A Layered AI Architecture for Team Based First Person Shooter Video Games

Phil Mike Graham

Doctor of Philosophy
Centre for Intelligent Systems and their Applications
School of Informatics
University of Edinburgh
2011

Abstract

In this thesis an architecture, similar to subsumption architectures, is presented which uses low level behaviour modules, based on combinations of machine learning techniques, to create teams of autonomous agents cooperating via shared plans for interaction. The purpose of this is to perform effective single plan execution within multiple scenarios, using a modern team based first person shooter video game as the domain and visualiser. The main focus is showing that through basic machine learning mechanisms, applied in a multi-agent setting on sparse data, plans can be executed on game levels of varying size and shape without sacrificing team goals. It is also shown how different team members can perform locally sub-optimal operations which contribute to a globally better strategy by adding exploration data to the machine learning mechanisms. This contributes to the reinforcement learning problem of exploration versus exploitation, from a multi-agent perspective.

Acknowledgements

I would like to acknowledge first and foremost my supervisor Dave Robertson, without whom I would not have had the opportunity to continue my studies in this area. I would also like to thank my family and friends for their never ending support. Lastly this work was performed with funding from ERPSC studentship: EP /P503795 /1 and for this I am deeply in gratitude.

Declaration

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

(*Phil Mike Graham*)

Table of Contents

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Overview

This thesis presents an architecture for controlling teams of artificial game players ("bots") in a complex adversarial 3D environment. The architecture contains three layers comprising machine learning, behaviour modules and team based planning. Although our system is similar to others, for similar domains, it is novel in its utilisation of on-line learning during games and multiple coordinated agents. It is these two factors which represent the most important attributes of our system.

We show that by allowing the behaviour modules to adapt, rather than changing the strategy at the high level (The standard approach to similar problems in dynamic planning [92]), a system is created for robust strategy execution across multiple different instantiations of the domain which can alter its play during games. We also argue that working in this way allows us to create strategies which can be considered for a particular game type in the absence of specifics about the details of the match e.g. the size of the environment.

The exploration versus exploitation problem [105] is presented both as an example of the type of issue which can be tackled using our system in a way not before possible, for our type of domain, and also as part of our approach to applying our architecture to the given domain. The angle presented is that instead of varying the amount of exploration/exploitation at the individual bot level, the strategy type can be varied across the team using different team members for the different types of activities. A direct comparison is made to the ε-greedy technique[105] to determine baseline benchmark performance.

1.1.1 Hypothesis and Claims

The following claims are made:

1. Stable cross match performance of a (statistically significantly) higher than baseline standard, in a complex, real-time, adversarial domain can be achieved using combinations of machine learning techniques, to learn about dynamic environmental factors within the domain.

2. A layered heterogeneous architecture, based on behaviour modules, is an effective way to combine these machine learning techniques to increase performance.

3. Communication can be used to control the learning rates of a team of multiple bots while they act in the domain environment.

4. Communication can be used to control the behaviours of a team of bots using a simple, easily understandable, multi-agent plan.

5. It is more effective to have bots in multiple different, adaptive, roles with varying degrees of learning and exploitation than to have single homogeneous bots learning locally.

1.2 Thesis Guide

The thesis is split into 14 chapters, clustered as 4 parts, which are described below.

1.2.1 Part 1: Introduction and Background

In part 1 we cover the basics of the system we developed as well as reviewing the current state of the art in similar systems to determine the overall contribution to the field.

Chapter 1: Introduction

Here we present a basic overview of the system constructed along with the thesis guide.

Chapter 2: The Domain

We present a description of the domain chosen and provide details of all the data-types. This chapter covers areas such as level of abstraction of data, environmental guides and descriptions of the operations possible within the domain.

Chapter 3: Literature Review

This contains a review of the current state-of-the-art in related systems. The focus is on recent developments in the field of adaptive game AI and multi-agent machine learning but some references to work from other fields, related in less direct ways, are also presented.

Chapter 4: Introduction Summary

In this chapter we present a big picture viewpoint on the system as a whole and its place within the general context of machine learning systems for video games. We detail the assumptions that we make with our system and outline its usefullness for both the general scientific community and specifically for games developers. This acts as both a summary of the introductory chapter and as an anchor for the thesis before we deal with the system details.

1.2.2 Part 2: Methodology

In part 2 we look at the system constructed in detail. This begins with a look at the overall architecture and then deals with each layer in turn.

Chapter 5: System Architecture

The chapter begins with a set of motivational experiments performed to give a domain specific motivation for the type of architecture constructed. It then proceeds to describe the architecture in detail giving descriptions of each of the three layers.

Chapter 6: Learning Mechanisms and Techniques: Introduction

A motivational chapter providing some background information on the machine learning techniques used along with the general approach taken to modelling.

Chapter 7: Learning Mechanisms and Techniques

The Learning Mechanisms chapter contains discussion, implementation and evaluation details for the various machine learning techniques that were considered for use in the system. The different techniques are explained and, where possible, evaluated separately in a modular fashion. We show how these could be used in the Unreal Tournament (UT) domain. At the end of the chapter we draw the work together, showing which models were used.

Chapter 8: Behaviour Modules

In this chapter we construct behaviour modules using the machine learning techniques of chapter 3. The modules are presented as pseudo-code giving reference to which machine learning models were used to construct them along with a brief description of expected behaviour.

Chapter 9: Team Strategies

In this chapter we construct team strategies using the behaviour modules of chapter 4. The strategies are presented as code, with a description and supplementary notes.

1.2.3 Part 3: Evaluation and Testing

In part 3 the system is evaluated and tested within the domain.

Chapter 10: Testing

The team strategies are tested to evaluate performance in the three game-types. The aim of this chapter is to show that the system created can execute understandable abstract strategies across multiple scenarios without large deviations in performance.

Chapter 11: Component Testing

A further stage of component testing was performed to determine which elements of the system were responsible for performance. We present the results and discuss the models used and their effectiveness.

Chapter 12: Efficiency Testing

In this chapter we perform an experiment to help suggest at what point the system becomes stable during the reinforcement learning process. This helps to show the effeciency level of the system.

Chapter 13: Exploration Versus Exploitation

One of the key themes throughout this work is the reinforcement learning problem, exploitation versus exploration. Due to the novel approach taken some tests were performed against a classical RL formulation to give a reference point and evaluate the contribution to this area. This also represents a test of the architecture to deal with a known problem in an effective way.

1.2.4 Part 4: Conclusions

In part 4 the work is brought together with conclusions and final remarks.

Chapter 14: Alternatives

In this chapter some possible alternative solutions are discussed. We present some arguments as to why these solutions were not adopted.

Chapter 15: Further Work

The thesis is concluded with a discussion of possible avenues of future research. The main concentration is on extending the work to yet further un-constrained domain areas but also on some more minor avenues of research pointed to by the machine learning section of the thesis along with changes which could be made to optimise the current system.

