This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

- This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated.
- A copy can be downloaded for personal non-commercial research or study, without prior permission or charge.
- This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author.
- The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.
- When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.
A REACTIVE ARCHITECTURE FOR AUTONOMOUS VIRTUAL AGENTS USING FUZZY LOGIC

Jafreezal Jaafar

Doctor of Philosophy
Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh
2009
Abstract

One of the fundamental aspects of a virtual environment is the virtual agents that inhabit them. In many applications, virtual agents are required to perceive input information from their environment and make decisions appropriate to their task based on their programmed reaction to those inputs. The research presented in this thesis focuses on the reactive behaviour of the agents. We propose a new control architecture to allow agents to behave autonomously in navigation tasks in unknown environments. Our behaviour-based architecture uses fuzzy logic to solve problems of agent control and action selection and which can coordinate conflicts among different operations of reactive behaviours. A Fuzzy Associative Memory (FAM) is used as the process of encoding and mapping the input fuzzy sets to the output fuzzy set and to optimise the fuzzy rules. Our action selection algorithm is based on the fuzzy \( \alpha \)-level method with the Hurwicz criterion. The main objective of the thesis was to implement agent navigation from point to point by a coordination of planning, sensing and control. However, we believe that the reactive architecture emerging from this research is sufficiently general that it could be applied to many applications in widely differing domains where real-time decision making under uncertainty is required. To illustrate this generality, we show how the architecture is applied to a different domain. We chose the example of a computer game since it clearly demonstrates the attributes of our architecture: real-time action selection and handling uncertainty. Experimental results are presented for both implementations which show how the fuzzy method is applied, its generality and that it is robust enough to handle different uncertainties in different environments. In summary, the proposed reactive architecture is shown to solve aspects of behaviour control for autonomous virtual agents in virtual environments and can be applied to various application domains.
Acknowledgements

Many people contributed in some way or other to the work presented in this thesis. The first acknowledgement must go to my supervisor Eric Mckenzie for his always kind support during my study. I am indebted to my second supervisor Alan Smaill for his valuable suggestions and comments on the thesis. Without his help, the thesis would not be as it is. I also would like to thank all IPAB staff and friends who have helped me in many ways on both studying and living in the UK.
Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Jafreezal Jaafar)
Dedication

To my wife and son.
Contents

Abstract i
Acknowledgements ii
Declaration iii
Dedication iv

1 Introduction 1
1.1 Introduction ........................................ 1
1.2 Aim .................................................... 3
1.3 Problem Description ............................. 4
1.4 The method ........................................... 6
1.5 Why Fuzzy Logic ................................ 7
1.6 Thesis Organization ............................. 8
1.7 Publications ......................................... 9

2 Literature Review 11
2.1 Introduction ........................................ 11
2.2 Virtual Environments ............................ 12
2.3 Autonomous Agent ............................... 13
  2.3.1 What is an Autonomous Agent? ............ 14
    2.3.1.1 Autonomy .................................. 15
    2.3.1.2 Autonomous Behaviour .................. 17
  2.3.2 Types of Agents ............................. 18
2.4 Control Architectures ........................... 19
  2.4.1 Deliberative Architecture .................. 21
  2.4.2 Reactive Architecture ....................... 22
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.3</td>
<td>Hybrid Architecture</td>
<td>24</td>
</tr>
<tr>
<td>2.5</td>
<td>Action Selection Mechanism</td>
<td>26</td>
</tr>
<tr>
<td>2.5.1</td>
<td>The Action Selection Problem</td>
<td>27</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Classification of Action Selection</td>
<td>29</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Reactive Action Selection Method</td>
<td>31</td>
</tr>
<tr>
<td>2.5.3.1</td>
<td>Arbitration</td>
<td>32</td>
</tr>
<tr>
<td>2.5.3.2</td>
<td>Command Fusion</td>
<td>36</td>
</tr>
<tr>
<td>2.6</td>
<td>Summary</td>
<td>41</td>
</tr>
<tr>
<td>3</td>
<td>The Methodology</td>
<td>42</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>42</td>
</tr>
<tr>
<td>3.2</td>
<td>Overview of Fuzzy Logic Approach</td>
<td>43</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Basics of fuzzy sets</td>
<td>43</td>
</tr>
<tr>
<td>3.2.1.1</td>
<td>Fuzzy sets</td>
<td>43</td>
</tr>
<tr>
<td>3.2.1.2</td>
<td>Elementary operators for fuzzy sets</td>
<td>44</td>
</tr>
<tr>
<td>3.2.1.3</td>
<td>Fuzzy relations</td>
<td>45</td>
</tr>
<tr>
<td>3.2.1.4</td>
<td>Fuzzy composition</td>
<td>45</td>
</tr>
<tr>
<td>3.2.1.5</td>
<td>Fuzzy Implication</td>
<td>46</td>
</tr>
<tr>
<td>3.2.1.6</td>
<td>Membership functions</td>
<td>46</td>
</tr>
<tr>
<td>3.2.1.7</td>
<td>Rule Base</td>
<td>47</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Fuzzy Systems</td>
<td>49</td>
</tr>
<tr>
<td>3.3</td>
<td>Modeling of the Control System</td>
<td>50</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Behaviour Conflicts and α—level Thresholds</td>
<td>51</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Rules Evaluation</td>
<td>52</td>
</tr>
<tr>
<td>3.4</td>
<td>Action Selection Method</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>Summary</td>
<td>58</td>
</tr>
<tr>
<td>4</td>
<td>Virtual Agent Navigation</td>
<td>60</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>60</td>
</tr>
<tr>
<td>4.2</td>
<td>Related Work</td>
<td>61</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Fuzzy Logic</td>
<td>65</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Behaviour-based Architecture</td>
<td>66</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Local Minima Problem</td>
<td>68</td>
</tr>
<tr>
<td>4.3</td>
<td>Behaviour Design</td>
<td>69</td>
</tr>
<tr>
<td>4.4</td>
<td>Navigation System</td>
<td>71</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Visual Sensor</td>
<td>73</td>
</tr>
</tbody>
</table>
CONTENTS

4.4.2 Virtual Motion ............................................. 75
4.5 The Fuzzy Controller ..................................... 76
  4.5.1 Fuzzy Associative Memory .............................. 76
    4.5.1.1 Establishment of FAM ............................... 77
    4.5.1.2 Generation of behaviour rules using FAM .......... 77
  4.5.2 Path-Planning Behaviour .............................. 78
    4.5.2.1 Turn Rules and Weight Rules ...................... 79
    4.5.2.2 Local Minima Algorithm ........................... 80
  4.5.3 Goal-Seeking Behaviour ............................... 82
  4.5.4 Obstacle Avoidance Behaviour ....................... 82
4.6 Experiments ............................................... 85
  4.6.1 Moving Towards the Goal ............................. 88
  4.6.2 Escape from Local Minima Problem ................... 91
  4.6.3 Navigating in a Complex Environment ............... 94
  4.6.4 Action Selection Method ............................. 98
  4.6.5 Performance Comparison .............................. 102
    4.6.5.1 Comparison Between Different Scenarios .......... 102
    4.6.5.2 Comparison with Other Fuzzy Methods ............ 106
4.7 Summary .................................................. 106

5 Virtual Agents in Computer Games .......................... 109
  5.1 Introduction ............................................. 109
  5.2 Background .............................................. 110
  5.3 Pacman .................................................... 112
    5.3.1 Self-playing Pacman .................................. 113
    5.3.2 Intelligent Control of The Ghost Behaviour ....... 115
  5.4 The Framework .......................................... 115
    5.4.1 Control Thread ....................................... 116
    5.4.2 Fuzzy Behaviour and Fuzzy Variables ............... 118
    5.4.3 Behaviour Selection .................................. 120
  5.5 Implementation .......................................... 120
    5.5.1 Performance Evaluation .............................. 122
    5.5.2 User Evaluation ...................................... 123
  5.6 Discussion .............................................. 124
  5.7 Summary .................................................. 125
List of Figures

2.1 Autonomous Agent Taxonomy [Franklin 97] .......................... 13
2.2 Virtual Agent Element .................................................. 14
2.3 Example of Framework for Virtual Agents [Iglesias 04] ........... 18
2.4 Three Types of Agent Control Architecture [Brooks 86, Pérez 00] . 20
2.5 Deliberative Architecture [Pokahr 03] ................................. 22
2.6 Reactive Architecture [Weyns 04] ....................................... 23
2.7 Hybrid Architecture [Franklin 97] ...................................... 25
2.8 Internal View of Agencies [Pirjanian 99a] .............................. 26
2.9 The Action Selection Module Inside the Animal Brain .............. 28
2.10 Classification of Action Selection (a) [MacKenzie 97] (b) [Saffiotti 97] 29
2.11 Agent AI Subsumption Architecture. An I node indicates inhibition; an S node indicates suppression. ................................. 34
2.12 Transition Relationship of the Finite State Agent .................. 35
2.13 An Example of a Behaviour Network ................................. 35
2.14 Goal-Action-Attribute Model .......................................... 37
2.15 Fuzzy Rule Based Controller [Vosinakis 07] ......................... 38
2.16 Multi-objective Behaviour Coordination [Kubota 07] ................ 40

3.1 Membership Functions of a Crisp Set \( C \) and a Fuzzy Set \( \mathcal{F} \) ........ 44
3.2 Membership Grades of \( x_0 \) in the Sets \( \mathcal{A} \) and \( \mathcal{B} \) ............ 46
3.3 Rule-based Fuzzy System with \( n \) Inputs and One Output .......... 47
3.4 Mamdani Implication with Crisp Inputs ............................... 48
3.5 Basic Fuzzy Controller ................................................. 49
3.6 Fuzzy Input Corresponding to \( x_0 \) and \( y_0 \) ......................... 53
3.7 Trapezoidal Fuzzy Numbers ............................................. 54
3.8 Trapezoidal Fuzzy Number ............................................. 56

4.1 An Example of Path Traversing Through All User-Specified Locations. 62
4.36 Example of Navigation Path in Cluttered Environment (a) Fuzzy-ASM (b) FRM (c) FPF ........................................ 107
4.37 Path Length Produced by FPF, FRM and the Proposed Fuzzy Method ......................................................... 107

5.1 Game Overall Framework ........................................ 116
5.2 Membership Functions for Distance. .......................... 119
5.3 Membership Functions for Pellet Time .......................... 119
5.4 Membership Functions for Average Lifetime ................. 120
5.5 Membership Functions for Pellet Consumption Rate ........ 120
5.6 Pacman Screen Design. ............................................ 121
5.7 Example of Ghost with (a) Random Movement (b) Hunting Pacman ......................................................... 121
5.8 Example of Pacman Trapped by the Ghosts. .................. 122
5.9 CPU Utilization. ...................................................... 122
5.10 Player Ratings of Fuzzy vs. Original Pacman. ............... 123
## List of Tables

2.1 Characteristics of Different Virtual Agent Controls [Ferreira 02] ........ 16  
2.2 Reactive Architectures and their Basic Characteristics [Arkin 98, Pérez 00] 24  
2.3 Examples of Action Selection Methods. .............................. 31  
2.4 Virtual Agent Action Selection Methods [Delgado 04] ............... 33  

3.1 Rule Table: *IF – THEN* rules ........................................... 52  
3.2 Decision Table with an Active Cell ................................. 53  

4.1 State Table ................................................................. 71  
4.2 Linguistic Input Fuzzy Set and Their Ranges .......................... 77  
4.3 Weight Rules for Path-Planning Behaviour ........................... 80  
4.4 Statistical Result for Navigation with One Obstacle ............... 89  
4.5 Comparison of Defuzzification Technique ............................ 92  
4.6 Navigating Two Walls ................................................... 92  
4.7 Navigating Through Narrow Passage ................................ 92  
4.8 Navigating Through Corner ............................................. 92  
4.9 The Performance of Fuzzy-ASM vs. FBF. ............................. 101  
4.10 Performance Comparisons .............................................. 104
Chapter 1

Introduction

The insight at the root of artificial intelligence was that these "bits" (manipulated by computers) could just as well stand as symbols for concepts that the machine would combine by the strict rules of logic or the looser associations of psychology.

- Daniel Crevier (1947-1993)

The Tumultuous History of the Search for Artificial Intelligence

1.1 Introduction

The Intelligent Virtual Environment (IVE) is concerned with the development of new technologies that emerged from the intersection between Virtual Environments (VEs) and Artificial Intelligence (AI). One of the main factors is the continuing growth in the amount of computing power that can be put on a desktop. The desktop not only supports a much higher degree of visual realism, but even leaves a little additional processing power that can be used to add intelligence [Aylett 00].

A virtual environment is a simulated environment that appears to have the characteristics of some other environment, and in which participants perceive themselves as interactive parts [ATIS 00]. AI is a branch of computer science dealing with the simulation of intelligent behaviour in computers, where the system has the capability to imitate intelligent human behaviour [Merriam Webster 56]. IVEs consider the use of AI techniques as a component which can be used to enhance the interactivity of a virtual environment.

One of the fundamental aspects of a virtual environment is the virtual agents that inhabit them. A virtual agent can be defined as an autonomous entity in a virtual environment. An autonomous virtual agent is situated within, and as 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 influence what it senses in the future [Franklin 97]. A virtual agent should not only look like, but also behave as a living organism in a synthetic 3D world, and be able to interact with the world and its inhabitants [Vosinakis 01].

Another important aspect of a virtual agent is to make it look real. One of the ways is to endow them with a strong personality, emphasizing differences among virtual agents. If a virtual agent has a visible personality, a user will be more willing to believe in them and overlook defects in their behaviour. Additional to that the virtual agent must combine aspects of an autonomous robot with some of the skills of a human actor in an improvisational theatre [Reynolds 99]. For example, virtual agent/characters used in computer games or in 3D animated films, are designed as autonomous agents and then complex behaviours can be produced with minimal intervention from the animator.

There are two main types of virtual agent in virtual environments which are representations of the user in the virtual environment (also known as avatar), and are computer controlled virtual agents with which the user can interact [Gillies 01]. Avatars in virtual environments are controlled by a user of the environment. They are the personification of the user in the environment and must perform the actions that the user wants to perform. Computer controlled virtual agents are entirely autonomous, their behaviour is entirely controlled by the computer. This means that they need behaviour animation as all the behaviour must be simulated.

In a virtual environment, objects and avatars are connected to a virtual agent, which reflect a behaviour with other virtual agents and users [Noll 99]. The virtual agent acts autonomously and improves interaction between objects. The virtual agent should be proactive in the user's interest [Ralph 97]. It fulfills its tasks based on internal states, rules, and goals, and does not need any guidance by a human.

When designing any virtual agent based system, it is important to determine how sophisticated the virtual agent's reasoning will be [Remondino 08]. This is because virtual agents can play different types of roles, accomplish different tasks and responsibilities. Depending on their role definitions, different virtual agents tend to differ in their autonomy, cooperation ability or intelligence. For example, a virtual agent that supplies decision support functionality acts autonomously and proactively to gather information, and makes recommendations. The ultimate decision will, however, be made by a human decision-maker. In contrast, a virtual agent may also assume a completely autonomous role. That is, the virtual agent is entirely responsible for the whole
process of problem solving. Not all virtual agents can exhibit smart problem solving behaviour; some do, and they are limited by the current state of the art in related fields. In some cases the individual virtual agents of a system may not be that intelligent at all, but in combination and cooperation they lead to the intelligence and smartness of an agent-system [Hermans 96].

As most researchers developed increasingly interesting, larger and more complex virtual environments, the ability of virtual agents to consciously find their way around the environment plays a more important role in their behaviour [Champandard 02]. This can be seen in various virtual agent applications such as engineering, entertainment and management. There may be a significant amount and variety of research going on in the field of autonomous virtual agents, but not all aspects of the problem have yet been explored.

1.2 Aim

This thesis aims to improve the performance of the reactive behaviour of autonomous virtual agents in virtual environments. The virtual agent acquires some capabilities of perceiving their environment and is able to react and make decisions, depending on this input. It is important that the virtual agent needs to be situated in a common environment otherwise, no interaction is possible.

In this thesis we focus on reactive (non-adaptive or engineering) approaches. The aim is to develop a new control architecture for virtual agents so that they can behave autonomously in virtual environments. Autonomy will be judged based on their capabilities to react to changes in the environment, reason and make decisions by themselves, based on acquired information. The objectives are:

1. to improve the level of autonomy by having reliable action-selection mechanisms;

2. to design individual behaviours, synchronization and fusion using fuzzy logic and to integrate these behaviours with a fuzzy controller and the virtual agent;

3. to generate smooth behaviour animation of the virtual agent in real-time.

The main implementation of the proposed method is for solving problems in autonomous virtual agent navigation in virtual environments. Autonomous virtual agent navigation can be described as the ability of a virtual agent to move purposefully without user
intervention. The basic problem of navigation is moving from one place to another by the coordination of planning, sensing and control. Not all of this information is known prior to the planning process, and the navigation path is generated according to on-line user specifications; and the virtual agent cannot be prepared ahead of time [Li 99]. Experiment and evaluation has been conducted to measure the robustness of the proposed method and the results have also been presented.

The secondary implementation is to investigate how the same method can be used in other domains. Computer game domains have been used. The main challenge in the development of virtual agents in computer games is what should be the general nature of this kind of virtual agent for interesting game playing; and what type of architecture will best facilitate such character and environment. This is important where each game has its own strategy, action, curiosity, challenge and fantasy that make the game unique and interesting and which can essentially motivate games players [Hsu 06]. Only simple performance and user evaluation is done in this case since full evaluation is conducted in the autonomous virtual agent navigation implementation.

1.3 Problem Description

Reactive virtual agents perceive their environment and respond in a timely fashion to changes that occur in it. They maintain no internal model of how to predict future states of the world. They choose actions by using the current world state as an index into a table of actions, where the indexing function’s purpose is to map known situations to appropriate actions. These types of virtual agent are sufficient for limited environments where every possible situation can be mapped to an action or set of actions. The major drawback is its lack of adaptability. This type of virtual agent cannot generate an appropriate plan if the current world state was not considered a priori. In domains that cannot be completely mapped, using reactive virtual agents can be too restrictive.

Reactive virtual agents simply retrieve pre-set behaviours similar to reflexes, without maintaining any internal state. In contrast, deliberative virtual agents behave more like they are thinking, by searching through a space of behaviors, maintaining internal state, and predicting the effects of actions. Although the line between reactive and deliberative agents can be somewhat vague, a virtual agent with no internal state is certainly reactive, and one which bases its actions on the predicted actions of other virtual agents is deliberative.

Besides reactive virtual agents are the deliberative ones. The key component of
Chapter 1. Introduction

a deliberative agent is a central reasoning system [Ginsberg 89] that constitutes the intelligence of the agent. Deliberative agents generate plans to accomplish their goals. A world model may be used in a deliberative agent, increasing the agent’s ability to generate a plan that is successful in achieving its goals even in unforeseen situations. This ability to adapt is desirable in a dynamic environment.

Therefore, when the deliberative virtual agent is dealing with real-time systems it has problems with reaction time [Pérez 00]. They behaved more like they are thinking, by searching through a space of behaviours, maintaining their internal state, and predicting the effects of their actions. For simple, well known situations, reasoning may not be required at all. In some real-time domains, minimizing the latency between changes in world state and reactions is important. The constraints with this kind of behaviour are conflicts with cost, time and quality. Optimization of one or two of the objectives, often results in a sacrifice of a third objective.

Most reactive (behaviour-based) systems rely on their modularity as their source of reactivity. Complex behaviour can be achieved by combining several simple behaviour-producing units. Any particular behaviour may express itself opportunistically or when needed [Bryson 00]. They provide a framework in which different sub-problems can be isolated, dealt with, and integrated. Unfortunately, this architecture gives rise to three main problems as follows:

1. how to design a simple behaviour that guarantees robust operation and decides which behaviour should be activated at each instant;
2. how to integrate the process at different levels and combine the results from different behaviours into one command to be sent to the virtual agent;
3. how to ensure consistency between behaviours used by different modules, at different levels of abstraction and affected by different types of uncertainty.

The control of a virtual agent is shared between multiple behaviours with different and possibly incompatible goals. Each behaviour is responsible for controlling the virtual agent to achieve or maintain a particular objective. The goal of one behaviour might be in conflict with the goals of others. Therefore, the main problem is to decide what next action to select. Action selection is the means by which a virtual agent (either an animal or an autonomous artificial system) determines at any instant what to do next. The questions are what and how is it being selected. Thus, a main consideration is the formulation of effective mechanisms for coordination of the behaviour activities into strategies for rational and coherent behaviour.
However in most cases the virtual environment itself also plays a major role, resulting in the failure of the virtual agent to reach its goal. The main reasons are [Latome 91, Zhukov 00, Lozano 02]:

1. intelligent virtual agents may have arbitrary complex locomotion capabilities that are required to simulate a real world or imaginary character;

2. all computation must be performed in real time;

3. depending on the environment description the global path obtained will contain the set of cell centroids the virtual agent must visit to reach its target goal;

4. knowledge of the environment is partial, uncertain, imprecise and approximate; and

5. the environment is vast and dynamic and the obstacles can move, appear or disappear.

Issues (4) and (5) affect the behaviour rule selection.

In the past, several works relating to virtual agents have been done which describe mathematical models [Lerman 01] and fuzzy logic systems [Yen 99] for behaviour selection. However the limitations are the insufficient knowledge based perception of the environment and the absence of a decision making capability similar to that of a human driver.

1.4 The method

The main idea is to incorporate a virtual agent with behaviour-based control using only fuzzy logic (such as fuzzy rules and fuzzy reasoning) for coordinating conflicts and competition among different operations of reactive behaviour. This can be done by subdividing the overall task into small independent behaviours that focus on execution of specific sub-tasks [Seraji 02]. A coordinator is needed in order to send only one command at a time for action execution. The basic structure consists of all behaviours, taking input from the sensors and sending output to the actuators. A new behaviour-based fuzzy controller is established to optimize the fuzzy behaviour rules using Fuzzy Associative Memory (FAM). FAM maps the complete input space to the outputs. The fuzzy $\alpha - level$ technique, as a behaviour selection method, has been developed to decide which behaviour task needs to be executed. The local minima algorithm has also been used to help the virtual agent escape from traps or dead-ends.
1.5 Why Fuzzy Logic

Reactive systems are systems whose role is to maintain an ongoing interaction with their environment rather than produce some final value upon termination. Additional to that, most reactive control systems do not utilize sets of behaviours; instead, they rely on a single type of behaviour to guide the system. The architecture sometimes is too complex and integration among different behaviours is very difficult. For that reason, one of the solutions is to use fuzzy logic.

Fuzzy logic does not need a mathematical description of how the output functionally depends on the input. It is relatively easy to implement a system that deals with many situations without defining an analytical model of the environment, by representing relations between inputs and outputs in an IF–THEN manner and constructing a knowledge base. It is reactive because there is no planning stage [Reignier 94].

Fuzzy logic provides a means of transforming a linguistic control strategy based on expert knowledge into an automatic control strategy [Ross 04]. It appears to be very useful for handling problems that are too complex to be analyzed by conventional quantitative techniques or when the available sources of information provide qualitative, approximate, or uncertain data. Reactive navigation of a mobile robot, for example, falls into this class of problems that fuzzy control systems cope well with. Fuzzy logic is suitable for multi-sensor fusion and integration.

The goal of behaviour-based systems is to subdivide the overall task into small independent behaviours that focus on execution of specific sub-tasks [Pérez 00]. Fuzzy logic can be used to design individual behaviour. Behaviour complexity can be reduced by a divide and conquer approach, which attempts to break down the overall problem into more manageable sub-behaviours.

Fuzzy logic also gives promising results in addressing the integration problem. Fuzzy control can be used to integrate explicit domain knowledge in the form of linguistic rules that describe the behavioural mapping from perception to action. These rules constitute an initial, sub-optimal behaviour that is later refined through experiences gathered from the virtual agent’s interaction with the environment [Hoffmann 03].

Behaviour coordination has two distinct problems which are: (i) how to decide which behaviour should be activated; and (ii) how to fuse the output of concurrent, possibly conflicting behaviours [Saffiotti 97]. IF-THEN rules can be used for the first problem and fuzzy connectives used for the second problem. Fuzzy IF-THEN rules allow for partial activation depending on how much that behaviour is relevant to the
current situation in which truth can assume a continuum of values between 0 and 1. This leads to a fusing of different local control laws (behaviours) into an overall complex control strategy.

Other AI techniques such as neural networks [Zurada 95], machine learning [Alpaydin 04] and evolutionary algorithms [Eiben 03] are an inspiration from the capabilities of animals and humans to adapt and learn in dynamic environments under varying conditions, situations and tasks. Fuzzy logic is inspired by the approximate type of reasoning that allows humans to make decisions under uncertain and incomplete information. In the context of the above mentioned trade-offs imposed on virtual agent learning, fuzzy techniques offer a means to sacrifice optimal performance for a reduction in complexity, elimination of unnecessary details and increased robustness of solutions.

Finally, a fuzzy controller provides efficient implementation. These characteristics are required for an autonomous virtual agent where a mathematical model of the environment is not available, sensor data is uncertain and imprecise and real-time operation is required.

1.6 Thesis Organization

Chapter 2: Literature Review

The literature review contains an overview of virtual environments and continue with what is an autonomous virtual agent, types of virtual agent and some of the applications. Then, we continue with virtual agent control architecture and how virtual agents can be fitted in solving specific problems. Finally, there is a discussion on the action selection problem, its classification and some of the methods that have been used in solving the problem.

Chapter 3: Methodology

This chapter describes the method that has been used in solving the problem of autonomous virtual agents. The behaviour design and modeling of virtual agent control systems are described here. Then, we continue with action selection methods and the integration with virtual agent control systems.
Chapter 1. Introduction

Chapter 4: Autonomous Virtual Agent Navigation

This chapter describes the behaviour-based architecture developed for autonomous virtual agents. We discuss the integration of the fuzzy controller with the virtual agent and how this integration can be used so that the virtual agent can perform navigation tasks in unknown virtual environments. We also describe the experimental objectives, setup and methods used for autonomous virtual agent navigation. The results from experiments are discussed and used to evaluate how and to what extent the fuzzy system solution has solved the problem.

Chapter 5: Autonomous Virtual Agent in Computer Game

The aim of Chapter 5 is to describe how the fuzzy method can be used in other domains. A computer game domain is selected which requires controlling virtual agents/characters. The Pacman game is used and modified to fit with our fuzzy method. Evaluation and results are discussed.

Chapter 6: Conclusion

The final conclusions of the thesis, its contribution and some future research directions have been proposed.

1.7 Publications

The following refereed publications have been published. They account for work carried out in this thesis.


Chapter 2

Literature Review

*Thinking is easy, acting is difficult, and to put one's thoughts into action is the most difficult thing in the world.*

- Johann Wolfgang von Goethe (1749-1832)
  German Playwright, Poet, Novelist and Dramatist

2.1 Introduction

Generally there is still a gap between the methods implemented by researchers from graphics backgrounds for controlling a virtual agent and those favoured by researchers from AI and ALife backgrounds. The dividing issue is often one of artistic or directorial control versus agent autonomy. Many researchers who have moved from animation still favour various kinds of scripting where AI and ALife researchers often think in terms of sensor driven behavioural control or of goal driven action supported by symbolic reasoning.

