

# Imitation Learning through Games: Theory, Implementation and Evaluation

Bernard Gorman, Bachelor of Science  
in Computer Applications

A dissertation submitted in fulfilment  
of the requirements for the award of  
Doctor of Philosophy (PhD) to the



Dublin City University Faculty  
of Engineering and Computing,  
School of Computing

Supervisor: Dr. Mark Humphrys

22 January 2009

# Declaration

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

Signed: 

Student ID: 99012588

Date: 22 January 2009

# Contents

|   |            |
|---|------------|
| ABSTRACT .....                                    | 5          |
| ACKNOWLEDGEMENTS .....                            | 8          |
| <b>1 INTRODUCTION.....</b>                        | <b>9</b>   |
| 1.1 INTRODUCTION .....                            | 9          |
| 1.1.1 Publications .....                          | 9          |
| 1.2 ARTIFICIAL GAME INTELLIGENCE .....            | 10         |
| 1.2.1 Games Industry .....                        | 10         |
| 1.2.2 Academia.....                               | 14         |
| 1.2.3 Machine Learning.....                       | 17         |
| 1.2.4 Games as Imitation Research Platforms ..... | 18         |
| 1.3 CONTRIBUTIONS .....                           | 22         |
| 1.3.1 Research Questions.....                     | 22         |
| 1.3.2 Contributions: Our Research .....           | 22         |
| 1.3.3 Thesis Outline .....                        | 24         |
| 1.4 CONCLUSION .....                              | 27         |
| <b>2 IMITATION, BEHAVIOUR AND AI.....</b>         | <b>28</b>  |
| 2.1 INTRODUCTION .....                            | 28         |
| 2.1.1 Publications .....                          | 28         |
| 2.2 IMITATION LEARNING .....                      | 28         |
| 2.2.1 Biological Imitation .....                  | 29         |
| 2.2.2 Application in Robotics .....               | 39         |
| 2.3 GAMES-BASED IMITATION LEARNING .....          | 53         |
| 2.3.1 Motivation and Existing Work.....           | 53         |
| 2.3.2 Believability and Performance.....          | 56         |
| 2.3.3 Computational Modelling for Games .....     | 58         |
| 2.3.4 Selecting a Testbed: Quake 2 .....          | 60         |
| 2.4 CONCLUSION .....                              | 63         |
| <b>3 THE QASE API .....</b>                       | <b>65</b>  |
| 3.1 INTRODUCTION .....                            | 65         |
| 3.1.1 Publications .....                          | 66         |
| 3.2 MOTIVATION .....                              | 66         |
| 3.2.1 Existing APIs.....                          | 67         |
| 3.3 QASE, ENVIRONMENTS AND INTERFACES.....        | 72         |
| 3.3.1 The Quake 2 Environment.....                | 74         |
| 3.3.2 QASE Network Interface.....                 | 79         |
| 3.4 QASE AGENT ARCHITECTURE.....                  | 92         |
| 3.4.1 Bot Hierarchy.....                          | 92         |
| 3.4.2 Gamestate Augmentation .....                | 95         |
| 3.4.3 Team-Based Play .....                       | 97         |
| 3.4.4 DM2 Parser & Recorder.....                  | 97         |
| 3.4.5 Environment Sensing.....                    | 98         |
| 3.4.6 Fixed Entity Sensing.....                   | 104        |
| 3.4.7 Inbuilt AI Constructs .....                 | 104        |
| 3.4.8 Waypoint Maps .....                         | 106        |
| 3.4.9 MatLab Integration.....                     | 109        |
| 3.5 CONCLUSION .....                              | 111        |
| 3.5.1 Adoption in Academia .....                  | 112        |
| <b>4 IMITATION LEARNING .....</b>                 | <b>113</b> |
| 4.1 INTRODUCTION .....                            | 113        |

|          |   |            |
|----------|---|------------|
| 4.1.1    | <i>Publications</i> .....                                     | 114        |
| 4.2      | BEHAVIOUR MODEL .....   | 115        |
| 4.2.1    | <i>Behaviour and Believability</i> .....                      | 117        |
| 4.3      | STRATEGIC BEHAVIOUR IMITATION .....                           | 118        |
| 4.3.1    | <i>Reinforcement Learning</i> .....                           | 119        |
| 4.3.2    | <i>Goal-Oriented Strategic Navigation</i> .....               | 122        |
| 4.3.3    | <i>Topology Learning</i> .....                                | 123        |
| 4.3.4    | <i>Deriving Movement Paths</i> .....                          | 127        |
| 4.3.5    | <i>Learning Utility Values</i> .....                          | 130        |
| 4.3.6    | <i>Multiple Weighted Objectives</i> .....                     | 133        |
| 4.3.7    | <i>Object Transience</i> .....                                | 134        |
| 4.3.8    | <i>Deployment</i> .....                                       | 135        |
| 4.3.9    | <i>Experiments</i> .....                                      | 135        |
| 4.4      | TACTICAL BEHAVIOUR IMITATION .....                            | 141        |
| 4.4.1    | <i>Motion Modelling and Tactical Behaviours</i> .....         | 142        |
| 4.4.2    | <i>Action Primitives</i> .....                                | 144        |
| 4.4.3    | <i>Bayesian Imitation</i> .....                               | 146        |
| 4.4.4    | <i>Experiments</i> .....                                      | 155        |
| 4.5      | REACTIVE BEHAVIOUR IMITATION .....                            | 159        |
| 4.5.1    | <i>Reactive Combat Behaviours</i> .....                       | 159        |
| 4.5.2    | <i>Learning the Inaccuracy</i> .....                          | 161        |
| 4.5.3    | <i>Methodology</i> .....                                      | 162        |
| 4.5.4    | <i>Experiments</i> .....                                      | 167        |
| 4.6      | CONCLUSION .....  | 172        |
| <b>5</b> | <b>BELIEVABILITY TESTING</b> .....                            | <b>173</b> |
| 5.1      | INTRODUCTION .....  | 173        |
| 5.1.1    | <i>Publications</i> .....                                     | 174        |
| 5.2      | MOTIVATION .....  | 174        |
| 5.2.1    | <i>Previous Contributions</i> .....                           | 176        |
| 5.2.2    | <i>The Turing Test and Game AI</i> .....                      | 180        |
| 5.3      | BOTT: THE BOT-ORIENTED TURING TEST .....                      | 184        |
| 5.3.1    | <i>Online Survey Structure</i> .....                          | 185        |
| 5.3.2    | <i>Experimental Protocol, Subjectivity and Bias</i> .....     | 189        |
| 5.4      | EVALUATION OF RESULTS .....                                   | 192        |
| 5.4.1    | <i>Computing the Believability Index</i> .....                | 192        |
| 5.4.2    | <i>The Confidence Index</i> .....                             | 193        |
| 5.5      | EXPERIMENTS .....   | 194        |
| 5.5.1    | <i>Methodology</i> .....                                      | 194        |
| 5.5.2    | <i>Results</i> .....  | 195        |
| 5.5.3    | <i>Feedback</i> .....   | 199        |
| 5.5.4    | <i>Remarks</i> .....  | 200        |
| 5.6      | CONCLUSION .....  | 201        |
| <b>6</b> | <b>CONCLUSION</b> .....                                       | <b>203</b> |
| 6.1      | INTRODUCTION .....  | 203        |
| 6.1.1    | <i>Full List of Publications</i> .....                        | 204        |
| 6.2      | THESIS REVIEW .....   | 205        |
| 6.3      | FUTURE WORK .....   | 209        |
| 6.3.1    | <i>QASE Development</i> .....                                 | 210        |
| 6.3.2    | <i>Potential Refinements to Existing Models</i> .....         | 212        |
| 6.4      | OPEN QUESTIONS .....  | 215        |
| 6.4.1    | <i>Meta-Behaviours</i> .....                                  | 215        |
| 6.4.2    | <i>Real-World Applicability and Biometric Imitation</i> ..... | 221        |
| <b>7</b> | <b>REFERENCES</b> .....                                       | <b>225</b> |



# Abstract

Despite a history of games-based research, academia has generally regarded commercial games as a distraction from the serious business of AI, rather than as an opportunity to leverage this existing domain to the advancement of our knowledge. Similarly, the computer game industry still relies on techniques that were developed several decades ago, and has shown little interest in adopting more progressive academic approaches. In recent times, however, these attitudes have begun to change; under- and post-graduate games development courses are increasingly common, while the industry itself is slowly but surely beginning to recognise the potential offered by modern machine-learning approaches, though games which actually implement said approaches on more than a token scale remain scarce.

One area which has not yet received much attention from either academia or industry is *imitation learning*, which seeks to expedite the learning process by exploiting data harvested from demonstrations of a given task. While substantial work has been done in developing imitation techniques for humanoid robot movement, there has been very little exploration of the challenges posed by interactive computer games. Given that such games generally encode reasoning and decision-making behaviours which are inherently more complex and potentially more interesting than limb motion data, that they often provide inbuilt facilities for recording human play, that the generation and collection of training samples is therefore far easier than in robotics, and that many games have vast pre-existing libraries of these recorded demonstrations, it is fair to say that computer games represent an extremely fertile domain for imitation learning research.

In this thesis, we argue in favour of using modern, commercial computer games to study, model and reproduce humanlike behaviour. We provide an overview of the biological and robotic imitation literature as well as the current status of game AI,

highlighting techniques which may be adapted for the purposes of game-based imitation. We then proceed to describe our contributions to the field of imitation learning itself, which encompass three distinct categories: *theory*, *implementation* and *evaluation*.

We first describe the development of a fully-featured Java API - the *Quake2 Agent Simulation Environment (QASE)* - designed to facilitate both research and education in imitation and general machine-learning, using the game Quake 2 as a testbed. We outline our motivation for developing QASE, discussing the shortcomings of existing APIs and the steps which we have taken to circumvent them. We describe QASE's network layer, which acts as an *interface* between the local AI routines and the Quake 2 server on which the game environment is maintained, before detailing the API's *agent architecture*, which includes an interface to the MatLab programming environment and the ability to parse and analyse full recordings of game sessions. We conclude the chapter with a discussion of QASE's adoption by numerous universities as both an undergraduate teaching tool and research platform.

We then proceed to describe the various imitative mechanisms which we have developed using QASE and its MatLab integration facilities. We first outline a behaviour model based on a well-known psychological model of human planning. Drawing upon previous research, we also identify a set of believability criteria - elements of agent behaviour which are of particular importance in determining the "humanness" of its in-game appearance. We then detail a reinforcement-learning approach to imitating the human player's navigation of his environment, centred upon his pursuit of items as *strategic goals*. In the subsequent section, we describe the integration of this strategic system with a Bayesian mechanism for the imitation of *tactical* and *motion-modelling* behaviours. Finally, we outline a model for the imitation of *reactive* combat behaviours; specifically, weapon-selection and aiming.

Experiments are presented in each case to demonstrate the imitative mechanisms' ability to accurately reproduce observed behaviours.

Finally, we criticise the lack of any existing methodology to formally gauge the believability of game agents, and observe that the few previous attempts have been extremely ad-hoc and informal. We therefore propose a generalised approach to such testing; the *Bot-Oriented Turing Test (BOTT)*. This takes the form of an anonymous online questionnaire, an accompanying protocol to which examiners should adhere, and the formulation of a *believability index* which numerically expresses each agent's humanness as indicated by its observers, weighted by their experience and the accuracy with which the agents were identified. To both validate the survey approach and to determine the efficacy of our imitative models, we present a series of experiments which use the believability test to evaluate our own imitation agents against both human players and traditional artificial bots. We demonstrate that our imitation agents perform substantially better than even a highly-regarded rule-based agent, and indeed approach the believability of actual human players.

Some suggestions for future directions in our research, as well as a broader discussion of open questions, conclude this thesis.

# Acknowledgements

Firstly, I'd like to thank Mark Humphrys for his supervision of my work; this thesis would not have come to fruition without his advice and encouragement. Thanks also to Alan Smeaton, who suggested that I apply for a PhD position in the first place. A special thanks must also go to Christian Thureau and Christian Bauckhage of Bielefeld University, with whom we enjoyed a very pleasant and productive collaboration over the course of our research. I'm especially indebted to them for their hospitality during my three-week stay at their campus, and for giving me the opportunity to discover just how terrible I am at waterskiing.

Thanks to everyone in my family for their amazing support; particularly my parents Ann and Barney and brother Conor, who managed to put up with me for the past four years while somehow retaining their sanity. Finally, a big thank-you to all my friends, both within and outside the college, for making the last few years so enjoyable. I honestly did try to make a list of you all, but DCU's guidelines are quite strict about a thesis' acknowledgements not being longer than its main findings; so on the slim chance that you someday feel compelled to wade through the following 230 pages, please accept this extraordinarily lame group-thanks.

# 1 Introduction

## 1.1 Introduction

This section provides a synopsis of the current state of AI in commercial computer games, the traditional attitudes of academia towards games research, and the potential advantages offered by machine- and imitation learning to both. We then proceed to discuss a number of research questions, before outlining our own contributions to the field under three distinct categories; *theory* (computational models of imitation for games), *implementation* (an API for machine and imitation learning in games) and *evaluation* (a rigorous testing procedure, designed to offer an objective, cross-agent measurement of *believability*). A detailed overview of the contents and organisation of this thesis concludes the chapter.

### 1.1.1 Publications

At the beginning of each chapter, we cite a number of our publications which are relevant to the contents of that section. A full list of publications is provided in the final chapter; see Section 6.1.1.

For further discussion of some topics covered in this chapter, see “*Learning Human Behavior from Analyzing Activities in Virtual Environments*” (Bauckhage, Gorman et al 2007), MMI-Interaktiv Journal, Nr. 12, April 2007, which addresses the role of imitation learning in games from a cognitive perspective.

## 1.2 Artificial Game Intelligence

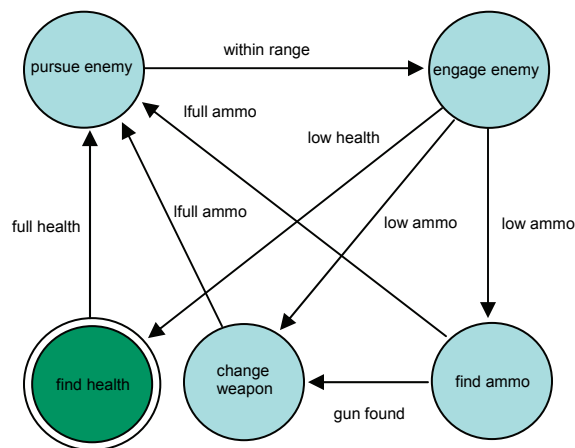
### 1.2.1 Games Industry

Despite the exponential growth of the games industry in recent years, and the parallel advance of academic artificial intelligence, the majority of computer games continue to rely largely upon symbolic AI techniques which were developed several decades ago [21, 24, 74, 91, 136]. Here, we briefly summarise the most common of these constructs, and their role in commercial game AI.

Perhaps the single most prevalent structure in game AI is the *finite-state machine* (FSM) [21], a concept dating from the Moore and Mealy machines of the 1950s [84, 89]. Favoured primarily due to their straightforward and logically intuitive design, they consist of hardcoded rules which determine the actions of the computer NPCs (*non-player characters*) for each potential state encountered during a game session. Modern game FSMs generally possess highly complex topologies, and are thus capable of producing a façade of intelligence adequate for their designers' purposes, but their limitations are obvious; the programmer must anticipate and correctly account for every scenario which may plausibly unfold in the course of a game session - a proposition which, given the complexity of current games, is increasingly untenable. A common complaint among experienced gamers are so-called *exploits* - areas of the state-action space which the designers did not foresee, and in which NPCs are therefore observed to act in a distinctly non-intelligent, often bizarre manner [120]. A simplified first-person combat FSM is shown in Figure 1-1.

Another common technique in commercial game AI is *scripting*, or *scripted behaviours*. Rather than hardcoding the NPCs' intelligence into the game itself, many games incorporate an interpreter and an accompanying high-level scripting language, allowing designers to quickly prototype game objectives and enemy intelligence semi-independently of the programmers [21, 102]. This approach serves to abstract

and separate NPC intelligence from the game code; changes to the AI can be implemented without necessitating a recompile of the entire game engine. Scripts can also be used to implement FSMs or other more advanced AI constructs, thereby providing a markedly greater degree of flexibility than FSMs alone. Nonetheless, they remain largely equivalent - the programmer must still define a set of rules accounting for as many in-game states as possible, however creatively these can be combined at runtime. The main attraction of scripted behaviours for game companies lies, instead, in the ease of development such a system provides; scripts can be used to define AI, create movie-like cutscenes, determine character interactions, and perform numerous other tasks, while the scripting engine itself can be re-used to greatly expedite the development of subsequent games [21].



**Figure 1-1 - A simple illustrative FSM for a first-person combat game**

A third significant component of commercial AI concerns *path-planning* - NPCs must naturally be capable of negotiating their virtual environments in a believable and efficient manner. Often, computer characters navigate by means of a *waypoint map* - a predefined graph whose edges represent legitimate, collision-free paths between important positions in the game environment, referred to as *nodes* or *waypoints*. The A\* algorithm is heavily employed in commercial games to traverse these environment graphs for the shortest path between the NPC's current location

and a desired goal node; in practice, a series of parallel “quick” and “full” path-planning operations are often used to avoid disturbances in the gameplay experience due to excessive processing time [103]. A variety of game-specific optimisations are already employed in existing commercial products [103], and additional improvements have been proposed in academia [121, 67]. More recently, some researchers have demonstrated the effectiveness of *probabilistic roadmaps* for pathfinding in virtual environments [93, 96], though industry has again shown little inclination to adopt these advanced methods.

In large part, the computer game industry’s reluctance to embrace more recent academic techniques can be attributed to their perceived *inscrutability*; programmers, working under strict schedules and pressured to deliver ever more impressive visual and audio experiences, are satisfied to implement shallow AI systems which can be easily and directly examined, analysed and altered as necessary, without the tuning required by more sophisticated techniques [21, 74]. There are some scattered exceptions to this rule - games which *do* adopt machine-learning approaches in the implementation of their AI. Lionhead Studio’s “*Black & White*” used neural networks to control its NPC protagonist, while Codemasters’ “*Colin McRae Rally 2*” used a similar approach to learn the ‘rails’ along which the computer-controlled opponent vehicles should travel. Even these games, however, resorted to deliberative approaches in more complex situations; Black & White used a BDI (*Belief-Desire-Intent*) model to define the NPC’s actual goals and ranges of behaviour [33], while Colin McRae used an FSM whenever the computer’s car was required to perform manoeuvres such as overtaking a competitor [48]. Millennium Interactive’s game “*Creatures*” used a genetic algorithm approach to evolve the behaviour of its characters, devoting approximately 50% of its computational cycle to AI routines. These games are, however, highly unrepresentative of the state of commercial game intelligence as a whole; indeed, in the cases of both Black & White and Creatures, their unique approach to AI was their entire *raison d’être* - the “gimmick” upon



which they were publicised and sold. In most games, only 1-5% of processor time is typically dedicated to AI [24].

The limitations of these AI techniques have, of course, a number of observable in-game consequences for the human player. Since the actions of each agent are generally determined in advance and endowed with only minimal flexibility, they represent an expression of the designers' expert knowledge - a top-down attempt to approximate what the programmer *thinks* a human would do, by examining his own decision-making processes - rather than an intelligence derived from practical experience of the game world. As a result, such characters frequently exhibit repetitive, mechanical or otherwise non-humanlike behaviour, particularly when confronted with uncommon situations, and are often rendered ineffective by the human player's natural aptitude for pattern-recognition within a relatively short space of time. This in turn gives rise to another problem regularly encountered by gamers - namely, that developers attempt to compensate for their NPCs' limited intelligence not by implementing new and better approaches, but simply by granting the computer-controlled protagonists abilities which defy the rules imposed by the game upon its human players [21]. Sometimes this involves exploiting the artificial players' direct access to the underlying gamestate, allowing them - in combat games, for instance - to determine the precise position and velocity of their opponents and strike with pinpoint accuracy. On other occasions, the cheating is more blatant; computer enemies in resource-management or real-time strategy games may, for example, be able to construct buildings or vehicles at a faster rate than human participants. In racing and sports games, opposing characters are often observed to acquire an inexplicable burst of speed if they fall too far behind, a phenomenon so common that it has entered gaming vernacular as "rubber-band AI" [110]. Not only do such 'shortcuts' detract from the perception of these characters as human, but they frequently lead to frustrating gameplay, as players find themselves pitted against foes with speed, reflexes and accuracy far in excess of their own.

Nonetheless, competent players are still capable of regularly defeating these superhuman opponents, implying that approaches based on imitating human play could ultimately produce both more humanlike *and* more challenging artificial agents.

### 1.2.2 Academia

Despite a history of games-based research, including Tesauro's TD-Gammon [125] and IBM's famous Deep Blue (which began as a research project at Carnegie Mellon University [57]), academia has generally regarded commercial games as a distraction from the serious business of AI, rather than as an opportunity to leverage this existing domain to the advancement of our knowledge. Part of this has been caused by the *type* of games typically used; while chess and its ilk provide a useful testbed for the development of efficient searching and pruning techniques, they present problems to which computers are inherently *suited*, and do not offer the potential that modern commercial computer games do. Chess is, from the human perspective, a deeply strategic game precisely *because* it requires skills in which humans are not naturally adept; the manipulation and mechanical searching of large state trees. From the computer's perspective, on the other hand, the configuration of the chess board is easily reducible to a matrix representation, the entire range of potential moves is known, and a heuristic value can be applied to each subsequent configuration with reasonable accuracy. Choosing the next moves therefore becomes an exercise in pure number-crunching - or, as the developer of Deep Blue put it, "brute-force computation has eclipsed humans in chess." [55] Indeed, some now argue that work in board-game AI has become so constrained that it not only does a disservice to the promotion of games-based research in general, but is barely research at all [91].

In addition, such games bear little or no relevance to the real world, and it is thus difficult to see how they can contribute significantly to an intelligence capable of the

broad range and scope of human behaviour. One of the recent pioneers of academic game AI, John Laird, argues that the field at large has become so fragmented, concerned with the development of ever more specialised algorithms designed to work in increasingly narrow problem domains, that the “holy grail” - the development of an integrated human-level intelligence - has been all but forgotten [74]. It is his contention, given the rich complexity of interactive computer game environments and the fact that they are inherently designed to present problems whose solutions require human reasoning abilities, that such games represent the single best platform upon which to tie the various strands of AI research into a cohesive whole and construct new models of artificial intelligence. Laird strongly advocates the closer collaboration of academia and industry in pursuit of this goal, while the games industry itself is beginning to realise the benefits such a partnership could bring; development studios are hiring AI PhDs in increasing numbers [74], and some experts predict that home games consoles of the near future may incorporate a dedicated AI processor in the vein of the specialised graphics hardware present in current systems [21]. All the industry needs, according to Laird, is for academia to tell them what is required.

Due in large part to the work of Laird and some others - aided, most likely, by increasing numbers of young computer scientists who have grown up playing games, and who therefore recognise their potential - there has been a steady increase in the volume of research utilising commercial games in recent years. Some illustrative examples are presented here. Laird has himself produced extensive work on artificial Quake 2 combat opponents using his SOAR architecture [74 - 78, 136], incorporating over 800 hardcoded rules which operate by recursively deconstructing high-level goals into a series of lower-level operations. Van Waveren [135] wrote a Master’s thesis based on his development of an advanced Quake 3 agent; this employed largely traditional game AI techniques, with its core intelligence consisting of a fuzzy logic module and a highly complex FSM. He also proposed a novel approach to

pathfinding using an interconnected network of collision-free 3D volumes, along similar lines to the PVS (*potentially visible set*) visualisation data stored in the game world's geometry files. It is worth noting that Van Waveren was also the author of the Quake 2 Gladiator bot [134], which we used as a comparator in our believability experiments (see Chapter 5).

Notably, some contributions to academic game research have served to indirectly reaffirm the criticisms of rule-based and traditional commercial AI enumerated earlier in this chapter. Solinger et al [118], for instance, present a knowledge-based approach to creating an agent in Microsoft's "Combat Flight Simulator". In explaining their decision to use hardcoded rules for decision-making, they observe that "the advantage... is that it is easy for people to understand the rules, which makes analysing pilot behaviour much easier." They proceed to describe the agent's difficulties when faced with unfamiliar situations, and foreshadow the increasingly unmanageable rule-bases necessary to account for every scenario in modern simulations: "the dogfight agent may fail when the human player is creative and executes a manoeuvre that is unknown to the agent... the easiest way to overcome this problem is to create more situations and rules in the reasoning parts of the system." Elsewhere, Norling and Sonenberg [94] designed an artificial Quake 2 player by interviewing expert human players, and incorporating their attitudes towards strategic and tactical decision-making into a BDI framework. While their work proved promising in its ability to reproduce the behaviours described by the interviewees, this premise does evoke a number of obvious conceptual objections, and again serves to clarify one of the difficulties raised earlier against top-down rule-based systems in general - that such approaches add an extra layer of subjectivity between the observable game world and the agent implementation. Instead of the traditional game-AI scenario, wherein a programmer utilises his expert knowledge to project how a human player would behave, Norling's approach involves programmers asking third-party experts to describe *their* reasoning processes, which

are then approximated. Considering that games provide us with the ability to obtain low-level data recordings of said reasoning processes in action, a more direct and rigorous alternative to either of these approaches is surely possible.

### 1.2.3 Machine Learning

As indicated above, the growing interest in commercial computer games as a legitimate area of research has nonetheless centred in large part upon rule-based or similar systems, as are already common in industrial game AI. Recent years have, however, also witnessed a number of contributions which apply *machine learning* techniques to the problem of game AI. Graepel et al [44], for instance, induced a customised agent to learn different combat styles (aggressive/evasive) in the one-on-one martial arts game “*Tao Feng*”, by pitting an online reinforcement-learning system against the game’s in-built AI. Bakkes and Spronck [9, 10] proposed a symbiotic framework for allowing FSM-controlled characters in team-based games to evolve co-operative intelligence, and demonstrated that it was capable of counteracting the advanced FSM AI in iD Software’s “*Quake 3: Team Arena*”. Björnsson et al [16] described a reinforcement learning approach to creature intelligence in a basic mobile/PDA game, using the human controller’s feedback to expedite convergence and to develop player-specific behaviour preferences. Spronck et al [119] also used reinforcement learning to develop a dynamic scripting approach to low-level NPC combat behaviours in the “*Baldur’s Gate*” role-playing game, while Ponsen and Spronck [101] applied the same techniques to global game states in the real-time strategy game “*Wargus*”. These approaches still, however, relied upon rulebases of manually constructed state-action pairings; reinforcement learning was simply used to progressively weight these rules according to their influence over the agents’ subsequent successes or failures.

Aside from general machine-learning approaches, there is another subdiscipline which holds immense potential as a means of developing intelligent game AI. By examining the manner in which gameplay sessions unfold, it becomes clear that a wide variety of interesting approaches based on *imitation learning* - that is, the automated acquisition of skills or behaviours via observation of an expert's demonstration - may be possible. Though prominently employed in robotics, imitation learning has seen surprisingly little application in games-based AI, and is scarcely represented in the literature. Before we discuss the research questions which presented themselves in the course of our own work, and the contributions which we have made to the field, it is worthwhile to consider some of the advantages which computer games offer as research platforms for imitation learning.

#### 1.2.4 Games as Imitation Research Platforms

Every day, millions of people play thousands of games both online and in LANs - from frantic first-person shooters to intricate role-playing simulations to contemplative traditional games like chess and bridge. Each time they do so, they generate vast quantities of raw behavioural data; data which could be of enormous use in the development of new and interesting approaches to artificial intelligence, but which too often simply vanishes, its potential unrecognised, into the ether of the internet. Fortunately for us, many serious gamesplayers routinely save and archive their own matches for posterity; but this is only a tiny fraction of the data which could be harvested, were a dedicated effort to be made at any LAN conference or similar event of the kind shown in Figure 1-2. Aside from the *volume* of data afforded to games-based researchers, there are a number of other advantages in using games as an imitation-learning research platform - and indeed, for artificial intelligence research in general. We outline some of these below.



**Figure 1-2 - The biannual DreamHack LAN gives some sense of the sheer volume of data potentially available to games-based imitation researchers. Such events represent an unrecognised and largely untapped resource. (Photo: DreamHack Winter 2007)**

In contrast to the fields of robotics and computer vision - wherein data is generally derived using expensive motion-capture equipment, or by subjecting video sequences to extensive analysis - many modern games allow the recording of entire sessions (in the form of network traffic) to comprehensive data files with nothing more than a single command. Training samples acquired in this manner are, furthermore - and again, in contrast to the aforementioned disciplines - entirely devoid of noise; they contain the exact locations of each participant and entity in the game world, their motion, internal states and all external influences. There is, therefore, no need to process the visual rendering of the game world as seen by the human player in order to analyse his behaviour; employing games as an imitation-learning platform allows us to essentially *skip* the considerable effort required for data acquisition in other fields, and instead to focus solely on the task of developing cognitive models to explain the encoded behaviours. Indeed, in cases where elements of the player's visual perception are required, it is possible to design models which recreate this perception from the bottom up - that is, to build a noise-free reconstruction of the player's experience of the virtual environment from the low-level data representation. This is precisely the approach we adopt in the development of our *reactive* imitation system, detailed in Section 4.5.

Furthermore, as noted by several researchers [e.g. 13] and discussed in later sections of this thesis, much of the robotics literature focuses primarily on the *engineering* challenge of achieving realistic humanoid physical motion. Rather than limb movement data or similar, however, recordings of gameplay sessions encode the actual *behaviours* of human players under rapidly-changing conditions and against opponents of comparative skill; in many ways, this represents a more attractive and fertile domain for AI researchers. Games essentially provide closed environments with well-defined, sophisticated sets of competing objectives - they can be viewed as the human equivalent of a typical laboratory rat maze. The goal of imitation learning



in computer games, expressed in its simplest form, is thus to construct models which explain and accurately reproduce behaviours recorded as sequences of network messages during human gameplay sessions, thereby '*reverse-engineering*' the player's decision-making process from its observed results.

A more general advantage of using games as research platforms is that they resemble idealised *robot navigation worlds*. Much of the work in this field is centred upon the elimination of noise in the robot's environment [17, 72], whereas computer games guarantee accurate, undistorted information. Furthermore, map editors for such games are common, allowing the user to create any environment he desires with minimal time and effort. By utilising appropriate games as testbeds, researchers can thus easily prototype their navigation techniques without needing to account for sensory or other complications, before later adapting them to the additional constraints of the real world. In similar fashion, certain game genres represent a relatively close abstraction of reality; thus, the mental competencies required to negotiate the virtual environment are likely to be quite similar to those required for real-world navigation. Games may therefore provide some interesting insights into human spatial reasoning and related faculties.

To summarise: employing games as a platform for imitation-learning research gives access to almost unlimited training samples, containing no noise or other artefacts, and which encode human planning and decision-making behaviours of potentially far greater interest than the limb-motion data typically found in robotic imitation. Some of these issues will be recounted and further contextualised at relevant junctures over the remainder of this thesis; further discussion of the advantages inherent in using games as research platforms - more specifically, the *first-person shooter* genre - can be found in Section 2.3.

## 1.3 Contributions

### 1.3.1 Research Questions

The pursuit of imitation in games raises a number of immediate questions:

- what game genres are most suited to the task?
- how can we represent and parse demonstrations of human play?
- what behaviours are exhibited in the recorded data?
- how should the data be analysed and used?
- how can the believability and performance of agents be objectively measured?
- to what degree, and in what areas, can imitation replace rule-based techniques?
- can techniques developed to reproduce human behaviour in computer games be applied in other fields? And if so, how?

In broader terms, how can human behavioural data, generated during interaction with a computer game environment, be collected and used to implement artificial agents which act in a similar manner? What testbed game(s) should be chosen to provide the best balance between complexity of encoded behaviour, real-world applicability, and ease of analysis? What techniques may prove useful in analysing the observation data? And how can lessons learned from the development of these agents be applied in other domains? Over the remainder of this and the following chapters, we propose answers to each of these questions.

### 1.3.2 Contributions: Our Research

As mentioned earlier, one of the most striking observations to confront us in the initial stages of this research was, simply, the lack of existing work. Our contributions to the field of imitation learning in games therefore developed in a somewhat linear manner, which can be summarised in three stages:

## **1. THEORY: MODELS OF COMPUTATIONAL GAME IMITATION**

Drawing upon biological evidence, approaches from the field of robotics, and our own experience in computer games, we developed a number of imitative models designed to learn and reproduce various aspects of observed human behaviour. These mechanisms fall into three distinct subcategories, closely mirroring a widely-used psychological model of human planning (see Section 2.2.1); *strategic* long-term behaviours, *tactical* medium-term behaviours, and *reactive* short-term behaviours. Our models, and the experiments conducted in order to validate each of them, are detailed in Chapter 4.

## **2. IMPLEMENTATION: A MACHINE-LEARNING API FOR GAMES**

One major obstacle encountered early in our research was the lack of an API that was suitable for our intended purposes. Existing development platforms were incomplete and ad-hoc, often with scattered documentation and unintuitive interfaces. Rather than writing a minimal library which provided only the specific features we required - thus leaving future researchers in exactly the same position in which we had found ourselves - we instead decided to develop a comprehensive API, encompassing all the functionality that students or researchers would require in order to conduct further work in this field. While geared principally towards machine- and imitation learning, it would also represent a platform for cognitive agent research in general. The resulting QASE API, and details of its adoption by various universities at both undergraduate and postgraduate level, is detailed in Chapter 3.

## **3. EVALUATION: THE BOT-ORIENTED TURING TEST (BOTT)**

As noted above, we present statistical validations of the functionality of each imitative model in their respective subchapters. However, given that one of our aims was to demonstrate the capacity of imitation learning to produce more *believable* game agents, this alone was not sufficient. We wished to

further evaluate the “humanness” of our agents, as perceived by actual observers. Here, we again ran into an obstacle; there was no rigorous means of evaluating this quality in agents. Some minor investigations of believability had been conducted, but these were invariably highly informal, and certainly not appropriate for inter-agent comparison. We thus decided to design a generalised approach to such testing, which we call the *Bot-Oriented Turing Test (BOTT)*. Our proposed system is detailed in Chapter 5; it consists of an anonymous online questionnaire, a protocol to which examiners should adhere in order to maintain the survey’s integrity, and the formulation of a *believability index* which numerically expresses the respondents’ perception of the agent, weighted by experience and the accuracy with which it was identified. A series of experiments comparing our imitation agents against actual human players and traditional rule-based agents served both to evaluate our agents, and as a test-case for the believability survey itself.

### 1.3.3 Thesis Outline

Although our research *conceptually* followed the above progression, in practice we naturally needed to build an API before we could effectively develop our imitative models. In Chapter 3, we therefore describe the first of our contributions - the Quake Agent Simulation Environment (QASE). We outline our motivations for developing QASE, discussing both the shortcomings of existing APIs and the steps which we have taken to circumvent them. We then describe QASE’s network layer, which acts as an *interface* between the local AI routines and the Quake 2 server on which the game environment is maintained, before detailing the API’s *agent architecture*. The latter consists of a *bot factory* which allows agents to be created from any of several levels of abstraction, together with a variety of data structures designed to provide these agents with additional functionality, including the ability to supply the agent with sensory data about its environment, the facilitation of high-level queries about

the agent’s condition by transparently amalgamating low-level data, the ability to record and parse demonstration of human gameplay, integration with the MatLab programming environment, inbuilt AI constructs, and topology-learning / navigation faculties drawn from our work in imitation learning. We conclude the chapter with a discussion of QASE’s adoption by numerous universities as both an undergraduate teaching tool and a research platform.

In Chapter 4, we describe a number of imitative mechanisms developed using QASE and its MatLab integration facilities. We first outline a behaviour model based on Hollnagel’s hierarchy of human planning [53], as discussed in Section 2.2.1. Drawing upon previous investigations, we also identify a set of *believability criteria* - elements of agent behaviour which are of particular importance in determining the “humanness” of its in-game appearance. In Section 4.3, we then proceed to detail a reinforcement-learning approach to the imitation of the human player’s navigation of his environment, centred upon his pursuit of item pickups as *strategic goals*. This involves the derivation of a topological environment map by clustering the set of positions occupied by the player in the course of the game session, and drawing edges between the resulting *waypoints* based on the player’s observed motion. We then examine the relationship between the items currently possessed by the player and his subsequent navigation routes, creating a system of rewards such that the agent is drawn along similar paths. We augment this by considering two further elements of human strategy; weighted pursuit of multiple parallel goals, and recognition of *object transience* - a feature of Quake 2 games whereby items become unavailable for a set period of time after collection. We show that this system is adept at capturing the human’s *program-level* behaviour; in other words, the imitation model has the capacity to identify and pursue the human’s *goals*, without necessarily reproducing his precise actions at each timestep.

In Section 4.4, we describe the integration of this strategic navigation system with a Bayesian mechanism for the imitation of *tactical* motion-modelling behaviours. We note that it is not sufficient for the agent to simply move along the ground in the game world; it must often negotiate obstacles such as chasms, lifts, doors, etc in order to reach its next goal. Motion-modelling provides the means to achieve this, as well as reproducing the smooth aesthetic effects that typify a human player. Our approach utilises the concept of *action primitives* - that is, aggregated representations of similar actions performed at different times in different places. We sequence these primitives using a heavily adapted version of Rao et al's model of action sequencing in human infants [106], which expresses the next action as a function of the agent's current state, next state and goal state - information which is supplied by the strategic navigation system. We also propose a technique, which we call *imitative keyframing*, to combine the consistency offered by *absolute* orientation primitives with the fine-grain adjustments provided by *relative* values.

In Section 4.5, we outline a model for the imitation of *reactive* combat behaviours; specifically, weapon-selection and aiming. Drawing upon previous work which found that direct imitation produced excessively accurate aim, our approach was to instead learn the *inaccuracy* of the human player's aim. This consists both of *intentional inaccuracy*, due to the player aiming ahead of the opponent in order to compensate for the speed of his weapon's projectiles, and *unintentional inaccuracy*, due to straightforward human error. To model this, we first reconstruct the human player's visual perception of his opponent's motion from the available low-level data, and then use it to train a series of three expert networks; one to select the most appropriate weapon, one to adjust the agent's aim, and a third to determine whether the agent should open fire. A number of interesting phenomena captured by the imitative mechanism are then discussed.

In each of the chapters listed above, we conducted experiments to provide a statistical validation of the respective models' functionality. This was, however, not sufficient on its own; we also wished to ascertain the degree to which the imitation agent is *perceived* as being "humanlike" by an actual observer. We note the lack of an existing means of rigorously gauging the believability of game agents, the few previous attempts having been extremely ad-hoc and informal; we therefore propose a generalised approach to such testing in Chapter 5 - the *Bot-Oriented Turing Test* (BOTT). This takes the form of an anonymous online questionnaire, an accompanying protocol to which examiners should adhere, and the formulation of a *believability index* which numerically expresses each agent's humanness as indicated by its observers. The protocol specifies certain steps which are necessary in order to maintain the integrity of the survey. The survey itself first ascertains the experience level of the respondent on a strictly defined scale, and proceeds to present him with a series of test pages; once complete, the believability index for each category of clips can be computed and compared. To both validate the survey approach and to determine the efficacy of our imitative models, we then present a series of experiments which used the believability test to evaluate our own agents against both human players and traditional artificial bots. These tests demonstrate that our imitative mechanisms are capable of producing agents which perform substantially better than even highly-regarded FSM-based agents such as the Quake 2 Gladiator bot [134], and indeed approach the believability of actual human players.

## 1.4 Conclusion

This chapter presented a discussion of the current attitudes towards computer game AI research, from the perspectives of both industry and academia. We then outlined a number of relevant research questions, and described our own contributions to the field in three distinct categories; *theory*, *implementation* and *evaluation*. Finally, a chapter-by-chapter overview of this thesis was provided.

## 2 Imitation, Behaviour and AI

### 2.1 Introduction

In this chapter, we discuss the concept of imitation learning in greater detail, and outline the various advantages it offers over more traditional artificial intelligence techniques. We ground our later work in a discussion of the biological impetus for learning by observation, demonstrating that imitation is a necessary component of higher intelligence, that it is a neurologically-supported evolutionary development rather than a learned behaviour, and that it is computationally explicable in terms of forward and inverse models. We then proceed to describe the adaptation of these biological precedents in the field of robotic imitation, highlighting specific approaches which are relevant to our own research. Finally, we present an overview of imitation as it applies to game AI, justify our choice of Quake 2 as an experimental testbed with regard to other game genres, discuss the role of believability- and performance-based metrics in our work, and formulate a simple but illustrative computational model of imitation learning in games.

#### 2.1.1 Publications

Some of the topics covered in this chapter are also discussed in “*Learning Human Behavior from Analyzing Activities in Virtual Environments*” (Bauckhage, Gorman et al 2007), MMI-Interaktiv Journal, Nr. 12, April 2007.

### 2.2 Imitation Learning

Imitation learning, as the name suggests, refers to an agent’s acquisition of skills or behaviours by examining a demonstrator’s execution of a given task. In spite of some



movement towards games-based machine-learning research, imitation has thus far received surprisingly little attention as a means to enhance the believability of game agents. Perhaps even more perplexing is the converse; that the academic AI community has neglected computer games as a platform for driving imitation learning research. Considering their obvious suitability for the purpose - as will be discussed in greater detail later - this dearth of previous work is quite inexplicable; it is readily apparent that the potential benefits, both to the games industry and to academia, are immense.

In contrast to the lack of existing games-based imitation learning research, imitation is heavily employed in the field of robotics, and has amassed a significant literature. As with our own work, robotic imitation is grounded - insofar as is possible - in biological precedent, drawing upon several strands of psychology, neurology and behavioural science. It is therefore instructive to examine some of the approaches which have been developed by roboticists, in order both to appreciate the advantages offered by employing computer games as one's research platform, and to examine these techniques for possible adaptation in our own work. In the following sections, we provide a brief overview of imitation learning as it is understood in nature and applied in robotics, before further discussing the potential roles for imitation research in computer games, and vice-versa.

## 2.2.1 Biological Imitation

### *2.2.1.1 Imitation and Mimicry in Infants and Animals*

There is much evidence from the fields of ethology, neuroscience, psychology and linguistics to suggest that imitation is an important component of higher intelligence. These disciplines have long been interested in the imitation of movements, vocalisations and behaviours; George Romanes [109] proposed a scale of imitative

abilities among different animal species as long ago as 1884, and his ideas attracted the attention of researchers for several decades afterwards [e.g. 126, 128]. In one of the most well-known studies of imitation in children, Jean Piaget [100] argued that humans develop their imitative capacities in a progressive manner. Infants, he claimed, are capable only of copying actions within their immediate perceptual field and of the same sensory modality (for instance, hand gestures where the child's own hand is also visible). Cross-modal imitation requiring proprioceptive information, such as the replication of facial expressions, follows after the first year of development, while deferred imitation - the ability to mentally retain observed actions, and reproduce them when the demonstrator is no longer present - does not emerge until 18 to 24 months [27]. Since Piaget considered the immediate aping of behaviours to be insignificant, and argued that true, *deferred* replication was a learned ability, the study of imitation as a form of intelligence became somewhat stigmatised. It remained so until the 1970s, when investigations by Meltzoff and Moore found that human neonates between 12 and 21 days of age are capable of imitating both facial and manual gestures [86]. As this could not be explained by conditioned mechanisms, they concluded that it was an innate faculty, contradicting Piaget's earlier findings. Subsequent results showed that even babies less than an hour old were capable of imitation, and went so far as to imply that infants devote most of their time to duplicating behaviours. Based on these observations, they proposed the "Active Intermodal Mapping" model [85], which posits that infants use their visual perception of a demonstrator's state as a target against which to direct their own body states, perceived proprioceptively. This subsequently formed the foundation of Demiris and Hayes' model of *passive robot imitation* [26], as will be discussed later. Today, psychologists regard imitation as being central to a child's development of symbolisation and representational skills.

With interest in the mechanisms and significance of imitation galvanised, further studies showed that many other animals are incapable of learning in this way,

reinforcing the view that imitation is indicative of higher intelligence. Indeed, humans are now considered the only species capable of *imitation* as opposed to *mimicry* - that is, the capacity to acquire and develop novel motor skills which are not already part of their repertoire, through observation [3]. Billard [13], in a useful primer on biological imitation and its relevance to robotics, distinguishes a number of different forms of pseudo-imitation. Simple *copying*, as seen in species such as rats and monkeys, refers to actions which are learned associatively and reproduced when the animal is placed in the same social context as that wherein the behaviour was originally observed. Copying of this kind is viewed as an example of social facilitation; the animals demonstrate the ability to reproduce the correct behaviour even when provided with incomplete contextual cues. More complex *mimicry* is observed in species such as apes and dolphins - these animals are capable of mastering sequences of manipulations, distinguishable from straightforward copying by the animals' ability to reproduce isolated subsections of (or variations upon) the imitated behaviours in contexts other than those in which they were learned. Dolphins also exhibit the comparatively sophisticated ability to map heterologous body structures to their own; for instance, they are capable of replicating a human demonstrator's limb movements using their fins and tails. Orca calves learn to catch prey by intentionally beaching themselves through a combination of mimicry and direct teaching from their parents [98].

#### *2.2.1.2 Human Imitation and Internal Models*

The most complex imitative faculties - facilitating what is generally termed *true imitation* - are exhibited by humans. Human imitation diverges from animal mimicry principally in that the range of imitative abilities we possess requires a theory of mind; that is, the ability to impute beliefs, desires and intentions to others. Human imitation has the capacity to abstract observed motions, parameterising them during

reproduction based on the contextual goal or intent of the imitator - the same motion can, for instance, be optimised for *aesthetic appearance* (as in dance), *efficiency* (as in sports) or *precision* (as in surgery) [13]. Human imitation also extends to verbal and facial expressions, and an innate ability to recognise gestures; observers can deduce actions from minimalistic point-light representations of the actor's extremities, and distinguish both general and specific features of the demonstrated movement (e.g. the weight of a lifted object).

Our ability to deduce complex movements from even such minimal visual cues strongly suggests that the brain maintains an internal representation of the human body's kinematics and ranges of possible motion; that the central nervous system employs *forward* and *inverse model* to predict the outcome of observed or replicated motor commands. This topic is discussed in greater detail in e.g. Wolpert et al [139], wherein the authors note that fast, co-ordinated limb movements cannot be controlled by sensory feedback mechanisms alone, since the relatively significant delay incurred by sensory processing, sensory-motor coupling and motor execution renders such a real-time control system implausible. They instead suggest that the cerebellum acts as a form of *Smith Predictor*; in other words, that the brain maintains separate forward models of limb dynamics and of the transport delays involved in sensory afference. The former produces a prediction of the limb's next state given the execution of a motor command, and this state output is then *transformed* by the delay model, in order to generate a final estimate of the sensory information as it is expected to subsequently arrive. Any discrepancies between the estimated state and the actual sensory feedback are then used to correct the actor's movement, and to refine the predictive models.

Furthermore, they note that the sheer number of contexts under which humans operate, and the variety of environmental variables involved even in simple actions,

precludes the possibility that the brain relies upon a single controller capable of encapsulating every such scenario. Instead, they posit a modularised system of paired models [138]. Each module consists of a forward model which captures a particular behaviour, and an inverse model controller which generates the motor command necessary to enact that behaviour. A *responsibility estimator* determines the degree to which each module's forward model accounts for the current context; each inverse model then contributes proportionally in producing the final efferent motor command. Errors between the predicted and observed states are propagated back to the modules based on their respective contributions, and the inverse models adjusted accordingly. Johnson and Demiris [66] propose a similar approach; here, however, a *hierarchy* of coupled forward and inverse models is employed instead of a parallel collection of modules. Abstract action representations generated at the topmost levels percolate downwards through the model's strata, where successive constraints are applied to produce progressively more specific actions; finally, the lowest level outputs an actual motor command. Johnson and Demiris explicitly present this model as a computational representation of the *mirror neuron system*, as is discussed in the following section.

A similar system of dividing responsibility proportionally across all known contexts is employed as part of our own *strategic imitation* framework; see Section 4.3.6.

### 2.2.1.3 *Mirror Neurons, Imitation and Language*

The ability of even newborn infants to accurately map perception onto action implied that certain areas of the brain are optimised for the task of imitation. This intuition was substantiated by Rizzolatti, Gallese et al [39] with the discovery of so-called *mirror neurons* in macaque primates, which were found to activate during both the observation of certain behaviours in other individuals and their own execution of related actions. The presence of a similar system in the human brain was confirmed

by Fadiga et al [34], who used magnetic stimulation to show that areas of the premotor cortex controlling the muscles involved in performing a given action also become excited when the subject witnesses a demonstration thereof. Subsequent investigations using PET scans served to clarify the specific pathways which activated during observation of a grasping motion, replication of the same action, and even when the subject *imagines* himself doing so [45, 107]. These studies also revealed some interesting aspects regarding the manner in which motor tasks are represented and executed. Some mirror neurons, for instance, were observed to only ‘recognise’ actions with highly specific features (e.g. when an object was held in a configuration using the thumb and index finger), while certain related brain regions appear to predict potential interactions with objects in the near future from sensory cues, such as the different ways in which an item can be grasped by the hand (“*grasping affordances*” [4]). These results, among others, strongly suggest that the brain’s perceptual and motor systems share a common representational substrate; that they are designed in a manner which lends itself perfectly to the task of imitation. Billard [13] notes that further evidence for the existence of a human mirror system can be found in brain lesion studies. Patients who have suffered lesions on their parietal lobes sometimes develop an inability to make gestures or use objects in response to an oral command (*ideomotor apraxia*). These patients are also occasionally unable to recognise gestures from a photograph or video, suggesting that the role of the parietal lobe in imitation is to translate the mental representation of a movement into actual motor commands. Patients who suffer lesions on the frontal lobe, by contrast, sometimes exhibit *compulsive* imitative behaviour; they are able to stop such imitation only with considerable effort. This strongly implies the existence of a neural subcircuit which continually processes visual information by activating the motor patterns responsible for producing the same movements - that is, a mirror neuron system - which is modulated and inhibited by the frontal lobe.

Billard further observes that our aptitude for imitation - the capacity to recognise and project others' states to our own - also provides a powerful mechanism for advanced *social cognition*, endowing us with the ability to infer others' intentions and even to estimate, manipulate or deceive their mental states. Indeed, there is substantial evidence that imitative abilities may have underpinned the evolution of the most complex human social interaction - language. Debate on this question has largely centred, over the past several decades, upon two contrasting views of language development [14]: the *nativist* approach, which holds that humans possess innate, biologically “hardwired” language-learning faculties independent of other cognitive skills, and that evolution has produced a *universal grammar* underlying all languages; and the *non-nativist*, which proposes that the ability of infants to rapidly learn their native tongue is a function of general cognitive learning processes applied to a specific task. Nativists point to the fact that infants master language at a far greater rate than is possible through simple association, while non-nativists stress the importance of both egocentric mental development and external social/behavioural cues - learned primarily through imitation - in the process of language learning. In recent years, however, mirror neurons have provided some compelling *neurological* basis for the latter view.