Chapter 16: Conclusions and Contributions

Conclusions regarding the work are presented in this chapter along with discussion showing how the work presented contributes to the fields of multi-agent machine learning, multi-agent architectures, game AI design, multi-agent planning and reinforcement learning. This takes the form of results extrapolation and a summery of why the work presented validates the claims made in part 1. Some analogies are drawn with the non-virtual world and how the system may be adapted to facilitate implementation in other domains. This chapter draws to a close all of the ideas presented and how they interact.

1.2.5 Chronology

The work presented was performed in quite an unusual way (for work in this type of domain). Having chosen a class of domain, and examined the problems this domain presented, the system was largely prototyped bottom up. The machine learning layer (1) was built before the behaviour modules layer (2) and then finally the team strategies layer (3) was developed on top of both of them. This follows the typical type of engineering which is carried out in the development of subsumption systems where lower layers in the architecture are fixed before higher (typically more abstract) behavioural layers are constructed. Although common for subsumption types of system the general approach presented here is well suited to any type of hierarchical system which depends on lower levels for higher level performance.

Because we had an initial skeleton of the system to work with, before we began (from [40]), it was possible to use some elements of layers 2 and 3 to test components in layer 1 but in general this was kept to the bare minimum until higher layer development was considered.

As such the methodology part of this thesis follows roughly the same time-line as the development did, the machine learning techniques are examined, then the behaviour modules are created, and finally the full strategies are designed based around these modules. As a consequence part 2 reads far more chronologically and offers an insight into the type of prototypical development used when working with a system such as ours.

The reason this process is worth mentioning is that part of the contribution of this work is to show not only the benefits and disadvantages of a system such as ours but also how to develop, design, debug, test and evaluate it in a meaningful and effective manner:

> “...few articles document the development processes used in obtaining the results. Discussing project development processes can aid groups in other universities to conduct applied artificial intelligence research projects with video game test beds successfully”[39]

As such we have included a narrative device throughout this thesis which details the thoughts, engineering challenges and general design decisions which were made at points of interest. These are contained in the following type of box:

Engineering Box

An example narrative box

1.2.6 Experimentation

Our general strategy for evaluating the performance of any given technique was to test it in a game scenario. As such there is a large amount of empirical testing data in this thesis, particularly Chapter 2. The purpose of this is to provide a body of evidence to suggest which elements of the system work and which do not.

1.2.7 Bots vs. Agents

Throughout this thesis the word *bot* is used to describe an artificial UT game player controlled by a decision process. This term is generally used when talking about a concrete example in the domain such as a description of where a bot might be running to in a game scenario. When we discuss the abstraction of this away from UT the term *agent* is used. We have tried to maintain this distinction between the two terms.

Chapter 2

The Domain

As the level of tasks being considered by modern AI systems increases, so too does the need for realistic virtual domains and visualisation systems. Rather than developing systems for visualisation for specified tasks, many researchers are now turning to video games as reasonable approximations to complicated real world problems.

To test our system we chose the Unreal Tournament(UT)[1] game. In principle any domain with similar characteristics (which are discussed in a further section) could have been used to test our system. We chose ours because of the ease of visualisation, configuration and the availability of an interface with the requisite level of generality (i.e. can be used with any TCP/IP socket enabled language). The decision was also based on the fact that other researchers were using UT for their systems showing it as a valid test domain in terms of complexity and realism.

UT is a chaotic, sophisticated, first person shooter (FPS) game for the PC. Game matches are played in various different 3D arenas, which differ in size and shape, using a variety of special "pickups" (which alter some property or characteristic of the player) within the game environment.

There are 3 main game-types which we will consider, each one offering slightly different characteristics and team goals. In all game-types teams of bots, situated in the 3D world, play against each other with the goal of winning the match. The criteria for winning the match is achieving a goal score (which is achieved in different ways for each game-type). The game is adversarial as there are other bots in the environment trying to win the game for their own team.

[1]The 2004 edition of the game was used, which differs significantly from the original version

2.1 Domain Instances

One of our main claims is that

> Stable cross match performance of a (statistically significantly) higher than baseline standard, in a complex, real-time, adversarial domain can be achieved using combinations of machine learning techniques, to learn about dynamic environmental factors within the domain.

With UT as our domain, single matches represent a domain instantiation. Therefore the following sub-sections show the domain components which can change on a per match basis and thus the environmental elements which it is viable to build models for.

2.1.1 Levels

The game levels differ along the following dimensions:-

Size of level Each level has a different size in terms of 3D spatial units. This effects the distance which bots need to travel and the distance between pickups.

Shape of level Levels can often have many different compartments and areas which are linked by smaller corridors or tunnels. In outdoor levels the areas are typically much larger. Abstracting away from the UT decoration these levels can be thought of as basic polygonal environments.

Obstacles There are a large number of obstacles which are often artefacts of the level design and structure. For instance in forest levels one might expect to find trees which cannot be passed through, often obscuring bot vision, or areas of the level which may be hazardous such as lava pits or water areas. We do not directly model these obstacles, as they are not included in the level description, but they effect play so our modelling will likely be affected by their presence.

Movement Devices On some levels there are jump spots, lifts and teleporters which are used to navigate the level. These affect some of the path finding issues and are dealt with in later sections.

Pickups The numerosity, placement and even existence of each pickup differs between levels. Pickups are described in more depth in a further sub-section.

Navigation Points The representation of the level is a set of 3D navigation point positions. These are placed in the level, at the point of creation, to allow bots to navigate. If they are used correctly in conjunction with the in-built game pathfinder they can allow bots to move around while avoiding obstacles.

Each level that we used in testing can be described in terms of these parameters and as such we have listed these here.

2.1.2 Pickups

Throughout levels there are various pick-ups which fall into the following categories:-

Health Pickups These can either be small, which only give a boost of 1-5 health points, or larger, which give a health boost of larger values, typically in the 50-100 region. Bots have 100 health points as standard at the start of a match. Having 0 health leads to a death.

Weapon Pickups These can either be weapons or ammunition (ammo) for weapons. Weapons are the main method of fighting adversaries.

2.1.2.1 Weapons

Each weapon has a different characteristic. Because we deal with the modelling of these in chapter 2 a guide to weapons is included below:

Shield Gun The shield gun is the default weapon which the player uses when they have no ammunition for other weapons. It does not do the enemy damage but offers the player a small amount of protective shield time.

Assault Rifle The assault rifle is the basic weapon that all players start each life with. It has a medium rate of fire but low accuracy and damage ratings. Generally it is always better to play with any other weapon using the assault as a default when other weapons cannot be obtained.

Link Gun The link gun is basically a more damaging version of the assault rifle.

Bio Rifle The bio rifle spreads blobs of damaging fluid when fired. These sit around until another player walks into them, causing health damage.

Rocket Launcher The rocket launcher fires rockets at a relatively slow rate. Rockets are very powerful but have a low accuracy rating.

Minigun The minigun is a rapid fire weapon but with a lower accuracy then an assault rifle. Once it begins firing the rate of fire is rapid but it takes a second or so to begin firing while the gun warms up thus the rate of fire is classed as medium/high.