Autonomy is recognizably and undeniably a critical issue in the field of intelligent virtual agents and multi-agent systems, yet it is often ignored or simply assumed. Autonomous virtual agents should operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state [Wooldridge 95]. They continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions [Hayes-Roth 95].

In relation to that, action selection mechanisms play a major role in autonomous agents. The agents are normally created to perform several different tasks. The acting agent typically must select its action in dynamic and unpredictable environments; act
in real time; and make decisions in a timely fashion. These tasks may conflict for resource allocation.

In recent years, researchers have proposed many approaches to solve the problem. These approaches, in general, can be divided into engineering (non-adaptive) and adaptive approaches. Both approaches can use reactive, deliberative and hybrid architectures, to achieve the same goal in different ways based on their basic features.

### 2.2 Virtual Environments

A virtual environment or virtual world is a simulated environment that appears to have the characteristics of some other environment, and in which participants perceive themselves as interactive parts [ATIS 00]. Virtual environments have been used in many different fields such as computer games, entertainment, engineering and manufacturing. Computer hardware is not a major issue; however, production of more dynamic and interesting virtual environment systems or applications remains a challenge to developers in this field.

A general definition of a virtual environment is as a computer-based simulated environment (computer-generated world) intended for its users to inhabit and interact via avatars [Durlach 95]. This habitation usually is represented in the form of two or three-dimensional graphical representations of humanoids (or other graphical or text-based avatars) [Ellis 94, Brooks 99] with which the user can interact, with the purpose of altering the state of the user or the computer [Youngblut 96]. The environment contains synthetic sensory information that leads to perceptions of environments and their contents as if they were not synthetic [Blascovich 02]. The challenge is to make that virtual environment look real, move and respond to interaction in real time, and even feel real. The user views the virtual environment indirectly through a computer monitor or some other display.

Virtual environments have been increasingly used for a variety of contexts as following [Aylett 01]:

1. adding a problem-solving component to the virtual environment;

2. building a knowledge level supporting conceptual scene representation, which can support high-level processing of the graphic scene itself, or interface with natural language processing systems;
2.3.1 What is an Autonomous Agent?

It is noticeable how the following definition includes the impact of the autonomous agent’s own current behaviours on its own future behaviours. An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors [Russell 03]. According to [PCAI 02], an autonomous agent is software that is given a particular mission, carries out that mission, and then reports back to the user. Therefore, the agent is a software routine that waits in the background and performs an action when a specified event occurs [ZDNet Dictionary 07]. The problem is that if the environment provides input and receives output, and considers input to be sensing, and produces output to be acting, then every computer program is a virtual agent.

[Franklin 96] has introduced an agent as 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. This includes all of the basic features of intelligent agents except their sociability. It provides a good approximation of the basic features of the large variety of intelligent agents now under development. An intelligent agent is a system that performs diverse behaviours in its efforts to achieve multiple goals in a dynamic, uncertain environment [Morignot 96].

There is a convergence of opinion that an autonomous agent is a computer software system whose main characteristics are [Wooldridge 95]:

**reactivity:** agents perceive their environment 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, when appropriate;
and learning from its own experience, its environment, and interactions with others.

**autonomy:** agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;

**sociability:** agents interact with other agents (and possibly humans) via some kind of agent-communication language;

In this thesis, sociability has not been considered as a characteristic of the virtual agent. The main reason is that each agent has its own preferred behaviour and works individually. Even though there is more than one agent, the agents do not interact with each other.

Each autonomous agent is situated in, and is a part of some environment. Each senses its environment and acts autonomously within it. No other entity is required to feed it input, or to interpret and use its output. Each acts in pursuit of its own agenda, whether satisfying evolved drives as in humans and animals, or pursuing goals designed in by some other agent. Each acts so that its current actions may affect its later sensing, that is, its actions effect its environment. Finally, each acts continually over some period of time [Franklin 97].

The notion of individuality is very important for autonomous agents because they should decide their actions according to internal and external states on their own. The final decision is made by the agents. Further, along with being reactive, an agent must also be proactive. That is, it must be able to take initiative and be opportunistic when necessary. The notion of planning their behaviours to anticipate future actions is also necessary and to plan sequences of actions to reach a specific goal.

### 2.3.1.1 Autonomy

Autonomous systems must be automatic systems and, in addition, they must have the capacity to form and adapt their behaviour while operating in the environment [Steels 95]. It is generally a necessary condition that the behaviour of an autonomous system is characterized by some capacity for stable and/or flexible interaction with its environment.

Autonomy has many interpretations in terms of the field in which it is being used and analysed, but the majority of the researchers in IVEs argue in favour of a strong and life-like notion of autonomy, which should first of all replace omniscience in virtual
worlds. As such, even from a practical perspective, autonomy is not a needless overhead. Since believability is considered as a crucial factor, virtual agents should appear to have limitations in their interaction with the environments, just as agents in the real world have [Arnellos 08]. Here we adopt the robotic definition that an autonomous agent has a sense-reflect-act cycle of its own operating in real-time in interaction with its environment. The amount of autonomy possessed by an agent is therefore related to its control architecture.

However, some researchers take autonomy as an all-or-nothing property: either a system is autonomous or it is not [Luck 95]. Table 2.1 shows the characteristics of different autonomous agent controls. It seems that the level of autonomy is a key property of the agent. The agent is engineered so as to be able to interact with its environment without requiring ongoing human intervention. It must be capable of satisfying some goal (or even of generating its own goals) and also have robust and flexible behaviour. The virtual agent, at least to some extent, is independent and also not entirely pre-programmed, but can make decisions based on information from its environment or other agents without intervention by any other agent [Monzani 02].

Concerning autonomy in behavioural choice, several levels exist depending on the importance of the user control of the virtual agents. [Boden 96, Blumberg 97] define dimensions of autonomy based on their degree of autonomy or levels of autonomy:

- The virtual agent is a direct extension of the user, but the desired level of interaction is such that the user wishes to provide control at a high level and rely on

Table 2.1: Characteristics of Different Virtual Agent Controls [Ferreira 02]

<table>
<thead>
<tr>
<th>Behaviour Control</th>
<th>Guided Agents</th>
<th>Programmed Agents</th>
<th>Autonomous Agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level of Autonomy</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Level of Intelligence</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Execution Frame-rate</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Complexity of Behaviour</td>
<td>Low</td>
<td>Variable</td>
<td>High</td>
</tr>
<tr>
<td>Level of Interaction</td>
<td>High</td>
<td>Variable</td>
<td>Variable</td>
</tr>
</tbody>
</table>
Chapter 2. Literature Review

the competence of the virtual agent to accomplish the task.

• The virtual agent is not directly driven by the user but interacts with him and other virtual agents in a relatively structured environment.

• The virtual agent is intended to give the illusion of being alive and of having an existence independent of the user.

• The extent to which responses to the environment are direct or indirect.

• The extent to which the controlling mechanisms are self-generated rather than externally imposed.

• The extent to which inner directing mechanisms can be reflected upon and/or selectively modified.

Fundamentally, autonomy is about choices, and about being self-contained. The implicit assumption is that the agent is constantly faced with non-trivial choices, and must decide on its own how to respond. It is self-contained in the sense that it does not rely on an external entity, i.e., a virtual agent or a centralized decision-maker to make its decisions for it.

2.3.1.2 Autonomous Behaviour

Behaviour refers to the actions or reactions of an object or organism, usually in relation to the environment. In other words, behaviour itself means a complex action of a human or other animal based on volition or instinct and the autonomous agent might need this. Behaviour of autonomous agents is generally viewed as goal-directed which contributes to the following features [Reynolds 99]:

1. action selection - noticing that the state of the world has changed and setting a goal;

2. steering - represented by the virtual agent, who decomposes the goal into a series of simple sub-goals; and

3. locomotion - taking the virtual agent’s control signals as input and moving in the indicated direction. This motion is the result of a complex interaction of visual perception, its sense of balance, and muscles applying torques to the joints of its skeleton.
Figure 2.3 shows an example of autonomous behaviour framework of a virtual agent that can reproduce several human behaviour features [Iglesias 04]. Each system can be broken up into smaller subsystems, each associated in turn with more specific routines. The physical system includes the perception and motion subsystems, while the behavioural system includes the analyzer, the knowledge motor, the internal states and the goal subsystems.

2.3.2 Types of Agents

An autonomous agent can be defined as an autonomous entity in a virtual environment. It should not only look like, but also behave as a living organism in a synthetic 3D world, and be able to interact with the world and its inhabitants [Vosinakis 01]. The virtual agent must combine aspects of an autonomous robot with some of the skills of a human actor in an improvisational theatre [Reynolds 99]. [Brooks 91] refers to embodiment and situatedness as the two cornerstones to the new approach to Artificial Intelligence, and defines them as follows:

**Situatedness**

The agents are situated in the world - they do not deal with abstract descriptions but with the here and now of the world directly influencing the behaviour of the system.

**Embodiment**

The agents have bodies and experience the world directly - their actions are part of a dynamic with the world and have immediate feedback on their own sensations.
The notion of situatedness is often forgotten compared to the others and is very important in real-time environments. It implies the use of a bottom-up approach. A situated agent is defined as an agent which [Steegmans 04]:

- is situated in an environment,
- is driven by a survival/satisfaction function,
- possesses resources of its own in terms of power and tools,
- is capable of perceiving its environment (but to a limited extent),
- has practically no representation of its environment
- possesses skills
- can perhaps reproduce

Situatedness places an agent in a context in which it is able to perceive its environment and in which it can (inter)act. The agent also acts in such a way as to possibly influence what it senses at a later time. It is structurally coupled to its environment [Maturana 75, Maturana 80]. Situated agents do not use long-term planning to decide what action sequence should be executed, but select actions based on the locally perceived state of the world and limited internal state. Contrary to knowledge-based agents, situated agents do not emphasize internal modelling of the environment. Instead, they prefer to employ the environment itself as a source of information. The environment can serve as a robust self-revising common memory for agents. This can unburden the distinctive agents from continuously keeping track of their knowledge about the system. The benefits of situatedness are well known: flexibility, robustness and efficiency.

### 2.4 Control Architectures

An agent's control architecture is the structure of its agent program, and the description of information and control flows through its different components. Different architectures can produce the same agent function, but their implementation will be different. Maes [Maes 91] defines an agent architecture as:

A particular methodology for building [agents]. It specifies how ... the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of
modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions ... and future internal state of the agent. An architecture encompasses techniques and algorithms that support this methodology.

Kaelbling [Kaelbling 91] considers an agent architecture to be:

A specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the data and control flow among the modules. A more abstract view of an architecture is as a general methodology for designing particular modular decompositions for particular tasks.

In general, the autonomous agents are controlled by four main approaches which are scripting, reactive, deliberative and hybrid approaches. Scripting allows a very detailed level of control, but is very inflexible [Thalmann 04]. Reactive architectures in Figure 2.4(b) use a bottom-up philosophy and react to the changing environment according to the sets of rules. They should be sufficiently flexible to adapt to changing environments and changing requirements. Reasoning strategies allow them to anticipate the consequences of possible actions and choose the most rational action.

![Diagram of three types of agent control architecture](image)

**Figure 2.4: Three Types of Agent Control Architecture [Brooks 86, Pérez 00]**

Deliberative architecture in Figure 2.4(a) is similar to reactive agents, and this most popular implementation is probably seen in games such as The Sims. Deliberation is typically a time and space consuming operation. The design of the control architecture is based on a top-down philosophy, and the control architecture is broken down into an orderly sequence of functional components, and the user formulates explicit tasks and goals for the system.

Hybrid architectures, in Figure 2.4(c), are a combination of deliberation with a reactive behaviour pattern to allow timely reactions within a dynamic environment [Wooldridge 95]. Hybrid systems attempt to compromise between bottom-up and top-down methodologies. Usually the control architecture is structured in three layers: the deliberative layer, the control execution layer and the functional reactive layer.
As an alternative to using the above mentioned approaches some researchers have used L-systems [Noser 95, Noser 05] and vision systems [Peters 02, Peters 03]. The L-system is a timed production system designed to model the development and behaviour of static objects, plant-like objects and autonomous creatures. It is based on a timed, parametric, stochastic and conditional production system, force fields, synthetic vision and audition, which are completely defined by production rules. Furthermore, in vision systems, the virtual agent senses external stimuli through a synthetic vision system. The vision system incorporates multiple modes of vision in order to accommodate a perceptual attention approach. A memory model is used to store perceived and attended object information at different stages in a filtering process.

2.4.1 Deliberative Architecture

Deliberative agents are also called cognitive agents, intentional agents or goal-directed agents, which is the classical architecture. The deliberative approach involves the agent knowing its environment, developing an internal world model, a map, and making decisions based on this information. This virtual agent will move about and perform tasks in a deliberate manner. It relies on planning and hypothesis exploration. A deliberative architecture is one that reasons about future events, takes into consideration the outcome of its action, and tries to build a set of actions towards a specific goal. It generally uses logic and symbolic reasoning. This approach has two main problems:

1. how to translate the environment into the appropriate symbolic description, and
2. how to symbolically represent information about a complex environment, and all that in time for the agent to act properly.

Computation time is not a problem in a step by step simulation, but it can be in a real-time context. This type of architecture lacks reactivity, having to consider several steps ahead before taking any action. Figure 2.5 shows an example implementation of the JADEX architecture [Pokahr 03] based on the BDI model [Rao 95]. The BDI model enables us to view an agent as a goal-directed entity that acts in a rational manner. Viewed from the outside, the agent is a black box, which receives and sends messages. Incoming messages, as well as internal events and new goals, serve as input to the agent's internal reaction and deliberation mechanism. Based on the results of the deliberation process these events are dispatched to already running plans, or to new
plans instantiated from the plan library. Running plans may access and modify the belief base, send messages to other agents, create new top-level or sub-goals, and cause internal events.

![Deliberative Architecture Diagram](Figure 2.5: Deliberative Architecture [Pokahr 03])

Some of the popular approaches are learning and evolutionary methods. For example, [Uhrmacher 00] developed a simulation layer of a Java Based Agent Modeling Environment for Simulation (JAMES) that implements a moderately optimistic strategy which splits simulation and external deliberation into different threads and allows simulation and deliberation to proceed concurrently by utilizing simulation events as synchronization points. [Lee 04a] used neural networks for the behaviour decision controller. The input of the neural network is decided by the existence of other agents and the distance to the other agents. The output determines the directions in which the agent moves. The connection weight values of this neural network are encoded as genes, and the fitness of individuals is determined using a genetic algorithm. Here, the fitness values imply how much group behaviours fit adequately to the goal.

### 2.4.2 Reactive Architecture

The Reactive approach involves the autonomous agent reacting to its environment with tight sensing - acting connections. These virtual agents do not have a plan, nor do they have a map. These agents explore their world and react to the environment as they encounter it. A reactive architecture relies on a quick response. They are based on the assumption that intelligent behaviour can be generated without explicit representation nor explicit reasoning (as it is the case in deliberative architectures) and that intelligence is an emergent property of certain complex systems.
Reactive agents are also called *situated* agents. The basic structure of a reactive agent consists of all behaviours, taking input from the sensors and sending output to the actuators [Pérez 00]. A coordinator is needed in order to send only one command at a time. The goal is achieved by subdividing the overall task into small independent behaviours that focus on execution of specific sub-tasks [Seraji 02]. The resulting architecture can be very simple, but fast (in computational time) and efficient. They perform well in quickly changing complex environments (in which time is important), though they lack the adaptability of deliberative planning. However, there are three main problems [Li 94, Pérez 00]:

1. it is hard to formulate reactive behaviour quantitatively and also there might be no applicable approach to coordinating conflict;
2. there is competition among different reactive behaviours to achieve a good performance; and
3. how to select the proper behaviours for robustness and efficiency in accomplishing goals.

An example of reactive architecture with a focus on the functional decomposition of an agent's behaviour is depicted in Figure 2.6. Agents produce influences into the environment and subsequently the environment reacts by combining the influences to deduce a new state of the world from them. The reification of actions as influences enables the environment to combine simultaneously performed activities.

Traditional architectures for reactive agents (see e.g. [Brooks 91, Tyrrell 93, Maes 97, Bryson 01]) take the viewpoint of the individual agent to select the most appropriate
action. This architecture can be summarized based on their characteristics, as in Table 2.2.

Table 2.2: Reactive Architectures and their Basic Characteristics [Arkin 98, Pérez 00]

<table>
<thead>
<tr>
<th>Control Architecture</th>
<th>Behavioural Choice and design</th>
<th>Assembling behaviours</th>
<th>Programming method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsumption architecture</td>
<td>Experimentally</td>
<td>Competitive, arbitration via inhibition and suppression</td>
<td>Augmented Finite State Machines (AFSM), Behaviour language or behaviour libraries</td>
</tr>
<tr>
<td>Action Selection Dynamics</td>
<td>Experimentally</td>
<td>Competitive, arbitration via level of activation</td>
<td>Mathematical algorithm</td>
</tr>
<tr>
<td>Schema-based approach</td>
<td>Ethologically</td>
<td>Cooperative via vector summation and normalisation</td>
<td>Parameterised behavioural libraries</td>
</tr>
<tr>
<td>Process Description Language</td>
<td>Experimentally</td>
<td>Cooperative via description and interaction of different processes</td>
<td>Process Description Language</td>
</tr>
</tbody>
</table>

More recent work by [Pisan 02] proposes an architecture that uses a logic-based truth maintenance system coupled with a rule engine to create articulate agents capable of having conversations with the player. [Weyns 06] introduced a virtual environment for agents to live in. This virtual environment offers a medium that agents can use to exchange information and coordinate their behaviour, and serves as a suitable abstraction to shield low-level physical processing from the agents. Since the only infrastructure available to the Automatic Guided Vehicles (AGVs) is a wireless network, the virtual environment is necessarily distributed over the AGVs. Synchronization of the state of the virtual environment is provided by ObjectPlaces, a middleware infrastructure that offers support to exchange and share information among nodes in mobile and ad-hoc networks.

2.4.3 Hybrid Architecture

Hybrid architectures attempt to combine deliberative and reactive processes to get the advantages of both types of architecture. The processes are used in parallel and allow both quick response and planned behaviours. The core of the architecture is that the behaviours of an autonomous agent can be specified as a dynamical system. This ar-
Chapter 2. Literature Review

Architecture includes reactivity (elementary behaviours level) and BDI (high behaviours level). The main function of the elementary behaviours level is the agent's basic actions. With the use of these behaviours the agent can accomplish simple tasks without coordination with other agents. Where the agents must collaborate with other agents, the high level behaviours can be named collaborative behaviour. These behaviours can act in complex tasks, which the elementary behaviours could not execute.

In some architectures, fuzzy logic is an alternative to a Bayesian approach [Berger 93]. The Bayesian approach is based on a rigorous theory with a vast amount of known results [Lindley 87]. The lack of ability to handle continuous input, requires a vast amount of storage and computational manipulation making this probabilistic method computationally infeasible. If integrated with the fuzzy logic approach of making the data members of discrete sets, the hybrid system should be able to handle all the demands of uncertainty [Rao 08].

More recent work by [Karim 06] has investigated ways in which different cognitively styled agents using knowledge representations of varying levels of abstraction can be combined into a hybrid architecture. They used a reactive learner known as Falcon, which is based on a reinforcement learning technique, with that of a high-level plan execution engine, and reactive plan execution engine based on BDI known as JACK. [Bajo 07] has developed deliberative planner agents using Case-Based Reasoning (CBR) systems. This hybrid architecture meets the conditions needed to introduce a representation and a reasoning based on the action. This is because a CBR-BDI agent uses case-based reasoning as a reasoning mechanism, which allows it to learn from initial knowledge, to interact autonomously with the environment as well as with

![Figure 2.7: Hybrid Architecture [Franklin 97]]
users and other agents within the system, and to have a large capacity for adaptation to the needs of its surroundings.

2.5 Action Selection Mechanism

In general, action can be referred to as the process or state of acting or of being active. A more precise definition comes from Webster’s Revised Unabridged Dictionary [DICT.Org 13] which defines action as:

A process or condition of acting or moving, as opposed to rest; the doing of something; exertion of power or force, as when one body acts on another; the effect of power exerted on one body by another; agency; activity; operation; as, the action of heat; a man of action.

In the autonomous agent context, action selection also refers to activation of a behaviour best suited to the agent. Thus, agents can have two roles in an agency: actions and action selection mechanisms. Based on these two roles, it subordinates an agent as an action selection mechanism, and with respect to its superior, an agent is viewed as action [Pirjanian 99a]; this is illustrated in Figure 2.8. The two roles of an agent in the agent hierarchy are that from its own point of view an agent is an action selection mechanism, whereas from a superior’s point of view it is an action.

One fundamental question about decision making or action selection is whether it is really a problem at all for an autonomous agent, or whether it is just a description of an emergent property of an intelligent autonomous agent’s behaviour. However, the history of intelligent systems, both artificial [Bryson 00] and biological [Prescott 07], indicate that building an intelligent system requires some mechanism for decision making or action selection. This mechanism may be highly distributed, or it may be one
or more special-purpose modules, and also, the following features might be required [Brom 06]:

- The acting agent typically must select its action in dynamic and unpredictable environments.
- The agents typically act in real time; therefore they must make decisions in a timely fashion.
- The agents are normally created to perform several different tasks. These tasks may conflict for resource allocation.
- The environment the agents operate in may include humans, who may make things more difficult for the agent (either intentionally or by attempting to assist).
- The agents are often intended to model humans and/or other animals. However animal behaviour is quite complicated and not yet fully understood.

The action selection techniques determine not only the agent’s actions in terms of its impact on the world, but also directs its perceptual attention, and updates its memory. These self-centered sorts of actions may in turn result in modifying the agent’s basic behavioural capacities, particularly in that updating memory implies some form of learning is possible. Ideally, action selection itself should also be able to learn and adapt, but there are many problems of combinatorial complexity and computational tractability that may require restricting the search space for learning.

### 2.5.1 The Action Selection Problem

The problem of action selection is central each time autonomous entities such as robots, virtual characters, or humans are designed. The system should decide what to do next according to its internal and external information without outside interventions. Action selection is a control structure for an autonomous agent (see Figure 2.9) and can be considered as the mind of the agent. The continuing task of mind is to produce the agent’s next action to answer the only really significant question there is: what shall I do next?

In the decision process, multiple conflicting objectives are considered simultaneously, subject to certain constraints dictated by the agent limitations [Pirjanian 97]. The constraints are based on the complexity of the environment, unpredictabilities and
an agent's limited resources. This implies that the action selection cannot be completely optimal. The action selection should be fast, robust and good enough for satisfying a decision [Simon 77]. The decision making process searches for 'good enough' options, rather than an optimum solution. With satisfying, decision making becomes something which is carried out in a limited time, and with some limits on the individuals concerned [Brown 05]. According to [Maes 89, Tyrrell 93] the following requirements are needed in the development of a good enough action selection mechanism:

1. Goal-orientedness - it favours actions that contribute to one or several goals.

2. Situatedness - it favours actions that are relevant to the current situation.

3. Persistence - it favours actions that contribute to the ongoing goal.

4. Planning - it looks ahead to avoid hazardous situations.

5. Robustness - it never completely breaks down, even when certain components fail.

6. Reactivity - it provides fast and timely responses.

7. Dealing with all types of sub-problem - the same action selection should handle all sub-problems.

8. Compromised actions - the need to choose actions that are best for the collection of behaviours rather than individual behaviour.

9. Opportunism - should allow the agent to interrupt the ongoing goal and pursue a new one.
Some of these requirements might be conflicting with each other, for example, planning is in conflict with reactivity [Pirjanian 99a]. This is not a major problem since it depends where and when the action selection is to be used. Also, there are some agent architectures and action selection methods that use both planning and reactive approaches, such as in hybrid systems. The main concern is how this method can fulfill the autonomous agent goal of good enough action.

At every instant the agent should choose the actions which can achieve its objective, given its internal state (e.g. food and water needs), its perception of its environment, and its repertoire of possible actions. Moreover, the temporal pattern of its behaviour should make sense as well. If it is working on a given goal, it should continue working on that goal until either the goal is satisfied or something better comes along. That is, it should be able to balance persistence with opportunism and have a sense of whether it is making progress, i.e., it should not get stuck in mindless loops [Maes 90]. Many problems are linked with action selection such as action persistence, evaluation of the action choice, chaining actions to obtain coherent behaviours, authorizing opportunistic and compromise behaviours [Blumberg 94].

2.5.2 Classification of Action Selection

Many action selection methods have been proposed, yet there is still no clear classification of the different techniques in the literature. A global classification, accepted by the majority of scientists [Ziemke 98], lists systems according to their adaptability and might depend on when and where it is being used, and also on how the action selection has been accomplished.

![Classification of Action Selection](image)

Figure 2.10: Classification of Action Selection (a) [MacKenzie 97] (b) [Saffiotti 97]

One of the earlier classifications was introduced by [MacKenzie 97] in Figure 2.10(a) and by [Saffiotti 97, Pirjanian 99a] in Figure 2.10(b). The classifications have some
similarity, in that arbitration (Competitive behaviour) and command fusion (Cooperative behaviour) corresponded to state-based and continuous approaches, respectively. Both also focus on the problem of behaviour coordination and command fusion.

Behaviour coordination is concerned with how to decide which behaviour to activate at each moment or state, and command fusion is concerned with how to combine the results from different behaviours into one command [Pirjanian 99a]. Alternatively, [Brom 06] classified action selection into a symbol-based system (classical planning), distributed solution and reactive planning (dynamic planning). Even though there is some similarity with [MacKenzie 97] and [Saffiotti 97, Pirjanian 99a] classification, the action selection has been classified based on agent architecture, as described in Section 2.4.

Arbitration and command fusion action selection mechanisms are mutually exclusive in that the same set of behaviours cannot use both mechanisms at the same time. However, it is still possible to use them together in the same architecture as long as there is a way to decide which selection mechanism gets to select behaviours at any given time, for example in hybrid architecture [Scheutz 04]. Furthermore, [Scheutz 02] expanded arbitration and command fusion into implicit behaviour and explicit behaviour.

Implicit behaviour selection uses structural features of the architecture to select behaviours. This can be seen, for example, through the relative strengths of inhibitory and excitatory connections among components as in the cooperative example of Braitenberg vehicles [Braitenberg 84]. In contrast, explicit behaviour selection uses specialized components. Implicit and explicit behaviour selection mechanisms are also mutually exclusive analogous to competitive and cooperative mechanisms. Similar to arbitration and command fusion, implicit and explicit behaviour also can coexist in one architecture.

Table 2.3 summarizes the classification of action selection mechanisms that have been discussed in this section. We have distinguished between arbitration (Competitive behaviour) and command fusion (Cooperative behaviour) action selection methods. Although both methods can be used in reactive and deliberative architectures, the distinction between reactive and deliberative behaviour selection is pertinent to the run-time instance of an architecture. They also can been used together, for example in a Hybrid architecture. Hybrid architectures may consist of a command fusion action-selection mechanism in the reactive layer, and an arbitration action-selection mechanism in the deliberative layer.
Table 2.3: Examples of Action Selection Methods.

