rigorously evaluated, the process of which was described in chapter 6. Evaluation of systems for the simulation of virtual characters is notoriously difficult (mainly due to the fact that the subjects to be evaluated are extremely subjective) and so this evaluation itself is considered one of the contributions of this work. The evaluation scheme used is based on a scheme developed by Isbister & Doyle for evaluating embodied conversational pedagogical agents. The key to this scheme is that it separates the evaluation problem into a number of different categories which can each be evaluated in isolation. By performing these separate evaluations, the success of the overall system can be determined. In this case the evaluation showed that the PPA architecture is ideally suited to the control of support characters in character-centric games.

7.2 Suggestions for Future Work

This section will discuss a number of significant additions which might be made to the PPA architecture in order to improve it.

7.2.1 Improvements to the Schedule Unit

Although the schedule unit used within the PPA architecture is relatively successful in its present state, there are a number of improvements which could make it far more effective. The first major change would be to stop using the hard start and end times that are used at present, but rather to introduce a notion of *urgency*. Each schedule entry could be given an optimum start and end time which, as they came closer, would cause the urgency to change over to the next schedule entry to increase. By integrating this notion of urgency into characters' PPAFCMs, the change-over between schedule entries could be made more realistic as more role-related or persona-based factors could affect it.

At present each schedule entry indicates the location which a character should go to before adopting a given role. While this is suitable for many entries, in particular those referring to a work place which should never change, it is not entirely so for many others. For example, when characters are pursuing leisure activities it might be more useful for the schedule to indicate this as a particular class of entry, which could then be pursued in any location which would facilitate it. This kind of schedule entry could also enable characters to arrange group activities. At present group activities only take place when characters independently go to the same location to perform the same kind of activity. If more generalised schedules were used, characters could arrange amongst themselves to perform these together.

7.2.2 Further Use of FCMs

At present the use of FCMs in the role-passing unit only occurs to a certain level of granularity. The behaviours implemented by the action nodes in a PPAFCM perform actions such as navigating to a certain location, interacting with a character, or using a particular object. These actions are hard-coded into the game-engine as a series of steps that a character must perform. However, some of these tasks could themselves be better implemented using FCMs, thus creating a hierarchical system of maps, as discussed in [Kosko, 1997]. Some work has taken place in this direction [Parentoën et al, 2001], where low-level navigation was performed using FCMs. An FCM borrowed from this work (which controls the steering behaviour of a sheep dog attempting to direct a sheep into a particular zone) is shown in figure 7-1, and it would be interesting to see how this low-level use of FCMs could be integrated into the PPA architecture.

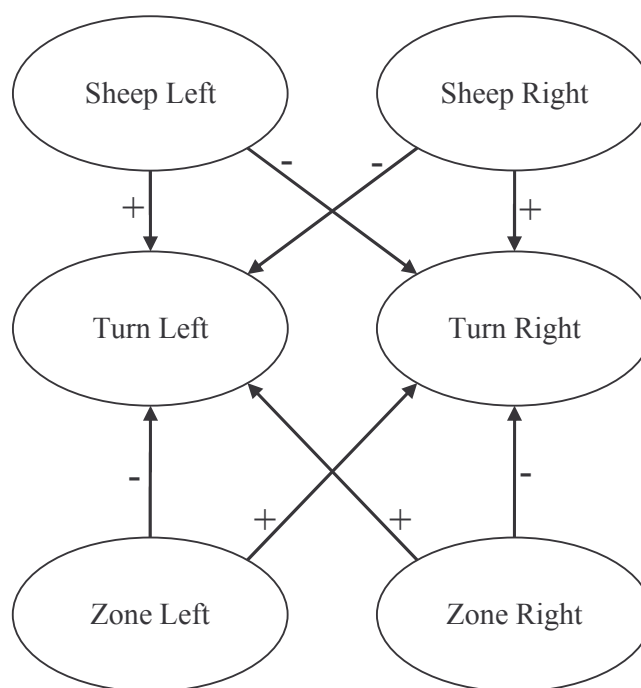


Figure 7-1: Borrowed from [Parentoën et al, 2001], an illustration of a low-level FCM for use to control a sheep dog as it tries to bring back a sheep

7.2.3 Further Use of Psychological Models

At present, the psychological models described in chapter 4 are only used within the μ -SIC system. However, these aspects of a character’s persona could be useful in other areas of the architecture, in particular within the role-passing unit. Within a PPAFCM, nodes could be included to represent the valance and arousal of a character’s mood and the details of their personalities. These nodes could be used to allow characters’ personalities and moods influence their behaviour at all times. However, determining exactly how a character’s mood and personality might influence the other concepts associated with a particular role could prove troublesome.

7.2.4 Improvement of the μ -SIC system

Simulation of social behaviours is an extremely complex subject. Although the μ -SIC system achieves it to an extent, it could be improved. One thing that the present version of μ -SIC does not do, is explicitly take into account the previous interactions in which a character has been involved, when deciding on its next interaction. Although there is some implicit modelling of this in the effects that interactions have upon an agent’s internal state, by modelling it explicitly characters could be enabled to display more believable

behaviour. However, this improvement would be made difficult by the amount of new data which would have to be made available in order to train the system.

Another improvement to the μ -SIC system would be the integration of natural language [Drennan, 2003] and gesture generation [Cassell et al, 2001] systems. The inclusion of such systems would add greatly to the realism of any simulation which uses the PPA architecture. However, these are both extremely complex research areas in their own right.

7.2.5 Further Evaluation

Chapter six described an evaluation scheme and how it was applied to the PPA architecture. Although the evaluations performed satisfactorily demonstrate the capabilities of the system, there is plenty of scope for more experiments to be undertaken. In particular the PPA believability experiment could be repeated, comparing the PPA architecture against other systems for the control of virtual humans. In practice this is difficult as other systems must be implemented, or comparative simulations must be created using them. Also, to further evaluate the effect that the PPA architecture has on the games in which it is used, a more in-depth play-test could be undertaken.

7.2.6 Addition of a Goal Based Planning Unit

The PPA architecture was developed to control the behaviours of support characters in character-centric games. However, in such games there is also a need for characters that proactively oppose the player, for example by pursuing goals contrary to those of the player, or even racing the player towards a common goal. The PPA architecture could be used to control such characters if extended to include a *goal based planning* unit. This extended version of the architecture is shown in figure 7-2.

The planner could use logical inference in order to formulate plans to achieve a character's goals. This could be implemented using an existing inference engine such as STRIPS [Fikes & Nilsson, 1971], or to overcome the resource limitations imposed by games, a more sophisticated system such as those developed by Funge [1999] and Narayek [2001]. One of the key challenges in making this addition to the architecture would be in the creation of an action selection mechanism, which would liaise between the original components of the architecture and the goal based planner.

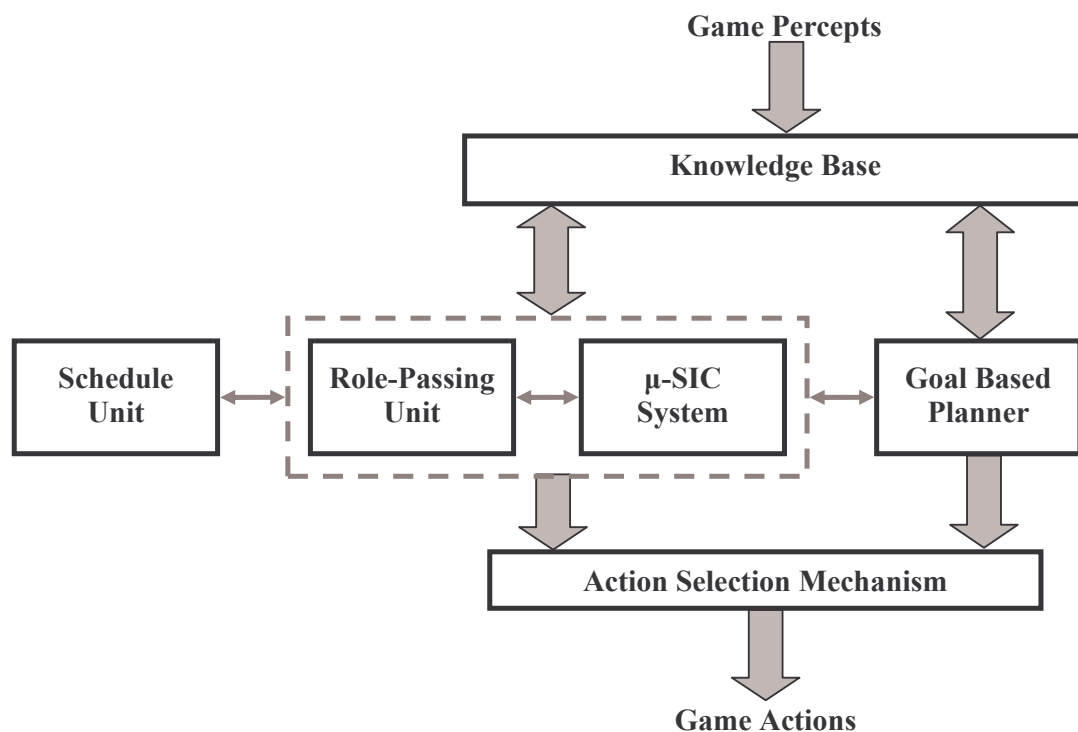


Figure 7-2: A schematic of an extended version of the PPA architecture which includes a goal-based planning unit

7.2.7 Implementation Optimization

Within the evaluation on computational issues, presented in chapter 6, resource related issues were addressed, and it was shown that the PPA architecture is suitable for use within modern computer games. However, at present the implementation of the architecture has not been heavily optimised. By performing a thorough optimization upon the system it is expected that its performance could be greatly improved, allowing the simulation of much larger numbers of NPCs.

7.2.8 Further Use of Level-of-Detail AI

Although level-of-detail AI is already used within the implementation of the PPA architecture, it is only done so to a limited extent. As the size of simulated worlds increases, it is expected that many more areas in which this technique might be applied will emerge. In particular, it would be useful to integrate the technique into the core components of the architecture.

7.2.9 Development of GUI based Design Tools

As the PPA architecture is intended for use in the creation of commercial computer games, it is important that game designers, who are generally not trained programmers, are able to

Thesis Summary, Conclusions & Suggestions for Future Work

use it in order to populate their games with NPCs. The creation of GUI based tools for the design of schedules and PPAFCMs would greatly ease this process. The usefulness of such tools has already been demonstrated by the tool created for designing game worlds for Berlin-End-Game.

7.3 The Last Word

This work has presented the proactive persistent agent architecture, an intelligent agent architecture for driving the behaviours of computer controlled support characters in character-centric games. Based on the notion of situational intelligence, this architecture allows game designers populate their game worlds with characters capable of behaving believably in a wide range of diverse situations, and displaying sophisticated social behaviours. Such characters allow the creation of games in which the realism of the virtual characters begins to approach that of the virtual worlds in which games are set.

References

While every effort has been made to ensure that all URLs given in this list are valid at the time of going to print, no guarantees can be made as to their future validity.

[Abbattista et al, 2002] F. Abbattista, P. Lops, G. Semeraro, V. Andersen & H.H.K. Andersen, “Evaluating Virtual Agents for E-Commerce”, In *Proceedings of the 1st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '02) - Conference Workshop: Embodied Conversational Agents - Let's Specify and Evaluate Them.* (2002)

[Alexander, 2002a] B. Alexander, “The Beauty of Response Curves”, In *AI Game Programming Wisdom*, S. Rabin (Ed.), Charles River Media, pp. 78-82. (2002)

[Alexander, 2002b] T. Alexander, “An Optimized Fuzzy Logic Architecture for Decision Making”, In *AI Game Programming Wisdom* (Editor S. Rabin), Charles River Media. (2002)

[Alt & King, 2002] G. Alt & K. King, “A Dynamic Reputation System Based on Event Knowledge”, In *AI Game Programming Wisdom* (Ed: S Rabin), pp 426 – 435. (2002)

[Axelrod, 1976] R. Axelrod, “Structure of Decision”, Princeton University Press, Princeton, New Jersey. (1976)

[Aylett & Luck, 2000] R. Aylett & M. Luck, “Applying Artificial Intelligence To Virtual Reality: Intelligent Virtual Environments”, In *Applied Artificial Intelligence*, 14(1), pp. 3-32. (2000)

[Aylett et al, 2000] R. Aylett, K. Dautenhahn, J.E. Doran, M. Luck, S. Moss & M. Tennenholtz. “Can Models of Agents be Transferred between Different Areas?”, In *Knowledge Engineering Review*, 15(2), pp. 199-203. (2000)

[Badler et al, 1999] N.I. Badler, R. Bindiganavale, J. Bourne, J. Allbeck, J. Shi & M. Palmer, “Real Time Virtual Humans”, In *Proceedings of the International Conference on Digital Media Futures.* (1999)

[Bates et al, 1992] J. Bates, A. Bryan Loyall & W. Scott Reilly, “Broad Agents”, In *Proceedings of the AAAI Spring Symposium on Integrated Intelligent Architectures*, Stanford University. (1991)

[Bates, 1992] J. Bates, “The Nature of Characters in Interactive Worlds and the Oz Project”, *Technical Report CMU-CS-92-200*, School of Computer Science, Carnegie Mellon University. (1992)

[Berger, 2002] L. Berger, “Scripting: Overview and Code Generation”, In *AI Game Programming Wisdom* (Editor S. Rabin), Charles River Media. (2002)

[Bishop, 1995] C.M. Bishop, “Neural Networks for Pattern Recognition”, Clarendon Press, Oxford. (1995)

[Blumberg, 1996] B. Blumberg, “Old Tricks, New Dogs: Ethology and Interactive Creatures”, Ph.D. Thesis, Media Lab, Massachusetts Institute of Technology. (1996)

[Brockington, 2002] M. Brockington, “Level-Of-Detail AI for a Large Role-Playing Game”, In *AI Game Programming Wisdom*, S. Rabin (Ed.), Charles River Media, pp. 419-425. (2002)

[Brooks, 1985] R. Brooks, “A Robust Layered Control System for a Mobile Robot”, MIT AI Lab Memo 864. (1985)

[Burke et al, 2001] R. Burke, D. Isla, M. Downie, Y. Ivanov & B. Blumberg, “Creature Smarts: The Art and Architecture of a Virtual Brain”, In *Proceedings of the Game Developers Conference 2001 (GDC 2001)*, pp 147 - 166. (2001)

[Buche et al, 2002] C. Buche, M. Parenthoën & J. Tisseau, “Learning by Imitation of Behaviours for Autonomous Agents”, In *Proceedings of Game-On 2002: The 3rd International Conference on Intelligent Games and Simulation*, pp 89-93. (2002)