Shock Rifle The shock rifle is like a rapid fire lightning gun but does less damage with each shot.

Lightening Gun/Sniper Rifle These are single shot weapons with a high damage rate. Both have slow firing rates because of the single shot nature and long reload times.

Flak Cannon The flak cannon fires a barrel of shots and has a wide range spread and medium firing rate. Shots can also be bounced off walls but the resulting angles of reflection can be hard to predict.

The weapons can be categorised via rate of fire, accuracy and amount of damage typically caused. Figure 2.1 shows how the weapons rate in each of these categories against a scale of 0 to 5. The scale is not an exact measure but is a good indicator of how the weapons compare relatively to each other.

2.1.3 Game-Types

The 3 game-types are *team death-match* (TDM), *capture the flag* (CTF) and *double domination* (DD):-

Team Death-Match Each team of bots must kill members of the opposing team. Each death results in the killer's team gaining a point. When a pre-set number of points is achieved by any team they win the match. If a team member kills themself then their team loses a point. After dying team members respawn.

Capture the Flag Each team has a flag which is situated at their base. To score a point a team must capture the other team's flag, by picking it up, and then return it to their own base. Should a flag carrier be killed the flag will then be dropped. If another member of the attacking team picks up the flag they can continue the

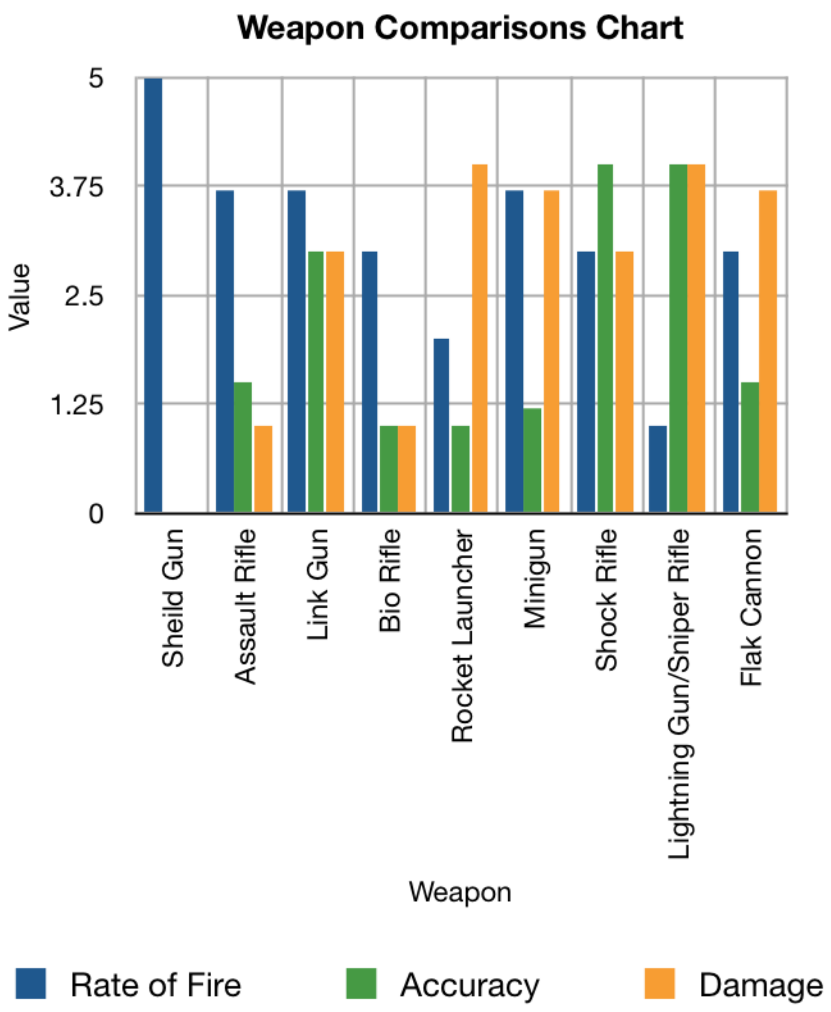


Figure 2.1: Weapon Categories

flag *run* and try to get it back to their base. Should the team who own the flag pick it up it will be automatically returned to their base and no point scored.

Double Domination There are two pre-set *domination points* in the level which at any one time can be controlled by either team. To score a point a team must control both of these points for 10 seconds. Control over the points is achieved by tagging them. Once a point is tagged it stays in that state until it is tagged by another team.

The most importance distinction between the three game-types is that they each present a different challenge.

TDM is purely about confrontation. Winning the match in the other two game types is only indirectly linked to successfulness in confrontations but in this game-type winning confrontations is the only way to win the match. As such decisions must be made about when to fight enemies, where the enemies may be, what best to collect when not engaging in fights to increase chances and the whereabouts of other team-mates.

DD presents a different challenge as it involves guarding a set target area for a specific length of time. Thus decisions must be made prior to approaching the target to maximise the likelihood of success. This also involves decisions about the best way to approach the target, and how many players to assign to each of the two targets. Thus the most important goal becomes not the enemy but the target.

CTF is the most complex game-type. There is again a target which is not the enemy but this time the target could be moving. The game involves attack to gain the flag but also defence of the player carrying the flag while they try to return it to the required base. This must be balanced with the goal of ensuring that no enemy obtains the defending team's flag and when they do attempts must then be made to regain it.

The factor which defines the order of difficulty of the game-types is the number of goals which must be traded against each other to win the match. CTF has the most goals, DD less and then TDM the least.

2.1.4 Levels Guide

The levels of the game can be roughly categorised by their characteristics. In the following sub-sections the levels have been classified via size and weapons criteria. Brief descriptions of the individual levels are included in the appendix.

2.1.4.1 Weapons

Not all maps feature the same selection of weapons. Table 2.2 shows the available weapons which are defined by the classes A to L.

Weapons	Class
Assault, Link, Minigun, Shock, Flak, Bio, Lightning, Rockets	A
Assault, Link, Minigun, Shock, Flak	B
Assault, Link, Minigun, Shock, Flak, Rockets	C
Assault, Link, Minigun, Shock, Flak, Lightning, Rockets	D
Assault, Link, Minigun, Shock, Flak, Bio, Rockets	E
Assault, Link, Shock, Flak, Bio, Lightning, Rockets	F
Assault, Link, Shock, Flak, Lightning	G
Assault, Lightning, Rockets	H
Assault, Link, Minigun, Shock, Bio, Lightning, Rockets	I
Assault, Link, Shock, Flak, Lightning, Rockets	J
Assault, Minigun, Shock, Bio, Lightning, Rockets	K
Assault, Minigun, Shock, Lightning, Rockets	L

Figure 2.2: Weapon Classes

2.1.4.2 TDM

The TDM maps tend to have less directional focus than those for the other game-types. They are also commonly smaller.