<table>
<thead>
<tr>
<th>Reactive</th>
<th>Command Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arbitration</strong></td>
<td><strong>Fusion</strong></td>
</tr>
<tr>
<td><em>Explicit</em></td>
<td><em>Agent Network [Maes 89]</em></td>
</tr>
<tr>
<td></td>
<td><em>Bayesian Network Analysis [Kim 03]</em></td>
</tr>
<tr>
<td></td>
<td><em>Probabilistic Method [Dix 00]</em></td>
</tr>
<tr>
<td><em>Implicit</em></td>
<td><em>Subsumption [Brooks 86]</em></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliberative</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arbitration</strong></td>
<td><strong>Fusion</strong></td>
</tr>
<tr>
<td><em>Explicit</em></td>
<td><em>Alliance [Parker 97]</em></td>
</tr>
<tr>
<td></td>
<td><em>Yamada [Yamada 01]</em></td>
</tr>
<tr>
<td><em>Implicit</em></td>
<td></td>
</tr>
</tbody>
</table>

2.5.3 **Reactive Action Selection Method**

In designing decision making architectures for virtual agents, two approaches exist which are Top-Down approach and the Bottom-Up approach. Reactive architectures (Behaviour-based architectures), used principally in robotics [Maes 94, Mataric 97, Arkin 98], follow the Bottom-up approach and have been implemented to fix problems with traditional planning architectures:

- Constructing a complete plan before beginning action. A planner cannot determine whether a plan is viable before it is complete. Many plans are in fact formed backwards because of opportunities and changes in the environment.

- Taking too long to create a plan, thereby ignoring the demands of the moment.

- Being unable to create plans that contain elements other than primitive acts.

- Being unable to manipulate plans and goals.

behaviour-based models are used to implement fully reactive agents. A reactive system is designed from the beginning to be situated in a complex, dynamic environment, which it must constantly monitor and to which it must instantly react. They can respond quickly to new, unexpected or opportunistic situations in the environment whereas a traditional planner will continue to execute its script until the end even if the intention of the agent or the conditions of the plans are changed. Reactive agents will notice and take decisions according to opportunities which can fulfill any of their goals. Moreover in reactive agents, the information is always up-to-date and consequently the
behaviour plan also. This is because no information is stored. All information is a reflection of the current environment.

This section has a focus on the reactive architectures. These methods were designed independently and are based on different ideas within the field of Behaviour-based Robotics. The methods and their basic characteristics can be seen in Figure 2.10 and Table 2.3. Table 2.4 shows some of the methods that have been developed by several researchers. There is some overlap between these methods, and in some cases there are techniques which do not fall into any of the categories. The next section will discuss some examples of reactive action selection methods which are based on [Saffiotti 97].

2.5.3.1 Arbitration

Arbitration requires the selection of an action based on the result of some competition process among different components, possibly followed by the arbitration of the current behaviour (if an action is different from the current one that was selected during competition) [Scheutz 02]. Arbitration ASMs allow one behaviour or a set of behaviours at the same time to take control for a period of time until another set of behaviours is activated. Arbitration mechanisms select one behaviour, from a group of competence modules.

Priority-based

Subsumption architecture [Brooks 86] is based on priority-based mechanisms. The architecture consists of series of behaviours, which constitute a network of handwired finite state machines. Action consists of higher-level behaviours overriding (subsuming) the output of lower level behaviour [Pirjanian 99a]. The behaviour which has higher priority is allowed to take control of assigned priorities. These innovations allowed the development of the first robots capable of animal-like speeds [Brooks 90].

Figure 2.11 shows the structure of a subsumption architecture for agents in computer games developed by [Kenyon 06] in which the one-per-layer behaviour modules are not networked except for strict downwards subsumption. The system allows for future expansion to a series of Finite State Machines (FSM). The characteristics of this architecture are that each layer can receive sensor information and make responses to the change of environment without waiting for the higher layer's order. The architecture can be divided into three layers according to the tasks that system should achieve.
### Table 2.4: Virtual Agent Action Selection Methods [Delgado 04]

<table>
<thead>
<tr>
<th>Author</th>
<th>Disciplines</th>
<th>Design</th>
<th>Combination Stimuli</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brooks</td>
<td>Robotic</td>
<td>Distributed network of finite state machines</td>
<td>Subsumed</td>
<td>Physical robot</td>
</tr>
<tr>
<td>Blumberg</td>
<td>Ethology</td>
<td>Hierarchical behaviour system using releasing mechanisms with learning</td>
<td>Summed</td>
<td>3D graphics</td>
</tr>
<tr>
<td>Tyrell</td>
<td>Ethology</td>
<td>Loose hierarchy of behaviour</td>
<td>Can be any function</td>
<td>Grid</td>
</tr>
<tr>
<td>Humphry</td>
<td>Reinforcement learning, Brooksian ethology</td>
<td>W-learning Minimising “worst unhappiness”</td>
<td>Synthesised and subsumed</td>
<td>Grid</td>
</tr>
<tr>
<td>Bryson</td>
<td>Ethology</td>
<td>Reactive hierarchy</td>
<td>Synthesised</td>
<td>Grid</td>
</tr>
<tr>
<td>Montes</td>
<td>Basal ganglia neurology</td>
<td>Neurological model of mammalian basal ganglia</td>
<td>Leaky integration</td>
<td>Robot</td>
</tr>
<tr>
<td>Martinez</td>
<td>Robotic ethology</td>
<td>Reactive behaviours blended used in conjunction to accomplish a navigation task</td>
<td>Context depending blending</td>
<td>Robot</td>
</tr>
<tr>
<td>Reynolds</td>
<td>Animal behaviour computer graphics</td>
<td>Flocking behaviour with collision avoidance, velocity matching and flock centring</td>
<td>Vectorial summation</td>
<td>3D (rough)</td>
</tr>
<tr>
<td>Barnes</td>
<td>Robotic</td>
<td>Reactive behaviour synthesis</td>
<td>Synthesised</td>
<td>Physical robotic</td>
</tr>
<tr>
<td>Tu</td>
<td>Ethology computer graphics, physic based modelling</td>
<td>Physics based modelling of artificial fish, hierarchical action selection</td>
<td>Winner-takes-all</td>
<td>3D graphics</td>
</tr>
<tr>
<td>Arkin</td>
<td>Ethology guided</td>
<td>Perceptual processes attached to motor schemas</td>
<td>Vector summation</td>
<td>Physical robot</td>
</tr>
<tr>
<td>Maes</td>
<td>ANN and robotics</td>
<td>Non-hierarchical distributed network</td>
<td>Summed</td>
<td>Robot</td>
</tr>
<tr>
<td>Negrete</td>
<td>Neuro-physiology</td>
<td>Non-hierarchical distributed network neuro-humoral neuron</td>
<td>Summed</td>
<td>2D</td>
</tr>
</tbody>
</table>
Chapter 2. Literature Review

The higher layers subsume the functions of lower layers. Adding new functions to the control system can be easily realized through building a new layer on the old levels of competence.

State-based

A set of behaviours is selected that is adequately competent to handle the situation corresponding to some given state. Action selection is done using state transition, where upon detection of a certain event a shift is made to a new state, thus a new action. In Discrete Event Systems [Kosecka 93] and Temporal Sequencing [Arkin 94], the agent and its interaction with the environment are modelled using FSM.

The Discrete Event System in Figure 2.12(a) shows a finite state mobile agent developed by [Yong 05]. The model used fabric architecture, named virtual organization (VO or group), to support the computation. The basic elements of virtual organization are nodes that connect via a network. The virtual group based fabric architecture is the platform of the mobile agent migration. By this method, the mobile agent can explore and move more effectively and it also can greatly decrease the mobile agent size when migration occurs.

Temporal Sequencing alternatively, uses FSM for the formulation of sequencing between a series of behaviours based on perceptual triggers. [Sevin 05] developed a motivational model of action selection based on this method, as shown in Figure 2.12(b). The model is for autonomous virtual humans in which overlapping hierarchical classifier systems, working in parallel to generate coherent behavioural plans, are associated
with the functionality of a free flow hierarchy to give reactivity to the hierarchical system.

A bayesian network analysis [Kristensen 97] showed that the competition of behaviours is the basic characteristic of a behaviour network. Each behaviour can get a higher activation level than other behaviours from forward and backward activation spreading. Among candidate behaviours, the one that has the highest activation level is selected and has control of the robot. The precondition is the sensor that is likely to be true when the behaviour is executed. The add list is a set of conditions that are likely to be true by the execution of the behaviour and the delete list is a set of conditions that are likely to be false by the execution of the behaviour. Figure 2.13 is a typical

Note: S(sensors), B(behaviour), G (goal), the solid line among behaviours represents a predecessor link and the dashed line represents a successor link

Figure 2.13: An Example of a Behaviour Network
example of a behaviour network. [Banerjee 00] presented research to enable bayesian network based modelers to select actions that lead to more accurate models about the nature of another agent. The mechanism involves the use of a max-min procedure for action selection that guarantees a minimum level of improvement in estimation of an agent’s trustworthiness irrespective of whatever action the latter selects.

**Winner-takes-all**

Action selection results from the interaction of a set of distributed behaviours that compete until one behaviour wins the competition and takes control of the agent. There are obvious similarities between the agent network architecture and neural network architectures. Perhaps the key difference is that it is difficult to say what the meaning of a node in a neural network is; it only has a meaning in the context of the network itself. Since competence modules are defined in declarative terms, it is very much easier to say what their meaning is.

Pattie Maes [Maes 89, Maes 91] has developed an agent architecture which is known as an **Activation Network**. The agent is defined as a set of competence modules. Each module is specified by the designer in terms of pre- and post-conditions, and an activation level, which gives a real-valued indication of the relevance of the module in a particular situation. The higher the activation level of a module, the more likely it is that this module will influence the behaviour of the agent. Once specified, a set of competence modules is compiled into a spreading activation network, in which the modules are linked to one-another in ways defined by their pre- and post-conditions. For example, if a module has a post-condition, and a module has a pre-condition of another module, then they are connected by a successor link. Other types of link include predecessor links and conflicted links. When an agent is executing, various modules may become more active in given situations, and may be executed. The result of execution may be a command to an effector unit, or perhaps the increase in activation level of a successor module.

**2.5.3.2 Command Fusion**

In command fusion, the action selection requires mechanisms that achieve some sort of behaviour (or command) fusion, integrating information from different sources to obtain the current action [Scheutz 02]. Command fusion allows multiple behaviours to contribute to the final control of the agent, which means combining recommendations
from multiple behaviours to form a control action that represents their consensus.

**Voting**

Voting interprets the output of each behaviour as votes for or against possible actions. The action with the maximum weighted sum of votes is selected. Action Voting [Hoff 95] is where each behaviour votes for an action which it determines the robot or agent should perform. Action choices are assumed to be mutually exclusive; a single action is selected for one time-step. The behaviour shows its preference for the action with a value in the range of 0 to 1. These base action votes are tallied, modifications are applied (discussed below), and the action with the highest total is selected for execution.

A Distributed Architecture for Mobile Navigation (DAMN) [Rosenblatt 97] is used for command fusion regarding the safety behaviours for turn and speed of the mobile robot. The beauty of the DAMN design is that the deliberative and reactive components of the architecture can operate at the same level and also it is scalable due to lack of hierarchy.

![Diagram](image_url)

**Figure 2.14: Goal-Action-Attribute Model**

Similar to DAMN, [Salehie 07] proposed a weighted voting mechanism which makes decisions based on a Goal-Action-Attribute Model (GAAM). Figure 2.14 illustrates the flow of the proposed decision process. Before making a decision, it is essential to determine which goals have been activated and which actions are feasible to take effect. The activated goals (G) are voters and the feasible actions (A) are eligible candidates. As shown in Figure 2.14, events are detected by the aspiration values
of each goal, \( ac_i \). In GAAM, low-level goals are used which are directly related to the attributes.

**Fuzzy/Multivalued Logic**

This is similar to voting, but uses fuzzy inferencing methods to formalize the voting approach. The result of inferencing is represented in a fuzzy variable and a defuzzify to get a crisp value that can be directly used to control the agent. Two main advantages of using fuzzy logic compared to other methods are that it deals with various situations without needing an analytical model of the environment; and it is easier to merge different strategies by means of the fuzzy rules depending on different situations [Chee 96].

[Vosinakis 07] proposed a fuzzy rule-based mechanism for the low-level decision process of autonomous agents in dynamic environments that operates using vague locations. The proposed architecture is presented in Figure 2.15. All sensor data are stored in the agent’s memory, which contains the known objects and their property values. The agent’s effectors operate using crisp positions. They have an equal number of fuzzy rule sets assigned to them, and they receive crisp input after a complete fuzzification - evaluation - defuzzification loop. Fuzzy rule sets contain condition or action rules that are defined by the designer using vague locations. The condition part of a rule may be a simple or a compound condition.
Superposition

Superposition based command fusion combines behaviour recommendations using linear combinations. *Potential Field* [Khatib 86], *Motor Schemas* [Arkin 89] and *Dynamics System* approaches fall under this method. In a potential field, the motor commands of the agent at any position in a potential field correspond to the vector on which the agent is situated. Goals attract and will have vectors pointing towards them, obstacles repulse and will be surrounded by vectors pointing away. [Katoh 04] for example, used the potential of the environment to give agents some criteria to assess environmental situations from their own perspective. The potential of each object represents its influence on the environment and the environmental potential, i.e., the summation of each object's potential, represents the global situation of the environment. An agent decision regarding their behaviour will be made by refining the policy obtained from the potential.

[Pezzulo 06] presents a schema-based agent architecture which is inspired by an ethological model of the praying mantis. It includes an inner state, perceptual and motor schemas, several routines, a fovea and a motor. The model includes six motor schemas: stay in path (the default behaviour), chase, escape, mate, hide and avoid obstacle. They have three components: a detector, which sets the value of the preconditions by monitoring the state of the perceptual schemas (e.g. detect prey is very active); a controller (an inverse model), which sends commands to the motor (e.g. move left); and a forward model. The motor schemas receive activation from the related perceptual schemas in the form of matched preconditions: a very active detect prey activates chase (which learns to interpret it as: there is prey). The motor schemas also receive activation from the inner states: a fearful mantis activates its motor routines for escaping even in the absence of real danger; as in the case of perceptual routines, they can only remain active if the right stimuli are in place. The main role of the controller is to send commands to the motor. The main role of the forward model is to produce expectations about perceptual stimuli (to be matched with sensed stimuli, including vision and proprioception).

Multi-objective

Each behaviour calculates an objective function over a set of permissible actions. The action that maximizes or minimizes the objective function corresponds to the action which best satisfied the objective. Multiple behaviours are blended into a single com-
plex behaviour that seeks to select the action that simultaneously satisfies all the objectives as closely as possible [Pirjanian 99b]. In general, an agent has a set of behaviours for achieving various objectives, and must integrate these behaviours according to the environmental conditions.

[Pirjanian 00] proposed the method for multi-objective behaviour coordination. Then [Ban 07] used [Pirjanian 00]'s method for solving decisions based on perception for behaviour animation of autonomous agents. First, the internal state of agents was modeled, which was caused by temporary stimuli and accumulation of physical and mental states. Secondly, the agents' desires were described which are generated by the internal state and used for guiding perception. Thirdly, the net value of the possible feature combinations for a given desire was figured out using decision-theoretic principles to determine whether the process on the feature combinations was worthy or not. Then, a multi-objective decision making algorithm was introduced to achieve a decision based on perception, thus the connection between the actual desires and behaviours are established.

![Multi-objective Behaviour Coordination](image)

**Figure 2.16:** Multi-objective Behaviour Coordination [Kubota 07]

[Kubota 07] proposed a multi-objective behaviour coordination to realize formation behaviours based on the integration of the intelligent control from the local viewpoint of individual intelligence and the spring model from the global viewpoint of collective intelligence. This method is composed of a behaviour coordinator and a behaviour weight updater.
2.6 Summary

In this chapter, we have seen that an autonomous agent will make its own decisions and have some degree of autonomy. It can be a situated agent or embodied agent depending on their use and function. Autonomous agents architectures can be reactive, deliberative or hybrid. Although virtual agents use different architectures, they can still be used for solving the same problem in a different way. Another important aspect of autonomous agents is the action selection mechanism: which of the many things an agent can do at any moment is the right thing to do?

We recognise that virtual agents should respond quickly to the environmental changes and manage autonomously the fulfillment of goals. With these requirements fulfilled, virtual agents are highly autonomous and distinct. The next chapter will describe how a fuzzy reactive architecture has been used for autonomous agents. A new fuzzy action selection method has been developed based on the $\alpha$-level ranking method.
Chapter 3

The Methodology

No sensible decision can be made any longer without taking into account not only the world as it is, but the world as it will be. . .

- Isaac Asimov (1920-1992)
  Humanist and a rationalist

Rational behaviour requires theory. Reactive behaviour requires only reflex action.

- W. Edwards Deming (1900-1993)
  American statistician, professor, author, lecturer, and consultant

3.1 Introduction

The behaviour-based control method is based on decomposing the problem of autonomous control by task rather than by function. Behaviour-based control is usually designed to be a reactive system, which maps a perceived situation to an action. However, this simple approach brings up three main problems which are [Li 94, Pérez 00]:

1. it is hard to formulate reactive behaviour quantitatively, and also there might be no applicable approach to coordinating conflict;

2. there is competition among different reactive behaviours to achieve a good performance; and

3. how to select the proper behaviours for robustness and efficiency in accomplishing goals.

Because of the above problems a reliable action selection mechanism is required. The role of the action selection mechanism is to compute which action should be executed
by a behaviour-based system using the internal state and the external perceptions of the virtual agent. Unfortunately, it is difficult to make good decisions that satisfy both goal and constraints. One of the main issues is how to define the required behaviours to accomplish the goal [Saffiotti 98]. This problem appears in the decision process when sensory data matches with several behaviour rules (conditional parts of the rules). As a result, behaviour rules conflict with one another, which means that more than one rule becomes active at one time.

We focus on the development of a reactive/behaviour-based architecture using fuzzy logic. The main advantages of this method are that no mathematical model is required and the ability to represent human expert knowledge on a control plan. The virtual agent will interact with the environment continuously, where action is executed without planning. The action is triggered by reacting to the environment rather than deliberation, or cognitive assessment.

### 3.2 Overview of Fuzzy Logic Approach

In this section, we briefly review the basic concepts of fuzzy sets and fuzzy logic which will be used in describing our fuzzy logic system.

#### 3.2.1 Basics of fuzzy sets

##### 3.2.1.1 Fuzzy sets

In classical set theory a set can be represented by enumerating all its elements using $\mathcal{A} = \{a_1, a_2, a_3, \ldots, a_n\}$. If these elements $a_i (i = 1, \ldots, n)$ of $\mathcal{A}$ are together a subset of the universal base set $X$, the set $\mathcal{A}$ can be represented for all elements $x \in X$ by its characteristic function

$$\mu_{\mathcal{A}}(x) = \begin{cases} 
1 & \text{if } x \in \mathcal{A} \\
0 & \text{otherwise} 
\end{cases} \quad (3.1)$$

In classical set theory $\mu_{\mathcal{A}}(x)$ has only the values 0 (false) and 1 (true), two values of truth. Such sets are also called crisp sets.

Non-crisp sets are called fuzzy sets, for which a characteristic function can be defined. This function is called a membership function. The membership of a fuzzy set is described by this membership function $\mu_{\mathcal{A}}(x)$ of $\mathcal{A}$, which associates to each element $x_0 \in X$ a grade of membership $\mu_{\mathcal{A}}(x_0)$. In contrast to classical set theory a
Chapter 3. The Methodology

A membership function $\mu_A(x_0)$ of a fuzzy set can have in the normalised closed interval $[0, 1]$.

Therefore, each membership function maps elements of a given universal base set $\mathcal{X}$, which is itself a crisp set, into real numbers in $[0, 1]$. The notation for the membership function $\mu_A(x)$ of a fuzzy set $\mathcal{A}$ is used.

$$\mathcal{A} : \mathcal{X} \rightarrow [0, 1]$$

(3.2)

Each fuzzy set is completely and uniquely defined by one particular membership function. Consequently symbols of membership functions are also used as labels of the associated fuzzy sets. That is, each fuzzy set and the associated membership function are denoted by the same capital letter. Since crisp sets and the associated characteristic functions may be viewed, respectively, as special cases of fuzzy sets and membership functions, the same notation is used for crisp sets, as in Figure 3.2:

The base set is introduced above as a universal set. In practical applications, physical or similar quantities are considered that are defined in some interval. When such quantities are described by sets, a base set can be generalised seamlessly to a crisp base set that exists in a defined interval.

3.2.1.2 Elementary operators for fuzzy sets

The basic connective operations in classical set theory are those of intersection, union and complement. These operations on characteristic functions can be generalised to fuzzy sets in more than one way. However, one particular generalisation, which results in operations that are usually referred to us as standard fuzzy set operations, has a
special significance in fuzzy set theory. In the following, only the standard operations are introduced. The following operations can be defined:

- The fuzzy intersection operator \( \cap \) (fuzzy AND connective) applied to two fuzzy sets \( A \) and \( B \) with the membership functions \( \mu_A(x) \) and \( \mu_B(x) \) is

  \[
  \mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}, \quad x \in X
  \] (3.3)

- The fuzzy union operator \( \cup \) (fuzzy OR connective) applied to two fuzzy sets \( A \) and \( B \) with the membership functions \( \mu_A(x) \) and \( \mu_B(x) \) is

  \[
  \mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}, \quad x \in X
  \] (3.4)

- The fuzzy complement (fuzzy NOT operation) applied to two fuzzy sets \( A \) with the membership function \( \mu_A(x) \) is

  \[
  \mu_A^C(x) = 1 - \mu_A(x), \quad x \in X
  \] (3.5)

3.2.1.3 Fuzzy relations

Fuzzy relation \( R \) from set \( X \) to set \( Y \) is a fuzzy set from the direct product \( X \times Y = \{(x,y) | x \in X, y \in Y\} \), and is characterised by a membership function \( \mu_R: \)

\[
\mu_R : X \times Y \rightarrow [0, 1]
\] (3.6)

Note, when \( X = Y \), \( R \) is known as a fuzzy relation on \( X \).

3.2.1.4 Fuzzy composition

If \( R \) is a fuzzy relation in \( X \times Y \) and \( S \) is a fuzzy relation in \( Y \times Z \) the composition of \( R \) and \( S \), \( R \circ S \), is a fuzzy relation in \( X \times Z \) as defined below:

\[
R \circ S \leftrightarrow \mu_{R \circ S}(x,z) = \bigvee_y \{\mu_R(x,y) \land \mu_S(y,z)\}
\] (3.7)

where \( \bigvee = \max \) and \( \land = \min \). This composition uses max and min operations, also known as max-min composition.
3.2.1.5 Fuzzy Implication

There are many possible ways to define a fuzzy implication [Mizumoto 88], but in control applications two common approaches are the Larsen implication and the Mamdani implication [Kovacic 06]. Let \( A \) and \( B \) be fuzzy sets in \( U \) and \( V \). A fuzzy implication, denoted by \( A \rightarrow B \), is a special kind of fuzzy relation in \( U \times V \) with the following membership functions:

- The Mamdani implication:

\[
\mu_R(x, y) = \min \{ \mu_A(x) \cdot \mu_B(y) \} \tag{3.8}
\]

- The Larsen implication:

\[
\mu_R(x, y) = \min \{ 1, [1 - \mu_A(x) + \mu_B(y)] \} \tag{3.9}
\]

3.2.1.6 Membership functions

The membership function \( \mu_A(x) \) describes the membership of the elements \( x \) of the base set \( X \) in the fuzzy set \( A \), whereby for \( \mu_A(x) \) a large class of functions can be taken. Popular functions are often piecewise linear functions, such as triangular or trapezoidal functions.

![Membership Grades](image)

Figure 3.2: Membership Grades of \( x_0 \) in the Sets \( A \) and \( B \)

The grade of membership \( \mu_A(x_0) \) of a membership function \( \mu_A(x) \) describes for the special element \( x = x_0 \), which grade it belongs to in the fuzzy set \( A \). This value is in the unit interval \([0, 1]\). Of course, \( x_0 \) can simultaneously belong to another fuzzy set \( B \), such that \( \mu_B(x_0) \) characterises the grade of membership \( x_0 \) of to \( B \). This case is shown in Figure 3.2.
3.2.1.7 Rule Base

In the previous section, elementary fuzzy terms and fuzzy logic operations have been introduced. In this section, the application to the treatment of rule-based knowledge follows. For this a rule-based fuzzy system is needed, containing a rule base and a reasoning algorithm, which is used to process crisp or fuzzy input values \( x_i, i = 1, 2, \ldots, n \) to a crisp or fuzzy output value \( y \), as in Figure 3.3.

![Figure 3.3: Rule-based Fuzzy System with \( n \) Inputs and One Output](image)

Using multiple inputs and one output implies no restriction as a multi-input-multi-output fuzzy system can always be decomposed into multiple systems. Such systems are the basis for the realisation of fuzzy controllers. As there are mostly crisp input values \( x_i \) from measurements and for controllers only a crisp output \( y \), a fuzzy system must contain additional components, fuzzification and defuzzification.

For example in Figure 3.3, if the rule base for a two-input and one-output controller consists of a finite collection of rules with two antecedents and one consequent of the form as in:

\[
\text{Rule}^i : \text{IF } x_1 \text{ is } A_1^i \text{ AND } x_2 \text{ is } A_2^i \text{ THEN } y^k \text{ is } B^k
\]

where:

- \( k = 1, 2, \ldots, r \)
- \( A_1^i \) and \( A_2^i \) are the fuzzy sets representing the \( k^{th} \) antecedent pairs,
- \( B^k \) is the fuzzy set representing the \( k^{th} \) consequent.

For a given pair of crisp input values \( x_1 \) and \( x_2 \) the antecedents are the degrees of membership obtained during the fuzzification: \( \mu_{A_1^i}(x_1) \) and \( \mu_{A_2^i}(x_2) \). Based on the Mamdani implication in equation (3.8), the strength of the \( \text{Rule}^i \) (i.e its impact on the outcome) is as strong as its weakest component:

\[
\mu_{B^k}(y) = \min[\mu_{A_1^i}(x_1), \mu_{A_2^i}(x_2)]
\]
If more than one activated rule, for instance $Rule^p$ and $Rule^q$, specify the same output action, (e.g. $y$ is $B^k$), then the strongest rule will prevail:

$$\mu_{B^k}(y) = \max \left\{ \min[\mu_{A_{11}^p}(x_1), \mu_{A_{12}^p}(x_2)], \min[\mu_{A_{11}^q}(x_1), \mu_{A_{12}^q}(x_2)] \right\}$$  \hspace{1cm} (3.12)

Figure 3.4: Mamdani Implication with Crisp Inputs

Figure 3.4 shows a simple interpretation of equation (3.12). The figure illustrates the analysis of two rules, where the symbols $A_{11}$ and $A_{12}$ refer to first and second fuzzy antecedent of the first rule, respectively, and the symbol $B_1$ refers to the fuzzy consequent of the first rule. The symbols $A_{21}$ and $A_{22}$ refer to first and second fuzzy antecedent of the second rule, respectively, and the symbol $B_2$ refers to the fuzzy consequent of the second rule. The minimum function in equation (3.12) is illustrated in Figure 3.4 and arises because the antecedents pair given in the general rule structure for this system are connected by a logical AND connective as in equation (3.10). The minimum membership value for the antecedents propagates through to the consequent and truncates the membership function for the consequent of each rule.

The truncated membership functions for each rule are aggregated using a disjunctive rule as follows:
Chapter 3. The Methodology

\[ \mu_y(y) = \max (\mu_{y1}(y), \mu_{y2}(y), \ldots, \mu_{yr}(y)) \quad \text{for } y \in Y \]  

(3.13)

So the aggregation operation max results in an aggregated membership comprised of the outer envelope of the individual truncated membership form from each rule.