[Caicedo & Thalmann, 2000] A. Caicedo & D. Thalmann, “Virtual Humanoids: Let them be Autonomous without Losing Control”, In *Proceedings of the 4th International Conference on Computer Graphics and Artificial Intelligence.* (2000)

[Carless, 2003] S. Carless, “Marc Laidlaw On Story And Narrative In Half-Life”, Gamasutra Article, August, (2003)

Available at www.gamasutra.com/features/20030808/carless_01.shtml

[Carlson & Hodgins, 1997] D.A. Carlson & J.K. Hodgins, “Simulation Levels of Detail for Real-Time Animation”, In *Proceedings of Graphics Interface '97*, pages 1-8. (1997)

- [Cassell et al, 2001] J. Cassell, H. Vilhjálmsón & T. Bickmore, “BEAT: Behavior Expression Animation Toolkit”, In *Proceedings of SIGGRAPH 2001*, pp 477–486. (2001)
- [Coen, 1995] M. Coen, “SodaBot: A Brief Introduction”, A Presentation Given at the Massachusetts Institute of Technology. (1995)
Available at: www.ai.mit.edu/people/sodabot/slideshow/total/index.html
- [Charles et al, 2002] F. Charles, J.L. Lugin, M. Cavazza, & S.J. Mead, “Real-Time Camera Control for Interactive Storytelling”, In *Proceedings of Game-On 2002: The 3rd International Conference on Intelligent Games and Simulation*, pp 73-76. (2002)
- [Christian, 2002] M. Christian, “A Simple Inference Engine for a Rule Based Architecture”, In *AI Game Programming Wisdom* (Editor S. Rabin), Charles River Media. (2002)
- [Coco, 1997] D. Coco, “Creating Intelligent Creatures”, *Computer Graphics World*, July. (1997)
- [Deighton, 1983] L. Deighton, “Berlin Game”, Htuchinsom & Co., London. (1983)
- [DePristo & Zubek, 2001] M.A. DePristo & R. Zubek, “being-in-the-world”, In *Proceedings of the 2001 AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*. (2001)
- [Dickerson & Kosko, 1996] J. Dickerson & B. Kosko, “Virtual Worlds as Fuzzy Dynamical Systems”, In *Technology for Multimedia*, B Sheu (Ed.), IEEE Press. (1996)
- [Dingliana, 2000] J. Dingliana & C. O’Sullivan, “Graceful Degradation of Collision Handling in Physically Based Animation”, In *Computer Graphics Forum*, Vol 19, Number 3, pp 239-247. (2000)
- [Drennan, 2003] P. Drennan, “Conversational Agents: Creating Natural Dialogue Between Players and Non-Player Characters”, In *AI Game Programming Wisdom 2* (Ed S. Rabin), Charles River Media. (2003)
- [Dybsand, 2003] E. Dybsand, “AI Middleware: Getting into Character, Parts 1 - 5”, *Gamasutra Articles*. (2003)
Available at: www.gamasutra.com/features/20030721/dybsand_01.shtml
- [Evans, 2002] R. Evans, “Varieties of Learning”, In *AI Game Programming Wisdom* (Editor S. Rabin), Charles River Media. (2002)
- [Eysenck & Rachmann, 1965] H.J. Eysenck & S. Rachman, “The Causes and Cures of Neuroses”, London: Routledge and Kegan Paul. (1965)
- [Fairclough & Cunningham, 2003] C. Fairclough & P. Cunningham, “A Multi-Player Case Based Story Engine”, In *Proceedings of Game-On 2003: The 4th International Conference on Intelligent Games and Simulation*, pp 41-46. (2003)
- [Faloutsos et al, 2001] P.M. Faloutsos, M. van de Panne & D. Terzopoulos, “The Virtual Stuntman: Dynamic Characters with a Repertoire of Autonomous Motor Skills”, *Computers and Graphics*, 25(6), pp 933–953. (2001)
- [Feng-Hsiung, 2002] H. Feng-Hsiung, “Behind Deep Blue: Building the Computer that Defeated the World Chess Champion”, Princeton University Press. (2002)
- [Fermier, 2002] R. Fermier, “Building a Data-Driven Game Engine for Age of Mythology”, In *Proceedings of the 2002 Game Developers Conference*. (2002)
Web lecture available at <http://www.gamasutra.com/features/slides/fermier/index.htm>
- [Fikes & Nilsson, 1971] R. Fikes & N. Nilsson, N., “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving” In *Artificial Intelligence*, vol. 2, pp. 198-208. (1971)
- [Foner, 1993] L. Foner, “What’s an Agent, Anyway? A Sociological Case Study”, *Agents Memo 93-01*, Agents Group, Media Lab, Massachusetts Institute of Technology. (1993)
- [Forbus et al, 1991] K. Forbus, Nielsen, P. and Faltings, B. “Qualitative Spatial Reasoning: The CLOCK Project”, In *Artificial Intelligence*, 51 (1-3), October, 1991. (1991)
- [Forbus et al, 2001] K. Forbus, J. Mahoney, & K. Dill, “How Qualitative Spatial Reasoning Can Improve Strategy Game AIs”, In *Proceedings of the AAAI Spring Symposium on AI and Interactive Entertainment 2001*. (2001)
- [Franklin & Grasser, 1996] S. Franklin & A. Graesser, “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents”, In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag. (1996)

- [Funge, 1999] J. Funge, "AI for Games and Animation: A Cognitive Modeling Approach", A. K. Peters. Natick, MA. (1999)
- [Giang et al, 2000] T. Giang, R. Mooney, C. Peters & C. O'Sullivan, "ALOHA: Adaptive Level Of Detail for Human Animation. Towards a new Framework", In *the Proceedings of the Eurographics 2000 Short Papers Programme*. (2000)
- [Generation5, 2001] "Interview with Jeff Hannan at Generation5.org". (2001)
Available at www.generation5.org/content/2001/hannan.asp
- [Hasegawa et al, 1997] T. Hasegawa Y.I. Nakano & T. Kato, "A Collaborative Dialogue Model Based on Interaction Between Reactivity and Deliberation", In *Proceedings of the First International Conference on Autonomous Agents*, pp. 83 – 87. (1997)
- [Hawes, 2002] N. Hawes, "An Anytime Planning Agent For Computer Game Worlds", In *Proceedings of the Workshop on Agents in Computer Games at The 3rd International Conference on Computers and Games (CG'02)*, pp 1-14. (2002)
- [Hayes-Roth & Doyle, 1998] B. Hayes-Roth & P. Doyle, "Animate Characters", In *Autonomous Agents and Multi-Agent Systems*, vol. 1 (2), pp. 195-230. (1998)
- [Hill et al, 1997] R.W. Hill Jr., J. Chen, J. Gratch, P. Rosenbloom & M. Tambe, "Intelligent Agents for the Synthetic Battlefield: A Company of Rotary Wing Aircraft", In the *Proceedings of Innovative Applications of Artificial Intelligence (IAAI-97)*. (1997)
- [Hodgins et al, 1998] J.K. Hodgins, J.F. O'Brien, J. Tumblin, "Judgments of Human Motion with Different Geometric Models", In *IEEE: Transactions on Visualization and Computer Graphics*, December 1998, Vol. 4, No. 4., pp 307-316. (1998)
- [Holland, 1973] J.H. Holland, "Genetic Algorithms and the Optimal Allocation of Trials", In *SIAM Journal on Computing*, 2(2):88-105. (1973)
- [Holmblad & Ostergaard, 1982] L.P. Holmblad, & J.J. Ostergaard, "Control of a Cement Kiln by Fuzzy Logic", In *Fuzzy Information and Decision Processes* (Editors M.M. Gupta & E. Sanchez), pp 389 – 399. (1982)
- [Höök, 2002] K. Höök, "Evaluation of Affective Interfaces", In *Proceedings of the 1st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '02) - Conference Workshop: Embodied Conversational Agents - Let's Specify and Evaluate Them*. (2002)
- [Horswill & Zubek, 1999] I.D. Horswill & R. Zubek, "Robot Architectures for Believable Game Agents", In *Proceedings of the 1999 AAAI Spring Symposium on Artificial Intelligence and Computer Games*, AAAI Technical Report SS-99-02. (1999)
- [Houlette & Fu, 2003] R. Houlette & D. Fu, "The Ultimate Guide to FSMs in Games", In *AI Game Programming Wisdom 2* (Ed. S. Rabin), Charles River Media. (2003)
- [IGDA, 2003] "Working Group on Rule-Based Systems", International Games Development Association. (2003)
Available at: www.igda.org/ai/report-2003/aiisc_rule_based_systems_report_2003.html
- [Isbister & Doyle, 2002] K. Isbister & P. Doyle, "Design and Evaluation of Embodied Conversational Agents: A Proposed Taxonomy", *Proceedings of the 1st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '02) - Conference Workshop: Embodied Conversational Agents - Let's Specify and Evaluate Them*. (2002)
- [Jones et al, 1999] R.M. Jones, J.E. Laird, P.E. Nielsen, K. Coulter, P. Kenny, & F. Koss, "Automated Intelligent Pilots for Combat Flight Simulation", In *AI Magazine*, Spring 1999, Vol. 20, No. 1, pp. 27-42. (1999)
- [Kallmann & Thalmann, 1998] M. Kallmann, & D. Thalmann, "Modeling Objects for Interaction Tasks", In *Proceedings of EGCAS98*, pp.73-86. (1998)
- [Khoo et al, 2002] A. Khoo, G. Dunham, N. Trienens & S. Sood, "Efficient, Realistic NPC Control Systems using Behavior-Based Techniques", AAAI Technical Report SS-02-01, AAAI Press. (2002)
- [Khoo & Zubek, 2002] A. Khoo, A. & R. Zubek, "Applying Inexpensive AI Techniques to Computer Games", In *IEEE Intelligent Systems Special Issue on Interactive Entertainment*, Vol. 17, No. 4, pp 48 - 53. (2002)

- [Klein, 2002] J. Klein, “breve: a 3D Simulation Environment for the Simulation of Decentralized Systems and Artificial Life”, In *Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems*. (2002)
- [Kosko, 1986] B. Kosko, “Fuzzy Cognitive Maps”, In *International of Journal Man-Machine Studies*, 24:65-75. (1986)
- [Kosko, 1993] B. Kosko, “Fuzzy Thinking: The New Science of Fuzzy Logic”, Harper Collins Publishers. (1993)
- [Kosko, 1997] B. Kosko, “Fuzzy Engineering”, Prentice Hall International. (1997).
- [Laird & Newell, 1983] J.E. Laird & A. Newell, “A Universal Weak Method”, Tech Report #83-141, Carnegie Mellon University Computer Science Department. (1983)
- [Laird et al, 1984] J.E. Laird, P.S. Rosenbloom, & A. Newell, “Towards Chunking as a General Learning Mechanism”, In *Proceedings of the 1984 National Conference on Artificial Intelligence (AAAI)*, pp 188 - 192. (1984)
- [Laird, 2000] J.E. Laird, “An Exploration into Computer Games and Computer Generated Forces”, In *Proceedings of the 8th Conference on Computer Generated Forces and Behaviour Representation*. (2000)
- [Laird & van Lent, 2000] Laird, J.E., M. van Lent, “Human-Level AI’s Killer Application: Interactive Computer Games”, In *Proceedings of the 17th National Conference on Artificial Intelligence*. (2000)
- [Laird et al, 2002] J.E. Laird, M. Assanie, B. Bachelor, N. Benninghoff, S. Enam, B. Jones, A. Kerfoot, C. Lauver, B. Magerko, J. Sheiman, D. Stokes, S. Wallace “A Test Bed for Developing Intelligent Synthetic Characters”, In *Proceedings of the 2002 AAAI Spring Symposium on AI and Interactive Entertainment*. (2002)
- [Lang, 1995] P.J. Lang, “The Emotion Probe: Studies of motivation and attention”, In *A Study in the Neuroscience of Love and Hate*, Lawrence Erlbaum Associates. (1995)
- [Laramée, 2002a] F. D. Laramée, “A Rule-Based Architecture Using the Dempster-Schafer Theory”, In *AI Game Programming Wisdom* (Editor S. Rabin), Charles River Media. (2002)
- [Laramée, 2002b] F.D. Laramée, “Genetic Algorithms: Evolving the Perfect Troll”, In *AI Game Programming Wisdom* (Editor S. Rabin), Charles River Media. (2002)
- [Lebowitz, 1985] M. Lebowitz, “Story-Telling as Planning and Learning”, In *Poetics*, Vol. 14 (6). (1985)
- [Leonard, 2003] T. Leonard, “Building an AI Sensory System: Examining The Design of Thief: The Dark Project”, In *Proceedings of the Game Developers Conference 2003*. (2003)
Available at: www.gamasutra.com/gdc2003/features/20030307/leonard_01.htm
- [Levison, 1996] L. Levison, “Connecting Planning and Acting via Object-Specific Reasoning”, PhD Thesis, Dept. of Computer and Information Science, University of Pennsylvania, USA. (1996)
- [Lloyd et al, 1984] P. Lloyd, A. Mayes, A.S.R. Manstead, P.R. Meudell & H.L. Wagner, “Introduction to Psychology: An Integrated Approach”, Fontana Paperbacks. (1984)
- [Loyall, 1997] B. Loyall, “Believable Agents: Building Interactive Personalities”, Ph.D. Thesis, Carnegie Mellon University. (1997)
- [Määttä, 2002] A. Määttä, “Realistic Level Design for Max Payne”, In *Proceedings of the 2002 Game Developers Conference (GDC 2002)*. (2002)
- [Mac Namee et al, 2002] B. Mac Namee, S. Dobbyn, P. Cunningham & C. O’Sullivan “Men Behaving Appropriately: Applying the Role Passing Technique to the ALOHA System”, In *Proceedings of Artificial Intelligence and the Simulation of Behaviour '02*. (2002)
- [Manslow, 2002] J. Manslow, “Learning and Adaptation”, In *AI Game Programming Wisdom* (Editor S. Rabin), Charles River Media. (2002)
- [Maslow, 1970] A.H. Maslow, “Motivation & Personality”, Harper Row. (1970)
- [McCrae & Costa, 1996] R.R. McCrae, & P. T. Costa Jr., “Toward a New Generation of Personality Theories: Theoretical contexts for the five-factor model”, In *J. S. Wiggins (Ed.), The five-factor model of personality: Theoretical perspectives* (pp. 51-87). Guilford. (1996)
- [McCulloch & Pitts, 1943] W.S. McCulloch & W. Pitts, “A Logical Calculus of the Ideas Immanent in Nervous Activity”, In *Bulletin of Mathematical Biophysics*, Vol. 5, pp 115-133. (1943)