Levels	Class	Size
Training Day, Gael	A	Small
Gestalt, Oceanic, Spirit, Trite, Leviathan, Crash, Idoma	B	Small to Medium
Asbestos, Mixer, Sulphur, Ironic, Squader, Rragir Corrugation, Phobos2, Desolation, Deck17	C	Medium
Junkyard, Kakori Forest	D	Medium to Large
Osiris2, Desert Isle	E	Large

Figure 2.3: TDM Level Size Classes

Map	Weapon Class
Asbestos,Mixer,Osiris2,Ironic,Corrugation, Leviathan,Kakori Forest,Trite,Crash,Idoma, Desert Isle	A
Training Day	B
Junkyard, Desolation	C
Gestalt	D
Oceanic,Squader,Rragir,Spirit,Phobos2	E
Deck17	F
Sulphur	G
Gael	H

Figure 2.4: TDM Map Weapon Classes

2.1.4.3 CTF

Many of the CTF maps follow a similar design theme and as such differ mainly on the axis of level size and weapon placement.

The general pattern is to have a flag base at either end of a largely symmetrical level with a single or dual access pathway between the two bases.

Levels	Class	Size
Maul	A	Small
Orbital2, Geothermal	B	Small to Medium
Face Classic, Citadel, Lost Faith, Chrome	C	Medium
Absolute Zero, Face3, Magma	D	Medium to Large
Moon Dragon, December, Double Damage Twin Tombs, Smote, Grassy Knoll	E	Large

Figure 2.5: CTF Level Size Classes

2.1.4.4 DD

The DD maps maps tend to be larger as is evident by the lack os a small maps class. This is mainly to induce the use of strategies required to get from point A to point B. If the points were too close together this could over simplify the game-play.

Map	Weapon Class
Absolute Zero, Double Damage, Lost Faith, Grassy Knoll, Chrome, Twin Tombs, Orbital2, Smote	A
Maul, Moon Dragon, Face3, Geothermal	D
Magma	E
December	I
Face Classic	J
Citadel	K

Figure 2.6: CTF Map Weapon Classes

Levels	Class	Size
Scorched Earth	B	Small to Medium
Atlantis	C	Medium
Renascent, Outrigger, Sepukku Gorge	D	Medium to Large
Sun Temple, Ruination	E	Large

Figure 2.7: DD Level Size Classes

Map	Weapon Class
Sun Temple, Ruination, Sepukku Gorge	A
Scorched Earth, Outrigger	C
Renascent	D
Atlantis	L

Figure 2.8: DD Map Weapon Classes

Chapter 3

Literature Review

In this chapter we deal with previous work related to our chosen area of research. Our focus is on fully related systems and large scale properties of related architectures. Obviously it is vital that we research each individual technique used in our learning modules as-well but these are considered separate studies and can be found inside the methodology part of this thesis alongside our studies and reports concerning their use.

3.1 Direct Previous Work

The starting point for this work is the system implemented in [40]. The project represented the first extensive system for UT which used a largely communication based multi-agent system to control game-playing bots in teams.

In the system we created a framework to allow the formation of simple plans to perform complicated game playing within the chaotic game environment. These plans were created in Lightweight Co-ordination Calculus (LCC) [85] and made by hand using only intuition about the game-type, no automatic plan generation was performed and no machine learning or strategy adaptation was implemented. The system worked in a reactive manner and deviated from the traditional approach to LCC parsing[2, 85]. At each game-step one clause was selected from the strategy, on the basis of agent role and situation constraints, to serve as a rule telling the agent which action to perform. This was repeated 4-5 times a second.

The problem was that the execution of these LCC plans was largely invariant of level type, size and individual scenario such as location of enemy etc. The bots could only react to circumstances which were encoded within the plans as constraints and as such these had to be pre-empted during strategy creation. This led to strategies which were largely predictable and inflexible. The results showed that the system achieved acceptable performance on certain levels of the game but cast doubt on their use outside of the specific scenario for which they were designed.

The main reason for this was that often the areas of the game which did not provide such consistent reactive information would lead to less useful/interesting play. For instance, when the bot was in locality of the flag and domination point areas very detailed control could be used to determine different roles and modes of behaviour. However on the larger levels this control did not benefit the bots as much. This pointed to a lack of control in the more open areas of the level where play is less goal orientated and reactive due to there being less to react to, especially when direct enemy presence is sparse. This is largely because the system adopted a reactive approach to LCC which deviated from the notion of classic LCC interpretation in systems such as those discussed in [2, 85]. In short, the system worked very well in areas where there was a large amount of changing information to react to. In areas where little changes between cycles the bots begin to falter in their play, becoming static. They always appear to take the same paths and to a human observer this would allow an easy win via simple counter planning formulation[87].

If the same types of simple high level plans could be formulated which were easily created and understood, in the abstract, then some of the problems could be tackled by improving individual bot's abilities to deal with level specifics. Being able to show that even with different low level, autonomous, play on different game levels and within different scenarios the plan has still been maintained, with regard to its semantics, is key to this.[1] Variability within the bot's actual play is what we are striving for but it is of primary importance that this variable play still fits the original strategy. For instance if a strategy, telling the bot to go to its own flag point, is created then it is not of concern which particular direction it goes or exactly how it gets there, so long as it is verifiable that the bot got to the flag point in all given successful executions.

One way to approach this task is to build knowledge into the system which is based on observations made during the game-play. If the bots are to play a truly non-deterministic game then they must be able to react to different *styles* and *patterns* of play by the opposing team. These cannot always be pre-empted or observed in advance and can be viewed as hidden variables within the game-state.

[1]It should be noted here that we take semantics as a general sense that the actions laid out in the plan and the intuitive understanding of these actions is maintained with respect to the actions of the agents concerned within the plan.

3.2 Motivational Work: Laying the Foundations

As-well as the specifics of the area it is important, prior to performing work in a field, to gain an understanding of the general reasons one is performing work there. In our case why use video games technology at all? What is the purpose when there are already numerous real world domains that could be tackled?

> "Video game technology can provide a rich platform for validating and advancing theoretical AI research"[39]

The quote above from a review paper looking into AI and CI (Computational Intelligence) gives the first clue as to why we use video games as our domain. The key is that we are considering an AI idea in the abstract and then validating the concept within a domain. We are not considering the domain as something critically important in itself but merely something which is suitable for the purposes of testing our ideas. So what is it that makes video games and in particular the type of video game which we consider in this work (First Person Shooter) so applicable for this task?

> "FPS games are vastly cheaper, simpler to handle and faster to run than physical robots, but also more complex and demanding than the toy problems traditionally used in CI research. We argue that such games are good test-beds for research on learning or otherwise developing controllers that perform complex tasks, in the sense of being composed of several simpler tasks."[114]

This sums up quite succinctly the main reasons. With this in mind we must also examine the current systems in play for controlling AI within these systems. After all there is no point trying to develop adaptive systems if those already implemented commercially are effective learners. The same paper has this to say of the traditional bots used in video games:

> "...do not include any form of learning and do no play the game under the same conditions as a human does..."[114]

What this means is that instead of performing machine learning from low level information, bots in games traditionally provide annotated and scripted high level behaviours which can appear, superficially, intelligent. So why take this approach to the problem? Surely a more considered approach would offer a more robust solution and facilitate future developments and changes to the particular game chosen?