Rizzolatti and Arbib [108] note that the F5 area of the macaque brain - where mirror neurons were originally discovered, and which exerts control over orofacial actions - is generally accepted as being homologous with Broca's region in the human brain, which is responsible for syntactical processing, plays a partial role in comprehension, and controls the motor functions necessary for speech production. They observe that Broca's area is also involved in the execution of hand and arm gestures, and in the mental imagining of same. Given the concentration of mimetic, linguistic and specific motor faculties in this region, and drawing upon studies of animal communication, they propose that speech developed as a consequence of the mirror system's progressive evolution. First, they argue, we developed the ability to

communicate via imitative facial cues, as is still observed in apes and monkeys - as well as humans - today. This highly limited form of one-to-one communication was later complemented by *manual* gestures, allowing for a dramatically wider range of expression and combinatorial possibilities; third parties, be they objects or other individuals, could now be referenced and described via vocal utterances. Through repetition and imitation, these vocalisations gradually assumed semantic and symbolic meaning, forming the basis of a simple vocabulary. Rizzolatti and Arbib thus posit that the emergence of Broca's area and its speech-related faculties from an F5-like precursor, and the concomitant migration of vocalisation control from the primitive brain-stem structures still employed by present-day animals to the higher brain, was driven by the increasing necessity for integration of laryngeal motor control, mirror properties, and close association with the adjacent premotor cortex as prerequisites for spoken language.

#### *2.2.1.4 Motor Primitives*

Some years before the discovery of mirror neurons, it had been found that by stimulating specific areas of a frog's spine, complete movement behaviours such as reaching or grasping could be generated; these could be further combined to form more complex movement sequences [43]. Since stimulation of similar areas of the spine in different frogs produced the same results, it was inferred that these were a phylogenetic rather than an ontogenetic feature. From these investigations, the notion of *movement primitives* developed - a comparatively small set of combinatorial motor programs which represent a *substrate* of autonomous behaviour. Rizzolatti & Gallese's paper [39] served to provide some further neurological corroboration of this concept, and clarified its relevance to imitation; they observed that certain mirror neurons became inactive when a task demonstration finished, whereas others remained active for a short while thereafter. Demiris [28], in presenting a computational model of the mirror neuron system, proposes that this phenomenon be



interpreted within the context of *composite behaviours* - that while a behaviour X (which we may view as an action primitive) becomes dormant upon completion, a behaviour X\* whose initial step is X will remain active, since it is still capable of generating predictions about the demonstrator's future states. Thoroughman & Shadmehr [127] found indications that the human brain uses similar primitives to generate action sequences, since this approach significantly reduces the parameters which need to be learned in order to produce complete movement behaviours.

As will be discussed later, the concept of motor primitives has received significant attention in robotic imitation, and we have successfully adapted it for use in our own work (see Section 4.4.2).

#### 2.2.1.5 Ethology

The field of *behavioural science* also provides a number of models which are of use in artificial imitation. For instance, Schaal [113] proposes a number of concepts derived from cognitive studies of imitation which suggest promising directions for agent learning and control. *Emulation*, where the actions of others serve to make goals in the environment more overt, may provide a basis for learning how to direct an agent towards favourable objectives. An approach based on *priming*, where environmental stimuli co-occur with the demonstration and increase the observer's recollection - such as Pavlov's famous salivating dog experiment - could be used to bias an agent's exploration towards similar stimuli. Hollnagel's COCOM (*Contextual Control Model*) [53, Figure 2-1], which describes a person's actions as a function of his situational context and available information, is often employed to analyse human behaviour [132]; as will be discussed later, an adapted version of this model was instrumental in the development of our own imitative mechanisms (see Section 4.2). Elsewhere, drawing upon their investigations of learning and co-operation in primates, Byrne and Russon [20] propose a two-level hierarchy based on differing

degrees of imitative abstraction. At the higher level, *program imitation* involves replicating the structural organisation of a complex process, while the low-level actions required to perform the task are learned independently; *action level* imitation, by contrast, involves the verbatim reproduction of observed actions. Atkeson and Schaal [8], in their adaptation of this model for robotic imitation, note that true action-level emulation requires full knowledge of the teacher's internal state, which is not feasible in most scenarios; they therefore propose an intermediate level, *task* imitation, whereby actions are abstracted from the internal state to an observable co-ordinate frame (for instance, the velocity of a demonstrator's hand). Again, the concepts of and distinction between *program-level* and *task/action level* imitation played an important role in our own work - our strategic navigation system is designed to capture the goal-oriented, *program-level* behaviour of the human player, while our tactical motion-modelling system uses the computer-game equivalent of both action-level and task-level motor commands (see Sections 4.3.2 and 4.4.3.4).

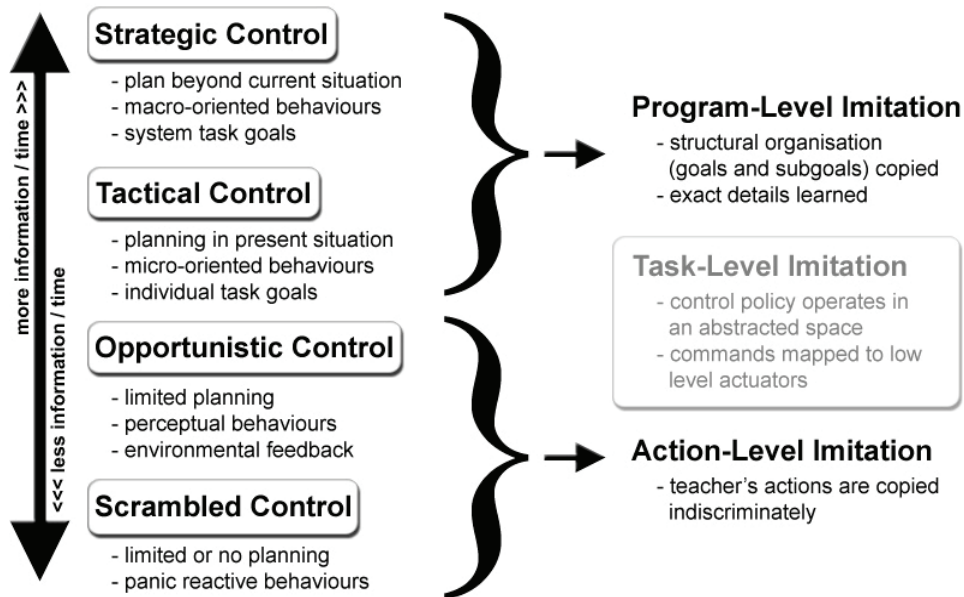


Figure 2-1 - Hollnagel's COCOM [53], the hierarchy of imitation proposed by Bryne and Russon [20], Atkeson and Schaal's distinction between task- and action-level imitation [8], and the correspondence between their respective layers in terms of *planning*. We use an adaptation of COCOM in our investigations (see Section 4.2 for further details).

## 2.2.2 Application in Robotics

Given the compelling biological basis outlined above, it is little wonder that researchers in the field of robotics have turned towards imitation techniques as a powerful means of “bootstrapping” the intelligence of their machines. In this section, we present an overview of some of the more common techniques in robotic imitation, and discuss their applicability to imitation learning in games.

### 2.2.2.1 Motivation

Robot programming at large can be broadly divided into three approaches; *human programming*, where the human simply writes the sequential program which the robot will execute, *reinforcement learning* [2, 69], where the robot is provided with a reward signal or optimization criterion and learns to perform a task through repeated attempts, and *programming by demonstration* or *imitation learning*, where the robot is shown a task a number of times and attempts to reproduce the observed action [95]. The first option is obviously extremely labour-intensive, and results in robots which can perform a limited repertoire of actions with a high degree of competence, but cannot generalise or learn new skills. The Honda humanoid robot [52], which utilises this hard-coding of behaviours, took a decade to develop; it is capable of locomotion and basic object manipulation, but requires direct human instruction to perform more complex tasks.

To alleviate the need for direct programming, *reinforcement learning* has been widely employed to the problem of extrapolating control policies for simple robots. Learning proceeds by exploring each action in every state, observing the reward received, and updating an evaluation function which assigns expected rewards to possible actions. However, as the complexity of these machines increases, there is an exponential explosion in the number of actions which can be taken in each state. The

Honda robot, for instance, has a total of 30 degrees of freedom - even if each of these could only take the values forward, backward or stop on each timestep, this would still result in  $3^{30}$  possible actions per state [113], which is an impossibly large space to search. A method of reducing such massive searches to a more manageable scale is required; imitation learning provides a means to this end.

Demiris and Hayes [27], in describing their own imitative model, give a concise summary of some of the main advantages offered by imitation learning:

- imitation provides the ability to cope with dynamic environments, where preprogrammed knowledge is unavailable or may become obsolete
- it increases the robot or agent's ability to cope with changes in their own state
- it reduces the cost of programming for specific tasks, and expedites learning as compared to a purely reinforcement-based approach
- imitation provides an impetus for the integration of multiple component technologies, a theme echoed by Laird in his arguments that games represent an ideal platform for the development of broad human-level intelligence.

The primary advantage of using imitation as the foundation of learning is that it can dramatically reduce the state-action space which must be considered, by focussing attention on areas which are actually relevant to the task at hand [7, 8, 113]. To build on the example above, a classic reinforcement learning agent may need to interact with its environment for a protracted period of time, or acquire an extremely large number of state-action samples, in order to obtain sufficient knowledge of its task to ensure that its policy converges towards the optimal [2, 69]. If the desired behaviour is first demonstrated by an expert, however, then the observed policy can be used to *guide* the agent's behaviour towards that goal. Cottrell et al [25] describe two approaches to reinforcement-based imitation using *Q-learning* [69], employing the concepts of *action biasing* and *estimated policy exploration*. The former uses a

modified  $\epsilon$ -greedy policy - that is, a policy which selects a random action with probability  $\epsilon$  instead of the current best-known action, to guard against local maxima - by weighting actions based on the number of times the observer was seen to take them; the weights decay over time, causing the expert's influence to lessen as the learner gains experience, and ultimately converging to a simple  $\epsilon$ -greedy policy. Estimated policy exploration involves making a maximum-likelihood estimation of the demonstrator's policy, and choosing actions probabilistically according to this; an *imitation rate* is used to specify the probability with which the learner uses the expert's policy as opposed to its own default policy, and again this decays over time. Using these techniques, Cottrell demonstrates the effectiveness of imitation Q-learning over exploration by training reinforcement agents to play the game "Pong".

Similar hybrid systems, which combine the benefits of both imitation learning and classic reinforcement techniques, have gained increasing popularity in recent years. These typically use imitation to rapidly bring the robot to a reasonable level of competence, and then refine its performance using direct reinforcement approaches which utilise task-specific expert knowledge [113]. The difference between this and our own work lies primarily in the fact that imitation in robotics is generally used as a means to drive convergence towards *optimal performance* of a known task, whereas we are faced with the problem of *deducing* uncertain relationships between the player's state and subsequent behaviour from recorded data, and are just as interested - indeed, in many cases *more* interested - in the various idiosyncratic suboptimalities inherent in human behaviour (see Section 2.3.3 for a more detailed discussion). Imitation can be further classified as either *passive* or *active* [26]; under the former approach, the learner perceives the environment and the demonstrated action, recognises it, and later attempts to reproduce it, while *active* imitation involves the learner internally generating and testing a series of candidate matching actions during demonstration. The passive imitation model is based very closely on Meltzoff and Moore's *Active Intermodal Mapping*, mentioned earlier [85].

### 2.2.2.2 Computational Formalisation and Imitation Paradigms

Schaal [112] offers a computational statement of imitation learning in robots. Motor control, he observes, requires that commands of the appropriate magnitude be issued at the correct time in response to both internal and external stimuli, with a specified behavioural objective. Formally, this involves finding a control policy  $\pi$  such that:

$$u(t) = \pi(z(t), t, \alpha)$$

where  $u$  is the vector of motor commands,  $z$  is the vector of relevant internal and external states, and  $\alpha$  is the set of parameters which need to be adjusted, e.g. the weights of a neural network. Secondly, he defines a generalised *evaluation function* which specifies the level of success achieved by an imitator:

$$J = g(z(t), u(t), t)$$

He notes that defining  $J$  for each particular imitative task is a complex problem. For instance, should the evaluation be strictly goal-based - that is,  $J$  is small if the imitator successfully grasps an object - or should the metric require that the *specifics* of the behaviour - the precise pose and manner in which the object was grasped - also be replicated? As part of our work, we will in later sections provide a computational formalisation tailored to the requirements of games-based imitation learning, and propose approaches to both numerical and perceptual evaluation of imitative agents.

Beyond this generalised formalisation of imitation learning, a number of different paradigms can be identified within the robotic imitation literature. Early investigations centred upon *symbolic imitation*, and relied largely upon FSM-like deliberative processes; as the field evolved and more modern techniques were adapted for imitative purposes, *control-based* and *statistical* approaches began to emerge, though the choice of paradigm in even contemporary research depends largely upon the specifics of the task at hand. Even more recently, *biological* models

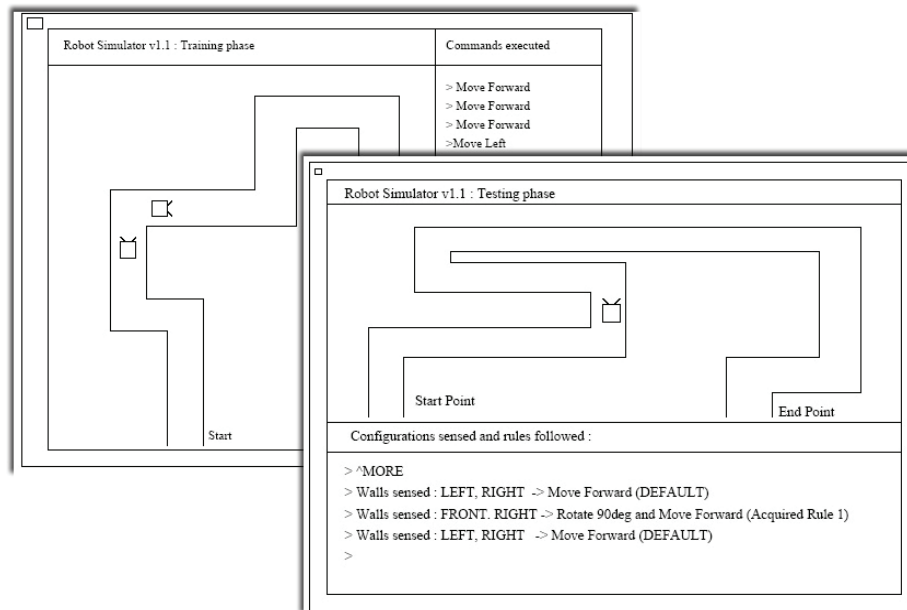
have been proposed, which seek to emulate the imitative functions of the human brain at a comparatively low level, drawing heavily upon the neurological and behavioural precedents discussed in the previous section. These aim to produce more general, less task-oriented imitative models; other researchers continue to focus on the goal of high-level demonstration learning, abstracting away from biological fidelity. The following sections outline these differing approaches, and provide examples of how imitation is carried out in each.

### 2.2.2.3 *Symbolic Imitation*

Symbolic imitation, as the name implies, involves constructing a symbolic representation of the environment, analysing the observed actions in terms of *subgoals*, and deriving a series of *if-then* rules. Prevalent during the early phases of robotic imitation-learning research in the 1980s, early efforts commonly involved physically pushing a robot through a movement sequence in order to proprioceptively extract the ruleset [112] or utilising industrial robots to perform demonstrated tasks [113]. In the latter case, the demonstrator would remotely operate a robotic appendage, recording sensor information regarding its orientation, position, velocity, etc; the example movements were then manually segmented into subgoals and state-action transitions, and finally consolidated into graph-based representations - that is, FSMs. Despite refinements such as robots capable of processing *visual* information or data-glove input, many approaches to imitation are still based upon similar symbolic techniques [112]; Kuniyoshi et al [73], for instance, use symbolic methods to teach a robot various block-assembly tasks by visual demonstration, while Demirir and Hayes [51] train an apprentice robot to negotiate a simple maze by following an expert robot and constructing a symbolic representation of its actions. Figure 2-2, a diagram of their experiment, serves as a typical example of symbolic imitation.

Although symbolic reasoning of this kind is widely employed in imitation, it remains a comparatively laborious process, and the benefits of learning by demonstration are

attenuated somewhat by the necessity for manual interpretation of the observed data - indeed, as detailed earlier, finite-state machines are precisely what we wish to *avoid* in our own work. In addition, many tasks are difficult to describe in purely symbolic terms [113]. While relations such as `move_forward` or `above(boxA, boxB)` are easily captured, how does one describe the dynamics of a tennis serve in such neat language? For these reasons, research into non-symbolic imitation techniques has become increasingly prevalent.



**Figure 2-2 - An example of symbolic imitation learning, as presented by Demiriz and Hayes [51]**

#### 2.2.2.4 Control-Based Imitation

Control-based imitation despatches with the need for a symbolic parsing of perceived actions. Here, the control policy is specified in advance - often as a predictive *forward model* - and the observation data is simply used to derive the *parameters* it requires; the demonstrator's role is thus to *shape* the behaviour of the control system. Atkeson and Schaal [7, 8], for instance, use prior knowledge of point-mass physics to learn a model from a single demonstration of the swing-up problem, where the robot



must swing a pendulum from the inert position to vertical and balance it at that point. They found that direct imitation fails due to unavoidable physical constraints, but by learning the *task-level intent* of the demonstrator and using his initial trajectory to seed an optimization process, the robot was able to adapt to the task after a number of trials. Amit and Mataric [3] present a hierarchical architecture which abstracts observation data from a series of forward-inverse control models at the lowest level to a probabilistic framework of sequence learners at the top. Grudic and Lawrence [46] describe an experiment wherein a robot apprentice equipped with a camera learns to mimic the behaviour of a human-controlled demonstrator robot; they accomplish this by approximating the multidimensional mapping from raw 1024-pixel input images to actuator outputs as the sum of a cascade of 2-dimensional functions, and show the ability of this approach to generalise from a small training set in the presence of considerable noise. Schaal [114] employs a system of nonlinear differential equations to describe the actions of a humanoid robot swinging a tennis racquet; demonstration data is then used to learn a set of local parameters for the control model (see Figure 2-3). Nakanishi et al [90] use a similar approach to reproduce humanlike biped locomotion, and succeed in improving upon the “bent knee” posture typically generated by standard walking algorithms.

Control-based approaches to the problem of imitation, while efficient in cases where the desired behaviour can be naturally modelled as a parameterised function, again prove difficult to apply in instances where the agent is required to reproduce a broader range of behaviours, or when insufficient information is available to allow an *a priori* specification of the control policy. In our case, it is clearly not feasible to describe the complex behaviours exhibited by humans during play as a set of formulae; indeed, it is the goal of our research to *derive* a control policy based on observation, rather than imposing one in advance. For this reason, approaches based on *statistical analysis* of the demonstration data would seem to be the best choice for imitation in games.

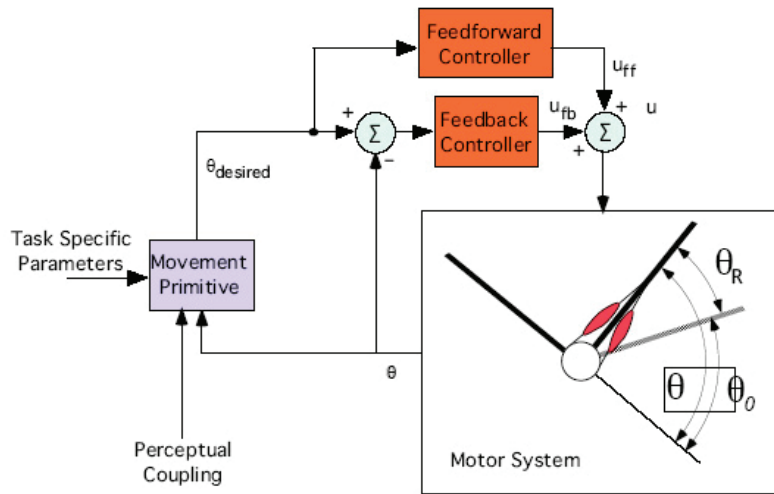
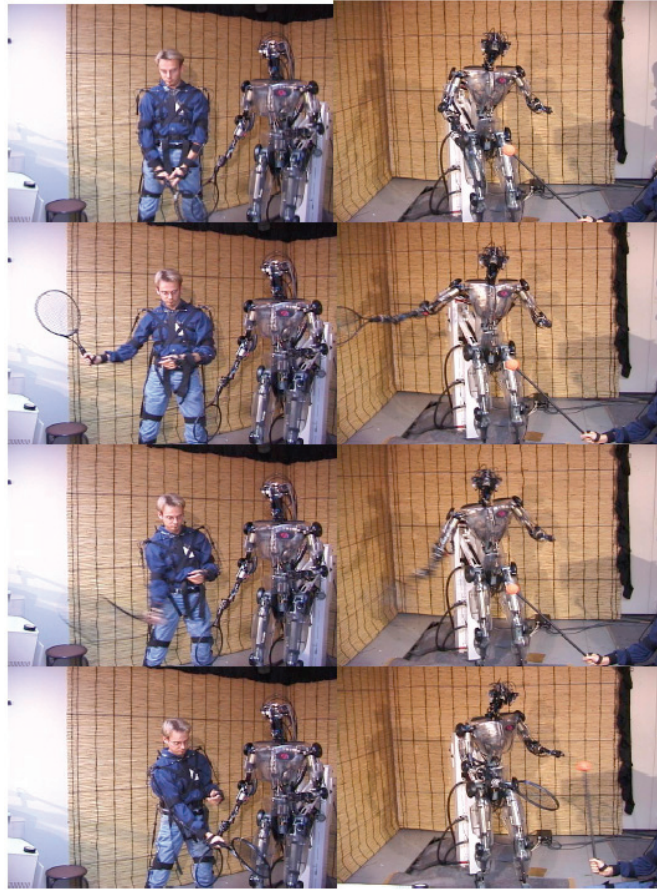


Figure 2-3 - Schaal's tennis-swing demonstration and the associated control model, as described in [114]. Note that the model parameters obtained from demonstration are termed *movement primitives*; this concept is discussed in Section 2.2.2.7.

#### 2.2.2.5 *Statistical Imitation*

*Statistical* approaches to imitation learning do not assume an a priori control model; instead, they attempt to match perception and action based on statistical analysis of the observation data. Dillman et al [29], for example, use fuzzy clustering and an automated segmentation algorithm to learn the hand trajectories of a human performing simple pick-up-set-down and peg-into-hole tasks, and employ radial-basis networks to reduce the perception and action spaces to those components which were relevant during demonstration. Mataric, Jenkins, Fod et al [37, 60-65] use various segmentation, dimension-reduction and clustering techniques to automate the derivation of movement sequences from human performance data - see Section 2.2.2.7 for a further discussion of their experiments with *motor primitives*. Statistical imitation learning has even been used to transfer skills from human experts to lesser-skilled human apprentices; Nechyba and Xu [92] train neural networks to learn the responses of individuals who successfully complete a virtual simulation of the pole-balancing problem, and use these networks to provide real-time advice to others who were unable to perform the task unassisted.

Given its lack of reliance upon expert knowledge, predefined models or manual processing, we concentrate on *statistical* approaches in our own work.

#### 2.2.2.6 *Biological Models*

Over the past few years, an increasing number of contributions have sought to develop artificial models of the cognitive processes which endow humans with their imitative faculties, by employing as little abstraction from the original biological systems as is feasible. Billard [13] defines three progressive levels of modelling for imitation learning:

- **theoretical modelling** - deriving conceptual models of the cognitive mechanisms behind imitation, based primarily on behavioural studies;
- **computational modelling** - constructing models corresponding to the brain areas and neural processes involved in these cognitive mechanisms, thereby providing an explicit functional description of the computation required for each form of imitation; and
- **robotic modelling** - designing hardware-implementable algorithms to realise the computational models.

Billard criticises the fact that much of the robotic literature concentrates almost exclusively on the *engineering* challenges inherent in designing machines with humanlike ranges of motion, realising their imitative faculties by simply recording the movements of the demonstrator's limbs as a series of torques and angular displacements, and utilising straightforward planning techniques to reproduce them. Herein lies something of a philosophical parallel with our own work; we have already noted that commercial games, often to their detriment, continue to employ simplistic approaches to artificial intelligence, and have observed that games - by virtue of providing detailed recordings of human cognitive and decision-making processes in operation, as opposed to mechanical limb data - are *in some ways* a more interesting platform than traditional robotic imitation learning. Billard argues that the objective of computational and robotic modelling should instead be to investigate the relationship between neurological studies and the imitative faculties we observe both in ourselves and in other animals; in other words, to establish the means by which various neurological phenomena contribute to high-level imitative function.

Biological models of imitation have, understandably, drawn heavily upon mirror neurons for their inspiration, since these represent the most compelling explanation for our imitative capabilities. Demiris [28] proposes a model of the mirror neuron system comprising a dual route structure; a *generative route* which uses a forward

model to make predictions about the next state of the system (i.e. the robot's body), and a *learning route* which activates if the demonstrated behaviour is unknown to the imitator. A *behaviour* controller - basically an inverse model - is used to produce the motor commands required to move the robot from one pose to another. In keeping with its biological foundation, the same system is capable of both executing a sequence of behaviours - by inputting its current state to the behaviour module and generating an estimate of the next state, against which the eventual afferent state can be compared - *and* of imitating actions, by inputting the demonstrator's *perceived* state instead of its own. Johnson and Demiris [66] subsequently extended this concept by arranging multiple instances of the system in a hierarchical manner, improving the accuracy of imitated behaviours by successively specialising candidate actions as they pass from more abstract graph-based representations at the higher levels to executable *motor commands* at the lowest. Oztop and Arbib [97] incorporate both mirror neurons and other brain areas necessary for the realisation of *grasping* motor behaviours in their MSN1 model (see Figure 2-4). They demonstrate, using a virtual arm simulation, that it is capable of learning a variety of such actions (*precision* grasping, *power* grasping and *side-grasping*) from visual demonstrations of a human performing the same manoeuvres, using the concept of *grasp affordances* - that is, the subconscious computation from sensory cues of the potential means of manual interaction with an object - discussed in the previous section.

Elsewhere, Billard [15] describes a model composed of interconnected modules which are explicitly designed to emulate specific regions of the central nervous system, following the same neural functional decomposition as found in primates. These modules are responsible for *visual processing* (the temporal cortex module), *motor control* (the primary motor cortex and the spinal cord) and *learning* (the cerebellum and premotor cortex modules, in the latter of which resides an implementation of the mirror-neuron system). Billard notes that - with the exception of the spinal cord module - the internal construction of each module does not

precisely duplicate that of the corresponding brain area; the model's biological component instead lies in its modular division, the manner in which said modules are connected, and the fact that each module *does* implement the relevant high-level functions of the corresponding brain regions, using abstract neurons as building blocks. Billard demonstrates that this model - though prone to low-amplitude and incorrectly-paced reproductions of the demonstrator's behaviour - has the capacity to imitate repeating patterns such as walking and dancing, can learn the movements of any limb, captures precise manipulations of the extremities, and is capable of using video data of a live human performance as its input.

There is one further biologically-inspired technique which has received sufficient attention - even in otherwise abstract, non-biological models - to warrant some additional discussion; namely, *movement primitives*. In the following section we discuss this concept, its ramifications, and its relevance to our own work

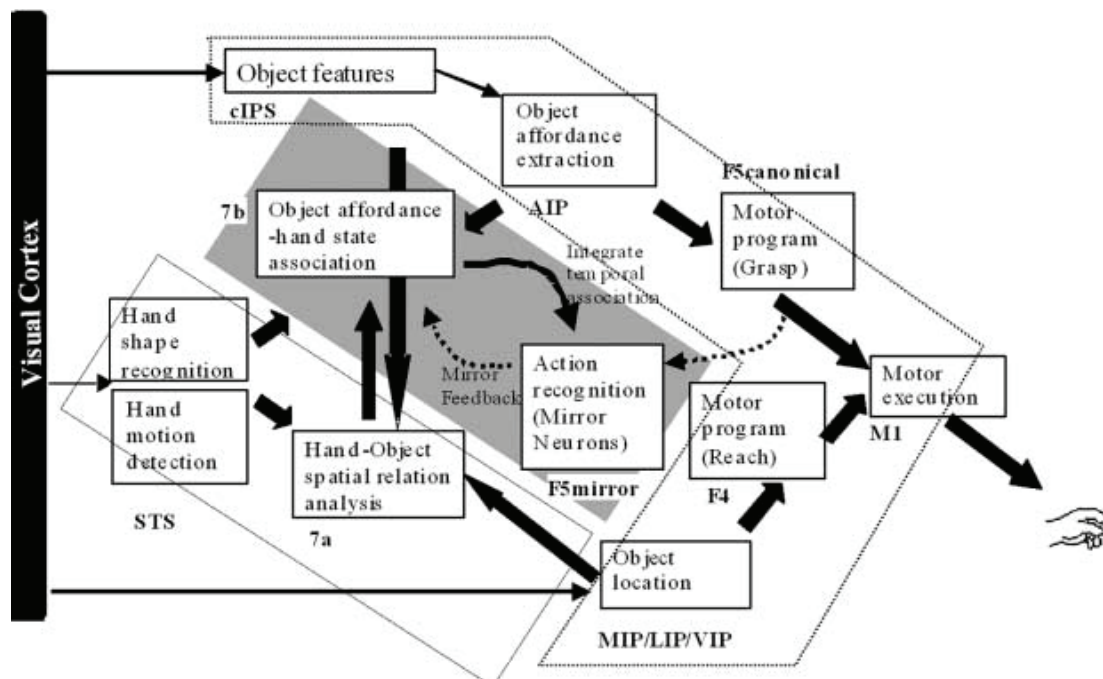


Figure 2-4 - The MNS1 model for imitation of grasping behaviours proposed by Oztop and Arbib, illustrating how biologically-inspired models of imitation seek to achieve the minimal possible deviation from natural systems.

### 2.2.2.7 Movement Primitives

*Movement primitives*, variously known as *motor primitives*, *action units*, and *motor schemas*, are a basis set of modularised motor programs which are sufficient, through combination operators, for generating entire movement repertoires [37]. Though clearly a biological model insofar as it draws upon evidence of similar schemas in humans and animals (see Section 2.2.1.4), approaches to the derivation and application of motor primitives span all three of the paradigms mentioned in the previous sections; a motor primitive may be something as simple as the symbolic “*move forward*” command from the Demiriz & Hayes experiment [51]. As Schaal notes, however, such low-level approaches do not scale well to complex systems [113]. The more information a movement primitive can incorporate without losing its modularity, the better; primitives such as “*reach*”, “*swing*” and “*grasp*” encode complete temporal behaviours, but are general enough to allow a degree of parameterisation and combination.

Leaving aside *symbolic* systems in which primitives must be defined manually, there are two principal types of motor primitive, reflecting the differing forms of imitation outlined earlier:

- ***Innate*** primitives, which are implemented using a predefined *control model*
- ***Learned*** primitives, which are *statistically* derived from the observation data

Schaal et al [115] describe an approach to *innate* skill learning based on using nonlinear differential equations as dynamic movement primitives; indeed, the robot tennis-swing experiment described above [114] used this system as its control model. The biped locomotion experiment outlined in [90] adopts the same approach, using a set of central pattern generators to drive the walking movement of the robot. Movement primitives have also attracted attention in computer vision. Ilg, Giese et al [41, 58] describe a system for extracting primitive movements from motion-captured

video, and use spatio-temporal morphable models to linearly combine them; they show the effectiveness of this technique in reproducing facial expressions and synthesizing realistic motion from demonstrations of different martial arts.

From our own perspective, some of the most interesting work in the field of *learned* motor primitives has been carried out by Mataric, Jenkins et al at the Robotics Research Lab of the University of South Carolina [31, 37, 60-65, 82]. They present a framework for the automated segmentation and derivation of movement primitives from recorded limb motion data. Each observation sequence records the relative angular positions of four degrees of freedom (DOFs) along the limb at each timestep [37]. These sequences are *segmented* at points where the sum of the squared angular velocities falls below an empirically-determined threshold, i.e.  $z = \dot{\theta}_1^2 + \dot{\theta}_2^2 + \dot{\theta}_3^2 + \dot{\theta}_4^2 < \delta$ , and the segments *interpolated* to a standard length of 100 elements. The vectors of each DOF are concatenated to give a single 400D vector representing the movement within that segment; PCA (*principal components analysis*), a technique which computes a new set of basis vectors for a given dataset such that the transformed components are uncorrelated and ordered by decreasing variability, is employed to reduce this vector to an 11D representation while retaining most of the motion information. Finally, the reduced segments are *clustered* to produce a small set of basis actions, which can be *reconstructed* by projecting back to the input space and temporally resizing the resulting vector to obtain the original movement. Each cluster thus represents a motion primitive - when executed in isolation, they result in full strokes of the arm from one pose to another. Later experiments used *spatio-temporal isomap* [60] to better capture the nonlinear, temporally-dependent nature of the observed behaviour. They further discovered that applying ST-isomap a second time on the reduced-dimension embedding uncovered *behaviour units*, consisting of sets of individual action primitives. This allowed for the sequencing of primitives to reproduce full temporal behaviours; by examining the last frame of the current action, a list of valid *successor candidates* is generated from



the behaviour unit set, based on whether or not a transition from the last frame of the current action to the first frame of the next would require an excessively large change in joint position.

A number of the concepts explored by Mataric et al proved useful in the development of our *tactical imitation* subsystem, as described in Chapter 4.4.

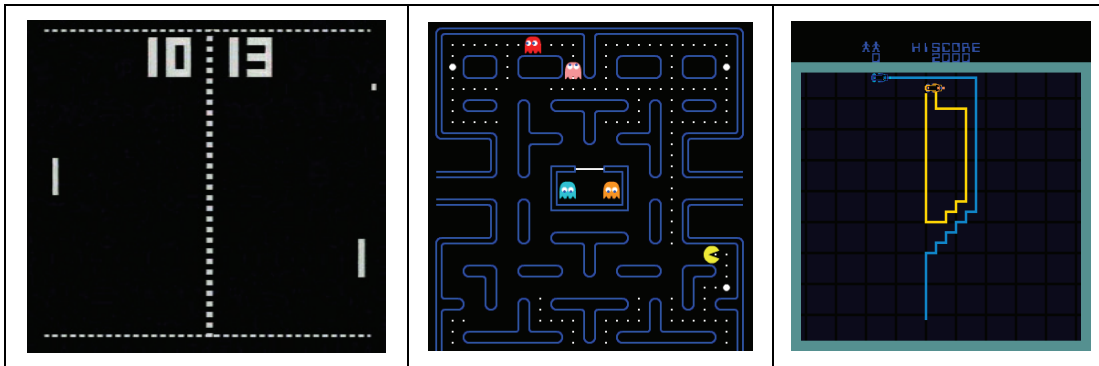
## 2.3 Games-Based Imitation Learning

### 2.3.1 Motivation and Existing Work

Despite the interest in imitation learning exhibited by the robotics community, very little work has been done in applying similar techniques to the field of game AI. Furthermore, as is clear from the overview above, much of the robotics literature focuses primarily on the engineering challenges of imitating physical human motion. Given that the process of generating training data for games is both rapid and straightforward compared to robot imitation, that some games already possess large online libraries containing vast number of samples, and that - rather than limb movement data or similar, as is common in robotics - these recordings encode the actual *behaviours* of human players under rapidly-changing conditions and against opponents of comparative skill, computer games would seem to offer huge potential for imitation learning and, indeed, the AI community at large. The goal of imitation learning in computer games, expressed in its simplest form, is to construct models which explain and accurately reproduce behaviours recorded during human gameplay sessions, thereby '*reverse-engineering*' the player's decision-making process from its observed results.

Some scattered examples of imitation learning in games do exist. Sklar et al [117], for instance, report on their work in developing a game based on the film “Tron” which allows humans to play online against computer-controlled opponents. These

matches are recorded, and the data used to train subsequent generations of agents. However, this game imposes an extremely limited repertoire of behaviours on the player - at each interval, the player can move left, they can move right, or they can go straight ahead. Many contributions still persist in using similarly restricted games - such as Pong [79, 111] or Pac-Man [140, 141] - which are too simplistic to yield much in the way of interesting results, except for highly specific investigations (Figure 2-5). Using commercial games for imitation research would seem to be a far more profitable enterprise - see Section 2.3.4.1 for further discussion.



**Figure 2-5 - Pong, Pac-Man and Tron are too simplistic to provide much scope for investigation.**

Geisler [40] in his Master's Thesis, investigates the relative performance of three machine-learning approaches - ID3 decision trees, naïve Bayesian classifiers, and multi-layer perceptrons - when applied to the task of learning simple actions from human demonstration in the FPS (*first-person shooter*) game "Soldier of Fortune 2". He demonstrates that an MLP performs significantly better than the other approaches, at the cost of an increased learning time. However, Geisler imposed a number of artificial restrictions on the recorded data used in the experiments; for instance, he discretises each of the low-level actions under investigation - *move direction*, *accelerate*, *face direction*, and *jump* - into either binary values or indices of four 90° sectors, as appropriate. In other words, the agent can move and aim only at right-angles, effectively reducing the simulation from a full 3D virtual world to an equally limited state as Sklar's "Tron" game. Additionally, his analysis comprises only an

inter-algorithm comparison of learning accuracy and speed of convergence; he does not include a discussion of the observable in-game behaviour resulting from each approach, and it is therefore difficult to ascertain how successful his agents actually were in imitating their human exemplars.

Sukthankar and Sycara [122] propose a means of automatically recognising sequences of team behaviours using the game “Unreal Tournament”. They remark upon the particular difficulty of determining the exact transition point between behaviours, a subject which we will revisit in the concluding chapter of this thesis. To circumvent the problem, they employ a series of overlapping time windows during which a single behaviour is assumed to be dominant, and train a set of Hidden Markov Models to recognise each. However, their work examined only 3 specific behaviours, which they had defined in advance and *instructed* the human volunteers to perform - the HMMs were therefore simply classifying the best-matching behaviour from an extremely limited list of known actions. This approach is obviously not suitable for bottom-up analysis of a freeform game session, though the complexity of distinguishing even between three *known* behaviours does provide some insight into the scope of the problem.

Elsewhere, Zanetti and el Rhalibi [142] employ an evolutionary neural network approach to imitation learning in the game “Quake 3”. Feedforward MLPs were defined for each of three behaviour categories - combat, combat movement, and general navigation - and trained from human gameplay samples using genetic algorithms to optimise the network’s weights. Interestingly, they used a waypoint system to deconstruct the human player’s navigation routes into sequences of simpler paths for learning. Their experiments demonstrated that this architecture was adept at capturing certain forms of behaviour, but susceptible to redundant and/or noisy sample data in other areas - and, most importantly from our perspective, that the resulting bot did not exhibit humanlike behaviour or strategic planning faculties.

These results are corroborated by Thureau et al in their early investigations [12], wherein they used *self-organising maps* (SOMs) to identify manifolds within the dataset, and trained a neural network at each node; it was found that such holistic attempts to imitate human gameplay did not capture the relevant behaviours at a granularity sufficient for accurate reproduction. Indeed, Zanetti and el Rhalibi note that their approach would likely benefit from a separation of concerns - the combat network, for instance, should be “subdivided into three [neural networks] to better learn and perform the three managed actions”.

It is for these reasons that we sought to develop separate, specialised approaches appropriate to each of the three clearly defined categories of behaviour - long term *strategic* behaviours, mid-term *tactical* behaviours, and short term *reactive* behaviours - outlined in the discussion of Hollnagel’s COCOM earlier. A further discussion of this issue will be presented in Section 4.2.

### 2.3.2 Believability and Performance

An important issue when considering how best to approach this work concerns the relative importance of *believability* and *performance* in game agents. Should agents, in other words, be designed in such a manner as to maximise the illusion of being a human player? Or should they provide the greatest challenge possible for their human opponents? The two are not necessarily synonymous. Harnad [49], in discussing the wider implications of Turing’s famous paper, notes that his test cannot be passed simply by virtue of an artificial participant *beating* a human - but only if, in the process, its opponent or an observer becomes convinced that the AI is *itself* a human. To illustrate this point, Livingstone [79] cites Big Blue’s defeat of Gary Kasparov in 1997. The feat itself was *not* evidence that the chess machine had passed the Turing test, but Kasparov’s subsequent accusations of duplicity - that he had actually been playing a human opponent in the guise of a machine - did raise the question, at least

in an academic sense. It is generally understood, however, that Kasparov's allegations arose due to his lack of expertise in the field, and his consequent incredulity at having lost to an artificial opponent.

In terms of our own work, we approach the domain first and foremost as artificial intelligence researchers. As such, we are more interested in using computer games as a research platform - in analysing human behaviour for the purposes of producing *believable* facsimiles - than in providing the player with a challenging opponent. In any case, as discussed earlier, it is a comparatively *trivial* task to write challenging game agents; the only drawback being that such agents often rely on repetitive patterns of behaviour, direct access to the gamestate, or outright cheating. A third factor, relating both to believability and to performance, should also be noted here; namely, that - largely due to their utilisation of the exploitative techniques listed above - the difficulty involved in defeating traditional agents does not necessarily result in a more *enjoyable* experience for the human participant, and often produces precisely the opposite effect. Yannakakis and Hallam [140, 141] demonstrate that as a computer opponent approaches optimal behaviour, human players report decreasing satisfaction with the game experience; their investigations indicate that *fair*, *diverse* and *strategic* behaviours are central to maximising the player's enjoyment. Believability is an infinitely more difficult goal than performance, but an agent which can accurately reproduce human motion, behaviour and planning will therefore provide a far more engaging experience for the gamesplayer than is currently offered by traditional NPCs. In this regard, academia has much the same goal as the games industry; Livingstone notes that the principal requirement for modern computer games is not *unbeatable* AI, but *believable* AI.

At the same time, as we have already mentioned, capable players generally outperform even exploitative traditional agents, providing the ultimate challenge for other human participants despite exhibiting inescapable human error and

suboptimality. In summary, while our primary interest lies in believability, we are also of the opinion that adopting imitative approaches to game AI will ultimately lead to agents which are less predictable, less repetitive, more adaptive, and thereby more *enjoyably challenging* for human opponents.

### 2.3.3 Computational Modelling for Games

The intuition behind imitative approaches to game AI is straightforward; an elegant (though, as he notes, highly generalised) formulation is provided by Thureau [129], deriving in part from Schaal’s formalisation of robotic imitation as discussed earlier. Given a situation encountered during a game session, which we view as a *state vector*  $s$ , and a collection of *environmental* factors  $e$ , a human player’s *behaviour* can be computationally modelled as the temporally-dependent sequence of meaningful *actions*  $a$  which he executes in response to that situation; that is,

$$a_{t+1} = f(s_t, s_{t-1}, \dots, s_{t-n}, e_t, e_{t-1}, \dots, e_{t-n})$$

where  $a_{t+1}$  is the action to be executed on the next timestep, and  $s_t$  is the observable state of the game world at time  $t$ . By examining recorded samples of human gameplay, it may be possible to derive the mapping  $f$  and thus to create an agent which can accurately imitate humanlike behaviour even in novel scenarios. Alternatively, Thureau notes that imitation can be understood as a classification problem rather than the regression outlined above; he proposes using a Bayesian approach to choose the action  $a_j$  at timestep  $t+1$  possessing the highest probability among all actions observed in the training dataset, given the current state  $s_t$ :

$$a_{t+1} = \arg \max_j P(a_j | s_t, s_{t-1}, \dots, s_{t-n}, e_t, e_{t-1}, \dots, e_{t-n})$$

The fundamental premise of our research, then, is to parse the set of states  $S = [s_1 \dots s_k]$  and corresponding actions  $A = [a_1 \dots a_k]$  from recorded gameplay samples,

and to deduce models which both explain the relationship between them and are capable of generating sensible predictions in unseen situations.

There are, of course, some important distinctions which must be drawn between the robotic imitation research outlined in the preceding sections and that proposed in this thesis. Firstly, the requirements for reproducing the actions of a gamesplayer in an interactive environment are obviously quite different; for instance, the player's behaviour is influenced in large part by his location and co-occurring events, whereas the majority of the approaches outlined above aim to reproduce isolated humanoid motions. Nonetheless, the mental faculties required to navigate a virtual world - when issues of actuator/muscle control are removed from consideration - are likely to be quite similar to those required in the real world, and games may thus provide some interesting insights into human spatial awareness, memory, reasoning, and so forth. Secondly, while the objective in robot imitation is generally to reduce the time taken for the machine to perform an action optimally, the goal of our research is to generate game agents with a convincing façade of humanity. A good example of this distinction can be found in Dillman et al [29], wherein the authors mention a number of inherent problems in robot imitation learning. Since humans naturally tend to perform tasks in a competent but suboptimal manner, they can generate a significant amount of “noise” during demonstration. The most prominent traits of human performance which can lead to imitation problems are:

1. *unnecessary actions* which do not contribute towards achieving the goal
2. *incorrect actions* which must be compensated at a later time
3. *unmotivated actions* which cannot be analytically linked to the sensor data

The authors then proceed to describe methods for the removal of these unwanted artefacts. From our perspective, on the other hand, these idiosyncratic, suboptimal, quintessentially *human* behavioural traits are an important facet of what we are

attempting to capture and must therefore be *incorporated* into our agents rather than being eliminated.

#### 2.3.4 Selecting a Testbed: Quake 2

Our first step, naturally, was to identify a suitable testbed which would allow for the analysis of human gameplay and the implementation of artificial agents trained using this data. It soon became apparent that iD Software's *Quake 2* was the *de facto* standard in research using commercial games; Laird had started the trend, and it had been picked up by others [12, 131, 132]. Laird's research centred upon rule-based systems rather than in learning by imitation, but given that Quake 2 has built-in features which permit the player to record himself during a match (to "demo" or *DM2* files), it remains a viable choice. In addition, the *first-person shooter* genre - of which Quake is a prominent example - is particularly attractive from an imitation-learning point of view; this is discussed at greater length in Section 2.3.4.2.

Quake 2, first released by ID Software in 1997, is a combat game played from a first-person perspective - that is, the player sees the game world through the eyes of a virtual avatar. Players explore a three-dimensional environment littered with weapons, bonus items, traps and pitfalls, with the objective of reaching a particular endpoint and/or defeating as many opponents as possible. In the single-player mode, these opponents are agents realised using the kind of finite-state machines described earlier; more interesting from our perspective is the multi-player mode, where humans compete against one another in specially-designed *deathmatch* arenas.

Quake 2's multi-player mode is a simple client-server model. One player starts a server and other combatants connect to it, entering whatever environment (known as a *map*) the instigating player has selected. These opponents then fight until a specified time or kill limit has been reached, at which point the game restarts on the



same map or another, as dictated by the server. Thus, in order to realize artificial *bots* (agents), a means of establishing a session with the Quake 2 server, handling incoming gamestate information and communicating the bot's actions back to the server must be implemented. Further pertinent details of the Quake 2 environment are discussed in Chapter 3.

Quake 2 further facilitates the *recording* of matches from the perspective of each player; these *demo* files contain an edited copy of the network packet stream received by the client during the game session, capturing the player's actions and the state of all entities at each discrete time step. A parser capable of reconstructing the gamestate from these files is therefore required.

Full details of the Quake 2 environment, and the QASE API designed to facilitate our research using it, are provided in Chapter 3.

#### 2.3.4.1 *Why Existing Commercial Games?*

Rather than take advantage of commercial games, many researchers - in the fields of both machine learning and traditional game AI - have often chosen to work in environments which inherently limit the scope of their investigation [91]. While it is true that specific aspects of human behaviour learning can be studied in specially-designed environments such as that presented by Sklar [117], there are a number of advantages in choosing an existing game as opposed to creating one. Firstly, Quake 2 and other commercial games were written to provide players with a compelling, challenging experience rather than with research in mind; they therefore represent *objective* models of human interaction, in situations far more complex than those faced in traditional robotic imitation learning or most purpose-built environments. Secondly, games can be seen - and should be utilised - as mass-produced *universal testbeds*, offering a vast array of genres suited to research of equally divergent varieties; researchers working in commercial games who wish to reproduce others'

work, or test it against their own, do not need to acquire testbeds from the original authors or attempt to recreate them from scratch. Thirdly, rather than focussing attention on a single element of human behaviour, commercial game sessions generally comprise a full spectrum of *multiplexed* reactive, strategic and tactical behaviours, executed in real-time competition against opponents of equal skill; data harvested from such sessions therefore provides extremely fertile ground for analysis, our task being to decouple and uncover the influences which drive the observed behaviours and implement algorithms to realise them. Finally, unlike other areas of machine learning where the collection of data requires considerable time and effort, experimenting with mature and well-established commercial games such as Quake 2 allows us to utilise vast online libraries of existing demos, and new samples can be generated rapidly and inexpensively.

#### 2.3.4.2 *Why First-Person Shooters?*

One of the primary advantages of using this type of game in preference to others is that it provides a rich, but comparatively *direct* mapping of human decisions onto agent actions. This is in contrast to many other genres [21, 74, 91], where the agent's behaviours are determined in large part by factors other than the player's decision-making process. In real-time strategy games, for instance, the player typically controls a large number of units, directing them in various scheduling and resource management tasks. Although the player is responsible for, say, instructing his followers to engage in battle against an enemy faction, the specifics of *how* the confrontation unfolds is handled on a per-unit basis by the game's AI routines. In sports simulations, only a single character is usually under the control of the human player at any given time; the interactions of his teammates, though perhaps influenced by the tactics chosen before play, are managed from one timestep to the next by the computer. In adventure games, imitating human performance would first require an AI capable of visually recognising everyday objects and comprehending

their significance, as well as an ability to understand and partake in conversations with other characters; this prerequisite level of common-sense reasoning makes the genre infeasible for imitation purposes, at least at present. While these games do offer many interesting challenges for AI research, as partially enumerated by Laird [74] and Fairclough et al [35], the attraction of Quake 2 and other first-person shooters - to researchers and gamers alike - lies in the minimal degree of abstraction they impose between the human player and his virtual avatar.