References

- [Meehan, 1976] J. Meehan, "The Metanovel: Writing Stories by Computer", Research Report #74. Computer Science Department, Yale University. New Haven, CT. (1976)
- [Minsky & Papert, 1969] M.L. Minsky & S. Papert, "Perceptrons, An Introduction to Computational Geometry", MIT Press, Cambridge, Massachusetts. (1969).
- [Mitchell, 1997] T.M. Mitchell, "Machine Learning", McGraw Hill. (1997)
- [Morris, 1999] D. Morris, "Neural-Net AI and Fuzzy Logic", In *PC Gamer*, July 1999. (1999)
- [Narayek, 2001] A. Narayek, "Constraint Based Agents", Springer Press. (2001)
- [O'Brien, 1996] L. O'Brien, "Fuzzy Logic in Games", In *Game Developer Magazine*, April/May 1996. (1996)
- [O'Sullivan & Dingliana, 2001] C. O'Sullivan & J. Dingliana, "Collisions and Perception" In *ACM Transactions on Graphics*, Vol. 20, No. 3. (2001)
- [Parenthoën et al, 2001] M. Parenthoën, P. Reignier & J. Tisseau, "Put Fuzzy Cognitive Maps to Work in Virtual Worlds", In *Proceedings of the 10th International Conference on Fuzzy Systems (Fuzz-IEEE'01)*, pp 252-255. (2001)
- [Paschedag, 2003] D. Paschedag, "Video Card History (1996 to the Present)", Article at FastSilicon.com, November, 2003. (2003)
Available at: <http://www.fastsilicon.com/showarticle.php?a=28>
- [PC Gamer, 1999] PC Gamer, November issue, 1999. (1999)
- [Perlin & Goldberg, 1996] K. Perlin & A. Goldberg, "Improv: A System for Scripting Interactive Actors in Virtual Worlds", In *Proceedings of the ACM Computer Graphics Annual Conference*, pp 205-216. (1996)
- [Peters et al, 2003] C. Peters, S. Dobbyn, B. Mac Namee & C. O'Sullivan, "Smart Objects for Attentive Agents", In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2003*. (2003)
- [Peters & O'Sullivan, 2002] C. Peters & C. O'Sullivan, "A Memory Model for Autonomous Virtual Humans", In *Proceedings of Eurographics Irish Chapter Workshop 2002*, pp 21-26. (2002)
- [Picard, 1995] R.W. Picard, "Affective Computing", In *Perceptual Computing*, Technical Report 321, MIT Media Lab. (1995)
- [Poole, 2002] S. Poole, "Masterpiece or Mindless Mush?", In *The Sunday Times, Doors Section*, May 12th 2002. (2002)
- [Preece, 1994] J. Preece, "Human-Computer Interaction", Addison-Wesley, Wokingham. (1994)
- [Rao & Georgeff, 1991] A.S. Rao, & M.P. Georgeff, "Modeling Rational Agents within a BDI Architecture", Technical Report 14, Australian AI Institute, Carlton, Australia. (1991)
- [Rao & Georgeff, 1992] A.S. Rao, & M.P. Georgeff, "An Abstract Architecture for Rational Agents", In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR '92)*, pp. 439 - 449. (1992)
- [Reynolds, 1987] C.W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioural Model", In *Computer Graphics*, 21(4), July 1987, pp. 25 -34. (1987)
- [Reynolds, 2003] J. Reynolds, "Team Member AI in an FPS", In *AI Game Programming Wisdom 2* (Ed. S. Rabin), Charles River Media. (2003)
- [Rosenblatt, 1957] F. Rosenblatt, "The Perceptron: A Perceiving and Recognizing Automaton", Technical Report 85-460-1, Project PARA, Cornell Aeronautical Laboratory, Ithaca, New York. (1957)
- [Rosenbloom et al, 1993] P.S. Rosenbloom, J.E. Laird & A. Newell, "The Soar Papers: Readings on Integrated Intelligence", MIT Press. (1993)
- [Rubin, 2003] J. Rubin, "Great Game Graphics... Who Cares", In *Proceedings of the Game Developer's Conference 2003 (GDC 2003)*. (2003)
- [Russell & Norvig, 1995] S. Russell & P. Norvig, "Artificial Intelligence: A Modern Approach", Prentice Hall. (1995)

References

- [Ruttkay et al, 2002] Zs. Ruttkay, C. Dormann & H. Noot, "Evaluating ECAs - What and How?", In *Proceedings of the 1st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '02) - Conference Workshop: Embodied Conversational Agents - Let's Specify and Evaluate Them.* (2002)
- [Ruttkay et al, 2003] Z. Ruttkay, Z. Huang, A. Eliens, "Reusable Gestures for Interactive Web Agents" In *Proceedings of Intelligent Virtual Agents 2003 (IVA '03)*, pp. 80 -87. (2003)
- [Schank & Abelson, 1977] R. Schank, & R. Abelson, "Scripts, Plans, Goals, and Understanding", Hillsdale, NJ: Lawrence Erlbaum. (1977)
- [Schaufler & W. Stürzlinger, 1995] G. Schaufler & W. Stürzlinger, "Generating Multiple Levels of Detail from Polygonal Geometry Models", In *Proceedings of Virtual Environments '95 (Eurographics Workshop on Virtual Environments 1995*, ed. Göbel, pp. 33-41, Springer Verlag. (1995)
- [Scott Neal Reilly, 1996] W. Scott Neal Reilly, "Believable Social and Emotional Agents", Ph.D. Thesis. Technical Report CMU-CS-96-138, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. (1996)
- [Sims,1991] K. Sims, "Artificial Evolution for Computer Graphics", *Computer Graphics*, 25(4), July 1991, pp. 319-328. (1991)
- [Slater, 2003] M. Slater, "How Colourful Was Your Day? Why Questionnaires Cannot Assess Presence in Virtual Environments", In *Presence: Teleoperators and Virtual Environments*, in press. (2003)
- [Smith, 2002] P. Smith, "Polygon Soup for the Programmer's Soul: 3D Path-finding", In *Proceedings of the Game Developer's Conference 2002.* (2002)
- [Stout, 1996] B.W. Stout, "Smart Moves: Intelligent Path-Finding", *Game Developer Magazine*, October 1996. (1996)
- [Terzopoulos et al, 1994] D. Terzopoulos, X. Tu & R. Grzeszczuk, "Artificial Fishes with Autonomous Locomotion, Perception, Behavior and Learning, in a Physical World", In (Editors: P. Maes & R. Brooks) *Proceedings of the Artificial Life IV Workshop*, MIT Press. (1994)
- [Thomas & Johnston, 1995] F. Thomas & O. Johnston, "The Illusion of Life: Disney Animation", Hyperion, New York. (1995)
- [Tolman, 1948] E.C. Tolman, "Cognitive Maps in Rats and Men", In *Psychological Review*, 42, 55, 189-208. (1948)
Available at: psychclassics.yorku.ca/Tolman/Maps/maps.htm
- [Tozour, 2002] P. Tozour, "Introduction to Bayesian Networks and Reasoning Under Uncertainty", In *AI Game Programming Wisdom* (Editor S. Rabin), Charles River Media. (2002)
- [van der Werf et al, 2002] E. van der Werf, J. Uiterwijk, J. van den Herik, "Programming a Computer to Play and Solve Ponnuki-Go", In *Proceedings of Game-On 2002: The 3rd International Conference on Intelligent Games and Simulation*, pp 173-177. (2002)
- [Werbos, 1974] P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", PhD thesis, Harvard University, Cambridge, Massachusetts. (1974)
- [Wikipedia, 2004] "Wikipedia – The Free Encyclopedia", from the *fuzzy set* entry. (2004)
Available at: en.wikipedia.org/wiki/Fuzzy+set
- [Wish et al, 1976] M. Wish, M. Deutsch & S. Kaplan, "Perceived Dimensions of Interpersonal Relations". In *Journal of Personality and Social Psychology*. Vol. 33 (6). (1976)
- [Woodcock, 2000a] S. Woodcock, "AI Roundtable Moderator's Report", In *Proceedings of the Game Developer's Conference 2000 (GDC 2000)*. (2000)
Available at: www.gameai.com/cgdc00notes.html
- [Woodcock, 2000b] S. Woodcock, "Game AI the State of the Industry", In *Game Developer Magazine*, August. (2000)
Available at: www.gamasutra.com/features/20001101/woodcock_01.htm
- [Wooldridge & Jennings, 1995] M. Wooldridge & N. R. Jennings, "Intelligent Agents: Theory and Practice". *The Knowledge Engineering Review*, 10(2):115--152. (1995)
- [Wright & Marshall, 2000] I. Wright, & J. Marshall, "More AI in Less Processor Time: 'Egocentric' AI", *Gamasutra*, June. (2000)
Available at: www.gamasutra.com/features/20000619/wright_01.htm

References

- [Yasunobu et al, 1983] S. Yasunobu, S. Miyamoto & H. Ihara, “Fuzzy Control for Automatic Train Operation System”, In *Proceedings of the 4th IFAC/IFIP/IFORS International Conference on Control in transportation Systems*, pp 33 – 39, Baden-Baden, Germany. (1983)
- [Zadeh, 1965] L. Zadeh, “Fuzzy Sets”, In *Information and Control*, vol. 8, pp 338-353. (1965)
- [Zarozinski, 2002] M. Zarozinski, “An Open-Source Fuzzy Logic Library”, In *AI Game Programming Wisdom* (Editor S. Rabin), Charles River Media. (2002)

Game References

Reference	Title	Developer	Publisher	Year
G-1	Half-Life	Valve	Sierra	1998
G-2	Space Invaders	Toshihiro Nishikado	Taito	1978
G-3	Wolfenstein 3d	id Software	Apogee Software	1992
G-4	Doom	id Software	id Software	1993
G-5	Deus Ex	Ion Storm	Eidos Interactive	2000
G-6	Unreal Tournament	Epic Games	GT Interactive	1999
G-7	Grand Theft Auto Vice City	Rockstar North	Rockstar Games	2002
G-8	Half-Life 2	Valve	Sierra	2004
G-9	Thief: The Dark Project	Looking Glass Studios	Eidos Interactive	1998
G-10	Star Trek Voyager Elite Force	Raven Software	Eidos Interactive	2000
G-11	The Rainbow Six series	Red Storm Entertainment	Ubi Soft	1999
G-12	Grim Fandango	Lucasarts	Lucasarts	1998
G-13	Adventure	William Crowther & Don Woods	-	1976
G-14	Zork	INFOCOM	Personal Software Inc	1980
G-15	The Monkey Island Series	Lucasarts	Lucasarts	1990 to 2000
G-16	The Gabriel Knight Series	Sierra	Sierra	1993 to 1999
G-17	Metal Gear Solid 2: Sons of Liberty	KCEJ	Konami	2001
G-18	Bladerunner	Westwood Studios	Activision	1997
G-19	The Tomb Raider Series	Eidos Interactive	Eidos Interactive	1996 to 2003
G-20	Indiana Jones and the Emperor's Tomb	Lucasarts	Lucasarts	2003
G-21	Baldur's Gate	Bioware	Interplay	1998
G-22	Dungeon Siege	Gas Powered Games	Microsoft Games	2002
G-23	The Final Fantasy Series	Square Enix	Square Enix	to 2003
G-24	Neverwinter Nights: Hordes of the Underdark	Bioware	Atari	2003
G-25	Star Wars Knights of the Old Republic	Bioware	Lucasarts	2003
G-26	Ultima Online	Origin Systems, Inc	EA Games	1997
G-27	Star Wars Galaxies: An Empire Divided	Sony Online Entertainment	Lucasarts	2003
G-28	The Age of Empires Series	Ensemble Studios	Microsoft Games	1993 to 2003
G-29	The Civilization Series	Firaxis	Microprose/Infogames	1995 to 2003

Game References

G-31	The Command & Conquer series	Westwood Studios	Virgin Interactive/EA Games	1995 to 2003
G-32	Warcraft III: Reign of Chaos	Blizzard Entertainment	Blizzard Entertainment	2002
G-33	The Populous Series	Bullfrog Productions Ltd	EA Games	1989 to 1999
G-34	Black & White	Lionhead Studios	EA Games	2001
G-35	The SimCity series	Maxis Software	EA Games	1989 to 2003
G-36	Republic the Revolution	Elixir Studios	Eidos Entertainment	2003
G-37	2002 FIFA World Cup	EA Sports	EA Sports	2002
G-38	Sega Bass Fishing	Sega	Sega	2000
G-39	ESPN International Track & Field	Konami	Konami	2000
G-40	Smash Court Tennis Pro Tournament	Namco	Namco	2002
G-41	Diablo II	Blizzard North	Blizzard Entertainment	2000
G-42	The Sims	Maxis	Maxis	2000
G-43	Grand Theft Auto III	Rockstar North	Rockstar Games	2001
G-44	Hidden & Dangerous	Illusion Softworks	Take Two International	1999
G-45	Medal of Honour: Allied Assault	2015	EA Games	2002
G-46	Creatures	Cyberlife	Mindscape	1996
G-47	Mindrover: The Europa Project	Cognitoy	Cognitoy	1999
G-48	Colin McRae Rally 2.0	Codemasters	Codemasters	2001
G-49	Homeworld	Relic	Sierra Entertainment	1999
G-50	Half-Life: Counter Strike	Valve	Sierra	2000
G-51	Quake	Id Software	Id Software	1996
G-52	Colin McRae Rally 04	Codemasters	Codemasters	2004

Appendices

Appendix A: Fuzzy Logic & Additive Fuzzy Systems

First introduced in the 1960s [Zadeh, 1965], fuzzy logic is a superset of conventional logic, extended to handle the concept of partial truth – thus allowing truth values between *completely true* and *completely false*. Although originally developed as a means to model the uncertainty of natural language, fuzzy logic has been applied to a wide range of application areas - finding particular success in the field of systems control, where applications have been as wide ranging as controlling a cement kiln [Holmblad & Ostergaard, 1982] to piloting subway trains in the Japanese city of Sendai [Yasunobu et al, 1983].