> "...current commercial game AI has little to do with current academic research in the AI and CI communities."[114]

The reason for this is largely to do with predictability. Scripted systems with prescribed scenarios and responses to these scenarios can always be predicted in a way not typically associated with learning systems found in AI research. Thus the biggest challenge for our work is to provide a system which is practically applicable to similar domains but which is more robust without sacrificing too much flexibility.

Burkey et al[22] suggest the following in the conclusions section of their paper concerning work in the quake III domain:

> “...by combining traditional AI techniques, a system can be developed that enables choices made by a conventional AI layer to be altered in response to feedback from the actions selected......The results showed interesting trends that indicate that, with more development and testing to determine optimum settings, the system developed could form the basis of a usable adaptable AI system.”

Our system is in essence exactly this, but with multiple-agents, an explicit architecture dealing with agent behaviour and a communication/cooperation mechanism. We show how using these extra elements we can tackle the problems associated with the chosen type of domain more effectively.

In [97] Spronck states as further work both the application of on-line machine learning techniques during game play and also

> “other machine learning techniques in combination with, for instance, subsumption architectures.”

Given the hierarchical nature of the system presented in this thesis, along with its similarity to subsumption architectures, a large part of this work is a response to this idea.

The terminology used in this thesis borrows from the basic grounding presented in [58]. In particular, reference is made to many of the issues described for a multi-agent machine learning system. This work by Kazakov and Kudenko is more angled towards showing the problems of adaptation in all kinds of agents and is aimed at domains such as robotics, but still provides some notion of why and how work in UT is useful (without directly referencing it). The thesis also deviates at the point where [58] describes machine learning from a knowledge based perspective, via inductive logic programming, and instead we employ more statistical based methods for learning about the environment and game-play. In this way our system contains largely *black-box* machine learning concepts where overall representation of the knowledge learned is not easily (without sufficient mathematical translation) interpreted by humans. This is in contrast to *white-box* techniques such as inductive logic programming where the eventual representation, in the form of rules, is more easily understood. We show how some of the techniques presented can be altered in such a way that some intuitions can be incorporated in a meaningful way which increases performance and facilitates better understanding of how the techniques are working.

3.3 Properties of Related Work

To understand fully the contribution of this thesis to the relevant fields, we must first consider similar architectures and their general characteristics. Some of these are positive aspects which can be considered good practice for building systems in this domain while others are negative and it is our goal to show how these can be fixed using our architecture design.

3.3.1 Adaptation

It is generally accepted that adaptability is a pre-requisite for any system to be considered intelligent. Almost all the papers which we have considered in this review are in some sense adaptable as it is the issues arising from adaptability which concern us most.

Many papers exist dealing with systems which are adaptable or can be in some sense considered to learn but there are not so many which exist for domains such as ours and consider a level of data which is similar. There are also none considering the exact set of circumstances and tasks which we tackle. This could be interpreted as implying that our area is too narrow and exclusive, but we argue that the general concept of our architecture is both applicable to similar, but not identical, domains and provides a set of general characteristics defining a profile for applicable domains.

Pieter Spronck's work has provided methods of performing automatic strategy adaptation and dynamic game AI which adapt their performance during a game in many complex domains [98, 94, 95, 92]. Typically his work focuses on role playing games such as Neverwinter Nights or Warcraft, but he has also performed some work in Quake III (A First person shooter game similar to Unreal Tournament)[95]. His work in Quake III used an adaptive team strategy method called TEAM2. Both TEAM and TEAM2 are essentially adaptations of his dynamic scripting method with added communication and evolutionary properties. A large amount of his work uses evolutionary methods alongside dynamic scripting to create intelligent teams of agents.

3.3.2 Symbolic Representations

A trait which is more prevalent amongst papers in the multi-agent learning domain, although maybe less so in the video games learning domain, is the use of high level symbolic representation learning systems.

The work presented here, re-interpreting the system from [40], is largely based on the observations in [58] about how machine learning frameworks can interact in a modular way within a multi-agent environment. In particular this thesis can be seen as a response to the future work section of [58] which states the following as an interesting avenue of research:

> "**More Complex applications:** Most MAL (Multi-agent learning) application domains are relatively simple. It would be interesting to see MAL research for more complex real-world applications. Eventually, such applications would encourage researchers to look beyond Q Learning"[58]

Although some work has been done since then many researchers are working with largely symbolic world representations (A mirror of the more simple worlds being tackled) focusing only on high level behaviours and properties of argumentation protocols and high level belief reasoning[103, 96, 58, 106, 3, 83]. In the domains typically chosen for these types of agent work, symbolic manipulation of the semantics performs well because there are few primitives to reason about which are mostly defined with a rigidly specified notion of truth. For more complex domains this type of reasoning becomes very difficult to work with as to reason only with things which were known to be true becomes a.) largely infeasible and b.) unlikely to yield a strategy which is either understandable or effective in the domain. In complex domains systems often have to work with assumptions based on observations which may or may not be true, or even may not be well grounded.

Spronck's symbolic dynamic scripting paper[96] presents an AI which can react to the opposing players for a sophisticated game environment (Neverwinter Nights) showing that it is still possible to work with symbolic representation in more complex environments provided the symbolics are probabilistically treated in a manner similar to strength values for predicates in statical knowledge base reasoning systems. This said the evaluation of Spronck's system is performed against known scripted opponents and largely in a simpler simulation rather than the full game. The paper presents a way of creating AI by integrating small reactive clauses into a dynamically created script based on experience within the game using each low level behaviour. This appears similar in overall idea to our solution except our overall script is static and allows the low level behaviour modules themselves to be altered via experience.

3.3.2.1 Typical Symbolic Worlds

The UT environment is more complex than the basic scenarios which most of the work in symbolic domains concentrates on, with the exception of [95]. In the symbolic multi-agent systems field in particular the worlds used are traditionally very prescribed, and more general approaches which allow for errors and things to go wrong are not generally considered. Papers are traditionally much more concerned with proving certain properties of interaction[73]. For us concern shifts to verifying the overall plan and that the learned techniques allow the plan to be executed satisfactorily with less focus on more traditional interactions such as argumentation protocols and guarantees of convergence. No work is performed on the process of automatically learning policies for the agents at the high level. Our point of view is that the issues surrounding problems hand-crafting plans for numerous different scenarios can be avoided by tweaking the low level behaviour, via machine learning, enough to allow one predefined plan to be used across multiple scenarios. Our assessment and evaluation, as such, is based much more strongly on agent behaviour rather than the theoretical issues and possibilities for problems within the interaction framework.

3.3.3 Simplified Domains

In general we have tried to deal with architectures which tackle similar domains to ours but there are some exceptions to this where the system being considered tackles domains which appear similar but are in fact simplified[6, 64].