Additionally, the FPS genre - while incorporating unrealistic features such as invincibility, teleportation, etc - is nonetheless an abstraction of the real world, as opposed to a board or puzzle game with no relevance to reality. Thus, the possibility exists that techniques developed to imitate in-game human behaviour may be adaptable to other areas; indeed, Thureau et al [132] report that they have discussed their approach to action primitives with biologists studying the courtship behaviour of zebra finches. Further discussion of the potential for real-world application of imitative techniques developed in games can be found in Section 6.4.2.

## 2.4 Conclusion

This chapter detailed the field of imitation learning in greater detail, outlining its various advantages over more antiquated artificial intelligence techniques. A discussion of the biological impetus for learning by observation was also presented, demonstrating that imitation is a necessary component of higher intelligence, that it is a neurologically-supported evolutionary development rather than a learned behaviour, and that it is computationally explicable in terms of forward and inverse models; these observations serve to ground the work described over the following chapters in biological precedent. An overview of robotic imitation was then sketched, its adaptation of natural imitative mechanisms explained, and specific approaches which are relevant to our own research highlighted. A discussion of imitation

learning in game AI, our choice of Quake 2 as an experimental testbed, the role of believability- and performance-based metrics in our work, and the formulation of a computational model for games-based imitation learning concluded this chapter.

## 3 The QASE API

### 3.1 Introduction

In the initial stages of our research, it became clear that the available testbeds and resources for game-based AI research of the kind we were pursuing were often scattered, ad hoc and incomplete. We felt that the absence of a unified, low-level yet easy-to-use development platform and experimental testbed was a major impediment to the adoption of commercial first-person shooter games in both academic research and education, as well as representing a major obstacle to our own work. We therefore decided to develop an API which would not only allow us to conduct investigations into imitation learning, but would also provide others wishing to pursue similar work with a solid framework upon which to build. This chapter details the result of this development; the Quake Agent Simulation Environment (QASE).

Typically, simulations consist of three principal components; an *agent*, an *environment*, and an *interface* between them. This chapter details each of these concepts as they relate to QASE. In Section 3.2, we first outline the motivations underlying this work, including those shortcomings of existing APIs which we sought to correct. Section 3.3 discusses elements of the Quake 2 *environment* and QASE’s network layer, which represents the *interface* component of the simulation. In Section 3.4, we then proceed to describe the most significant elements of QASE’s *agent architecture*, including its provision of a bot hierarchy which allows agents to be created from any of several levels of abstraction, its ability to supply the agent with sensory data about its environment, its facilitation of high-level queries about the agent’s condition by transparently amalgamating low-level data, its ability to record and parse demonstration of human gameplay, its integration with the MatLab programming environment [83], its inbuilt AI constructs, and its topology-

learning/navigation faculties, which are drawn from our work in imitation. Some further observations on QASE’s potential for application in both games-based AI research and education, and a discussion of the various institutions that have already adopted QASE in both capacities, conclude this chapter.

### 3.1.1 Publications

The work in this section was published as (Gorman et al 2005) “*QASE - An Integrated API for Imitation and General AI Research in Commercial Computer Games*” in the proceedings of the 7th International Conference on Computer Games (CGAIMS ‘05), where it won the “Best Paper” award.

An extended version was subsequently selected for publication as “*The QASE API - An Integrated Platform for AI Research and Education Through First-Person Computer Games*” in the International Journal of Intelligent Games and Simulations, Volume 4 Issue 2, June 2007 (Gorman et al 2007).

## 3.2 Motivation

In the initial stages of our research, it became clear that the available testbeds and resources for game-based AI research were often scattered, frequently incomplete, and consistently ad hoc. Existing APIs were unintuitive, unreliable and lacking in functionality. Network protocol and file format specifications were generally unofficial, more often than not the result of reverse-engineering by adventurous fans [42]. Documentation was sketchy, with even the most rudimentary information spread across several disjoint sources. Above all, it was evident that the absence of a unified, low-level yet easy-to-use development platform and experimental testbed was a major obstacle to our own work, and a significant impediment to the adoption of first-person shooter games in academic research and education in general.

As a result, we decided to adopt a two-track approach. We would develop techniques for imitation learning in games, while simultaneously building a comprehensive programming interface designed to provide all the functionality necessary for others to engage in this work. This interface should be powerful enough to facilitate high-end machine- and imitation-learning research, while at the same time being suitable for use in undergraduate courses geared towards classic AI and agent-based systems.

The Quake 2 Agent Simulation Environment (QASE) was developed to meet these requirements. It is a fully-featured, integrated API, designed to be as intuitive, modular and transparent as possible. It is Java-based, ensuring an easily extensible object-oriented architecture and allowing it to be deployed on any combination of hardware platforms and operating systems. It amalgamates and improves upon the functionalities of several existing applications, removing the need to rely on ad-hoc software combinations or to comb through a multitude of different documentations; QASE consolidates all relevant information into a single source. It is geared towards machine and imitation learning, but is also appropriate for use with more traditional forms of agent-based AI. Put simply, QASE is intended to provide as much of the functionality the researcher or student will require in their experiments with cognitive game agents as possible.

### 3.2.1 Existing APIs

In the initial, exploratory phase of our work, we investigated a number of candidate APIs, originally intending to adopt one of them for use in our research; ultimately, however, we were not satisfied that any of them provided the tools we would need. Here, we discuss these APIs. While each has positive elements, we explain why in each case we felt that they fell short of our ideal platform, and in what ways we designed QASE to be a preferable alternative.

### *3.2.1.1 Quake 2 Bot Core*

One of the earliest attempts to facilitate bot programming in Quake 2, the Bot Core [123] comprises an implementation of the game’s client-side network protocol written in pure C, together with a very basic template for creating an AI cycle. It soon became apparent that it was a less than ideal platform for our work; parts of the network protocol had been neglected, other parts were not functioning reliably, it required that each agent be compiled into a separate executable, and its potential extensibility was extremely limited, making it an unsuitable choice for high-end research applications in general.

### *3.2.1.2 GameBots*

The GameBots project [1] allows communication between Epic Games’ *Unreal Tournament* and other software, channelling messages to and from the server via a socket interface. Messages are both sent and delivered as ASCII strings, adhering to a predefined format. While this allows a wide range of programs to interface with the server, it is a bare-bones system; the user must write his own parser for the game messages, there are no supporting AI structures or logic included in the API, and communication is handled exclusively through scripting, with no low-level access available. Additionally, the socket interface which exposes the gamestate to external control is implemented via a modified version of the game server; it is therefore not possible to create an agent and connect it to an arbitrary Unreal Tournament match - agents can only communicate with a server running the GameBots mod.

### *3.2.1.3 Quagents*

The Quagents project [19] is intended primarily to propose Quake as a virtual testbed for robot and “ant colony”-style agent simulations. Its approach is quite similar to that of the GameBots API described above; Quagents is a recompiled modification of



the Quake 2 libraries, which exposes the internal workings of the game to external manipulation via predefined script commands. Again like GameBots, the implementation of the actual agent controller itself is left to the user, although a sample application allowing for real-time control of the bot is supplied. However, Quagents also modifies the content of the game itself, implementing a stripped-down version of the original by eliminating many items, simplifying the agent's movement functions, and reducing the interactivity of the bot with the game world (Quagents, for instance, cannot engage in combat against each other).

In the case of both GameBots and Quagents APIs, their respective constraints - the elimination of features present in the original game, the limitations imposed by requiring the server to have the relevant modification installed, the lack of supporting structures and functionality - were among the primary reasons for our ultimate decision to reject them as viable candidate platforms for our research.

#### *3.2.1.4 FEAR SDK*

The most mature of the pre-existing APIs we investigated, FEAR [23] is once again a modification of the game's shared libraries; unlike GameBots and Quagents, however, FEAR also provides a framework for creating the agent controller itself. These must be compiled as DLLs and placed in appropriate subdirectories under the main FEAR mod folder; bots are then deployed by issuing commands from the server console during a game session. This has the drawback, however, of requiring not only that the server be running FEAR - as with GameBots and Quagents - but that the code for all desired agents be present on the same machine when the game session begins. If, for instance, multiple researchers from different institutes wish to compare their agents, they cannot simply connect to a common server and deploy them. FEAR also includes a variety of in-built AI structures, including FSAs, decision trees, neural networks and rule-based systems. While such features are welcome, we felt that the inability (or at least, significant difficulty) of allowing external processing

during the agent's AI cycle was highly limiting. MatLab, for instance, provides a vastly greater range of functionality than that embedded within FEAR, and is already a familiar working environment for many researchers. The SDK is also quite unintuitive in certain respects, which attenuates its utility as an educational aid. Moreover, the fact that the SDK is inextricably linked to the game engine renders it incompatible with other mods, such as the CTF (*capture-the-flag*) team-based modification described in Section 3.4.3, unless the user were to subsume the CTF code into FEAR.

### 3.2.1.5 TIELT

The Testbed for Integrating and Evaluating Learning Techniques [88] is a middleware platform designed to act as a generic intermediary between game engines and decision systems, similar in concept to QASE's MatLab integration architecture as described later in this chapter. For each game, users employ TIELT's inbuilt editor and scripting language to develop a series of knowledge bases; these consist of XML files defining elements such as the objects contained in the gamestate, events that may occur, state transition rules, messaging formats for communication with the game server and decision system, etc. The majority of applications have thus far centred upon real-time strategy games, although a bot has also been created for Unreal Tournament as proof-of-concept. While TIELT is an excellent general-purpose tool for research across different games, it is by necessity removed from the low-level details of each; we felt that our work - and that of other groups interested in pursuing research in first-person shooter games - would be better served by a consolidated API with a wide range of inbuilt functionality. Had we adopted TIELT for the purposes of interfacing our decision systems with Quake 2, we would have needed to write separate software for reading human behaviour data from demo files, dealing with BSP geometry and environmental entities like lifts and doors,

constructing navigation graphs, and so forth; this would have resulted in precisely the kind of ad-hoc, non-reusable amalgamation of software that we wished to avoid.

### *3.2.1.6 QASE: OUR APPROACH*

Having examined the platforms outlined above, we concluded that none provided the combination of simplicity, power, modularity, reusability and extensibility that an educational and research tool of this kind required. We felt, as noted earlier, that the lack of such a platform was a major impediment to the adoption of first-person games in both areas, and sought to remedy this; it was always our intention that, while geared primarily towards our own work in imitation- and machine-learning, the API would be appropriate for the widest possible range of applications. As such, QASE incorporates not only solutions to the various hurdles we encountered in the course of our research, but all the features which occurred to us as being potentially beneficial for others. Written in Java, the API itself consists of a single ~160kb JAR library, which can run unmodified on any JRE-enabled machine. As detailed later in this chapter, it permits low-level access to gamestate and environmental information for those who want it, while also supplying high-level interfaces and convenience functions which perform the necessary gamestate-handling tasks transparently. It provides illustrative built-in AI structures for educational purposes, and draws upon our own research in imitation learning to facilitate the automatic generation (or, optionally, manual construction) of full navigational systems. Its efficient and flexible MatLab integration provides an extremely powerful back-end engine with which researches are already intimately familiar. Unlike the approaches adopted by several of the APIs described above, its network layer encapsulates a full client-side implementation of the Quake 2 network protocol, meaning that it is cleanly decoupled from the server implementation; a QASE agent can connect to and be deployed upon any arbitrary Quake 2 server, Windows or Linux, modified or otherwise. It allows game sessions to be recorded to DM2 files and parsed at a later time, as well as

automated loading and querying of local BSP (*binary space partition*) geometry files to provide the agent with sensory information about its environment. It comes with full, detailed documentation and a series of articles focussing on the network protocol, the BSP file structure, and other subjects which will be of use to researchers embarking upon work in this area. Finally, it is worth noting that QASE is still in active development, and is evolving in response to the comments of groups and individuals who have adopted it; the other APIs mentioned above are, as far as we can determine, no longer maintained.

Over the remainder of this chapter, we will explain each of these QASE features - and how they relate to the environment provided by Quake 2 - in detail.

### 3.3 QASE, Environments and Interfaces

Quake 2, first released by ID Software in 1997, belongs to the genre of *first-person shooter* games. Players explore a three-dimensional environment littered with various *entities* - weapons, bonus items, traps and pitfalls - with the objective of reaching a particular endpoint and/or defeating as many opponents as possible. Most interesting from the perspective of imitation learning is the multi-player mode, where players compete against one another in specially-designed *deathmatch* arenas. A screenshot of the Quake 2 environment and some important features is given in Figure 3-1.

As outlined in Chapter 1, Quake 2 was adopted for a number of reasons. Firstly, it was prominent in the literature, having been employed by Laird in his experiments using the Soar architecture [75, 76, 78]; it is a mature game, and as such the existing resources are somewhat more substantial than for other games. Secondly, rather than focussing attention on a single element of human behaviour, Quake 2 matches require a full spectrum of ‘multiplexed’ reactive, strategic and tactical behaviours from the agent, executed in competition against opponents of equal skill; from the perspective of research, this offers a huge number of potential avenues for

investigation. Thirdly, the first-person shooter genre - while incorporating unrealistic features such as invincibility, teleportation, etc - is nonetheless an abstraction of the real world, as opposed to a board or puzzle game with no relevance to reality. Techniques developed in the game world may therefore prove applicable in other areas, as we discuss at the end of this thesis. Additionally, the major attraction of Quake 2 from our perspective lies in the minimal degree of abstraction the game imposes between the human player and his virtual avatar; recorded Quake 2 game sessions provide very *direct* mapping of human decisions onto observable agent actions. Finally, Quake 2 provides facilities for the recording of game sessions to so-called *demo* or *DM2* files. This is of great benefit to our work, since the ability to capture samples of human gameplay is a prerequisite for imitation learning.

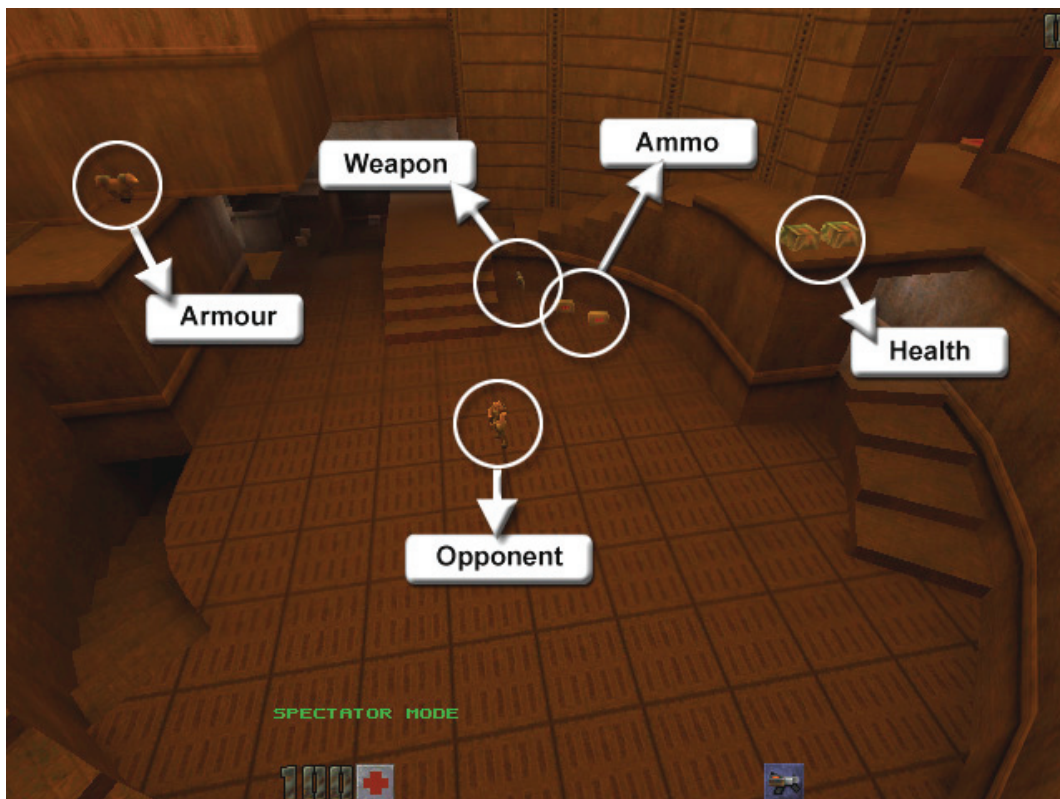


Figure 3-1 - The Quake 2 environment. Weapons are collected to enable the character to fight opponents. Ammunition is collected separately; certain guns share common types of ammo. Armour reduces damage taken for a period of time. Health packs replenish the local character's energy, which when depleted results in temporary death and the loss of a point to the opponent.

As outlined earlier, a simulation typically consists of three components - an *agent*, an *environment*, and an *interface*. In our case, the environment component is provided by the Quake 2 server. We must supply both the agent and interface elements; QASE fulfils both these requirements. In this section, we outline QASE's network layer and its provision of an *interface* to the Quake 2 environment. We also outline the most important features of the environments themselves. The third component, QASE *agents*, will be detailed in Section 3.4.

### 3.3.1 The Quake 2 Environment

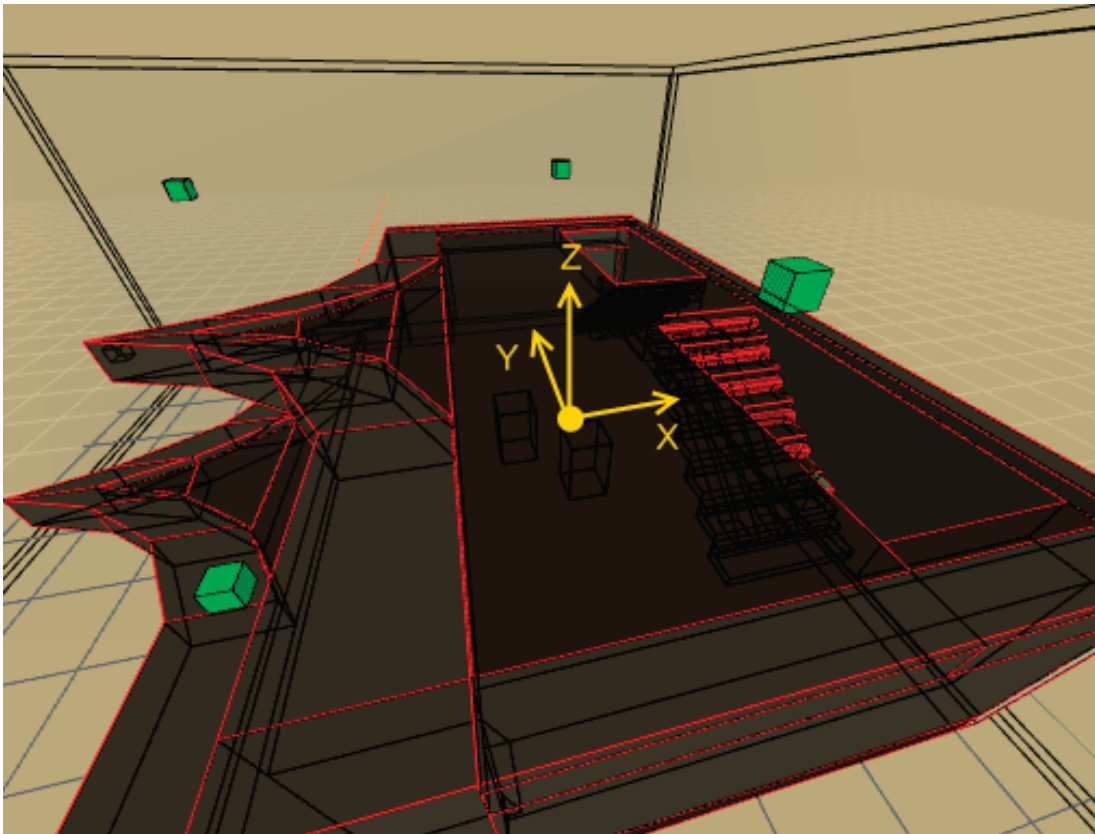
As stated above, the *environment* component of the simulation is provided by the Quake 2 server. But what, exactly, does this environment contain? How are entities categorised? What attributes do they possess? How is the game world itself represented? Here, we outline some of the more important concepts in the Quake 2 environment, a knowledge of which will be required for later sections in this chapter and, indeed, for the chapters to follow.

#### 3.3.1.1 Maps

Each distinct environment in Quake 2, also called a *map*, is defined in the same manner. Physical obstacles, called *brushes*, define the outer boundaries and local features of the environment, while the agent is free to move in the unoccupied space between them. The map itself, and everything contained within it, are projected onto a three-dimensional orthogonal co-ordinate system, using a rotation whereby the vertical axis is denoted as Z in all game data and communications. Details of the maps themselves are not communicated from the server, but are instead stored locally on each client machine, as Binary Space Partition (BSP) files. Further details on BSP files, and their importance in providing the agent with sensory information on its environment, are provided in Section 3.4.5. A wireframe model illustrating Quake 2's co-ordinate system and the concept of brushes is provided in Figure 3-2.

### 3.3.1.2 Environment Entities

Aside from the definition of the map itself, the local BSP files also contain information on certain *fixed* entities within the environment. These are not to be confused with the dynamic entities discussed below; fixed entities are elements such as moving platforms, lifts, teleporters, buttons etc, which the player may in some cases be able to interact with, but which nonetheless remain part of the environment. QASE's BSP parser, in addition to allowing sensory perception of the map's topology, also facilitates high-level queries of these fixed entities; for example, the agent can tell with a single method call whether or not it is currently standing on top of an elevator. This is further discussed in Section 3.4.6.



**Figure 3-2** - A typical Quake 2 map, demonstrating the "brush" concept and the in-game co-ordinate system. Environments are defined by a set of volume boxes; the player is constrained to moving in the free space between them ([inside3d.com](http://inside3d.com)).

### 3.3.1.3 Dynamic Entities

In addition to these environmental features, each Quake 2 map contains numerous *dynamic* entities. These are entities whose original positions are also defined in the static BSP files, but which are managed throughout the course of the game session by continuous server-client communication, due either to the fact that they are *mobile* (i.e. they may be located at any arbitrary point in the map on each timestep) or that they may on some occasions be *absent* (e.g. when a player collects an item, that item vanishes for a certain period of time). On each timestep, the server transmits data to each client, specifying which attributes of what entities have changed since the last update; this information is then merged into the existing gamestate by the client. Dynamic entities fall into one of four main categories, some of which have a number of subcategories and other attributes.

- **Weapons**

Obviously, this category consists of the various guns scattered throughout the environment. There are a large number of distinct weapons in the game; the blaster, shotgun, super shotgun, machine gun, chaingun, hand grenades, grenade launcher, rocket launcher, hyper blaster, railgun, and BFG10K. Each of these guns has different properties - discharge rate, projectile radius and speed, damage inflicted, etc - and their usefulness varies depending on the specific situation in which the agent finds itself. Combat behaviours involving these weapons are discussed in Section 4.5.

- **Items**

The Item category encompasses all non-weapon collectible entities in the game world. These consist of “power-ups” - items which, when obtained, boost the combat abilities of the player. Four subcategories exist:

- **Ammunition:** five distinct types of ammunition are found in the game world, some of which are shared between multiple guns; bullets, cells,



shells, rockets, and grenades. Only a certain maximum quantity of each type of ammunition can be carried by the agent.

- **Armour:** flak jackets, armour shards, combat suits, and body armour packs all fall under this subcategory. These items, once collected, will reduce the amount of damage taken by the game character.
- **Health:** med kits and first aid kits, when collected, will boost the remaining health of the game character, undoing any damage it had previously taken.
- **Special items:** some maps may also contain one or more “special” items - collectibles which provide a short but significant boost to the character, or which are necessary in order to negotiate certain elements of the terrain. These include the “quad damage” pickup, which multiplies the damage the character does by a factor of 4, “invulnerability”, which briefly makes the agent immune to all damage, and the “underwater breathing” suit, which allows the agent to stay submerged without losing health due to drowning.

- **Objects**

Objects are dynamic entities which are closely associated with the fixed environmental entities discussed earlier. They are necessary in cases where environmental entities must *activate* in response to some action by the player, e.g. levers, buttons, automatic doors, lifts which only ascend when a player stands upon them, etc. Object entities should therefore be seen as “triggers” for the underlying fixed entities, rather than entities in their own right.

- **Players**

The final type of entity present in the game environment are Player entities, i.e. the player himself and the opposing game characters against whom he is fighting. Player entities are defined by a set of attributes, as follows:

- **Name:** an identifier, specified by the player upon connection to the game server. Can be any combination of alphanumeric characters.
- **Skin:** determines the in-game character model of the player, used when visualising the environment.
- **Origin:** the current XYZ location of the Player entity within the map.
- **Angles:** the current orientation of the Player entity; this determines the direction in which the player is currently looking/aiming. Yaw and pitch are the important quantities from the perspective of data analysis - the Roll value is only altered upon events such as a player's death.
- **Velocity:** determines the current trajectory and speed of the player along each of the X, Y and Z axes.

The local Player can, of course, also obtain information on various other elements of his status; he can query his inventory to find out which items he is carrying, determine which weapon he is currently wielding, track how much health/armour he has remaining, and so on. Opposing player entities as reported by the server are distinct from the local bot, in that only a reduced set of their attributes are visible from the network communication stream. Only the Name, Skin and Origin attributes of enemy players are visible; their inventories, health etc are hidden from view.

#### *3.3.1.4 Inventory*

The set of all items and weapons currently possessed by the player is called his *inventory*. Each time the player dies, his inventory is reset to its default contents. The local client does not maintain a continuous record of the player's inventory; it is retained on the server-side, and the full listing is only transmitted to the client when the user requests it. This is problematic from the perspective of data analysis, since many models will naturally require knowledge of the player's inventory at each timestep, and recorded game sessions will not contain this data. QASE, as will be

discussed in Section 3.4.2, circumvents this problem by constructing the complete inventory listing in real-time as the agent picks up and uses items.

#### 3.3.1.5 Respawning

As noted earlier, one of the attractions of first-person shooters from the perspective of imitation learning is that they represent an abstraction of the real world; and consequently, that techniques developed in-game may be adapted for a wider range of applications. However, certain phenomena exist in the game world which have no identifiable real-world counterpart. The most important of these is the concept of *respawning*, which refers to the spontaneous regeneration of dormant entities. If an item is collected by a player, it vanishes for a fixed interval; when that period elapses, the item *respawns* at its original position, ready to be collected again. Similarly, if a player is killed, he need only press a key to reappear at one of the random *player spawn points* around the arena. The purpose of the respawning mechanic is to ensure that the game world does not become starved of item resources, which would cause the session to come to a premature stalemate. While respawning is obviously not applicable to real-world scenarios, it does present a valuable additional element of human strategic thinking from the perspective of imitation learning research - a human player will intuitively track the times at which he collected various items, and if one such item is due to respawn in the near future, will adjust his pursuit of strategic goals accordingly. This topic is dealt with in greater detail in Section 4.3.7.

#### 3.3.2 QASE Network Interface

We have described how the Quake 2 environment and its constituent entities are represented; we must, however, still develop a means of communication between the environment and agent. QASE's network layer provides this *interface*.

Quake 2's multi-player mode is a standard client-server model. One player starts a server and other combatants connect to it, entering whatever environment (map) the instigating player has selected. These opponents then fight one another until a specified time or kill limit has been reached, at which point the game restarts on the same map or another, as dictated by the server. The standard Quake 2 game client also incorporates a rendering engine, which is responsible for visualising the map and the various entities within it during human play; this is, naturally, unnecessary from the perspective of artificial agents. The network protocol itself, including the format of each message exchanged between client and server in the course of a game session, has already been reverse-engineered from recorded DM2 matches by Uwe Girlich [42]. In order to both manifest *agents* (or *bots*) within the game world and to provide the *interface* component of the simulation, a means of establishing a UDP session with the Quake 2 server, handling incoming state information, and communicating the bot's actions back to the server must be implemented.

The current incarnation of QASE achieves this via a network layer which incorporates a full implementation of the Quake 2 client-side protocol. This was, however, not originally the case, as we discuss below.

### *3.3.2.1 Early Development*

QASE's network layer originated as a project at Blekinge Institute of Technology, written by Martin Fredriksson in 1999, which we discovered during the initial phase of our research. In its original form, it consisted only of a skeleton implementation of certain parts of the Quake 2 client network protocol. Unfortunately, since it had been written as an experimental teaching aid and had not been maintained, it also contained several major bugs, possessed the bare minimum of functionality - indeed, substantially *less* functionality than even the Quake 2 Bot Core discussed earlier - and was certainly not ready for deployment in research work. Nonetheless, we felt

that it would provide a decent and (as a Java-based application) extensible starting-point, while its lack of existing functionality would enable us to mould it into an API of the kind we had envisioned. We therefore contacted Mr. Fredriksson and proposed that we should assume development of the API, to which he agreed.

Given the presence of the aforementioned bugs, and the obvious requirement that agents should have a clean, stable, comprehensive interface to the game world, our first task was to rewrite the network interface from the ground up, eliminating problems and improving its efficiency, while extending it to cover the full client protocol. Some of the issues which were encountered during QASE's early development are enumerated below.

- **Incorrect gamestate handling.** As further discussed in Section 3.3.2.2 later, Quake 2 uses a system of *delta coding* to minimise the amount of information transmitted at each interval. As such, the client API needs to be capable of handling this information correctly, merging the gamestate updates with the current representation to produce a complete view of the game world. QASE did not originally do so; it was found that the API was “dropping” updates, meaning that the client was only receiving an accurate view of the game world - including its own position and direction data - once every 12 frames, when the server defaulted to baseline mode. Clearly, a 92% loss of sensory information was untenable, and needed to be rectified. To achieve this, the gamestate-handling code was rewritten so that every entity was propagated forward from the previous to the current gamestate at each interval, before the specific changes are merged in. The actual merging process was also rewritten, to prevent updates from overwriting existing entity attributes in cases where the update does not include new values. Finally, the process was made vastly more efficient by eliminating the redundant duplication of component objects from one timestep to the next.

- **Missing entities.** Calls to the gamestate requesting a list of the entities in the environment generally returned only a tiny subset of the actual entities present. Opposing players were found, while all other entities - weapons, ammunition, various collectible items - were not. This was due to QASE's incorrect handling of the *PacketEntities* message, which together with *PlayerInfo* represents the core of the gamestate information updated by the server on each timestep; as such, the bug was quite debilitating from the agents' perspective. To remedy this problem, both the *PacketEntities* class and the associated message-handling routines were rewritten, such that the entities are correctly decoded and added to the gamestate on each update.
- **Unimplemented messages.** Originally, QASE's network layer did not implement many of Quake 2's server messages at all, simply ignoring them whenever they were received. These included the *Inventory* message, which is used to pass a list of the player's current item possessions from the server to the client, the *Reconnect* message, which is used to indicate that a new map has been chosen by the server, the *TemporaryEntity* message, which represents irregular entities (projectiles, etc) when they are active in the game world, the *ServerSound* message, which provides information about auditory effects, and the *ServerDownload* message, which allows the client to download content from the server if it does not possess it locally. As a result, agents were unable to perform such basic tasks as determining which items they were carrying and in what quantities, and were incapable of successfully moving from one environment to the next when the server invoked a change of map. All messages in the Quake 2 protocol have now been implemented, allowing the bots to perform these and other such tasks without user intervention. The *Inventory* message - which represents most of the agent's internal state - was a particularly crucial omission, both in

relation to our own work and to models of Quake agent AI in general. See Section 4.3 for an example of its importance in imitation learning.

- **Thread safety.** Originally, QASE agents had to be written from scratch, using a simple polling mechanism. Under the initial implementation, this meant that the *Proxy* - running on a dedicated thread - could write to the gamestate while it was being read from elsewhere, or vice-versa. The only effort to counteract this was a series of class-scope booleans designed to prevent multiple threads from calling the accessor and mutator methods of the *Proxy* and *World* objects simultaneously. As this was a rather haphazard and inefficient approach, which was often observed to result in deadlocks and consequent game freezes, we replaced the boolean pseudosemaphores by *synchronizing* each of the relevant methods. Furthermore, the bots can now operate in a new *high thread-safety mode*, whereby the *Proxy* will synchronize on the *World* object when updating the gamestate, and the agent will do likewise when reading it; this guarantees that the agent's view of the gamestate cannot be compromised by intervening write operations. As detailed in Section 3.4.1, we also added a second approach to agent implementation using the *observer* pattern. Each of these has their respective advantages, as will be discussed later.
- **Client ID conflict.** This bug prevented multiple QASE agents from entering the same server. On connection, the client is asked to provide a unique ID number to the server; in the original QASE implementation, the same number was used by every agent, meaning that only the first connection attempt would be accepted. This was solved by maintaining a list of all previously allocated client IDs, and randomly assigning an exclusive number to each bot.

- **Incorrect byte-stream translation.** It was discovered that QASE was incorrectly converting numerical data to the byte-stream format required for network transmission. For instance, Quake 2 clients send their movement information to the server in the form of a *forward* velocity and a *right* velocity, with negative numbers indicating motion to the back or left; as a result of the bug, the agents' movements were restricted to the 90° quadrant directly in front and to their right, which was obviously not acceptable. This error also affected all other similar numerical conversions, leading in many cases to erratic and unpredictable behaviour.
- **Spontaneous crashes.** The agents were found to crash under numerous, incongruous circumstances; for instance, if any opposing player disconnected from the server while the bot was active, it would trigger a crash. This was traced to the original code's handling of the *Config* table, a global repository of data on each entity in the game world.

As is clear from the above list, these issues - and several other similar problems - needed to be addressed before the API could be considered usable, let alone stable. Beyond this, it contained no functionality, not even a framework for the creation of game agents; users were required to write bots from scratch each time, and to manually handle menial tasks such as defining the bot's profile, detecting when it has been killed, re-entering the game, etc. In Section 3.4, we describe our development of a formal agent-creation hierarchy, and numerous associated features.

First, however, we detail the operation of QASE's rewritten network interface.

### 3.3.2.2 *QASE Network / Agent Interface*

In its simplest form, QASE's network layer represents an intermediary between the server and the QASE agent's AI routines on the local machine (see Figure 3-3). All



two-way communication is handled via the *Proxy* class, which is responsible - through delegation to the *CommunicationHandler* and various *Message* classes - for establishing the session, sending the agent's movement on each timestep, and receiving and correctly processing data about the environment. On each update, all incoming information is amalgamated into a *World* object, which represents the current *gamestate* (that is, the status of every entity within the virtual world). This gamestate can then be accessed and queried by the agent's AI cycle; once done, the Proxy is instructed to transmit the desired agent actions back to the server for application on the next timestep. A simplified illustration of this process is shown in Figure 3-6 towards the end of this section.

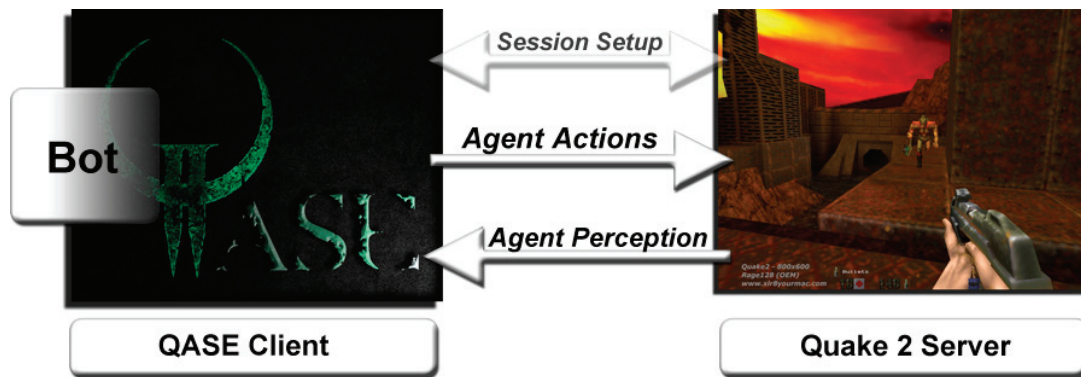


Figure 3-3 - QASE implementation of *agent* (bot) and *interface* components

Network communication between a QASE client and the Quake 2 server takes three forms: *session setup*, wherein the client engages in a series of exchanges with the server in order to gain the permissions necessary to manifest a character in the game world, *irregular messages*, which the client can send to the server at any arbitrary time in order to perform actions such as changing weapons or using an item, and *game messages*, which the client and server exchange at regular 100ms intervals, and which keep the agent apprised of changes to the environment's state. Below, we detail the most important concepts in each of these three categories.

### 3.3.2.3 Session Setup

Each game session is established using *connectionless packets*, a special type of datagram which can be sent to the server in the absence of a proper connection. These packets consist simply of a header containing the sequence number -1, followed by a text string indicating the desired command. The session setup procedure is as follows:

- The client issues a message containing the string **getchallenge**
- The server replies with a message containing the string

- challenge 12345

where 12345 is a code used to ensure that the client is not using a spoofed IP

- The client responds with

- connect 34 789 12345 <userdata>

where 34 is the network protocol revision, 789 is a unique client ID, 12345 is the code sent by the server earlier, and <userdata> is a text string containing information about the player's desired character model, name, etc.

- The client then receives a *ServerData* message providing global information about the game world and entities, as well as a series of *SpawnBaseline* messages which provide details of the various entities' initial (baseline) attributes.
- Finally, the client receives a *StuffText* message indicating that it is ready to enter the game world. One last connectionless packet is sent to the server:

- o begin <levelkey>

where `levelkey` is an identifier for the current map, received in `ServerData`. Once all this is complete, the character is *spawned* into the world and can now interact with the other entities using the in-game protocol described below.

#### 3.3.2.4 Irregular Messages

Connectionless packets are also used to send *irregular* queries and commands to the server, as distinct from the standard movement and aiming information transmitted on every frame update. These include requests for a listing of the client's inventory, to talk to other players, to use inventory items, or to switch weapons. The QASE bot hierarchy (Section 3.4.1) provides convenience methods for all these features, hiding the actual message construction from the user; direct calls to the `sendCommand` methods of the Proxy class are also facilitated for added low-level control.

#### 3.3.2.5 In-Game Protocol

Once the session has been established and the agent has entered the game, the server and QASE client begin exchanging messages. Every tenth of a second, the server sends updates to each connected client; unless exceptional circumstances are encountered, these updates consist of two sequence numbers and three *messages*:

- ***ServerFrame***, which specifies the current and delta reference frames
- ***PlayerInfo***, which contains information about the client's current position, velocity, aim, weapon, score, etc.
- ***PacketEntities***, which contains information about all game entities.

Upon receiving a frame, the client must *merge* the updated information into its existing gamestate record (see below). This done, QASE issues a *ClientMove*

message to the server, indicating its desired velocity ( $forward, right, up \in [-400, 400]$ ), aim ( $yaw \in [-180, 180], pitch \in [-90, 90]$ ), and whether or not it is firing its gun. These values together define the scope of the agent's legal in-game movement.

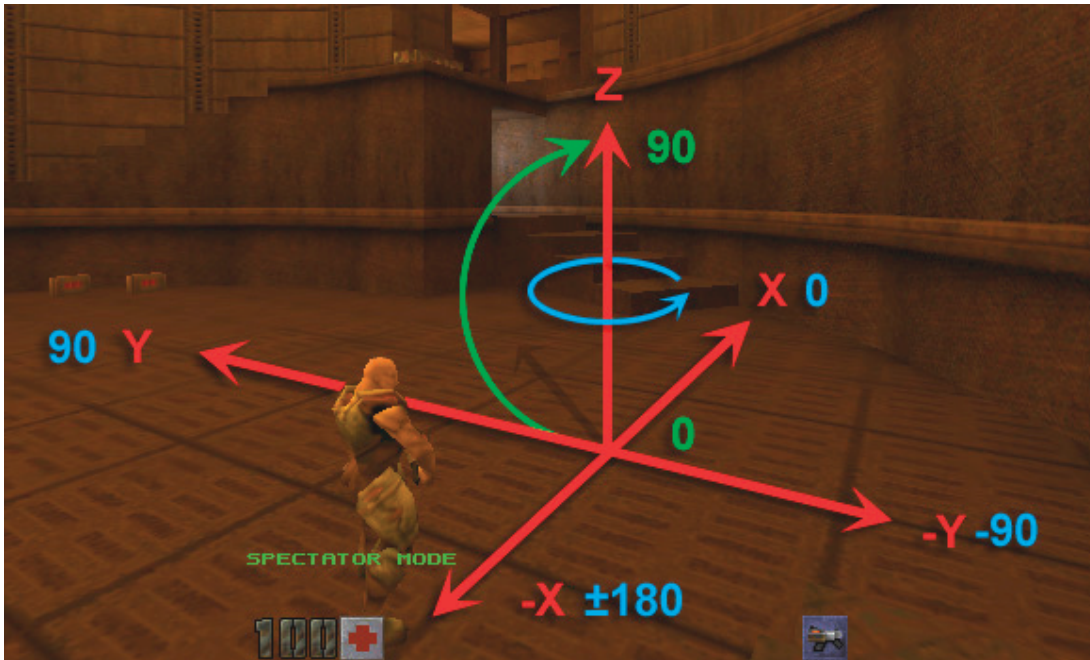


Figure 3-4 - Server's view of the in-game character's range of actions

An important point to note is that the client and server interpret these values differently. From the server's perspective, the *forward* velocity corresponds to *velocity along the global x-axis*, *right velocity* is *velocity along the global y-axis*, and *angular* measurements are *absolute*. From the client's perspective, the forward velocity is *velocity in the direction towards which the agent is currently facing*, the right velocity is *perpendicular* to this, and the angular measurements are *relative to the agent's local axes*.

Figure 3-4 shows the server's perspective of the character, Figure 3-5 shows the client's. The significance of this is that recorded DM2 demo files use the *server* format to represent the character's movement, and so the API must be capable of moving back and forth between *local* bot co-ordinates and *global* server co-ordinates.

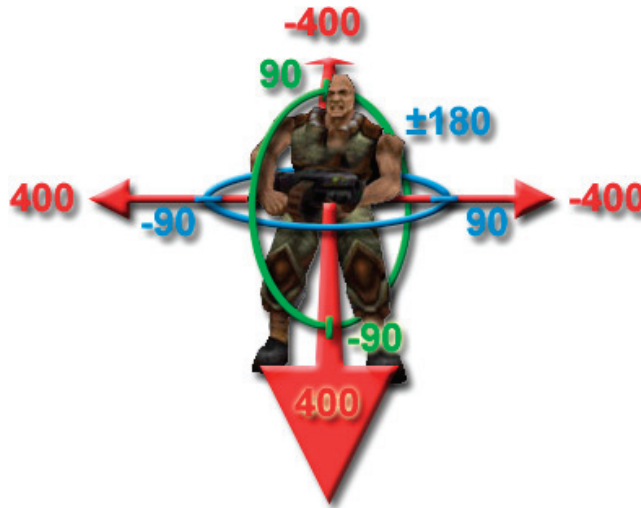


Figure 3-5 - Client's view of the in-game character's range of actions

Since Quake is a real-time game in which potentially several dozen participants need to be updated every hundred milliseconds, it is important that network latency be minimised. As such, the UDP protocol is used for communication in preference to TCP; this means that packets may arrive out of order, or be lost altogether. This approach is not as reckless as it may first appear - the loss of a player update message (for instance) can be compensated in subsequent updates, whereas using TCP would cause any lost packets to hold up the data stream while the client waits for resent information which is, by that point, out of date. Given a suitably low-overhead mechanism for acknowledging and ordering incoming packets, the integrity of the gamestate can be ensured while simultaneously maximising the available bandwidth.

Quake 2 implements such a scheme by means of *sequence numbers* and *delta coding*. Both the QASE client and the server store old frame updates to allow for packet loss; the server stores the previous 12 frames, while the client stores the previous 16. To reduce bandwidth consumption, the server only transmits information on entities within a certain viewable distance of the client, and will only send the specific entity attributes which have changed since the last frame received by the client. Each update packet sent from client to server has two sequence numbers as its header; the

first specifies the number of the current frame, while the second specifies the number of the last frame received from the server. Using this information, the server can determine which packets were lost, and what cumulative changes have occurred in the game world since that point. On the next update, the server will transmit those changes, along with the reference frame against which the client should *delta code*; QASE then *merges* the updates into its stored copy of the reference frame to produce a complete gamestate representation. If the number of lost packets exceeds the number of stored frames, then the server enters a special mode where it transmits entity updates based on the *baseline* (initial) values established during the client's connection. It will continue to do so until the client indicates that it has re-synchronized itself and is ready to resume delta coding.

Figure 3-6 shows a highly simplified outline of the relationship between the QASE *network interface* and the Quake 2 server *environment*. Some elements of QASE's *agent architecture* (Section 3.4) are also shown for reference, most notably the *BasicBot* class, which represents any agent derived from one of the base classes in the bot hierarchy (Section 3.4.1). See the caption which accompanies the diagram for further details of the interface's operation.

Finally, an interesting point to note is that, because the network layer is largely decoupled from the higher-level classes in the QASE architecture, it is highly *portable*. Adapting the QASE API to games with similar network protocols, such as Quake 3 and its derivatives, therefore becomes a relatively straightforward exercise; by extending the existing classes and rewriting the data-handling routines, they could conceivably be adapted to any UDP-based network game. Thus, QASE's network structures can be seen as providing a *template* for the development of artificial game clients in general. See Section 6.3 for further discussion.



## 3.4 QASE Agent Architecture

In this section, we describe QASE's final simulation component; the *agent architecture*. This consists both of a formal agent factory, with pre-built base agents which require the programmer to provide nothing more than a single-point-of-insertion AI cycle, and structures which provide the faculties necessary to construct said cycle. The latter includes the ability to record and parse demonstration of human gameplay, the facilitation of high-level queries about the agent's condition by transparently amalgamating low-level data, integration with the MatLab programming environment, inbuilt AI constructs, and topology-learning / navigation abilities drawn from our work in imitation learning.

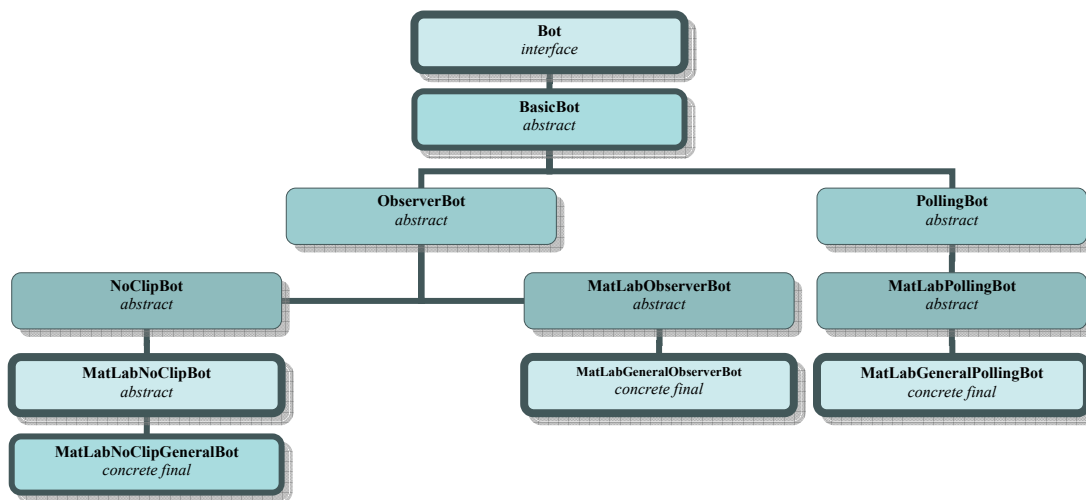


Figure 3-7 - The QASE Bot Hierarchy

### 3.4.1 Bot Hierarchy

The interface methods described earlier operate at too low a level to be practical for general use; they do not provide a structured, formal mechanism for the creation of game agents. Indeed, requiring that agents be constructed using only basic network functions - a laborious, inefficient and inherently ad-hoc process - runs contrary to



the core design principles of our proposed API; the network layer alone is, therefore, clearly inadequate. To address this, we develop a comprehensive *bot hierarchy* - an agent-factory framework allowing users to build bots from any of a number of levels of abstraction, while automating menial tasks such as defining the bot's profile, respawning when it has been killed, managing the detection of gamestate updates from the Proxy, and so forth. The framework ranges from a simple interface class to full-fledged bots incorporating an exhaustive range of user-accessible functions. The bot hierarchy comprises three major levels; these are summarised below.

#### *3.4.1.1 Bot Interface*

A template which specifies the interface to which all bots must conform, but does not provide any functionality. The programmer is entirely responsible for the actual implementation of the agent, and may do so in any way he chooses.

#### *3.4.1.2 BasicBot*

An abstract agent which provides most of the basic functionality required by Quake 2 agents, such as the ability to determine whether the bot has died, to *respawn* (re-enter the game after the agent has been defeated), to create an agent given minimal profile information, to set the agent's movement direction, speed and aim and send these to the server with a single method call, to obtain sensory information about the virtual world, and to record itself to a demo file. All that is required of the programmer is to write the AI routine in the predefined *runAI* method, and to supply a means of detecting and handling server updates according to whatever paradigm he wishes to use. The third level of the hierarchy (see below) provides ready-to-use implementations of two such paradigms.

*BasicBot* provides embedded access to the *BSPParser* class for environment sensing (see Section 3.4.5), and the *WaypointMap* class for navigation (see Section

3.4.8), by incorporating pass-through methods which relay calls to the appropriate object. Some of these pass-through methods automatically set parameters to the most useful default values; for instance, the bounding-box used to perform collision detection is set to the size of the agent's in-game character. *BasicBot* will transparently find, load and query the BSP file associated with the current game level when one of the environment-sensing methods is invoked for the first time. Users can also obtain a pointer to the underlying objects, allowing full access to their facilities.

#### *3.4.1.3 ObserverBot & PollingBot*

The highest level of the Bot hierarchy consists of two classes, *ObserverBot* and *PollingBot*, which represent fully-realised agents. Each of these implements a method of detecting changes to the gamestate as indicated by their names, as well as a single point of insertion - the programmer need only supply the AI routine to be invoked on each update in the *runAI* method, while all other communication and data-manipulation requirements are handled by the API. Thus, the agent is notified of each update as it occurs, and a copy of the gamestate is presented to it. The user-defined AI routines then set the required movement, aiming and action values for the next update, and the API automatically transmits the changes.

The *ObserverBot* uses the *observer* pattern to register its interest with a *Proxy*, and is thereafter notified whenever a game update takes place. Since this approach is single-threaded, a separate thread is created to check whether the bot has been killed, and to respawn as necessary. The advantages of this approach are twofold:

- it guarantees consistency of the gamestate; since the *Proxy* thread invokes a method call in the *ObserverBot*, it must wait until the agent's AI routine is complete before receiving any further updates. This means that the gamestate cannot be written in mid-AI cycle.