### 3.2.2 Fuzzy Systems

The Fuzzy Inference System (FIS) is defined as a process of mapping from a given input to an output, using the theory of fuzzy sets. Fuzzy rules are linguistic IF – THEN constructions that have the general form as follows:

\[ \text{IF } x \text{ is } A \text{ AND } y \text{ is } B \text{ THEN } z \text{ is } C \]  

(3.14)

where \( x, y \) and \( z \) are linguistic variables for the inputs and outputs of the fuzzy controller and \( A, B \) and \( C \) are the terms of the variables \( X, Y \) and \( Z \).

There are specific components characteristic of a fuzzy controller to support a design procedure. In the block diagram in Figure 3.5, the controller is between a input and a output.

![Figure 3.5: Basic Fuzzy Controller](image)

Most commercial fuzzy products are rule-based systems that receive current information in the feedback loop from the device as it operates and control the operation of a mechanical or other device [Ross 04]. Crisp input information from the device is converted into fuzzy values for each input fuzzy set with the fuzzification block. The universe of discourse of the input variables determines the required scaling for correct per-unit operation. The scaling is very important because the fuzzy system can be retrofitted with other devices or ranges of operation by just changing the scaling.
Chapter 3. The Methodology

of the input and output. The decision-making-logic determines how the fuzzy logic operations are performed, and together with the knowledge base determine the outputs of each fuzzy IF-THEN rule. Those are combined and converted to crispy values with the defuzzification block. The output crisp value can be calculated by the center of gravity or the weighted average.

3.3 Modeling of the Control System

A Fuzzy Associative Memory (FAM) is used as a process of encoding and mapping the input fuzzy sets to the output fuzzy set [Kosko 92]. Consider a set of fuzzy rules, \( R = \{ R_1, R_2, \ldots, R_i, \ldots, R_k \} \), where \( R_m \) is the \( m^{th} \) rule of the fuzzy controller. The rule \( R_m \) is given as follows:

\[
\text{IF } X_1 \text{ is } A_1^m \text{ AND } X_2 \text{ is } A_2^m \text{ AND} \ldots \text{ AND } X_n \text{ is } A_n^m \text{ THEN } Z \text{ is } C_n^m
\] (3.15)

The following fuzzy relation will implement \( R_i \):

\[
R_m(X_1, X_2, \ldots, X_n, Z) = (A_1^m \times A_2^m \times \ldots \times A_n^m \rightarrow C_n^m)(X_1, X_2, \ldots, X_n, Z)
\] (3.16)

We can rewrite equation (3.16) as below:

\[
R_m(X_1, X_2, \ldots, X_n, Z) = [A_1^m(X_1) \land A_2^m(X_2) \land \ldots \land A_n^m(X_n)] \rightarrow C_n^m(Z)
\] (3.17)

where \( X_1, X_2, \ldots, X_n \) are input variables which are the sensor data of the virtual agent, \( A_1^m, A_2^m, \ldots, A_n^m \) are the input fuzzy sets, \( C_n^m \) is the output fuzzy set, \( Z \) is the output variable, \( n \) is the dimension of the input vector.

In order to create an \( n \) fuzzy input vector \( \overline{X} = \{ \overline{X}_1, \overline{X}_2, \ldots, \overline{X}_n \} \), the system needs to compose the input vector \( \overline{X} \) with the calculated fuzzy relation \( R_m \) to produce the following output \( C_m^i \). i.e.,

\[
C_i^i = (\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_n) \circ R_m
\] (3.18)

where \( \overline{X}_n \) is the fuzzified crisp value of \( X_n \) into a fuzzy output class \( C_m^i(Z) \). The output of the \( m^{th} \) rule \( C_m^i(Z) \) is defined as:

\[
C_m^i(Z) = [A_1^m(X_1) \land A_2^m(X_2) \land \ldots \land A_n^m(X_n)] \Rightarrow C_m^i(Z)
\] (3.19)
The weighted sum $C$ for each individual membership can be defined by using minmax aggregation [Ross 04] operators as given below:

$$C = \sum_{m=1}^{k} U_m C_m$$

$$= \sum_{m=1}^{k} U \left( \left[ A_1^n (X_1) \land A_2^n (X_2) \land \ldots \land A_n^n (X_n^m) \right] \Rightarrow C_m (Z) \right)$$

(3.20)

where the non-negative weight $U_m$ summarises the strength of the $m^{th}$ FAM rule for the $m^{th}$ FAM entry and $n \times m$ is the number of rules in the system. In order to relate the $n^{th}$ fuzzy set of the $m^{th}$ fuzzy rule, the fuzzy implication model using the Mamdani min operator [Wang 97] interprets the logical rules for rule firing. Combining the equations (3.16) and (3.20), we obtain:

$$\mu_{R_m} (X_1, X_2, \ldots, X_n, Z) = \min_{m=1}^{n} \left[ \mu_{A^n_m} (X_n) \right]$$

(3.21)

Then, the final defuzzification response for a $k$ output membership function $\mu_C (z)$ is defined as:

$$\mu_C (z) = \max_{m=1}^{k} \left[ \min_{X=U}^{n} \left[ \mu_{A^n_m} (X_n), \mu_{R_m} (X_1, X_2, \ldots, X_n, Z) \right] \right]$$

(3.22)

Equations (3.17) and (3.22) are used to derive the FAM model and the output fuzzy system, respectively.

### 3.3.1 Behaviour Conflicts and $\alpha$–level Thresholds

The $\alpha$–level fuzzy logic methodology is established and used to resolve the behaviour conflicts. $\alpha$–level [Nguyen 99] thresholds control the behaviour rules that are fired during navigation. An $\alpha$–level makes all the truth membership functions between the threshold intervals to be true and the remaining values to be zero. When an $\alpha$–level threshold is applied to the truth of a rule’s predicate, it determines whether or not the truth is sufficient to fire the rule. When an $\alpha$–level threshold is applied to a fuzzy set, it establishes a line through the truth membership function, which gives a value between $\alpha$ intervals. Truth membership values below and above the intervals are considered to be equal to zero. An $\alpha$–level threshold fuzzy set is defined as the crisp set of elements which belong to a fuzzy set $A$ at least to the degree of $\alpha$. The $\alpha$ fuzzy set $A$ is defined mathematically as below:
Chapter 3. The Methodology

\[ A_\alpha = \{ x | \mu_A(x) > \alpha \}; \quad \alpha \in [0, 1] \] (3.23)

\[ A_\delta = \{ x | \mu_A(x) \geq \alpha \}; \quad \alpha \in [0, 1] \] (3.24)

where \( A_\alpha \) and \( A_\delta \) are the crisp fuzzy sets, \( x \) is the linguistic variable of the universal set \( U \), and \( \mu_A \) is the membership function of the sets. In order to illustrate the Fuzzy Inference System (FIS) in conjunction with the \( \alpha \)-level fuzzy set, the following rule table (Table 3.1) is formulated with the number of rules \((n \times m)\), which are the products of the number of terms of each input linguistic variable \( A_1 \) and \( A_2 \). The rules with the possible fuzzy outputs labeled \( C_{ij} \) are presented symbolically in Table 3.1.

Table 3.1: Rule Table: IF \(-\) THEN rules

<table>
<thead>
<tr>
<th>( C = \text{output} )</th>
<th>( A_2_1 )</th>
<th>( \cdots )</th>
<th>( A_2_j )</th>
<th>( A_2_{j+1} )</th>
<th>( \cdots )</th>
<th>( A_2_m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_1_1 )</td>
<td>( C_{1,1} )</td>
<td>( \cdots )</td>
<td>( C_{1,j} )</td>
<td>( C_{1,j+1} )</td>
<td>( \cdots )</td>
<td>( C_{1,m} )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_1_i )</td>
<td>( C_{i,1} )</td>
<td>( \cdots )</td>
<td>( C_{i,j} )</td>
<td>( C_{i,j+1} )</td>
<td>( \cdots )</td>
<td>( C_{i,m} )</td>
</tr>
<tr>
<td>( A_1_{i+1} )</td>
<td>( C_{i+1,1} )</td>
<td>( \cdots )</td>
<td>( C_{i+1,j} )</td>
<td>( C_{i+1,j+1} )</td>
<td>( \cdots )</td>
<td>( C_{i+1,m} )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_1_n )</td>
<td>( C_{n,1} )</td>
<td>( \cdots )</td>
<td>( C_{n,j} )</td>
<td>( C_{n,j+1} )</td>
<td>( \cdots )</td>
<td>( C_{n,m} )</td>
</tr>
</tbody>
</table>

3.3.2 Rules Evaluation

The measurement values of input parameters obtained from the sensors have to be translated to the corresponding linguistic variables. Normally any reading has a crisp value, which has to be matched against the appropriate membership function representing the linguistic variable. The matching is necessary because of the overlapping of terms as shown in Figures 3.6(a) and (b), and this matching is known as fuzzification.

In these figures, the reading \( x_0 \in U_1 \) corresponds to two values \( \mu_{A_1}(x_0) \) and \( \mu_{A_1+i}(x_0) \) which are called fuzzy inputs. They can be interpreted as the truth values of \( x_0 \) related to \( A_i \) and to \( A_{i+1} \), respectively. In the same way, the fuzzy inputs are obtained corresponding to the reading \( y_0 \in U_2 \) also. In both the figures, only a few terms of the fuzzy sets \( A_1 \) and \( A_2 \) are presented.

The straight line passing through \( x_0 \) parallel to the \( \mu \) axis intersects only the terms \( A_{1_i} \) and \( A_{1_{i+1}} \) of \( A_1 \) giving the result denoted as shown below:

\[ A_{1_i}(x_0) = \{ \mu_{A_{1_i}}(x_0), \mu_{A_{1_{i+1}}}(x_0) \} \] (3.25)
Chapter 3. The Methodology

Figure 3.6: Fuzzy Input Corresponding to $x_0$ and $y_0$

Similarly, the line passing through $y_0$ intersects only the terms $A_{2i}$ and $A_{2i+1}$ of $A_2$, giving the crisp values as shown below:

$$A_{2i}(x_0) = \{\mu_{A_{2i}}(x_0), \mu_{A_{2i+1}}(x_0)\} \quad (3.26)$$

The active rules, which are shown in Table 3.2, are from Table 3.1. Four cells in Table 3.2 contain nonzero terms. These cells are called active cells. Table 3.2 shows that only four rules are active. The rest of the rules will not produce any output.

Table 3.2: Decision Table with an Active Cell

<table>
<thead>
<tr>
<th>Sensor $S_3$</th>
<th>$\mu_{A_{2i-1}}(y_0)$</th>
<th>$\mu_{A_{2i}}(y_0)$</th>
<th>$\mu_{A_{2i+1}}(y_0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor $S_4$</td>
<td>$\mu_{A_{1i}}(x_0)$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$\mu_{A_{1i+1}}(x_0)$</td>
<td>0</td>
<td>$\mu_{C_{ij}}(z)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\mu_{C_{i+1j}}(z)$</td>
</tr>
</tbody>
</table>

The process of conflict resolution is illustrated by using these four rules, $R_1$ to $R_4$, and thus are given below:

$$R_1: \text{IF } x \text{ is } A_{1i}(x_0) \text{ AND } y \text{ is } A_{2j}(y_0) \text{ THEN } z \text{ is } C_{ij}$$

$$R_2: \text{IF } x \text{ is } A_{1i}(x_0) \text{ AND } y \text{ is } A_{2j+1}(y_0) \text{ THEN } z \text{ is } C_{ij+1} \quad (3.27)$$

$$R_3: \text{IF } x \text{ is } A_{1i+1}(x_0) \text{ AND } y \text{ is } A_{2j}(y_0) \text{ THEN } z \text{ is } C_{i+1j}$$

$$R_4: \text{IF } x \text{ is } A_{1i+1}(x_0) \text{ AND } y \text{ is } A_{2j+1}(y_0) \text{ THEN } z \text{ is } C_{i+1j+1}$$

The $THEN$ part of each rule is called the strength of the rule. The strengths $\alpha_{ij}$ of the rules are obtained as shown below:
The numbers expressing the strength of the rules are output fuzzy sets of Table 3.2. The Control Output (CO) of each rule is defined by the conjunction operation applied based on the rule strength. They are:

\[
CO_1 = \min(\alpha_{ij}, \mu_C(z)) \\
CO_2 = \min(\alpha_{i,j+1}, \mu_{C_{i+1,j}}(z)) \\
CO_3 = \min(\alpha_{i+1,j}, \mu_{C_{i+1,j}}(z)) \\
CO_4 = \min(\alpha_{i+1,j+1}, \mu_{C_{i+1,j+1}}(z))
\]  

(3.29)

This is equivalent to performing a \(\min\) operation on the corresponding elements in the active cells. The output of the four rules in equation (3.29), have to be combined or aggregated in order to produce one control output with a membership function, namely \(\mu_{\text{agg}}(z)\), as shown below:

\[
\mu_{\text{agg}}(z) = (\alpha_{ij} \land \mu_{C_{ij}}(z)) \lor (\alpha_{i,j+1} \land \mu_{C_{i+1,j}}(z)) \\
\lor (\alpha_{i+1,j} \land \mu_{C_{i+1,j}}(z)) \lor (\alpha_{i+1,j+1} \land \mu_{C_{i+1,j+1}}(z))
\]  

(3.30)

In the equations (3.29) and (3.30), the operation \(\land\) (\(\min\)) is performed on a number and a membership function of a fuzzy set. In this context, the real number \(\alpha\) and the output fuzzy set \(C\) with membership function \(\alpha C(\mu_C)\) can be obtained as shown below:

\[
\mu_{\alpha}(z) \land \mu_C(z) = \min(\mu_{\alpha}(z), \mu_C(z))
\]  

(3.31)

Figure 3.7: Trapezoidal Fuzzy Numbers

The final output as shown in Figure 3.7, uses a trapezoidal shape. It represents a
non-normalized clipped fuzzy number. The final output of the fired rule is obtained using equation (3.31).

### 3.4 Action Selection Method

In the decision-making process, multiple conflicting objectives should be considered simultaneously, subject to certain constraints dictated by the virtual agent limitations [Pirjanian 97]. A major issue in the design of systems for controlling an autonomous virtual agent is the formulation of an effective mechanism for the combination of multiple objectives into strategies for rational and coherent behaviour [Pirjanian 97]. For example, given a set of actions, \( X = \{x_1, x_2, \ldots, x_n\} \), the virtual agent has to decide which is the most appropriate or the most relevant next action to take at a particular moment, when facing a particular situation [Maes 89].

The fuzzy action selection method is inspired by the ranking method of [Huang 89], [Mabuchi 88] and [Yuan 91] and uses \( \alpha - level \) and fuzzy subtraction operations to calculate the area of a new fuzzy number, which is produced by the comparison of two fuzzy numbers. If there are \( m \) fuzzy numbers, then \( m(m-1)/2 \) pairs of fuzzy numbers must be compared to determine overall rank. Our proposed method will reduce the redundancy of calculating \( m(m-1)/2 \) pairwise comparisons to \( m \) pairwise comparisons by the fuzzy subtraction operation.

In general, when comparing \( m \) different fuzzy numbers produced by each behaviour the height and common maximizing and minimizing barriers are used. Let \( \mu_{\tilde{X}}(x) \) be the membership function of a fuzzy number, \( \tilde{X} \) (behaviour output), defined on \( R \). Unlike convexity, assumptions about the normality of \( \mu_{\tilde{X}}(x) \) are made.

In Figure 3.8, an arbitrary, bounded fuzzy number \( \tilde{X} \) has given. Suppose \( \tilde{X}_{\alpha_0}, \tilde{X}_{\alpha_1}, \ldots, \tilde{X}_{\alpha_n} \) are the \( \alpha - level \) interval numbers of \( \tilde{X} \) and they have the following properties:

1. \( \tilde{X}_{\alpha_i} = [l_i, r_i], i = 0, 1, \ldots, n \), where \( l_i \) is the left spread of \( \tilde{X}_{\alpha_i} \), and \( r_i \) is the right spread of \( \tilde{X}_{\alpha_i} \).

2. \( \alpha_0 = 0, \alpha_n = 1 \) and \( \alpha_0, \alpha_1, \ldots, \alpha_n \) is strictly increasing sequence.

3. The distance between each two adjacent \( \alpha - level \) values are equal, i.e. \( \alpha_i - \alpha_{i-1} = \alpha_i - \alpha_{k-1} \), \( \forall i, k \in \{1, 2, \ldots, n\} \).

Based on [Choobineh 93], the loci of the left or right spreads and the maximum and minimum barriers of the \( \alpha - cut \) of the fuzzy number, \( \tilde{X} \), are \( \mu_{L_{\alpha}}(x) \) and \( \mu_{R_{\alpha}}(x) \),
Chapter 3. The Methodology

56

\[ f(x) = h \]

---

Figure 3.8: Trapazoidal Fuzzy Number

\[ 0 \leq \alpha \leq h \tilde{X}, \text{ respectively, where } h \tilde{X} \text{ is the height. If } \tilde{X}_\alpha \text{ is denumerable or connected, then:} \]

\[
\begin{align*}
\mu^L_{\tilde{X}_\alpha}(x) &= \min_{\alpha} \left\{ x | x \in \tilde{X}_\alpha \right\}, 0 \leq \alpha \leq h \tilde{X}, \text{ and} \\
\mu^R_{\tilde{X}_\alpha}(x) &= \max_{\alpha} \left\{ x | x \in \tilde{X}_\alpha \right\}, 0 \leq \alpha \leq h \tilde{X},
\end{align*}
\]

(3.32)

In addition the maximixing and minimizing barriers can be defined as:

- The crisp maximizing barrier, \( U_{\tilde{X}} \), of the membership function for the fuzzy number \( \tilde{X} \) is defined as \( \mu_{U_{\tilde{X}}}(x) = \frac{h \tilde{X}}{d^*} \), where \( \max_{\alpha} \left\{ \mu^R_{\tilde{X}_\alpha}(x) \right\} = d^* \leq d \leq \infty. \)

- The crisp minimizing barrier, \( L_{\tilde{X}} \), of the membership function for the fuzzy number \( \tilde{X} \) is defined as \( \mu_{L_{\tilde{X}}}(x) = \frac{h \tilde{X}}{c^*} \), where \( 0 \leq \alpha \leq \alpha^* = \min_{\alpha} \left\{ \mu^L_{\tilde{X}_\alpha}(x) \right\}. \)

The height, maximizing and minimizing barriers are set to:

\[
\begin{align*}
h_{\tilde{X}}(x) &= \max_i \left\{ \mu_{\tilde{X}}(x) i = 1, 2, \ldots, m \right\}, \\
c &= \min_{\alpha} \left\{ \mu^L_{\tilde{X}_\alpha}(x) i = 1, 2, \ldots, m; 0 \leq \alpha \leq h_{\tilde{X}} \right\}, \quad (3.33) \\
d &= \min_{\alpha} \left\{ \mu^R_{\tilde{X}_\alpha}(x) i = 1, 2, \ldots, m; 0 \leq \alpha \leq h_{\tilde{X}} \right\}.
\end{align*}
\]

Based on equation (3.33), \( h_{\tilde{X}}(x) \) is the maximum value of the height of all \( m \) fuzzy numbers. The variables \( c \) and \( d \) are at the minimum value of the left spread and the minimum right spread of all fuzzy numbers, respectively. To simplify the fuzzy subtraction between the fuzzy number \( \tilde{X} \) and referential rectangle \( \tilde{R} \), at \( \alpha_i \) level, interval
subtraction is used:

\[
\bar{x}_\alpha (-) \bar{R} = [l_i, r_i] [-] [c, d] = [l_i - d, r_i - c], i = 1, 2, \ldots, n
\] (3.34)

then, the behaviour weight, \(\omega\) of equation (3.34) becomes:

\[
\omega \left( \bar{x}_i, \bar{R} \right) = \frac{\sum_{i=0}^{n} (r_i - c)}{\sum_{i=0}^{n} (r_i - c) - \sum_{i=0}^{n} (l_i - d)}
\] (3.35)

where \(n\) is the number of \(\alpha\) levels. \(n\) approaches to \(\infty\), the summation becomes the area measurement.

In equation (3.35), \(\sum_{i=0}^{n} (r_i - c)\) is a positive value and \(\sum_{i=0}^{n} (l_i - d)\) is a negative value. Here, the denominator represents the total area as \(n\) approaches \(\infty\). In addition, if all of the aggregated fuzzy numbers are normal and within the unit interval, then \(h_x = 1, c = 0, d = 1\), and equation (3.35) becomes:

\[
\omega \left( \bar{x}_i, \bar{R} \right) = \frac{\sum_{i=0}^{n} r_i}{\sum_{i=0}^{n} r_i - \sum_{i=0}^{n} (l_i - 1)}, \text{ and } n = \infty
\] (3.36)

In our case, the behaviour weight value \(\omega\) from equation (3.36) will be used. For every \(\omega\), we use the Hurwicz criterion, which selects the lowest value from each behaviour as \(\delta_1\); and then selects the highest value from each behaviour as \(\delta_2\). The index of optimism [Chen 97], \(\sigma\), is used to represent the level of optimism of the virtual environment.

When selecting one particular action from a range of possible actions, the selection is based on the Hurwicz criterion [Choobineh 93] which is defined as:

\[
\eta = \sigma \cdot (\min_{i=0}^{n} \omega_{ij}) + (1 - \sigma) \cdot (\max_{i=0}^{n} \omega_{ij})
\] (3.37)

where \(\eta = \begin{cases} 
\sigma = 0 \rightarrow \text{max-min criterion} \\
0 < \sigma < 1 \rightarrow \text{compromise opinion} \\
\sigma = 1 \rightarrow \text{max-max criterion}
\end{cases}\) (3.38)

Based on the above discussion, the following algorithm has been developed. Let
and uses the summation of each $\alpha$-level interval which does not require normalization to measure the summation for the ranking order of the fuzzy numbers. The behaviour rules containing $\alpha$ intervals of inputs and output spaces are easily integrated with a virtual agent.
Chapter 4

Virtual Agent Navigation

A young sailor boy came to see me today. It pleases me to have these lads seek me on their return from their first voyage, and tell me how much they have learned about navigation.

-Maria Mitchell (1818-1889)
American astronomer

4.1 Introduction

Navigation is the process where people control their movement using environment cues and artificial aids such as maps so that they can achieve their goal without getting lost [Darken 93]. Navigation in a virtual environment can be a difficult task, particularly in unfamiliar environments. People have severe problems in navigation in unknown environments. However, current implementations of virtual environments provide little support for effective navigation.

Research work on navigation in virtual environments can be classified into two main categories [Salomon 03]:

1. Understanding the cognitive principles of navigation.

   These focus on human factor or body centered interaction methods by evaluating various interaction techniques [Usoh 99]. For example, using navigation aids (visual, sound or character) to provide feedback to the user such as a virtual map [Grammenos 02], 3D location pointing [Chittaro 04] and perceptual interface techniques [Konrad 04].
2. Developing navigation techniques for a specific task and application.

These focus on path planning, motion planning and autonomous navigation. Examples are, behavioural animation [Reynolds 99] and using AI techniques such as a neural network [Lozano 02]. Unfortunately, most of the work focuses on knowledge and abilities required by the user and comprise real world navigation to navigate in virtual environments [Van Dijk 03]. For this reason we focus on developing navigation techniques for behaviour animation of virtual agents as one of our contributions to this area.

Autonomous virtual agent navigation in a virtual environment can be described as the ability of a virtual agent to move purposefully without user intervention. The navigation task may be decomposed into three sub-tasks: mapping and modeling the environment; path planning and selection; path following and collision avoidance [Wan 03]. Virtual agent navigation can occur in known and unknown environments. For a known environment, the virtual agent will have knowledge about the environment and can generate the navigation path. The methods used are based on optimization and computational intelligence. In contrast, in an unknown environment in which the virtual agent does not have any knowledge about the environment, the navigation path is generated according to user specifications and the virtual agent cannot be prepared ahead of time [Li 99].

4.2 Related Work

The basic problem of navigation is moving from one place to another by the coordination of planning, sensing and control. The challenge is generating an optimal traversing sequence through the user-specified locations of interests and computation of a collision free path. This task comprises of [Crowley 84, Noser 95]:

1. local navigation - use direct input from the environment to reach the goals or sub-goals of global navigation and to avoid unexpected obstacles; and

2. global navigation - use a pre-learned model of the domain which may be a somewhat simplified description of the virtual environment and might not reflect changes in the environment.

Figure 4.1 illustrates an example of a path traversing through all user-specified locations [Li 99]. In order to navigate in an unknown environment, a virtual agent needs to deal with the environment in a timely manner.
Steering rules are a special reactive technique often used for some of the navigation problems, primarily those concerning flocks or herds of virtual agents. It is based on superposition of attractive and repulsive forces that effect the virtual agent. Steering is based on the original work of [Reynolds 87]. By means of steering, one can achieve a simple form of:

- navigating towards a goal,
- obstacle avoidance behaviour,
- wall or path following behaviour,
- fleeing enemies and avoiding predators, and
- coordinated behaviour (non-interference) by crowds.

The advantage of steering is that it is computationally very efficient. In computer games, hundreds of soldiers can be driven by this technique. In cases of more complicated terrain (e.g. a closed-space in a building), however, steering must be combined with path-finding, which is a form of planning.

The use of personal virtual agents such as navigating in a virtual environment, have many common characteristics with agent navigation in real worlds. However, there is some limitation on the performance variety of the task for example achieving a lifelike virtual agent. [Chittaro 03, Van Dijk 03] have used animated characters to guide visitors through automatically generated tours in a 3D virtual world. For dynamic virtual environments, [Yan 03] proposed a three level control model. Level-0 and Level-1 are for collision avoidance and path planning. Level-2 is an expert system for intelligent
navigation. This has shown an improvement, by using subsumption architecture and adding a global planning ability.

Other approaches such as discrete grid based [Bandi 00], central path computation [Chaudhuri 04] and roadmap with tactical information approaches [Rook 05] have been used for collisions free path planning. For example [Stilman 04] in Figure 4.2 studied navigation among static and movable obstacles. The planner takes advantage of the navigational structure through state-space decomposition and a heuristic search. The planning complexity is reduced to the difficulty of the specific navigation task, rather than the dimensionality of the multi-object domain.

Figure 4.2: Path Generated by Initial Position and Final Heuristic Plan [Stilman 04].

Inspired by studies in human behaviour, [Lamarche 04] proposed a general model to simulate the navigation process inside indoor and outdoor environments. This model is composed of four parts:

1. a spatial subdivision algorithm detecting bottlenecks inside the environment;
2. a hierarchical path planning algorithm based on the abstraction and generalization of topological properties extracted from the spatial subdivision;
3. an efficient structure computing neighbourhood relations between entities; and
4. a general and modular algorithm which handles reactive navigation and includes visual optimization of the trajectory and collision avoidance. The human behaviour is configured through complementary modules describing rules inspired by psychological studies.

Techniques such as set hierarchy, regular graph, artificial potential field and corner graph have been used but are only suitable for 2D environments. One of the reasons is those algorithms require high computational resource in 3D environments. For a
3D environment, navigation mesh and waypoint graph techniques are very popular. A navigation mesh technique is a representation that covers the walkable surface of the world with convex polygons [Tozour 03]. Waypoint is a set of points in the 3D environment with reachability links between them [Elusive 98], where we can place a waypoint at any point in 3D space. The disadvantages of these two techniques are large memory usage, and they require a powerful processor. Even though some of these techniques have been used in computer games, it is still not clear that these approaches have been used in autonomous navigation in virtual environments [Salomon 03].