In [108] Thomson and Levine present a neuro-evolution based system for controlling a single agent in a first person shooter game called “Bruce World” based on the die hard films. On the face of it this seems highly similar in terms of architecture (hierarchical planning layers building on trained lower level actuators) but examination of the domain shows that are many simplifications in comparison to UT. The world is 2D, movement is fixed in 4 basic directions at a pre-defined rate, obstacles within the world are sparse and there are only two types of weaponry considered (knives and guns). It is easy to write off these differences as merely articles of implementation, dealt with during prototyping, but as we show throughout this thesis, these minor issues are not always that simply rectified.

3.3.3.1 Different Angles to the Same Domain

Another key difference with a lot of the work considered is that the same domain has been used but has been approached from a different angle or at a different level of abstraction.

Consider the work of Parker et al[74, 75] in the Quake II domain. Superficially the domain looks similar as they consider a modern first-person shooter game with a similar array of weapons and pickups to UT. Their paper however deals with a system for finding and shooting an enemy within a small single room, essentially amounting to a genetic algorithm (GA) applied to an automatic visual servoing task. The system is working from a much lower level using actual pixel data and is concerned with the visual recognition task.

Another example of this is the system presented in [114] for dealing with a single agent for UT. Only a very small subset of the levels are considered (3), only a single weapon is used and only single floors of the levels are considered with lifts and ramps removed. The work also concerns itself with level discovery, assuming the agent has no prior concept of a map or area and is trying to construct one from experience. While this assumption is consistent with a single player in a video game, the assumption that a team of players may have some terrain data prior to a game or some notion of how not to run into walls is consistent with a different set of tasks placed on the learner. That is not to say that our system could not be adapted to incorporate work on domains such as these, in fact part of our argument is that the modularity in our system easily allows this, but it is not our main concern to work at either of these higher or lower levels of data abstraction.

3.3.4 Single Process Models

Most research in the machine learning for video games area utilises one particular technique for the entire AI algorithm and decision procedure. Authors often present systems which can be thought of as one main process or idea rather than being composed of a number of interacting parts[23, 92, 17, 3, 96]. In many cases the limitations of such systems are discussed but it is not common practice to try to compensate for some of the limitations of one particular technique using another. Very often critical theoretical results regarding a technique are cited as major problems without considering that in practice a lot of these theoretical issues can be resolved using a differing technique to mediate the first[57].

In [99] the following fact is stated:

> "If agents are to adapt and change in real-time, a powerful and reliable machine learning method is needed."

This shows precisely the type of single process based thinking which is problematic to advancement in this area.

We now move on to consider two of the most common single-process model types.

3.3.4.1 Neuro-evolution

Without doubt the most popular single process technique for games AI work is evolutionary neural nets (Neuro-Evolution[6, 75, 109, 114, 82, 108]):

> "Artificial Neural Networks (ANNs) combined with Evolutionary Algorithms (EAs) is a popular approach to agent design."[109]

In particular the NEAT system and variations of this are common [99, 93, 39, 69], providing a general framework for neuro-evolutionary systems and a well supported code-base.

One reason for the prevalence of neuro-evolution in the domain could be that in any given genetic algorithm it is widely accepted that the most difficult part of the process is that of problem representation. The weights and connectivity systems found in artificial neural nets lend themself as an elegant solution to this problem due to the ease of designing genetic operators around combinations of these. Because neural networks can then be viewed as universal function approximators this gives a lot of potential power for the GA to work with. The single direction decision surface of the sigmoid activation function for each hidden unit also helps to deal with some of the problems concerning the genetic process exploiting blind avenues in the search process, a problem common to genetic programming.

3.3.4.2 Evolutionary Designs

In recent times the field of game AI design has also seen a relatively large amount of research into other forms of evolutionary agent design[98, 94, 95, 92, 79, 17, 74].

In general these approaches have been either adhoc GA designs with genetic operators for the particular domains or alterations of existing techniques to incorporate an evolutionary process to allow adaptability.

Research has sometimes focused on applications where the problem has lent itself to this type of solution or been part of the game. For instance the game *Black and White* [1] allowed creatures in the game to evolve over time in response to actions of the player, fitting the notion of the player being a god, a central theme in the plot of the game.

3.3.5 Layered Architecture

In many papers the idea of an explicit architecture is not really considered or seen as the work's main contribution. Authors are generally concerned with an algorithm or a method but it is arguable that the architecture which these fit into can be equally if not more important. A good modular architectural concept can allow easy integration of multiple complex components if it is carefully designed and well engineered.

Of the papers which do consider the architecture, layered systems are common[108, 114] with different layers controlling different types of tasks, often at different levels of abstraction. Generally layers are considered to be feed-forward without complicated back and forth interactions, keeping each layer separate. More complex layered architectures involving are less common but do exist.

3.3.5.1 Homogeneity versus heterogeneity

When papers deal with multi-layered systems the layers themselves are either considered homogeneous [107, 114] or heterogeneous. Heterogeneous systems tend to fit more with the modularity paradigm and multiple process models, where as homogeneous systems tend to fall into the single process category and are typically much less modular.

3.3.5.2 Subsumption Architectures

One of the most popular layered architectural designs is the subsumption idea that upper layers can be built upon lower layers and their behaviours. It is very common [107, 114] to use a neural net and fix lower layers in the design while higher layers are trained. Thus the lower layers behaviours and outputs are assumed to perform low level tasks well and the upper layers can then be trained to act upon these low level outputs, giving a higher level of abstracted performance. There are some arguments that this type of training of neural nets actually offers no theoretical benefit over standard training methods such as scaled conjugate gradients but in practice it seems to allow this type of hierarchy to be achieved more easily.

The similarity of our architecture to subsumption architectures is along the same lines as Gat's Atlantis system[38] for controlling autonomous robots, which borrows largely from the subsumption architecture paradigm. Gat's system is designed for adaptive planning and recovery from failure, with most of the concentration on maintaining consistency between the different world representations at each of the three layers. We share a similar view of higher levels dealing with a higher level of abstraction but introduce machine learning to the lowest layer and communication to the top level.

3.3.6 Modularity

Coupled with a layered architecture, or any system involving multiple interacting parts, is the notion of modularity[119]. In a system with multiple layers/models it becomes key to understand how to deal with them effectively. One of the most effective methods currently is to allow the system to be modular, affording the switching in and out of different elements and parts for others in order to perform testing. This then forces the design of the individual components to be more carefully considered.

Although no adaptation takes place, the system presented in [71] for controlling cars in the TORCS racing engine shows a good example of the type of modularity which we are striving for. Individual components control different parts of the car's behaviour with the overall behaviour being a combination of their combined performance, mediated by a fuzzy logic system.

The system presented in [22] shows how multiple techniques can be combined, for a modern FPS video game, without an explicit architecture, instead choosing to integrate machine learning into an already functioning system at the individual component level.

3.3.7 Off-line Learning

One of the strongest themes [15, 74, 37, 119, 64, 75, 114, 108, 93, 107] in work concerning video game learning systems is the use of off-line learning from large banks of data (usually obtained from human players). Systems are then trained before matches or games using this data in order to mimic what is considered intelligent human behaviour.