In traditional logic, variables are restricted to taking values of only true or false. This is not the case with fuzzy logic. Rather, variables can take on *fuzzy set* values. A fuzzy set is different from a classical set in that rather than members of the Universe being either in the set or not in the set, members can possess degrees of membership (based upon a *membership-degree function*) in the interval $[0, 1]$. The value 0 means that a member is not included in a set, while 1 describes a fully included member. Members with membership values between 0 and 1 are said to be fuzzy members. The issue of membership can be put more succinctly as, *for the universe X and given the membership-degree function $f : X \rightarrow [0, 1]$, the fuzzy set A is defined as $A = \{(x, f(x)) \mid x \in X\}$.*

To illustrate the ideas behind fuzzy logic, a simple system which regulates the speed of the fan in an air conditioning unit in order to keep a room to its optimum temperature, will be considered. The only control variable for this system is the current air temperature. This variable can take on the fuzzy set values COLD, COOL, JUST-RIGHT, WARM or HOT. A *fuzzy set curve* is drawn for each of these sets in order to show how membership of the set changes as the value of the air temperature changes. These curves define the membership degree functions for each set. Figure A-3 illustrates how membership of the fuzzy set COOL changes with the value of the air temperature. This is juxtaposed against an illustration of the membership function for a conventional set.

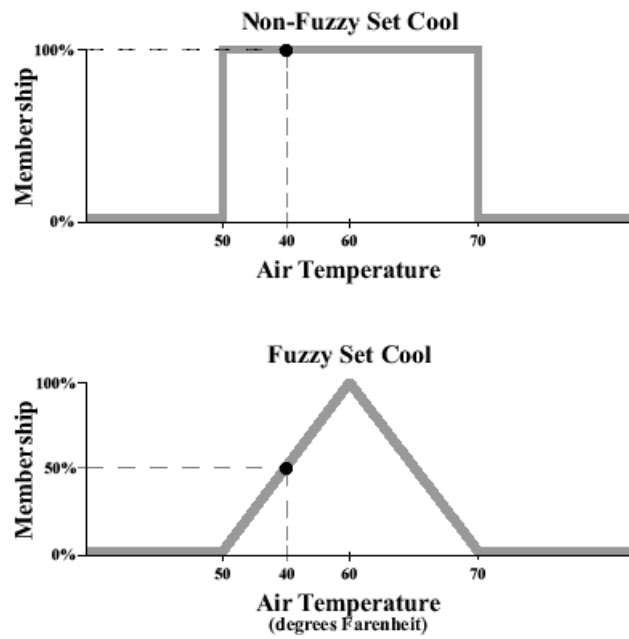


Figure A-7-3: An illustration of membership for the cool set as both a fuzzy and non-fuzzy set

In the illustration of the fuzzy set it can be seen that a value of 60° Fahrenheit has 100% membership in the set, while other values between 50° and 70° have varying fuzzy degrees of membership. For example 40° has membership of 40%. It is important to note that in the non-fuzzy set all of these values would have 100% membership in the set.

Additive fuzzy systems are used to put fuzzy logic to use for the control of real world devices. An additive fuzzy system consists of a set of fuzzy *if-then* rules. These rules have the form “*If input conditions hold, then output conditions hold*” or “*If X is A, then Y is B*”, where X and Y are variables and A and B are fuzzy sets.

Any such system can be considered a function approximator, and each of these fuzzy rules can be considered to approximate a portion of the function in question. If such a function is plotted as a graph, then each rule forms a *fuzzy patch* on top of that graph. Figure A-4 shows how the real function $f: X \rightarrow Y$ can be approximated by a set of fuzzy patches in the input-output product space $X \times Y$. It is important to notice that the small set of large patches shown in figure A-4 (A) performs a much more crude approximation than the larger set of smaller patches shown in figure A-4 (B). Although it is always possible to improve the accuracy of a function approximation by using larger numbers of smaller patches, this comes at the cost of increased complexity and often unrealistic storage requirements.

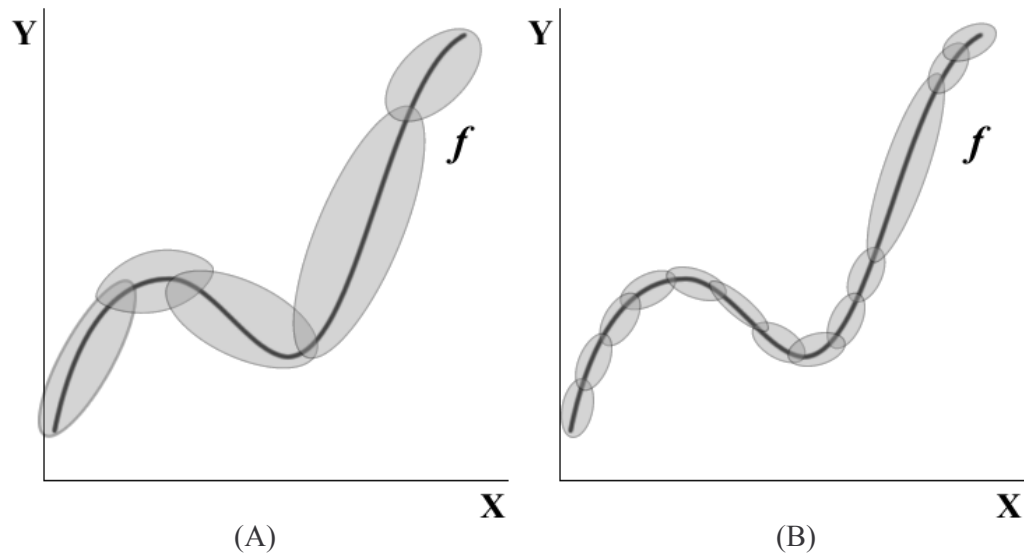


Figure A-7-4: An illustration of how a set of fuzzy patches can be used to approximate a continuous function. (A) shows how a small set of large patches crudely approximates a function, while (B) shows how a larger set of small patches can more accurately approximate the same function

Figure A-5 illustrates a set of Cartesian products $A \times B$ resulting from a set of fuzzy rules. The if parts of these rules are shown on the X axis, where the triangles represent to what extent a rule fires for a particular input value. The value at the apex of the triangle fires the rule maximally, while other values fire each rule to a lesser degree. Similarly, the triangles on the Y axis show how each output value relates to the firing of a rule. An additive fuzzy system fires all of these rules in parallel, and so the input variable X can be said to be a member of all of the fuzzy sets $A_1 - A_5$ to at least some degree (usually zero). The resulting output sets B_j (most of which will be zero) are scaled by the amount that each rule fires, and averaged to give the output fuzzy set B. A defuzzification process is performed on this output set to give a value, or control signal.

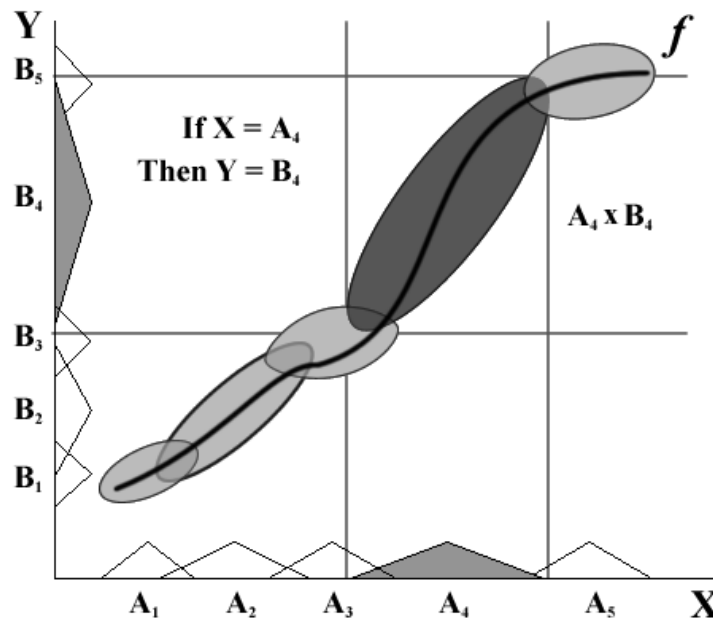


Figure A-7-5: An illustration of a set of if-then rules covering a function in fuzzy patches. A single rule (if X is A4 then Y is B4) has been highlighted

As an example of a fuzzy additive system, the air conditioning unit mentioned previously will now be considered. The purpose of this system is to maintain the temperature in a room constantly at 70° Fahrenheit. The system maps the current temperature in the room to the speed of the output fan. The current temperature is represented by the fuzzy sets COLD, COOL, JUST-RIGHT, WARM and HOT which are shown on the bottom axis of the large graph in figure A-6. The fan’s output speed can be a member of the fuzzy sets OFF, SLOW, MEDIUM, FAST and BLAST which are shown on the left axis of figure A-6. This mapping from input to output is achieved through a series of if-then rules, three of which are shown in the illustration. Each rule maps a fuzzy input state, to a fuzzy output state. For example, the first rule indicates that when the air temperature is COOL, then the fan speed should be turned to SLOW, while the third rule says that when the air temperature is WARM, then the fan speed should be turned up to FAST.

Inputs to the system are presented as exact values, from which membership of the fuzzy input sets is determined. An example of this is shown at the bottom left of figure A-6, where it is determined that the input value of 68° Fahrenheit has 20% membership of the set COOL and 70% membership of the set JUST-RIGHT. Membership of all other fuzzy sets is zero.

Any input to the system causes all of the if-then rules in which the if part is activated to fire. In this example an input of 68° causes “if COOL, then SLOW” and “if JUST-RIGHT, then MEDIUM” to fire. The amount by which a rule fires is determined by its inputs’ membership

in the sets which are part of the if-part of the rule. The then-parts of the rules that fire are scaled according to set membership, and summed to give a system's output set. At the bottom right of figure A-6 the two then parts of the fired rules, SLOW and MEDIUM, are shown scaled, along with the resulting output set. In order to produce a usable output value this output set is defuzzified. In this example defuzzification is achieved by taking the centroid of the output set to give an output value of 47 rpm for the air conditioner's fan speed.

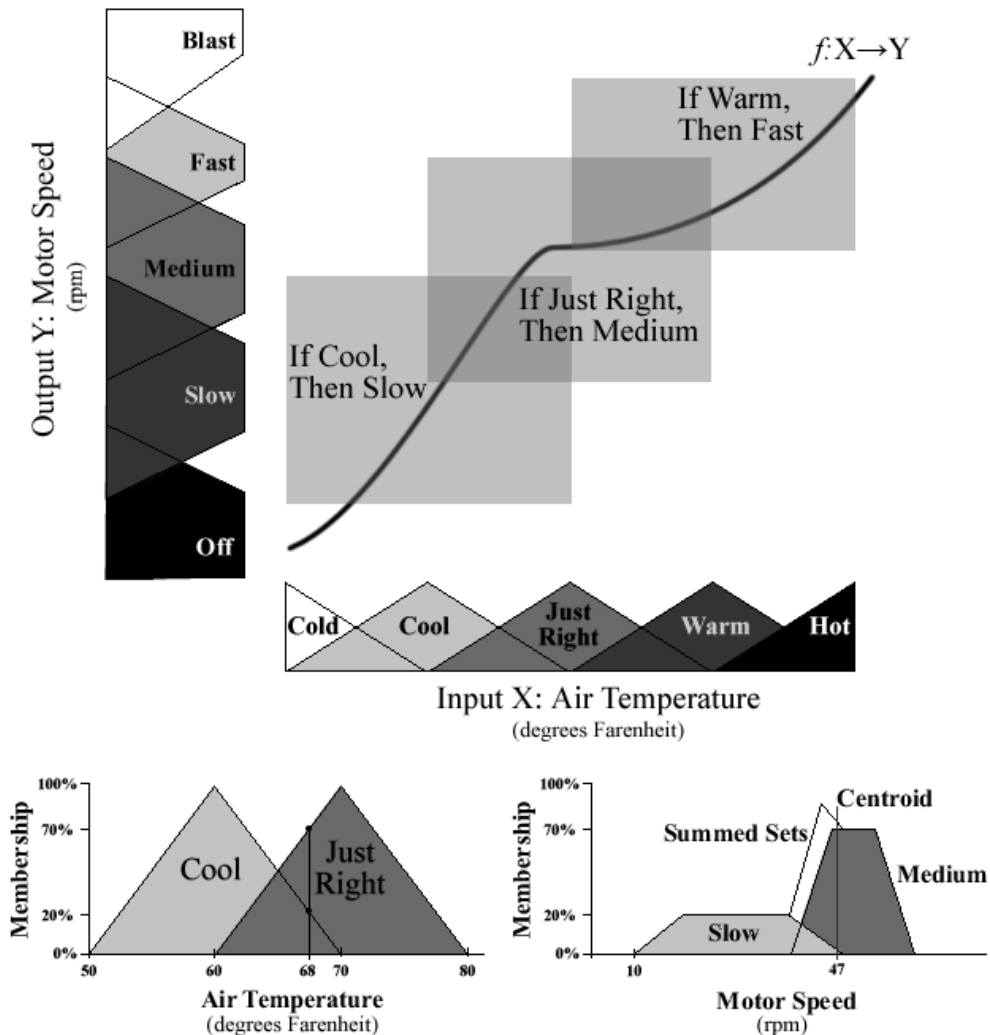


Figure A-7-6: An illustration of the fuzzy additive system used to control the speed of the fan in an air conditioning unit in order to maintain a constant room temperature

This simple example shows the main advantage of fuzzy systems, which is that they can smoothly control a system's output in a much smoother way than a set of rigid rules. This is why fuzzy systems have found such success in industry, and is also what makes them suitable for use in games.

Appendix B: The PPA Believability Experiment

This appendix describes the details of the PPA believability experiment. First the questionnaire will be shown followed by a detailed breakdown of the responses given by subjects.

B.1 Questionnaire

Thank you for taking part in the Proactive Persistent Agent architecture evaluation. The two screens in front of you each show a simulation of a bar environment populated by a collection of virtual humans. Except for the behaviour of these virtual humans, everything about the two simulations is exactly the same. The purpose of this evaluation is to determine the effects of the differences of the virtual humans' behaviours in the two simulations on their believability. This evaluation is not concerned with:

- The quality of the graphics in the simulations
- The quality, or realism, of animations in the simulations
- Agents' ability to plan paths around the environment and avoid each other

Rather, the evaluation is concerned with the motivations driving virtual humans' behaviours, and the actions which they choose to perform. Also, please ignore the behaviour of the barman character as this is the same in both cases.

Question 1: Please tick the box indicating the screen displaying the simulation in which you thought agents' behaviour was most believable:

Screen 1: Screen 2:

Question 2: In the simulation that you have chosen try to describe the behaviour of the bar patron characters. Focus on what is driving the characters' behaviour and what actions they choose to perform.