Artificial intelligence techniques, for example neural networks [Lozano 02], genetic algorithms [Velagic 06] and reinforcement learning [Seo 00] have been used. [Wang 02] presented a multi-agent based evolutionary artificial neural network (ANN) for general navigation. The virtual creature explores unknown environments as far as possible with obstacle avoidance. Through constant interaction with the environment, the virtual agent systems co-decide and consult with each other for the move decision. [Lozano 02] have integrated attention and navigation skills in a 3D virtual agent. They divided their neural model into two main phases. First of all, the environment categorization phase, online pattern recognition and categorization of the virtual agent current input sensor data is carried out by an adaptive resonance driven self organizing neural network. Then, the model must learn how and when to map the current short term memory state into navigation and the attention of the virtual agent. However the majority of 3D virtual agents focus on low cost global techniques to solve navigation problems and attention is less frequently considered in virtual worlds.

The reactive virtual agent [Piaggio 97] is capable of carrying out autonomous navigation. The virtual agent extends the artificial potential field approach, used for trajectory formation, to environment exploration and symbolic feature detection. The virtual agent’s capabilities range from obstacle avoidance to maze navigation, carried out autonomously or under the supervision of higher cognitive levels. Other methods by [Salomon 03] have been used in a known environment. On the other hand, in an unknown environment, methods such as sensor based control in [Wan 03] use Adaptive Dynamic Points of Visibility (ADPV) for moving virtual agents in dynamical unconfigured environments.

A fuzzy logic system is one of the potential ways to solve problems in autonomous navigation in virtual environments. However, it is difficult to maintain the correctness, consistency and completeness of a fuzzy rule base constructed and tuned by a human expert [Cang 03]. There is argument between researchers using fuzzy logic and other
techniques, such as neural networks or genetic algorithms. All these techniques try to achieve the same goal, but with different approaches.

4.2.1 Fuzzy Logic

Fuzzy logic has been utilized in navigation systems for mobile robots for over a decade. Early in 1991, Yen and Pfluger [Yen 91] proposed a method of path planning and execution using fuzzy logic for mobile robot control. The two main advantages in using a fuzzy logic approach are the ability to express partial and concurrent activations of behaviours, and the smooth transitions between behaviours [Saffiotti 97]. From that time, the advantage of using fuzzy logic in mobile robot navigation systems has been demonstrated and several new solutions for navigation problems in unknown environments have been proposed such as [Aguirre 00, Anmin 04, Velagic 06].

Researchers such as [Wang 04] have proposed a generalized framework for a behaviour-based navigation strategy for autonomous robotics which is independent of any specific robotic development platform. [Tunstel 97] have used hierarchical fuzzy behaviour control for an autonomous mobile robot. [Hercok 99] used a multi-layer control system for two co-operating mobile robots, which uses fuzzy logic to adapt the relative importance of a set of reactive behaviours.

There also exist methods combining fuzzy logic with other algorithms. For example, [Chronis 99] used a fuzzy genetic algorithm as an agent learning mechanism for mobile robot navigation. A neural network as learning algorithm by [Zhu 05] is used to tune the parameters of membership functions, which smooth the trajectory generated by the fuzzy logic system. [Cang 03] also used fuzzy logic with supervised learning assisted by a reinforcement learning algorithm for obstacle avoidance in unknown environments. Other algorithms have also been used with fuzzy logic such as potential fields [Huh 02], roadmap [Ma 04], multi-objective [Nojima 03], Dempster-Shafer [Kim 02] and the Agoraphilic algorithm [Ibrahim 01].

Although there are successful implementations of fuzzy logic in robot navigation, the technique has not commonly been used in virtual environments. In order to maximize the fuzzy logic concept, [Walker 00] used a cell decomposition strategy. This strategy has similar characteristics to those found in fuzzy systems. As a result, fuzzy logic improves the description of the virtual agent's environment. Fuzzy logic does not modify a virtual agent's environment, but instead describes the environment in greater detail. This allows virtual agents to navigate better among obstacles. [Gatzoulis 04]
have developed intelligent virtual agents, with the intention of learning and enhancing their task performance in assisting humans in housekeeping. The learning systems are incorporated into the decision-making process of the Virtual Robot Servant to allow it to understand and evaluate the fuzzy value requirements and enhance its performance. In agent-based game design, fuzzy logic has provided a natural way of modelling the games creatures with high flexibility and low coupling allowing verity of behaviour. This has improved the quality of the interaction [Yifan 04].

Fuzzy reinforcement learning has also been used to represent vague goals as well as uncertain environments [Seo 00]. This has been done by defining the fuzzy reinforcement function using the fuzzy goal and the fuzzy state with extended fuzzyQ-learning. The results show that with fuzzy reinforcement the agent learned faster than with Q-Learning. The limitations of this method are the requirement of exacting analysis to generate the proper fuzzy sets to the vague goal and the uncertain environments. Moreover, the intelligent virtual agent should be able to explore and adapt to dynamic environments and distributed environments.

Fuzzy Cognitive Maps (FCMs) can structure virtual worlds that change with time. A FCM links causal events, virtual agents (actors), values, goals, and trends in a fuzzy feedback dynamical system [Dickerson 96]. A FCM lists the fuzzy rules or causal flow paths that relate events. It can guide a virtual agent in a virtual world as the virtual agent moves through a web of cause and effect and reacts to events and to other actors.

4.2.2 Behaviour-based Architecture

Since the first behaviour-based architecture was proposed by [Brooks 86], much work has been done to improve the architecture. Two main issues in behaviour coordination for virtual agents are how to decide which behaviour should be active at each moment and how to combine the results from different behaviours into one command to the virtual agent [Saffiotti 98]. To overcome these problems, research such as [Yen 99, Chrysanthakopoulos 04] proposed behaviour-based architectures that had three major features which need to be implemented [Kaelbling 86]:

- **Modularity** - the controller should be developed incrementally from small components that are easy to implement and understand;

- **Awareness** - the system should be able to react to unexpected sensory data; and
- **Robustness** - the system should be able to continue to behave plausibly in novel situations and when one of its sensors is not working or impaired.

Behaviour coordination for a virtual agent can be divided into two categories: arbitration and command fusion schemes. Both comprise of two paradigms in designing distributed systems: hierarchical (top-down) approaches and non-hierarchical (bottom-up) approaches [Mataric 01]. In hierarchical approaches, a set of behaviours at the lowest level are activated using prior knowledge of the system and ensures goal-directed decision-making. On the other hand, in non-hierarchical approaches, all behaviours are concurrently active eliminating the requirement of prior knowledge of the system which makes the system more reactive.

Behaviour arbitration allowed one behaviour or a set of behaviours at the same time to take control for a period of time until another set of behaviours is activated [Jaafar 07c]. Arbitration mechanisms are suitable for competitive behaviours, but unfortunately always have problems of instability and starvation. Instability arises when the control of the virtual agent/robot alternates between two behaviours and starvation occurs when a behaviour does not gain control of the virtual agent for a long period of time [Huq 07].

One of the earliest works on a fuzzy behaviour arbitration navigation system was the Fuzzy Behaviourist Approach (FBA) [Pin 94, Pin 96]. Fixed arbitration schema based on suspension and inhabitation mechanisms were used based on subsumption architectures. This approach is based on the representation of the system's uncertainties using Fuzzy Set Theory based approximations and on the representation of the reasoning and control schemes as sets of elemental behaviours. The system also checks for completeness of the rule base and for non-redundancy of the rules (which has traditionally been a major hurdle in rule base development). [Hombal 00] used active perception layers instead of whole behaviour components. This provides for context sensitive behaviour arbitration. Each behaviour is tied to an active perception unit that may implement a behaviour selection mechanism. [Hendzel 04] proposed a fuzzy combiner which can fuse low-level behaviours. The fuzzy combiner is a soft switch that chooses more than one low-level action to be active with different degrees through fuzzy combination at each time step.

The command fusion mechanism allowed multiple behaviours to contribute to the final control of the virtual agent, which means combining recommendations from multiple behaviours to form a control action that represents their consensus [Jaafar 07c]. However, a common problem arises in command fusion techniques when competing
behaviours issue conflicting control commands, which lead to oscillation of the virtual agent/robot or stagnation during navigation [Pirjanian 99a]. To overcome such a problem [Saffiotti 97] used context dependent blending of behaviours, which used a set of fuzzy rules to define a fuzzy behaviour. Another set of fuzzy rules, called meta rules, are used to control the activity of individual fuzzy behaviours by detecting conflicting situations based on current sensory information. Other work such as [Fraichard 01] used what they called Execution Monitor (EM). EM generates commands for the servo-systems of the vehicle so as to follow a given nominal trajectory while reacting in real time to unexpected events. EM is designed as a fuzzy controller, i.e. a control system based upon fuzzy logic, whose main component is a set of fuzzy rules encoding the reactive behaviour of the vehicle. Most recent work such as [Shou-Tao 06] addresses the generation of complex hierarchical behaviours by the combination of simpler behaviours as the lowest level of a hybrid architecture. The architecture is based on the theoretical foundations in designing of the Compound Zeno Behaviour [Shou-Tao 05]. Zeno Behaviour refers to primitive behaviour where two primitive behaviours make an infinite number of discrete transitions in finite time and Compound Zeno Behaviour is where more than two primitive behaviours join in the discrete transitions for infinite times in finite time.

4.2.3 Local Minima Problem

One of the major problems for local navigation is being trapped in local minima. Significant efforts have been dedicated to overcome this problem, often by using approaches from other disciplines of study such as harmonic functions [Keymeulen 94] and Maxwell’s equations [Hussein 02]. Others used randomized or optimization driven path planning algorithms including discrete grid based [Bandi 00], central path computation [Chaudhuri 04] and roadmap algorithms [Rook 05]. Unfortunately, these algorithms can be expensive in particular environments, and may even fail to reach the goal state. Approaches such as the Bug algorithm [Lumelsky 91], random walks [Chang 96], virtual target [Zou 03] and wall-follower [Yun 97] are commonly used in dealing with local minima. This can be done by iteratively modifying a global path under the influence of the obstacle’s artificial potential. These techniques focus on evaluating whether the virtual agent is in a trap before suitable action is invoked. The main weaknesses are that the virtual agent has to self-react to the sensory information and self-learn from repeated action.
Chapter 4. Virtual Agent Navigation

One of the solutions to this problem might be to start building a map, and to plan the agent's path out of the local minima situation. Artificial intelligence techniques such as neural networks [Wang 99, Lozano 02], genetic algorithms [Gordon 04] and reinforcement learning [Conde 04] have been used. All of these approaches have integrated attention and navigation skills in a virtual agent. A drawback of this approach is that it relies on training to relate inputs to outputs. What really happens during training is not quite explicit and the algorithms perform a randomised global search of an environment which requires more resources. However, this is not really necessary, if the agent can detect when it makes a U-turn and keep reasonable track of its position relative to the goal, then a few simple heuristic rules can allow it to escape from any wall and reach the goal [Jaafar 07b]. Fuzzy logic, on the other hand, is potentially a way to solve the local minima problem for autonomous navigation in virtual environments. Compared with other approaches, fuzzy logic demonstrates less computational cost. In order to overcome this problem, [Xu 00] introduces a disturbance signal to attract a robot away from the local minima and [Wu 05] introduces a fuzzy logic algorithm with back-tracking ability.

4.3 Behaviour Design

We consider virtual agents that have no internal state and that simply react to immediate stimuli in their environment, also known as stimulus-response (S-R) agents [Nilsson 98]. Virtual agents must also fulfill two main requirements [Lozano 02]:

- for any goal position, the virtual agent must reach it autonomously; and
- virtual agents must reach their goal without collision with any obstacles.

In general, each behaviour $B$ can be described in terms of a desirability function [Petropoulakis 00, Ruspini 91, Saffiotti 98]:

$$Des_B : state \times control \rightarrow [0, 1] \quad (4.1)$$

For example, based on Equation 4.1, a simple Goal-Seeking behaviour by moving toward a goal $G$, might map information from a virtual environment into a function $F$, which moves the virtual agent toward its target goal, based on function $F$. The function $F$ is a measure of the success of behaviour $C$. This will associate each instance
of the current system state $s$ with a fuzzy set of control values $c$ characterized by the membership function:

$$\Delta_C(c) = FG(s, c)$$  \hspace{1cm} (4.2)

Where the function $F$ is:

$$FG : state \times control \rightarrow [0, 1]$$  \hspace{1cm} (4.3)

In this case, $c$ is a vector of set and variable points which are position, angle, direction and distance of the virtual agent to the goal and nearest obstacle in a virtual environment.

Figure 4.3 shows a navigation framework for a virtual agent navigation. The inference process $I$, performed by the virtual agent reasoning system, can be defined as a relationship between the input space $U$ and the output space $V$. The input space $U$ defines distance and direction of the virtual agent to obstacle or goal, and the output
Chapter 4. Virtual Agent Navigation

space $V$ defines the steering angle. Thus it is expressed as [Wang 04]:

$$I : U \left(s_1, s_2, \ldots, s_j, \ldots, s_n\right) \rightarrow V \left(c_1, c_2, \ldots, c_j, \ldots, c_m\right)$$

(4.4)

The virtual agent will sense the virtual environment and the goal. If the virtual agent has identified the goal, the navigation will stop. However, if it is not a goal and it is an obstacle, then it will check the obstacle direction and distance. After obstacle direction and distance have been identified, the virtual agent will select an appropriate behaviour to avoid the obstacle using the proposed steering angle retrieved from the fuzzy controller. Finally, after avoiding the obstacle, the virtual agent will move forward toward the goal. This framework allows multiple individual behaviours, and the models of the behaviours to be executed in parallel.

The set of fuzzy logic navigation rules will drive the virtual agent from a known initial position to a goal position, regardless of obstacles etc. Once the virtual agent is aligned with the goal direction, it then proceeds towards the goal position on a straight path. Table 4.1 shows an example of a state table of virtual agent action.

<table>
<thead>
<tr>
<th>STATE</th>
<th>CONDITION</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start/Sensing</td>
<td>Goal Found</td>
<td>Stop navigation</td>
</tr>
<tr>
<td></td>
<td>Goal not found</td>
<td>Search obstacle</td>
</tr>
<tr>
<td>Obstacle Avoidance</td>
<td>Obstacle found</td>
<td>Move forward/towards goal</td>
</tr>
<tr>
<td></td>
<td>Check direction and distance</td>
<td>Calculate turning angle</td>
</tr>
<tr>
<td></td>
<td>Obstacle not found</td>
<td>Move forward/towards goal</td>
</tr>
<tr>
<td>Steering Angle</td>
<td>If direction is X AND distance is Y</td>
<td>turn Z</td>
</tr>
</tbody>
</table>

The obstacle located to the left-hand side of the virtual agent is considered an obstacle with a negative heading angle. Likewise, an obstacle located to the right-hand side of the virtual agent is considered to be of a positive heading angle. The output of the fuzzy controller is an escape vector consisting of a distance and heading angle, leading the virtual agent to an approximate target. Once this vector has been followed the virtual agent then returns to a recalculated heading vector.

4.4 Navigation System

The navigation system can be divided into three main components, which are the fuzzy navigator, virtual agent and the environment, as in Figure 4.4. The main component
of the navigation system is the virtual agent itself. The virtual agent should be able to make its own decisions; does not require any information about the virtual environment; and does not require any training or learning before the navigation task.

The fuzzy navigator is the main engine for the virtual agent. It comprises of three main components:

1. **Fuzzy Logic Controller (FLC)** - using a behaviour-based architecture which comprises of Path-Planning Behaviour (PP), Goal-Seeking Behaviour (GS) and Obstacle-Avoidance Behaviour (OA).

2. **Local Minima Solver (LMS)** - responsible for helping the virtual agent escape from dead-ends.

3. **Fuzzy Action Selection Mechanism (Fuzzy-ASM)** - to make the final decision in selecting the possible action required by the virtual agent to reach the goal.

The fuzzy navigator receives input from the visual sensor and produces the final action needed to be executed by the virtual agent. Each component in the fuzzy navigator is integrated and works independently.

In order to make the above features possible, the virtual agent is equipped with two main components, which are:

1. **visual sensor** (Section 4.4.1) - retrieves information in real-time and sends it to the fuzzy navigator; and
Chapter 4. Virtual Agent Navigation

2. virtual motion (Section 4.4.2)- translates information from the fuzzy navigator into a navigation task to reach the goal.

4.4.1 Visual Sensor

The main information between environment and virtual agent is retrieved using a visual sensor. This visual sensor differs from vision systems in robotics, since all information about pattern recognition and noisy images can be ignored [Kuffner 99]. The visual sensor captures the information about the virtual environment or identifies which part of an obstacle can be seen from the position of the virtual agent as in Figure 4.5. Also, the visual sensor only identifies whether a square (cell) in the vision range is occupied by an obstacle or not. The assumption has been made that all objects are opaque.

Figure 4.5: Example of Vision Field and Sensor’s Region based on location.

The visual sensor field of the vision range is 180°. The vision field is divided into eight main sectors which are represented as S0, S1, S2, S3, S4, S5, S6 and S7. Hence, there is a probability that the cells located in the proximity may be occupied. Cells well inside the vision field sector are likely to be empty. An occupancy grid is essentially a data structure that indicates the certainty that a specific part of space is occupied by an obstacle. It is a representation of an environment as a two-dimensional array. Each element of the array corresponds to a specific square on the surface of the actual world, and its value shows the certainty that there is some obstacle there.

The visual sensor in [Wang 99] has been modified by using Dempster-Shafer evidence theory [Shafer 76]. Whenever the virtual agent moves, it catches new information about the environment and updates the map. To facilitate building an occupancy map [Velagic 06] of the environment, a grid representing the whole space needs to be constructed. Every discrete region of the map (each cell) may be in two states,
Empty is \((E)\) and Full is \((F)\). Then, a frame of discernment, \(\kappa\), is defined by the set \(\kappa = \{E, F\}\), where \(E\) and \(F\) represent the possibility that a cell is Empty or Full. The advantage of this technique is that the building of occupancy maps is well suited to path planning and obstacle avoidance [Kim 02].

**Review of [Kim 02] Use of Dempster-Shafer’s Theory of Evidence**

A basic probability assignment is a function \(m : \kappa \rightarrow [0, 1]\), where \(\Gamma\) is a set of all subsets of \(\kappa\). In our case, \(\Gamma = 2^\kappa = \{\emptyset, \{E\}, \{F\}, \{E, F\}\}\). The state of each cell is described by assigning a basic probability number to each label in \(\Gamma\). For each cell \((i, j)\) in the grid, it is required that:

\[
m_{i,j}(\emptyset) = 0
\]  

\[
\sum_{A \in \Gamma} \{m_{i,j}\}(A) = m_{i,j}(\emptyset) + m_{i,j}\{E\} + m_{i,j}\{F\} + m_{i,j}\{E, F\} = 1
\]  

Every cell in the environment is initialized as follows:

\[
m_{i,j}\{E\} = m_{i,j}\{F\} = 0
\]

\[
m_{i,j}\{E, F\} = 1
\]

Then, the virtual agent moves and scans the environment. If \(n\) cells exist in the vision field sector, the basic probability assignment for the vision field sector is as follows:

\[
m_{i,j}(F) = \frac{1}{n}, m_{i,j}(E) = 0, \forall \text{cells}(i, j) \in \text{sector}
\]

\[
m_{i,j}(F) = 0, m_{i,j}(E) = 0, \forall \text{cells}(i, j) \notin \text{sector}
\]

By adding subscripts \(S\) and \(M\) to basic probability masses \(m\), we can describe the basic probability assignment of the sensor as equations (4.12) and (4.13):

\[
K = 1 - m_M(E)m_S(F) - m_M(F)m_S(E)
\]

\[
m_M \oplus m_S(E) = \frac{m_M(E)m_S(E) + m_M(E)m_S(\{E, F\}) + m_M(\{E, F\})m_S(E)}{K}
\]
\[ m_M \oplus m_s(F) = \frac{m_M(F)m_s(F) + m_M(F)m_s\{E,F\} + m_M\{E,F\}m_s(F)}{K} \quad (4.13) \]

However, the number of states can be reduced to two \((m_{i,j}(E), m_{i,j}(F))\), assuming that \(m_{i,j}(\emptyset) = 0\) and applying equation 4.6. The state \((0,0)\) means total ignorance, and so \(m_{i,j}\{E,F\} = 1\). When the virtual agent is sure about cell occupancy, \(m_{i,j}(F) = 1\), the other labels are made equal to zero. On the other hand, \(m_{i,j}(E) = 1\) when the virtual agent is sure that the cell is empty.

The input value \(\Theta\) of the virtual agent, which is a real number normalized in the interval \([0,1]\), then results from a weighted sum of all the points in the visual field.

\[ \Theta = \sum_{x} (2^{-2d(x)}\mu(x)) \quad (4.14) \]

summed over all \(x\) in visual field

where \(d(x)\) is the distance of a point \(x\) from the current position of the virtual agent, and \(\mu(x)\) indicates the availability of the point \(x\). Since the visual sensor is related to availability of spaces in the visual field, it is independent of specific environments and objects. The result is that the occupancy of cells is increased. This process will be carried on until the virtual agent reaches the goal.

### 4.4.2 Virtual Motion

The virtual agent only has two types of virtual motion: \(\text{MotorMove}\) and \(\text{TurnAngle}\). \(\text{MotorMove}\) is used to move the virtual agent one step forward. \(\text{TurnAngle}\) is used to turn the virtual agent by a specified angle. By computing the difference between desired angle, \(\alpha_d\) and current turning angle, \(\alpha_v\),

\[ \alpha = \alpha_d - \alpha_v \quad (4.15) \]

the new position of the virtual agent can be updated by the following equations [Wang 99]:

\[ x' = x + r\cos(\alpha') \]
\[ y' = y + r\sin(\alpha') \]
\[ \alpha' = \alpha + \theta \quad (4.16) \]
4.5 The Fuzzy Controller

The architecture of the fuzzy controller is comprised of three behaviours. The behaviours operate at three different ranges, with Path-planning (PP) and goal-seeking (GS) behaviours in global path planning, and the obstacle-avoidance (OA) behaviour in local path planning, as in Figure 4.6.

![Diagram of behaviour-based architecture](image)

Figure 4.6: Behaviour-based Architecture

4.5.1 Fuzzy Associative Memory

The relationships between fuzzy sets and rules are represented as the Fuzzy Associative Memory (FAM). FAM is a process of encoding and mapping the input fuzzy sets to output fuzzy sets. In accordance to the FAM methodology, each dimension of the matrix of FAM represents the fuzzy sets assigned to an independent variable. As the number of variables in the model increases, the number of rules used to describe the complete behaviour of the model grows exponentially. For example, a virtual agent using eight inputs and two outputs is considered. If each input is represented by three fuzzy sets and each output is represented by seven fuzzy sets, then a single layer of inference will require about $3^8 = 6,561$ rules to be established for the virtual agent. The proposed FAM approach reduces this number of rules significantly. Without it, the rules are difficult to determine, processing is time consuming and would make real time operation difficult.
4.5.1.1 Establishment of FAM

The virtual agent, is supported with eight visual sensors which provide object detection and range information for recognition of features, as well as navigation around obstacles. Figure 4.5 shows the visual sensors array with sensing angles. The input and output fuzzy sets and their ranges are shown in Table 4.2. These values are used to establish the FAM Table and fuzzy logic controller.

### Table 4.2: Linguistic Input Fuzzy Set and Their Ranges

<table>
<thead>
<tr>
<th>Distance Range</th>
<th>Variable</th>
<th>Notation</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Near</td>
<td>Ne</td>
<td>$\ell \leq 2$</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>Me</td>
<td>$2 &lt; \ell &lt; 5$</td>
<td></td>
</tr>
<tr>
<td>Far</td>
<td>Fa</td>
<td>$\ell \geq 5$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Turn angle or orientation angle $\theta$ in deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>Negative Large</td>
</tr>
<tr>
<td>Negative Small</td>
</tr>
<tr>
<td>Zero</td>
</tr>
<tr>
<td>Positive Small</td>
</tr>
<tr>
<td>Positive Large</td>
</tr>
</tbody>
</table>

4.5.1.2 Generation of behaviour rules using FAM

In the establishment of the proposed methodology, the FAM uses the fuzzy set as an index to a lookup table. The following assumptions are made in developing behaviour rules:

1. It is assumed that the virtual agent can only move in the forward direction by using the front vision sensor.

2. The rule combinations should have a tendency to select the direction that is closest to the forward direction, so that the virtual agent does not make unnecessary rotations; and

3. Rule combinations that yield empty sets should be eliminated.

In the first step of the FAM methodology, the most traversable sector in the right and left regions are found independently with a preference towards the forward direction. In the second step, the best sector among the Preferred-Right (PR), Preferred-Left
Chapter 4. Virtual Agent Navigation

(PL), and front sectors is determined. FAM uses a fuzzy inference system to derive the behaviour rules. Figure 4.7 shows the FAM model for the PR sectors and PL sectors independently, and also for the PR and PL sectors combined with the front sectors. The rules obtained from the FAM are used in the fuzzy logic controller to obtain the fuzzified output rules and the related Fuzzy Inference System (FIS) is shown in Figure 4.7.

Figure 4.7: FAM for Fuzzy Rule Representation.

4.5.2 Path-Planning Behaviour

The Path-planning (PP) behaviour is used to develop simple fuzzy rules for determination of the virtual agent turn angle as shown in Figure 4.8. This behaviour will monitor sensor information and identify a local minima situation while the regular fuzzy controller is working.

In general, the fuzzy controller for PP behaviour contains two main parts, which are turn rules and the weight rule. The local minima algorithm identifies if the virtual agent is trapped in a local minima or not. Here, the fuzzy controller will generate the potential turning angle and the weight rule used for the virtual agent in its behaviour.
4.5.2.1 Turn Rules and Weight Rules

The $\alpha$ value of PP behaviour is represented by three linguistic fuzzy sets (LOW, MEDIUM, HIGH), and is derived directly from both the obstacle distance $d_{\text{Obs}}$ and obstacle direction $\theta_{\text{Obs}}$, using the rule sets in Figure 4.9.

The virtual agent turn angle is represented by five linguistic fuzzy sets {NL, NS, ZE, PS, PL}, with the membership functions shown in Figure 4.10, where NL is negative-large, NS negative-small, ZE zero, PS positive-small, and PL positive-large. Negative and positive mean that the virtual agent turns to the left and right, respectively.
The weighting factor represents the strength by which the PP behaviour recommendation is taken into account to calculate the final motion command. The weight of PP behaviour is represented by three linguistic fuzzy sets {SMALL, MEDIUM, LARGE}, and is derived directly from both the obstacle distance and obstacle direction, using the rule sets in Table 4.3.