Although effective, off-line learning does pose a certain problem concerning adaptivity during a game. A system trained before a match begins, although possibly exhibiting intelligent behaviour, is no less static during a match than one which is scripted. The only change is the representation of the policy being put into action. Ideally a system should be able to adapt at game time, during a match.

The following quote shows both the general attitude towards on-line learning for video games AI and why our work in this area is so important to the field:

> "We observe, however, that learning effective behaviour while the game is in progress (i.e.e, 'on-line'), typically requires an inefficiently large number of learning trials. It is not uncommon that a game has finished before effective behaviour could be established, or that game characters in a game do not live long enough to benefit from learning." [113]

In contrast there are systems which allow on-line learning but they are definitely the minority. The system presented in [99] allows on-line adaptivity of a team of agents during a game match using a system known as rtNeat (Real-time evolution of neural networks). The key component of their system is a collection of artificial robotic agents who play in a manner resembling swarm technology. Each robot is controlled by a neural net representing a chromosome and the swarm represents the population. As the match progresses certain agents are removed from the population if their fitness becomes low, replaced by new evolved members. The idea is that as the population itself only changes gradually the effects are not drastic or notched, displaying a smoother transition towards high performance.

The closest work, in terms of on-line learning, is found in Burkey et al's work within the Quake III domain[22]. Their system allowed the on-line adaptation of a weapon selection policy - a task central to our work - for a domain very similar to ours.

Our work shows that on-line learning in a complex domain is in fact both achievable and effective even with only a relatively small adaptive dataset gained throughout the proceedings of a single match. We show some of the changes to the standard type of system which must be made to accommodate this and introduce ways of thinking about different team members that facilitate this type of learning.

3.3.7.1 Knowledge Transfer

When a system employs off-line learning this forces the issue of knowledge transfer [90] between data runs.

> "Transfer Learning is a widely researched topic. It can be defined as using knowledge learned on a source task to improve learning on a target task."[6]

This is especially pertinent with systems which allow adaptation after the initial learning has been performed. We give no concern to knowledge transfer between matches as our goal is to provide robustness from only the learn-able data found during a single match of the game. In chapter 10 we offer a brief discussion on why we didn't consider this as an option.

3.3.8 Single Agent

The single largest contribution which our architecture presents over others in similar domains is the control of multiple agents with a single strategy. Although this is not unique when considering the multi-agent machine learning community, in general it is for the AI games community and the domains typically associated with research in this area.

Generally papers concern the creation of a single, albeit very intelligent, agent [71, 22, 74, 64, 75, 114, 108]. Issues to be resolved typically concern working with as low level data as possible and relevance to real world human situations and what would be available to a human approaching the same task.

Papers with multiple agents in these domains typically employ the technique of creating clever single agents and assuming that, with enough intelligence and autonomy, effective team behaviours will emerge with little or no overall control [99]. Research from the multi-agent learning community in simpler domains show that this type of emergent autonomous team behaviour is difficult to achieve without some form of centralised control or communicative policy. There is evidence from the following quote from Dawson suggesting that there is some understanding of this in the machine learning for games community as-well:

> "Today, formations are expected for any type of cohesive group behaviour. From squad based first-person shooters to sport simulations to real-time strategy games, any time that a group is moving or working together it is expected to do so in an orderly, intelligent fashion"[29]

In [18] Bradley and Hayes try to show how multi-agent reinforcement learning can tackle this problem but the formulation of the solution only allows for application to domains where a state-action pairing and complimentary group utility function can be applied, lowering its generality as a solution. We provide, with our system, a way of tackling multi-agent reinforcement learning which is not tied to any particular representation or machine learning mechanism.

3.4 Summing Up

From the research in this chapter we can see that there is a wealth of crossover between the machine learning, computational intelligence, AI and games development communities. We have seen that there exists a field of work concerned with the testing of AI techniques within video games due to a variety of properties which they posses as a domain.

We have then examined some of the properties and shortcomings of current systems as well as developing a picture of the common types of approach to a system within the field and the given domain types. It is obvious that a system demonstrating on-line adaptivity, multi-agent control, with a well abstracted modular architecture and operating within a complex modern FPS domain would represent a significant advancement of current technologies in the field of games AI.

Chapter 4

Introduction Summary

Having introduced our chosen domain and given some of the basic details and an overview of the system presented in this thesis, we now present a summary of this information. The main purpose of this is to allow the details within the rest of this thesis to sit neatly into a big picture. It also allows the contributions of the work presented to be correctly ascertained from the offset.

4.1 The System

The main scientific contribution of this thesis is an architecture for controlling multiple agents in an adversarial 3D environment (In our case a first person shooter video game). This architecture has 3 layers, each with a seperate function to perform.

The bottom layer in the architecture provides machine learning mechanisms which model the environment. These adapt over the course of a game match based on experience within that match. It is at this layer that all of the statistical learning in our system occurs.

The middle layer consists of sets of behaviour modules which can be switched on and off. These modules are static but many of the low level decisions are informed by the machine learning layer below.

The top layer consists of a team strategy. This informs all the agents within a team of which particular behaviour module they should be running at any given time on the basis of team performance, and a collection of roles within the strategy. Each agent will be fulfilling a certain singular role at any given time.

4.2 Robustness

Beyond the goal of simply performing in the domain our system is designed to fulfil several other goals. The most important of these is robustness across game matches.

In any system dealing with video games there is a level of concern for single match performance and being able to measure how the system deals with an optimum situation. A broader concern than this is the issue of how this performance deviates when circumstances are changed. The general approach in video games AI is to craft individual behaviours for each different circumstance. This means a lot of thought prior to strategy creation and, very often, concern over details which are not really important to the overall viewpoint of the behaviour designer. For instance take the goal of a player trying to get an object from point A to point B. It might be that because of some overall set of game goals we know that the player has to get the object from point A to point B. What we dont really care about is the exact route taken, we only care that this route is in some way good and that the end result is verifiable; the player got to the end point. We care about the end result of high level strategic instructions, but not the particular details.

The way our system meets this goal is to allow the creation of fixed high level strategies which use fixed performance behaviour modules but allow the exact details of how these modules perform their tasks to be fluid, based on learned information. To learn this information we use a set of machine learning components at the lowest level in the architecture. This idea is best illustrated by an example.

Lets say we want to control a team of 3 agents to deliver a package to one side of a level. It might be that we specifically only want two agents to do this and the other to decide when it is best to perform this action. We can encode this top level information as roles in the strategic layer. Thus one of the agents will take a role of controller, and the other two can be package carriers. The controller could send a message based on team performance to say that it is time for the other two to begin the package run. The strategy could also encode that they should take slightly different roles. One of a package carrier and the other of a package carrier helper. Each of these roles could then be attached to a slightly different behaviour module. The package carrier could have a module which picks a path and runs across the level with the package. At the strategic level the package carrier could also send messages to the carrier helper to tell them where they are. The package carrier helper then feeds these messages into its module and follows the first agent.