- it allows multiple *observers* to connect to a single *Proxy*. This can be useful if the programmer wishes, for instance, to have a second observer perform some operation on the incoming data in the background.

The *PollingBot* operates by continually polling the *Proxy* and obtaining a copy of the gamestate *World* object. If a change in the current frame number is detected, the agent knows that an update has occurred, and will enter its AI routine. Because the *Proxy* and agent are operating on separate threads, the *Proxy* is free to receive updates regardless of what the agent is currently doing; this ensures that no server frames will be lost, but may result in changes to the gamestate while the agent is executing its AI cycle. To prevent this, the bot can be set to *high thread safety mode*, in which the agent and *Proxy* both synchronize on the gamestate object; the agent therefore cannot read the gamestate while it is being written, and the *Proxy* cannot write the gamestate while it is being read.

#### 3.4.1.4 Miscellaneous Bots

Beyond this, several convenience classes are available, which provide extended bot implementations tailored to specific purposes. The *NoClipBots* allow the user to move the agent through otherwise solid walls to any point in the environment before starting the simulation, which we have found to be extremely useful in the course of our own research; indeed, this bot category was added specifically to address our need for such functionality. The *MatLabBot* branches facilitate integration with the MatLab programming environment, and will be explained later.

### 3.4.2 Gamestate Augmentation

Rather than simply providing a bare-bones implementation of the client-side protocol, QASE also performs several behind-the-scenes operations upon receipt of

each update, designed to present an *augmented* view of the gamestate to the agent. In other words, QASE transparently analyses the information it receives, makes deductions based on what it finds, and exposes the results to the agent; as such, it may be seen as representing a *virtual extension* of the standard Quake 2 network protocol.

For instance, the standard protocol has no explicit *item pickup* notification; when the agent collects an object, the server takes note of it but does not send a confirmation message to the client, since under normal circumstances the human player will be able to identify the item visually. QASE compensates for this by detecting the sound of an item pickup, examining which entities have just become inactive, finding the closest such entity to the player, and thereby deducing the entity number, type and inventory index of the newly-acquired item. Building on this, QASE records a list of which items the player has collected and when they are due to *respawn*, automatically flagging the agent whenever such an event occurs. QASE detects and flags a number of other implicit events in similar fashion; it alerts the agent when an enemy dies, determines how long it can remain underwater before drowning, and so forth.

Similarly, recordings of Quake 2 matches do not encode the full inventory of the player at each timestep - that is, the list of how many of which items the player is currently carrying. For research models which require knowledge of the inventory, as is the case in our own work, this is clearly untenable. QASE circumvents the problem by monitoring item pickups and weapon discharges, ‘manually’ building up an inventory representation from each frame to the next. This can also be used to track the agent’s inventory in online game sessions, removing the need to explicitly request a full inventory listing from the server on each update.

### 3.4.3 Team-Based Play

QASE is fully compatible with the popular Threewave CTF modification for Quake 2, in which players join either a Red or Blue team and attempt to capture the enemy faction's flag, by collecting it and returning it to their own base. Methods are provided which enable the agent to join a specific team, or to join randomly; further methods allow the agent to determine whether a particular player is a member of its own or the opposing force. In cases where the server type is not known in advance, the API will automatically determine the game mode, and if necessary will join an arbitrary team. QASE is, therefore, well suited to researchers whose interest lies in investigating team-based behaviours and interactions.

### 3.4.4 DM2 Parser & Recorder

One of the most fundamental requirements of the API from the perspective of our own research was that it should allow *demo* (DM2) files to be parsed; these are saved recordings of the network packet stream received during a game session, which thereby encode a full account of the human player's observed movements and actions. QASE facilitates this via its *DM2Parser* class. Demo files are organised into *blocks*, each of which consists of a header indicating the length of the block followed by a series of concatenated *messages*, as described in Section 3.3.2.5. The DM2 parser operates by treating the file as a *virtual server*, sequentially reading blocks and updating the gamestate as if it were receiving data online.

Furthermore, QASE incorporates a DM2 *recorder* which enables each agent to save a demo of itself during play; this actually improves upon Quake 2's standard recording facilities, by allowing demos spanning multiple maps to be recorded in playable format. This is done by separating the header information (*ServerData* and *SpawnBaseline*) received when entering each new level from the stream of

standard packets received during the course of the game. The incoming network stream is sampled, edited as necessary, and saved to file when the agent disconnects from the server or as an intermediate step whenever the map is changed.

### 3.4.5 Environment Sensing

The network packets received from the Quake 2 server do not encode any information about the actual environment in which the agent finds itself, beyond its current state and that of the various game entities. This information is contained in files stored locally on each client machine; thus, in order to provide the bot with more detailed sensory information (such as determining its proximity to an obstacle or whether an enemy is currently visible), a means of locating, parsing and querying these map files is required.

As discussed earlier, each Quake 2 map is constructed from a number of *brushes*, a generic term for any convex polyhedron such as a pyramid, cuboid, cylinder, etc; the term “convex” here means that any line through the object will have precisely one entry point and one exit point, the importance of which will be seen later. The planes formed by the faces of these objects divide the environment into a set of *convex regions*, or the areas of open space within (some of) which the actual gameplay takes place. Because Quake 2 levels are often very large and encompass a vast number of brushes, an efficient means of representing them and the convex regions they enclose is required; for this purpose, a *Binary Space Partition Tree* is used.

#### 3.4.5.1 *Binary Space Partitioning*

The concept of a *Binary Space Partition Tree*, devised by Fuchs, Kedem, and Naylor [38], provides an economical way of representing a virtual scene by specifying which other polygons lie *in front of* and *behind* the plane formed at the surface of a given

polygon. The resulting tree structure can then be used to perform collision detection or to determine the order in which the polygons should be rendered, given the position of the viewer. A BSP tree is created by first selecting a polygon to act as the *root node*, with an associated *splitting plane*; all remaining polygons are then classified as being in front of or behind this plane, and are added as its left and right children of the root node, respectively. In cases where a polygon intersects the plane, that polygon is divided in two at the point of intersection and both are added to the binary tree. The process is recursively applied to each of the child lists, gradually creating a series of subtrees until the list is exhausted. An example of BSP tree construction is shown below.

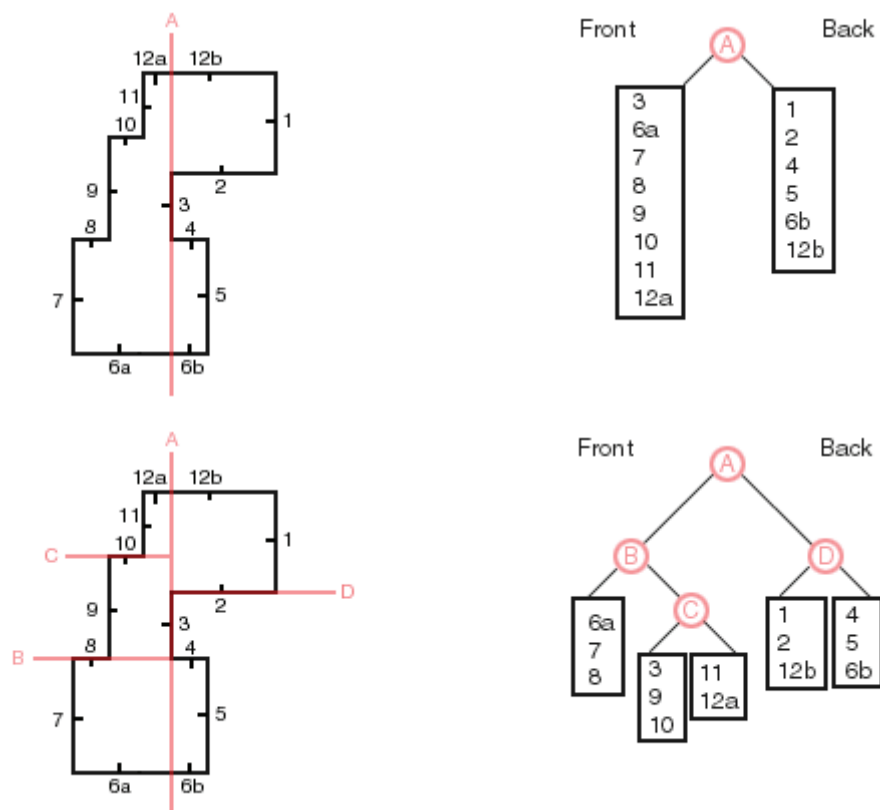


Figure 3-8 - Initial and final stages in the construction of a 2D BSP tree. Image from 3dtechdev.com.

### 3.4.5.2 Quake 2 BSP Format

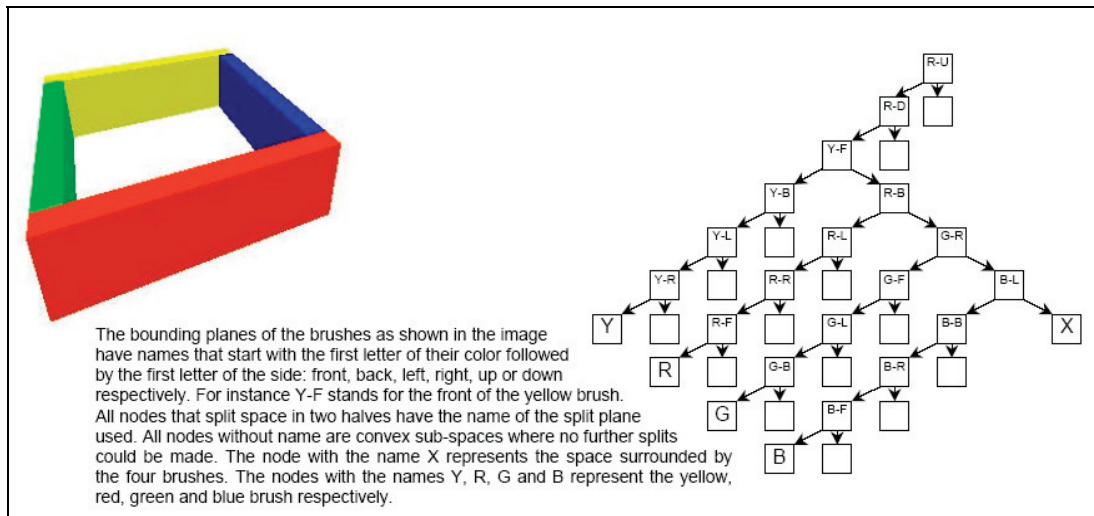
Quake 2 BSP files consist of a *directory* indexing a series of *lumps*, each of which contains information about a different element of the environment (entities, textures, models, etc). From the perspective of obtaining sensory input about the game world, the most important of these are:

- the **Node** and **Leaf** lumps, each of which represents a single element in the BSP tree. Each *Node* contains an index to the plane equation used to split the tree at that point; this is stored as a *normal* with an associated *distance*. The *Leaf* object, representing a convex region within the level (including the spaces *within* objects), indexes each brush which bounds the empty space.
- the **LeafBrush** lump, which indexes the brushes surrounding each hull.
- the **Brush** lump, which contains information about each brush in the level. This includes the number of sides the brush has and its *content*, that is, whether it is a solid object (as a wall) or may be passed through (water).
- the **BrushSide** lump, which indexes each brush side's associated *plane*
- the **Plane** lump, which provides the coefficients of each plane equation; these will later be used to determine the side of the associated face on which the player's avatar is currently located.

QASE's **BSPParser** class is used to load the contents of a Quake 2 BSP file, and to build the tree it represents. It also provides methods to perform *collision detection*, by sweeping a line, box or sphere through the environment between two specified points and reporting the first instance of a collision with a solid object; as an extension of this, *visibility detection* may be performed by checking whether there are any opaque



surfaces between the player's position and a given point. For convenience, the *BasicBot* class in the agent hierarchy provides embedded access to *BSPParser* - calls to various environment-sensing functions are passed through to the appropriate *BSPParser* methods. The actual BSP file corresponding to the active map in a given game session may be stored in the default game directory, a custom directory, or in any of Quake 2's PAK archives; its filename may or may not match the name of the map, which is the only information possessed by the client. If the user sets an environment variable pointing to the location of the base Quake 2 folder, *BasicBot* will automatically find the BSP file by searching each location in order of likelihood. This is done transparently from the agent's perspective; as soon as any environment-sensing method is invoked, the map is silently located, loaded and queried.



**Figure 3-9 - A 3D Quake BSP tree from van Waveren [135]. Each leaf is an area - possibly within one of the brushes - bounded by a set of planes, and is known as a convex region or hull. Note that RU and RD nodes subsume the splitting planes for YU, YD, GU, GD, etc, as these are *coplanar*.**

### 3.4.5.3 Collision Detection

Collision detection is performed by tracing the course of a line through the BSP tree and determining the first solid brush with which the line makes contact. The

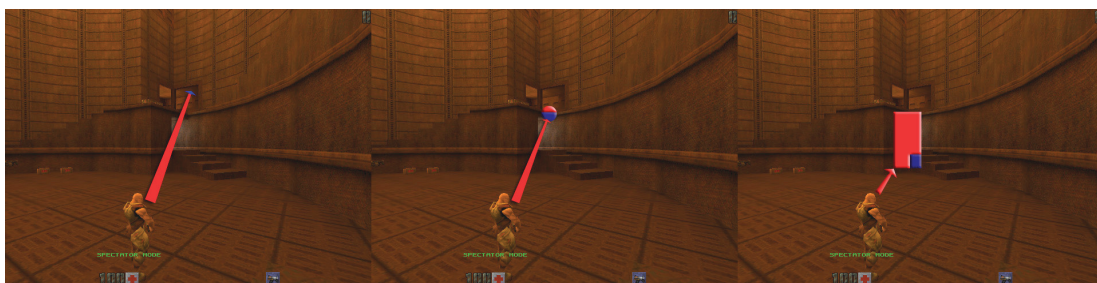
algorithm used to implement this is a pruned depth-first search; given a *start* and *end* point, and letting the first node be the root node, it proceeds as follows:

1. At the current node, compute the dot product of the start and end points with the splitting plane's normal, and subtract the plane's distance from this. If the result is positive, then the point is in front of the plane; if it is negative, it is behind the plane. Points that lie *on* the plane are treated as being in front of it. This gives two possible scenarios:
  - a. **both points are on the same side of the plane.** In this case, we need only traverse the front or back child of the current node.
  - b. **the line between the two points is intersected by the plane.** In this case, we must split the line into two segments at the point of intersection, and continue to trace the relevant segment down the front and back children of the current node. Generally, we decide which to check first by choosing either the side closest to the start of the trace, or the one on which most of the divided line now lies.
2. If the newly-chosen node is a leaf, then proceed to step 3. Otherwise, recursively perform step 1 with the new node and (possibly) line segment.
3. Upon reaching a leaf, we know that the current trace segment resides within one of the convex subspaces of the map, possibly *inside* a particular brush; it now remains to check whether a collision has occurred. Each *Leaf* node is, as mentioned earlier, associated with a (possibly empty) set of *Brushes* which bound it. Each brush is further associated with a number of *BrushSides*, representing the different faces of the brush, and each of these brush sides itself indexes one of the splitting plane equations. By computing the dot product of the full line's start and end points with these planes for each brush side, it can be determined whether the line crosses one

of the brush's polygons, and how far this collision lies from the start of the trace. If this distance is less than the current minimum collision distance found, it is adopted as the new collision distance.

4. Return back up the tree to the point at which the last branching decision was made, and continue with step 1 until all leaf nodes at which a collision may have occurred have been visited. The final result of the algorithm is the shortest distance between the start point and a collision with a solid subspace.

The above algorithm outlines the procedure for tracing a *line* through the game world; however, this may produce misleading feedback if the space through which the line passes is not large enough to accommodate the player, such as a window. To account for this, QASE also provides the ability to perform collision detection using a *bounding box* or *sphere* of customisable size. The process is much the same, but collision detection is extended in each dimension to the outer extents of the volume. Thus, line-tracing can be used for visibility checks, while the bounding box approach determines the maximum distance which the player can travel in a given direction. The sphere-tracing function can be used to determine whether a projectile will successfully reach a given point, or will strike an intervening surface.



**Figure 3-10 - BSP traces with line, sphere and box. Collision occurs at different points.**

The environment-sensing faculties of QASE illustrate another advantage of using computer games in research; they represent *idealised robot navigation worlds*, as discussed earlier. A great deal of work in this field is centred upon the elimination of

*noise* in the robot's environment [17, 72]; by contrast, computer games guarantee accurate, undistorted sensory information. Furthermore, map editors for such games are common, allowing the user to create any environment he desires. By using Quake or other appropriate games as testbeds, researchers can easily prototype their navigation techniques, before adapting them to the constraints of the real world.

### 3.4.6 Fixed Entity Sensing

Aside from pure geometric data, the BSP files also contain information about certain active features within the game environment. These entities, which include doors, lifts, teleporters and buttons, should not be confused with the entity information received from the server on each update, which relates primarily to player movements and weapon spawns / despawns. The QASE API transparently parses and extracts the details of all such entities upon the first BSP query, and performs additional processing in order to allow the resulting data to be queried from high-level contexts. For instance, graph-style edge links are created between teleporters and their destination portals, while methods within the BasicBot class can be used to easily determine whether the player is currently standing on a moving platform. QASE can also return a list of the locations in the game environment at which certain potentially hazardous features - lava, poison, etc - are found.

### 3.4.7 Inbuilt AI Constructs

For education purposes, QASE incorporates implementations of both a *neural network* and a *genetic algorithm generator*. These are designed to be used in tandem - that is, the genetic algorithm controller gradually causes the neural network's weights to evolve towards a given fitness function. The main classes involved in this process are found in the `soc.qase.ai.gann` package:

- ***NeuralNet***

Builds a *neural network* given design parameters, controls the retrieval and updating of its weights, facilitates output using *logsig* or *tansig* functions, and computes the net's output for given input. Also allows the network to be saved to disk and loaded at a later time.

- ***Genetic***

A *genetic algorithm* (GA) generator class, which maintains the gene pool, records fitness stats, controls mutation and recombination, and generates each successive generation when prompted. The class also provides methods to save and load Genetic objects, thereby allowing the genetic algorithm process to be resumed rather than restarted.

- ***GANNManager***

Provides the basic template of a 'bridge' between the GA and NN classes, and demonstrates the steps required to evolve the weights of a population of networks by treating each weight as a nucleotide in the GA's genome. The class provides two modes of operation. For offline experiments - that is, those performed outside a live Quake 2 match - the GANNManager can be run as a thread, continually assessing the fitness of each network according to a user-defined function, recombining the associated genomes, and evolving towards an optimal solution for a specified duration of each generation and of overall simulation time. For online experiments, the class can be attached as an Observer of one or more Proxy objects, providing direct feedback from the Quake 2 game world. The class is abstract; it must be subclassed to provide the necessary fitness and observer functions, and to tailor its operation to the specific problem at hand. The class also allows the user to save an entire simulation to disk, and resume it from the same point later.

A *k-means calculator* package (`soc.qase.ai.kmeans`) is also included, to serve as an illustration of clustering techniques; it is also used extensively in QASE's *waypoint map generator*, to derive the graph nodes from raw observation data. The k-means facilities are composed of two classes, *KMeansCalc* and *KMeansData*. The latter acts as a wrapper for the data output by the algorithm, whereas the former performs the actual clustering according to the following formulae:

$$b_i^s = 1 \text{ if } \|x^s - c_i\| = \min_j \|x^s - c_j\|, \text{ and } 0 \text{ otherwise}$$

$$c_i = \frac{\sum_s b_i^s x^s}{\sum_s b_i^s} \text{ while } c_{i-1} \neq c_i, i = 1 \dots k$$

where  $k$  is the number of clusters,  $x$  is a point in the dataset, and  $c$  is a cluster centroid

These features are intended primarily to allow students to experiment with some AI constructs commonly taught in undergraduate classes - for more demanding research applications, QASE allows MatLab to be used as a back-end (see Section 3.4.9).

### 3.4.8 Waypoint Maps

One of QASE's most useful features, particularly from an educational point of view, is the *waypoint map generator*. The most important requirement of any agent is that it be capable of negotiating its environment. Although this can be done using the environment-sensing facilities outlined above, to rely exclusively upon BSP tracing would be a cumbersome and computationally expensive solution. Most traditional methods of navigation instead employ *waypoint maps* - topological graphs of the level, indicating the paths along which the agent can move. With this in mind, QASE provides a package, `soc.ai.waypoint`, specifically designed to facilitate the rapid construction of such maps.

While the two principal classes of this package, *Waypoint* and *WaypointMap*, can be used to manually build a topology graph from scratch, QASE also offers a far more elegant and efficient approach to the problem - the *WaypointMapGenerator*. A prime example of how our research in imitation learning has guided the development of the QASE API, *WaypointMapGenerator* draws on concepts developed in the course of our work in strategic navigation imitation (see Section 4.3). Users need only present the *WaypointMapGenerator* with a pre-recorded DM2 file; it will then automatically find the set of all positions occupied by the player during the game session, cluster them using the inbuilt k-means classes to produce a smaller number of indicative *waypoints*, and draw *edges* between these waypoints based on the observed movement of the demonstrator. The items collected by the player are also recorded, and the Floyd-Warshall algorithm [36, 137] is then applied to find the matrices of distances and shortest paths between each pair of points. Given an  $N \times N$  edge matrix  $E$  specifying the adjacency between the  $N$  waypoints, this algorithm proceeds as follows:

```

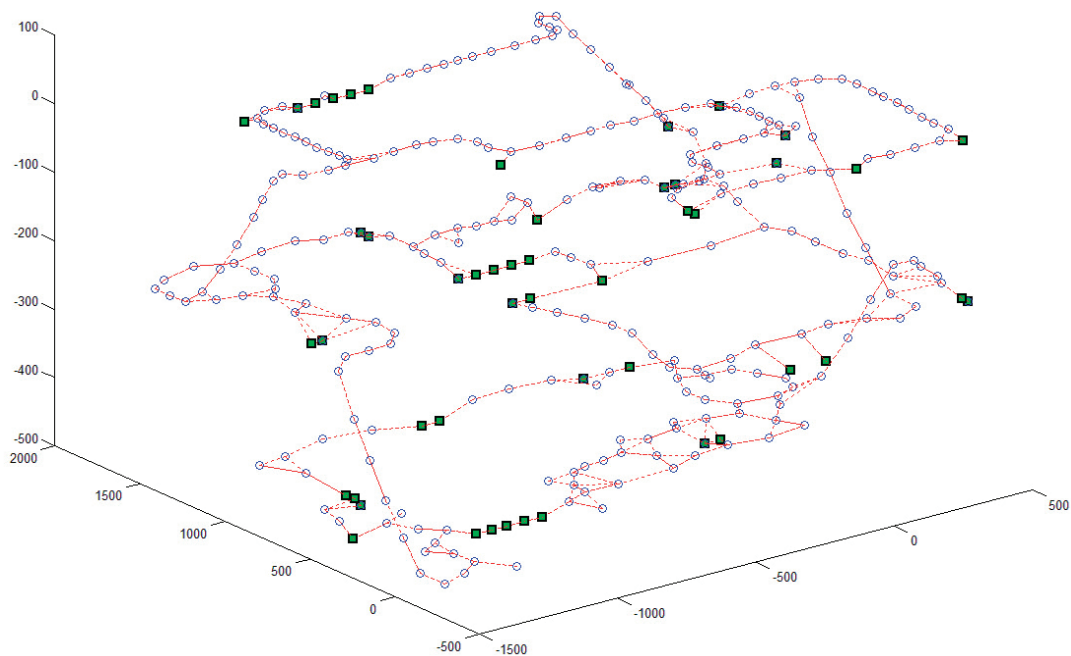
pathMatrix(i,j)={NIL if (i,j)  $\notin$  E | i=j, i if (i,j)  $\in$  E & i!=j}
cost(i,j)={0 if i=j,  $\infty$  if i!=j & (i,j)  $\notin$  E, dist(i,j) if i!=j & (i,j)  $\in$  E}

for b = 1 : N
  for a = 1 : N
    for c = 1 : N
      if cost(a,b)  $\neq$   $\infty$  & cost(b,c)  $\neq$   $\infty$ 
        if cost(a,b) + cost(b,c) < cost(a,c)
          cost(a,c) = cost(a,b) + cost(b,c)
          pathMatrix(a,c) = pathMatrix(b,c)

```

**Figure 3-11 - The Floyd-Warshall algorithm**

The shortest distance between two points can be determined directly from the final *cost* matrix, while the shortest path is found by “backtracking” through the *path* matrix from the end node to the start. The map returned to the user at the end of the process can thus be queried to find the shortest path from the agent’s current position to any needed item, to the nearest opponent, or to any random point in the level. Rather than manually building a waypoint map from scratch, then, all the student need do in order to create a full navigation system for their agent is to record themselves moving around the environment as necessary, collect whatever items their bots require, and present the resulting demo file to QASE.



**Figure 3-12 - A waypoint map, generated automatically by QASE from recorded human data and exported to MatLab for visualisation. Green squares indicate item pickups.**

The waypoint map functionality is again embedded into the *BasicBot* class - that is, it provides shortest-path methods which the agent transparently passes on to an underlying *WaypointMap* object. The ability to retrieve the map as raw positional and edge data is also provided; this is particularly convenient for reading the map



into MatLab, as shown in Figure 3-12. Additionally, *WaypointMap* permits instances of itself to be saved to disk and reloaded, thereby enabling users to generate a map once and use it in all subsequent sessions rather than recreating it each time.

### 3.4.9 MatLab Integration

For the purposes of our work in imitation learning, we need a way to not only obtain, but also statistically analyse the observed in-game actions of human players. Rather than hand-coding the required structures from scratch, we opted instead to integrate the API with the Mathworks MatLab programming environment [83]. Given that it provides a rich set of built-in toolboxes for neural computation, clustering, classification and countless other techniques - in addition to its status as one of the most widely-used research platforms available - MatLab was an ideal choice to act as an optional back-end for QASE agents.

In order to use MatLab's functions to create Quake bots, a means of linking QASE to MatLab was required. Initial experiments involving a JNI interface to the MatLab DLL engine proved to be extremely sluggish, and certainly not feasible for the demands of an agent in a real-time game. A different approach was needed; one which would allow MatLab to quickly and efficiently obtain the specific gamestate information necessary to carry out its computations, thereby optimizing the agent's performance. QASE provides two different mechanisms whereby agents can be instantiated and controlled via MatLab.

For simple AI routines, one of the standalone *MatLabGeneralBots* shown in Figure 3-7 is sufficient. A MatLab function is written which creates an instance of the agent, connects it to the server, and accesses the gamestate at each update, all entirely within the MatLab environment. The advantage of this approach is that it is intuitive and very straightforward; a template of the MatLab script is provided with

the QASE API. In cases where a large amount of gamestate and data processing must be carried out on each frame, however, handling it exclusively through MatLab can prove quite inefficient.

For this reason, we developed an alternative paradigm designed to offer greater performance. As outlined earlier in Section 3.4.1, QASE agents are usually created by extending either the *ObserverBot* or *PollingBot* classes, and overloading the *runAI* method in order to add the required behaviour. In other words, the agent's AI routines are *atomic*, and encapsulated entirely within the derived class. Thus, in order to facilitate MatLab, a new branch of agents - the *MatLabBots* - was created; each of these possesses a three-step AI routine as follows:

1. On each server update, QASE first ***pre-processes*** the data required for the task at hand; it then flags MatLab to take over control of the AI cycle.
2. The MatLab function obtains the agent's input data, processes it using its own internal structures, passes the results back to the agent, and signals that the agent should reassume control.
3. This done, the bot applies MatLab's output in a ***postprocessing*** step.

This framework is already built into QASE's *MatLabBots*; the programmer need only extend *MatLabObserver / Polling / NoClipBot* to define the handling of data in the preprocessing and postprocessing steps, and change the accompanying MatLab script as necessary. By separating the agent's ***body*** (QASE) from its ***brain*** (MatLab) in this manner, we ensure that both are modular and reusable, and that cross-environment communications are minimised. The preprocessing step filters the gamestate, presenting only the minimal required information to MatLab; QASE thus enables both MatLab and Java to process as much data as possible in their respective native environments. This framework has proven very successful, both in terms of

computational efficiency and ease of development; its advantage lies in the fact that custom bots can be written for each task, minimising the amount of work that both QASE and MatLab need to perform on each update. In contrast, the *MatLabGeneralBots* do not pass any information to MatLab or expect any results - the bot's entire AI cycle must be implemented from within MatLab.

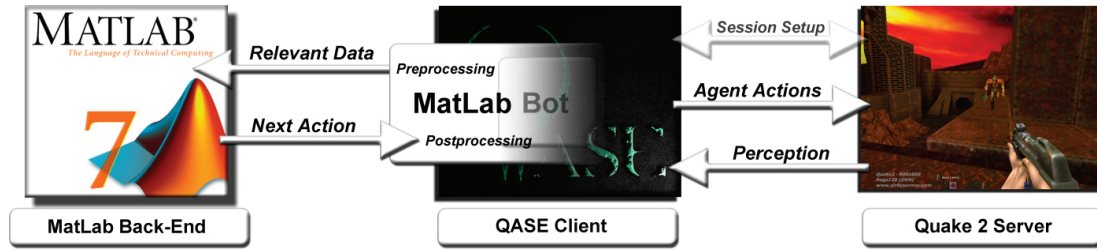


Figure 3-13 - MatLab - QASE integration. MatLab acts as a back-end in the agent's AI cycle.

### 3.5 Conclusion

In this chapter, we proposed our QASE API as a comprehensive tool for developing first-person agents in Quake. We outlined our motivation in undertaking this work, providing illustrative comparisons between the functionality of QASE and that of existing APIs. We detailed the most relevant concepts in the Quake 2 *environment*, and how they relate both to QASE and the work described later in this thesis. We described the network layer of QASE, and how it provides an *interface* between the agent and its environment, noting that its modular design provides a generic framework for the creation of agents in other games. We then detailed QASE's *agent architecture*, including its *bot hierarchy* which allows agents to be built from various levels of abstraction, its ability to supply the agent with sensory data about its environment, its facilitation of high-level queries about the agent's condition by transparently amalgamating low-level data, its ability to record and parse demonstration of human gameplay, its integration with the MatLab programming environment, its inbuilt AI constructs, and its topology-learning / navigation faculties drawn from our work in imitation learning.

### 3.5.1 Adoption in Academia

Since its release on the Sourceforge software development site, QASE has attracted attention from several quarters. From regular correspondence, we know that it is currently used in work being conducted at Bielefeld University, Germany; it has been adopted by researchers at China's Huazhong University, and at Deutsche Telekom; it is used at both under- and post-graduate level at Blekinge Institute in Sweden, where it has also been employed as the experimental platform for at least one Master's Thesis [22]; and it is used as both a teaching aid and research tool at the University of Las Palmas de Gran Canaria. Agents developed by other researchers using QASE have also been demonstrated at DreamHack 2008, the largest LAN gaming convention in the world [30]. It is our hope that QASE's continued adoption will help to foster interest in game-based machine learning and general AI research/education, by providing an intuitive, integrated development framework.

## 4 Imitation Learning

### 4.1 Introduction

In this chapter, we describe the models and mechanisms which we have developed to imitate human gameplay from recorded Quake 2 sessions. Section 4.2 introduces a hierarchy - or more accurately, a *gradient* - of observable in-game behaviours, based on Hollnagel's contextual control model of real-world human planning; this identifies *strategic* long-term planning behaviours, *tactical* mid-term planning behaviours, and *reactive* short-term stimulus-response behaviours. We further identify, based on previous work by Laird/Duchi and Livingstone/McGlinchey, a number of core criteria for agent believability. Building upon this theoretical foundation, we proceed to outline imitative models for each major level along the gradient, noting instances wherein some overlap exists between the different layers.

In Section 4.3, we first detail an approach to long-term *strategic* imitation and navigation, modelling the human player's observed navigation strategies as a Markov Decision Process and applying reinforcement-learning techniques to discern the relationship between his current situation and subsequent behaviour. The system further incorporates such features as the human's ability to account for dynamic changes in the local and global environment, the weighting and pursuit of multiple objectives in parallel, and the derivation of a goal-oriented topology of the game environment from recorded data.

In Section 4.4, we then outline the integration of this system with a Bayesian approach to *tactical* motion-modelling based on the concept of action primitives; this allows the agent to imitate a human player's characteristically smooth motion,

behaviours such as environment-relative weapon selection, and complex medium-term actions such as Quake's signature *rocket jump*.

Finally, in Section 4.5 we present an approach to the imitation of *reactive* weapon-selection and combat behaviours using an interconnected series of expert neural networks. This involves first reconstructing the human player's visual perception of his opponent's motion from the available low-level data, and then emulating the inaccuracy - both intentional and unintentional - demonstrated in his aiming behaviours with varying weapon, angle, elevation and range.

For each of the imitative subsystem enumerated above, we also describe experiments designed to statistically validate their functionality.

### 4.1.1 Publications

The strategic navigation system detailed in Section 3.1 was published as "*Towards Integrated Imitation of Strategic Planning and Motion Modelling in Interactive Computer Games*" (Gorman & Humphrys 2005) in Proc. 3rd ACM Annual International Conference in Computer Game Design and Technology (GDTW 05), where it won the "Best Presentation" award

This paper was subsequently selected for publication in the ACM Computers in Entertainment Journal, Volume 4, Issue 4 (October-December 2006).

The integrated strategic planning and Bayesian tactical/motion-modelling system detailed in Section 3.2 was published as "*Bayesian Imitation of Human Behavior in Interactive Computer Games*" (Gorman, Thureau et al 2006) in Proc. Int. Conf. on Pattern Recognition (ICPR'06), volume 1, pages 1244-1247. IEEE, 2006.

The reactive combat system detailed in Section 3.3 was published as “*Imitative Learning of Combat Behaviours in First-Person Computer Games*” (Gorman & Humphrys 2007) in Proc. 10th International Conference on Computer Games: AI, Mobile, Educational & Serious Games, 2007.

## 4.2 Behaviour Model

One of the first questions to arise when considering the problem of imitation learning in games is, quite simply, “what behaviours does the demonstration encode?” Recorded Quake 2 matches, as with most modern games, typically comprise a wide variety of simultaneous, *multiplexed* activities; a well-structured model of the human player’s behaviour is therefore needed, to facilitate an orderly analysis of the observation data. To this end, Thureau et al [131] propose a three-level hierarchy of in-game behaviour based closely on Hollnagel’s Contextual Control Model [53]. Our visualisation of Thureau’s model, with slight modifications, is shown in Figure 4-1.

In this model, ***strategic*** behaviours refer to actions the player takes with long-term goals in mind. Goals may include maximising the number of weapons or items he possesses (thereby also *preventing* opponents from obtaining them), controlling certain areas which offer an advantage in terms of position or cover, and cycling around the map in an efficient manner. Strategic goals will shift as the situation warrants; if a player is low on ammunition but has plenty of health, then he will naturally attempt to find an ammunition refill or a different weapon in preference to an armour upgrade.

***Tactical*** behaviours encode a degree of planning, but are more localized than the global strategies. Tactical behaviours can take two main forms; *environment-relative* and *opponent-relative*. Environment-relative tactical behaviours include jumping in

order to cross a ravine, looking around while moving in order to spot enemies, or executing more complex manoeuvres such as rocket jumps (see later). Opponent-relative tactical behaviours concerns the manner in which a player reacts to an opponent in his immediate vicinity - how he moves to minimise the danger to himself, how to evade or engage the opponent in a premeditated fashion, and so forth. Tactical behaviours, particularly of the environment-relative kind, are closely linked to *motion modelling*.

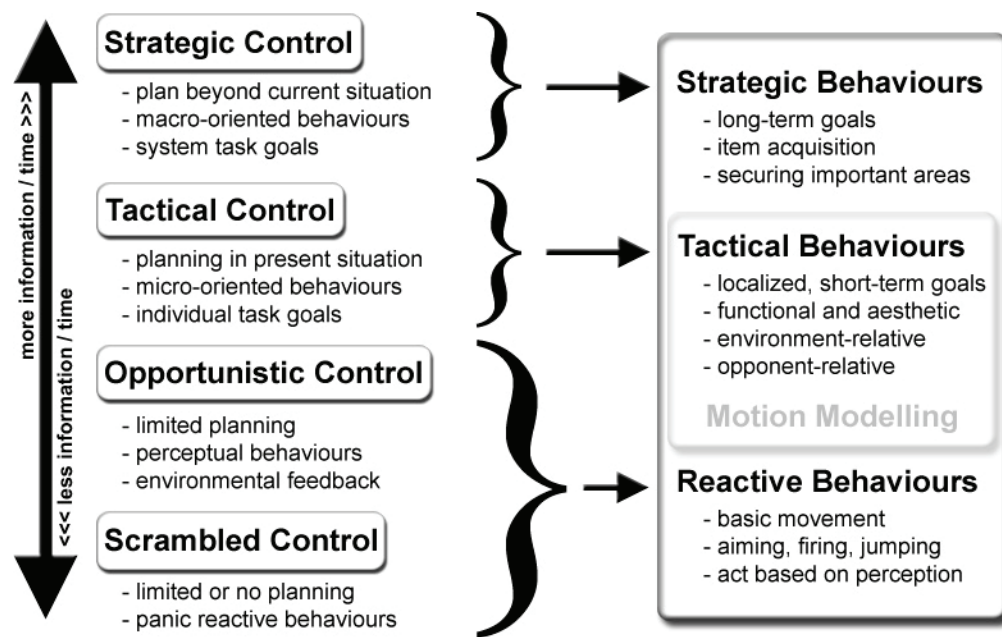


Figure 4-1 - Thurau's adaptation of Hollnagel's COCOM [131]

**Reactive** behaviours are those which involve little or no planning; the player simply reacts to stimuli in his immediate surroundings. An unexpected close-quarters encounter with an opponent would give rise to such behaviours, as would auditory input such as the sound of nearby gunfire. Basic aiming and firing behaviours also fall under this category.



***Motion modelling*** refers to the imitation of human movement as the agents execute their AI routines. This serves three functions; firstly, it manifests a smoother, more nuanced, more *humanlike* motion between points on the bot's path. Secondly, it prevents the artificial agents from performing actions which would be impossible for the human player's mouse-and-keyboard interface. Thirdly, it has an important *functional* aspect which overlaps with tactical behaviours; motion modelling allows the agent to actually perform the tactical manoeuvres - jumping, strafing, etc - necessary to navigate certain areas of its environment.

It is important to note that the behaviours encoded in a typical game session often cannot be strictly classified as belonging to a single category. For instance, while weapon selection and aiming behaviours would generally be considered *reactive*, they involve a distinct *tactical* component as well; in Section 4.5, we note that our imitative agents were observed to behave reactively when aiming at a fast-moving opponent in close proximity, whereas its aiming behaviours became more deliberate and tactical at longer ranges. The behaviour hierarchy should thus be viewed as a *gradient* along which the human's behaviours fall.

### 4.2.1 Behaviour and Believability

A number of previous contributions have examined the specific elements of agent behaviour which contribute most heavily towards their perceived 'humanness'. Laird and Duchi [78] define a set of such parameters - decision time, tactical complexity and aiming skill - which they expected to have a noticeable impact upon the agent's believability; for each of these, they instantiate agents at different values of the relevant parameter, and have them play against a human expert. They then showed recordings of these game sessions to a panel of judges, and analysed the relationship between the varying parameter values and the observer's perception of the agent. They found that the agent was judged most humanlike when it exhibited noticeable

strategic and tactical planning behaviours, and was least believable when its aim was excessively accurate. Elsewhere, McGlinchey and Livingstone [111] perform a series of similar experiments using the game Pong as a testbed, while Livingstone [79] extends the concept to consider Non-Player Character (NPC) agents, before providing a summary of the most important components of agent believability. With respect to our own work, the relevant *believability criteria* are as follows:

- the agent should not be merely a reactive bot; specifically, it should
  - exhibit strategic planning in the medium- to long-term
  - exhibit tactical planning in the short- to medium-term
- the agent should react to changes in its local environment
- the agent should act and aim with human-level precision and reaction times

In the following sections, we describe imitative mechanisms which fulfil each of these criteria. Further discussion on the importance of believability studies can be found in Chapter 5.

## 4.3 Strategic Behaviour Imitation

In a number of contributions [75, 78], the ability of agents to exhibit long-term strategic planning consistently emerges as a key factor in determining its “believability”. In order to learn strategic behaviours from human demonstration, we develop a model designed to emulate the notion of *program level imitation* discussed by Byrne and Russon [20]; in other words, to identify and imitate the demonstrator’s *goals* and *intent*, rather than simply reproducing his precise actions. On each timestep, the human player evaluates his current status, exercises his knowledge of the game environment, and makes a strategic decision to pursue a particular goal. Our task, then, is to identify and define the principal *goal-states* within the recorded game session, and to derive those factors which caused the player to move towards

them. In the context of Quake, strategic planning is mostly concerned with the efficient collection and monopolisation of *items* and the control of certain important areas of the map; we therefore define the goal-states as being the locations within the game world at which the agent collected a new item.

Thureau et al [131] present an initial approach to emulating such behaviours based on artificial potential fields; here, we instead apply a combination of reinforcement learning and fuzzy clustering to the problem, while adding a number of mechanisms designed to imitate other important aspects of strategic planning.

### 4.3.1 Reinforcement Learning

*Markov Decision Processes* [2, 69] are based on the idea that a problem can be modelled as a set of *states*, *actions* which may be taken in each, *transitions* which result from taking a particular action in a particular state, and *rewards* associated with certain states. Formally, an MDP consists of:

- a set of *states*,  $S$
- a set of *actions*,  $A$
- a *transition function*  $T$ , which defines  $P(s'|s, a) \forall s, s' \in S, a \in A$
- a *reward function*  $R$ , which defines  $R(s, a) \forall s \in S, a \in A$

On each timestep, the agent determines its current state, decides upon an action, transitions to a successor state, and receives a corresponding *reward* from the environment. Since we know the desired goal but not the correct actions to take, rewards are typically given upon reaching a particular state rather than for taking a particular action.

The objective is to find a optimal *policy*, that is, a mapping from states to actions which defines the agent's behaviour in a each state,  $\pi : S \rightarrow A, a_t = \pi(s_t)$ . The *value*  $V^\pi(s_t)$  of a policy  $\pi$  is the expected cumulative reward that will be received while the agent follows the policy from state  $s_t$ . The *optimal* policy  $\pi^*$  is thus the policy such that

$$V^*(s_t) = \max_{\pi} V^\pi(s_t), \forall s_t \in S$$

Depending on the application, the amount of knowledge we possess about the world may vary. Typically, we do not know the values of  $P(s'|s, a)$  and  $P(r|s, a)$ ; we must therefore *explore* the world in order to construct a valid policy, which we can then *exploit* to find the best route to the desired goal. One of the best-known exploration algorithms is *Q-Learning* [69], which operates by summing the rewards of future actions to build up a table of *quality values* associated with each  $(s, a)$  pair. The update rule generally used in Q-Learning is as follows:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a' \in A} Q(s', a'))$$

where  $\gamma$  is the *discount factor* used to reduce the value of future rewards,  $s'$  is the state to which the agent transitions upon taking action  $a$ , and the *learning rate*  $\alpha$  is generally taken as  $(1/t)$ , where  $t$  is the current timestep. The *control policy* of the agent is defined using a soft-max function such as the Boltzmann distribution:

$$P(s, a) = \frac{e^{\frac{Q(s, a)}{T}}}{\sum_{a' \in A} e^{\frac{Q(s, a')}{T}}}$$

where  $T$  is the *temperature* of the system. By starting with a high value of  $T$  and decreasing it as learning progresses, the agent gradually moves from a stochastic

*explorative* policy when the Q-values are at their most inaccurate, to a deterministic *exploitative* policy as its knowledge of the states and actions improves.

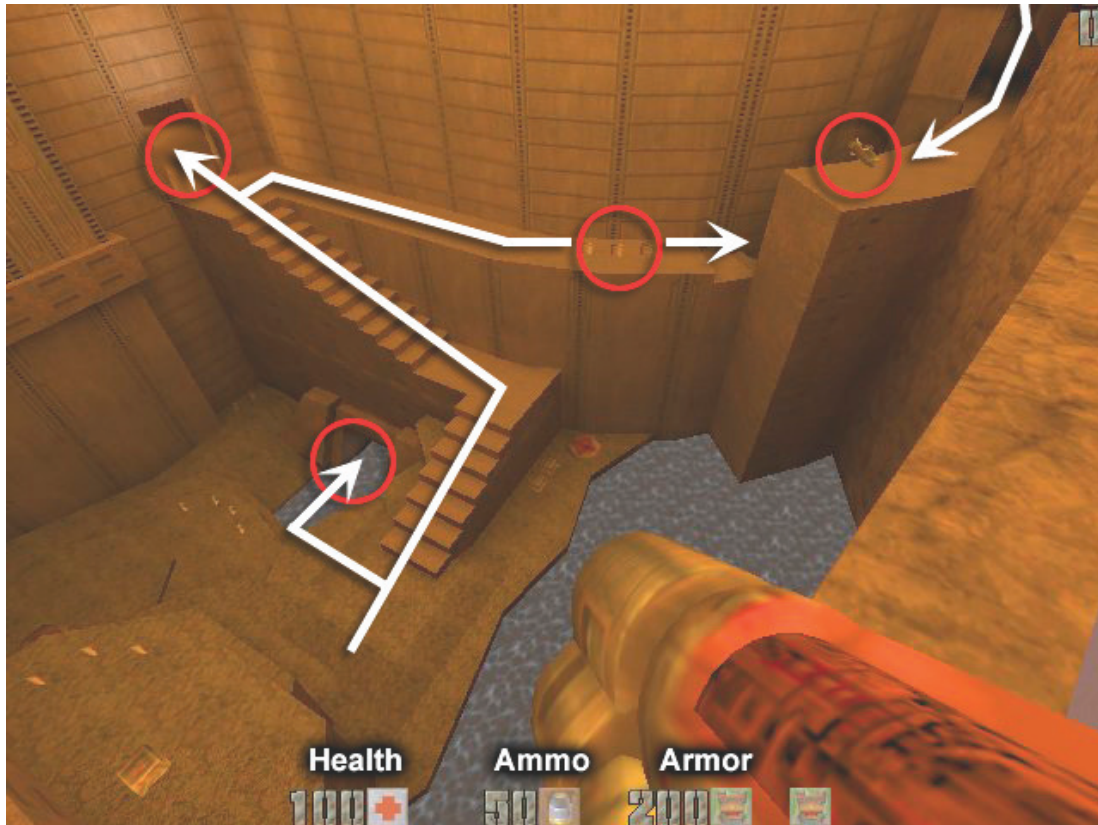
Unfortunately, it is not feasible to apply these techniques in an environment such as Quake 2; given the number of states - a Q2 map can, at maximum, be a 32k x 32k x 32k grid - and the fact that significant changes in the rewards can potentially occur every hundred milliseconds, the exploration time would be prohibitive. While an approach similar to that employed by Cottrell [25] might prove beneficial in adapting reinforcement learning to imitation, a more direct solution does present itself. Since we can compute the values of  $P(s'|s, a)$  and assign rewards to produce  $P(r|s, a)$  based on the observed actions of the player, one of two *iteration* techniques can be adopted to learn the optimal policy without needing to perform exploration.

The *value iteration* algorithm [69, Figure 4-2] is used to compute the *value*  $V(s_t)$  of each state, and has been shown to converge to the correct  $V^*$  values. The algorithm operates by storing the maximum Q-values of each state across all actions, updating them on each iteration; convergence occurs when the maximum variation between two successive updates falls below a given threshold. Once the values have been computed, the policy can be defined by simply choosing the transition which leads to the successor state with the highest utility value.

|  |
|--|
| <p><i>For all</i> <math>s</math> <i>in</i> <math>S</math></p> <p style="padding-left: 40px;"><i>For all</i> <math>a</math> <i>in</i> <math>A</math></p> <p style="padding-left: 80px;"><math>Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} P(s' s, a)V(s')</math></p> <p style="padding-left: 40px;"><math>V(s) \leftarrow \max_a Q(s, a)</math></p> <p><i>Until</i> <math>\max_{s \in S}  V^{l+1}(s) - V^l(s)  &lt; \delta</math></p> |
|--|

**Figure 4-2 - The Value Iteration algorithm**

A second algorithm, *policy iteration* [69] is used to obtain the optimal policy directly rather than indirectly over the utility values. Though very similar in form, policy iteration is not detailed here, since the value iteration algorithm is sufficient for the purposes intended.



**Figure 4-3 - More than one path exists through each environment. The player chooses his route in such a manner as will maximise his arsenal - and thus, his advantage over opponents - based on his current health, armour, weapon, ammunition and other inventory contents. Experienced players also attempt to maintain control over particularly important areas of the map.**

### 4.3.2 Goal-Oriented Strategic Navigation

In Quake 2, experienced players traverse the environment methodically, establishing control over important areas of the map and collecting items to strengthen their character (see Figure 4-3). Thus, we define the player's long-term strategic goals to

be the *items* scattered at fixed points around each level. By learning the mappings between the player's status and his subsequent item pickups, the agent can adopt observed strategies when appropriate, and *adapt* to situations which the player did not face. A strategic imitation system must therefore implement:

- a means of deducing the map's topology from demonstration
- a means of mapping the agent's current state (inventory) to the appropriate goal state (item pickup location) and actions (movement path)
- a means of allowing the agent to adapt to unseen states when deployed
- a means of allowing the agent to react to dynamic changes in its environment

Our system, designed to meet all these criteria, is described in the following sections.

### 4.3.3 Topology Learning

Learning the human player's strategic navigation of his environment first requires a means of building a representation of that environment's topology. Traditional AI bots in first-person shooter games generally employ *waypoint maps* - graphs of interconnected nodes, whose edges specify the paths along which the agent is free to move - which are manually constructed by the game designers. In keeping with our focus on imitation learning, we would prefer to *deduce* a topology from recorded game sessions. Adopting this approach has an additional advantage from the perspective of strategic imitation: by deriving a topological representation of the map from human demonstration, we implicitly discard those areas which the player did not visit, and which we can therefore infer were of little or no strategic value.

To learn the topology of the environment, we first read the set of all player locations  $\bar{l} = \{x, y, z\}$  from the recording, and *cluster* these points to obtain a reduced set of *prototype* player positions. By examining the sequence of player positions, we also construct an  $n \times n$  matrix of edges  $E$ , where  $n$  is the number of clusters, and  $E_{ij} = 1$  if

the player was observed to move from node  $i$  to node  $j$  and 0 otherwise. Thus, we derive a waypoint map based on the demonstrator’s traversal of his environment.