Question 3: For the following statements please indicate whether you agree or disagree with the following statements for **BOTH** simulations:

Agents appear to act under their own volition																													
<table border="1" style="width: 100%; text-align: center;"> <tr> <td>Strongly Disagree</td> <td>Disagree</td> <td>No Opinion</td> <td>Agree</td> <td>Strongly Agree</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </table> <p>Screen 1</p>					Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>Strongly Disagree</td> <td>Disagree</td> <td>No Opinion</td> <td>Agree</td> <td>Strongly Agree</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </table> <p>Screen 2</p>					Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree																									
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																									
Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree																									
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																									
Agents' behaviours appear to be goal driven																													
<table border="1" style="width: 100%; text-align: center;"> <tr> <td>Strongly Disagree</td> <td>Disagree</td> <td>No Opinion</td> <td>Agree</td> <td>Strongly Agree</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </table> <p>Screen 1</p>					Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>Strongly Disagree</td> <td>Disagree</td> <td>No Opinion</td> <td>Agree</td> <td>Strongly Agree</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </table> <p>Screen 2</p>					Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree																									
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																									
Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree																									
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																									
Agents behaviours possess a random component																													
<table border="1" style="width: 100%; text-align: center;"> <tr> <td>Strongly Disagree</td> <td>Disagree</td> <td>No Opinion</td> <td>Agree</td> <td>Strongly Agree</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </table> <p>Screen 1</p>					Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>Strongly Disagree</td> <td>Disagree</td> <td>No Opinion</td> <td>Agree</td> <td>Strongly Agree</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </table> <p>Screen 2</p>					Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree																									
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																									
Strongly Disagree	Disagree	No Opinion	Agree	Strongly Agree																									
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																									

B.2 Experiment Results

13 subjects took part in the PPA believability experiment, however one of these subjects gave only responses to question 1 and 2 and no response to question 3. Subjects responses are shown in the table below. For question 1 a response of P indicates that a subject chose the PPA based simulation as the most believable, while a response of R indicated that a subject chose the random simulation as more believable. For question 3, responses refer to the following:

SD	D	NO	A	SA
Strongly disagree	Disagree	No opinion	Agree	Strongly agree

Table B-7-1: Subjects' responses to questions 1 and 3 of the PPA believability experiment

	Subject Responses												
	A	B	C	D	E	F	G	H	I	J	K	L	M
Question 1	P	R	P	P	P	P	P	R	P	P	P	R	P
Question 3: PPA	SA	A	SA	A	A	-	A	NO	A	A	A	D	A
Statement 1 Random	SD	A	NO	D	D	-	SD	A	D	D	NO	A	D

Question 3: PPA	SA	NO	A	NO	SA	-	A	NO	A	A	SA	NO	SA
Statement 2 Random	D	A	D	A	A	-	A	A	NO	D	A	A	D
Question 3: PPA	D	A	A	NO	D	-	SD	A	NO	D	D	A	A
Statement 3 Random	SA	A	A	A	A	-	SA	NO	A	A	SA	NO	SA

Appendix C: The Berlin-End-Game Evaluation Experiment Questionnaire

Thank you for taking part in the Berlin-End-Game evaluation. Please play both versions of Berlin-End-Game (game-1 and game-2), each for approximately 25 minutes. Once this is complete please answer the following questions.

Name:

Familiarity with Computer Games (delete as appropriate):

Not Familiar - Somewhat Familiar - Familiar - Very Familiar

Question 1: Was there a noticeable difference between version one and version two of the game?

Yes No

If you answered **YES**, please describe the differences that you noticed:

Question 2: If you answered **YES** to question 1, in which version of the game did you find the game-playing experience most immersive:

Version 1 Version 2

Please explain your choice:

Appendix D: Implementation of the PPA Architecture

The PPA architecture was implemented entirely in object-oriented C++, the de facto standard language for computer game development. The implementation strove to satisfy the following goals:

- To make the integration of the architecture into existing game applications as easy as possible
- To ensure the implementation was as modular as possible so as components can be included and excluded with ease

In order to go some way towards satisfying the first criterion, the implementation of NPCs that make use of the PPA architecture is split into two classes: CHARACTER and BRAIN. Every NPC within a game world is implemented through an instance of the CHARACTER class, each of which contains an instance of the BRAIN class. The use of separate BRAIN and CHARACTER classes eases the process of the integration of the PPA architecture into existing applications. All application specific details (for example, these might include the model used to visually represent a character and a character's position in the world) are contained within the CHARACTER class which can fundamentally change from one application to the next. The separation between the CHARACTER class and BRAIN class is illustrated in figure D-7.

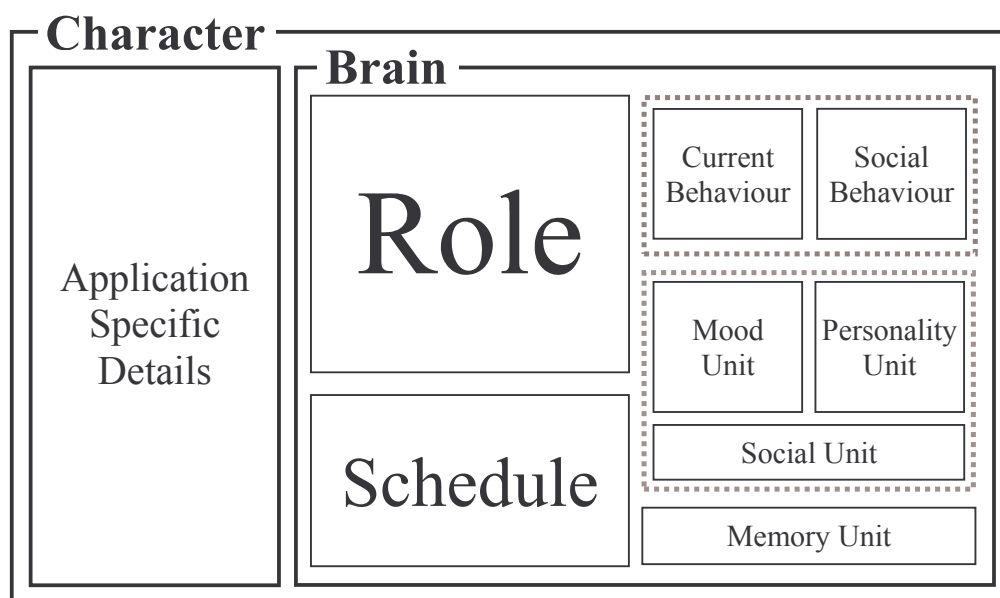


Figure D-7-7: An illustration of the division of the implementation of the PPA architecture into a Character and Brain class

Within the BRAIN class all of the aspects of the PPA architecture that have been discussed previously (particularly in chapter 4) are implemented. The following sections will discuss the implementation of each of these components.

D.1 Implementing the Schedule Unit

The purpose of the schedule unit is to dictate where an NPC should be and what role they should adopt at any time within a simulation. As was previously discussed in section 4.3, schedules can be written by game designers in XML. These XML schedules (an example of which is given in appendix G) are loaded by a character's BRAIN object at the beginning of a simulation. A character's schedule is loaded into a SCHEDULE object which stores an ordered list of SCHEDULEENTRY objects, each of which corresponds to a schedule entry given within the XML listing. Each SCHEDULEENTRY object stores the following pieces of information:

- **Start Time:** The time at which an NPC should adopt the role associated with this schedule entry
- **End Time:** The time at which an NPC should discard the role associated with this schedule entry
- **Area:** The area of the game world in which the NPC should be when they first adopt the role associated with this schedule entry
- **Role Details:** The details of the role associated with this schedule entry, stored as XML

The SCHEDULE object can be considered a direct implementation of the schedule unit discussed previously. As a simulation progresses, the schedule unit is regularly polled to determine if the adoption of a new schedule entry is required by comparing the current simulation time to the start and end times of relevant schedule entries. When a new schedule entry becomes active, an NPC's current role is removed, the character navigates to the location associated with the new schedule entry (navigation is achieved through the adoption of a special role designed for the purpose), and finally, the role associated with the new schedule entry is adopted. All of this is achieved by the role-passing unit which will be discussed in the next section.

D.2 Implementing the Role-Passing Unit

The role-passing unit is the most important within the implementation of the PPA architecture. It is responsible for the control of NPC behaviour when a particular role is has been adopted, and for the process by which roles are adopted and discarded. The BRAIN object of each NPC contains an instance of the ROLE class which maintains a list of the names of all of the roles which a character has currently adopted (usually just a basic role and one role specific to the character's current situation) and a PPAFCM which stores the nodes associated with all the roles which are currently adopted. The PPAFCM is represented by an instance of the FCM class. An object of this type stores all of the elements (including concepts, motivations, links etc) which are a part of a particular PPAFCM. These nodes are stored in a series of hash-tables which map a particular role name to the objects which are part of the PPAFCM for that role. There is a master list which stores all of the PPAFCM objects for a particular role, and a separate list for each type of object which stores links to the PPAFCM objects of that specific type which are stored within the master list. This hash-table structure is used to enable easy access to the PPAFCM objects associated with a particular role.

The various PPAFCM objects which can be used to implement a particular role are implemented in a class hierarchy, a portion of which is shown in figure D-8. At the top of this hierarchy is the FCMOBJECT class. Every possible element which can be included within a PPAFCM extends this class, which itself simply stores a label for the element and the name of the role of which it is a part. The FCMOBJECT class is extended by the FCMLINKOBJECT, FCMEVENTOBJECT, FCMTHRESHOLDOBJECT and FCMNODEOBJECT classes.

The FCMLINKOBJECT class is used to represent links between other types of objects, and is in turn extended to represent more exotic links such as the AND link and the OR link.

FCMNODEOBJECT is the most heavily extended class in the hierarchy. Out of this class all of the node objects used within a PPAFCM (i.e. concepts, motivations, rules and persistent flags) are created as sub classes. The FCMNODEOBJECT class includes a list of input and output links amongst its attributes, and adds the virtual methods *updateActivation* and *getNodeActivationLevel* through which the activation of node objects can be updated and queried. The way in which these methods function is implemented in the classes which extend FCMNODEOBJECT to implement particular node types, for example action nodes or concept nodes. Many of these implementations are themselves further extended to

implement specific flavours of certain node types. For example, the FCM ACTION OBJECT class is extended to represent specific objects such as FCM ACTION OBJECT_GoTo or FCM ACTION OBJECT_SATISFYNEED which represent particular actions which agents can perform.

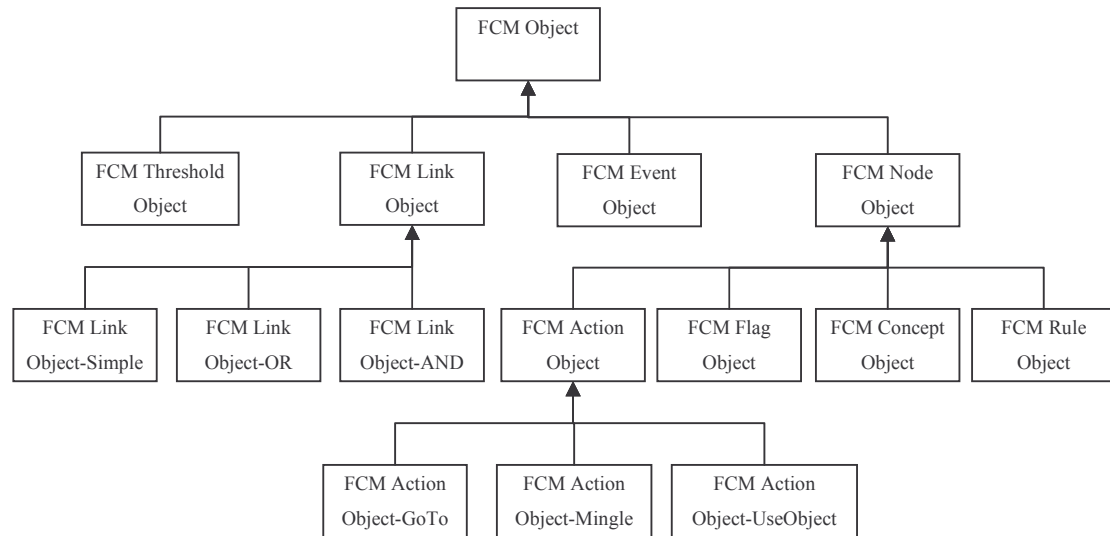


Figure D-7-8: A portion of the class hierarchy used to implement the different PPAFCM objects used within the FCM class

The details of the PPAFCM objects associated with a particular role are stored as XML within each SCHEDULEENTRY object. These XML listings are similar to those shown in the schedule listing given in appendix G. When a new role is adopted a set of PPAFCM objects are instantiated based on the XML listing. All of these new objects are added to the FCM object stored by a character’s ROLE object. Converting an XML listing to a set of PPAFCM objects is a trivial task and involves simply parsing the XML listing and creating appropriate objects based on the details therein. A library of utility methods was created for this purpose.

Once a PPAFCM has been filled with PPAFCM objects the process of updating it is relatively simple. At each update step within the game world each PPAFCM object is updated individually. Updating PPAFCM objects involves performing any task specific to a particular object (for example incrementing a motivation or evaluating a rule) and then summing the activations coming from the object’s input links. Updating objects in a PPAFCM is performed in a particular order beginning with motivations and rules, then concept nodes and finally action nodes. This order stems from the fact that concept objects are the key components within any PPAFCM. By firstly updating motivations and rules, new activation is allowed to enter a PPAFCM, and then input into the concept objects through which activation propagates. Finally, once the concept activations have settled, the

activations at the action objects are computed. The frequency at which characters' PPAFCMs are updated can be adjusted so as to lower the processing burden to the game engine. This can be done without greatly affecting characters' perceived behaviours.

As well as updating the PPAFCM at each character update cycle, the current behaviour being pursued by each NPC is also updated. The BRAIN object of each NPC contains two behaviour objects - one representing the primary action which the character is currently pursuing and a second which implements a social behaviour, allowing NPCs concurrently interact with each other while performing other actions. The behaviours which characters pursue are atomic and operate at a relatively high level, performing such actions as moving to a particular location, using a particular type of object or satisfying a particular need. A character's current and social behaviours are updated at every character update cycle so that the character is always pursuing some behaviour.

Within the implementation of the PPA architecture behaviours are implemented within a class hierarchy with a generalised BEHAVIOUR at the top, which is extended to implement specific behaviours such as using an object (BEHAVIOUR_USEOBJECT) or moving to a location (BEHAVIOUR_GOTO). A portion of this class hierarchy is shown in figure D-9. The BEHAVIOUR class contains a virtual method *stepOn* which is implemented by each of the class's children and controls the progression of a particular behaviour.