Table 4.3: Weight Rules for Path-Planning Behaviour

<table>
<thead>
<tr>
<th></th>
<th>HIGH</th>
<th>MEDIUM</th>
<th>LOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Danger</td>
<td>Danger</td>
<td>Danger</td>
</tr>
<tr>
<td>Medium</td>
<td>Danger</td>
<td>Danger</td>
<td>Uncertainty</td>
</tr>
<tr>
<td>Low</td>
<td>Danger</td>
<td>Uncertainty</td>
<td>Safe</td>
</tr>
</tbody>
</table>

4.5.2.2 Local Minima Algorithm

Reaction based navigation has been considered more suitable for navigation in complex and dynamically changing environments, because it controls the agent in a real-time manner as it moves around while avoiding collision using its perceptual system to gather information about the environment [Ding 05]. However, a well-known drawback of reactive navigation is that the agent suffers from local minima problems in that it uses only locally available environmental information without any previous path memory [Luh 06].

The local minima problem also occurs when a virtual agent navigating to pass obstacles towards a desired target with no a priori knowledge of the environment gets trapped in a loop. This happens especially if the environment consists of concave obstacles and mazes. To come out of the loop, the virtual agent must comprehend its repeated traversal through the same environment, which involves memorizing the part
of the environment already seen. To do so, intelligent computing-based methods such as neural networks [Lozano 02], genetic algorithms [Gordon 04] and reinforcement learning [Conde 04] have been used. Some of these methods provide good performance in specific environments. A drawback of these approaches is that they rely on training to relate input to outputs. What really happens during training is not quite explicit and the algorithms perform a randomised global search of an environment which requires more resources. [Brock 01] combine the advantages of reactive controllers and the advantages of planners by an elastic band or elastic strip formulation. However, the algorithm is relatively complex, time-consuming, and is not very useful for real-time applications.

In our architecture, the Local Minima Solver (LMS) with one-step memory is used to monitor sensor information and identify a local minima situation while the regular fuzzy controller is working. One-step memory is used for the virtual agent to know its previous step information. The main focus is on escaping from a dead-end or local minima without learning or memorizing detail of the environment. The main objective is to imitate the way a human might understand being in a trapped state and by guesswork make recovery decisions based on available information surrounding them. This can be done by recognizing its trapped state (infinite loop), where the virtual agent oscillates between two points. At this point, the agent will move one step back based on one-step memory. This will help the virtual agent to escape the dead-end situation easily. The major steps are described below:

Step 1: All the associated input variables are first fuzzified into linguistic labels within the universe of discourse, and the membership value is calculated based on the membership functions described in Table 4.3;

Step 2: The virtual agent oscillates between two points, if loop number, $\pi > 10$. If no PP behaviour evaluates to Danger, $W = 0$ and go to regular fuzzy controller. Otherwise, the virtual agent will move one step to its previous step and move to step 3;

Step 3: Calculate the new weight value $W_{pp}$. If $W_{pp} > 1$, go to regular fuzzy controller; else, if $W_{pp} \leq 1$ and PP behaviour evaluates to Danger, get new sensor information, else, send the $W_{pp}$ value to the behaviour selection module.

Step 4: The behaviour selection module will decide what action needs to be executed, such as a larger right/left turn, or to back up to a nearest safe point along the safe
Chapter 4. Virtual Agent Navigation

path, and then make a turn, to pull the virtual agent out of its trap.

4.5.3 Goal-Seeking Behaviour

The Goal-Seeking behaviour generates a turning angle based on the location of the goal. A simple analytical model rather than a set of fuzzy logic navigation rules have been used. Two assumptions has been made, that the GS behaviour:

1. does not influence the speed of the virtual agent, and contributes only to the turning angle; and
2. the turn angle recommended by the GS behaviour is the heading error between the current virtual agent heading and goal direction. Thus, the value domain of this turn angle is (-180°, 180°).

The visual sensor provides information that the virtual agent is presently at the position (x,y) with azimuth, and the goal location is at (x_1, y_1). Equations (4.15) and (4.16) are used in calculating goal direction.

Once the goal direction has been identified, GS behaviour will produce a potential turning angle towards the goal. The weight of GS behaviour $w_{gs}$ has also been calculated based on the weights of both OA and PP behaviours. Figure 4.11 shows the weight determination of three behaviours. The simple weight rules are as follows:

- IF $W_{pp}$ is Large OR $W_{OA}$ is Large, THEN $W_{GS}$ is Small
- IF $W_{pp}$ is Small AND $W_{OA}$ is Small, THEN $W_{GS}$ is Large

Importantly, the weight of GS behaviour is suppressed and small when any one weight of the OA and PS behaviours is not SMALL. When the weights of both OA and PS are SMALL, the GS behaviour makes a dominant contribution to the final control command. Although the GS behaviour is often suppressed, the GOAL factor is reflected in the turn rules of both OA and PP behaviours.

4.5.4 Obstacle Avoidance Behaviour

The local Obstacle-Avoidance (OA) behaviour is actually a sensor-based behaviour which implements a control strategy based on external sensing. OA behaviour is effective if obstacles are close. The visual sensors of the virtual agent are grouped into five sectors (Left, LeftFront, Front, RightFront and Right). If the virtual agent has
a ring of eight forward vision fields, these will produce a set of obstacle distances $(d_0,d_1,d_2,d_3,d_4,d_5,d_6,d_7)$. From these, we obtain three groups of obstacle distances by the following equations:

- $d_{left} = \min(d_0,d_1)$;
- $d_{from} = \min(d_2,d_3,d_4,d_5)$;
- $d_{right} = \min(d_6,d_7)$.

The obstacle distance of each sector is represented by three linguistic fuzzy sets \{NEAR, MEDIUM, FAR\}. The turn rules for the OA behaviour are summarized in Figure 4.12. Observe that the rules exhibit the behaviour characteristic: if the obstacle distance in any sector is NEAR, the virtual agent should turn away to find a safer direction. For instance, the $(1,3)$ element of the bottom layer in Figure 4.12 can be written out as the rule:

- **IF** $d_{from}$ is Far AND $d_{left}$ is Far AND $d_{right}$ is Near, **THEN** $\theta_{oa}$ is PS

Note that, in Figure 4.12, when the virtual agent needs to turn, but the left and right sectors have the same obstacle distance, then the recommended turn angle is GOAL, where GOAL implies that the recommended turn angle should be toward the direction closest to the goal location. This is similar to the turn rules for PP behaviour. One last important note: when the three sectors have the same NEAR obstacle distance as shown in the $(3,0)$ element of the top layer in Figure 4.12, a large left turn (PL) angle
Figure 4.12: Turn Rules for OA Behaviour

Figure 4.13: Weight Rules for OA Behaviour
is recommended. This turn rule enables the virtual agent to escape from its current dead-end situation.

Similarly to the weights of PP behaviour, the weights of OA behaviour \( W_{oa} \) are represented by three linguistic fuzzy sets \{SMALL, MEDIUM, LARGE\}, and are derived directly from obstacle distances in the three sectors. The weight rules for the OA behaviour are summarized in Figure 4.13.

### 4.6 Experiments

The objective is to measure the performance of the reactive architecture based on fuzzy logic developed in the previous section, Fuzzy-ASM. The performance is based on robustness and quality of path produced by an autonomous virtual agent during a navigation task. In order to test the performance of the reactive architecture based on fuzzy logic, five different experiments have been carried out in simulation. The experiments are moving towards the goal; escape from local minima; navigation in a complex environment; the action selection method; and performance comparison.

**Experiment Criteria**

In the experiments, the navigation task is in an unknown environment. The only information known by the virtual agent are the coordinates of the start and target/goal points. The navigation task requires the virtual agent to activate each of the behaviours separately in its own context of applicability. The shapes of obstacles are not used as parameters. The following criteria have been taken into account during the experiment:

1. Each experiment will be run 25 times.
2. A limited 2000 steps per run was set, and if reached, it meant the run was unsuccessful and the virtual agent failed to reach its goal.

**Assessment Strategy**

The performance metric is a measure based on virtual agent navigation tasks in different unknown environment settings, based on the concept of *run*. A run is a path from a new start point to the randomly selected goal. The performance of each experiment has been summarised and compared. The evaluation can be divided into:
Chapter 4. Virtual Agent Navigation

1. **Batch of trials**

   Within the batch, a group of runs will be executed, corresponding to different types of virtual environment. The performance of groups within the batch will be summarised and compared.

2. **Performance comparison**

   The same parameters will be used to provide a direct way of comparing results. Two types of comparison are used:

   (a) using different behaviour weights, as in Section 4.6.5.1

   (b) comparison with other methods - The actual rule been used in our method had been modified to match with method been used in the evaluation. Most of the modification is declaration structure of the rule since those methods using difference parameter in their fuzzy controller. The numbers of rules still the same.

   i. Fuzzy-ASM vs. Fuzzy Behaviour Fusion (FBF) [Cang 00] in Section 4.6.4. FBF uses a behaviour-based architecture with behaviour fusion as action selection method. Since it use the same architecture and the main difference is in action selection method, this will help us to measure the performance of our action selection method compared with the behaviour fusion method.

   ii. Fuzzy-ASM vs. Fuzzy Potential Field (FPF) [Makita 94, Katoh 04] vs. Fuzzy Roadmap (FRM) [Sanchez 04, Lee 04b] in Section 4.6.5. FPF and FRM use different architectures for the same purpose as Fuzzy-ASM. Both methods are commonly used with fuzzy logic. This will help us to measure performance of our architecture compared to other known architectures in solving the same problem. There is some modification in rules been used depend on the architecture.

**Performance Parameters**

Throughout the experiments we aim to maintain correctness, consistency and completeness of the virtual agent navigation in a virtual environment. Two characteristics will be used as quality indices during the experiments:
Chapter 4. Virtual Agent Navigation

1. **Path Quality** - The quality of navigation should satisfy the following criteria [Overmars 01]:

   • *short path* - it should not contain long detours when significantly shorter routes are possible and visible;
   
   • *smooth path* - containing no sharp turns;
   
   • *path clearance* - the distance of any point on the path from the closest obstacle should not be lower than some prescribed value.

   Note that the requirements for a smooth path and path clearance may conflict with the short path criteria in the case when it is possible to considerably shorten the path by taking a shortcut through a narrow passage. In such cases we may prefer a path with less clearance (and perhaps containing sharp turns).

2. **Robustness** – the capability of a virtual agent to carry out a successful navigation in environments with disturbed conditions [Hoshino 98]. Specifically, to what extent does it accomplish its goals in specified environments, and are those methods applicable across many different environments and tasks?

The following performance parameters [Yen 95] are defined in identifying robustness of different coordinators.

1. *Average distance to the nearest obstacle* ($\overline{d_0}$)

   \[
   d_0 = \frac{1}{K} \sum_{i=1}^{K} d_0(i)
   \]  

   Where $K$ is the total number of decision cycles and $d_0(i)$ is the distance to the closest obstacle in $i^{th}$ decision cycle. High value of $\overline{d_0}$ indicates safer navigation.

2. *Total traveled distance* ($d$) - Low value of $d$ is expected to optimize the traveled distance.

3. *Total navigation time* ($t$) - Low value of $t$ expected for fast navigation.

4. *Total number of collisions* ($C$) - should be zero for safe navigation.

5. *Safety Index* ($SI$) - $\overline{c}$ percentage of the simulation runs in which the agent successfully reaches the goal without collision.
Chapter 4. Virtual Agent Navigation

6. Steering Smoothness Index (SSI)

\[ \bar{\alpha} = \frac{\sum_{i=1}^{k} |\Delta \tilde{\theta}_i|}{k} \]  

where \( |\Delta \tilde{\theta}_i| \) stands for absolute average steering angle in the \( i^{th} \) simulation run.

7. Velocity Smoothness Index (VSI)

\[ \bar{\alpha} = \frac{\sum_{i=1}^{k} |\Delta \tilde{v}_i|}{k} \]  

where \( |\Delta \tilde{v}_i| \) stand for the absolute average of the velocity change in the \( i^{th} \) simulation run.

8. Average radius of curvature \( R_{cur} \) which is defined as:

\[
\rho = \frac{1}{K-1} \sum_{i=3}^{K} \rho(i), \rho(i) = \frac{[\Delta x_r(i)]^2 + \Delta y_r(i)^2}{[\Delta x_r(i)^2 y_r(i) - \Delta y_r(i)^2 x_r(i)]^{\frac{3}{2}}} \Delta x(i) \\
= x_r(i) - x_r(i-1), \Delta y_r(i) \\
= y_r(i) - y_r(i-1), \Delta x_r(i) \\
= \Delta x_r(i) - \Delta x_r(i-1), \Delta y_r(i) - \Delta y_r(i-1) \\
\]

where \((x_r(i), y_r(i))\) is the agent coordinate in \( i^{th} \) decision cycle. Higher value of \( \bar{\rho} \) indicates smoother trajectory of navigation.

4.6.1 Moving Towards the Goal

The experiment is conducted in the environment with one obstacle and the result is shown in Figure 4.14. The virtual agent moves toward the goal, when it reaches the obstacle, the virtual agent starts to turn to the right slowly to avoid the obstacle. At the same time it still maintains its path toward the goal.

Table 4.4 shows statistical results for time, path length and number of decisions taken by the virtual agent during navigation to reach the goal. The results show the
mean for time taken, $t = 5.4507\text{s}$, the length is $d = 60.7833$ and total number of decisions is $K = 28.6544$. Looking at the Mean Standard Error (S. E.), these are very low, at $0.0325$ for time, $0.3684$ for length and $0.2369$ for number of decisions. The differences between maximum and minimum values are also low, which are $\Delta t = 0.5500\text{s}$, $\Delta d = 6.1167$ and $\Delta K = 4.550$.

The result also indicates that the number of decision become very high mainly when the virtual agent starts making a turn. This makes the virtual agent change their angle frequently. Not all steps been shown in Figure 4.14 but it enough to show the frequency of steps along the path. The result also shows that eventhough the virtual agent has high number of decision to be made, it still maintans its speed by taking shortes path and shortes time to reach the goal.

**Table 4.4: Statistical Result for Navigation with One Obstacle**

<table>
<thead>
<tr>
<th></th>
<th>Time ($t$)</th>
<th>Distance ($d$)</th>
<th>Decision ($K$)</th>
<th>$\theta_{\text{min}}$</th>
<th>$\theta_{\text{max}}$</th>
<th>$\Delta \theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>5.7467</td>
<td>63.9000</td>
<td>31.2000</td>
<td>0.028058</td>
<td>0.028900</td>
<td>0.003000</td>
</tr>
<tr>
<td>Min</td>
<td>5.1967</td>
<td>57.7833</td>
<td>26.6500</td>
<td>0.021667</td>
<td>0.023500</td>
<td>0.000500</td>
</tr>
<tr>
<td>$\Delta$(max-min)</td>
<td>0.5500</td>
<td>6.1167</td>
<td>4.5500</td>
<td>0.006392</td>
<td>0.005400</td>
<td>0.002500</td>
</tr>
<tr>
<td>Mean</td>
<td>5.4507</td>
<td>60.7833</td>
<td>28.6544</td>
<td>0.025113</td>
<td>0.026574</td>
<td>0.001461</td>
</tr>
<tr>
<td>Mean S.E. ($\bar{e}$)</td>
<td>0.0325</td>
<td>0.3684</td>
<td>0.2369</td>
<td>0.000349</td>
<td>0.000338</td>
<td>0.000141</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>0.1625</td>
<td>1.8422</td>
<td>1.1845</td>
<td>0.01745</td>
<td>0.016878</td>
<td>0.00704</td>
</tr>
</tbody>
</table>

From the graph in Figure 4.15 we can see the average turning angle for each of the 25 test runs. Additionally in Table 4.4, we show statistical results for turning angle produced by the virtual agent. The path produced can be considered smooth since there is no major angle turn produced. The difference for the minimum turning angle is 0.0005 and for the maximum is 0.003, which can be considered low. The mean for the minimum turning angle is 0.0251 and for the maximum turn angle is 0.0266, and the Mean S.E.s are 0.000349 for the minimum and 0.000338 for the maximum
turn angles. The differences $\Delta \theta$ for each test run are very low and consistent. This indicates that the turn angle produced by the virtual agent during the navigation task is small. As a result, this indicates smooth turning during avoidance of the obstacle.

![Figure 4.15: Turning Angle for One Obstacle.](image)

Weight transition between behaviours has been measured to verify the smoothness of behaviour transition of the fuzzy controller. Behaviour weight, $W$, versus the time step graph is plotted in Figure 4.16. The graph shows that the behaviour weight, $W$, increases and decreases gradually when encountering or leaving an obstacle. Therefore the fuzzy controller transits between behaviours gradually instead of switching between them.

![Figure 4.16: Behaviour Transition](image)

In terms of defuzzification techniques, three techniques have been tested. The objective of this testing was to evaluate the robustness of the fuzzy controller used
with different defuzzification techniques. These are our proposed technique, Mean of Maximum (MOM) and Center of Area (COA). The index of optimism (as discussed in Section 3.4) has been fixed to 0.1 for all methods so that the agent will have a high level of uncertainty. The results in Table 4.5 show that the proposed defuzzification technique has produced a safer and smoother control of the virtual agent compared to the MOM and COA techniques. This also shows that our technique is computationally faster and easier and gives fairly accurate results. In contrast, COA is computationally difficult because of having complex membership functions and MOM computationally faster but is only accurate for peaked output.

In addition to obstacle avoidance and reaching the goal, navigating a narrow path has also been tested. Three narrow paths have been used which are to navigate between two walls, a narrow passage and a corner. The virtual agent produced consistent results (time, number of decisions and path length) for all 25 test runs with a mean error less than 5%. The path produced can be considered smooth since there is a minimum of sharp turns as in Figure 4.17.

For example, in all basic navigation skills the results show that the virtual agent produced consistent results for all 25 test runs. From Tables 4.6, 4.7 and 4.8 the range of $\Delta \theta$ and Mean S.E. of the turning angle is very small, which produces a smooth path. The result for two walls and corner shaped obstacles was very similar.

![Figure 4.17: Basic Navigation Skill for Two Wall, Narrow Passage and Corner](image-url)

### 4.6.2 Escape from Local Minima Problem

Navigation can be very difficult because the virtual agent only uses sensory information, and has no prior knowledge about the environment. One of the major problems for local navigation is being trapped in local minima [Jaafar 07a]. The local minima problem occurs when a virtual agent navigating past obstacles towards a desired target with no prior knowledge of the environment gets trapped in a loop. This happens especially if the environment consists of concave obstacles, mazes or something similar. A new local minima solution has been implemented as in Section 4.5.2.2.
Table 4.5: Comparison of Defuzzification Technique

<table>
<thead>
<tr>
<th>Method</th>
<th>Smoothness</th>
<th>Safety Index</th>
<th>Step/run</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOM</td>
<td>2.698</td>
<td>0.415</td>
<td>175</td>
</tr>
<tr>
<td>COA</td>
<td>0.620</td>
<td>0.907</td>
<td>217</td>
</tr>
<tr>
<td>Proposed</td>
<td>0.550</td>
<td>0.980</td>
<td>153</td>
</tr>
</tbody>
</table>

Table 4.6: Navigating Two Walls

<table>
<thead>
<tr>
<th>Video</th>
<th>Time (t)</th>
<th>Distance (d)</th>
<th>Decision (K)</th>
<th>( \theta_{\text{min}} )</th>
<th>( \theta_{\text{max}} )</th>
<th>( \Delta \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>8.6967</td>
<td>137.7833</td>
<td>95.3000</td>
<td>0.0211</td>
<td>0.0229</td>
<td>0.0005</td>
</tr>
<tr>
<td>Min</td>
<td>7.8370</td>
<td>144.0333</td>
<td>99.8500</td>
<td>0.0275</td>
<td>0.0285</td>
<td>0.0038</td>
</tr>
<tr>
<td>( \Delta \text{(max-min)} )</td>
<td>0.8597</td>
<td>6.2500</td>
<td>4.5500</td>
<td>0.0065</td>
<td>0.0057</td>
<td>0.0033</td>
</tr>
<tr>
<td>Mean</td>
<td>8.2390</td>
<td>140.8233</td>
<td>98.1744</td>
<td>0.0236</td>
<td>0.0252</td>
<td>0.0017</td>
</tr>
<tr>
<td>Mean S.E. (( \bar{\varepsilon} ))</td>
<td>0.0466</td>
<td>0.3432</td>
<td>0.2388</td>
<td>0.0003</td>
<td>0.0004</td>
<td>0.0001</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>0.2330</td>
<td>1.7161</td>
<td>1.1941</td>
<td>0.0018</td>
<td>0.0016</td>
<td>0.0010</td>
</tr>
</tbody>
</table>

Table 4.7: Navigating Through Narrow Passage

<table>
<thead>
<tr>
<th>Video</th>
<th>Time (t)</th>
<th>Distance (d)</th>
<th>Decision (K)</th>
<th>( \theta_{\text{min}} )</th>
<th>( \theta_{\text{max}} )</th>
<th>( \Delta \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>2.5767</td>
<td>66.7833</td>
<td>129.8500</td>
<td>0.0275</td>
<td>0.0286</td>
<td>0.0030</td>
</tr>
<tr>
<td>Min</td>
<td>2.1967</td>
<td>60.2833</td>
<td>124.1000</td>
<td>0.0215</td>
<td>0.0225</td>
<td>0.0002</td>
</tr>
<tr>
<td>( \Delta \text{(max-min)} )</td>
<td>0.3800</td>
<td>6.5000</td>
<td>5.7500</td>
<td>0.0060</td>
<td>0.0060</td>
<td>0.0028</td>
</tr>
<tr>
<td>Mean</td>
<td>2.3850</td>
<td>63.6873</td>
<td>127.8544</td>
<td>0.0250</td>
<td>0.0265</td>
<td>0.0015</td>
</tr>
<tr>
<td>Mean S.E. (( \bar{\varepsilon} ))</td>
<td>0.0200</td>
<td>0.3458</td>
<td>0.3024</td>
<td>0.0003</td>
<td>0.0003</td>
<td>0.0002</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>0.0998</td>
<td>1.7292</td>
<td>1.5118</td>
<td>0.0017</td>
<td>0.0017</td>
<td>0.0008</td>
</tr>
</tbody>
</table>

Table 4.8: Navigating Through Corner

<table>
<thead>
<tr>
<th>Video</th>
<th>Time (t)</th>
<th>Distance (d)</th>
<th>Decision (K)</th>
<th>( \theta_{\text{min}} )</th>
<th>( \theta_{\text{max}} )</th>
<th>( \Delta \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>3.7467</td>
<td>22.5333</td>
<td>31.2000</td>
<td>0.0281</td>
<td>0.0289</td>
<td>0.0020</td>
</tr>
<tr>
<td>Min</td>
<td>3.2067</td>
<td>19.2833</td>
<td>26.6500</td>
<td>0.0217</td>
<td>0.0235</td>
<td>0.0005</td>
</tr>
<tr>
<td>( \Delta \text{(max-min)} )</td>
<td>0.5400</td>
<td>3.2500</td>
<td>4.5500</td>
<td>0.0064</td>
<td>0.0054</td>
<td>0.0195</td>
</tr>
<tr>
<td>Mean</td>
<td>3.4475</td>
<td>20.7433</td>
<td>28.6144</td>
<td>0.0251</td>
<td>0.0266</td>
<td>0.0015</td>
</tr>
<tr>
<td>Mean S.E. (( \bar{\varepsilon} ))</td>
<td>0.0258</td>
<td>0.1823</td>
<td>0.2300</td>
<td>0.0003</td>
<td>0.0004</td>
<td>0.0200</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>0.0258</td>
<td>0.1823</td>
<td>0.2300</td>
<td>0.0003</td>
<td>0.0004</td>
<td>0.0200</td>
</tr>
</tbody>
</table>

Figure 4.18: Escaping from bench, corner and dead-end.
We verify the performance of our approach by applying it to three types of basic obstacle shape which are a bench, corner and a U-shape as shown in Figure 4.18. The simulation shows that the virtual agent avoids the obstacle, then follows the wall and heads for the target. When reaching a dead-end or trap (local minima), the virtual agent successfully escapes from the situation. The virtual agent increases its speed when there is no obstacle and moves forward, with less decision making needed to be made. When reaching a trap situation, the agent slows down since it needs to sense the obstacle and to decide which turn angle needs to be taken. The time and number of decisions needed to be taken depends on the sharpness of the turn angle needed to be taken. The path generated by the virtual agent can be considered smooth even though it is not the shortest path. In general, the experiments demonstrate that the method is robust and the agent has an adequate capability to escape from a local minima situation.

Furthermore, to verify its performance, a long-wall environment with a U-shape trap has been used, as shown in Figure 4.19. The path produced can be considered smooth even though there are sharp turns, especially in the beginning and in the area of the dead-end. The path produced is also not too far from the obstacle trying to minimize the distance to the obstacle in order to produce a possible shortest path. The number of steps is high and the virtual agent's speed slows since it requires extra processing time for decision making and making a turn. The virtual agent changes to normal speed and time step when leaving the dead-end, making a deversion and following the wall. The speed gradually decreases and time steps start to increase when the agent moves along the wall and needs to make a small deviation to the goal. The reason is that the virtual agent needs to make a left turn to the goal but at the same time needs to move away from the long wall. The virtual agent start its turn to the left at the end of the wall with time step and speed becoming normal.
Chapter 4. Virtual Agent Navigation

Figure 4.20 shows the turn angles recommended by different behaviours. The turn angle recommended by OA and PP behaviours are consistent during the navigation task and the GS turn angle is more than in OA and PP. At some point, the goal is switched from the left of the virtual agent to the right, or from the right to the left. This is why the virtual agent leaves the wall (obstacle) and turns toward the goal direction. The virtual agent starts in the central point of the U-shape obstacle and starts to move at a normal speed toward the goal, the distance can be considered as short, which makes the turn angle small. \( W_{PP} \) and \( W_{OA} \) are small, however \( W_{GS} \) is large and as a result GS behaviour contributes to the final motion of the virtual agent. This is because the facing obstacle is distant. When the virtual agent is near to the obstacle, the weight of the OA behaviour becomes larger. When the OA and PP behaviours are dominant, the GS behaviour is suppressed and its weight is small. When the virtual agent is far from an obstacle and approaching the goal, the weight of OA and PP behaviours are small and only the GS behaviour is dominant.

4.6.3 Navigating in a Complex Environment

In this section, we address the problem of collision-free navigation of virtual agents moving in a complex environment. The experiment is concerned with the ability of the virtual agents to navigate in cluttered and maze environments. The first experiments evaluate virtual agent navigation in cluttered and maze environments using different degrees of uncertainty. Figure 4.21(a) shows a result for a cluttered environment. Fig-
Chapter 4. Virtual Agent Navigation

Figure 4.22(a) shows the degradation of the Safety Index (SI) and Steering Smoothness Index (SSI) of the fuzzy controller as the degree of uncertainty increases. The steering smoothness index, (SSI) increases only to 1.47 times larger while the Velocity Smoothness Index (VSI) increases to 4.35 times larger, meaning that the VSI has more influence on the degree of uncertainty. The maximum value of SSI is $4.2^\circ$ and the degradation of SSI is graceful. The maximum value of VSI is 1.65, this is relatively large compared to the maximum velocity of the virtual agent.

![Figure 4.21: Navigation path (a) Cluttered and (b) Maze Environment.](image)

Figure 4.21(b) shows a path produced in a maze environment. There are some sharp turns but other parts of the path are still considered as smooth. Similar results have been produced for the cluttered environment. Figure 4.22(b) shows the degradation of the Safety Index (SI) and Steering Smoothness Index (SSI) of the fuzzy controller as the degree of uncertainty increases. The Smoothness Index, (SI) increases only to 1.89 times larger while the Velocity Smoothness Index (VSI) increases to 5.24 times larger, meaning that the VSI has more influence on the degree of uncertainty. The maximum value of SSI is $3.5^\circ$, the degradation of SSI is graceful. The maximum value of VSI is 1.78 and this is relatively large compared to the maximum velocity of the virtual agent.