We quickly discovered, however, that naïve clustering often resulted in a topological map which was of insufficient granularity to capture the most important points in the dataset - that is, the points at which the player was observed to collect an item. **Figure 4-5** provides an illustration of this phenomenon. Here, the agent may inadvertently *miss* certain items which it was supposed to collect, since the placement of the waypoint nodes is such that the graph’s edges - that is, the agent’s traversable paths - do not intersect with the item collection points, represented as green squares.

We address this using a custom modification of the *k-means* algorithm. This involves introducing a collection of *immutable* “anchor” centroids, which specify points of particular importance in the dataset; in our case, these are the positions at which the player’s inventory (list of items) changed. The process operates by adjusting the set of initial cluster centroids, such that each anchor replaces the centroid nearest to it. Once done, the algorithm runs as normal, with the added stipulation that the anchors remain fixed throughout. In this manner, the points at which items pickups occurred stay constant, while the remaining clusters are distributed around them - in effect, providing us with an inherently *goal-oriented* discrimination of the level’s topology.

Formally, the algorithm proceeds as follows:

1. A set of *initial* centroids is first derived from the data; various approaches to the selection of these centroid positions are given in [18]. In our case, since the data encodes the near-continuous motion of an avatar through its environment with time, we sample the datastream at evenly-spaced intervals to produce the set of initial centroids.
2. The initial centroid which lies closest to the position of each *anchor* centroid (as defined above) is removed, and replaced by that anchor.



3. For each non-anchor centroid, recompute the cluster centre positions  $c_1 \dots c_n$  as the centroids of the set of data vectors they currently represent, i.e

$$c_i = \text{avg}(\sum x_j) \quad \begin{array}{l} c_i \notin \text{anchors} \\ \forall j \mid \arg \min_k (\text{dist}(x_j, c_k)) = i \end{array}$$

4. If the algorithm has *converged* (i.e. no change occurs in the assignment of input vectors to reference vectors for two successive iterations), then stop. Otherwise, go to step 2.

In practice, we also employ Elkan's *fast k-means* modification [32] to greatly expedite the clustering process. An illustration of how the waypoint map would appear if visualised in-game is shown in Figure 4-4 below; the effectiveness of our approach in capturing item collection points is shown in Figure 4-5 and Figure 4-6.

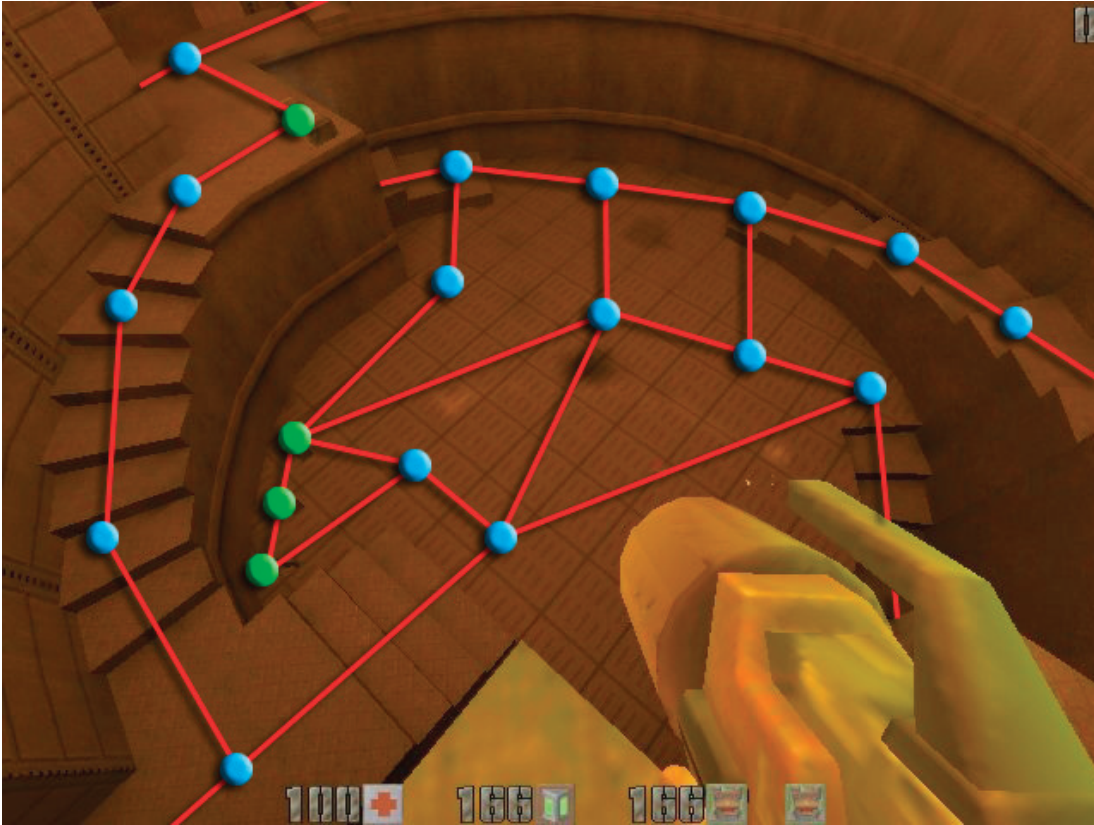
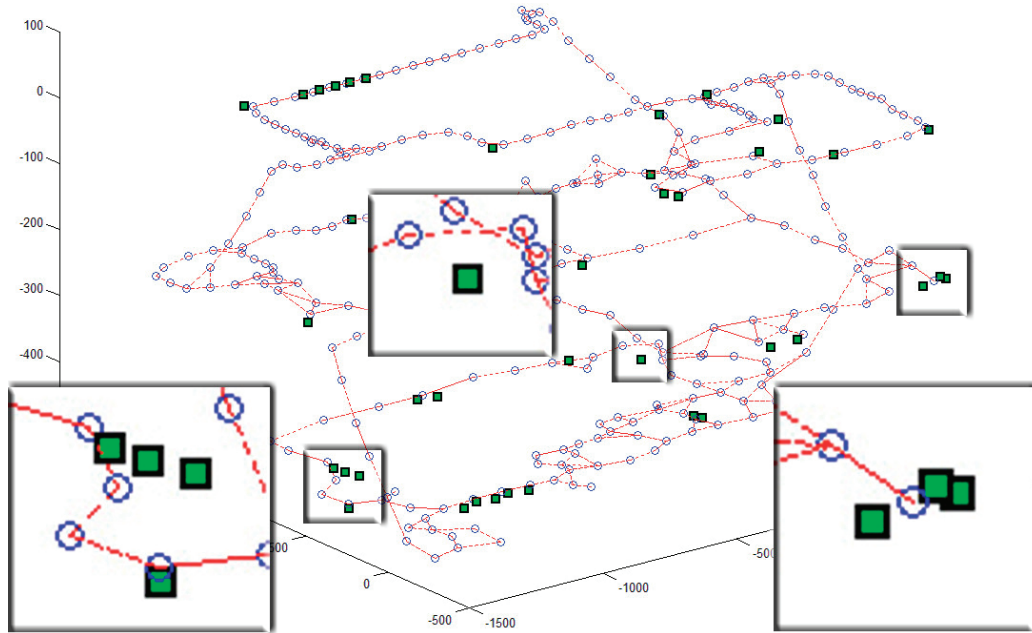
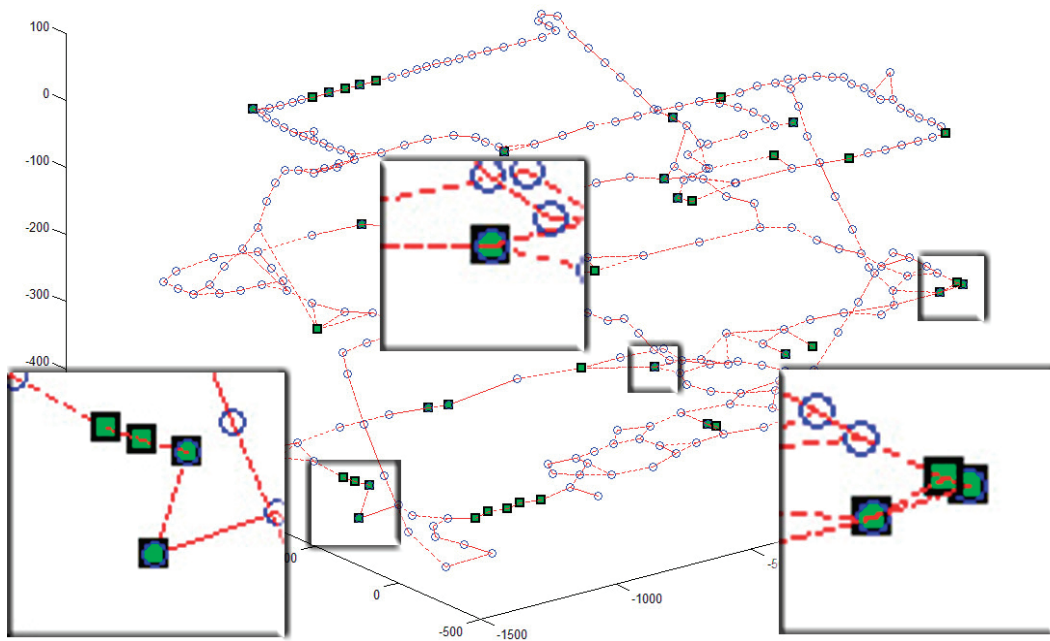


Figure 4-4 - An illustrative projection of the topological waypoint map onto a visualisation of an in-game scene. Nodes are blue, edges are red, nodes at which items reside are green.



**Figure 4-5 - A naive clustering of the player's positions reveals the topology of the level using the unmodified fast k-means algorithm. Note that several items (green) are not 'captured' by the waypoints (blue) and the edges between them (red).**



**Figure 4-6. The modified algorithm produces a clustering which arranges itself around important points (i.e. strategic objectives) in the dataset, rather than treating them incidentally. It proves effective in capturing all item collection events.**

#### 4.3.4 Deriving Movement Paths

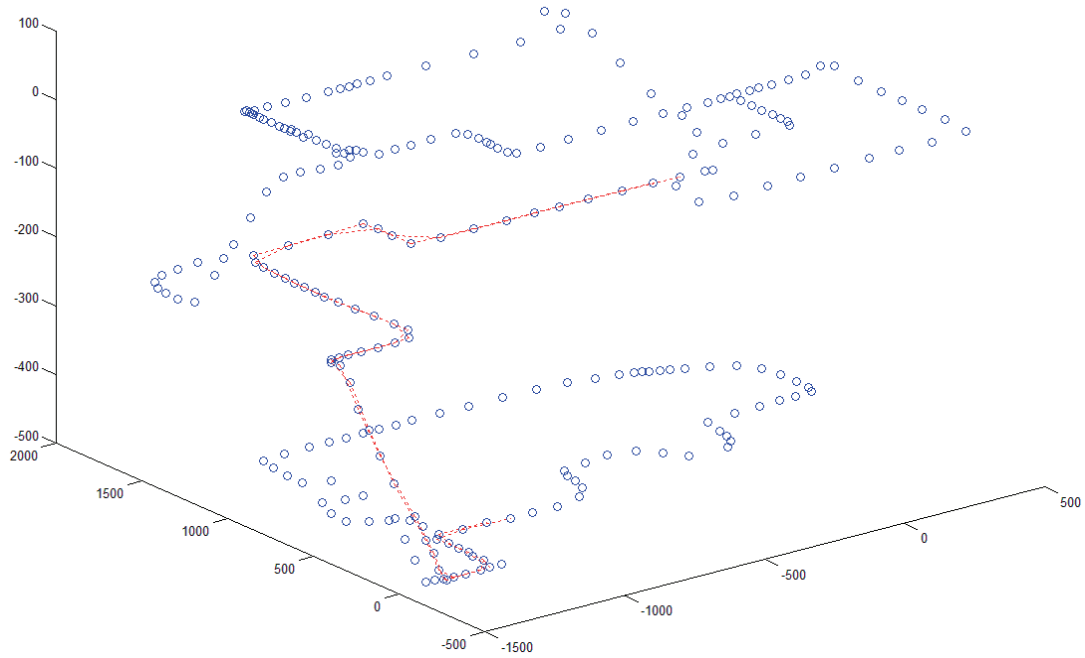
Typically, traditional agents use shortest-path methods to navigate from their current location to a desired waypoint node. From the perspectives of both imitation learning and humanlike behaviour, this is unacceptable. Firstly, human behaviour is inherently sub-optimal; while an experienced player will generally pursue a *good* path to his objective, it will not necessarily be the *best*. Secondly, an imitative approach to learning movement patterns has the effect of co-opting elements of human strategy that would be either difficult to achieve with a traditional agent, or would be overlooked entirely. A human player, for instance, may deliberately choose to take a longer path to his ultimate objective if it offers other benefits; a particular corridor may be poorly illuminated in the game world, and will thus make it more difficult for an opponent to spot and kill his avatar. A more circuitous route may permit him to collect ammunition for a weapon situated at the end of his current movement path, allowing it to be used immediately. Though the imitation agent will naturally not *know* that it is pursuing a specific path for these reasons, it will nonetheless gain the benefits of the human's strategic thinking. This ability to *subsume* elements of human reasoning, by deducing relationships between state and behaviour from observable outputs - without requiring that the reasoning processes themselves be manually coded - is what makes imitation such a powerful mechanism.

To capture this, we view the topological map of the game environment as a *Markov Decision Process* [54], with the clusters corresponding to states and the edges to transitions. We also read the player's *inventory* from the demo at each timestep - that is, the list of weapons and ammunition currently in his possession. We construct an *inventory state vector* specifying the player's *health* and *armour* values together with the *weapons* he has collected and the amount of *ammo* he has for each. The set of unique state vectors is then obtained; these *state prototypes* represent the varying

situations faced by the player during the game session. We can now construct a set of *paths* which the player followed while in each such situation. These paths consist of a series of transitions between clusters:

$$t_i = [c_{i,1}, c_{i,2}, \dots, c_{i,k}]$$

where  $t_i$  is a transition sequence (path), and  $c_{i,j}$  is a single node along that sequence. Each path begins at the point where the player enters a given state, and ends where he exits that state - in other words, when an item is collected that causes the player's inventory to shift towards a different prototype. Figure 4-7 illustrates a typical path followed in one such prototype.



**Figure 4-7 - An example of a path followed by the player while in a particular inventory state**

#### 4.3.4.1 Assigning Rewards

Having obtained the different paths pursued by the player in each inventory state, we turn to reinforcement learning to reproduce his behaviour. In this scenario, the

MDP's actions are modelled as the *choice to move to a given node from the current position*. Thus, the transition probabilities are

$$P(s' = j \mid s = i, a = j) = E_{ij}$$

where  $s$  is the current node,  $s'$  is the next node,  $a$  is the executed action, and  $E$  is the edge matrix. Since each action in each state leads unambiguously to a single successor state, this produces a deterministic mapping of the form

$$s, a \rightarrow s'$$

This has an added advantage in that each entry of the probability matrix can be represented as a single bit, offering a significant reduction in storage overhead. To guide the agent along the same routes taken by the player, we assign an increasing reward to consecutive nodes in every path taken under each prototype, such that

$$R(p_i, c_{i,j}) = j$$

where  $p_i$  is a prototype, and  $c_{i,j}$  is the  $j^{\text{th}}$  cluster in the associated movement sequence. Each successive node along the path's length receives a reward greater than the last, until the final cluster (at which an inventory state change occurred) is assigned the highest reward. If a path loops back or crosses over itself en route to the goal, then the higher values will overwrite the previous rewards, ensuring that the agent will be guided towards the terminal node while ignoring any non-goal-oriented diversions along the way. Thus, as discussed earlier, the agent will emulate the player's *program-level* behaviour - that is, it will identify and pursue the human demonstrator's *strategic goals*, rather than simply duplicating his precise actions. An example of such behaviour is shown in Figure 4-8 below.

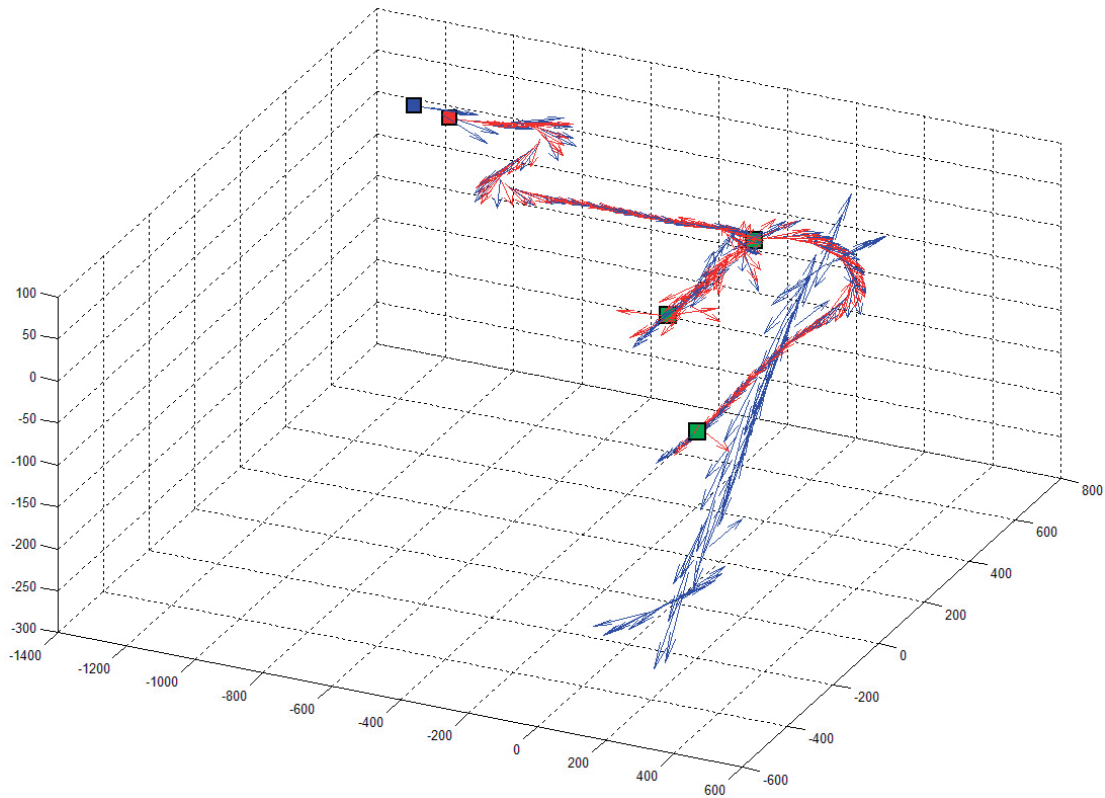


Figure 4-8 - A quiver plot illustrating program-level imitation. Pictured above are the player's (blue) performance of an item collection sequence, and the agent's reproduction of same (red); starting positions are given by their respective coloured squares, items are in green. In the middle of the sequence, the player descends and re-ascends a staircase, with no objective benefit. The agent ignores this non-goal-oriented movement, bypassing the stairs and heading directly towards the final item pickup at the near-right.

### 4.3.5 Learning Utility Values

With the transition probabilities and rewards in place, we can now run the *value iteration algorithm* in order to compute the utility values for each node in the topological map under each inventory state prototype. The value iteration algorithm iteratively propagates rewards outwards from terminal nodes to all others, discounting them by distance from the reward signal. Once complete, these utility values will represent the “usefulness” of being at that node while moving to the goal.

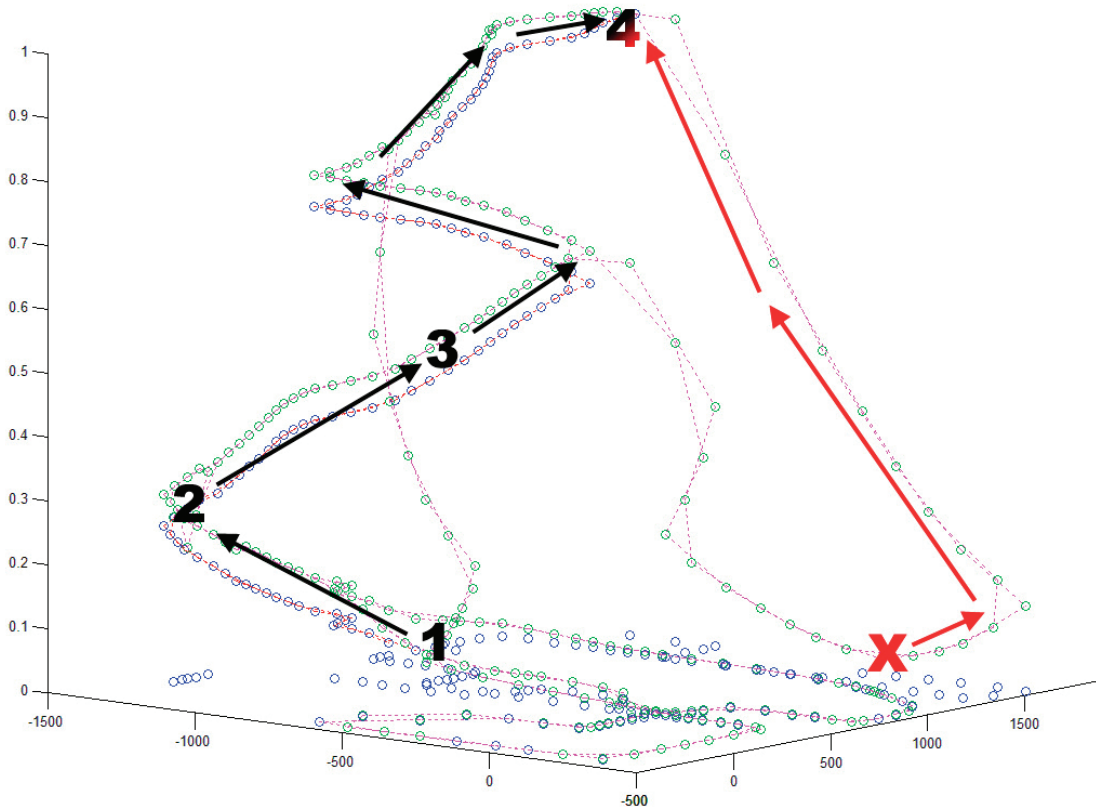


Figure 4-9 - Rewards (red & blue) and learned utility values (magenta & green) shown on the vertical axis. Note that every node has a nonzero utility value; the agent will be drawn towards the goal from any point on the map. The program-level approach effectively abstracts the agent's *situation* from the implementation of its *solution* - if the agent encounters this particular inventory state while at position X, for instance, it will follow an alternate route to the terminal node 4 rather than copying the human demonstrator's path (1-2-3-4).

Generally, the algorithm's stopping criterion is based on the variation of the utility values between consecutive iterations; however, in a complex world such as that presented here, convergence may be prohibitively slow. In our case, it is important that *every* node in the map should possess a utility value under *every* state prototype by the end of the learning process, thereby ensuring that the agent will always receive strong guidance towards its goal. We therefore adopt the *game value iteration* approach outlined by Hartley et al [50]; the algorithm is applied until all nodes have been affected by a reward at least once - that is, until every node has a non-zero

utility value. Figure 4-9 shows the result of applying value iteration to one particular path through the environment.

Furthermore, the value iteration algorithm itself can be greatly expedited by exploiting the deterministic nature of this particular MDP. Using the INRA MDP Toolbox 2.0 for MatLab, the main value iteration code appears as follows:

```
for s = 1:S
    for a = 1:A
         $Q(a) = R(s,a) + discount * P(s,:,a) * V;$ 
    end;

     $V(s) = max(Q);$ 
end;
```

**Figure 4-10 - Standard value iteration code**

where  $R$  is a two-dimensional reward matrix, and  $P$  is a three-dimensional transition matrix. In an example run of 442 nodes, this takes 139 seconds to affect every state for a particular path around the map. In our scenario, however, each action inherently implies a movement to the associated state; the code can thus be rewritten as follows:

```
for s = 1:S
     $V(s) = max(R(s, :) + discount * (P(s, :) .* V));$ 
end;
```

**Figure 4-11 - The rewritten value iteration code**

The transition matrix is reduced to two dimensions, and inner loop is *vectorized* so that the rewards of all actions in a given state are computed simultaneously. On the same dataset and machine, this code produces the same results as above in just 1.016 seconds.



### 4.3.6 Multiple Weighted Objectives

Beyond the mapping of inventory status onto navigation decisions, a number of other features of human planning behaviour must also be taken into account. Faced with a situation where several different items are of strategic benefit, for instance, a human player will intuitively *weigh* their respective proximities and importance before deciding on his next move. To model this, we adopt a *fuzzy clustering* approach. On each update, the agent's current inventory is expressed as a *membership distribution* across all prototype inventory states. This approach produces an aggregated set of utility values which reflect the similarity of the agent's current state to each of those experienced by the player, thereby allowing the agent to utilize its knowledge of the human's actions across the entire game session rather than considering only the single most analogous situation. For instance, in cases where item A is less attractive than item B given the agent's current status, but item A is located closer to its current position, the agent will be drawn towards the goal state associated with A. This is computed as follows:

$$m_p(s) = \frac{\|\vec{s} - \vec{p}\|^{-1}}{\sum_{i=1}^P \|\vec{s} - \vec{i}\|^{-1}}$$

where  $s$  is the current inventory state,  $p$  is a prototype inventory state,  $P$  is the number of prototypes, and  $m_p(s)$  is the degree to which state vector  $s$  is a member of prototype  $p$ , relative to all other prototypes.

This approach has the advantage of increasing the agent's *adaptability* when deployed. In the case of the recorded human data, we can determine the player's precise objective on every timestep, simply by reading ahead in the packet stream and determining which item he eventually collected. When the imitation agent is deployed, however, it will likely be required to begin its navigation from a different point on the map, and its inventory states will therefore be markedly different to

those observed during the learning process. Weighting across all objectives in this manner ensures that the agent will pursue consistent behaviours even in such cases.

### 4.3.7 Object Transience

Another important element of planning behaviour is the human's understanding of *object transience* - that is, a collected item will be unavailable until the game regenerates it after a fixed interval. A human player intuitively tracks which items he has collected from which areas of the map, can easily estimate when they are scheduled to reappear, and adjusts his strategy accordingly. To capture this, we introduce an *activation* variable in the computation of the membership values; inactive items are nullified, and the membership values are redistributed among items which are still active:

$$m_p(s) = \frac{a(o_p) \|\vec{s} - \vec{p}\|^{-1}}{\sum_{i=1}^P a(o_i) \|\vec{s} - \vec{i}\|^{-1}}$$

where  $a$ , the *activation* of an item, is 1 if the object  $o$  at the terminal node of the path associated with prototype state  $p$  is present, and 0 otherwise. The utility configurations associated with each prototype are then weighted according to the membership distribution as described above, and the adjusted configurations *superimposed*; we also apply a *discount* to prevent backtracking. The formulae used to compute the final utility values and select the next position cluster is thus:

$$U(c) = \gamma^{e(c)} \sum_{p=1}^P V_p(c) m_p(s)$$

$$c_{t+1} = \max_y U(y), y \in \{x \mid E_{c,x} = 1\}$$

where  $U(c)$  is the final utility of node  $c$ ,  $\gamma$  is the discount,  $e(c)$  is the number of times the player has entered cluster  $c$  since the last state transition,  $V_p(c)$  is the original value of node  $c$  in state prototype  $p$ , and  $E$  is the edge matrix. Imitating the human's

comprehension of object transience allows not only the adjustment of long-term strategies as items respawn in distant areas of the map, but also fulfils the believability criterion as outlined in Section 4.2.1 that an agent should react to changes (in this case, item respawns) in its immediate environment.

Further discussion, and detailed examples, of both the objective-weighting and transience mechanics are presented in the Experiments section and Figure 4-12.

### 4.3.8 Deployment

When deployed, the agent passes its current location and inventory to MatLab via the QASE interface on each timestep. MatLab then determines the closest-matching topological node, computes the similarity of each state prototype to the given inventory state, weights the utility values accordingly, finds the set of nodes connected to the current node by examining the edge matrix, and selects the successor node with the highest utility value. As the agent traverses its environment, item pickups and in-game events will cause its inventory to change, resulting in a corresponding switch in the utility configuration and attracting the agent towards its next objective.

### 4.3.9 Experiments

Here, we present a statistical validation of the strategic planning system. A comprehensive evaluation of the mechanism as it pertains to the agent's perceived *believability* or *humanness* is presented in Chapter 5.

To evaluate the agents' ability to learn the human's strategic behaviour, we first conducted 150 separate tests on 30 sequences of simple item pickups spanning four different game levels. For each set of five tests, the first attempt placed the agent at

the same starting position as the player; the four subsequent tests placed the bot at a random position along the sequence. When starting at the same location as the player, the agent was expected to reproduce the exact sequence of pickups. When starting at each of the random positions, it was expected to *attempt* to reproduce the sequence if possible; however, if its starting location caused it to collect an object while en route to the first item in the sequence, the agent was expected to deal with this unencountered situation appropriately. Each topological map was constructed at a cluster ratio of 35%; that is, the number of waypoints was 35% of the total number of positions encoded in the underlying dataset. Because these samples represented isolated pickup sequences rather than continuous gameplay, the agent did not take item respawns into account - this was tested separately on more extensive samples, as described later. Also, to account for the random positioning of the agent, we made each internodal link in the topological map bi-directional. For each test, the number of seconds taken for the bot to complete its task was recorded, and can be compared with the human player's time. These results are summarized in Table 4-1 below.

| <b>Dataset</b> | <b>Player</b> | <b>Bot (Same SP)</b> | <b>Rand 1</b> | <b>Rand 2</b> | <b>Rand 3</b> | <b>Rand 4</b> |
|----------------|---------------|----------------------|---------------|---------------|---------------|---------------|
| 1              | 24.8          | 14.2                 | 25.4          | 36.7          | 23.6          | 20.1          |
| 6              | 12.0          | 11.8                 | 16.1          | 15.3          | 21.4          | 18.1          |
| 7              | 15.8          | 10.8                 | 11.1          | 10.8          | 12.6          | 15.9          |
| 9              | 79.2          | 51.0                 | 76.9          | 72.1          | 61.3          | 76.9          |
| 10             | 60.8          | 47.1                 | 46.8          | 71.7          | 50.9          | 72.5          |
| 11             | 32.8          | 43.1                 | 56.0          | 68.7          | 58.2          | 65.7          |
| 18             | 54.4          | 20.6                 | 31.2          | 22.7          | 20.4          | 37.2          |
| 19             | 20.8          | 12.7                 | 43.3          | 12.9          | 43.5          | 20.7          |
| 20             | 50.4          | 29.3                 | 76.9          | 31.0          | 33.4          | 31.2          |
| 21             | 20.0          | 17.3                 | 15.8          | 13.5          | 11.2          | 12.2          |
| 22             | 38.4          | 51.3                 | 43.0          | 40.1          | 59.8          | 47.4          |
| 27             | 28.8          | 37.9                 | 55.8          | 35.2          | 35.6          | 52.1          |
| 30             | 24.8          | 29.6                 | 28.3          | 30.0          | 34.0          | 35.1          |

**Table 4-1 - Illustrative results of human navigation reproduction using strategic imitation. Numbers are the time taken for the agent to complete the human's pickup sequence from the same starting position and each of four random positions. Results were chosen for varying complexity, duration and agent performance.**

As can be seen, the agent successfully completed each task, although with greatly varying times. When the agent started at the same point as the human, the times are generally quite similar; in some cases, the agent actually manages to complete the collection sequence faster than the demonstrator, since it will (as noted above) ignore any non-goal-oriented movements present in the gameplay sample. The deviations recorded in the random-spawn tests are due primarily to the fact that the agent often started around the midpoint of the sequence, requiring it to either collect all subsequent or all previous items before doubling-back to pick up the remainder. Certain samples also saw noteworthy differences in the time taken for the agent to complete its task when starting from the *same* position as the human player. In these cases, it was observed that the bot tended to move in rough, circuitous motions between what should have been adjacent points; this was found to be caused by a concentration of complex paths through relatively small spaces, the slightly high (35%) number of clusters used, and the resulting dense placement of nodes within the topological map. Subsequent experiments would clarify the detrimental effects of high waypoint densities on agent navigation (see Section 4.4.4). Some other minor issues also contributed to the observed time differences - most notably the occasional repetition of paths, caused by the artificial addition of bidirectional edges - but overall the bot showed impressive performance and adaptability, as evinced by its successful completion of pickup sequences when starting from arbitrary locations.

While useful for the purposes of numerical comparison, however, these isolated pickup sequences do not fully test the agent's planning abilities. A more instructive examination of its performance can be made by supplying more extensive gameplay samples, wherein the player is observed to cycle continuously around the map in accordance with his changing state. We therefore trained the agent on four extended demonstrations across two different levels, where the human player had more scope to develop the kind of elaborate strategy typically seen in a live game session. Here,

the *direction* in which the player traverses the map is often an important element of his strategy, allowing him to collect weapons before their relevant ammunition, or to build up armour reserves before entering the open areas of the level; consequently, edges in the topological map are taken as being unidirectional, except in cases where the player was explicitly observed to move along them in both directions. The objective-weighting and item transience mechanisms resulted in the emergence of a noticeably more complex, more involved long-term planning behaviour. Rather than simply cycling the map from one pickup to the next, the agents were instead observed to react dynamically to changes in an object's activation, to concentrate on areas of the map which were favoured by the human player over the course of his demonstration, and to give added significance to regions which were more heavily populated with beneficial items. For instance, in several samples the player was observed to first obtain a full complement of weaponry, after which he would patrol specific areas of the map which contained many of the most useful items; the agent took this into account by concentrating on those same areas throughout its test run.

One issue noted during these tests was an occasional 'indecisiveness' at points where multiple paths intersected - the agent would sometimes move among several neighbouring nodes, due to local maxima caused by overlapping utility values. In such cases, however, this problem was quickly resolved by the discount factor applied to recently-visited position clusters; this acted to push the bot away from such intersections and towards more attractive areas of the map, resulting in little overall disruption to the agent's movement.

Illustrative examples of the goal-weighting and object transience mechanisms are shown in Figure 4-12 on page 140. These demonstrate how the object transience approach fulfils the believability criterion, outlined in Section 4.2.1, that the agent should react to changes in its local environment. It is important to note, however, that

these microcosmic examples are not the sole effects of the system. While it is fortunate that some items happened to respawn in the bot's immediate vicinity, thus providing ideal visualisations of the mechanism in operation, the very fact that the agent was able to resolve these reappearances in the short-term negates their long-term planning importance. It bears re-stating that the agent is internally tracking the times at which the inactive items are scheduled to respawn, and therefore perceives these reappearances no differently than it does the reactivation of items in other, more distant areas of the map. The strategic implications of this mechanism occur when the agent realizes that items at medium or greater distances from its current position have respawned, leading it to adjust its long-term goals accordingly. The navigation system, while primarily a strategic mechanism, therefore incorporates situation-dependent elements of both *reactive* (when items at close range respawn) and *strategic* behaviour (when items at longer ranges respawn).

Finally, it is instructive to note that the agent was out of line-of-sight of these items as they respawned. In the case of a human player in the same scenario, he would enter the area knowing that the items were due to respawn in the near future, but not the precise time. When the items respawn, however, they emit a distinctive noise; this would alert the human player that they had reappeared, even if he had already moved past them and could no longer see them in his field of vision. From an observer's perspective, then, the imitation agents in the sequences below would be perceived as exhibiting *reactive behaviours* in response to an *auditory stimulus*. Along with the strategic imitation of the human's tendency to opt for poorly-lit corridors discussed earlier, this is an excellent example of how imitation learning can serve to co-opt aspects of human perception, and their influence upon behaviour, which would not be evident in a traditional rule-based agent.

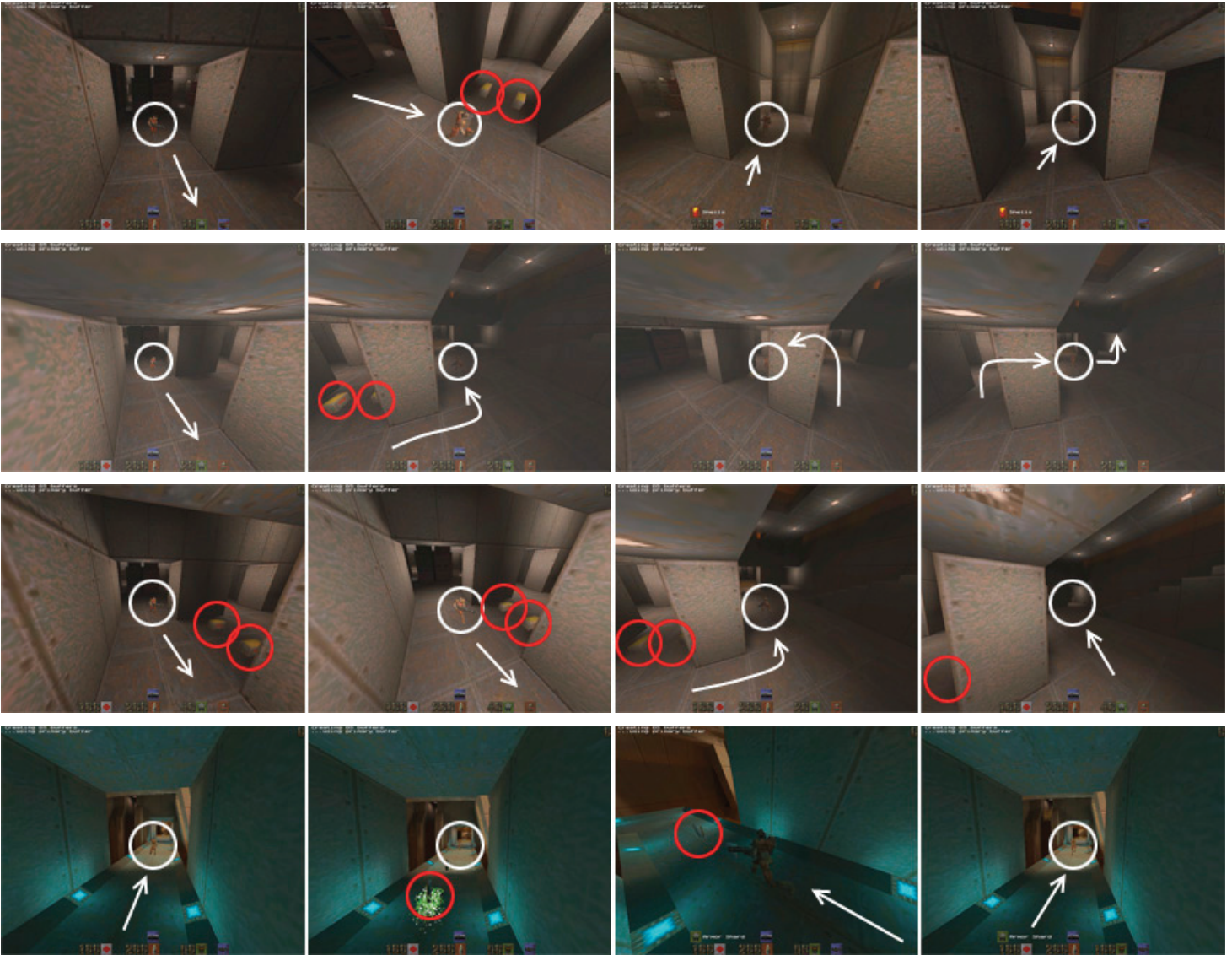
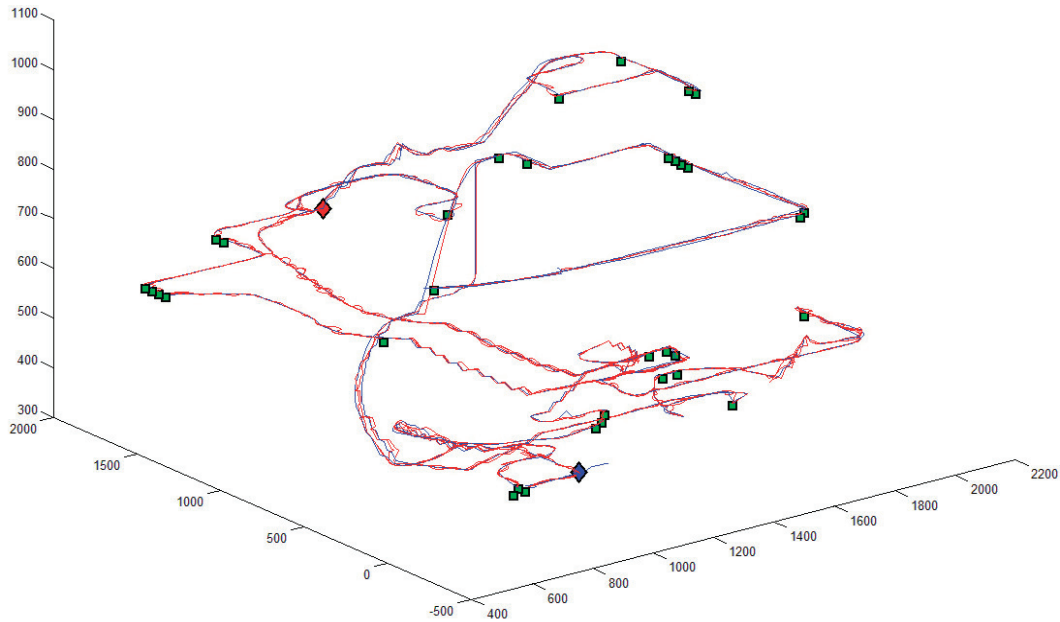


Figure 4-12 - Four sequences showing the objective-weighting and item transience mechanisms. Top, the agent (white circle) comes across two shotgun ammunition pickups (red circles) for the first time (1.1, 1.2), and collects them (1.3) before moving on (1.4). Having cycled around part of the level, the agent returns before the items have respawned (2.1), and since their inactivity nullifies the associated utilities, the bot initially passes by (2.2); however, their sudden re-emergence (2.2) causes the utilities to reactivate, and the agent is drawn to collect them (2.3) before continuing (2.4). Later, the agent returns once again (3.1). The items are now active, but since the agent has already collected several shotgun pickups, it has moved away from the prototype states associated with these items. The relevant membership values are thus insignificant; as a result, the agent ignores the pickups (3.2, 3.3), and continues on towards more attractive objectives (3.4). Finally, in a different session, the agent is moving along a path (4.1) as a row of armour shards respawns behind it (4.2). It reacts at once, turning around to collect the items (4.3), before resuming its course (4.4).





**Figure 4-13 - A continuous gameplay samples; the agent (red) successfully reproduced the observed behaviour (blue) from a different starting position. Item locations are represented as green squares.**

A full gameplay example is visualised via QASE/MatLab in Figure 4-13 above. Here, the agent (red) successfully reproduced the observed human behaviour (blue) having started at a different point (red diamond) than the demonstrator (blue diamond). Because of this, it was not possible to simply reproduce the exact sequence of pickups - instead, the agent was forced to rely on its objective-weighting mechanism to attract it towards each successive item location.

Further evaluation of the strategic navigation system, both in terms of imitative accuracy and in the context of its contribution to the perceived “humanness” of the agent, is detailed in Chapter 5.

## 4.4 Tactical Behaviour Imitation

In this section, we first outline the relationship between motion modelling and tactical imitation, and discuss the importance of their role in creating a believably

humanlike agent. We then describe the development of our approach to integrating a motion modelling and tactical imitation mechanism with the strategic navigation system described above, based on the theory of *action primitives*.

#### 4.4.1 Motion Modelling and Tactical Behaviours

While the strategic navigation approach described in the previous section does provide an accurate reproduction of the recorded player's long-term planning and his traversal of the environment, the agent's movement between nodes in the topological map appears jerky and stilted - it does not capture the aesthetically smooth motion which human players typically exhibit. Since the illusion created by even the most intricate planning behaviours will be shattered if the agent is observed to move in an unconvincing manner, it therefore becomes imperative to implement a mechanism capable of imitating the human player's *motion*. We refer to this requirement as *motion modelling*. In addition, human players often exhibit behaviours which serve no observable purpose during game sessions; unnecessary jumps, arbitrary weapon discharges and other such actions are commonplace. Even simple actions, such as standing in place while looking around for enemies or turning the camera to visually inspect an upcoming obstacle, are distinctively human behaviours; these are generally not exhibited by traditional agents, since visual examination of the environment is redundant from their perspective. While emulating such behaviours will not contribute towards the agent's effectiveness, their absence would greatly detract from the perception of the agent as a human player; thus, by imitating such idiosyncrasies, it may be possible to make the artificial opponent appear more lifelike. As discussed in Chapter 1, this is in marked contrast to most applications of robotic imitation learning, where these unmotivated and unnecessary actions are considered nuisances which detract from the optimality of the solution [29].



**Figure 4-14 - A player (white circle) wishes to collect an item (red circle) located on top of a high platform. Directional navigation alone is not sufficient to reach all strategic goals; a complementary tactical imitation system is required to negotiate obstacles such as this.**

Beyond this, motion modelling has an important *functional* role in the realisation of environment-relative *tactical* behaviours; an example is shown in Figure 4-14. As discussed earlier, tactical behaviours are those which operate over a shorter term than strategic behaviours, but nonetheless encode a degree of localized planning. The virtual environments manifested in Quake, as with most modern games, are highly complex; simply navigating along the environment surface from one point to another is not sufficient. Players may need to jump across a ravine, stand on an elevating platform, use doors or switches, and so forth. They perform tactical weapon-switching behaviours, such as drawing their rocket launcher as they approach an obstacle in order to perform a *rocket jump*, or choosing the Hyperblaster and firing at the entrances to an open area to cover their movements from potential enemy

encroachment. In many cases, the player can only reach certain areas of the map and attain certain goals by performing one or more such actions at the appropriate time. The goal-oriented navigation system, in other words, can be interpreted as determining *where* the agent needs to go, while the tactical motion model provides the *means* to get there.

For the purposes of our agent, then, an ability to imitate the behaviours discussed above is essential. To do so, we draw on the theory of *action primitives*.

#### 4.4.2 Action Primitives

Our initial attempts to integrate tactical motion modelling with the strategic navigation system utilised an adaptation of an approach proposed by Thureau et al [132], based on the theory of *action primitives* as described in Chapter 2. Action primitives are sets of modularised motor commands which are capable, via combination, of generating entire movement repertoires [37]; they are widely documented in the behaviour, biological and robotic literature [28, 43, 60, 127].

To extract primitives from the recorded Quake 2 data, we first read the sequence of *actions* executed by the player at each interval during the game session. This results in a set of four-dimensional vectors as shown below:

$$a = [yaw, pitch, fire, jump]$$

That is, the angular orientation of the player in each plane, whether he is firing his weapon, and whether he is jumping. We then *cluster* the set, thereby aggregating similar actions executed at different times or locations, and reducing the data to a smaller set of *prototypical actions*, otherwise known as *primitives*.

In order to reconstruct the human player's motion, the agent must also learn how to *sequence* these primitives. To this end, we employ a series of *probability functions*

similar to Thureau [132]. In that case, however, the primitives were used in isolation; they were not underpinned by a dedicated navigation system. We must therefore adapt the approach, such that the *strategic planning* system outlined earlier is responsible for navigation around the environment, and the *motion modelling* system is layered on top.

Since the player's action depends on his current location in the environment, the probability of executing a particular action primitive  $a_j$  when the agent is within the Voronoi region of waypoint  $w_k$  - that is, closer to  $w_k$  than to any other node in the topological map - was originally written as

$$P(a_t = a_j) = P(a_j | w_k)$$

However, in our new model, this could lead to situations where an incongruous primitive is chosen if the agent is currently at a node with more than one possible successor; for instance, if the navigation system chooses to move left, but the motion-modelling system chooses a primitive which the player executed while moving to the right. To constrain the motion model such that it adheres to the paths chosen by the navigation system, we choose primitives based not on the current *node*  $w$ , but rather on the *edge*  $e$  along which the agent is travelling; that is,

$$P(a_t = a_j) = P(a_j | e_k)$$

The conditional probabilities can be computed by re-examining the recorded data. Each action in the training set can be associated with an edge in the topological map, and it can be associated with an action primitive. By counting the number of times a particular action primitive is observed to be executed on a given edge, an  $m \times n$  matrix is formed where  $m$  is the number of edges and  $n$  is the number of action primitives; the probabilities are then deduced by normalising the rows of the matrix. An entry at position  $(k, j)$  thus indicates the probability  $P(a_j | e_k)$  as above.

We must also consider the fact that not every action primitive can be executed as the successor of every other; there is a constrained ordering of primitives, imposed largely by the limitations of the player's physical interface (see Section 4.4.3.4). As such, it is necessary to introduce a condition which expresses the probability of performing a particular action given the previously-executed action:

$$P(a_t = a_j) = P(a_j | a_{t-1})$$

where  $a_{t-1}$  is the action which was executed on the last time step. These probabilities are computed in a similar manner to the above; an  $n \times n$  transition matrix is constructed, and populated by observing the sequence of actions from the demo. Since the conditional probabilities  $a_{t-1}$  and  $e_k$  are independent - the current state (edge) affects only the next action, not the previous one - the overall probability of executing action primitive  $a_j$  is therefore given as

$$P(a_t = a_j) = P(a_j | a_{t-1}, e_k) = \frac{P(a_j | a_{t-1})P(a_j | e_k)}{P(a_j)}$$

While this system proves capable of manifesting humanlike motion while pursuing the agent's strategic goals, it is a less than satisfactory solution. One of its primary shortcomings is its inherently *disjoint* nature - that is, the motion modeling system is layered on top of the strategic-planning system, with no communication and only minimal relation between them. The planning module picks the next edge to traverse, and the motion module picks the actions on that basis, with no regard for the agent's condition or objectives. For this reason, we continued to investigate alternative approaches, with the aim of developing a more robust integration.

### 4.4.3 Bayesian Imitation

During a research visit to Bielefeld University, we discovered that Thureau et al were developing an improved approach to primitives based on Rao et al's Bayesian model

of action sequencing in infants and robots [106]. This formulates the choice of action at each timestep as a probability function of the subject's current state  $s_t$ , next state  $s_{t+1}$  and goal state  $s_g$ , as follows:

$$P(a_t = a_j | s_t, s_{t+1}, s_g) = \frac{P(s_{t+1} | s_t, a_j)P(a_j | s_t, s_g)}{P(s_{t+1} | s_t, s_g)}$$

where the *normalisation constant* is computed by marginalising over actions:

$$= \frac{P(s_{t+1} | s_t, a_j)P(a_j | s_t, s_g)}{\sum_u P(s_{t+1} | s_t, a_u)P(a_u | s_t, s_g)}$$

It was immediately apparent that, by viewing these *states* as the waypoints in our topological map, Rao's model was an ideal fit for the strategic navigation approach described in Section 4.3; the position states  $s_t$  and  $s_{t+1}$  are defined by the utility values, while the current goal(s) are defined by the inventory membership distribution. We therefore decided to develop a unified model of strategic and tactical imitation, by integrating these two systems.