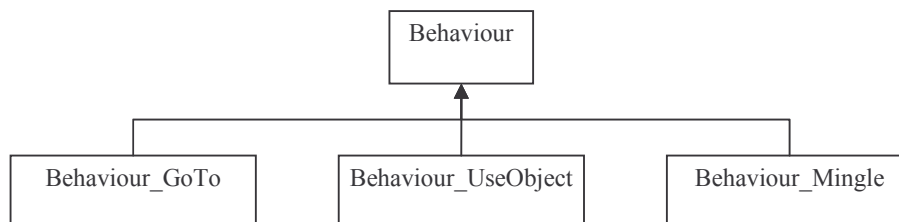


Figure D-7-9: A portion of the class hierarchy used to implement behaviours within the role passing unit of the PPA architecture implementation

As was mentioned previously, the role-passing unit also manages the way in which roles are adopted and discarded. When the schedule unit dictates that it is time for a new role to be adopted the role passing unit is informed. The first task which is then performed is that the TIME TO CHANGE ROLE persistent flag object is activated, so as any actions which must be performed before a role is discarded can be instigated. This is ensured by the NOT READY TO CHANGE ROLE concept node. As long as the activation of this node is above zero, the current role will not be discarded. However, eventually the activation of this node will reach zero and the PPAFCM objects associated with the current role will be removed. This

is easily achieved by the hash-table structure used within the FCM object, through which the PPAFCM elements associated with a particular role can be easily accessed.

Before adopting a new role a character must first move to the new location specified within the schedule entry which is being acted upon. Navigating to a new location is achieved by adopting a temporary role designed for this purpose. Once the character reaches the required location this temporary role is discarded and the new set of PPAFCM objects associated with the new role are instantiated and added to the character's PPAFCM.

D.3 Implementing the Social Unit

The social unit controls the social interactions which are performed by NPCs within a game world. The social unit is implemented through a class named SOCIALUNIT, the key attribute of which is a link to a single instance of an implementation of the μ -SIC system, which is shared amongst all of the NPCs within a game world. Implementing the μ -SIC system using a feed forward neural network has been discussed previously in section 4.5. The neural network itself was implemented in bespoke code (in a class named MLP) as part the implementation of the PPA architecture as a whole. The reason a bespoke version of this relatively common piece of software was used, was that the implementation was available from a previous project and a bespoke implementation gave the high level of control over the code which is necessary in order to satisfy the processing requirements of commercial games.

As well as implementing the μ -SIC system the social unit also manages all of a character's relationships with other characters or players within a game world. All of a character's relationships are stored within a list in the SOCIALUNIT class. Each relationship is represented by a RELATIONSHIP object which stores the name of the subject of the relationship and all of the values associated with the relationship model covered in section 4.5.1, describing the original agent's relationship to this character.

Social behaviours are instigated by the social behaviour object stored within a character's BRAIN object. This behaviour allows an NPC to constantly interact with the other characters in its locality. To ensure that the character does not attempt to interact with other characters at inappropriate times the social behaviour can be enabled and disabled by the role-passing system, as appropriate.

D.4 Implementing the Memory Unit

The memory unit of the PPA architecture is an implementation of the knowledge base discussed in section 4.6. The scheme described in that section is implemented (in a class named MEMORYUNIT) as a series of lists storing the events of which the character is aware. These events are separated into those that have not yet been processed into character sighting records and those that have, with separate lists being maintained for each category. The character sighting records which are created by means of processing events are stored in a separate list. Events, facts and character sighting records implemented as data structures which describe each type of object, through the classes EVENT, FACT and CHARACTER SIGHTING RECORD. The event fact and character sighting record lists can all be searched in order to allow characters answer questions about events they have witnessed, and to gossip with each other on the same subject.

In order to avoid the size of these lists becoming unwieldy, their size is limited. If a list becomes too big then the oldest entries which it contains are discarded to make way for new ones. This is not a sophisticated system and could benefit from a more elaborate scheme through which less important event records could be discarded, rather than simply older ones.

D.5 Ancillary Technologies

In order to allow the PPA architecture be used in games, a number of ancillary technologies are required to support the implementation of the core architecture. While these are not directly relevant to the implementation of the architecture itself, they are worthy of discussion as, without them it would not be possible to create fully functioning applications. Two of the most important ancillary technologies used by the PPA architecture are the path-finding system used to allow agents navigate a virtual world and the smart object inspired system which allows agents successfully interact with virtual objects that are located within game worlds.

Path Finding

Almost all games require that NPCs realistically navigate around a game world. This is generally referred to as path-finding and there are a number of algorithms widely used for the purpose. However, by far the most commonly used is the A* algorithm (described in detail in [Stout, 1996]). In order to satisfy the path-finding needs of the PPA architecture, an implementation of the A* algorithm was used.

that that instances of that type could be searched across. The SEARCHABLE interface simply included methods to determine the position and path value of a searchable object. By using this interface the same A* code was used to search across the path cells in areas themselves, and to search across areas within a virtual world allowing agents navigate across much larger areas.

Smart Objects

Virtual worlds of the kinds in which the PPA architecture is used are generally filled with a large number of virtual objects with which agents should be capable of performing interactions. These can range from the mundane (for example doors and chairs), to the more exotic (for example gun turrets and code breaking machines). In order to allow agents perform believable interactions with a wide array of very different objects, without imposing a massive processing burden, the technique of *smart objects* [Kallmann & Thalmann, 1998] has been used. Smart objects are virtual objects that contain more information than just intrinsic object properties, such as position, mass and appearance [Levison, 1996]. This extra information is referred to as the *interaction features* of an object, and may vary from what part of a door should be grasped when it is opened, to the fact that an agent's thirst should decrease after using a drinking fountain.

The original smart object system has been extended to make objects central to interactions between characters [Peters et al, 2003], thus further promoting the ideas of situational intelligence. Within the smart object model used in this work, objects store information about the sequences of events which allow multiple users perform interactions based on the use of a single object. For example, a virtual concierge desk in a restaurant environment should indicate the sequence of interactions through which a customer requests a table and has one allocated to them. The smart object should also govern the transfer of information involved in such an interaction.

The first component involved in the implementation of the smart object model is a *user slot*. A user slot indicates a position at which an agent can use an object, for a particular purpose. A smart object might have any number of user slots associated with it, and these can be of any number of types. Before an agent can begin to use a smart object they must first obtain ownership of a free user slot of an appropriate type.

Figure D-13 shows an illustration of a smart bar object. Two user slots are shown: the BARMAN slot and the GENERAL slot. These slots show that in order for two characters to perform an interaction based on the bar object (i.e. for a customer to buy a drink from a

barman) the barman character should stand behind the bar facing the customer, who should stand at the other side of the bar facing the barman.

User-slots are a departure from the smart object model used in [Kallmann & Thalmann, 1998] and are used as they avoid many of the concurrency problems which arise through the use of rules alone (which is the way in which the original smart object model operates). This is particularly important if smart objects are to be central to agent interactions.

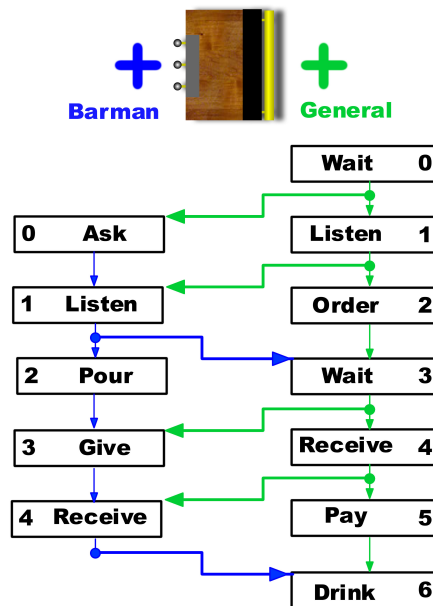


Figure D-7-13: An illustration of a smart bar object, showing its user slots and the sequence of events which characters must perform in using the object

In order to dictate the process by which an agent uses a smart object at a particular user slot, each slot contains a number of *usage steps* which describe, in a step by step manner, how an agent should proceed in using an object. Each usage step contains the following pieces of information:

- The information required to animate an agent at this step
- The conditions which must be met in order for an agent to move onto the next step.
- Details of any changes which are to be made to either an agent's attributes or the object's attributes on completion of this usage step
- Details of any information which should be passed to users of this object, or new objects which should be created on completion of this step
- Whether or not an agent is free to socially interact with other agents while at this step
- Points of interest on the object upon which an agent should focus while at this usage step

After the animation information, the most important aspect of a usage step is the condition which allows a character at that step to move on to the next usage step. Conditions can take one of two forms. The first indicates that the agent must only wait for the animations required by the current step to be complete in order to move onto the next step.

The second, and more interesting, form that a condition may take is that an agent at a particular slot must wait for an agent at another slot to reach a particular usage step in order to move onto the next step. It is in this way that smart objects are made central to character interactions.

Figure D-13 shows the usage steps involved in using a bar object at both the BARMAN and GENERAL user slots. The arrows show the conditions involved in moving from one usage step to the next. Usage steps with arrows going directly from that step to the next (for example going from step 0 (ASK) to step 1 (LISTEN) of the BARMAN user slot) only require that the animations required by the first step have been completed in order to move on to the next step.

If an arrow leads to the start of a step of another user slot from a point along an arrow moving from one step to the next, then a character must wait until the character at the other slot reaches the usage step indicated by this arrow before moving onto the next usage step. For example, if an agent at the GENERAL user slot is at step 1 (LISTEN), then they must wait until the agent at the BARMAN slot has reached step 1 (LISTEN), before they can move on to step 2 (ORDER).

Within the implementation of the PPA architecture smart objects were represented by instances of a SMARTOBJECT class. One compelling advantage of the use of the smart object system was that objects could be defined entirely in data files, without the need to write any new code, thus sitting comfortably with the notions of games as data. Objects were defined in XML files which were read by the game engine at run time and so instantiated new instance of the SMARTOBJECT class.

Appendix E: Implementation of Berlin-End-Game

In order to demonstrate the capabilities of the PPA architecture a character-centric adventure game was created and populated with a large cast of support characters, who performed a wide range of roles. The game, entitled Berlin-End-Game, was loosely based on the Bernard Samson novels by Len Deighton [1984], and put the player in the role of Bernard Samson, a British secret service agent on the trail of Stasi agent, Eric Stinnes in 1980s Berlin.

E.1 Implementing the Game World

The virtual world in which the game was set (a simple schematic of which is shown in figure E-14) was only loosely based on real-world Berlin, but some key locations from Deighton’s books, such as Checkpoint Charlie and Hotel Hennig, were included.

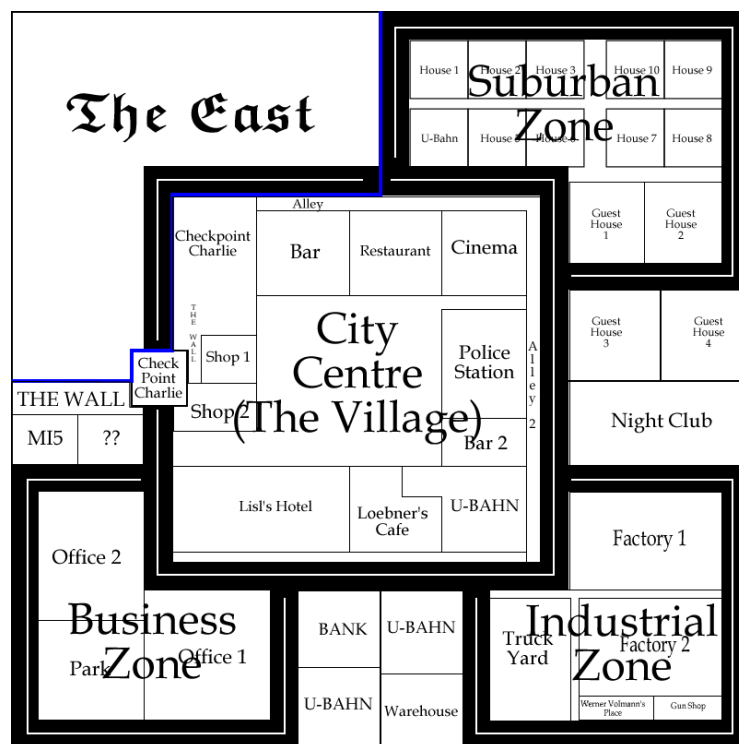


Figure E-7-14: A schematic of the virtual world in which Berlin-End-Game was set
 Within the implementation of the game, the virtual world was represented by an instance of a WORLD class. The key attributes of the WORLD class were lookup tables of area names to NPCs and smart objects, and an instance of the AREA class. The AREA class represented a particular area within the game world. The key attributes of this class were an area name, a set of image and path tiles defining the ground in an area and a list of *child areas* which a particular area contained. The use of child areas allowed the virtual world to be defined

hierarchically, which was advantageous when it came to searching across the world for objects or people. A portion of such a hierarchy is shown in figure E-15 which begins with the parent area Berlin and goes all the way down to the toilets in Loebner's bar.

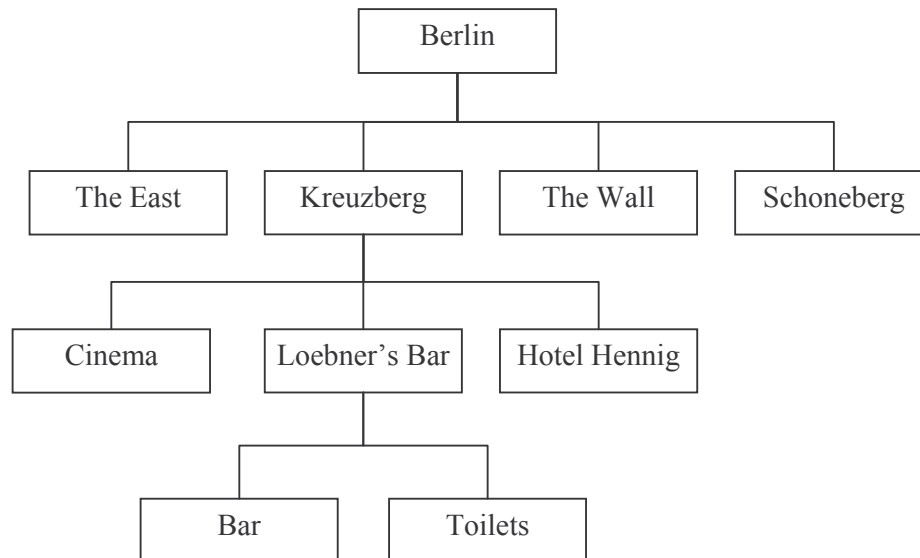


Figure E-7-15: A portion of the area hierarchy used within Berlin-End-Game. Parent areas contain all child areas

The hierarchical world structure was defined in an XML world file along with all of the information regarding the appearance of all areas, and the details of all smart objects. These XML files were pain staking to write by hand, and so a bespoke tool was developed to allow game designers create virtual worlds and have the XML files created automatically. An example of an XML world file is not given due to the amount of data involved - the world file used in Berlin-End-Game would run to over 1600 pages! The following section will briefly describe the world designer tool.