Figures 4.21 and 4.22 show that even if the value of uncertainty is increased as high as 1.0, the smoothness of the path still can be maintained. The fuzzy controller is still able to avoid the obstacle in most cases since the standard deviation of uncertainty measurement is not as large as 60% of the actual value in most cases. This means the fuzzy controller has high robustness to the level of uncertainty of the environment.

Experiments were also conducted to observe the effect of using different degrees of optimism, $\sigma$, by the virtual agent to navigate in complex environments. Figure 4.23 shows the result of the experiment conducted in a cluttered environment using different degrees of optimism, $\sigma$, which are (a), $\sigma = 0.9$ and (b) $\sigma = 0.4$. The environments contain different sizes of obstacle and narrow passages. The virtual agent in Figure 4.23(a)
Chapter 4. Virtual Agent Navigation

Figure 4.22: Navigation result (a) Cluttered and (b) Maze Environment

Figure 4.23: Different Degrees of Optimism (a) $\sigma = 0.9$ and (b) $\sigma = 0.4$. 
Chapter 4. Virtual Agent Navigation

has produced a shorter path compared to the virtual agent in Figure 4.23(b). However the number of steps is higher compared to Figure 4.23(b). The main reason is that the virtual agent is required to go through a narrow passage in order to produce the shortest path. In Figure 4.23(b), the virtual agent has made a sharp turn and high number of time steps at this point. As a result the virtual agent take a big turn to the wider passage before turning and reaching the goal. Time steps at the rest of the path are consistent since there is no complex obstacle to avoid. The results show that the decision process by the virtual agent is affected by the degree of optimism. Using a higher value of $\sigma$ makes the virtual agent enter the narrow passage compare to a low value of $\sigma$ which makes the agent prefer to select the wider passage. However the number of decisions and steps might vary depending on the complexity of the environment.

Further experiments with complex environments have been conducted. The environments contain a combination of maze and cluttered obstacles and three random goals have been selected. The degree of optimism, $\sigma = 0.5$, was used for the first trial. Unfortunately this value did not give a very promising result as in Figure 4.24. The virtual agent had successfully reached the goal, but paths produced are long with many sharp turns and a high number of time steps.

Figure 4.24: Navigating in combination of cluttered and maze environment ($\sigma = 0.5$).

Figure 4.25: Navigating in combination of cluttered and maze environment ($\sigma = 0.8$).

Based on result in Figure 4.23, using a higher value of $\sigma$ will give a better result. A new value of $\sigma = 0.8$ has been selected. Figure 4.25(a) and (c) produce smooth and short paths compared to the results in Figure 4.24(a) and (c). In Figure 4.25(b),
the virtual agent follows a similar path compared to Figure 4.24(b) but with a small number of sharp turns. From the figures, we also notice that the virtual agent does not take the narrow path at $X$. One probable is that the passage is too narrow and might require a higher value of $\sigma$. However having a higher value of $\sigma$, the virtual agent might follow a longer and unsafe path.

Also in Figure 4.24(b) and Figure 4.25(b), notice that the virtual agent does not produce the shortest path. The virtual agent moves forward to the goal even though there are a walls and a dead-end. Then the virtual agent makes a left turn to escape from dead-end and follow the wall toward the goal. The virtual agent tried to reach the goal by moving straight ahead towards the goal by having a high value for Goal-Seeking behaviour. The virtual agent starts to switch to Path-Planning behaviour and Obstacle-Avoidence behaviour when it encounters an obstacle and needs to make a turn to reach the goal. This shows that the virtual agent has imitated how a human might make decisions during a navigation task in an unknown environment by making a good assumption that the path to the goal is ahead of them even though they cannot see the goal.

The experiment has shown that the $\sigma$ value might vary depending on complexity of the environment. This is because some environments might have many narrow passages or two walls. With a high value of $\sigma$, the virtual agent can go through the narrow passage. Alternatively, with a low value of $\sigma$, the virtual agent might look for a wider passage. The paths produced might not be the shortest paths but they are safe paths (no collision). This is due to the ability of the virtual agent to identify its goal and the capability of the visual sensor in detecting potential obstacles.

**4.6.4 Action Selection Method**

A central issue in the design of reactive control architectures for autonomous virtual agents is the formulation of effective action selection mechanisms (ASMs) to coordinate the behaviours. Experiments will evaluate the Fuzzy-ASM method and compare the results with the behaviour fusion method (FBF) by Cang [Cang 00].

Four test cases have been used which are the virtual agent being moved from the same start point to different target points as in Figure 4.26 to 4.29 (Test Case 1, 2, 3 and 4). Figure 4.29 shows the example of the path produced by the virtual agent in Test Case 4. Figure 4.29(a) is the path produced by Cang's Method and Figure 4.29(b) shows the path produced by our Fuzzy-ASM. The path produced by the Fuzzy-ASM
Chapter 4. Virtual Agent Navigation

Figure 4.26: Test Case 1

Figure 4.27: Test Case 2

Figure 4.28: Test Case 3

Figure 4.29: Test Case 4
Figure 4.30: (a) Time ($t_n$), (b) Distance ($d_r$) and (c) Decisions ($K$).
is shorter than Cang’s method even though the smoothness of the path is similar.

Table 4.9: The Performance of Fuzzy-ASM vs. FBF.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>$P_s(m)$</th>
<th>Fuzzy-ASM</th>
<th>FBF</th>
<th>Fuzzy-ASM</th>
<th>FBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20.30</td>
<td>21.32</td>
<td>22.42</td>
<td>4.93</td>
<td>10.44</td>
</tr>
<tr>
<td>2</td>
<td>22.35</td>
<td>28.81</td>
<td>na</td>
<td>28.90</td>
<td>na</td>
</tr>
<tr>
<td>3</td>
<td>14.64</td>
<td>20.91</td>
<td>na</td>
<td>42.83</td>
<td>na</td>
</tr>
<tr>
<td>4</td>
<td>23.01</td>
<td>23.86</td>
<td>25.12</td>
<td>3.69</td>
<td>9.17</td>
</tr>
</tbody>
</table>

Note:
- $P_s$ - the shortest path length
- $P_a$ - the actual path length
- $E$ - the performance factor, which if small means that the performance of the method is better.

Moreover, Table 4.9 shows that Fuzzy-ASM had a better performance with a small performance factor ($E$), which was 4.38% in Test Case 1 and 2.80% in Test Case 4 compared to a FBF of 11.31% and 9.39%, respectively. The difference in paths produced between the Fuzzy-ASM and FBF are 1.50 in Test Case 1 and 1.56 in Test Case 4. The difference in $E$ and path produced shows that the Fuzzy-ASM has produced better results compared to the FBF. Unfortunately, Test Case 2 and Test Case 3 showed that FBF was trapped in local minima and failed to complete the task. Fuzzy-ASM had a successful escape form the trap and reached the target point.

Further testing has also been conducted with nine different goal locations as in Appendix A. Figure 4.30 shows the result of (a) Time ($t_n$), (b) Distance ($d_i$), and (c) Decisions ($K$) taken by the virtual agent for all nine goal locations. The results show that Fuzzy-ASM has taken less time and a shorter distance to complete the task. The average percentages of $\Delta t_n$ and $\Delta d_i$ are 16% and 17.4%, respectively. When we compare the number of decisions made by each method, Fuzzy-ASM has made fewer decisions. The average number of decisions is 8.04% less than Wang’s method. Fewer decisions leads to a faster and more reliable decision making process.

Our tests also show that the success rate for the Fuzzy-ASM is higher than Wang’s method, as shown in Figure 4.31. Success rate refers to the percentage of test runs (total of 25 runs) for each test where the virtual agent successfully reached the goal. In test 1 to test 4, the fuzzy ASM had a 100% success rate. Wang’s method starts to
Chapter 4. Virtual Agent Navigation

4.6.5 Performance Comparison

4.6.5.1 Comparison Between Different Scenarios.

The aim of this experiment is to measure the robustness of the Fuzzy-ASM using different behaviour weights. Three behaviour weights have been used to compare the navigation results:

- **Behaviour Weight 1 (BW1)** - the Fuzzy-ASM.
- **Behaviour Weight 2 (BW2)** - The behaviour weights are randomly defined as:
  
  \[ B_{OA} = 0.46, \quad B_{PP} = 0, \quad B_{GS} = 0.79 \]

  to see how the virtual agent behaves when having permanent weight.

- **Behaviour Weight 3 (BW3)** - All behaviour weights are set to 1,
  
  \[ i.e., \quad B_{OA} = B_{PP} = B_{GS} = 1. \]

  to see how the virtual agent behaves when having equal weight.

For the experiments, the same virtual agent is used and visual sensors are employed to obtain range measurements in order to avoid obstacles. For unmodulated coordination, the sampling time period (i.e., the length of a decision cycle) is set at \( t_p = 50ms \). Three different scenarios have been created.
Case I: An open U-shaped obstacle is placed on the way to the target.

Case II: Narrow passage is placed to obstruct the way to the target.

Case III: Two walls are created on the way to the target.

For the navigation examples, only odometry is used for localization of the virtual agent. In each experiment the virtual agent path is traced and the results are shown in Figures 4.32 to 4.34 and Table 4.10.

Figure 4.32: A U-shaped Obstacle is Placed on the Way to the Target.

Figure 4.33: Narrow Passage is Placed to Obstruct the Way to the Target.

Figure 4.34: A Two Walls are Created on the Way to the Target

In all navigation scenarios the virtual agent was able to reach the target locations using BW1 (Figures 4.32(a), Figure 4.33(a) and Figure 4.34(a)). However, Table 4.10 shows that the virtual agent collided with obstacles in Case III (BW3, Figure 4.34(c)) due to the non-significant contribution of the avoid-obstacle schema. Hence, the performance criteria associated with these cases are ignored since this method violates
Table 4.10: Performance Comparisons

<table>
<thead>
<tr>
<th>Behaviour Weight</th>
<th>$d_o$</th>
<th>$d_1$</th>
<th>$t_n$</th>
<th>$C$</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Case I</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>854</td>
<td>5654</td>
<td>1328</td>
<td>0</td>
<td>$2 \times 10^{10}$</td>
</tr>
<tr>
<td>2</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>3</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td><strong>Case II</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>994</td>
<td>2620</td>
<td>240801</td>
<td>0</td>
<td>$2 \times 10^{12}$</td>
</tr>
<tr>
<td>2</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>3</td>
<td>1062</td>
<td>2767</td>
<td>25801</td>
<td>0</td>
<td>$10 \times 10^{11}$</td>
</tr>
<tr>
<td><strong>Case III</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>786</td>
<td>2840</td>
<td>25041</td>
<td>0</td>
<td>$1 \times 10^{13}$</td>
</tr>
<tr>
<td>2</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>3</td>
<td>862</td>
<td>2918</td>
<td>26044</td>
<td>1</td>
<td>$2 \times 10^{11}$</td>
</tr>
</tbody>
</table>

the safe navigation objective set. In Case II, BW3 (Figure 4.33(c)) produces lower values of $d_o = 1011$ and $\rho = 11 \times 10^{11}$, and higher values of $d_1 = 3767$ and $t_n = 25801$ compared to BW1 (Figure 4.33(a)) which produces high values of $d_o = 1098$ and $\rho = 3 \times 10^{12}$, and low values of $d_1 = 3620$ and $t_n = 23080$. Both behaviour weights (BW1 and BW3) produce safe navigation. BW3 produced unsafe navigation, un-smooth trajectory, higher traveled distance and navigation time, and inconsistent motion commands as compared to BW1.

1. **Behaviour Weight 1**

In this navigation task the behaviours are weighted according to environmental contexts. Appropriate behaviours are selected, Figure 4.35 demonstrates the weights generated in Case I.

In Case I, Obstacle-Avoidance and Path-Planning behaviour are weighted heavily compared to Goal-Seeking behaviour. This causes the virtual agent to avoid local minima in the presence of U-shaped obstacles (Figure 4.32(a)). Similarly, in Case II the virtual agent was able to avoid local minima in the presence of narrow passage obstacles as in Figure 4.33(a). In Case III, appropriate weights are produced to generate safe navigation in the presence of two walls shown in Figure 4.34(a).

The performance analysis reveals that the Fuzzy-ASM of motor schemas pro-
Chapter 4. Virtual Agent Navigation

vides the best performance in all of the three cases. The Fuzzy-ASM has shown a capability of escaping from a dead-end, is robust to handle environments with a narrow passage and is without collision in two wall environments. Table 4.10 illustrates that for successful safe navigation BW1 produces higher values of $d_o$ and $\bar{p}$, and lower values of $d_i$ and $t_o$ than Behaviour Weight 2 and 3. As a result, this approach produces relatively safe navigation, smooth trajectory, lower traveled distance and less navigation time as compared to Behaviour Weights 2 and 3.

2. Behaviour Weight 2

Behaviour Weight 2 in Cases I (Figure 4.32(b)), II (Figure 4.33(b)) and III (Figure 4.34(b)), show the virtual agent is trapped in a local minimum and fails to reach the target. For all cases, the virtual agent seems to struggle to pass obstacles towards a desired target and gets trapped in a loop. The main reason was that the agent keeps repeating turn angle which lead the virtual agent to just move left, right and left continuously. In these cases, the strengths of the weights alternates and this causes inconsistency in the motion commands to the virtual agent.

This method leads to unsuccessful navigational tasks in Cases I, II and III, where
the virtual agent fails to reach the target location. As a result, performance criteria are not evaluated for these cases.

3. **Behaviour Weight 3**

In Behaviour Weight 3 the behaviours are weighted equally. The resultant behaviour weights fail to generate a safe heading direction since Obstacle-Avoidance behaviour is weakened by Goal-Seeking and Path-Planning behaviours. Therefore, in Case I (Figure 4.32(c)) the virtual agent exhibits oscillatory trajectories and in Case III (Figure 4.34(c)), the virtual agent experienced a collision with an obstacle at point $P$. However, the reduced influence of obstacle-avoidance also leads to successful virtual agent navigation in Case II, where the virtual agent navigates through closely spaced obstacles (Figure 4.33(c)).

### 4.6.5.2 Comparison with Other Fuzzy Methods.

Two other fuzzy methods have been used for comparison and these are the Fuzzy Potential Field (FPF) [Makita 94, Katoh 04] and Roadmap (FRM) [Sanchez 04, Lee 04b] methods. The FPF method is based on an artificial potential field, which is used extensively for obstacle avoidance. FRM is a sensor-based version of the probability road-map method and is used to exploit the information obtained from sensors and to compute a feasible collision-free path.

Figure 4.36 shows an example of navigation path produced by all three methods. All three methods produced smooth paths which did not contain any sharp turns and did not collide with any obstacle. Figure 4.37 shows nine test results with each test using different goal locations. The Fuzzy-ASM produced a shorter distance compared to the FRM and FPF. The Fuzzy-ASM was an average of 6.33% shorter than FRM, and an average of 11.59% shorter than FPF. This showed that the Fuzzy-ASM required less time to reach the goal compared to the other two methods.

### 4.7 Summary

A new deterministic approach to resolve the behaviour conflicts in a complex situation during virtual agent navigation is developed and validated in the experiments. The proposed fuzzy $\alpha$ - level level method (Section 3.4) has been used in Fuzzy Controller and Action Selection module as in Figure 4.4. The number of navigation rules is drastically reduced without reducing the size of input and is purely sensor based for
building navigation rules. This approach has a modular based structure. Hence, the decoupled nature of the rules or sensor data significantly reduces the number of rules needed for navigation. Path-Planning behaviour helps to identify a local minima situation in real-time without any form of learning of the environment. The behavioural selection module has to decide which behaviour task needs to be executed when local minima problems occur. The behaviour rules are adapted easily and the virtual agent has deviated with minimum distance when it encountered obstacles.

In this chapter we described and demonstrated the capability of our virtual agent navigating in various unknown virtual environments. The testing was divided into four parts: (a) moving towards the goal; (b) escaping from local minima; (c) navigating in complex environment; and (d) comparison with other methods. The aim of the testing was to evaluate the performance of our fuzzy method in terms of robustness and quality of path generated by the virtual agent.

In general, the virtual agent is robust enough to handle different uncertainty levels in various types of environment. It also meets all the basic skills required in order to navigate in unknown environments. In complex environments such as in dead-end situations, mazes and cluttered environments, the virtual agent successfully reached
the goal. The qualities of path produced are reasonably smooth, short and clear of collision. The method also produced a better performance compared to other fuzzy methods used in this testing in terms of safety and speed.
Chapter 5

Virtual Agents in Computer Games

"Computer games don’t affect kids, I mean if Pac Man affected us as kids, we’d all be running around in darkened rooms, munching pills and listening to repetitive music."
Marcus Brigstocke (b.1973)
English comedian

"We want computer games to move people emotionally, like a great piece of art, a great movie or a great piece of music."
Neil Young (b.1945)
Canadian Singer and Guitarist

5.1 Introduction

There are several orthogonal dimensions along which agent applications could be classified. They can be classified by the type of the virtual agent, by the technology used to implement the agent, or by the application domain itself. In this chapter, we describe how the proposed method can be applied in different domains. Domains such as decision support in financial forecasting [Collan 03], knowledge management [Elst 03] and computer games [Sanornoi 04] have been identified as potential examples for application of our architecture. We choose to use the computer game domain, since this view fits best with the proposed method and available expertise. The same general framework and architecture is used with minor modifications and tuned to suit the game design. The same fuzzy control system and action selection method for behaviour selection has been used. The main differences are in the fuzzy rules and the virtual agent behaviours since computer games have different requirements compared to autonomous virtual agent navigation.
In the domain of games, classical rule-based systems often fail to attain subtlety, fuzzy rule-based systems allow the nuances among inputs to be captured and further reflected in the decisions at relatively low computational cost [Yifan 04]. With fuzzy logic it is also easier to write logic for reasoning with probabilities and the resulting probabilities take into account all the rules, not just the first/best. Fuzzy logic plays an increasingly important role in computer games (described in Section 5.2), yet it still has gaps compared with other popular methods which have been used in developing computer games.

5.2 Background

Computer games can be traced back to the 1950s when computers were in their early stages. Early games were text oriented with simple user interfaces. Since then, computer games have evolved into highly sophisticated computer software. Computer games are programs that enable a player to interact with a virtual game environment for entertainment and fun. Each game has its own strategy, action, curiosity, challenge and fantasy that make each game unique and interesting, which can motivate game players [Hsu 06].

Although computer games mainly provide entertainment and fun to the user, games have long been a popular area for academic research in AI. This is because games are challenging yet easy to formalize. They can also be used as platforms for the development of new AI methods and for measuring how well they work. In addition, games can demonstrate that machines are capable of behaviour generally thought to require intelligence without putting human lives or property at risk [Miikkulainen 06]. However, the main question is whether and to what extent, AI techniques can be applied to modern computer games, since most of the games involve four main issues, which are [Nareyek 00]:

- **Real Time** - There is only very limited time for reasoning.
- **Dynamics** - Computer games provide a highly dynamic environment.
- **Incomplete Knowledge** - A game character generally has incomplete knowledge of the world.
- **Resources** - The game character’s/world’s resources may be restricted.
Nowadays there is a trend toward creating a virtual environment which is populated with distinctive characters or virtual agents. Whether these characters are faceless guards in a top-secret facility, players on a football pitch, or evil plumber-battling princess-kidnapping despots, they can all be viewed as examples of (more or less) intelligent virtual agents. The field of agent building in AI is very wide ranging, incorporating robotics, simulation, philosophy, vision and so on. Computer games need the application of these techniques if they are to increase the intelligence of their characters, and consequently the appeal and quality of the games [Lent 99]. The use of autonomous characters has contributed to two main issues [Pisan 02]:

- what should the general nature of this virtual agent be for interesting game playing; and

- what type of architecture will best facilitate such characters and environments?

Another question when we deal with virtual agents is regarding goal-directed behaviour. The common approach to implementing this behaviour is using a predefined behaviour pattern, for example, IF-THEN rules. Recently, learning or adaptive behaviour has been introduced, such as neural networks [Cho 05], genetic algorithms [Hussain 06] and reinforcement learning [Merrick 06]. However, the pure reactive property has still not been overcome and we still struggle with real-time responsive behaviour because of some limitations in algorithms and restrictive processor availability.

One of the most recent works is the D-FSM (Dynamic Finite State Machine) method by [Yoon 07]. The method collects and analyzes the action patterns of game players. The game player patterns are modeled using a FSM (Finite State Machine). The results obtained by analyzing the data on game players is used for creating NPCs (Non-Player Characters) which show new action patterns by altering the FSM defined previously. These characters are adaptable NPCs which can learn the action patterns of game players. Unfortunately, there is no performance information yet reported. Other researchers, such as [Hicks 04], apply a Bayesian Network for multi-source data fusion to achieve the situational awareness that supports C2 decision making, and [Gorman 06] describe an approach to the imitation of strategic behaviour and motion; and propose a formal method of quantifying the degree to which different virtual agents are perceived as humanlike.

Intelligent virtual agents and fuzzy logic are two techniques that can improve the quality of interaction in computer games. Virtual agents present a new architecture
in game design which contribute to more flexible interaction, and at the same time fuzzy control offers a practical method for generating subtle behaviour [Yifan 04]. Virtual agents with subtle behaviour enhance the perceived complexity, enjoyability and credibility of the virtual environment.

One of the earlier works is the development of the BattleCity.net game by [Yifan 04]. The game uses a BDI-style framework but lacks any human interaction, which is one of the important aspects of most computer games [Shaout 06]. [Sanornoi 04] have developed intelligent virtual agents using a behaviour-based control approach. Unfortunately, the overall architecture and behaviour selection has not been described in detail.

Close Combat and S.W.A.T. 2 [Woodcock 07] are examples of successful implementations of fuzzy logic in commercial computer games. The fuzzy logic is used for action selection that the soldier needs to perform. To prevent inconsistency in the selection of actions (in Close Combat, the soldier decides to go prone, then the next instant decides to stand up, then go prone, etc.) a series of weights are associated with good behaviour. In the case of bad behaviour, the soldier is restricted from choosing a good behaviour action until certain conditions or a time limit has been met. During the development, the main problem is in the balancing of the game engine. Often this just involves adding more parameters to the engine to account for the new circumstances. However, several attempts are required for an adjustment to the current values by putting more weight on one or more parameters. This subsequently can cause other behaviours to get out of balance given slightly different circumstances.

5.3 Pacman

Pacman is an arcade game made by Namco in 1981 [Namco 07] and is one of the best selling coin operated games in history. The purpose of the game is to move the Pacman through a small maze collecting every dot and scoring points, and after collecting every dot the player advances to the next level. Ghosts spawn in the middle of the map and try to catch Pacman. On contact with a ghost, Pacman will die and the level will be restarted with the remaining dots. A limited number of power pills are found in the map and when eaten will reverse the roles of Pacman and Ghosts, allowing Pacman to eat them and score extra points. A few times in a level, fruit will appear and if eaten gives extra points. At first glance, it looks like a simple game; however playing well requires advanced strategies [KiLLerCloWn 07]. It was not until 1998 that Billy
Chapter 5. Virtual Agents in Computer Games

Mitchell played the perfect game achieving a score of 3,333,360 and taking 6 hours to complete it. He beat all 256 screens eating every dot, fruit, and ghost (all four ghosts were eaten with each power pellet) using only one life.

Game artificial intelligence can be designed without the use of fuzzy techniques. For most versions of Pacman, including the original, they used crisp control mechanisms. The deterministic logic that controlled the ghosts caused them to react consistently in predictable ways to player action. They did not learn from previous player performance and consequently adjust the level of difficulty. To create the illusion of intelligence, the original implementation of Pacman had special (crisp) rules for each of the four ghosts. For instance, certain ghosts would try to approach Pacman from different sides. Although this allowed the ghosts to behave more realistically, it undoubtedly added to the size and complexity of the code.

There is a small amount of work that has been conducted toward the application of AI to Pacman or similar games. The work can be divided into two areas, which are approaches for a self-playing Pacman system (artificial virtual agent that learns to play the Pacman game) and approaches for controlling and optimization of the Ghosts in Pacman.

5.3.1 Self-playing Pacman

One of the earlier works used genetic programming to demonstrate the task prioritization in the Pacman agent [Koza 92]. The approach relies on the set of predefined control primitives for perception, action and program control. It uses a population of 500, and a fitness function based on the score. The resulting algorithm was successful, achieving a score of 9,220 points eating all the food and finishing the first level. However, this approach only works in one particular maze and is not a virtual agent approach that would work in a different maze.

Pac-Tape (Self-playing Pacman system) was developed by [Gugler 97], which used the original Pacman arcade game emulated on a desktop PC. The approach is based on a brute force search, but has not been described in detail or tested [Gallagher 03]. On the other hand, [Lawrence 99] used the same Pac-Tape by applying a genetic algorithm (GA). Unfortunately, due to poor interaction between the genetic operator and the representation used, the system was not very successful [Gallagher 03]. Similar work by [Bonet 99] applied an embodied intelligence approach to learn playing strategies in Pacman. They used a creature centered perception network with reinforce-
ments based on previous rewards (reinforcement learning). Both the Ghost and Pacman agents were evolved, starting from simple boards and working up to a complete one. The representation of the world state was multiple 2D bit arrays representing different items in the game. There has also been some work on learning routes for Pacman, but this approach is not viable for Ms.Pacman [Lucas 05].

A simple state machine and a parameter rules set, with a population based on an incremental learning (PBIL) algorithm was developed for artificial virtual agents that learn to play a simplified version of Pacman [Gallagher 03]. The representation had very serious limitations for scaling up the intelligence of the virtual agent, which might contribute to a very high dimensional optimization problem. As a result, the system required a large amount of computational time to allow enough generations for the algorithm to produce a good result. [Lucas 05] and [Yannakakis 05] describe an approach of evolving a Pacman playing agent based on evaluating a feature vector for each possible next location given by the current location of Pacman. A neural network was used as the control algorithm. The experimental results showed that useful behaviours can be evolved that are frequently capable of clearing the first level, but still at risk of making a poor decision.

More recently [Gallagher 07] developed a Pacman agent that learned game-play based on minimal on-screen information which was based on an evolutionary algorithm which is a neural network. The result showed that this neuroevolution is able to produce a virtual agent that displayed novice playing ability, with the minimum amount of on-screen information, no knowledge of the rules of the games and a minimally informative fitness function. Unfortunately, no virtual agent was able to clear a maze of dots during the experiment. The performance is lower than previous approaches in [Gallagher 03] and [Lucas 05] and the results are inconclusive with respect to the influence of a number of system parameters.

Alternatively [Szita 07] use reinforcement learning by defining a set of high-level observation and action modules, from which rule-based policies are constructed automatically. In these policies, actions are temporally extended, and may work concurrently. The policy of the agent is encoded by a compact decision list. The components of the list are selected from a large pool of rules, which can be either hand-crafted or generated automatically. A suitable selection of rules is learnt by the crossentropy method, a recent global optimization algorithm that fits the framework smoothly. Crossentropy-optimized policies perform better than our hand-crafted policy, and reach the score of average human players.
5.3.2 Intelligent Control of The Ghost Behaviour

There is a small amount of work in intelligent control of the ghost behaviour or non-player characters in the Pacman game. The evolutionary algorithm used by [Kalyanpur 01] optimized the genes of ghosts by a combination of genetic algorithms and a neural network. The genes of the ghost were an array representing the intersections of corridors in the map whose values contained the direction the ghost would turn. The back-propagation algorithm with neural network is used to determine best values for mutation and crossover probability given success times. Although this strategy was somewhat successful, the ghosts had predetermined moves, and therefore possessed no real-time decision-making or task prioritization.