#### 4.4.3.1 Primitives Extraction and Sequencing

As with the approach to primitives detailed in Section 4.4.2, we first read the sequence of *actions* executed by the player on each timestep. This results in a set of extended action vectors  $a$  such that

$$a = [\text{yaw}, \text{pitch}, \text{walkstate}, \text{jump}, \text{weapon}, \text{firing}]$$

where  $\text{yaw} \in [-180, 180]$  and  $\text{pitch} \in [-90, 90]$  define the player's view orientation,  $\text{walkstate} \in [0, 2]$  determines whether the player is stationary, walking or running,  $\text{jump} \in [-1, 0, 1]$  indicates whether the player is crouching, standing or jumping,  $\text{weapon} \in [1, 11]$  signifies his current weapon, and  $\text{firing} \in [0, 1]$  determines whether

or not the player is discharging his weapon. As before, we cluster these action vectors to produce an aggregated set of basis behaviours.

According to Rao's model, the probability of executing each action at a given timestep can be computed according to:

$$P(a_t = a_j) = P(a_j \mid w_t, w_{t+1}, w_g)$$

where  $w$  is used to denote a *waypoint* in the topological map, to avoid confusion between the *states* in Rao's model and the strategic imitation system's *inventory states*. However, a familiar problem confronts us. In the original model, Rao assumes that each action is independent of the last; in Quake 2, by contrast, each action is constrained by its predecessor, due to the limitations of the human player's interface (see Section 4.4.3.4 for further discussion). To account for this, we must introduce an extra conditional probability in our calculations. Since, as noted by Thureau [130], the action observed on the previous timestep  $a_{t-1}$  is independent of  $w_t$ ,  $w_{t+1}$ , and  $w_g$  - sequencing of actions due to the human's external input modality does not affect in-game environmental conditions - the new probabilities are computed as:

$$\begin{aligned} P(a_t = a_j) &= P(a_j \mid w_t, w_{t+1}, w_g, a_{t-1}) \\ &= \frac{P(w_{t+1} \mid w_t, a_j) P(a_j \mid w_t, w_g, a_{t-1})}{P(w_{t+1} \mid w_t, w_g)} \end{aligned}$$

but,

$$\begin{aligned} P(a_j \mid w_t, w_g, a_{t-1}) &= \frac{P(a_j \mid w_t, w_g) P(a_j \mid a_{t-1})}{P(a_j)} \\ \Rightarrow P(a_t = a_j) &= \frac{P(w_{t+1} \mid w_t, a_j) P(a_j \mid w_t, w_g)}{P(w_{t+1} \mid w_t, w_g)} \frac{P(a_j \mid a_{t-1})}{P(a_j)} \end{aligned}$$



Therefore,

$$P(a_t = a_j) = P(a_j | w_t, w_{t+1}, w_g) \frac{P(a_j | a_{t-1})}{P(a_j)}$$

#### 4.4.3.2 Adaptations of the Rao/Shon/Meltzoff Model

Despite the suitability of Rao et al’s model [106] for our work, the requirements of a game agent navigating a virtual environment are naturally quite different than those of a behavioural model of infant imitation (see Chapter 1). Several further adaptations were required before this approach could be used in the game world.

Firstly, Rao’s model assumes that transitions between states are instantaneous upon execution of the chosen action; in the context of our waypoint map, however, multiple actions may be performed while moving between successive position states. The importance of this distinction became clear in our initial tests, as a direct implementation of the model resulted in only a single action having a non-zero probability for each  $(w_t, w_{t+1}, w_g)$ . This, in turn, meant that the agent only executed a new primitive upon reaching each waypoint along its path. The result in the game world was that the agent was observed to make a sharp change in orientation as it passed each waypoint, and would then “lock” to this orientation while moving along the edge towards the next node. This was particularly pronounced in areas of the map with sparse waypoints and long edges, and had the effect of rendering the agent’s appearance highly artificial. To rectify this, we instead express  $P(w_{t+1}|w_t, a_t)$  as a probability distribution across all actions on edge  $E_{w_t, w_{t+1}}$  in the topological map.

Second, Rao assumes a single unambiguous goal, whereas we deal with multiple weighted goals in parallel. As discussed in Section 4.3.6, we thus derive the player’s specific goals during the learning process, and perform a weighting across all active

goal clusters when the agent is deployed. With this in mind, the probabilities are restated as follows:

$$P(a_t = a_j) = \sum_g m_g P(a_j | w_t, w_{t+1}, w_g) \frac{P(a_j | a_{t-1})}{P(a_j)}$$

where  $m_g$  is the weighting of the current inventory with respect to goal state  $g$ .

Finally, while the above is sufficient to produce an accurate imitation of the human's behaviour, a further optimisation presents itself. In the course of our initial investigation, some features of the action representation became apparent:

- the range of possible values of the latter four elements is quite limited in comparison to that of the orientation variables; the total state space of *[walkstate, jump, weapon, firing]* consists of only  $(3 * 3 * 11 * 2) = 198$  distinct states. Indeed, this is small enough to make clustering largely unnecessary, particularly due to the fact that the number of actual observed states will typically be much smaller.
- while the player's firing and jumping are in some cases dependent upon his position, goal *and* orientation (for instance, performing a rocket jump, firing at a particular point, jumping through a narrow gap), his orientation is dependent only on his position and goal state.

Therefore, we can *decouple* the yaw and pitch elements of the action vector from the remainder, add a dependency upon the former to the computation of the latter's probabilities, and sequence them individually. The action vectors are now rendered as separate *orientation* and *interaction* vectors, where

$$a^o = [\text{yaw}, \text{pitch}]$$

$$a^i = [\text{walkstate}, \text{jump}, \text{weapon}, \text{firing}]$$

The corresponding probabilities are calculated as:

$$P(a_t^o = a_j^o) = \sum_g m_g P(a_j^o | w_t, w_{t+1}, w_g) \frac{P(a_j^o | a_{t-1}^o)}{P(a_j^o)}$$

and

$$P(a_t^i = a_j^i) = \sum_g m_g P(a_j^i | w_t, w_{t+1}, w_g) \frac{P(a_j^i | a_{t-1}^i, a_t^o)}{P(a_j^i)}$$

This serves both to produce a more fine-grain clustering of the orientation primitives, and to more accurately model their sequencing; an additional clustering optimisation will be discussed in Section 4.4.3.4.

#### 4.4.3.3 Probability Computation and Deployment

With the model defined, we can now examine the demonstration data to extract the probabilities  $P(w_{t+1}|w_t, a)$ ,  $P(a|w_t, w_g)$ , and  $P(a|a_{t-1})$ . The first term is a *forward model* of the player's future state given his current action, as Rao et al discuss in the context of infant behaviour [106]. They propose that neonates learn the equivalent model through *body babbling* - that is, the proprioceptive development of an internal correspondence map between muscle commands and resulting limb configurations, acquired via trial-and-error experimentation with random body movements. Our task is somewhat simpler; a set of probability matrices are constructed based on the observed incidence of each primitive in the gameplay sample, indexed by position, goal and previous action in accordance with the formulae given above.

Once deployed, the *strategic navigation* system determines the agent's current position  $w_t$ , obtains the goal membership distribution  $m$ , computes the utility values for each waypoint in the topological map, and chooses the highest-valued successor

node  $w_{t+1}$ . The *tactical* subsystem then assumes control, using the data supplied by the navigation module - in addition to its own knowledge of the actions executed on the preceding timestep - to extract the relevant values from the probability matrices. Finally, it computes the likelihood of each primitive and selects the action with the *maximum a posteriori* probability. Note that it is an entirely valid alternative to choose the primitive by roulette-wheel selection over candidate actions; we use maximum a posteriori simply because it maximally exploits the knowledge gained from the human player's demonstration.

#### 4.4.3.4 Imitative Keyframing

With respect to the *orientation* primitives, there is still one issue to resolve regarding their precise representation. Two options present themselves:

- *relative* primitives. Under this approach, the vector extracted from the gamestate on each timestep consists of the *change* in orientation from the preceding timestep, i.e.

$$a^o = [\Delta yaw, \Delta pitch]$$

- *absolute* primitives. Using this approach, the orientation vector extracted from the gamestate consists of the *absolute* yaw and pitch values of the player at that timestep, i.e.

$$a^o = [yaw \in [-180, 180], pitch \in [-90, 90]]$$

These two are somewhat equivalent to the *action space* and *task space* representation of state data discussed by Schaal [8] in the context of robotic motor control; that is, relative primitives require knowledge of the agent's internal state, whereas absolute primitives are specified in observable game co-ordinates. In the case of relative primitives, the comparatively small range of observed values of  $\Delta yaw$  and  $\Delta pitch$  -

due both to the constraints of the human’s physical interface and the short (100ms) period between samples, which preclude exaggerated actions such as turning  $180^\circ$  in a single timestep - means that action primitives derived from the demonstration data are highly accurate, even at low cluster densities. Agents trained using this approach exhibited the ability to make very fine adjustments to their orientation while navigating, as is typical of human players. However, our initial tests identified a problem. In cases where the agent adopted a different trajectory through the environment compared to its human exemplar - as is inevitable given the complexity of the game world and our adoption of program-level navigation - it would occasionally traverse the distance between two waypoint several timesteps faster or slower the human player had. Thus, a primitive that the human was observed to execute on edge B would be appended to the agent’s current orientation when it had only completed (to exaggerate for illustrative purposes) *half* of the observed player’s motion from edge A. Since relative primitives encode no self-corrective information and implicitly depend upon all previous primitives, these transition errors tended to accumulate as the agent traversed successive edges. The observable result of this phenomenon was that, while the agent’s modelling of the human would initially appear very accurate, after a period of time its orientation became increasingly incongruous with its direction of movement.

Absolute primitives, by contrast, are not vulnerable to this effect, since they encode the full yaw and pitch of the player on every timestep. However, since the state space of absolute actions is far larger than that of relative actions, primitives derived from absolute data tended to produce significantly coarser orientation adjustments than relative primitives, even at higher cluster densities; this had the effect of rendering the agent’s in-game motion substantially less humanlike. Given that one of the principal objectives of our work is to improve the believability of game agents, an alternative is clearly needed.

Our solution, which we call *imitative keyframing*, allows us to borrow the strengths of both the relative and absolute approaches. The central concept is analogous to *keyframing* in computer animation, whereby the artist specifies a set of start and end positions within an animation sequence, and the computer creates the smooth transitions between them - although naturally in our case, the inter-keyframe actions are derived from observation data rather than being generated artificially. More aptly, it is similar to the action primitive sequencing system proposed by Fod, Mataric and Jenkins [37], as discussed in Chapter 2. In that contribution, they derived movement primitives from human limb-motion data, and sequenced them by generating a list of potential successor candidates observed at the end of each discrete action; they then selected the primitive which would result in the least perturbation of the robot's limb trajectory between movements.

In our case, we record a set of *initial primitives* for each  $(E_{ct,ct+1}, c_g)$  pair in the recorded data - these are *absolute* primitives which the player was observed to execute upon first entering each new edge in each goal state. Depending on the topological complexity of the map, these can either be used directly or clustered to form a smaller set of primitives; a slight improvement in the smoothness of the agent's motion can also be achieved by recording the minimum and maximum observed perturbations for each edge-goal pair, which can later be used to *clamp* the bot's orientation on each edge. Aside from these initial actions, we record the *relative* changes in orientation on every timestep. The relative actions are clustered to derive a set of primitives and the probability matrices are constructed as above, with the addition of an extra matrix representing the valid *relative successor* primitives for each *initial absolute* primitive in each edge-goal state.

When the agent is deployed, the mechanism employs relative primitives and operates exactly as before while the bot is moving along the edges of the topological map. Upon reaching a node, however, a slight alteration is introduced. The agent consults the *initial primitives'* probability matrix, selects the highest-valued action, and

applies it. Then, it generates a list of all possible *relative* successor primitives, chooses the one which will minimise the change in orientation upon the next timestep, executes it, and reverts back to normal operation. Thus, the relative primitives are used when traversing the edges of the topological map, while the absolute primitives are used to *keyframe* a single timestep at each waypoint. In this way, the ability of the relative primitives to manifest fine-grain orientation adjustments is retained, while the absolute primitives ensure that no error accrued during traversal of a particular edge can propagate to the next.

#### 4.4.4 Experiments

Here, we present a statistical validation of the unified strategic and Bayesian tactical system, and examine the effect of different cluster densities upon the accuracy of the resulting imitation. A comprehensive evaluation of the mechanism as it pertains to the agent’s perceived *believability* or *humanness* is presented in Chapter 5.

To test the model, we utilized 35 recorded gameplay samples of varying length and complexity, spread across three distinct environments. In each case, the agent attempted to reproduce the human’s behaviour with cluster densities of 20%, 30% and 50% of the total number of samples, both for constructing the topological map and for deriving the action primitives. In each case, the gameplay samples included features which would require the Bayesian motion-modelling system to learn and reproduce certain *tactical actions* - jumping, interaction with environmental features such as elevating platforms, etc - in order to successfully complete the human’s traversal. Because, as mentioned in Section 4.3, the agent’s task was not to simply *copy* the human - but rather to deduce his objectives and pursue them independently - a frame-by-frame comparison of the bot’s actions against the demonstrator’s was not appropriate. Instead, we compute the error between the agent’s position and actions while traversing each edge and those of the human while moving along the same edge in the same goal state.

| Cluster density      | 20%   | 30%   | 50%   |
|----------------------|-------|-------|-------|
| Position MAE (Units) | 21.32 | 19.22 | 17.76 |
| Position MAPE (%)    | 2.69  | 2.42  | 2.32  |
| Yaw RMSE (°)         | 4.78  | 3.84  | 2.71  |
| Pitch RMSE (°)       | 0.30  | 0.17  | 0.11  |

**Table 4-2 - Variation of mean error with cluster density**

| Sample | Position MAPE (%) | Yaw RMSE (°) | Pitch RMSE (°) |
|--------|-------------------|--------------|----------------|
| 1      | 0.93              | 3.83         | 0.16           |
| 5      | 1.63              | 10.3         | 0.16           |
| 10     | 1.26              | 1.55         | 0.07           |
| 15     | 3.28              | 3.93         | 0.10           |
| 20     | 3.56              | 8.75         | 1.34           |
| 25     | 4.15              | 8.06         | 1.36           |
| 30     | 7.25              | 3.45         | 0.33           |
| 35     | 3.33              | 1.98         | 1.09           |

**Table 4-3 - Representative error rates for 20% cluster density**

| Sample | Position MAPE (%) | Yaw RMSE (°) | Pitch RMSE (°) |
|--------|-------------------|--------------|----------------|
| 1      | 0.71              | 1.52         | 0.10           |
| 5      | 0.91              | 0.20         | 0.16           |
| 10     | 0.98              | 1.63         | 0.03           |
| 15     | 2.88              | 2.29         | 0.07           |
| 20     | 2.23              | 7.23         | 0.09           |
| 25     | 3.21              | 4.85         | 0.09           |
| 30     | 5.83              | 2.06         | 0.13           |
| 35     | 2.11              | 0.71         | 0.09           |

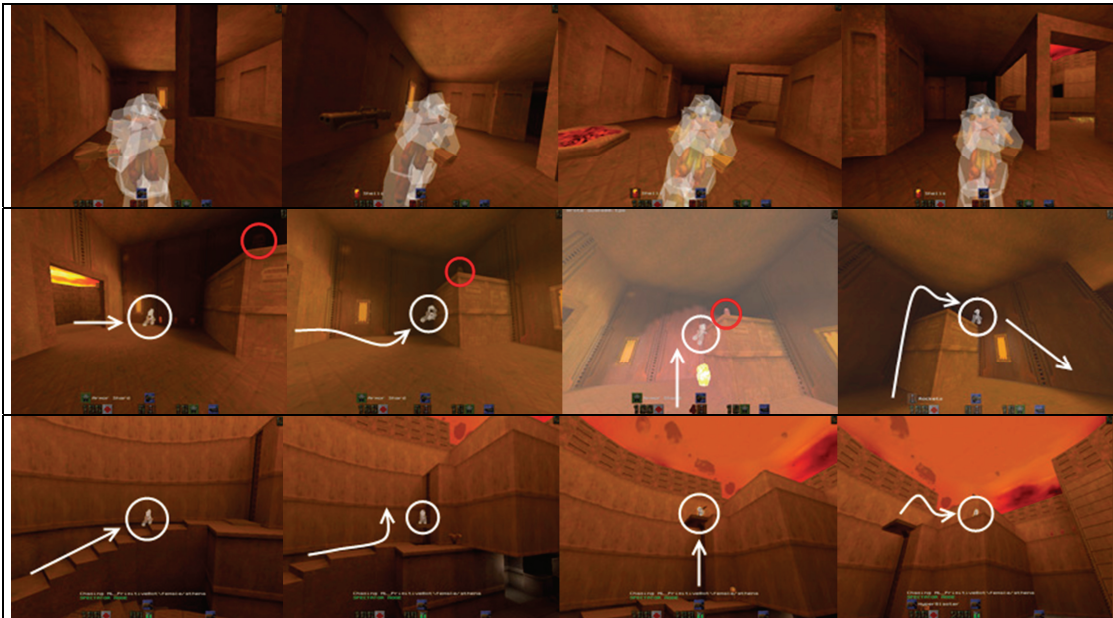
**Table 4-4 - Representative error rates for 50% cluster density**

The results are summarized above. As can be seen from Tables 4.2-4.4, the model produced a very accurate reproduction of the human player's behaviours, even at low cluster densities. The MAPE (*mean absolute percentage error*) between the agent's position and that of the human, calculated from the centroid of the dataset, did not exceed 7.25% at 20% density, and was generally much lower; the highest recorded



RMSE (*root-mean-square error*) in the agent's yaw was  $10.3^\circ$  at the lowest density, while its pitch stayed within  $1.36^\circ$  of the demonstrator's. The low pitch error reflects the fact that the agent was engaged in imitating the player's strategic *navigational* behaviours - variations in the horizontal plane were therefore greater (moving around corners, etc) than those in its vertical, which would be more pronounced in a combat scenario. The numerical results correlated with the very humanlike *visual* appearance of the imitation agents.

While the choice of cluster densities revealed itself to be highly dependent upon the complexity of the environment and the size of the gameplay sample, the results demonstrate an expected trend towards greater accuracy with more clusters. In certain cases, however, this did not hold true. We found this to be caused by the *overcrowding* of the topological map with waypoint nodes, which forced the agent to make significantly more course corrections than were necessary for accurate reproduction of the player's navigation; this, in turn, often had a negative impact on the accuracy of the action primitives. Given the relatively small error reduction that results from increasing the topological density versus the more significant error reduction when the number of primitives is increased (see Table 4-2), and considering that higher topological cluster densities are undesirable in any case since they increase the duration of the learning process, a useful guideline derived from these experiments is to use the minimum number of topological clusters required to provide a sufficiently detailed representation of the environment while maximizing the number of action primitives in accordance with any relevant resource considerations. In general, we found that cluster densities of less than approximately 20% were not sufficient to capture the topology of the environment, while densities of greater than 50% often resulted in a decrease in performance, both numerically and perceptually.



**Figure 4-15- Examples of the aesthetic (top) and functional (middle) aspects of the tactical motion-modelling system; the bottom sequence demonstrates a combination of both.**

Recall that, in Section 4.4.1, we discussed the role of the motion-modelling system in reproducing both the *aesthetic* qualities of the human player's motion - a vital component in the perceived *humanness* of the agent, as will be further detailed in Chapter 5 - and its *functional* elements, which comprise the *environment-relative tactical behaviours* necessary for the agent to negotiate its environment. Some examples of both the aesthetic and tactical aspects of the Bayesian system are shown in Figure 4-15 above.

The top sequence shows the agent *leaning into* and *strafing* (pivoting) around a corner, turning its view towards the corridor into which it is about to move. From the agent's perspective, such aesthetic motion is pointless; to an observer, however, it immediately conveys the impression of a human player, who needs to actually *look* towards his destination in order to ensure that (for instance) no opponents are lying in wait.

In the middle sequence, the agent's next goal is an item on top of the large box. As it approaches, it switches weapon to the rocket launcher, looks downwards, jumps, and

fires a rocket into the ground to propel itself upwards. This so-called *rocket jump* is considered one of the most advanced moves in Quake 2, and is commonly employed by experienced players to reach otherwise inaccessible areas.

The bottom sequence shows the agent interacting with a lift by stepping onto it, standing still as it ascends (functional) and then jumping off at the top, an unnecessary action which is nonetheless common among human players (aesthetic).

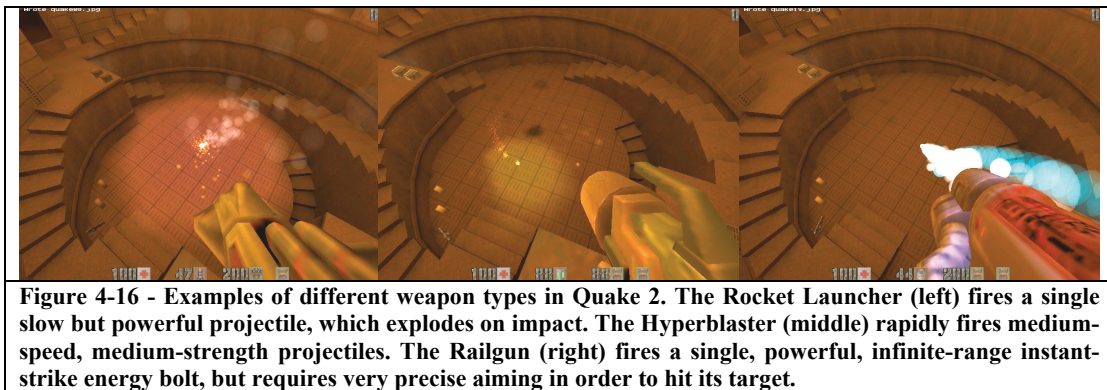
## 4.5 Reactive Behaviour Imitation

In this section, we describe our work in imitating *reactive* behaviours; these are actions which involve very little planning, which the human takes in response to stimuli in his or her immediate environment. In the context of Quake, reactive actions principally refer to the *aiming* behaviour of the agent while in combat with an opponent. We first briefly recap the problems with existing agent aiming behaviours, and some related work in this field, before describing our imitative approach. A series of experiments is then detailed to demonstrate its effectiveness in capturing humanlike weapon-manipulation behaviours.

### 4.5.1 Reactive Combat Behaviours

In Quake 2, as in all first-person shooter games, players employ a wide variety of different weaponry while engaging in combat against one another. Though the specific weapons vary from game to game, a number of common characteristics prevail; certain guns fire slow-moving but powerful projectiles, others fire quick but weak bursts. Some are ideally suited to ranged combat, whereas others are more useful in close quarters (Figure 4-16). Depending on the situation, the player may

need to “lead” his opponent - that is, deliberately aim *ahead* of the enemy - in order to give the projectile sufficient time to meet its target. Experienced human players utilise these weapons in such a way as to maximise their effectiveness, instantly weighing an array of variables (proximity to opponent, remaining ammunition, etc) before deciding upon their course of action. In comparison to this lucid play style, the performance of traditional rule-based artificial agents leaves much to be desired. Designed top-down, they exhibit at best a crude approximation of the human player’s context-sensitive weapon handling; often, they will simply retain whichever gun they happened to last pick up, or whichever the game designer has specified as being the most powerful. To compound this problem, developers often compensate for their agents’ lack of intelligence by endowing them with superhuman capabilities far beyond the constraints imposed upon the human player by his mouse-and-keyboard interface, enabling them to read the position of their human opponent from the gamestate and strike with pinpoint accuracy.



The very fact that such bots are consistently unpopular with gamers demonstrates that sacrificing believability to artificially increase the competitiveness of agents is an inherently misguided approach - even when losing to a superior human opponent, gamers invariably enjoy the experience more than fighting an unrealistic bot [140, 141]. Laird and Duchi [78] have previously investigated the influence of aiming skill upon Quake agents’ perceived humanness; they report that

bots which exhibit excessive accuracy elicited highly negative reactions from the judges in their experiments, and go so far as to suggest that constraining combat skills to realistic human levels be made a core criterion in FPS agent design. It should be noted, however, that even in cases where designers have taken measures to add an element of inaccuracy to rule-based agents' aiming behaviours, this generally consists of little more than adding a simple random offset from their opponent's true position. This can itself introduce artificialities, since the agents will tend to fire periodic volleys of shots in a halo around the player, resulting in predictable miss-miss-hit-hit-miss-type patterns.

### 4.5.2 Learning the Inaccuracy

With this in mind, we look to imitation learning to provide an alternative. Our previous contributions have primarily concentrated on the strategic and tactical layers; this work is situated on the boundary of the reactive (close-quarters combat with limited available time to plan) and tactical strata (long-range combat with slightly greater scope for weapon selection and aim planning).

Thureau and Bauckhage [11] present an initial approach to reactive combat imitation, using a mixture of 3 expert networks to aim and switch between three predefined weapons - one short range, one medium, one long - in a single plane. Counter-intuitively, their experiments revealed that the weapon competencies were not allotted bijectively to the networks, but rather produced a holistic representation distributed across all three experts; they speculated that the handling of different weapons does not differ as widely in the data space as the visual in-game result suggests. However, they report that - because the underlying "aim" function is quite straightforward, as embodied by the traditional bots discussed earlier - the network

learned to aim *too* well, resulting in precisely the kind of pinpoint accuracy we wish to avoid.

It is from these findings that we draw the intuition underpinning this work; rather than attempting to imitate the player's *aim*, we instead learn what distinguishes a human from a traditional game bot - his *inaccuracy*. As sometimes occurs in our work, this places us at odds with our counterparts in the field of robotics, who use imitation as a means of quickly attaining optimal performance; since we are primarily interested in producing realistically human behaviour, we often need to deliberately strive for the competent yet suboptimal. The human-produced "noise" which Dillman et al [29] regard as a drawback of imitation learning - *incorrect*, *unmotivated* and *unnecessary* actions - are precisely the behavioural traits we are attempting to capture here.

The inaccuracy of the player's aim derives from two distinct components:

- **unintentional inaccuracy**, due simply to human error
- **intentional inaccuracy**, the player's practice of accounting for his opponent's motion while aiming

Using this as our starting point, we attempt to learn context-sensitive weapon switching and aiming across the entire range of available weaponry, and in both the horizontal and vertical planes.

### 4.5.3 Methodology

#### 4.5.3.1 Data Extraction

To facilitate the learning process, we must first reconstruct the player's perception of his opponent's motion using the available low-level data. For each frame, we read the

player's current position, his orientation (i.e. the direction in which he is aiming), and the position of the nearest opponent. From this, we derive the opponent's current directional vector, and the vector running from the player's position to the enemy's - that is, the 'perfect' line along which the player *would* be aiming, were there no considerations of projectile speed or human error to take into account; the angular difference between this line and the player's observed aiming direction is his *inaccuracy*. We can now express the opponent's velocity relative to the player's field of vision, by computing the projection of his directional vector onto the vectors parallel and perpendicular to the player's view angle, as shown in Figure 4-17:

$$v_{par} = comp_{\vec{a}} \vec{c} = \frac{\vec{a} \cdot \vec{c}}{|\vec{a}|} \quad v_{perp} = comp_{\vec{b}} \vec{c} = \frac{\vec{b} \cdot \vec{c}}{|\vec{b}|}$$

where  $\vec{a}$  is the player's viewing vector,  $\vec{b}$  is perpendicular to the viewing vector, and  $\vec{c}$  is the direction of the opponent's motion. We also record the opponent's velocity in the vertical plane as the change in his Z-axis position between successive frames:

$$v_{vert} = z_t - z_{t-1}$$

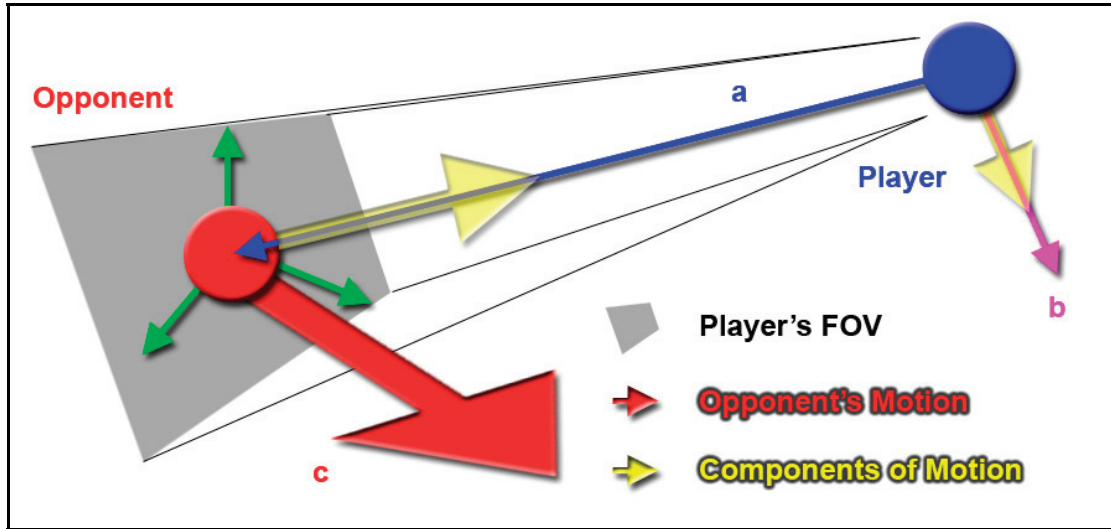


Figure 4-17 - Reconstruction of the player's visual perception from low-level data.

However, we must also consider the impact that *elevation* has on the player's perception of his opponent's motion. Consider the scenario where the opponent is directly above or below the player; in this case, any movement along the Z-axis (i.e. vertically towards or away from the player) will have no bearing upon his aim. If, on the other hand, the opponent is on the same horizontal plane as the player and begins to ascend or descend, the player will need to compensate by adjusting his pitch. Similarly, if the opponent is on the same horizontal plane as the player and is moving towards him, the parallel component of the opponent's movement will have little influence on the player's resulting aim; if, however, the opponent is above or below the player and moving along the parallel vector, then he will again need to adjust his pitch accordingly as the enemy approaches.

To account for this, we weight both the parallel and vertical velocities according to the angular elevation of the opponent relative to the player; that is

$$v_{par} = \sin(\phi) \left( \frac{\vec{a} \cdot \vec{c}}{|\vec{a}|} \right) \quad v_{vert} = \cos(\phi) (z_t - z_{t-1})$$

As an aside, it is worth mentioning that this is good example of the benefits of using computer games for imitation (and, indeed, general AI) research. In robotic imitation, data is commonly derived either through video analysis or by using expensive motion-sensing equipment; here, by contrast, we can build a noise-free reconstruction of the player's visual perception of the virtual environment, without having to perform any image processing of the game session's graphical representation. Rather than using our access to the gamestate data to *cheat*, as is the case with the traditional agents discussed earlier, we instead use it to build a model which provides our agent with the same information as was obtained visually by the human player during the game session.

In addition to the above, we must also read the following data on each timestep:



- the player's current weapon
- a “bitmask” indicating which weapons he currently possesses and has ammunition for
- the player's distance from the opponent
- whether or not the player is firing his weapon

Together, these will form the training set for the neural network architecture.

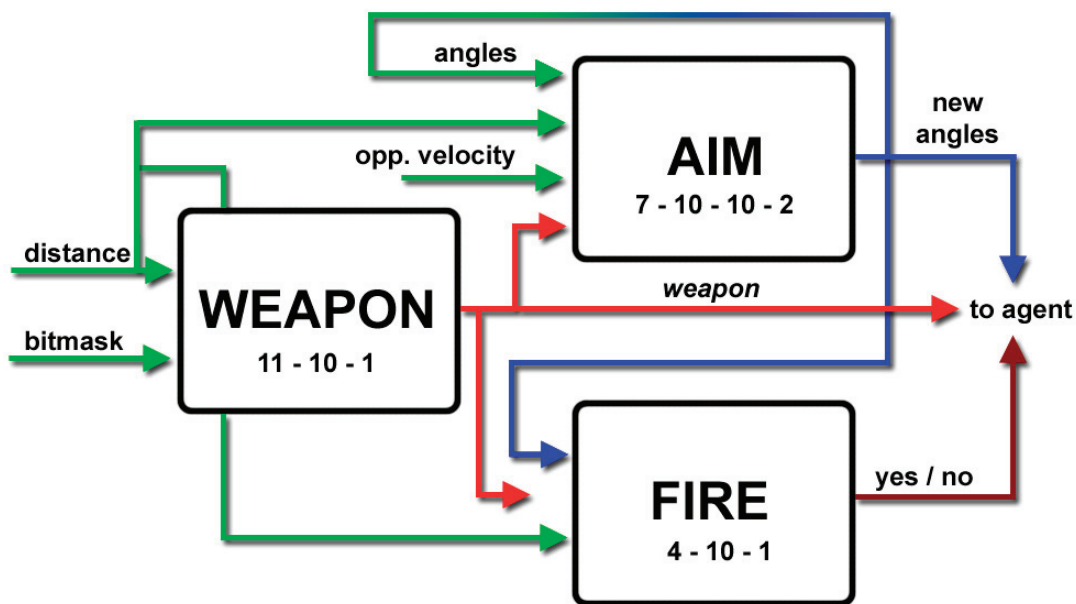


Figure 4-18 - A series of interconnected neural networks is employed to learn and reproduce the observed combat behaviours

#### 4.5.3.2 Network Architecture

The neural network architecture itself consists of three interconnected networks, each of which is trained to perform a specific function; the model's structure is outlined in Figure 4-18 above. Each network is trained as follows:

##### **WEAPON**

The Weapon network is responsible for selecting the most appropriate gun, given the available options. Its inputs consist of the distance between the player and opponent

on each timestep, together with a bitmask indicating weapon availability. The target output is the active weapon on the subsequent timestep.

## **FIRE**

The Fire network is responsible for determining whether or not the agent should fire its gun. Its inputs consist of the current weapon, the distance to the opponent, and the angular difference between the vector from player to opponent and the player's current aiming vector. Its outputs are 1 if the player should fire his gun, 0 otherwise.

## **AIM**

The most important component of the architecture, the Aim network is responsible for adjusting the aim of the agent in accordance with the opponent's manoeuvres and the attributes of the selected weapon. Its training data consists of the distance from player to opponent, the parallel, perpendicular and vertical velocities of the enemy, the current weapon, and the previous angular inaccuracy for some temporal context. In keeping with the approach discussed earlier, its target on each timestep is the angular difference between the vector running from the player to the opponent and the player's current aim vector (i.e. the inaccuracy of the player's aim).

When trained and deployed, the relevant feature vectors are obtained from the agent's gamestate on each timestep. The opponent's proximity and the weapon bitmask are presented to the Weapon network, producing the index of the gun to be used on the next timestep. This is in turn supplied both to the Aim network - thereby producing the pitch and yaw inaccuracy of the agent's aim on the next frame - and to the agent, for implementation on the next timestep. The Aim network supplies its angular output to the bot and also passes it to the Fire network, where together with the new weapon index and the distance between the player and enemy it determines whether or not the agent should commence firing its weapon.

## 4.5.4 Experiments

In testing our model, we wished to use data derived from a free-flowing game session, rather than it being generated under more controlled circumstances. To that end, we first built a custom-purpose Quake 2 environment, designed specifically to maximise the variation of distance, angle and weaponry experienced by a player in the course of a game session (see Figure 4-19). We then had two players of comparable experience play on this map for just over 60 minutes, resulting in a wealth of data with which to train the networks. For the purposes of these experiments, we use the victor's data as the *training* set, and the runner-up's as a *validation* or *test* set.

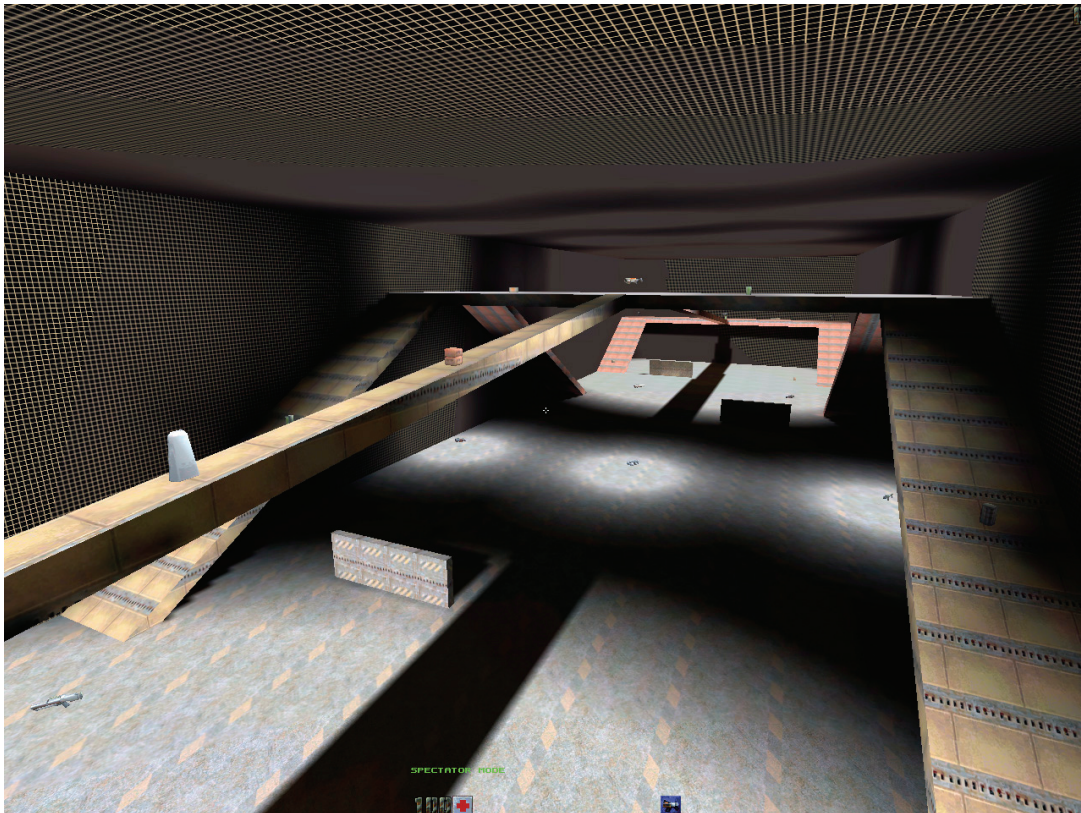


Figure 4-19 - The custom Quake 2 map used in our experiments

In the cases of both the Weapon and Aiming networks, we limit the dataset to those periods during which the player was actively engaged in combat. This approach was taken because, during non-combat phases, the player generally pursues *navigational* behaviours; changes in his orientation and field of view are thus intended to facilitate visual inspection of his surroundings, and are unrelated to aiming. This was obviously not appropriate for the Fire network - in its case, we extracted every available sample, whether the player was in combat or not.

| Network       | Training Set # | Test Set # |
|---------------|----------------|------------|
| <b>Aim</b>    | 10184          | 6008       |
| <b>Weapon</b> | 10184          | 6008       |
| <b>Fire</b>   | 37981          | 35678      |

**Table 4-5 - Sizes of training and test datasets**

We then used this data to train each network using a range of topologies, employing Levenberg-Marquardt backpropagation across 3000 iterations. Ultimately, we found that the topologies shown in Figure 4-18 worked best - some insight may be gained by noting that the tests considered 10 distinct weapons. Very little improvement in performance was observed by increasing the number of neurons or layers in the case of the Fire and Weapon networks; we therefore opted for a 10-neuron single hidden layer configuration. We also noted that the evaluation set error was in some cases not as close to the training set error as had been expected; this was not mirrored in the in-game trials, where the networks proved quite capable of generalising to unseen circumstances. On reflection, however, this seems reasonable - while the general rules of weapon handling will apply to all competent players, the individual quirks of two distinct human gamers will always lead to diversities of this kind. The effect was likely amplified by the fact that the victorious human player, from whom the training data was drawn, had quickly come to dominate the match, leaving his opponent with scarce opportunity to retaliate.

#### 4.5.4.1 RESULTS

The agent was observed to switch between different weapons dependent upon its proximity to the opponent, mirroring not only the concepts outlined above, but also the human exemplar's specific weapon preferences. For instance, in close quarters combat, one might justifiably choose among the Blaster, Hyperblaster, Shotgun or Chaingun; the agent exhibited a preference for the Chaingun, switching to one of the others when it ran out of ammunition. When the opponent moved to medium range, the agent opted to use the Hyperblaster or Chaingun; at medium-to-long ranges, the Railgun or Rocketlauncher was preferred. Table 4-6 shows the mean-square error for each network/topology combination for both the training and test sets.

| <i>Network</i> | <i>Topology</i> | <i>MSETrain</i> | <i>MSETest</i> |
|----------------|-----------------|-----------------|----------------|
| <b>Aim</b>     | 7-5-2           | 0.0065          | 0.0102         |
|                | 7-10-2          | 0.0056          | 0.0089         |
|                | 7-10-10-2       | 0.0038          | 0.0054         |
| <b>Weapon</b>  | 11-5-1          | 0.0287          | 0.0483         |
|                | 11-10-1         | 0.0272          | 0.0416         |
|                | 11-10-10-1      | 0.0261          | 0.0455         |
| <b>Fire</b>    | 4-5-1           | 0.1333          | 0.1910         |
|                | 4-10-1          | 0.1292          | 0.1881         |
|                | 4-10-10-1       | 0.1254          | 0.1844         |

**Table 4-6 - Training phase results**

Beyond this, a number of interesting phenomena were observed. When in close-to-medium range combat, where the player perceives the opponent as moving very fast across his field of vision, the agent exhibited only limited “leading” behaviour (that is, aiming ahead of its opponent to account for projectile speed). We believe this to be due partly to inadequate reaction time caused by the opponent's proximity, and partly to the constraints imposed by the human's interface - the player is physically limited, by the range of motion possible using a hand-driven mouse, in the speed and

accuracy with which he can turn. When the opponent moved further away, however, the leading behaviour became more pronounced and deliberate. Thus it appears that the weapon-handling behaviour encompasses both *reactive* and elements of *tactical* behaviours from the Hollnagel model discussed earlier; the practice of leading the opponent becomes more prominent as the amount of time available to the player increases. As discussed at the start of this section, we expected to see this effect to some degree at the outset of the experiment.

It was observed, during some trials, that the Weapon network tended to switch between multiple guns in quick succession instead of simply picking one which was range-appropriate. This was due to the human exemplar’s practice of rapidly cycle through his inventory with the mouse wheel until he reached his desired gun, rather than choosing it directly; a practice which introduced considerable *noise* into the weapon-selection data. While serving as a good example of how imitation learning automatically captures elements of human behaviour which would ordinarily be overlooked by an AI designer, this rapid weapon-switching was in some cases observed to impact negatively upon the agent’s ability to aim. We therefore propose an alternative approach. Given that there is a relatively discrete logical partitioning of guns with varying range, we can cluster the distances between the player and opponent observed during combat, and select probabilistically. At timestep  $t$ , the weapon is chosen according to:

$$w_t = \arg \max_k \frac{P(w_k | d_t) b_k}{\sum_u P(w_u | d_t) b_u}$$

where  $w$  is a weapon,  $d_t$  is the clustered distance to the opponent at timestep  $t$ , and  $b_k$ , the bitmask element corresponding to  $w_k$ , is 1 if the agent possesses weapon  $w_k$  and has ammo for it, 0 otherwise. This was observed to produce more decisive weapon selection behaviour. Choosing whether to adopt the probabilistic or neural approaches thus becomes a matter of priorities; between replicating a minor but compelling aspect of common human behaviour, or ensuring a more rigid but

perhaps more consistent mechanism. Naturally, if the training data is drawn from a player who selects his weapons directly rather than by spinning the mouse wheel, this issue becomes moot.

One of the most interesting (and unanticipated) observations was that, in close combat with the opponent rapidly moving left and right around the agent, it seemed to be somewhat more capable of tracking the enemy as it moved anti-clockwise around the agent than as it moved clockwise. At first, we naturally assumed that the network had simply learned to reproduce one motion more accurately than the other; however, the same phenomenon was observed when the network was re-trained with differing topologies, and even when the evaluation dataset from the second human player was employed as the primary training set. Having reviewed the recordings of both the original game session and the agent's reproduction, we suspect that this is due to the fact that both players were *right-handed*; their range of motion while sweeping the mouse inward - that is, the movement required to track an opponent anti-clockwise in close proximity - is therefore slightly greater, due to it being a more ergonomically comfortable motion than an outward sweep. While reproducing this was not an intended consequence of our imitative model, it once again serves to aptly illustrate the benefits of imitation learning in producing humanlike behaviour; it implicitly captures and manifests elements of the human gameplay experience - in this case, a bias introduced by the player's physical interface - which would be entirely absent in a rule-based agent designed from the top down.

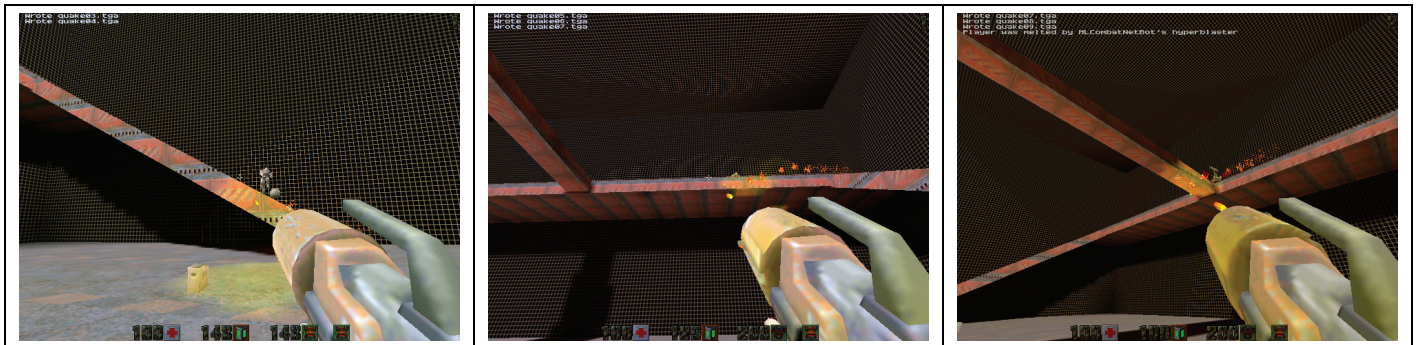


Figure 4-20 - An example of the agent "leading" an opponent. The bot aims ahead of its enemy, such that the Hyperblaster's projectiles coincide with his arrival. The agent's aim tracks the opponent up the ramp, ultimately defeating him and scoring a point.

## 4.6 Conclusion

In this chapter, we described the models and mechanisms which we have developed to imitate human behaviours from recorded Quake 2 game sessions. A hierarchy - or more accurately, a *gradient* - of observable in-game actions was first presented, based on Hollnagel's contextual control model of real-world human behaviour; this identifies *strategic* long-term planning behaviours, *tactical* mid-term planning behaviours, and *reactive* short-term stimulus-response behaviours. We further identified, based on previous work by Laird/Duchi and Livingstone/McGlinchey, a number of core criteria for agent believability.

Building upon this theoretical foundation, we proceeded to outline imitative models for each major level along the behaviour gradient, noting instances wherein some overlap between the different layers existed; in each case, we also described experiments which served to statistically validate the proposed systems. We detailed a reinforcement-learning approach to long-term *strategic* imitation and navigation, including such features as the human's ability to account for dynamic changes in the local and global environment, the weighting and pursuit of multiple objectives in parallel, and the derivation of a goal-oriented topology of the game environment from recorded data. We then outlined the integration of this system with an approach to *tactical* motion-modelling based on the theory of action primitives; this allowed the agent to imitate a human player's characteristically smooth motion, in addition to such behaviours as environment-relative weapon selection and complex medium-term actions such as Quake's signature *rocket jump*. Finally, we presented an approach to the imitation of *reactive* weapon-selection and combat behaviours, which involved first reconstructing the human player's visual perception of his opponent's motion from the available low-level data, and then emulating the inaccuracy - both intentional and unintentional - demonstrated in his aiming behaviours with varying weapon, angle, elevation and range.



# 5 Believability Testing

## 5.1 Introduction

One significant impediment to work in this field - indeed, in the fields of game AI and virtual human agents in general - is the current lack of a formal, rigorous standard for determining how ‘humanlike’ an artificial agent is, or any strict means of comparing the believability of different agents. In this chapter, we propose such a system - the **Bot-Oriented Turing Test (BOTT)** - and use our own imitation agents to demonstrate its effectiveness.

In Section 5.2, we detail the motivation underlying this work; the reasons for the lack of a standardised approach to believability testing, the applicability of the Turing test to the problem, the informal nature of previous attempts at such analysis, and the necessity that this problem be addressed.

In Sections 5.3 and 5.4, we state the specific objectives of what we refer to as our *believability-test framework*. We then detail each component of this framework; the construction of an anonymous online survey to profile the experience level of participants and collate their judgements of the agents’ believability in comparison to human players and other agents, the development of an experimental procedure or *protocol* to remove subjectivity and bias, and the computation of a *believability index* to numerically express the artificial players’ “humanness” as a function of the respondents’ experience and the accuracy with which they identified the agents.

In Section 5.5, we outline an experiment conducted using the believability testing framework, which employed our own imitation agents as a test case, human players as a control group, and a traditional AI bot as a comparator. This has the dual purpose

of demonstrating the framework itself in practice, and of assessing the believability of our imitative mechanism as described in the preceding chapters. We find that our agents were judged to be significantly more humanlike than the traditional agents, and conclude that this provides strong evidence in favour of our thesis that imitation learning has the potential to produce more believable AI agents than traditional AI techniques alone.

Some further observations on the test, and its applicability to the wider question of believability assessment, conclude this chapter.

### 5.1.1 Publications

The work in this chapter was published as “*Believability Testing and Bayesian Imitation in Interactive Computer Games*” (Gorman, B., Thureau, C., Bauckhage, C., and Humphrys, M.) in Proc. 9th Int. Conf. on the Simulation of Adaptive Behavior (SAB'06), volume LNAI Springer, 2006.

It was also published as part of a discussion on the role of imitation learning in games from a cognitive perspective, as “*Learning Human Behavior from Analyzing Activities in Virtual Environments*” (Bauckhage, Gorman et al 2007), MMI-Interaktiv Journal, Nr. 12, April 2007.