E.2 World Designer Tool

The purpose of the world designer tool is to allow game designers create worlds to be populated by PPA based characters. Worlds are based on a large set of simple *terrain tiles*. Terrain tiles are square cells about the size of a character which can have an image attached and be given a path value which is used by the path-finding system. When a world is first created a designer simply specifies the size of the world, which is created as one large area divided into terrain tiles. Game designers can divide this set of terrain tiles into areas, simply by selecting and combining groups of cells.

The tool also allows designers choose the images to be used on terrain tiles, set their path values and add smart objects to areas. All of this is done through simple menus, GUI

controls and lists which display the available tile images and smart objects. A screenshot of the tool is shown in figure E-16.

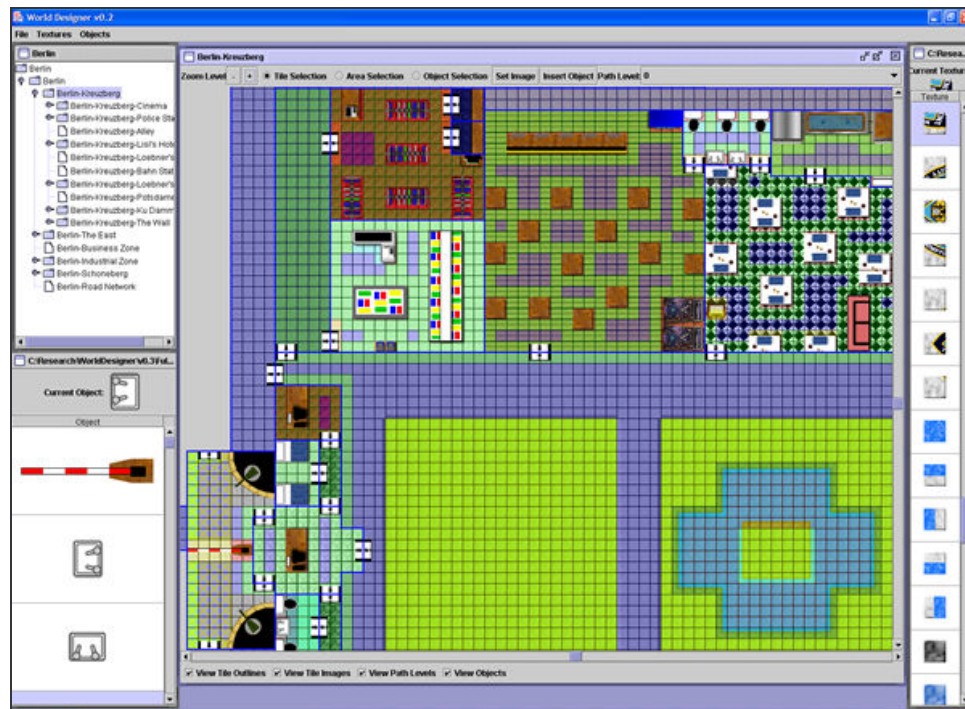


Figure E-16: A screenshot of the world designer tool created in order to aid the design of worlds used within the Berlin-End-Game application

E.3 The Player

Berlin-End-Game required the creation of a player character to act as the player's avatar within the game world. The player character was implemented within a class named `PLAYER` which was similar to the `CHARACTER` class mentioned previously. The only difference was that the `PLAYER` class, rather than containing an instance of the `BRAIN` class which implemented the PPA architecture, contained an instance of a more simple `PLAYERBRAIN` class. The `PLAYERBRAIN` class contained a single behaviour object which was updated at every world update cycle.

The player character is controlled using the mouse, allowing players to:

- Click on a location to instruct their avatar to move there
- Click on an object to instruct their avatar to use it
- Click on an NPC to instruct their avatar to interact with it

Whenever the player performed any of these actions, a new behaviour object was created and replaced the behaviour in the player avatar's brain.

Appendix F: Integration of the PPA Architecture into the ALOHA System

The *adaptive level of detail human animation* (ALOHA) system [Giang et al, 2000] was developed by members of the Image Synthesis Group at Trinity College, Dublin in order to apply level-of-detail techniques to the animation of virtual humans. The integration of the PPA architecture into ALOHA was performed during the development of both systems, which both aided and hindered the process. Characters within ALOHA are implemented through a `VIRTUALHUMAN` class. Previously, when the world was updated characters simply performed animation tasks according to a script. With the introduction of the PPA architecture the `BRAIN` class discussed previously was added to the ALOHA `VIRTUALHUMAN` class. Introducing this new class to the ALOHA system led to a number of interesting challenges, some of which will be discussed in the remainder of this section.

- **Waiting for animations:** Before its addition to the ALOHA system the PPA architecture had only been used in the Berlin-End-Game application. In this application all character actions were manifested as small action bubbles which appeared above a character's head for a given amount of time. The ALOHA system used sophisticated animations to display character actions and so this had to be taken into account. To achieve this, an abstract `ANIMATION` class was created which could be used to initialise any kind of animation and instruct an instance of the `BRAIN` class when an animation was complete. This `ANIMATION` class was subsequently used in both applications which used the PPA architecture.
- **Integrating the PPA architecture with the ALOHA world model:** Many of the behaviours which PPA based characters perform involve accessing the world object to determine what other characters and objects are nearby. The ALOHA system maintained a sophisticated scene database which was not at all similar to the way in which lists of objects were stored by Berlin-End-Game. Again a generalised interface was created which could be layered atop any world object allowing it to mimic the kind of access that the PPA `BRAIN` class expected to perform.
- **Matching processing requirements:** The processing power required to display an ALOHA simulation were far beyond those required to display Berlin-End-Game. As the PPA architecture was added to the ALOHA system its processing

requirements had to be brought into line with what was available. This resulted in the implementation of the PPA architecture becoming much more efficient.

The integration of the PPA architecture into the ALOHA system resulted in massive improvements to both systems, making it an enormously worthwhile undertaking.

Appendix G: A Sample XML Character Listing for a Character in Berlin-End-Game

The following shows a sample cast file from the Berlin-End-Game application. The cast in this case includes just a single character named Isaac. The listing first shows details of the character's name, location within the game world, personality, initial mood and finally appearance including the image used to represent the character.

After the initial details the details of the character's basic PPAFCM is given. This includes character motivations such as fear and hunger, concepts for each of these, appropriate actions for these motivations and links between all of the different objects.

After the basic PPAFCM the character's schedule is given. In this case the character spends some time in a bar, relaxes and sleeps at home and works as a chef in a restaurant. For each of the schedule entries the details of a PPAFCM are given for that particular role.

Finally, the details of facts of which the character is aware and the character's initial relationships are given. In this case the character knows about where he works and lives and has a positive relationship with a character named Bernard Samson.

```

<cast>
  <character>
    <name>Isaac</name>
    <location>
      <x>2700</x>
      <y>1650</y>
      <rotation>10</rotation>
      <area>Berlin-Kreuzberg-Loebner's-Restaurant-Dining Area</area>
    </location>
    <personality>
      <neuroticism>0.7</neuroticism>
      <extroversion>0.85</extroversion>
    </personality>
    <mood>
      <arousal>0.5</arousal>
      <valance>0.5</valance>
    </mood>
    <appearance>
      <gender>male</gender>
      <attractiveness>1</attractiveness>

```

```

    <imageFile>Graphics/People/Chef</imageFile>
  </appearance>
  <basicFCM>
    <name>basic</name>
    <motivations>
      <motivation>
        <name>fear</name>
        <autoIncrement>false</autoIncrement>
        <randomise>false</randomise>
        <responseCurve>
          <risingFileName>Motivations\fearMotivationCurveUp.mot</risingFileName>
          <fallingFileName>fearMotivationCurveDown.mot</fallingFileName>
        </responseCurve>
      </motivation>
      <motivation>
        <name>bladder</name>
        <autoIncrement>true</autoIncrement>
        <randomise>true</randomise>
        <responseCurve>
          <risingFileName>bladderMotivationCurveUp.mot</risingFileName>
          <fallingFileName>bladderMotivationCurveDown.mot</fallingFileName>
        </responseCurve>
      </motivation>
      <motivation>
        <name>fatigue</name>
        <autoIncrement>true</autoIncrement>
        <randomise>false</randomise>
        <responseCurve>
          <risingFileName>fatigueMotivationCurveUp.mot</risingFileName>
          <fallingFileName>fatigueMotivationCurveDown.mot</fallingFileName>
        </responseCurve>
      </motivation>
      <motivation>
        <name>hunger</name>
        <autoIncrement>true</autoIncrement>
        <randomise>true</randomise>
        <responseCurve>
          <risingFileName>hungerMotivationCurveUp.mot</risingFileName>
          <fallingFileName>hungerMotivationCurveDown.mot</fallingFileName>
        </responseCurve>
      </motivation>
    </motivations>
  </basicFCM>

```

```

<motivation>
  <name>socialise</name>
  <autoIncrement>true</autoIncrement>
  <randomise>true</randomise>
  <responseCurve>
    <risingFileName>socialiseMotivationCurveUp.mot</risingFileName>
    <fallingFileName>socialiseMotivationCurveDown.mot</fallingFileName>
  </responseCurve>
</motivation>
</motivations>
<flags>
  <flag>
    <name>timeToChangeRoles</name>
    <state>0</state>
  </flag>
</flags>
<concepts>
  <concept>
    <name>fear</name>
  </concept>
  <concept>
    <name>bladder</name>
  </concept>
  <concept>
    <name>fatigue</name>
  </concept>
  <concept>
    <name>hunger</name>
  </concept>
  <concept>
    <name>socialise</name>
  </concept>
  <concept>
    <name>notReadyToChangeRole</name>
  </concept>
</concepts>
<actions>
  <action>
    <name>flee</name>
    <actionName>flee</actionName>
  </action>

```

```
<action_satisfyNeed>
  <name>pee</name>
  <actionName>pee</actionName>
  <needName>bladder</needName>
  <selectionMode>0</selectionMode>
</action_satisfyNeed>
<action_namedObject>
  <name>goToBed</name>
  <actionName>goToBed</actionName>
  <objectName>guestHouse0Bed0</objectName>
  <slotType>gen</slotType>
</action_namedObject>
<action_satisfyNeed>
  <name>eat</name>
  <actionName>eat</actionName>
  <needName>hunger</needName>
  <selectionMode>0</selectionMode>
</action_satisfyNeed>
<action_generalMingle>
  <name>mingle</name>
  <actionName>mingle</actionName>
  <canMove>true</canMove>
</action_generalMingle>
</actions>
<links>
  <link>
    <weight>1.0</weight>
    <threshold>0</threshold>
    <originNode>basic_motivation_fear</originNode>
    <destinationNode>basic_concept_fear</destinationNode>
  </link>
  <link>
    <weight>1.0</weight>
    <threshold>0</threshold>
    <originNode>basic_concept_fear</originNode>
    <destinationNode>basic_action_flee</destinationNode>
  </link>
  <link>
    <weight>1.0</weight>
    <threshold>0</threshold>
    <originNode>basic_motivation_bladder</originNode>
```

```
<destinationNode>basic_concept_bladder</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0.7</threshold>
  <originNode>basic_concept_bladder</originNode>
  <destinationNode>basic_action_pee</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNode>basic_motivation_fatigue</originNode>
  <destinationNode>basic_concept_fatigue</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>1.0</threshold>
  <originNode>basic_concept_fatigue</originNode>
  <destinationNode>basic_action_goToBed</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNode>basic_motivation_hunger</originNode>
  <destinationNode>basic_concept_hunger</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0.6</threshold>
  <originNode>basic_concept_hunger</originNode>
  <destinationNode>basic_action_eat</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNode>basic_motivation_socialise</originNode>
  <destinationNode>basic_concept_socialise</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
```

```
<originNode>basic_concept_socialise</originNode>
  <destinationNode>basic_action_mingle</destinationNode>
</link>
<link>
  <weight>-0.5</weight>
  <threshold>0</threshold>
  <originNode>basic_concept_fear</originNode>
  <destinationNode>basic_concept_bladder</destinationNode>
</link>
<link>
  <weight>-0.5</weight>
  <threshold>0</threshold>
  <originNode>basic_concept_fear</originNode>
  <destinationNode>basic_concept_hunger</destinationNode>
</link>
<link>
  <weight>-0.5</weight>
  <threshold>0</threshold>
  <originNode>basic_concept_fear</originNode>
  <destinationNode>basic_concept_fatigue</destinationNode>
</link>
<link>
  <weight>-0.5</weight>
  <threshold>0</threshold>
  <originNode>basic_concept_fear</originNode>
  <destinationNode>basic_concept_socialise</destinationNode>
</link>
<link>
  <weight>-0.3</weight>
  <threshold>0</threshold>
  <originNode>basic_concept_fatigue</originNode>
  <destinationNode>basic_concept_bladder</destinationNode>
</link>
<link>
  <weight>-0.3</weight>
  <threshold>0</threshold>
  <originNode>basic_concept_fatigue</originNode>
  <destinationNode>basic_concept_hunger</destinationNode>
</link>
<link>
  <weight>-0.3</weight>
```

```

    <threshold>0</threshold>
    <originNode>basic_concept_fatigue</originNode>
    <destinationNode>basic_concept_socialise</destinationNode>
</link>
<link>
    <weight>-0.2</weight>
    <threshold>0</threshold>
    <originNode>basic_concept_bladder</originNode>
    <destinationNode>basic_concept_socialise</destinationNode>
</link>
<link>
    <weight>-0.2</weight>
    <threshold>0</threshold>
    <originNode>basic_concept_hunger</originNode>
    <destinationNode>basic_concept_socialise</destinationNode>
</link>
</links>
</basicFCM>
<schedule>
  <scheduleEntry>
    <startTime>1</startTime>
    <endTime>20160</endTime>
    <location>Berlin-Kreuzberg-Loebner's-Bar</location>
    <role>
      <name>barPatron</name>
      <motivations>
        <motivation>
          <name>thirst</name>
          <responseCurve>
            <risingFileName>thirstMotivationCurveUp.mot</risingFileName>
            <fallingFileName>thirstDownward.mot</fallingFileName>
          </responseCurve>
        </motivation>
        <motivation>
          <name>sit</name>
          <responseCurve>
            <risingFileName>relaxMotivationCurveUp.mot</risingFileName>
            <fallingFileName>relaxMotivationCurveDown.mot</fallingFileName>
          </responseCurve>
        </motivation>
      </motivations>
    </role>
  </scheduleEntry>
</schedule>