Other work explored the possibilities of real-time behaviour of live animals for the Pacman game [Eck 06]. Instead of computer code, they used animals controlling the ghosts. The real maze for the animals to walk around had been built, with its proportions and layout matching the maze of the computer game. The position of the animals in the maze is detected using colour-tracking via a camera, and linked to the ghosts in the game. This way, the real animals are directly controlling the virtual ghosts. During the tests, one of the crickets stopped moving, and it was shedding its skin. The cricket’s new skin was very light, and therefore it did not get detected by colour tracking anymore. After about half an hour the cricket’s skin turned dark enough to be noticed by the colour tracking system, resulting in five ghosts being put in the game.

5.4 The Framework

The original Pacman for MATLAB was developed by [Bauerbach 04]. It is designed so that the same system can be easily modified to be used with other games. In our implementation the original Pacman game environment is used, the main difference is the game engine had been modified to integrate with our proposed fuzzy architecture.

In Figure 5.1, we present a proposed overall framework for the Pacman game. The framework can be divided into three main parts which are the virtual agent (ghost), Fuzzy Controller and Control Tread. The virtual agent (ghost) acts as an interface between the Fuzzy Controller and the Control Tread. The Control Tread and fuzzy behaviour is adapted from [Shaout 06]. The fuzzy controller is based on FAM as described in Chapter 3. It takes the new representation and feeds it through the network.
The output consists of 4 real values representing directions. The direction with the highest value is taken as the execution for the corresponding movement control. This information is sent to the game control system. Because these controls are the same as those used in our proposed reactive architecture, nothing needs to be changed in the game control system.

![Fuzzy Controller Diagram](image)

**Figure 5.1: Game Overall Framework**

The next section describes the Control Thread (Section 5.4.1), fuzzy behaviour and fuzzy variables (Section 5.4.2) of the [Shaout 06] Pacman game which was used in this implementation.

### 5.4.1 Control Thread

The Control Thread [Shaout 06] is the main interface between the player and the game and executes most of the game logic. The Control Thread is a large two-dimensional array of integers defining the map and a set of Ghost Objects and Thing Objects with grid associated positions. This map information needs to be interpreted in a way that can be used as input to the Fuzzy Controller and be updated as objects move in the game. The representation for the Fuzzy Controller focuses on a square grid, centered on the ghost's position.

The Control Thread checks if a new level needs to be initialized when a level ends or a new game has begun. It also checks if Pacman is powered-up. Pacman becomes temporarily powered-up after eating special power pellets, which enable him to eat the ghosts. The Thread also checks for a collision between Pacman and a ghost. This happens when Pacman and a ghost occupy the same location on the game grid or
Pacman and a ghost have swapped positions. The Control Thread also handles the event of Pacman losing a life. Pacman loses a life if the collision flag is set and the power pellet time is 0. The number of lives is decremented by 1. After that, the ghosts and Pacman are reset to their initial starting positions for the level.

The ghost must move in a direction that will take it towards the area of the map with the highest pellet density. The following steps are required to find this area:

1. Divide the map into nine overlapping sections based on combinations of the following fractions of the x size and y size of the level map 0 to $\frac{1}{2}$, $\frac{1}{4}$ to $\frac{3}{4}$, and $\frac{1}{2}$ to 1;
2. Sum the total number of pellets in each section;
3. Select the section with the highest number of pellets;
4. Return the coordinates of the middle pellet in that section:

   The middle pellet is found by traversing the pellets in that section from left to right, top to bottom and stopping when the number of pellets encountered is half the total number of pellets in that section.

In order to determine the direction of the shortest path the A* algorithm [Matthews 02] has been used as follows:

1. Calculate the city-block distance between the source ghost and the other ghosts;
2. Select the ghost that is closest to the source ghost;
3. Determine the differences in x and y location between the source ghost and the closest ghost;
4. If the x difference is greater or equal to the y difference:
   - If the square in the x direction from the source ghost away from the closest ghost is not blocked, return the direction from the source ghost to that square;
   - Else if the square in the y direction from the source ghost away from the closest ghost is not blocked, return the direction from the source ghost to that square;
   - Else return one of the two remaining directions (whichever leads to a path that is not blocked);
5. Else the $y$ difference is greater than the $x$ difference:

- If the square in the $y$ direction from the source ghost away from the closest ghost is not blocked, return the direction from the source ghost to that square;
- Else if the square in the $x$ direction from the source ghost away from the closest ghost is not blocked, return the direction from the source ghost to that square;
- Else return one of the two remaining directions (whichever leads to a path that is not blocked).

5.4.2 Fuzzy Behaviour and Fuzzy Variables

The Fuzzy Controller contains four main behaviours which are hunting, defence, deploy and random [Shaout 06]. Each behaviour works independently to produce their behaviour weight ($\omega$). The behaviours can be described as below:

**Hunting** - the ghost will actively seek for Pacman;

**Defence** - the ghost will protect the area that has the most pellets;

**Deploy** - the ghosts will spread out and cover the entire level (Shy in [Shaout 06]);

**Random** - this approach is chosen when no other behaviour is a preferred choice, a ghost will randomly move about the level.

The fuzzy behaviour contains three main fuzzy variables which are distance, time and rate [Shaout 06].

**Distance Variables**

Figure 5.2 shows the membership functions of the distance variable with: Near, Medium, and Far. Two types of distance variables use the same membership functions which are the distance between Pacman and each of the ghosts; and the distance between every possible pair of ghosts.
Chapter 5. Virtual Agents in Computer Games

Figure 5.2: Membership Functions for Distance.

Time Variables

Figures 5.3 and 5.4 show the membership functions for pellet time and average lifetime with Short, Medium, and Long. Two types of time variables are used which are a measurement of the amount of time since Pacman has eaten a pellet, and the average period of time that Pacman has gone without losing a life.

Figure 5.3: Membership Functions for Pellet Time

Rate Variables

The membership function for pellet consumption rate is show as in Figure 5.5. Three linguistics variable types are defined for the pellet rate which are Good, Medium and Poor. The pellet rate represents the ratio of the number of pellets eaten to the number of ticks that have passed since the game started.
5.4.3 Behaviour Selection

Behaviour selection is based on our proposed method described in Section 3.4. The $\alpha$ values from each behaviour, and the behaviour weight, $\omega$, for each behaviour can be calculated using equation (3.36). Once the behaviour weight value is determined for each behaviour, the final action is selected based on the Hurwicz criterion using equation (3.37).

5.5 Implementation

The original features of the Pacman game have still been used in order to reduce any confusion for the user. Figure 5.6 shows an example of the Pacman game screen design.

Figure 5.7(a) shows an example of Pacman game using a low weight factor ($\sigma = 0.2$). The Ghosts moved randomly and were searching for Pacman, since they do not
know the exact location of Pacman. Random behaviour was selected and the Ghost’s will keep searching for the Pacman. If one of the Ghosts find Pacman, its behaviour will be changed to hunting behaviour.

Then again, in Figure 5.7(b), by having a high weight factor ($\sigma = 0.8$), the player must be aware that the Ghosts will aggressively move towards Pacman. This will make the game more challenging to the player. In some situations such as in Figure 5.8, the Ghosts might also be capable of trapping the player. This happens when the all the Ghosts move toward Pacman at the same time.
Chapter 5. Virtual Agents in Computer Games

5.5.1 Performance Evaluation

The aim of performance evaluation is to measure CPU utilization when running the Pacman game. The experiment used the same experiment setup as used by [Shaout 06]. The main reason of using this setup is to observe if there is any performance improvement using a Fuzzy method compared to original method. The original Pacman game [Bauerbach 04] and Fuzzy-ASM were run on a Pentium IV 2.6 GHz.

Figure 5.9 shows the result of the evaluation. There are some improvements in CPU utilization. For the easy level the Fuzzy version has shown a 25% improvement which is 1.8% less than the original method, 7.2% and 5.4% respectively. This is similar to the hard level where the fuzzy version (10.3%) had improved by 21.7% which is 3.5% less as compared to the original method (13.8%). This demonstrates that as the
degree of difficulty increases, the fuzzy version of Pacman gets significantly better performance as compared to the original method.

Similar issues have been identified as raised in [Shaout 06], where some of the fuzzy rules and behaviour weights need to be modified. This is to make sure that the game is not too difficult and the ghost behaviour is not too rigid. The main reason is to make sure the game is enjoyable and fun to play.

### 5.5.2 User Evaluation

The aim of user evaluation is to observe the player opinions of what they think about the ghosts behaviour and response. The eight criteria that have been used as proposed by [Shaout 06] are: difficulty levels (easy, medium and hard); predictability; responsive (feel); human-like; fun and overall impression.

Ten different players with various skill have been selected. They are required to answer a short questionnaire by rating each categorie on a scale of 1 to 10, where 1 is the lowest and 10 is highest.

![Player Ratings of Fuzzy vs. Original Pacman.](image)

Figure 5.10: Player Ratings of Fuzzy vs. Original Pacman.

The average score of each game in each category is presented in Figure 5.10. In
Chapter 5. Virtual Agents in Computer Games

In general, when compared with results in [Shaout 06], the trend is the same even though [Shaout 06] has given a better result in two of the criteria, hard and responsive.

The results also show that the players rated our version higher in each criterion compared to the original Pacman game. The Ghost in Fuzzy-ASM behaves more human-like and has a good response to the player which makes the game become more interesting. The game becomes more challenging and the player will have a different experience each time playing the game. As in [Shaout 06], the ghost has demonstrated capabilities of intelligent, rational entities, which humans would expect. In terms of the level of difficulties (easy, medium and hard), the Fuzzy Pacman has matched its level of difficulties with the player’s skills. The overall rating shows most of the players like to play with the Fuzzy Pacman because each of the ghosts has its own preferred behaviour and intelligence level, compared to the original Pacman which has one level of intelligence.

5.6 Discussion

The implementation has demonstrated how different components from [Bauerbach 04] and [Shaout 06] can be integrated easily with our fuzzy controller and behaviour selection method. Only minimum modification was needed while maintaining the original features of the Pacman game. For example, by changing some of the parameters so that each component can read the same values.

Fuzzy-ASM Pacman also reduces code complexity as compared to other methods described in Section 5.2 and . It is easy to add to or change ghost behaviour. For example, if new rules or behaviours need to be added, only the relevant component has to be modified. Other components can still work independently.

Compared to the original Pacman game in terms of performance evaluation, the Fuzzy-ASM Pacman game has shown better performance in CPU utilization which is lower than the original game. The main reason is that fuzzy rules are simpler to execute as compared to deterministic logic. Other methods such as learning algorithms and evolution algorithms require high CPU resources and need to be trained. They also have complex behaviour selection which requires more processing before any decision can be executed.
5.7 Summary

In this chapter we have examined how a proposed fuzzy logic method and intelligent virtual agent architecture can be applied to other domains. The Pacman game was used as an example implementation domain. The same architecture and Fuzzy-ASM developed for virtual agent navigation has been used. The main modification is in the fuzzy rules and membership functions. The performance evaluation and user evaluation also gave very promising results, even though full evaluation has not been conducted. This has given us an indicator that the proposed architecture is flexible and easily adapted to be used in other domains.
Chapter 6

Conclusion

Most, probably, of our decisions to do something positive, the full consequence of which will be drawn out over many days to come, can only be taken as a result of animal spirits.
- John Maynard Keynes (1883-1946)
British economist

6.1 Summary

This thesis presents the design of a control architecture for autonomous virtual agents in virtual environments. The main focus is to improve the performance of the reactive behaviour of virtual agents, with the intention that the virtual agents take their decisions continuously in real-time, according to internal and external factors. Indeed, virtual agents in virtual environments have to keep on choosing what to do next even after they have finished the specific task.

Our architecture is based on a fuzzy behaviour-based approach and Fuzzy Associative Memory (FAM) is used to optimize the fuzzy behaviour rules. Each behaviour will produce its own behaviour weight and this value will be used by the behaviour selection module for action selection. The action selection method is based on fuzzy $\alpha$-level with the Hurwicz criterion. The method will reduce the redundancy of calculating $m(m-1)/2$ pairwise comparisons to $m$ pairwise comparisons by the fuzzy subtraction operation. The behaviour rules containing $\alpha$ intervals of inputs and output spaces are easily integrated with a virtual agent.

In autonomous agent navigation, the experimental results show that our autonomous virtual agent had successfully navigated various virtual environments. It also eliminates the existing problems of autonomous virtual agents:
1. basic navigation for one obstacle, two walls, narrow passage and corner;

2. trap situations due to local minima; and

3. navigating in complex environments (cluttered and maze).

The experimental results in Section 4.6.1 show that the virtual agent had successfully fulfilled the requirement for basic navigation. It also showed our defuzzification method produced better results compared with the other two defuzzification methods. In Section 4.6.2 the virtual agent had escaped from a local minima and reached the goal, although time, path length and number of decisions may have varied for each test, which depends on the complexity of each local minima situation. Section 4.6.3 showed the results for complex environments. The results show the virtual agent had reached its goal successfully and was robust enough to handle those conditions.

In Section 4.6.5.1 we showed the performance results when using different behaviour weights. Behaviour Weight 1 produces a superior performance compared to Behaviour Weights 2 and 3. Behaviour Weight 3 is prone to being trapped in local minima and produces an oscillatory trajectory. Behaviour Weight 2 experiences the highest number of collisions leading to unsafe navigation. Besides, the agent trajectories produced in Behaviour Weight 2 produced high travel distances, high navigation time, and an inconsistent velocity.

When comparing FPF and FRM the results clearly demonstrated that our method produces better results, as in section 4.6.5.2. Similar results have also been recorded in Section 4.6.4. The results show our method required less decision making which means more reliable decisions had been generated and redundant decisions can be reduced. This will in turn reduce the processing time and help the agent to reach the goal quicker.

The quality of path from all the tests has shown that most of the paths produced were safe from collision with obstacles and reached the goal successfully. Paths produced were reasonably smooth even though there were some sharp turns in complex environments. Although the path length is not the shortest path, it does not divert too far from the potential shortest path. The shorter paths mean less time is required to reach the goal. When compared with FBF, FPF and FRM methods, our virtual agent still produced the smoothest and shortest path. In some cases sharp turns were necessary, especially for 90° corners and narrow passages, our method tried to minimize the sharp turns in order to produce smoother paths. However, it still produced better results when compared with other methods in the experiments.
In controlling the ghosts (agents) in the Pacman game, our fuzzy method had made the game more interesting and challenging. The level of difficulty in the game is based on player selection and capabilities. For example, for the easy level, less weight is given to hunting Pacman and for the harder levels, the weights are configured so the ghosts are more likely to hunt Pacman than to choose other behaviours.

The performance evaluation in Section 5.5.1, shows there are some improvements in CPU utilization since for the easy level of the fuzzy version of the Pacman game it was 15% better than the original method, similar to the hard level where the fuzzy version was 23% better compared to the original method. This demonstrated that the fuzzy method had better performance compared to the original method. For user evaluation in Section 5.5.2, we noticed that players rated the ghosts in the fuzzy version as more responsive and more human-like. It is also interesting to note that players felt the ghosts in the fuzzy system were more predictable. This is due to the fact the ghosts were designed as intelligent, rational entities, meaning they demonstrate logical behaviours that humans would expect.

In general, the experimental results from both implementations show how our fuzzy architecture can be used for agent control, action selection and escape from local minima in two different application domains. It has produced better performance compared to other fuzzy methods that have been used in both applications. The virtual agent is robust enough to handle different uncertainties in various types of environment. In agent navigation, it meets all the basic skills required in order to navigate in unknown environments. Then again, in the Pacman game we rectified many of the deficiencies found while maintaining code simplicity. The architecture allows rules to be altered or created and then integrated into the control logic with a change in only a single line of code. The rules can be changed independently, and all variables are always scaled to a common range.

During the implementation and evaluation, we have identified the following limitations:

1. The virtual agent navigation task depends on the complexity of the environment. For example, types of local minima, distance between two walls and narrow passages.

2. The minimum distance between walls and/or obstacles in a maze and cluttered environment must not be less than 5 grid squares.

3. Shape of obstacle, such as disc or triangular prism has not been tested, in order to
see how it can effect the virtual agent during navigation.

4. Information about pattern recognition and noisy imaging has been ignored in this implementation. In the real world the environment is very different compared to a virtual environment. An implementation using a physical agent such as a robot would help measure the robustness of our method in the real world.

5. There is no interaction between virtual agents in the Pacman game implementation. The virtual agents behave individually and conflict between virtual agents has not been tested.

### 6.2 Contributions

The work described here has made a number of contributions to the study of reactive behaviour for autonomous agents, especially on the implementation of behaviour-based architecture and action selection methods based on fuzzy logic. These contributions are summarized below.

**Behaviour-based Architecture**

1. A practical solution to the heuristic design and implementation of flexible and uncertainty tolerating agent behaviour using behaviour-based fuzzy logic.
   
   - The interface is simple, thus speeding up the development life cycle.
   - The system is more reliable in terms of fault tolerance.
   - The experience of human reaction to the environment is used to derive fuzzy reasoning rules.

2. A new solution to the problem of behaviour coordination in behaviour-based architecture.
   
   - The fuzzy controller with a behaviour selection module is based on the self-reaction of an agent in the environment. It can effectively be used to control an agent based on the different sensing information.
   - Control is more versatile, in the sense that this method facilitates the simultaneous use of several controllers/behaviours based on different techniques (each with its own errors and response time depending on the problem state).
Chapter 6. Conclusion

3. The architecture is generic and applicable where decision making with uncertainties is required and is not limited to any specific domain. This can be done with some modifications in its architecture to suit the implementation domain requirements. This can be seen in how we implement this architecture in virtual agent navigation (Chapter 4) and in a computer game (Chapter 5).

Action Selection Method

1. Formulation of action selection mechanisms based on multiple behaviour decision making using a Fuzzy $\alpha$-level approach with Hurwicz criterion.

   - The method considers the loci of left and right spreads at each $\alpha$-level of a group of fuzzy numbers and the horizontal-axis locations of the group of fuzzy numbers based on their common maximizing and minimizing barriers, simultaneously. The ranking method combines the above techniques with the summation of interval subtractions as an area measurement to make them more effective and efficient compared to the existing ranking methods that use only one of $\alpha$-cut, Hamming distance, left/right score, centroid index or area measurement techniques.

   - The method for $m$ fuzzy numbers uses only $m$ comparisons to the same referential rectangle as opposed to the $m(m - 1)/2$ pairwise comparisons needed by existing methods.

   - The method uses very few $\alpha$-cuts such as 3 or 4 $\alpha$-cuts and uses the summation of each $\alpha$-level interval which does not require normalization to measure the summation for the ranking order of the fuzzy numbers.

   - The final behaviour selection has been done using the Hurwicz criterion. It takes into account the optimistic and pessimistic view of different behaviours for various agent tasks, which is something in between the maximin and maximax solution. From the overall behaviour selection, a Hurwicz criterion is derived using an optimism-pessimism index.

2. Continuous real-time decision-making

   - Autonomous virtual agents will keep on making decisions according to their internal and external factors. The virtual agent will choose the next action, i.e. its task is not finished once a specific task is solved.
• Indeed at certain moments in time, the user cannot control the virtual agent because he is not always present in the virtual environment. Therefore the virtual agent has to be able to take its own decisions when it is not employed in specific tasks such as interacting with users.

• Experiments have demonstrated the virtual agent architecture to be capable of handling various types of situations in different domains. In these cases, virtual agents had a reasonable level of autonomy and reactive behaviour.

3. Fuzzy decision making has been used rarely in autonomous virtual agents. This implementation has shown that the method is simple and easy to integrat with autonomous virtual agents.

Uncertainty Handling

1. Decision making under uncertainty

• The action selection method reflects both subjective judgment and objective information for fuzzy decision making problems in real life situations. In essence, determination of weights is objective and automatic.

• Therefore, the final decision results are relatively reasonable and reliable. This may present a new way to solve fuzzy decision making problems under complex environment conditions.

2. New solution for autonomous virtual agent escape from local minima situations.

• The local minima solver is based on the characteristics of the self-reaction of a virtual agent in the environment. The agent can recognize its trapped state (infinite loop), where the virtual agent oscillates between two points.

• The algorithm is simple and works well in both implementation domains.

3. A Dempster-Shafer theory has been used to integrate visual sensor and model information.

• It is interval based, as defined by the upper and lower probability bounds which allow a lack of data to be modeled adequately. Thus, this method no longer requires a full description of conditional (or prior) probabilities and small incremental evidence can be adequately incorporated.
Chapter 6. Conclusion

- The building of occupancy maps is well suited to path planning and obstacle avoidance.

6.3 Further Work

Our work on the behaviour-based architecture and action selection method using fuzzy logic is just a start. There are a number of improvements and extensions that could be done to the work described in this thesis. Some of the work which can be pursued in the future is as follows.

1. Hybrid system

Ordinary fuzzy systems do not have the capability to learn from examples but the membership functions can easily be formed. By using a hybrid architecture both of the above characteristics can be utilized. In hybrid architectures the integration of reactive system components with deliberative planning components allow long term planning. The important issue is to investigate the role of our method for multiple behaviours in hybrid architectures. In particular how can a planning component be integrated with our architecture and action selection method for agent behaviour selection?

2. Type-2 Fuzzy Logic Controller (Type-2 FLC)

Type-2 FLCs are an attractive alternative because they can cope better with modeling uncertainties. Unfortunately, type-2 FLCs are computationally intensive. The main challenge is how our method can be used with Type-2 FLCs to reduce the computational burden by providing faster type-reduction methods and a simpler architecture.

3. Action Selection Mechanism

There are similarities between decision making under uncertainty and multi-behaviour (multicriteria) decision making problems, two areas which have been developed in almost completely independent ways until now. The multicriteria decision problem is usually viewed in these models as the joint satisfaction of the set of criteria, with or without compensation between the levels of satisfaction, taking into account the levels of importance of the criteria [Dubois 00]. The main problem for action selection for multiple behaviour is a combination of complexity for each behaviour. Furthermore, all computation takes both time
and space (in memory), agents cannot possibly consider every option available to them at every instant in time. The challenge is how our action selection method can solve this problem. Additionally, by implementing Fuzzy-ASM in domains such as decision support for finance, knowledge management and medical applications we could investigate how this method can benefit them by producing more accurate results.

4. Multi-Agent System (MAS)

There is some need for an application which requires multiple agents that can work together. A multi-agent system is where agents interact to solve problems that are beyond the individual capacities or knowledge of each problem solver. Some of the major issues of MASs are that each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; there is no system global control; data are decentralized; and computation is asynchronous [Sycara 98]. There is potential in how our method can be expanded in this type of system for example in Beliefs, Desires, and Intentions (BDI) architectures.

5. Behaviour Animation

Behavioural animation is a type of procedural animation, which is a type of computer animation. In behavioural animation an autonomous character determines its own actions, at least to a certain extent. This gives the character some ability to improvise, and frees the animator from the need to specify each detail of every character’s motion. Furthermore, in behavioural animation, virtual agents acquire capabilities of perceiving their environment, are able to react and make decisions, depending on this input. The problem is how to populate virtual environments with virtual agents so that they can behave autonomously.

6.4 Final Remark

The thesis shows a novel architecture for reactive behaviour for autonomous agents using behaviour-based fuzzy logic. The two implementation domains are examples of how this architecture can be used, and how it might be implemented in other domains such as financial analysis, decision support systems etc. We hope that this work provides a step towards exploring this fascinating area.
Bibliography


Marcus Gallagher & Mark Ledwich. *Evolving Pac-man player: can we learn from raw input?* In IEEE Symposium on Computational Intelligence and Games, pages 282–287, Honolulu, Hawaii, USA, 2007. IEEE.


<table>
<thead>
<tr>
<th>Reference</th>
<th>Author</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>Details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Bibliography


Z. M. Ma, Froduald Kabanza, Khaled Belghith & Roger Nkambou. *Towards a robot path planner for dangerzones and desirezones*. In Tenth IASTED International Conference on
Bibliography


Bibliography


Bibliography


<table>
<thead>
<tr>
<th>Bibliography</th>
<th>150</th>
</tr>
</thead>
</table>
Bibliography


Bibliography


D. Weyns, K. Schelthout & T. Holvoet. Exploiting a virtual environment in a real-world application. In Environments for Multi-Agent Systems II. Second International Workshop,


Bibliography


Appendix A

Action Selection Method

Test Success Rate

(a) Wang Method

(b) Fuzzy-ASM

Test 1

Test 2
Appendix B

Behaviour Rules

The following list of rules contributes to the hunting behavior:

• **IF pacman_near AND skill_good, THEN hunting_behavior**
• **IF pacman_near AND skill_med AND pellet_med, THEN hunting_behavior**
• **IF pacman_near AND skill_med AND pellet_long, THEN hunting_behavior**
• **IF pacman_med AND skill_good AND pellet_long, THEN hunting_behavior**
• **IF pacman_med AND skill_med AND pellet_long, THEN hunting_behavior**
• **IF pacman_far AND skill_good AND pellet_long, THEN hunting_behavior**

The following rules relate to the defence behavior:

• **IF pacman_far AND skill_bad AND ghost_far AND pellet_short, THEN defence_behavior**
• **IF pacman_far AND skill_bad AND ghost_far AND pellet_med, THEN defence_behavior**
• **IF pacman_far AND skill_bad AND ghost_med AND pellet_short, THEN defence_behavior**
• **IF pacman_far AND skill_bad AND ghost_med AND pellet_med, THEN defence_behavior**
• **IF pacman_far AND skill_med AND ghost_far AND pellet_short, THEN defence_behavior**
• **IF** pacman\_med AND skill\_bad AND ghost\_far AND pellet\_short, **THEN** defence\_behavior

The following rules are associated with the *deploy* behavior:

• **IF** pacman\_far AND skill\_bad AND ghost\_near AND pellet\_short, **THEN** deploy\_behavior

• **IF** pacman\_far AND skill\_bad AND ghost\_near AND pellet\_med, **THEN** deploy\_behavior

• **IF** pacman\_far AND skill\_bad AND ghost\_med AND pellet\_short, **THEN** deploy\_behavior

• **IF** pacman\_far AND skill\_bad AND ghost\_med AND pellet\_med, **THEN** deploy\_behavior

• **IF** pacman\_far AND skill\_med AND ghost\_near AND pellet\_short, **THEN** deploy\_behavior

• **IF** pacman\_med AND skill\_bad AND ghost\_near AND pellet\_short, **THEN** deploy\_behavior

The following rules are related to the *random* behavior:

• **IF** NOT (hunting\_behavior) AND NOT (deploy\_ghost\_behavior) AND NOT (defence\_behavior), **THEN** random\_behavior

The following fuzzy rules are used to define the player skill variables:

• **IF** TimeLife is Short OR PelletRate is Poor **THEN** Skill = Poor

• **IF** TimeLife is Medium OR PelletRate is Medium **THEN** Skill = Medium

• **IF** TimeLife is Long AND PelletRate is Good **THEN** Skill = Good