## 5.2 Motivation

As discussed in the preceding chapters, one of the primary aims of our work in imitation learning is to improve the *believability* of artificial agents - that is, the degree to which they are perceived by observers as exhibiting humanlike traits, or indeed of *being* human. The question thus naturally arises: how do we objectively determine this “degree of believability”? One significant impediment to work in this

field is the current lack of a formal, rigorous standard for determining how ‘humanlike’ an artificial agent is, or any strict means of comparing the believability of different agents. MacNamee, in noting that the question of evaluation has been largely overlooked in virtual-human and game AI research, posits that part of the reason lies in the fact that such sophisticated agents are a relatively *new* achievement; and thus, that models and implementations have necessarily occupied researchers more than metrics and evaluation. Systems which achieved a degree of believability were inherently novel, and therefore a success in their own right. As he concludes, however, the field is now well established, and the current situation is no longer tenable [80].

MacNamee further acknowledges, in his discussion of believability testing for virtual human simulations, some of the difficulties involved; namely, that “believability” is an inherently subjective concept, and that asking an individual to simply rate the humanness of an agent’s behaviour in isolation would invariably cause the results to be overtly influenced by the judges’ familiarity, or lack of same, with the field of artificial intelligence. He raises the possibility of using implicit measures, such as the agent’s ability to perform a given task or series of tasks, as a means of eliminating subjectivity; this functional or *numerical* approach is echoed by Wallace & Laird [136], who propose that agent performance be measured by representing behaviours as hierarchies of temporally-constrained sub-objectives, and measuring the degree to which the sequence of behaviour states exhibited by the agent is consistent with that of the human. We have already presented similar numerical investigations of our imitation mechanism’s various components in the preceding chapters. While this is an effective and necessary means of verifying its functionality, and while casual examination has indicated its ability to reproduce recognisable human behaviour, a formal assessment and quantification of the system’s observable in-game believability is also required. MacNamee ultimately concurs with this view,

remarking that the only credible method by which believability may be ascertained is by conducting a survey.

### 5.2.1 Previous Contributions

Some existing contributions have attempted to address the problems of evaluating believability in game AI. While we consistently found the testing procedures employed in these experiments to be highly informal and ad-hoc, they do offer some valuable insights.

MacNamee's aforementioned work [80] concerns NPC (Non-Player Character) agents - that is, characters which populate a virtual world but which are not controlled by humans. In contrast to the *opponent* AI with which our own work is concerned, his simulation therefore involves no human interaction, instead visualising high-level, universally familiar real-world behaviours; in this case, a group of agents socialising at a bar. As there is no human exemplar against which to compare, and not wishing to evaluate the agents in isolation for the reasons outlined earlier, MacNamee adopts the approach of presenting the judges with two *separate* artificial simulations and asking which exhibits the *more humanlike* behaviour. From the perspectives of both our own work and that of opponent AI in general, this is clearly not appropriate. Firstly, we are attempting to imitate the strategic and tactical behaviours observed during a human player's interaction with the game world; we must therefore incorporate the data recorded from said humans as the *control group* in our survey. Secondly, the simulation against which MacNamee's agents were evaluated simply executed *random* behaviours at each timestep. While this may be suitable for determining whether one's agents act in a comparatively goal-driven manner in an autonomous simulation with no human interaction, a first-person shooter opponent bot which executed actions at random would be of little use for the purpose of comparison against our own agents. Thirdly, the test is presented as a

binary choice; either one or other of the observed simulations must be chosen as more believable. Although MacNamee posits some follow-up questions, this approach inherently limits the scope of the analysis that may be carried out upon the survey results. The test as a whole, while appropriate to the specific context in which it was employed, is thus quite informal, does not produce a result that can be easily translated for comparison against other agents, and does not represent a standard model that could be applied in more diverse scenarios.

Elsewhere, McGlinchey and Livingstone [111] outline a series of believability experiments conducted using the game Pong, wherein opposing players each control a paddle at opposite ends of the game field, and attempt to prevent a moving ball from exiting the screen on their side by bouncing it back towards their opponent. While the paper's main concern is to investigate what general elements of behaviour contribute most to perceived "humanness" rather than to design a rigorous believability test, it does provide some relevant observations. In their tests, a Self-Organising Map is used to train an artificial player from human demonstration. A rule-based AI is also used for comparison purposes; this operates by projecting the location at which the ball will arrive, and moving to this position. In similar fashion to the aiming mechanisms of traditional FPS agents discussed in previous chapter, this means that the artificial player can essentially be perfect; a degree of fallibility was manually added by inserting a delay before the paddle starts moving towards the ball, reducing the speed of the paddle's motion, and using a random  $\pm 20$  pixel offset from the known endpoint of the ball's trajectory. Eight games, involving different combinations of the human, hardcoded AI and imitation AI were then shown to a number of judges, who were asked to indicate whether they thought each paddle was being controlled by a human or artificial player. Although the paper does not discuss why such combinations were used, it is a shrewd approach which - for reasons detailed in Section 5.3.2 - we also used in our own believability tests. The results indicated that most of the judges' identifications varied significantly from chance, but

not always in the desired manner; one respondent, for instance, correctly identified 14/16 players, while another misidentified 14/16. McGlinchey speculates that this indicates a measurable distinction between the human and AI players. However, it is likely that the results were influenced by the fact that Pong is, as discussed in section 2.3, an extremely simplistic game; players can move up, or move down, or they can vary their speed somewhat. Within such a limited repertoire of behaviours, there exists little opportunity for a human player to distinguish itself from an artificial one. The believability test itself is highly informal - once again, it employs binary identification, with only some qualifying questions asked afterwards; no overall results are given, nor in-depth analysis performed. This is no doubt due to the paper's focus on, as its title indicates, "What Believability Testing *Can Tell Us*." Ultimately, though, it seems clear that the accuracy and reliability of this endeavour would be best served by a more rigorous, structured testing process, whose results could be utilised by other researchers examining the same questions. Indeed, the results reported by McGlinchey seem to support this; the respondents who correctly identified most players, *and* those who misidentified most, gave the *same reasons* for their choices.

Laird and Duchi [78] also describe some believability experiments conducted using their Quake 2 Soarbot; again, they approach the issue from the perspective of determining which behaviours they should implement in order to maximise the believability of rule-based agents. Their approach was to define a set of parameters - decision time, tactical complexity and aiming skill - which they expected to have a noticeable impact upon the agent's perceived humanness; for each of these, they instantiate agents at different values of the relevant parameter, and have them play against a human expert. Five human players of differing skill then play against the same expert. Laird notes, from their experience during previous investigations, that in a typical game session the human player and the agent may only be in contact for brief periods at a time, and consequently - in the context of a believability study - that

the observer may not have the opportunity to examine the agents in sufficiently fine detail; he therefore uses video clips of the recorded sessions, which show the game world from the human or artificial player's point of view. We use the same approach in our own believability experiments - it is well-suited to our purposes, particularly since our tests involve the strategic and tactical motion modelling systems rather than combat scenarios. Laird's results showed that the judges found agents with medium reaction times and complex tactical behaviours the most convincing, while bots with high aiming skill were judged to be more artificial. The analyses presented in Laird and Duchi's work are more detailed than in the previous cited examples, and convincingly demonstrate the effects of varying rule-based agent competencies on their perceived believability. However, some serious deficiencies persist in the testing procedure. Firstly, it is not clear whether the experimenters were present during the judges' observation of the gameplay samples, but the fact that the matches were recorded and played back on *videotape* suggest that they may have been; this raises the possibility that experimenter bias may have influenced the results. Secondly, on the subject of the videotaped recordings, it is not clear why Laird and Duchi opted to take this approach, since facilities exist to generate digital AVI videos directly from recorded DM2 files. While this may have had some impact upon the quality of the videos presented to the judges, the more serious consequence is that any extraneous, non-behaviour-related indicators which were present during the recording process - for instance, different players may have used different in-game character models, screen layouts, etc - were retained and seen by the judges. This has the effect of giving the judges "clues" as to which samples come from a common source, and could lead to them making decisions based on logical deduction rather than observation. Even in cases where their deductions are *wrong*, this will have an unavoidably deleterious effect upon the accuracy of the results. The importance of removing all such indicators is detailed in Section 5.3.2; in our tests, we run a script over all the gameplay samples to homogenise them to a common presentation format,

before generating AVIs directly from the DM2 recordings. Thirdly, the evaluation procedure itself remains highly ad-hoc. The judges viewed each gameplay sample individually; they were then asked to rate its humanness on a scale from 1-10, and to make a binary decision as to whether it was a human or artificial player. As detailed earlier, asking individuals to make a snap-judgement on abstract concepts such as “humanness” in isolation, without providing any basis for comparison, has the effect of greatly increasing the amount of guesswork involved - it is not a plausible way to judge the relative believability of different agents. Finally, out of the eight judges, only three viewed all the gameplay samples - the remainder viewed only a subset of them. This is insupportable for obvious reasons, not the least of which is that it introduces the possibility of selection and ordering bias (see Section 5.3.2). Under our proposed protocol, incomplete responses are discarded before the analysis process, while the selection and ordering of clips is randomised.

Throughout this thesis, we have stressed the numerous ways in which imitation learning holds obvious potential as a means of producing more believable agents; but, as can be seen from the examples above, there has thus far been no credible means of *rigorously assessing* this believability - the need for a perception- and behaviour-based analogue to the Turing test is clear. But which elements of, and to what degree, is the Turing test an appropriate tool in evaluating our imitation system?

### 5.2.2 The Turing Test and Game AI

It is perhaps of some historical interest, from the perspective of our work, that the original Turing test was itself conceived as an *imitation game* - the computer was given the task of equalling the ability of a human male to convince a judge, through typewritten interrogation, that he was a woman [133]. Livingstone [79] notes that Turing specifically discounted considerations of *how* the machine intelligence operated, realising that this would make the test susceptible to the other-minds



problem - that the only way to definitively *prove* a machine is thinking, is to be that machine and be aware of one's own thoughts. He therefore focussed his attention on machines which could *demonstrate* what would commonly be considered intelligence. For Turing, if a machine were capable of emulating the conversational and imitative abilities of a human, then the judge would have no good reason to ascribe intelligence to one and not the other, apart from an arbitrary bias arising from the biological nature of the human participant versus the artificial provenance of the machine.

As Livingstone further notes, this has come to be seen as an unsatisfactory attitude among many researchers, who are nowadays more concerned with investigating and building models to replicate intelligence from biological example, or with constructing robots capable of performing tasks involving real-world manipulation (see Chapter 2). This has the advantage of dividing the problem of human-level AI into constituent subgoals as opposed to a single pass-or-fail test, allowing progress in multiple areas to be made in parallel. Here, we find that each of the two primary facets of our work reflects one of these opposing viewpoints. We have ourselves deconstructed the multiplexed behaviours encoded in game sessions into a series of subobjectives, as per the Hollnagel-derived behaviour model described in Section 4.2; we have also used biologically-inspired models as the basis of our imitative mechanisms. Furthermore, as discussed in Section 2.3, many modern commercial games - including Quake 2 - represent an abstraction of the real world, and techniques developed to imitate in-game human behaviours may therefore be adapted for real-world applications (see Section 6.4.2 for more on this topic). Thus, from the perspective of *computer games as a platform for imitation-learning research*, we have more in common with the post-Turing ethos. However, from the perspective of believability, or *imitation learning as a means of producing better computer game AI*, Turing's position remains relevant; if a player is engaged in a game session and observes an opponent behaving in a manner that is typically associated with human

players, he has no reason to believe it is anything other than human<sup>1</sup>. The observer's perception is the only important criterion. Thus, Turing's approach is still appropriate for the evaluation of our agents.

But we must still consider that the Turing test is concerned with natural language, whereas we are dealing with navigational and tactical behaviours in a virtual environment. How, specifically, does the test relate to our work? In a contribution which explores the logical implications of Turing's paper, Harnad [49] notes a number of unresolved issues with the original test. He first distinguishes between *functional* and *structural* indistinguishability, and argues that Turing only accounts for the former, while disregarding the latter. Although Turing readily concedes and addresses this in his paper, treating the physical form in which the artificial mind is embodied as being irrelevant to the question of its intelligence, Harnad argues that discarding structural indistinguishability - that is, the correspondence between the physical bodies of the human and artificial participants - has the effect of excluding other, potentially salient aspects of functionality along with it. In the case of communication, the typewritten interrogation format removes tone, inflection, body language and so forth from consideration, thereby reducing the range of behaviours upon which the interrogator may base his judgement. Consequently, Harnad proposes a *Turing hierarchy* of benchmarks, with each successive level imposing increasingly stringent requirements upon the candidate machine:

- **t1: subtotal fragments of human functionality.** Any model which seeks to emulate an arbitrary component of full human capacity; these may represent constituent modules in an eventual human-level AI.
- **T2: total symbolic function.** The machine can perform any symbol-manipulation task of which a human is capable. The original Turing test

---

<sup>1</sup> It is worth observing that, in Turing's original paper, the issue of whether the judge is *aware* that one of the participants might be an artificial intelligence is ambiguous. In this regard, Turing's imitation game is quite similar to a typical Quake 2 match.

resides at this level, though as Harnad points out, it is susceptible to Searle's Chinese Room counterexample [116].

- **T3: total external sensorimotor (robotic) function.** The machine can perform any real-world task of which a human is capable. Harnad posits this as the most reasonable ultimate goal for AI research.
- **T4: total internal microfunction.** Though still composed of synthetic materials, T4 level robots exhibit all the microfunctions of human bodies; pupillary dilation, blushing, bleeding, and so forth.
- **T5: total empirical indistinguishability.** At this level, there exists no test which can distinguish the robot from a human; they are bio-engineered at the molecular level. Harnad proposes T4 and T5 for the sake of completeness, but states that he regards them as being unnecessarily overdefined - a machine which passes T3 can be assumed to possess a mind.

Our imitation-learning mechanisms reside on level  $t1$  in this hierarchy; they are behaviour modules implementing subtotal fragments of human functionality. This is, however, little cause for alarm - as Harnad explains, *every* model across *every* scientific discipline is currently at varying stages of their equivalent  $t1$  levels, and seem likely to remain so for the foreseeable future (Livingstone [79] points to the fact that recent winners of the annual Loebner Turing contest have all been deemed either 'probably a computer' or 'definitely a computer'; in the case of the 2000 contest, the judges' predictions were 91% accurate after five minutes). The advantage of the Harnad  $t1$  classification is that it represents a generic abstraction of the Turing test, releasing us from the constraints of natural language conversation and providing us with a formal basis upon which to conduct the evaluation of our agents. Models representing any component(s) of the full range of human functionality can be implemented for analysis; in our case, these are strategic planning/navigation and tactical motion modelling behaviours.

## 5.3 BOTT: The Bot-Oriented Turing Test

To summarise the discussion thus far: due in part to the novelty of sophisticated autonomous virtual human agents until recent times, and in part to the reluctance of the AI community to address a difficult subjectivity problem, there exists no rigorous method of gauging the ‘believability’ of game bots, nor of objectively comparing this quality in different agents. Given that one of the central aims of our work lies in improving the believability of these bots, and that the lack of such a methodology is a major impediment to the orderly evaluation of artificial game agents in general, this is clearly a shortcoming which needs to be addressed. The only feasible means of determining the degree to which agents are perceived as human is to conduct a survey; this, of course, immediately raises questions of subjectivity, experimenter influence, and so on. In order to produce a credible assessment of agent believability, any proposed system must be designed with these concerns in mind.

Our aims, then, are as follows:

- i) to construct a *framework* which facilitates rigorous, objective testing of the degree to which game agents are perceived as human; this framework will consist of an anonymous web-based *survey* and a *protocol* to which testers should adhere, in order to eliminate potential subjectivity and bias
- ii) to formulate a *believability index* expressing this ‘humanness’ as a function of the user’s *experience* and the accuracy with which the agents/humans are identified, and which facilitates the comparison of different agents.

The system developed to fulfil these criteria, which we call the **Bot-Oriented Turing Test (BOTT)**, is described below. We outline the structure of the survey and its applicability to the testing of agents in general, using our own experiments to illustrate key concepts; we then describe these experiments and their results in detail.

### 5.3.1 Online Survey Structure

To counteract any potential observer bias, the test takes the form of an anonymous online survey. Respondents are first presented with a detailed page of instructions covering all aspects of the test (Figure 5-1, page 187). Before starting, they are further required to estimate their experience in first-person shooter games, at one of five different levels. Subjective judgements are avoided by explicitly qualifying each experience level:

1. Never played, rarely or never seen
2. Some passing familiarity (played / seen infrequently)
3. Played occasionally (monthly / every few months)
4. Played regularly (weekly)
5. Played frequently (daily)

Respondents are asked to specify the option which best expresses their level of experience *at any point in the past*; that is, if (s)he played a FPS game daily at one time but no longer does so, (s)he should nonetheless select the ‘*Played frequently*’ option. This will later be used to compute the weighted believability index.

Upon proceeding to the test itself, the respondent is presented with a series of pages, each of which contains a group of video clips (Figure 5-2, page 188). Each group shows similar, but not identical, sequences of gameplay from the perspective of the in-game character. This approach was adopted to allay concerns that asking respondents to view individual clips in isolation, with no basis for comparison against similar samples, would lead to a significant amount of subjectivity and guesswork, thereby rendering the test essentially meaningless. Generally, three categories of clip should be employed; clips of an actual human player, which will

serve as the survey's *control group*, clips of the artificial agent under investigation, and clips of a second *comparator* artificial agent, which will act as a measure of baseline agent believability.

Within each group, the clips may depict any combination of human and artificial players. In other words, for each page, any of the following may be true:

- the clips may depict two human players and an agent in any order
- the clips may depict two agents and a human player in any order
- all three clips may be of human players
- all three clips may be of artificial players

The respondent is required to examine the appearance and behaviour of the character in each clip, and indicate whether (s)he believes it is a human or artificial player. The clips are rated on a gradient, as follows:

|       |                |            |                     |            |
|-------|----------------|------------|---------------------|------------|
| 1     | 2              | 3          | 4                   | 5          |
| Human | Probably Human | Don't Know | Probably Artificial | Artificial |

This rating is the central concept of the survey, and will later be used to compute the believability index. The respondent is also asked to specify how many times (s)he viewed the clip, and to provide an optional comment explaining his/her choice; the latter is used in cases where some aspect of the clip strikes the respondent as particularly noteworthy. In cases where (s)he indicates that (s)he believes the agent to be artificial, (s)he will be further asked to rate how "humanlike" (s)he perceives its behaviour to be, on a scale of 1 to 10. This more subjective rating is not involved in the computation of the believability index, but may be used to provide additional insight into users' opinions of different agents in cases where they were correctly identified as such. Having completed all required sections on each page, the user submits his/her answers and moves on to the next.

## AI & Games: Believability Study

Bernard Gorman, Christian Thureau, Christian Rauckhage, Mark Humphrys

Welcome, and thanks for taking part in this online survey! Please read the following instructions carefully before proceeding.

First, a few technical notes. Please ensure that cookies are enabled before proceeding, and that the [Macromedia Flash Player 7 or later](#) is installed to view the videos; if you are able to watch the clip on the right, your browser is correctly configured. For optimum viewing, a maximised window with a resolution of at least 1024x768 is recommended. Please allow a few seconds for each clip to load in, since playing the video before it is fully buffered will cause it to fast-forward through to the end - if this happens, simply return to the start and click the play button again. Finally, do not use the browser's *[Back]* button to revisit pages once you have passed them; the order in which the tests are presented is randomly-generated, and this may invalidate your session.

The purpose of this survey is to determine the believability of artificial agents in computer games - in this case, iD Software's *Quake 2* - and the degree to which it is possible to distinguish between artificial and human players. The structure of the survey is simple. On each of the following pages, you will see three video clips. These clips will show similar, but not necessarily identical, sequences of movement. Each group of three clips may depict any combination of human and artificial players; in other words, for each page, any of the following may be true:

- the clips may depict two human players and an agent in any order
- the clips may depict two agents and a human player in any order
- all three clips may be of human players
- all three clips may be of artificial players

For each clip, you will be asked to examine the movement and behaviour of the character, and indicate whether you believe it is a human or artificial player. Combat is, at present, omitted from the considerations of the study; your assessment should take into account factors such as tactical movement and pursuit of items, idiosyncratic motion, performance of necessary actions, etc. Each clip is marked on a gradient, as follows:

☐ Human

☐ Probably Human

☐ Don't Know

☐ Probably Artificial

☐ Artificial

In each case, you will also be asked to specify how many times you watched the clip (to a maximum of 3 times), and to provide an optional comment explaining your choice; the latter should be used in cases where some aspect of the clip strikes you as particularly noteworthy. If you indicate that you think the agent is artificial, you will be further asked to rate how "humanlike" you perceive its behaviour to be, on a scale of 1 to 10. Having completed all required sections on each page, you should click the *[Submit and Continue]* button to proceed to the next. Once you have completed the test, you will given the overall percentage accuracy with which you identified the clips, together with a unique session identifier.

As you submit each page, a cookie will be auto-saved to your computer containing details of your current progress through the survey. If, at any point, you wish to suspend your viewing of the tests and come back to them later, click the *[Save and Continue Later]* button at the bottom of the test page; the session can be resumed by returning to this intro page and clicking the *[Load Last Session]* button below. The cookie will remain valid for a period of three days, allowing you to pause and resume the questionnaire at will. The auto-save feature also serves to guard against any potential technical problems - if you encounter an error, simply return here and click *[Load Last Session]* to recover your previous state from the cookie. If you attempt to load a session when one has not been saved, you will be redirected to the top of this page. Once the tests have been completed, the cookie stored on your computer is destroyed.

Please bear the following guidelines in mind when assessing the clips:

- be as definitive as possible in your answers:** do not select either of the *[Probably]* options simply as a means of 'hedging your bets'. Only use these options in cases where you suspect that the agent is human or artificial, but with a *substantial degree of doubt*.
- at the other extreme, **avoid outright guesswork**, do not select an option just for the sake of making a choice. In cases where you are genuinely unsure, simply leave the *[Don't Know]* option checked.
- avoid the temptation to 'average out' your answers** over the course of the survey - that is, do not rate a clip as *[Human]* simply because you rated several previous clips as *[Artificial]*, or vice versa. As mentioned above, the tests do not adhere to any such averages. You should instead rate each clip entirely on its own appearance and merits.

Before starting the survey, please indicate your familiarity with first-person shooter computer games as detailed below. Users should select the description which best fits their experience level **at any point in the past**; that is, if you *used* to play a FPS game daily but no longer do so, you should nonetheless select the *[Played frequently (daily)]* option.

- ☐ Never played, rarely or never seen
- ☐ Some passing familiarity (played / seen infrequently)
- ☒ Played occasionally (monthly / every few months)
- ☐ Played regularly (weekly)
- ☐ Played frequently (daily)

Proceed with Test

Load Last Session


Figure 5-1. BOTT Intro Screen

### Test 1 of 15: 3 Clips

Please fill in each of the required sections below, and then click the *[Submit and Continue]* button.

Please assess the clip on the right:

- ☐ Human
- ☐ Probably Human
- ☒ Don't Know
- ☐ Probably Artificial
- ☐ Artificial



PAUSED

0:00:00.000

Please explain your choice:

How many times did you view this clip, either in part or in full?


1

If you selected *[Probably Artificial]* or *[Artificial]*, please indicate how humanlike you found this clip to be:

5

Please assess the clip on the right:

- ☐ Human
- ☐ Probably Human
- ☒ Don't Know
- ☐ Probably Artificial
- ☐ Artificial



PAUSED

0:00:00.000

Please explain your choice:

How many times did you view this clip, either in part or in full?

1

If you selected *[Probably Artificial]* or *[Artificial]*, please indicate how humanlike you found this clip to be:

5

Please assess the clip on the right:

- ☐ Human
- ☐ Probably Human
- ☒ Don't Know
- ☐ Probably Artificial
- ☐ Artificial



PAUSED

0:00:00.000

Please explain your choice:

How many times did you view this clip, either in part or in full?

1

If you selected *[Probably Artificial]* or *[Artificial]*, please indicate how humanlike you found this clip to be:

5

Submit and Continue

Save and Continue Later

Figure 5-2. Main BOTT Screen



### 5.3.2 Experimental Protocol, Subjectivity and Bias

Aside from the observer effect, there are several areas in which the potential for subjectivity and the introduction of bias exist. Since our aim is to provide an *objective* measure of believability, these must be accounted for. The issues in question are enumerated below; in each case, we also outline an approach to their elimination. This *protocol* should be observed by experimenters whenever the believability test is conducted.

#### 1. Selection of Gameplay Samples

The first obvious pitfall is the selection of video clips - the selector may deliberately choose certain clips in an effort to influence the respondents. To guard against this, we propose two preventative measures: first, we ensure that the number of samples is sufficient to embody a wide variety of behaviours, and secondly, we cede control of the selection of the specific behaviours to an unbiased arbiter. In our case, we wished to compare the believability of our imitation agents against both human players and traditional rule-based bots; thus, we first ran numerous simulations with the traditional agent - over whose behaviour we had no control - to generate a set of gameplay samples, and then proceeded to use human clips embodying similar behaviour both in the believability test and to train our imitation agents. An alternative method would be to have a human third party select the clips, although this would in itself introduce the possibility of further prejudice; we feel that our approach is appropriate for the purposes of eliminating selection bias.

#### 2. Order of Presentation

Similarly, the *order* in which the videos are presented could conceivably be used to guide the respondents' answers. To prevent this, we randomize the order in which the groups of clips are displayed to each user, as well as the sequence of clips within each page; the test designer thus has no control over the order of the samples seen by

the user. Additionally, the filenames under which the clips are stored are randomized, such that the respondent cannot determine the nature of each clip based on examining the webpage source (e.g. clip 1 always human, clip 2 always artificial, etc).

### **3. User Sabotage (intentional or unintentional)**

Another issue concerns the possibility that users will *sabotage* the survey, either deliberately or inadvertently. They may, for instance, choose the ‘*Probably*’ options in a deliberate effort to artificially minimize their error and ‘beat’ the test, or make random guesses rather than selecting the ‘*Don’t Know*’ option, or attempt to average out their answers over the course of the survey - that is, they may rate a clip as ‘human’ for little reason other than that they rated several previous clips as ‘artificial’, or vice-versa. To discourage this, we include notes on the introduction page to the effect that the test does not adhere to any averages, that the user’s ratings should be based exclusively upon their perception of the character’s behaviour in each clip, and that the user should be as definitive as possible in their answers. In our case, the requirement that the test should not adhere to averages - i.e. a combination of one human, one imitation, and one artificial clip on every page - ultimately lead us to use 12 human clips, 15 traditional agent clips, and 18 imitation clips in our experiments, since pages without imitation or artificial bots would be of little benefit in assessing the believability of our system as compared to traditional agents.

### **4. User Fatigue**

A related problem is that of user fatigue; as the test progresses, the user may begin to lose interest, and will consequently invest less and less effort in examining each successive clip. We address this by including a feature enabling users to save their progress at any point, allowing them to complete the survey at their convenience.

### **5. Identification by Deduction**

It is also imperative to ensure that the test is focused upon the variable under investigation - namely, the believability of the agent’s movement and behaviour. As

such, the survey must be structured so as not to present extraneous ‘clues’ which might influence the respondents, and cause them to rate clips based on *deduction* rather than observation. For instance, the tester should ensure that all clips conform to a standard presentation format, so that the respondent cannot discern between different agents based on visual indicators; different players may have used different in-game character models, individual player names, different screen layouts, etc. To this end, we run a script over the demo files to remove all such indicators and *homogenize* them to a common format.

In the specific case of our imitation agents, this requirement that all extraneous indicators be removed raises a conflict between two of our goals in conducting the survey. If the players in two of the three clips on each page begin from the same location and exhibit near-identical behaviour, the respondent may conclude - through deduction rather than observation - that (s)he is probably viewing a human and imitation agent, and consequently that the remaining clip is more likely to be a traditional artificial agent. Note that this might not necessarily be *true*, but even an incorrect answer based on factors other than believability will adversely affect the accuracy of the results. We circumvent this problem by training imitation agents with different (but similar) samples of human gameplay to those actually used in the test. The resulting clips are therefore comparable, but do not ‘leak’ any additional information; respondents must judge whether or not they are human based solely on their appearance. At the same time, however, we obviously wish to test how accurately our agents can capture the aesthetic appearances of their direct human exemplars. To satisfy both requirements, a small minority of imitation agents *are* trained using the same human data as presented in the survey; in the experiments described below, **two** of the imitation agents were direct clones, while the remainder were trained on different data.

## 5.4 Evaluation of Results

Before evaluating the results of the survey, one should ensure that there have been a substantial number of responses with a decent distribution across all experience levels; a good ‘stopping criterion’ is to run the test until the average experience level is at least 3 (i.e. a typical gamer). Standard analyses - precision, recall, etc - can be carried out on the results, as we have done in the experiments section below; however, we also wish to formulate a *believability index* which is specifically designed to express the agent’s believability as a function of user experience and the certainty with which the clips were identified.

### 5.4.1 Computing the Believability Index

Recall that each clip is rated on a scale of 1 to 5, with 1 corresponding to ‘definitely human’ and 5 corresponding to ‘definitely artificial’. Obviously, the *true* value of each clip is always either 1 or 5. Thus, we can express the degree to which a clip persuaded an individual that the visualised character was human as the normalised difference between that person’s rating and the value corresponding to ‘artificial’:

$$h_p(c_i) = \frac{|r_p(c_i) - A|}{\max(h)}$$

where  $h_p(c_i)$  is the degree to which person  $p$  regarded clip  $i$  as depicting a human,  $r_p(c_i)$  is person  $p$ ’s rating of clip  $i$ ,  $A$  is the value on the rating scale corresponding to ‘artificial’, and  $\max(h)$  is the maximum possible difference between a clip’s rating and  $A$ . In other words,  $h_p(c_i)$  will be 0 if the individual identified a clip as artificial, 1 if he identified it as human, and somewhere in between if he chose one of the ‘Probably’ or ‘Don’t Know’ options. We now *weight* this according to the individual’s experience level:

$$w_p(c_i) = \frac{e_p h_p(c_i)}{\text{avg}(e)}$$

where  $e_p$  is the experience level of person  $p$  and  $\text{avg}(e)$  is the mean experience level of all respondents. This serves to *boost* the ratings of the more experienced participants within the sample by attaching extra weight to their responses, while *discounting* the ratings of less experienced respondents accordingly.

Finally, we sum the weighted accuracies across all clips and respondents, and take the average:

$$b = \frac{\sum_p^n \sum_i^m w_p(c_i)}{nm}$$

where  $b$  is the believability index,  $n$  is the number of individual respondents, and  $m$  is the number of clips. The believability index therefore lies in the range (0, 1), and is essentially a weighted representation of the degree to which a given type of clip was identified as human - which, for clips involving an artificial agent, means the degree to which its behaviour *deceived* the viewer into believing it to be human.

### 5.4.2 The Confidence Index

The weighting mechanic described above serves to bias the believability index towards the responses of more experienced survey participants, relative to the respondents' level of experience as a whole; it does not, however, indicate what that level of experience was. In order to express the overall *strength* of the result and to facilitate comparison between agents evaluated in different surveys, we therefore compute a *confidence index* as follows:

$$c = \frac{\text{avg}(e)}{\text{max}(e)}$$

where  $avg(e)$  is the average experience of the respondents, and  $max(e)$  is the maximum experience level; the confidence index is thus conditioned upon a sufficient absolute level of expertise among respondents. Decoupling the indices in this manner produces a final believability index which is internally representative of the respondent's ratings in each individual survey, while the confidence index ensures that the results can be compared against those obtained in other tests or using other agents. In practice, then, a 'good' result for an AI agent would involve a high value of  $b$  for both the agent and human clips - indicating that the respondents both accurately identified true human players, and *misidentified* the artificial agent as being human - together with a confidence index of 0.6 or more (indicating that respondents were of a significant level of experience).

## 5.5 Experiments

In this section, we detail an experiment carried out using the believability test in conjunction with our imitation agents, as described in the previous chapter. The purpose of this experiment was twofold; first, to evaluate the believability-test framework itself, and second, to examine how believable our imitation agents were in comparison with human players and traditional rule-based artificial agents.

### 5.5.1 Methodology

The experiment consisted of 15 groups of 3 clips; these clips were up to approximately 30 seconds in length. We first ran numerous simulations involving the rule-based artificial agent to derive a set of gameplay samples, and then used similar samples of human players both in the test itself and to train our imitation agents. The rule-based agent used was the Quake 2 Gladiator bot [134], which was chosen due to its reputation as one of the best bots available. Indeed, it was so highly regarded that

its author, JP van Waveren, was hired by iD Software to adapt it as the official Quake3 bot; this subsequently formed the basis of his Master's thesis [135].

With the video clips in place, the URL of the survey site was distributed to the mailing lists of several colleges in Ireland and Germany. After a one-week test period, we had amassed a considerable number of responses. Having discarded incomplete responses, we were left with 20 completed surveys, totalling 900 individual clip ratings; the average experience level of respondents was 3.2, giving a confidence index of 0.64 for the survey as a whole.

| Clip Type  | Believability | Confidence | Recall (%) | Precision (%) |
|------------|---------------|------------|------------|---------------|
| Human      | 0.69          | 0.64       | 68.37      | 78.39         |
| Imitation  | 0.69          |            | 68.60      |               |
| Artificial | 0.36          |            | 36.69      | 50.87         |

**Table 5-1 - Believability/Confidence indices, Recall and Precision values. Recall values consider classifying the imitation and artificial agents as 'human' to be the desired results. Precision is estimated over [human or imitation] identified as human, and FSM agent identified as artificial.**

## 5.5.2 Results

As can be seen from Table 5-1 above, the survey produced a very favourable impression of our imitation agent compared to the traditional artificial agent. The believability indices for human, imitation and artificial clips were 0.69, 0.69 and 0.36, respectively. In other words, the imitation agents were misidentified as human 69% of the time, while the rule-based agents were mistaken as human in only 36% of cases (weighted according to experience). Clips which actually *did* depict human players were correctly identified 69% the time.

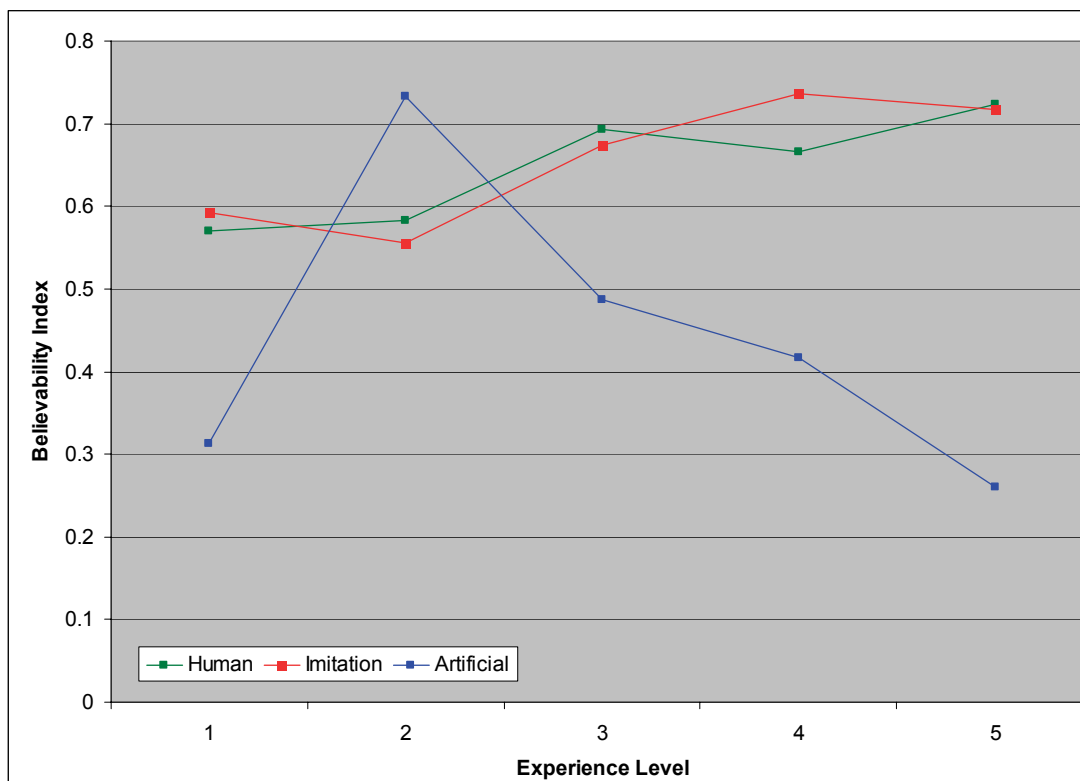
Essentially, it seems that respondents were generally unable to discern between the human players and our imitation agents; these results are corroborated by the recall values, which indicate that both the human and imitation clips were classified as

human in approximately 68.5% of cases, while the rule-based agent was classified as human only 36.69% of the time. Since the human sources used to train the imitation agents were *different* than those human clips presented as part of the test, this implies that the results are based on the general abilities of the imitation mechanism, rather than any factors unique to the clips in question.

| Rating     | Imitation | Human | Artificial |
|------------|-----------|-------|------------|
| Human      | 225       | 145   | 102        |
| Artificial | 103       | 67    | 176        |
| Neutral    | 32        | 28    | 22         |

**Table 5-2 - Confusion matrix showing overall classification of clips. Classification threshold is taken to be a rating of ‘Probably Artificial/Human’ or more definite**

Further indication of the imitation agents’ effectiveness is evident in the graph of accuracy against experience level show in Figure 5-3 below:



**Figure 5-3 - Variation of believability for human, imitation and artificial clips with experience**



The chart generally follows an intuitive trend; as experience level rises, respondents correctly identify human clips more frequently, and misidentify the traditional agent as human less frequently. The identification of imitation agents as human, by contrast, closely parallels that of genuine human clips. These trends may be explained by the fact that more experienced players have a greater knowledge of characteristically human behaviours - smooth strafing, unnecessary jumping, pausing to examine the environment, and similar idiosyncrasies - which the traditional agent would not exhibit, but which were captured and reproduced by the imitation bots. This interpretation is supported by many of the comments submitted by the most experienced respondents, some of which are recounted in Table 5-3 below.

| Experience | Comment  |
|------------|--|
| 5          | Bunny hop for no reason, also seems to be looking for enemies                                      |
| 5          | Fires gun for no reason, so must be human  |
| 5          | Unnecessary jumping  |
| 5          | human player since it performs complex moves   |
| 5          | Shooting to cover possible enemies with rocket launcher  |
| 5          | takes short cuts, shows smart aiming/shooting towards the end                                      |
| 5          | Stand and wait. Ai wouldn't do this (?)  |
| 5          | paused and jumped while pointing the gun around the corner first                                   |
| 5          | Human as they knew how to Rocket jump  |
| 5          | Human - it appears as if it is responding to objects appearing.                                    |
| 5          | fast moves, smart item cycling   |
| 5          | at the beginning there was a short period where the character was waiting, that made it seem human |
| 5          | The rocket jump and the short sequence of backward running at the end suggest this was human       |

**Table 5-3 - Sample comments from imitation clips misidentified as human**

Some further comments on the interestingly atypical results for the traditional artificial agent at experience levels 1 and 2 are required. These ratings come from individuals who respectively describe themselves as never having seen first-person shooter games before, or as having minimal familiarity with them. At the lowest experience level, respondents correctly identified the artificial agents with unusual

accuracy; at experience level 2, by contrast, the artificial agents were disproportionately misidentified as human. Though we hesitate to venture an in-depth psychological exploration of these results, we regard the most likely explanation for the accuracy of identification at experience level 1 to be *oversensitivity* caused by a complete lack of knowledge of FPS games, while the inaccuracy observed at experience level 2 is likely due to *overconfidence* resulting from the individual's minimal knowledge. In the case of subjects of experience level 1, our conclusion is based on an examination of the comments left by respondents in the course of the survey. We noticed a recurring theme in their remarks:

| Experience | Comment  |
|------------|--|
| 1          | no-one could be looking at how jagged all those turns are without being dizzy! |
| 1          | just looks like bizarre movement   |
| 1          | this clip is confusing because the movement is very erratic                    |
| 1          | Running up steps - not good!   |

**Table 5-4 - Sample comments from correctly-identified traditional agent clips**

As discussed in previous chapters, humans tend to exhibit smooth, purposeful motion in comparison to traditional artificial agents, an aspect of human behaviour that we seek to imitate in our own work. However, it appears that respondents of experience level 1, having no prior frame of reference upon which to base their conclusions, were *oversensitive* to this effect, believing the agent to be moving in a manner that would be *impossible* or *nonsensical* for humans when this was not the case. See, for instance, the last comment in the table above; the user judged the clip as depicting an artificial agent due to the fact that the agent ran up some stairs, which in most environments is a strict necessity for humans and agents alike. This explanation gains credence from comments left by respondents of experience levels 3 to 5, which were found to indicate substantially less certainty and more nuanced reasoning *even when correctly identifying an artificial agent*, as shown in Table 5-5 below.

| Experience | Comment  |
|------------|--|
| 3          | character seemed human, but went into a corner which was no use and left very quickly, didn't need time to look around |
| 3          | very calculated, didn't jump up the stairs   |
| 5          | appears very human-like but something is not quite right   |
| 5          | incoherent movement ... either an artificial player or a newbie  |
| 5          | Some odd movement. Could just be a poor player.  |

**Table 5-5 - Sample comments from correctly-identified traditional agent clips**

Unfortunately, in the case of experience level 2, no comments were left for the incorrectly identified artificial agents; we are therefore unable to draw any firm conclusions about the respondents' reasoning. Note, however, that at both experience levels 1 and 2, the graph of our imitation agents' believability closely follows that of actual human players. Essentially, it seems that even minimally-experienced respondents agreed that the imitation agents closely resembled the human players in their behaviour and appearance, but disagreed on how human players behave in the first place. Their responses are perhaps most noteworthy for the contrast they provide with individuals of experience level 3 and above (that is, semi- to very frequent first-person shooter players), who identify the artificial agents with increasing accuracy in proportion to their experience.

### 5.5.3 Feedback

Several respondents also commented on the believability-testing system itself. While opinions on the structure of the survey, the user interface, and the testing mechanism were generally very positive, a frequent complaint pertained to the number of clips which users were required to evaluate; many individuals found it to be excessive, even with the ability to save and resume at any point. This also goes some way towards explaining the half-finished tests mentioned at the start of the experiments section. While we did attempt to balance our need for a comprehensive survey with

the potential for user fatigue, we clearly overestimated the number of clips which respondents would consider acceptable; based on the responses, we would propose that perhaps 10 groups of three clips is a more appropriate figure, so long as the experimenter ensures that the videos embody a diverse variety of behaviours.

In addition, those components of the survey which users perceived as being “optional” - the fields which allowed the respondent to specify how many times (s)he watched each clip, and in the case of supposed artificial agents to provide an estimation of their ‘humanness’ on a scale of 1 to 10 - were scarcely utilised at all. While these variables are not employed in the computation of the believability/confidence index or other statistical analyses, they could potentially provide some very interesting fine-grain ancillary data, both on individual samples and across entire categories. These features are, however, rendered effectively redundant if the users only choose to use them occasionally or not at all. It might therefore be beneficial, for future reference, to describe them as being mandatory on the survey intro screen, or even to prevent the user from moving to the next group of clips until all such fields on the current page are completed.

#### 5.5.4 Remarks

As a final note, it is instructive to reflect upon the consequences of both Harnad’s gradated hierarchy, and the strong relationship between experience level and accuracy of identification established in our own experiments, with respect to Turing’s original test. As Harnad notes, the simplistic pass-or-fail nature of the Turing test has caused many researchers to turn away from it as a feasible yardstick of intelligence, preferring instead to pursue gradual progress in many parallel fields. Turing himself did not address the question of “experience” in his paper, most likely regarding it to be redundant with respect to so common a human ability as natural language conversation. But surely, this is too simple - after all, linguistic aptitude is

still dependent upon experience and an accompanying variety of factors. If a machine is capable of deceiving, for instance, a non-native speaker, should it then be regarded as intelligent? To what degree? What of a dyslexic native speaker? One could not imagine denying the intelligence of a human simply because he or she suffers from an impairment of their symbol-manipulation capacities - if a machine were capable of convincing such an individual that it is a fellow human, or indeed a fellow *dyslexic* human, upon what grounds would we withhold the same judgement? And what of a juvenile interrogator? If the artificial interlocutor can fool an eight-year-old, should it then be regarded as *possessing* the intellect of an eight-year old?

## 5.6 Conclusion

Due both to the novelty of sophisticated autonomous virtual human agents until recent times, and the reluctance of the AI community to address a difficult subjectivity problem, there exists no rigorous method of gauging the ‘believability’ of game bots, nor of objectively comparing this quality in different agents. While previous investigations have used human judges to pinpoint the attributes which contribute most to their agent’s perceived ‘humanness’, the believability tests employed in these experiments have been consistently informal, ad-hoc and non-modular. Given that one of the central aims of our work lies in improving the believability of game bots, and that the lack of a standard testing methodology is a major impediment to the orderly evaluation of artificial game agents in general, this is clearly a shortcoming which needs to be addressed. In order to produce a credible assessment of agent believability, we outlined a *framework* which facilitates rigorous, objective testing of the degree to which game agents are perceived as human; this consists of an anonymous web-based *survey* and a *protocol* to which testers should adhere, in order to eliminate potential subjectivity and bias. To ensure that the results of the believability test can be compared across different agents, we also formulate a

*believability index* expressing this ‘humanness’ as a function of the user’s experience and the accuracy with which the agents/humans are identified.

We then conducted a series of experiments with the believability framework, using our own imitation agents as a test-case, together with actual human data as a control group and a well-regarded traditional artificial agent as a comparator. The purpose of these experiments was twofold; firstly, to demonstrate the effectiveness of the testing methodology, and secondly to evaluate our imitative mechanisms. The survey produced a very favourable impression of our agents; as the experience level of the respondents increased, they correctly identified the human players more frequently, and misidentified the traditional agent as human less frequently. Our imitation agents, by contrast, were judged human in close parallel with the actual human players. We consider this to be very strong evidence in support of our original premise - namely, that imitation learning has the potential to produce more believable game agents than rule-based systems or other traditional AI techniques alone.

# 6 Conclusion

## 6.1 Introduction

In this chapter, we look back over the contents of this thesis, and suggest some future directions for our research. We begin by providing a full list of our publications, which have been listed in the introduction to each preceding chapter according to their relevance. In Section 6.2, we review these chapters in turn.

In Section 6.3, we discuss a number of possible areas for continuing research in our own work. First, we describe several future improvements and side-projects relating to the QASE API. We then outline some potential refinements to our various imitative models; these approaches deviate somewhat from the core principles of the research as presented throughout this thesis, and so we discuss in each case why we opted not to implement them previously.

In Section 6.4, we consider *open* questions; as distinct from the previous section, these are avenues of research which follow from our work, but require the development of new models rather than improvements to existing ones, or involve fields of research which are quite distinct from that of games-based AI. We first consider the issues of behaviour-switching and anticipation, outlining the difficulty of realising either via purely imitative approaches. We then consider the applicability of our game-based imitative models to real-world scenarios, before discussing the potential refinements to our imitative agents which could result from the incorporation of real-world biometric data.

Some closing remarks, acknowledgements and thanks conclude this chapter.

## 6.1.1 Full List of Publications

For quick reference, the following is a list of the publications which we have achieved while conducting our research. Papers which are relevant to the content under discussion are also listed in the introduction to each chapter of this thesis.

### 6.1.1.1 Conference Proceedings

- **Gorman, B. and Humphrys, M.**  
*Towards Integrated Imitation of Strategic Planning and Motion Modelling in Interactive Computer Games*, in proc. 3rd ACM Annual International Conference in Computer Game Design and Technology (GDTW 05), Liverpool, November 2005, pages 92-99
  - winner, "Best Presentation" award
- **Gorman, B., Fredriksson, M. and Humphrys, M.**  
*QASE - An Integrated API for Imitation and General AI Research in Commercial Computer Games*, in Proc. 7th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games, Angoulême, November 2005, pages 207-214
  - winner, "Best Paper" award
- **Gorman, B., Thureau, C., Bauckhage, C., and Humphrys, M.**  
*Bayesian Imitation of Human Behavior in Interactive Computer Games*, in Proc. Int. Conf. on Pattern Recognition (ICPR'06), volume 1, pages 1244-1247. IEEE, 2006
- **Gorman, B., Thureau, C., Bauckhage, C., and Humphrys, M.**  
*Believability Testing and Bayesian Imitation in Interactive Computer Games*, in Proc. 9th Int. Conf. on the Simulation of Adaptive Behavior (SAB'06), volume LNAI. Springer, 2006
- **Gorman, B. and Humphrys, M.**  
*Imitative Learning of Combat Behaviours in First-Person Computer Games*, in Proc. 10th International Conference on Computer Games: AI, Mobile, Educational & Serious Games, 2007



### 6.1.1.2 Journal Papers

- **Gorman, B. and Humphrys, M.**  
*Towards Integrated Imitation of Strategic Planning and Motion Modelling in Interactive Computer Games*, ACM Computers in Entertainment, Volume 4, Issue 4 (October-December 2006), ISSN 1544-3574, Article 10
- **Bauckhage, C., Gorman, B., Thureau, C., and Humphrys, M.**  
*Learning Human Behavior from Analyzing Activities in Virtual Environments*, MMI-Interaktiv, Nr. 12, April 2007, ISSN 1439-7854
- **Gorman, B., Fredriksson, M. and Humphrys, M.**  
*The QASE API - An Integrated Platform for AI Research and Education Through First-Person Computer Games*, International Journal of Intelligent Games and Simulations, Volume 4 Issue 2, June 2007

## 6.2 Thesis Review

In Chapter 1, we outlined the central concepts of imitation learning, and provided an overview of its use in fields such as robotics and computer vision. We then proposed that - considering the ease with which they can record noiseless, objective data, the availability of vast existing sample libraries, and the fact that such recordings encode nothing less than the real-time *planning behaviours* of the human player, as opposed to the simple limb movement data typically used in robotic imitation - computer games provide an ideal platform for academic imitation learning research. Similarly, we noted that imitation learning holds the potential to produce more believable, challenging and interesting game agents, which would naturally be of interest to the games industry and players alike. Our task, then, was to examine the stream of network communications recorded during gameplay sessions, and reverse-engineer the player's decision-making process from its observable in-game results. To that

end, we defined the principal contributions which our work would make to the field, in three distinct categories:

- **THEORY: MODELS OF COMPUTATIONAL GAME IMITATION**

Drawing upon biological evidence, approaches from the field of robotics, and our own experience in computer games, we developed a number of imitative models designed to learn and reproduce various aspects of observed human behaviour. These mechanisms fall into three distinct subcategories, closely mirroring a widely-used psychological model of human planning; *strategic* long-term behaviours, *tactical* medium-term behaviours, and *reactive* short-term behaviours.