```

```

    <name>entertainment</name>
    <responseCurve>
      <risingFileName>entertainmentMotivationCurveUp.mot</risingFileName>
      <fallingFileName>entertainmentMottionCurveDown.mot</fallingFileName>
    </responseCurve>
  </motivation>
</motivations>
<flags>
  <flag>
    <name>sitting</name>
    <state>0</state>
  </flag>
</flags>
<concepts>
  <concept>
    <name>thirst</name>
  </concept>
  <concept>
    <name>sit</name>
  </concept>
  <concept>
    <name>entertainment</name>
  </concept>
</concepts>
<actions>
  <action_typeOfObject>
    <name>getADrink</name>
    <actionName>useTypeOfObject</actionName>
    <objectType>bar</objectType>
    <slotType>gen</slotType>
    <selectionMode>0</selectionMode>
  </action_typeOfObject>
  <action_typeOfObject>
    <name>sitDown</name>
    <actionName>useTypeOfObject</actionName>
    <objectType>barTable</objectType>
    <slotType>gen</slotType>
    <selectionMode>1</selectionMode>
    <relatedFlags>
      <actionCompletionFlag>
        <flagName>barPatron_flag_sitting</flagName>

```

```

        <onOrOff>1</onOrOff>
        <includeObjectRef>true</includeObjectRef>
    </actionCompletionFlag>
</relatedFlags>
</action_typeOfObject>
<action_satisfyNeed>
    <name>entertainMe</name>
    <actionName>satisfyNeed</actionName>
    <needName>entertainment</needName>
    <selectionMode>0</selectionMode>
</action_satisfyNeed>
</actions>
<links>
    <link>
        <weight>1.0</weight>
        <threshold>0</threshold>
        <originNode>barPatron_motivation_thirst</originNode>
        <destinationNode>barPatron_concept_thirst</destinationNode>
    </link>
    <link>
        <weight>1.0</weight>
        <threshold>0.9</threshold>
        <originNode>barPatron_concept_thirst</originNode>
        <destinationNode>barPatron_action_getADrink</destinationNode>
    </link>
    <link>
        <weight>-1.0</weight>
        <threshold>0</threshold>
        <originNode>barPatron_concept_thirst</originNode>
        <destinationNode>barPatron_concept_sit</destinationNode>
    </link>
    <link>
        <weight>-0.5</weight>
        <threshold>0</threshold>
        <originNode>barPatron_concept_thirst</originNode>
        <destinationNode>barPatron_concept_entertainment</destinationNode>
    </link>
    <link>
        <weight>1.0</weight>
        <threshold>0</threshold>
        <originNode>barPatron_motivation_entertainment</originNode>

```

```

        <destinationNode>barPatron_concept_entertainment</destinationNode>
    </link>
    <link>
        <weight>1.0</weight>
        <threshold>0</threshold>
        <originNode>barPatron_concept_entertainment</originNode>
        <destinationNode>barPatron_action_entertainMe</destinationNode>
    </link>
    <link>
        <weight>-1.0</weight>
        <threshold>0</threshold>
        <originNode>barPatron_concept_entertainment</originNode>
        <destinationNode>barPatron_concept_sit</destinationNode>
    </link>
    <link>
        <weight>1.0</weight>
        <threshold>0</threshold>
        <originNode>barPatron_motivation_sit</originNode>
        <destinationNode>barPatron_concept_sit</destinationNode>
    </link>
    <link>
        <weight>-1.0</weight>
        <threshold>0</threshold>
        <originNode>barPatron_flag_sitting</originNode>
        <destinationNode>barPatron_action_sitDown</destinationNode>
    </link>
    <link>
        <weight>1.0</weight>
        <threshold>0</threshold>
        <originNode>barPatron_concept_sit</originNode>
        <destinationNode>barPatron_action_sitDown</destinationNode>
    </link>
</links>
<thresholdChanges>
    <thresholdChange>
        <linkName>basic_link_basic_motivation_fatigue_TO_basic_concept_fatigue</linkName>
        <threshold>1.1</threshold>
    </thresholdChange>
</thresholdChanges>
</role>
</scheduleEntry>

```

```
<scheduleEntry>
  <startTime>20161</startTime>
  <endTime>80640</endTime>
  <location>Berlin-Schoneberg-Guest House 0-Living Room</location>
  <role>
    <name>atHome</name>
    <motivations>
      <motivation>
        <name>sitDown</name>
        <responseCurve>
          <risingFileName>sitMotivationCurveUp.mot</risingFileName>
          <fallingFileName>sitMotivationCurveDown.mot</fallingFileName>
        </responseCurve>
      </motivation>
    </motivations>
    <concepts>
      <concept>
        <name>sitDown</name>
      </concept>
    </concepts>
    <flags>
      <flag>
        <name>sitting</name>
        <state>0</state>
      </flag>
    </flags>
    <actions>
      <action_typeOfObject>
        <name>sitDown</name>
        <actionName>sitDown</actionName>
        <objectType>couch</objectType>
        <slotType>gen</slotType>
        <selectionMode>1</selectionMode>
        <relatedFlags>
          <actionCompletionFlag>
            <flagName>atHome_flag_sitting</flagName>
            <onOrOff>1</onOrOff>
            <includeObjectRef>true</includeObjectRef>
          </actionCompletionFlag>
        </relatedFlags>
      </action_typeOfObject>
```

```
</actions>
<links>
  <link>
    <weight>1.0</weight>
    <threshold>0</threshold>
    <originNode>atHome_motivation_sitDown</originNode>
    <destinationNode>atHome_concept_sitDown</destinationNode>
  </link>
  <link>
    <weight>-1.0</weight>
    <threshold>0</threshold>
    <originNode>atHome_flag_sitting</originNode>
    <destinationNode>atHome_action_sitDown</destinationNode>
  </link>
  <link>
    <weight>1.0</weight>
    <threshold>0</threshold>
    <originNode>atHome_concept_sitDown</originNode>
    <destinationNode>atHome_action_sitDown</destinationNode>
  </link>
</links>
</role>
</scheduleEntry>
<scheduleEntry>
  <startTime>80641</startTime>
  <endTime>161280</endTime>
  <location>Berlin-Kreuzberg-Loebner's-Restaurant-Kitchen</location>
  <role>
    <name>chef</name>
    <motivations>
      <motivation>
        <name>relax</name>
        <responseCurve>
          <risingFileName>relaxMotivationCurveUp.mot</risingFileName>
          <fallingFileName>relaxMotivationCurveDown.mot</fallingFileName>
        </responseCurve>
      </motivation>
    </motivations>
    <concepts>
      <concept>
        <name>relax</name>
```

```

</concept>
<concept>
  <name>dropOffDinners</name>
</concept>
<concept>
  <name>cook</name>
</concept>
<concept>
  <name>getIngredients</name>
</concept>
<concept>
  <name>getOrders</name>
</concept>
<concept>
  <name>notInKitchen</name>
</concept>
</concepts>
<rules>
  <rule_hasObjectType>
    <name>hasIngredients</name>
    <objectType>ingredients</objectType>
  </rule_hasObjectType>
  <rule_hasObjectType>
    <name>hasOrders</name>
    <objectType>order</objectType>
  </rule_hasObjectType>
  <rule_hasObjectType>
    <name>hasDinners</name>
    <objectType>dinner</objectType>
  </rule_hasObjectType>
  <rule_objectInGivenState>
    <name>newOrdersPresent</name>
    <conditions>
      <ruleAndChain>
        <rule_objectOfType>
          <objectType>order</objectType>
        </rule_objectOfType>
        <rule_objectFree />
      </ruleAndChain>
    </conditions>
  </rule_objectInGivenState>

```

```

<rule_characterNotInArea>
  <name>notInKitchen</name>
  <areaName>Berlin-Kreuzberg-Loebner's-Restaurant-Kitchen</areaName>
</rule_characterNotInArea>
</rules>
<flags>
  <flag>
    <name>cooking</name>
    <state>0</state>
  </flag>
</flags>
<actions>
  <action>
    <name>chill</name>
    <actionName>idle</actionName>
  </action>
  <action_pickUpTypeOfObject>
    <name>pickUpOrders</name>
    <actionName>pickUpOrders</actionName>
    <objectType>order</objectType>
    <selectionMode>0</selectionMode>
  </action_pickUpTypeOfObject>
  <action_typeOfObject>
    <name>getIngredients</name>
    <actionName>useFridge</actionName>
    <objectType>fridge</objectType>
    <slotType>chef</slotType>
    <selectionMode>0</selectionMode>
  </action_typeOfObject>
  <action_typeOfObject>
    <name>dropOffDinners</name>
    <actionName>useOrderBar</actionName>
    <objectType>orderBar</objectType>
    <slotType>chef</slotType>
    <selectionMode>0</selectionMode>
  </action_typeOfObject>
  <action_typeOfObject>
    <name>cook</name>
    <actionName>useCooker</actionName>
    <relatedFlags>
      <actionCreationFlag>

```

```

        <flagName>chef_flag_cooking</flagName>
        <onOrOff>1</onOrOff>
        <includeObjectRef>>false</includeObjectRef>
    </actionCreationFlag>
    <actionCompletionFlag>
        <flagName>chef_flag_cooking</flagName>
        <onOrOff>0</onOrOff>
        <includeObjectRef>>false</includeObjectRef>
    </actionCompletionFlag>
</relatedFlags>
<objectType>cooker</objectType>
<slotType>chef</slotType>
<selectionMode>0</selectionMode>
</action_typeOfObject>
<action_goToArea>
    <name>goToKitchen</name>
    <actionName>goToKitchen</actionName>
    <areaName>Berlin-Kreuzberg-Loebner's-Restaurant-Kitchen</areaName>
</action_goToArea>
</actions>
<links>
    <link>
        <weight>1.0</weight>
        <threshold>0</threshold>
        <originNode>chef_rule_hasDinners</originNode>
        <destinationNode>chef_concept_dropOffDinners</destinationNode>
    </link>
    <link>
        <weight>1.0</weight>
        <threshold>0</threshold>
        <originNode>chef_concept_dropOffDinners</originNode>
        <destinationNode>chef_action_dropOffDinners</destinationNode>
    </link>
    <link>
        <weight>1.0</weight>
        <threshold>0</threshold>
        <originNode>chef_rule_newOrdersPresent</originNode>
        <destinationNode>chef_concept_getOrders</destinationNode>
    </link>
    <link>
        <weight>1.0</weight>

```

```

<threshold>0</threshold>
<originNode>chef_concept_getOrders</originNode>
<destinationNode>chef_action_pickUpOrders</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_rule_hasIngredients</originNode>
  <destinationNode>chef_concept_cook</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_flag_cooking</originNode>
  <destinationNode>chef_concept_cook</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_concept_cook</originNode>
  <destinationNode>chef_action_cook</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_rule_hasOrders</originNode>
  <destinationNode>chef_concept_getIngredients</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_concept_getIngredients</originNode>
  <destinationNode>chef_action_getIngredients</destinationNode>
</link>
<link>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_rule_notInKitchen</originNode>
  <destinationNode>chef_concept_notInKitchen</destinationNode>
</link>
<link>

```

```
<weight>1.0</weight>
<threshold>0</threshold>
<originNode>chef_concept_notInKitchen</originNode>
<destinationNode>chef_action_goToKitchen</destinationNode>
</link>
<link>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_concept_dropOffDinners</originNode>
  <destinationNode>chef_concept_getOrders</destinationNode>
</link>
<link>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_concept_dropOffDinners</originNode>
  <destinationNode>chef_concept_cook</destinationNode>
</link>
<link>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_concept_dropOffDinners</originNode>
  <destinationNode>chef_concept_getIngredients</destinationNode>
</link>
<link>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_concept_cook</originNode>
  <destinationNode>chef_concept_getOrders</destinationNode>
</link>
<link>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_concept_cook</originNode>
  <destinationNode>chef_concept_getIngredients</destinationNode>
</link>
<link>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNode>chef_concept_getIngredients</originNode>
  <destinationNode>chef_concept_getOrders</destinationNode>
</link>
```

```

<namedLink>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_getIngredients</originNodeName>
  <destinationNodeName>basic_concept_socialise</destinationNodeName>
</namedLink>
<namedLink>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_dropOffDinners</originNodeName>
  <destinationNodeName>basic_concept_socialise</destinationNodeName>
</namedLink>
<namedLink>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_cook</originNodeName>
  <destinationNodeName>basic_concept_socialise</destinationNodeName>
</namedLink>
<namedLink>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_getIngredients</originNodeName>
  <destinationNodeName>basic_concept_socialise</destinationNodeName>
</namedLink>
<namedLink>
  <weight>-1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_getOrders</originNodeName>
  <destinationNodeName>basic_concept_socialise</destinationNodeName>
</namedLink>
<namedLink>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_dropOffDinners</originNodeName>
  <destinationNodeName>basic_concept_notReadyToChangeRole</destinationNodeName>
</namedLink>
<namedLink>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_cook</originNodeName>
  <destinationNodeName>basic_concept_notReadyToChangeRole</destinationNodeName>

```

```

</namedLink>
<namedLink>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_getIngredients</originNodeName>
  <destinationNodeName>basic_concept_notReadyToChangeRole</destinationNodeName>
</namedLink>
<namedLink>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_getOrders</originNodeName>
  <destinationNodeName>basic_concept_notReadyToChangeRole</destinationNodeName>
</namedLink>
<namedLink>
  <weight>1.0</weight>
  <threshold>0</threshold>
  <originNodeName>chef_concept_notInKitchen</originNodeName>
  <destinationNodeName>basic_concept_notReadyToChangeRole</destinationNodeName>
</namedLink>
</links>
<thresholdChanges>
  <thresholdChange>
    <linkName>basic_link_basic_motivation_fatigue_TO_basic_concept_fatigue</linkName>
    <threshold>1.1</threshold>
  </thresholdChange>
</thresholdChanges>
</role>
</scheduleEntry>
</schedule>
<facts>
  <fact>
    <subject>Isaac</subject>
    <text>Isaac works in Loebner's Restaurant</text>
  </fact>
  <fact>
    <subject>Isaac</subject>
    <text>Isaac lives in Berlin-Schoneberg-Guest House 0</text>
  </fact>
</facts>
<relationships>
  <relationship>

```

```
<name>Bernard Samson</name>
<likeDislike>0.9</likeDislike>
<attractedness>0</attractedness>
<dominance>0.5</dominance>
<intimacy>0.9</intimacy>
<interest>0.9</interest>
</relationship>
</relationships>
</character>
</cast>
```