- **IMPLEMENTATION: A MACHINE-LEARNING API FOR GAMES**

One major obstacle encountered early in our research was the lack of an API that was suitable for our intended purposes. Existing development platforms were incomplete and ad-hoc, often with scattered documentation and unintuitive interfaces. Rather than writing a minimal library which provided only the specific features we required - thus leaving future researchers in exactly the same position in which we had found ourselves - we instead decided to develop a comprehensive API, called the *Quake Agent Simulation Environment (QASE)*, which would encompass all the functionality necessary to conduct further work in this field. While geared principally towards machine- and imitation learning, it would also represent a platform for cognitive agent research in general.

- **EVALUATION: THE BOT-ORIENTED TURING TEST (BOTT)**

As noted above, we present statistical validations of the functionality of each imitative model in their respective subchapters. However, given that one of our aims was to demonstrate the capacity of imitation learning to produce

more *believable* game agents, this alone was not sufficient. We wished to further evaluate the “humanness” of our agents, as perceived by actual observers. Here, we again ran into an obstacle; there was no rigorous means of evaluating this quality in agents. Some minor investigations of believability had been conducted, but these were invariably highly informal, and certainly not applicable to inter-agent comparisons. We thus decided to design a generalised approach to such testing, which we call the ***Bot-Oriented Turing Test (BOTT)***.

In Chapter 3, we described the first of these contributions - the Quake Agent Simulation Environment (QASE). We adopted iD Software’s *Quake 2* as our testbed for a number of reasons; it was prominent in the literature, existing resources were somewhat more substantial than for other games, and as a *first-person shooter* it offered an attractively direct mapping of human decisions onto observable agent actions. We proceeded to outline our motivations for developing QASE, discussing both the shortcomings of existing APIs and the steps which we had taken to circumvent them. We then described QASE’s network layer, which acts as an *interface* between the local AI routines and the Quake 2 server, before detailing the API’s *agent architecture*, which allows bots to be created from any of several levels of abstraction and provides a wide variety of supporting facilities. We concluded the chapter with a discussion of QASE’s adoption by numerous universities as both an undergraduate teaching tool and research platform.

In Chapter 4, we described a number of imitative mechanisms developed using QASE and its MatLab integration facilities. We first outlined a behaviour model which distinguishes between long-term *strategic*, medium-term *tactical* and short-term *reactive* behaviours. In Section 4.3, we proceeded to detail a reinforcement-learning approach to the imitation of the human player’s navigation behaviours, centred upon his pursuit of item pickups as *strategic goals*. This involved deriving a

topological map of the environment by clustering the set of positions occupied by the player in the course of the game session, drawing edges between the resulting *waypoints* based on the player’s observed motion, and employing *value iteration* to learn the relationship between the player’s current state and subsequent navigation routes. This system proved adept at capturing the human’s *program-level* behaviour; the imitation model identified and pursued the human’s *goals*, without necessarily reproducing his precise actions at each timestep.

In Section 4.4, we described the integration of this strategic navigation system with a Bayesian mechanism for the imitation of *tactical* motion-modelling behaviours. Our approach involves reading the player’s orientation from the demo file on each timestep, along with values indicating his movement speed, posture, and whether he is firing his weapon. We then cluster this data to produce a set of *action primitives* - that is, aggregated representations of similar actions performed at different times in different places. We sequence these primitives using a heavily adapted version of Rao et al’s model of action sequencing in human infants [106], which expresses the next action as a function of the agent’s current state, next state and goal state - information which is supplied by the strategic navigation system. We also proposed a technique, *imitative keyframing*, which combines the consistency offered by *absolute* orientation primitives with the fine-grain adjustments provided by *relative* values.

In Section 4.5, we outlined a model for the imitation of *reactive* combat behaviours; specifically, weapon-selection and aiming. Drawing upon previous work which had found that direct imitation produced excessively accurate aim, our approach was to instead learn the human player’s *inaccuracy*. To model this, we first reconstructed the player’s visual perception of his opponent’s motion from the available low-level data, using their respective positions, velocities, and orientations on each timestep. We then trained a series of three expert networks using this data; one to select the most appropriate weapon, one to adjust the agent’s aim, and a third to determine

whether the agent should open fire. A number of interesting phenomena - and some associated problems - were captured by the imitative mechanism, such as the inability to adjust aim in certain directions as quickly as in others, caused by the limitations of the human's mouse input. We noted that this was an excellent example of humanlike behaviour which would not be implemented by traditional artificial agents, both because a top-down designer would not think to include it, and because it would be difficult to reproduce accurately using a rule-based approach.

For each of the imitative models described above, we conducted experiments to provide a statistical validation of their functionality. We also wished, however, to ascertain the degree to which the imitation agent is *perceived* as being “humanlike” by an actual observer. We noted that there was no existing means to rigorously gauge the believability of game agents, the few previous attempts having been extremely ad-hoc and informal. We therefore proposed a generalised approach to such testing in Chapter 5 - the **Bot-Oriented Turing Test**, or **BOTT**. This took the form of an anonymous online questionnaire, an accompanying protocol to which examiners should adhere in order to maintain the integrity of the survey, and the formulation of a *believability index* which expresses each agent's humanness as indicated by its observers. To validate the survey approach, we presented a series of experiments which used the believability test to evaluate our own imitation agents against both human players and traditional artificial bots. The results showed that the imitation agents were perceived to be significantly more believable than traditional bots; we considered this to be strong evidence in favour of our thesis that imitation learning holds the potential to produce markedly more humanlike game agents.

## 6.3 Future Work

In this section, we describe some potential future improvements to the QASE API and our existing imitative models.

### 6.3.1 QASE Development

While the current version of QASE provides a comprehensive set of tools for agent construction - essentially incorporating every feature which occurred to us as being potentially useful, both from the machine-learning and general agent research perspectives - there are a number of further developments which we wish to undertake. These projects will be conducted in parallel with any changes implemented due to comments or suggestions from existing QASE users.

#### 6.3.1.1 *QASE*<sup>3</sup>

One of the first additions we intend to make is to extend QASE such that it is capable of realising agents in Quake 3 multiplayer matches. As discussed in Section 3.3.2, QASE's network layer is quite cleanly decoupled from the other structures in the API, meaning that such extension is a relatively straightforward process. This extension will be transparent from the user's perspective; *QASE*<sup>3</sup>, as we intend to call the update, will automatically determine the game running on the specified server, and will instantiate base or derived objects as necessary. Not only will this serve to provide another testbed for researchers to investigate - Quake 3 is a more fast-paced, reactive and team-based game than its predecessor; sufficiently different to make it an attractive alternative platform for certain areas of research - but it will also act as proof-of-concept for a related sub-project, as discussed below.

#### 6.3.1.2 *QNET*

Building upon *QASE*<sup>3</sup>, we intend to release the network layer of QASE as a standalone framework, entitled *QNET*. Again, as discussed in Chapter 3, the decoupling of the network layer from the remainder of the API allows the communication components to be viewed as a generic foundation for the creation of

client agents in a wide variety of multiplayer game. The network layer itself will be also updated to support TCP-based games, in addition to the more common UDP. It is our hope that QNET will assist in furthering interest in computer games as a viable platform for both academic research and education.

#### *6.3.1.3 QScript*

Many undergraduate AI and programming courses employ some variation on a “robot world” - that is, a simulation in which the student, often using a simplified pseudolanguage, must write control programs to navigate through a 2D maze. We intend to augment QASE so that it may be adopted for the same purposes. This will involve the definition of a simple scripting language, *Qscript*, which will include standard programming concepts such as loops, conditional statements, etc. Atomic commands (*moveForward*, *moveBackwards*, *turnLeft*) will also be provided. The scripts written by the student, either in a text editor or in a supplied GUI front-end, will serve to define the agent’s AI cycle; they will be read into the QScript interpreter and processed upon each gamestate update. The obvious advantage of using QASE in preference to existing robot world simulators is that Quake 2 allows a much greater range of programming skills and AI concepts to be explored; furthermore, when students are ready to being writing full agents using standard code, they will already be familiar with the API and testbed.

#### *6.3.1.4 GANN Interface*

One final addition which we intend to make is a front-end interface to QASE’s genetic neural network architecture. This will take the form of a GUI which permits various parameters of the neural network and genetic algorithm manager - the number of inputs, hidden layers and outputs, the size of the gene pool, the rate of mutation, the duration of each generation, etc - to be specified, and the relevant

objects automatically instantiated. We will also provide an inbuilt example of the GANN mechanism in operation; this will involve a single-room Quake map with items scattered at various positions, with the agents having the task of collecting as many pickups as possible while continuously aiming at the nearest opponent. Although this is a comparatively minor addition to the API's existing functionality, observing the agents' emergent intelligence as they learn to navigate and aim in a full 3D world will provide an excellent conceptual illustration for undergrad students.

### 6.3.2 Potential Refinements to Existing Models

At various points during our research, a number of minor potential refinements to our existing imitation models occurred to us. Though we did not implement these modifications - due both to time constraints and to our skepticism that they would provide a noticeable improvement, for the reasons enumerated - some of these refinements are presented here for the purposes of future study.

#### *6.3.2.1 State Compression*

In any game session, there may exist one or more items which the human player was rarely observed to collect, and whose associated elements in the set of state vectors used by the strategic navigation system are therefore minimally variant. By employing a dimensionality-reducing technique such as PCA - or simply by eliminating those elements directly - the vector representation could be compressed, reducing the state space and leading to greater storage and computational efficiency.

However, this is likely to have the significant disadvantage of reducing the agent's ability to adapt to uncommon or unseen situations. It is entirely possible that the agent may, in fact, regularly collect items which the human player did not. For instance, the human player may pass a particular item location after the item itself



has already been collected by an opponent, and is therefore inactive; the agent, by contrast, may pass while the item is present and collect it. Similarly, enemy players have the ability to discard their weapons, which then become collectible - if the enemy possesses a weapon which the imitation agent's human exemplar did not, the same issue arises once again. These problems are compounded when we consider that our strategic imitation model compares the agent's current state against *all* observed human states, and weights the agent's goals accordingly. Thureau et al [132] previously investigated PCA in their work on action primitives, but did not use it to reduce the dimensionality of their state vectors; they later reported that they had found all the state features to be necessary for consistent imitation.

It was for these reasons that we opted against investigating state-compression techniques in our current work. Nonetheless, it is possible that an alternative, more suitable means of reducing the agent's state representation could be developed; and the benefits of finding an appropriate solution would be significant. We therefore present state compression as an issue which is worth revisiting in the future.

#### *6.3.2.2 Weapon Similarity Matrices*

As discussed in Section 4.5, a variety of different weapons exist in Quake 2, as is the case with all first-person shooter games. These weapons conform to a number of common characteristics; certain guns fire slow-moving but powerful projectiles, others fire quick but weak bursts. Some are ideally suited to ranged combat, whereas others are more useful in close quarters. Additionally, different sets of guns share a common ammunition type; the shotgun and super-shotgun use shells, the hyperblaster and BFG use cells, the machine gun and chaingun use bullets, and so forth. By constructing a matrix which defines the comparative similarity of each pair of weapons and their shared ammunition, it may be possible to leverage this information to the benefit of the various imitation models. For instance, if a particular

weapon is not currently active on the map, the strategic navigation system could proportionally boost the path values associated with similar guns. If the agent does not have a specific weapon, the reactive combat weapon-selection model could switch to the most similar weapon which the agent *does* possess.

The disadvantage of such an approach, from the perspective of our own research, is that it inherently introduces some very high-level expert knowledge into the system, which we strenuously sought to avoid. In addition, defining an *a priori* matrix of weapon similarities comes dangerously close to enforcing a top-down rule-based approach; *if* weapon X is unavailable *then* choose weapon Y *else* weapon Z. It also leaves the system open to *designer bias*, since there is no strict means of gauging the precise similarity of two weapons - the matrix would therefore reflect only the personal opinion of its creator. These factors place the proposed modification in direct opposition to the overriding ethos of our work.

Perhaps most importantly, explicit encoding of weapon similarities is likely to be entirely unnecessary, since *the human player is already aware of such similarities*. If a particularly desirable weapon is unavailable, he will pursue it or an equivalent weapon on the map, or switch to a comparable gun which he already possesses; such behaviours are captured by the strategic navigation and reactive combat imitation models. Note also that this *implicit* weapon-similarity reasoning affords the agent a far greater degree of nuance than would a top-down definition of the kind described above; the player may exhibit preferential behaviour only with respect to *certain* similar guns, rather than adhering to a blanket rule across all weapons. Our existing imitative mechanisms should, therefore, capture not only the human player's understanding of weapon similarities, but also the degree to which this effect is present.

## 6.4 Open Questions

In this section, we describe potential avenues of future research which require either the development of new models, or involve fields of research which lie outside that of games-based artificial intelligence.

### 6.4.1 Meta-Behaviours

The mechanisms presented in the preceding chapters of this thesis do not, as yet, constitute a fully-realised imitation agent. In order to achieve this, models which implement one additional category of behaviour must be developed; we call this category *meta-behaviours*. Our existing models operate by encountering a particular state, and executing learned motor commands accordingly. Meta-behaviours are decision processes which do not define any motor commands of their own, but instead combine and control the functionalities of the existing strategic, tactical and combat systems to produce high-level *global* behaviours. Meta-behaviours comprise two main subcategories, *anticipation* and *behaviour-switching*. Unfortunately, as we discuss below, the development of models to capture these phenomena is extremely problematic from the perspective of imitation learning.

#### 6.4.1.1 Anticipation

Anticipation refers to the human player's ability to project the probable short- to medium-term actions of his opponent, and adjust his tactics accordingly. For instance, if the player is pursuing an opponent who enters a room with only one other exit, the player may decide to navigate an alternative route to that point and ambush the enemy as he emerges. If the player knows that his opponent has no weapon, he may move to collect the nearest available gun and again lie in wait for the enemy. As outlined above, anticipation is therefore a tactical/strategic reasoning behaviour

which does not require additional motor functions in order to be implemented; a combination of the existing mechanisms is sufficient. Laird [76] describes a rule-based approach to modelling anticipation in his SOAR Quakebot, which consists of estimating the opponent's current state, projecting what the agent would do in that same state, and taking countermeasures accordingly.

However, it is difficult to imagine a purely imitative implementation of such a mechanism. Even in Laird's approach, the rules for detecting and counteracting each anticipation scenario - specifically, the two which were outlined above - must be explicitly stated. Indeed, it was necessary to define the rules in such detail that, for instance, the agent was instructed not to attempt an ambush in a narrow corridor, but to wait for the opponent to enter a more open space before engaging him. This neatly illustrates the core difficulty of implementing an anticipation system with respect to imitation learning; it is a high-level reasoning, co-ordination and execution task which inherently requires expert knowledge of the game. While it would be possible for our imitation agents to construct an internal estimation of the opponent's current state, and project his likely behaviour on that basis, the question then becomes: what action should the agent take? And how should we actually execute the decision process? We could simply instruct the agent to go to the appropriate location, but this would be indistinguishable from a rule-based agent. We could boost the utility values of the relevant nodes in the strategic navigation map in order to draw the agent in that direction, but this would again require explicit hardcoding. There is no clear way to *deduce* and passively pursue the appropriate behaviour from demonstration, since anticipation is an innately top-down, pre-emptive behaviour.

A further problem concerns the question of *when* it is appropriate for the agent to begin anticipating the opponent's actions, rather than pursuing its own objectives. In Laird's case, this was again achieved by specifying a series of hardcoded rules - the agent would begin anticipating if the opponent was visible, and was greater than X

distance units from the bot, and had its back turned, and was moving in the direction of a known item in an adjoining room. We could certainly set “triggers” at various points in the environment to induce such behaviours - the embedding of such tactical information in waypoint nodes has already been proposed by Liden and Hancock [104] - or attempt to establish a relationship between the player’s movement and that of the opponent while he is visible and in certain areas of the map, but this would once again involve the introduction of hardcoded rules and highly detailed expert knowledge; in contrast to our existing mechanisms, it would also mean that the model itself would have to be rewritten to account for the specific terrain of each environment.

In any case, the question of *when* to anticipate - or indeed, when to perform any other behaviour - leads into a larger related problem; that of the second meta-behaviour component, *behaviour-switching*.

#### 6.4.1.2 Behaviour-Switching

Behaviour-switching refers to the human player’s ability to dynamically adjust the balance between his primary, secondary and tertiary behaviours in accordance with his circumstances. In terms of the model described in Section 4.2, this allows him to move along the gradient from strategic to tactical to reactive, or vice-versa. Consider a player who is engaged in strategic cycling of the environment, collecting important items as rapidly as he can. He spots an opponent in the distance, but decides that he does not yet possess a sufficient arsenal to challenge him. He therefore adjusts his behaviour slightly. While ensuring that he stays out of the enemy’s sight, he begins to follow him, continuing to pick up items along the way; his behaviour has moved to the tactical/strategic boundary of our gradient. When the player is satisfied that he can defeat his opponent, he proceeds to anticipate the enemy’s destination and ambushes him; his behaviour has moved to the reactive/tactical end of the gradient, becoming increasingly reactive as the opponent fights back. Once he has defeated his

enemy, the player will generally revert to pursuing strategic goals. Behaviour-switching is the process by which these decisions to move between the different levels of the hierarchy are made.

In order to derive an appropriate switching function from the recorded game session, the datastream must first be segmented into individual “periods” of behaviour; their frequency and interrelations could then be analysed to produce an imitative model. However, the segmentation process is greatly complicated by the fact that similar behaviours are executed at different locations and times in substantively different ways; that they are occasionally, as in the case of anticipation, extremely difficult to represent without expert knowledge of the map; and that behaviours are both temporally continuous and non-binary - a player can, as demonstrated by the example above, engage in strategic/tactical/reactive behaviours simultaneously and in any ratio, and thus there are few “hard” boundaries between observed actions in the datastream. In the opening chapter of this thesis, we discussed the reasons why control-based methods of imitation were unsuitable for learning the behaviours of human gamers. To briefly recap, they require a predefined model of the task at hand - a human arm swinging a tennis racquet can be described using pendulum physics, for instance - and imitation is simply used to learn the *parameters* of the controller. A control-system approach to imitative learning of in-game human behaviours would therefore require an expert model of human reasoning itself; if we possessed such a model, then this and all other research would be redundant, since we would already have solved the problem of AI. A similar Catch-22 exists here - if we had a segmentation process capable of determining when the human’s behaviour changed and to what degree, then our work would already be done.

Examining some other segmentation approaches serves to highlight the difficulties inherent in the task of imitative behaviour-switching. Mataric, Jenkins et al [31, 37, 60-65, 82], for instance, describe an approach to the automated segmentation of

human limb motion into its constituent actions. This, however, relies on the fact that the arm's range of motion and degrees of freedom are known, and that there is a very obvious hard boundary between behaviours - when the arm stops moving in a particular direction, i.e. where the sum of the squared angular velocities in each degree of freedom falls below a given threshold. In our case, we do not know the equivalent "degrees of freedom" of the human's decision-making process, and there is no convenient boundary condition to look for. Elsewhere, Sukthankar and Sycara [122] propose a means of automatically recognising team behaviours using the game Unreal Tournament. They too remark upon the difficulty of determining the exact transition point between behaviours; to circumvent this problem, they employ a series of overlapping time windows during which a single behaviour is assumed to be dominant, and train a set of Hidden Markov Models to recognise each. However, their work examined only 3 specific behaviours, which they had defined in advance and *instructed* the human volunteers to perform; the HMMs were therefore simply classifying the best-matching behaviour from an extremely limited list of known actions. This approach is obviously not suitable for bottom-up analysis of a freeform game session, though the complexity of distinguishing even between three *known* behaviours does provide some insight into the scope of the problem.

To further illustrate this point, consider the anticipation scenario given earlier, where the opponent enters a room and the player moves to ambush him at the opposite side. Determining that the player is initially engaged in "pursuit" behaviour should not prove difficult. But what happens when the opponent moves out of sight into the next room? How would the segmentation system ascertain whether the player's subsequent behaviour was due to a decision to circle around an intercept the enemy, or whether he had opted to return to his normal strategic navigation of the environment, and re-encountered the opponent simply by chance? Perhaps we could create a specific rule for this situation - if Opponent B moves out of sight, and Player A moves to position X, and remains stationary for some period of time, and then re-

engages the opponent all within a 60-second timeframe, then this is classed as anticipation. This might be possible, although it would require specialised knowledge of the environment terrain and could certainly not be derived from the network stream alone. But we would then presumably be forced to define similar segmentation rules for every such scenario - of which there are potentially hundreds. And even then, how would we know that we had accounted for all of them? Would the rules need to be redefined for the specific terrain of each individual map? Almost certainly so. And beyond these fundamental considerations, with what degree of accuracy would these relatively crude top-down rules capture the complexities of behaviour switches? As one moves further along this line of thought, it becomes increasingly untenable to view the proposed solution as either a bottom-up imitation learning approach, or a general model of behaviour-switching; it gradually becomes indistinguishable from a rule-based system or a symbolic segmentation of the data.

#### *6.4.1.3 Remarks*

For the reasons detailed above, it may be the case that anticipation and behaviour-switching simply involve too high a level of abstract reasoning to be reproducible from low-level observation data; that they may define the limits of a purely imitative approach to agent intelligence in games, a question which we sought to explore throughout our work. There is, though it lies outside our area of interest in this research, nothing to preclude other learning approaches or rule-based systems from being layered on top of the imitative mechanisms; this is, as we discussed at the beginning of the thesis, a common practice in robotic imitation. At any rate, we would consider a full exploration of imitative anticipation and behaviour-switching to be sufficient basis for a PhD in its own right, and a project which we would be delighted to see undertaken.



## 6.4.2 Real-World Applicability and Biometric Imitation

Here, we discuss both the possibility that our in-game models may be adapted to real-world applications, and the potential benefits offered by incorporating biometric measurements of the human player into our imitation systems.

### *6.4.2.1 Applications in the Real World*

Our work sought primarily to develop imitative artificial intelligence techniques, both for the inherent benefit they would provide to the field and to demonstrate the advantages of using computer games as an imitation learning platform. Nevertheless, one of the reasons we chose modern commercial games - and the first-person shooter genre in particular - was that they offered the potential for real-world applicability. As has been discussed before, first-person games represent a relatively close approximation of the real world, and impose a very direct mapping of human decisions onto the observable actions of the game avatar; thus, techniques which have been developed in-game may be adaptable for wider use. Following the work of Thureau et al, for instance, biologists at Bielefeld University are investigating action primitives as a means of modelling the courtship behaviours of zebra finches [132]. Here, we discuss some other potential uses.

One potential real-world application was suggested by a researcher at DCU, who noted that the strategic imitation system detailed in Chapter 4.3 could provide the ideal foundation of an automated mapping and navigation aid for blind pedestrians. Indeed, it is not difficult to imagine how such a mechanism would operate. A college campus, which offers a closed but structurally complex system, would provide an excellent testbed for the project. The training data could be harvested by a number of sighted volunteers wearing GPS tracking units while conducting their daily business; as with in-game strategic imitation, this would take advantage of the human's innate

navigation skills, while giving extra weight to those paths which were followed more frequently than others. Goals would be defined as the various buildings scattered around the campus. After a certain period, the data would be processed as detailed earlier to produce the topological map; this data could conceivably be used to create a hierarchy of maps of different granularities, representing the external environment of the campus and the internal layout of each building. A sensor placed at the entrance to each structure could signal the device to switch to the relevant internal map. In terms of deployment for the blind end-user, the system would likely take the form of an audio signal of some kind, perhaps a pair of headphones connected to a modified GPS device. A series of regular stereo tones emitted by the device would then indicate - by varying their balance and intensity - when the user should turn, in which direction, and how sharply.

The Bayesian tactics system, too, would play an important role in the proposed system. As in the game world, it is not sufficient to simply define the start, end and intermediate points of a particular navigation path; there will inevitably be intervening obstacles which need to be negotiated. Tactical modelling of the kind discussed in Chapter 4.4 could be used to trigger an alert if, for instance, a door blocked the user's path, if a step or stairway lay ahead of him, if he needed to use an elevator, and so forth. Once again, the navigation system would define where to go, while the tactical primitives would provide details of how to get there.

In terms of more large-scale uses, it has been suggested that a similar approach could be applied to modelling patterns of traffic flow, by affixing GPS units to volunteers' vehicles. Given a sufficiently representative sampling, the imitation system could be used to determine not only the most common points of traffic congestion, but also the alternative paths which are most frequently used by motorists who find themselves approaching a bottleneck - and, therefore, which routes are in greatest need of attention. While the data-harvesting process would need to be repeated at regular

intervals, to account for changes in the physical topologies and in motorists' resulting behaviour, it could nonetheless be a valuable tool for urban planners and developers.

#### *6.4.2.2 Biometric Imitation*

In the same way that techniques developed in-game may be adapted for the real world, it is interesting to consider the potential usefulness of real-world data in training future agents. It is possible that biometric measurement of the human player could be used to augment the directly-observed gameplay data, resulting in further refinement to the "humanness" of the imitation. A common observation among experienced gamers is the correlation between emotional state and consequent in-game behaviour. If a particular player is defeated in multiple encounters in quick succession, for instance, his resulting frustration frequently leads to a temporary alteration in his play style. The specific nature of this alteration can differ greatly from one player to the next. Player A may adopt a more cautious approach on the next encounter, attempting to minimise the chance that he will suffer the humiliation of another defeat; in the context of our imitation model, his behaviour becomes more strategic and tactical, and less reactive. Player B, by contrast, may become more obviously agitated, opting to run in with guns blazing in the hope of breaking his losing streak; his behaviour has shifted from the strategic end of the imitation gradient to the reactive. The longevity of this shift depends in large part upon its results. If the player successfully redresses the imbalance between his score and the opponent's, he will generally revert to his standard behaviour model. If the player continues to be defeated, he will often 'give up' on the current match, becoming careless and listless as he waits for the scores to reset at the start of the next round.

At present, of course, there is no means of associating the emotional state of the player with the low-level data recorded from the network stream. These occasional deviations are, therefore, not differentiated from the player's standard behaviours;

they are simply subsumed into the general model during the training process. While it would be possible to encode some rules which, for instance, treat the behaviours observed within a 60-second timeframe after the player has sustained five consecutive defeats as being non-standard, this would at best be a crude approximation of the true effect, and at worst would simply lead to a hopelessly inconsistent bot. A better approach would be to collect real-world biometric data along with the gameplay recording - heartrate, temperature, perhaps the vocal utterances of the player, as are common in multiplayer LAN competitions - and analyse them in parallel. If a correlation between the player's in-game state, biometric measurements, and subsequent behaviour could be ascertained and reproduced, it would serve to greatly increase the humanness and believability of the imitation agent.

## 7 References

1. Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S., Kaminka, G.A., Schaffer, S. and Sollitto, C. "GameBots: A 3D virtual world test bed for multiagent research", in proc. Second Int'l Workshop on Infrastructure for Agents, MAS, and Scalable MAS, 2001
2. Alpaydin, E. "Introduction to Machine Learning", MIT Press, 2004
3. Amit, R and Mataric, M "Learning Movement Sequences from Demonstration", Int. Conf. on Development and Learning, 2002
4. Arbib M.A., Billard A., Iacobonic M. and Oztop E., "Synthetic brain imaging: grasping, mirror neurons and imitation", Neural Networks, Volume 13 Issues 8-9, pp. 975-997, 2000
5. Arbib, M.A., "The Mirror System, Imitation, and the Evolution of Language", in Kerstin Dautenhahn and Chrystopher Nehaniv ed., Imitation in Animals and Artifacts, MIT Press, 2002
6. Arkin, R.C. "Behavior-Based Robotics". MIT Press, 1998
7. Atkeson, C. G. and Schaal, S. "Learning tasks from a single demonstration", IEEE International Conference on Robotics and Automation (ICRA97), pp. 1706-1712, 1997
8. Atkeson, C. G. and Schaal, S. "Robot learning from demonstration", in proc. International Conference on Machine Learning, pp. 11-73, 1997
9. Bakkes S. and Spronck P. "Symbiotic Learning in Commercial Computer Games", in proc. CGAMES '05, pp. 116-120, 2005
10. Bakkes S., Spronck P. and Postma E. "TEAM: The Team-Oriented Evolutionary Adaptability Mechanism", in proc. Int'l Conference of Entertainment Computing, LNCS 3166, pp. 273-282, Springer-Verlag 2004

11. Bauckhage C. and Thureau C. "Towards a Fair 'n Square Aimbot - Using Mixtures of Experts to Learn Context Aware Weapon Handling", in Proc. GAME-ON, p20-24, 2004
12. Bauckhage, C, Thureau, C. & Sagerer, G. "Learning Humanlike Opponent Behaviour for Interactive Computer Games", in Pattern Recognition, Vol 2781 of LNCS Springer-Verlag, 2003
13. Billard, A. "Imitation", Handbook of Brain Theory and Neural Networks, MIT Press, 2002
14. Billard, A. "Imitation: a means to enhance learning of a synthetic proto-language in an autonomous robot", in C. Nehaniv and K. Dautenhahn ed., Imitation in Animals and Artifacts, MIT Press, 1999
15. Billard, A. "Learning motor skills by imitation: a biologically inspired robotic model", Cybernetics & Systems, 32, pp. 155-193, 2001
16. Björnsson, Y., Hafsteinsson, V., Jóhannsson, Á. and Jónsson E. "Efficient Use of Reinforcement Learning in a Computer Game", in proc. Computer Games: Artificial Intelligence, Design and Education (CGAIDE), pp. 379-383, 2004
17. Borenstein, J. and Koren, Y. "Real-time obstacle avoidance for fast mobile robots", IEEE Transactions on Systems, Man, and Cybernetics, 19(5):1179--1187, 1989
18. Bradley, P. S. and Fayyad, U.M. "Refining Initial Points for K-Means Clustering", in proc. 15th International Conf. on Machine Learning, 1998
19. Brown, C., Costello, D., Ferguson, G., Hu, B. and van Wie, M. "Quagents: Quake Agents", <http://www.cs.rochester.edu/research/quagents/>
20. Byrne, R.W and Russon, A.E. "Learning by Imitation: A Hierarchical Approach", Behavioral and Brain Sciences 21, 667-721, 1998
21. Cass, S. "Mind Games", IEEE Spectrum, p40-44, December 2002.

22. Cazorla, H.V. "Multiple Potential Fields in Quake 2 Multiplayer", Master's Thesis, Blekinge Institute of Technology, 2006
23. Champandard, A.J. "The FEAR SDK", <http://fear.sourceforge.net>
24. Charles, D. and McGlinchey, S. "The Past, Present and Future of Artificial Neural Networks in Games", in proc. Int'l Conference on Computer Games, Artificial Intelligence, Design and Education (CGAIDE), pp163-168, 2004
25. Dahlstrom, D., Wiewiora, E., Cottrell, G. and Elkan, C. "Imitative Policies for Reinforcement Learning", UC San Diego Seminar on Learning Algorithms, Spring 2001
26. Demiris, J. and Hayes, G. "Active and passive routes to imitation", in proc. AISB Symposium on Imitation in Animals and Artifacts, 1999
27. Demiris, J. and Hayes, G., "Imitative learning mechanisms in robots and humans", in proc. 5th European Workshop on Learning Robots, pp. 9-16, 1996
28. Demiris, Y. "Mirror neurons, imitation and the learning of movement sequences", in proc. 9th International Conference on Neural Information Processing, pp. 111-115, 2002
29. Dillmann, R., Kaiser, M. and Ude, A. "Acquisition of elementary robot skills from human demonstration", International Symposium on Intelligent Robotic Systems (SIRS'95) (pp. 1-38), 1995
30. DreamHack Biannual LAN Gaming Convention, <http://web.dreamhack.se>
31. Drumwright, E., Jenkins, O.C. and Mataric, M.J. "Exemplar-Based Primitives for Humanoid Movement Classification and Control", IEEE International Conference on Robotics and Automation, pp. 140-145, 2004
32. Elkan, C. "Using the Triangle Inequality to Accelerate k-Means", in proc. 20th Int'l Conf. on Machine Learning, pp. 147-153, 2003

33. Evans, R. "Black & White AI Analysis", <http://shorl.com/dryjubribrefrafra>
34. Fadiga L., Fogassi L., Pavesi G. and Rizzolatti G. "Motor facilitation during action observation: a magnetic stimulation study", *Journal of Neurophysiology*, 73(6), pp. 2608-2611, 1995
35. Fairclough, C., Fagan, M., MacNamee, B. and Cunningham, P., "Research Directions for AI in Computer Games", Technical report, Trinity College Dublin, 2001.
36. Floyd, R.W. "Algorithm 97, Shortest path", *Comm. ACM*. 5(6), 345, 1962
37. Fod, A. Mataric, M.J. and Jenkins, O.C. "Automated Derivation of Primitives for Movement Classification", *Autonomous Robots* 12 (1), pp.39-54, 2002
38. Fuchs, H., Kedem, Z., and Naylor, B., "On Visible Surface Generation by A Priori Tree Structures", in *proc. SIGGRAPH '80*, 14(3), 124--133, 1980
39. Gallese V., Fadiga L., Fogassi L. and Rizzolatti G. "Action recognition in the premotor cortex", *Brain* 119 pp. 593-609, 1996
40. Geisler, B. "An Empirical Study of Machine Learning Algorithms Applied to Modeling Player Behavior in a First Person Shooter Video Game", Master's thesis, Dept. of Computer Sciences, University of Wisconsin-Madison, 2002
41. Giese, M.A., Knappmeyer, B. and Bulthoff, H.H. "Automatic synthesis of sequences of human movements by linear combination of learned example patterns", in H.H. Bulthoff, S.W. Lee, T. Poggio, and C. Walraven, editors, *Biologically Motivated Computer Vision*, volume 2525 of LNCS, pages 538-547, Springer, 2002
42. Girlich, U., "The Unofficial Quake 2 DM2 Format Description", <http://demospecs.planetquake.gamespy.com/dm2/>
43. Giszter, S.F, Mussa-Ivaldi, F.A. and Bizzi, E, "Convergent force fields organized in the frog's spinal cord", *Journal of Neuroscience*, 13(2):467-491, 1993



44. Graepel T., Herbrich R., and Gold J., "Learning to Fight", in proc. CGAIDE 2004, pp. 193-200
45. Grafton S.T., Arbib M.A., Fadiga L. and Rizzolatti G., "Localization of grasp representations in human by PET: 2. Observation versus imagination", *Experimental Brain Research* 112, pp. 103-111, 1996
46. Grudic, G. Z. and Lawrence, P. D. "Human-to-robot skill transfer using the SPORE approximation", *International Conference on Robotics and Automation*, pp. 2962-2967, 1996
47. Hand, D., Mannila, H. and Smyth, P. "Principles of Data Mining", MIT Press 2001
48. Hannan, J. "Colin McRae Rally 2 AI Analysis", <http://shorl.com/prysovuneniny>
49. Harnad, S. "Minds, Machines and Turing - The Indistinguishability of Indistinguishables", *Journal of Logic, Language and Information* 9, 425-445, 2001
50. Hartley, T., Mehdi, Q. and Gough, N. "Applying Markov Decision Processes to 2D Real-Time Games", in proc. Int'l Conference on Computer Games, Artificial Intelligence, Design and Education (CGAIDE): p55-59, 2004
51. Hayes, G. and Demiris, J. "A robot controller using learning by imitation", in A. Borkowski, & J. L. Crowley (Eds.) *proc. 2nd International Symposium on Intelligent Robotic Systems*, pp. 198-204, 1994.
52. Hirai, K., Hirose, M., Haikawa, Y. and Takenaka, T. "The development of Honda humanoid robot", *IEEE International Conference on Robotics and Automation* (pp. 1321-1326). Leuven, Belgium: IEEE, New York, 1998.
53. Hollnagel, E. "Human reliability analysis: Context and control", London: Academic Press, 1993

54. Howard, Ronald A. "Dynamic Programming and Markov Processes", The M.I.T. Press, 1960
55. Hsu, F.H. "Cracking GO", IEEE Spectrum, October 2007
56. Hurford, J. "Language beyond Our Grasp: What Mirror Neurons Can, and Cannot, Do for the Evolution of Language", Evolution of Communication Systems: A Comparative Approach, pages 297--313. Cambridge, MA: MIT Press, 2004
57. IBM Research, "The Making of Deep Blue", <http://www.research.ibm.com>
58. Ilg, W. and Giese, M.A. "Modeling of movement sequences based on hierarchical spatiotemporal correspondences of movement primitives", in H.H. Bulthoff, S.W. Lee, T. Poggio, and C. Walraven, editors, Biologically Motivated Computer Vision, volume 2525 of LNCS, pages 528-537. Springer, 2002
59. Jacobs, R.A., Jordan, M.I., Nowlan, S.J. and Hinton, G.E. "Adaptive mixtures of local experts", Neural Computation, 3(1):79-87, 1991
60. Jenkins O.C. and Mataric, M.J. "Automated Derivation of Behavior Vocabularies for Autonomous Humanoid Motion", in proc. Second International Joint Conference on Autonomous Agents and Multiagent Systems, 2003
61. Jenkins O.C. and Mataric, M.J., "Deriving Action and Behavior Primitives from Human Motion Data" in IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2551-2556, 2002
62. Jenkins O.C. and Mataric, M.J., "Primitives and Behavior-Based Architectures for Interactive Entertainment", AAAI Fall Symposium on on AI and Interactive Entertainment, Stanford, CA, CA, Mar 26-28, 2001
63. Jenkins, O. C., Mataric, M.J. and Weber, S. "Primitive-Based Movement Classification for Humanoid Imitation", IEEE International Conference on Humanoid Robots, 2000

64. Jenkins, O.C., "Data-driven Derivation of Skills for Autonomous Humanoid Agents", PhD Dissertation, University of Southern California, 2003
65. Jenkins, O.C. and Mataric, M.J. "A spatio-temporal extension to Isomap nonlinear dimension reduction", in proc. 21st International Conference on Machine Learning, 2004
66. Johnson M. and Demiris Y., "Hierarchies of coupled inverse and forward models for abstraction in robot action planning, recognition and imitation", in proc. AISB Symposium on Imitation in Animals and Artifacts, 2005
67. Jönsson, F.M. "An optimal pathfinder for vehicles in real-world digital terrain maps", Department of Numerical Analysis and Computing Science, Royal Institute of Science, Stockholm, Sweden, 1997
68. Jordan, M.I. and Jacobs, R.A. "Hierarchical mixtures of experts and the EM algorithm", Neural Computation, 6:181--214, 1994
69. Kaelbling, L.P., Littman, M. L., and Moore, A.W. "Reinforcement learning: A survey", Journal of Artificial Intelligence Research, 4:237-285, 1996
70. Khatib, O. "Real-time Obstacle Avoidance for Manipulators and Mobile Robots", International Journal of Robotics Research. Vol 5, No. 1, Spring 1986
71. Kirby, M. and Sirovich, L. "Application of the Karhunen-Loeve procedure for the characterization of human faces". IEEE Transactions on Pattern analysis and Machine Intelligence, 1990
72. Koren, Y. and Borenstein, J., "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation", in proc. IEEE International Conference on Robotics and Automation, pp.1398-1404, 1991
73. Kuniyoshi Y., Inaba M. and Inoue H., "Learning by watching - extracting reusable task knowledge from visual observation of human-performance", IEEE Transactions On Robotics And Automation, 10 (6): 799-822, December 1994

74. Laird, J. E., van Lent, M., "Interactive Computer Games: Human-Level AI's Killer Application", in Proc. AAAI 2000, pp. 1171-1178.
75. Laird, J.E., "Using a Computer Game to develop advanced AI", IEEE Computer, pages 70 - 75, July 2001.
76. Laird, J.E., "It Knows What You're Going To Do: Adding Anticipation to a Quakebot", in Proc. Fifth International Conference on Autonomous Agents, 2000
77. Laird, J.E., "The Soar Tutorial, Part VI", <http://sitemaker.umich.edu/soar>
78. Laird, J.E. and Duchy, J.C., "Creating Human-like Synthetic Characters with Multiple Skill Levels: A Case Study using the SOAR Quakebot", AAAI Fall Symposium: Simulating Human Agents, 2000
79. Livingstone, D. "Turing's test and believable AI in games", ACM Computers in Entertainment (CIE) Vol. 4, Issue 1, 2006
80. MacNamee, B. "Proactive Persistent Agents: Using Situational Intelligence to create Support Characters in Character-Centric Computer Games", PhD Dissertation, Dept. of Computer Science, University of Dublin, Ireland, 2004
81. Mataric, M.J. "Sensory-motor primitives as a basis for imitation: Linking perception to action and biology to robotics", in C. Nehaniv and K. Dautenhahn, editors, Imitation in Animals and Artifacts, MIT Press, 1999
82. Mataric, M.J., Jenkins, O.C., Fod, A. and Zordan, V.B "Control and Imitation in Humanoids", AAAI Fall Symposium on Simulating Human Agents, 2000
83. Mathworks Matlab, <http://www.mathworks.com>
84. Mealy, G.H. "A Method for Synthesizing Sequential Circuits", Bell System Tech. J. 34 p1045-1079, 1955

85. Meltzoff, A. and Moore, M.K. "Explaining facial imitation: a theoretical model", *Early Development and parenting*, 6:2, pp. 157.1-14, 1997
86. Meltzoff, A. N. and Moore, M. K. "Imitation of facial and manual gestures by human neonates". *Science*, 198, 74-8, 1977.
87. Minsky, M.L. and Papert, S.A. "Perceptrons" (1969), MIT Press
88. Molineaux, M. and Aha, D. "TIELT: A testbed for gaming environments", *proceedings of the 16th National Conference on Artificial Intelligence*, pp. 1690-1691, 2005
89. Moore, E F. "Gedanken-experiments on Sequential Machines", *Automata Studies, Annals of Mathematical Studies* (34) p129–153. Princeton University Press, 1956
90. Nakanishi, J., Morimoto, J., Endo, G., Schaal, S., and Kawato, M. "Learning from demonstration and adaptation of biped locomotion with dynamical movement primitives", *Workshop on Robot Learning by Demonstration, IEEE International Conference on Intelligent Robots and Systems*, 2003
91. Naraeyek, A. "Computer Games - Boon or Bane for AI Research?", *Künstliche Intelligenz*, pages 43-44, February 2004
92. Nechyba, M. C. and Xu, Y. "Human skill transfer: neural networks as learners and teachers", *IEEE/RSJ International Conference on Intelligence Robots and Systems*, pp. 314-319, 1995.
93. Nieuwenhuisen D., Kamphuis A., Mooijekind M., and Overmars M.H. "Automatic construction of roadmaps for path planning in games", in *proc. Int'l Conf. on Computer Games: Artificial Intelligence, Design and Education (CGAIDE)*, pp285-292 2004.
94. Norling, E. and Sonenberg, L. "Creating Interactive Characters with BDI Agents", *Australian Workshop on Interactive Entertainment*, 2004

95. Noy, L. "Movement Imitation: Linking Perception and Action", Weizmann Institute, Faculty of Mathematics and Computer Science, 2004
96. Overmars, M.H "Path Planning for Games", in Proc. International Game Design and Technology Workshop and Conference (GDTW 05) ACM, 2005.
97. Oztop E. and Arbib, M.A., "Schema design and implementation of the grasp-related mirror neuron system", *Biological Cybernetics* 87(2): pp. 116-140, 2002
98. Pain, S. "Culture Shock", *The New Scientist*, Issue 2283, 2001
99. Pellegrino, G. di, Fadiga, L., Fogassi, L., Gallese, V. and Rizzolatti, G. "Understanding motor events: a neurophysiological study", *Exp. Brain Res.*, 91, 176-80, 1992
100. Piaget, J. "Play, Dreams, and Imitation in Childhood", New York: Norton, 1962
101. Ponsen, M. and Spronck, P., "Improving Adaptive game AI with Evolutionary Learning", in *proc Computer Games: Artificial Intelligence, Design and Education (CGAIDE)* pp. 389–396, 2004
102. Rabin S. ed. "AI Game Programming Wisdom" (Chapter 10), CRM, 2002.
103. Rabin S. ed. "AI Game Programming Wisdom" (Chapter 2), CRM, 2002.
104. Rabin S. ed. "AI Game Programming Wisdom" (Chapters 4.5 & 5.1), CRM, 2002.
105. Rabiner, L.R. "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. of IEEE*, vol. 77, no. 2, pp. 257--286, February 1989
106. Rao, R.P.N., Shon, A.P. and Meltzoff, A.N. "A Bayesian Model of Imitation in Infants and Robots", in K. Dautenhahn and C. Nehaniv, editors, *Imitation and Social Learning in Robots, Humans, and Animals: Behavioural, Social and Communicative Dimensions*. Cambridge University Press, 2004.

107. Rizzolatti G., Fadiga L., Matelli M., Bettinardi V., Paulesu E., Perani D. and Fazio F.,  
“Localization of grasp representations in human by PET: 1. Observation versus execution”,  
Experimental Brain Research 111, pp. 246-252, 1996
108. Rizzolatti, G. and Arbib, M. A. “Language within our grasp”, Trends in Neurosciences,  
21(5):188 – 194, 1998
109. Romanes, G. J. and Darwin, C. “Mental evolution in Animals”, D. Appleton & Co, 1884
110. “Rubber-band AI”, <http://www.nationmaster.com/encyclopedia/Rubberband-effect>,  
[http://en.wikipedia.org/wiki/Game\\_artificial\\_intelligence#Cheating\\_AI](http://en.wikipedia.org/wiki/Game_artificial_intelligence#Cheating_AI)
111. McGlinchey, S., and Livingstone, D. “What Believability Testing Can Tell Us”, in proc.  
Int’l Conference on Computer Games, Artificial Intelligence, Design and Education  
(CGAIDE), pg273, 2004
112. Schaal S., Ijspeert A.J. and Billard, A. "Computational Approaches to Motor Learning by  
Imitation", Philosophical Transactions: Biological Sciences, Vol. 358 No. 1431, pp. 537-  
547, 2003
113. Schaal, S. “Is imitation learning the route to humanoid robots?”, Trends in Cognitive  
Sciences, 3(6):233-242, 1999
114. Schaal, S., “Movement planning and imitation by shaping nonlinear attractors”, in proc.  
12th Yale Workshop On Adaptive And Learning Systems, 2003
115. Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. “Learning movement primitives”, in  
proc. International Symposium on Robotics Research (ISRR 2003), Springer, 2004.
116. Searle, J. "Minds, Brains, and Programs," Behavioral and Brain Sciences 3, 417-457, 1980
117. Sklar, E., Blair, A.D., Funes, P. and Pollack, J. “Training Intelligent Agents Using Human  
Internet Data”, in Proc. 1<sup>st</sup> Asia-Pacific Conference on Intelligent Agent Technology, 1999

118. Solinger D., Ehlert P. and Rothkranz L., "A Flight Simulator Dogfight Agent for Real-Time Decision-Making in the Cockpit", in proc. CGAMES '05, pp. 129-133, 2005
119. Spronck P., Sprinkhuizen-Kuyper I. and Postma E., "Online Adaptation of Game Opponent AI in Simulation and in Practice", in proc. 4th Int'l Conference on Intelligent Games and Simulation (GAME-ON) 2003
120. Spronck P., Sprinkhuizen-Kuyper I. and Postma E.: "Improving Opponent Intelligence through Machine Learning", 3rd Int. Conf. on Intelligent Games and Simulation, 2002.
121. Stentz, A., "Optimal and Efficient Path Planning for Partially-Known Environments", in Proc. IEEE International Conference on Robotics and Automation, May 1994
122. Sukthankar, G. and Sycara, K. "Automatic Recognition of Human Team Behaviors", Modeling Others from Observations, Workshop at IJCAI05, 2005
123. Swartzlander, B., "Quake 2 Bot Core", <http://www.telefragged.com/Q2BotCore/>
124. Tenenbaum, J.B., de Silva, V. and Langford, J.C "A global geometric framework for nonlinear dimensionality reduction", Science 290: 2319-2323, 2000
125. Tesauro, G. "TD-Gammon, a self-teaching backgammon program achieves master-level play", Neural Computation, 6, 215-219, 1994
126. Thorndike, E. L. "Animal Intelligence", New York: Macmillan, 1911
127. Thoroughman, KA, and Shadmehr, R "Learning of action through adaptive combination of motor primitives", Nature 407: 742-747, 2000
128. Thorpe, W.H. "Learning and Instinct in Animals", London, Methuen, 1956/1963
129. Thureau, C., "Behavior Acquisition in Artificial Agents", Ph.D., Bielefeld University, 2006
130. Thureau, C., Paczian, T. and Bauckhage, C. "Is Bayesian Imitation Learning the Route to Believable Gamebots?", in Proc. GAME-ON North America, pages 3–9, 2005



131. Thureau, C., Bauckhage, C. and Sagerer, G. "Learning Humanlike Movement Behaviour for Computer Games", in Proc. 8<sup>th</sup> Intl. Conf. on the Simulation of Adaptive Behaviour, 2004
132. Thureau, C., Bauckhage, C. and Sagerer, G. "Synthesising Movement for Computer Games", in Pattern Recognition, ed CE Rasmussen, HH Bulthoff et al, LCNS 3175, Springer, 2004
133. Turing, A.M. "Computing machinery and intelligence", *Mind* 59, 433-460, 1950
134. Waveren, J.M.P. van, Q2 Gladiator bot, <http://leesvoer.com/zooi/projects/gladiator/>
135. Waveren, J.M.P. van, "The Quake III Arena Bot", Master's thesis, TU Delft, June 2001
136. Wallace, SA and Laird, JE: "Behavior Bounding: Toward Effective Comparisons of Agents & Humans", in proc. IJCAI 2003, pages 727-732
137. Warshall, S. "A theorem on Boolean matrices" *ACM* 9 (1): 11-12, 1962
138. Wolpert, D. and Kawato, M., "Multiple paired forward and inverse models for motor control", *Neural Networks* 11, 1317-1329, 1998
139. Wolpert, D.M., Miall, R.C., Kawato, M., "Internal models in the cerebellum", *Trends in Cognitive Sciences*, Vol. 2, No. 9, pp. 338-347, 1998
140. Yannakakis, G. and Hallam, J. "Evolving Opponents for Interesting Interactive Computer Games", 8th Int'l Conference on the Simulation of Adaptive Behavior, p499-508, 2004
141. Yannakakis, G. and Hallam, J., "Interactive Opponents Generate Interesting Games", in proc. International Conference on Computer Games: Artificial Intelligence, Design and Education, pp. 240-247, 2004
142. Zanetti S. and El Rhalibi, A. "Machine learning techniques for FPS in Q[uake]3", *Advances in Computer Entertainment Technology* pp. 239-244, 2004