The flag carrier's module then makes use of the path finder component which uses information about likely enemy position and previous experience, to form the best path.

The behaviour that would be exhibited here is that of two bots moving across the level with a package while a third waits for them. The two bots would form a pairing where one followed the other and they would pick the best path. This path could change entirely from game to game but the overall goal of the strategy itself has still been met. The system has used communication at the strategic level to allow coordinated team performance. The behaviour modules have allowed the creation of complex individual behaviours. It is the machine learning component layer which has added the tweaks to this behaviour that make it adapt to changes across different games. By embedding the adaptive components in the lower level of a hierarchically fixed architecture we create a system which is robust by being adaptable to the details which change from match to match but also verifiable in its behaviour owing to a fixed, hard coded strategy.

4.3 Key Assumptions

In building our system we make the following key assumptions. It is these assumptions which differentiate our system from many others and it these which show why what we have done is of interest to both the scientific community but also, importantly, the game developers community.

4.3.1 Within Trial Adaptivity

The system only learns/adapts within trials. At no point do we consider learning between trials or treating multiple trials as a larger dataset. As such, we do not deal with the issue of knowledge transfer in any practical capacity. There are two main reasons for this decision.

4.3.1.1 The Industry Reason: Frequency of off-line learning systems

As discussed in the literature review section almost all systems for video games currently perform offline learning (otherwise known as between trial learning, or knowledge transfer). This is partly due to an adherance to certain techniques which have this as a pre-requisite but it is also largely to do with predictibility and stability of performance using this method. Game developers in particular are known to favour systems which are inherantly stable due to the main motivating factor of video games development being popularity and marketability. It is difficult to market a system in which the behaviour cannot be predicted for certain circumstances. Many hours are put into making sure that every facet of the game is behaving exactly as specified.

With this in mind we can see that a system for performing online adaptivity which could be not only predicted but controlled and allow the creator to input what they desired of the system at an abstract level would be of very high value.

4.3.1.2 The Scientific Reason: Interest

Coming from a scientific point of view, allowing knowledge transfer and offline learning simplifies the task of performing effectively in our type of domain. This makes such systems less challenging problems.

4.3.2 A Fixed Plan

In our system we fix the high level plan that controls the agents. We also fix the behaviour modules and both are input manually by the system designer. In principle this need not be true, we could in both cases allow these elements to adapt. This would create a system with elements of low level statistical parameter estimation alongside a more traditional automated planning system. The reason we make this assumption is that one of the main criteria for systems such as ours to have, for the video games type of domain, is a layer of controllability. In recognition of this practical consideration, we allow control at the top level and adaptiveness at the bottom level to provide stability and adaptiveness to change.

Allowing the entire system to adapt at every level is more powerful because it lets the system change higher level components such as strategy but it means that the actual performance in any given scenario no longer has any predictability. Although a set of scripts could be setup such as those in Pieter Spronck's dynamic scripting system, it becomes extremely difficult to predict performance or even place any sort of probabilistic estimation on what the team of agents may do.

By allowing only the lower level learning, which we have setup to perform very specific parameter estimation learning tasks, to adapt we create a system which allows the user to specify a particular strategy for a team of agents that can be observed in action. The behaviour specified in the strategy can be verified but it can also be observed that the lower level details are modified in response to the individual agent's experience.

For video games designers this is the ideal trade-off: a system that allows controlled thought out strategies to be encoded while allowing details which can tweak individual scenario performance to be adapted.

4.4 Putting the Details into Place

Because of this use of a fixed architecture design with a fixed plan, the actual details of each individual component within the system are not important for assessing the actual architecture itself. They are important for dealing with the architecture in an applied setting. As such it is important to understand that at any given point in this thesis in which we deal with a particular component or part of the system they are just this, a component chosen for our domain, but not integral to the overall contribution of the architecture. They could easily have been replaced with outher components, modules or plans.

The purpose of testing the system with specific components, modules and plans is to show that for a particular domain (representative of the type of domain which the system can be used for) we can show that performance is acheived which meets the goals of the system design.

4.5 Generalisation

One of the key components of any system for video games is the question of how well does it generalise and most importantly what would be the cost in porting the system to new environments. The ideal scenario is a system which can be ported unchanged from one system to another with zero extra effort. Currently, this is largely impossible. This is mainly due to the fact that there does not exist a single interface to all similar video games. It is this lack of standardisation which leaves us in a less than ideal position.

This leaves two engineering options for porting the system to another similar environment. Re-write all the domain specific parts of the architecture for the new environment or write a piece of middle-ware which sits between our system and the environment, interpreting communications from each side. Of the two, the second option gives a more portable system but the first option provides a more robust solution.

A further reason to consider the full re-write as opposed to the middle-ware solution is that we argue part of the appeal of our system is the fact that strategies and behaviour can be carefully controlled with a hand-written plan but also allowed to adapt. As such these plans are almost bound to be tied into the particular domain chosen, and the way in which they are written will likely reflect this.

With regards to our system, there are certain elements which are portable. The top layer strategies for instance could easily be ported, given matching behaviour modules and a re-writing of predicates to take this into account. The middle layer of the architecture is portable if we have acceptable learning mechanisms at the low level. The low level is not really portable because the particular learning mechanisms are tied, strongly, into the environment itself. This said, if the afforementioned middle-ware layer could be written for novel environments then I see no reason that these could not also be ported without significant changes.

In general, it is not the specific layers here or the details that can be taken to a new system, it is the architecture itself. The concept of 3 layers with the properties described, of how to perform effective online learning, of how to control multiple agents in a game environment through communication.

What follows is a description of how our system was built and the engineering challenges that this created. What should be taken from this is how to go about building your own version of the architecture for your own domain, along with the practical performance benifits and stability that this will likely provide.

Part II

Methodology

Chapter 5

System Architecture

In this section we describe the main contribution of the thesis, the architecture of the system, in more detail. We concentrate on generic features of it that apply across a range of appropriate domains (although we use UT as a concrete example). We present the motivations for this particular design and also draw attention to a few key concepts associated with it.

5.1 Layers

The design is a heterogeneous layered architecture, as shown in Figure 5.1, with three key layers. Each layer can be thought of as distinct in operation, but with information transferred between layers to facilitate smooth operation.

The bottom layer of the architecture contains modular machine learning mechanisms which model elements of the environment. These operate on raw data from the game and are utilised by higher layers. The learning mechanisms are communal and each bot has access to them.

The middle layer contains the behaviour modules. These use data from the machine learning mechanisms in the bottom layer to make decisions which translate to bot actions within the game. Each behaviour module defines a specific state that a bot may adopt. No two modules can be operational within a bot at once and hybridisation of behaviour modules is not supported.

The top layer contains the plan which turns behaviour modules on and off. This controls the dynamic of the team and the roles which members are assigned to. Roles do not directly correlate to behaviour modules but there are connections which could be thought of as a functional mapping.

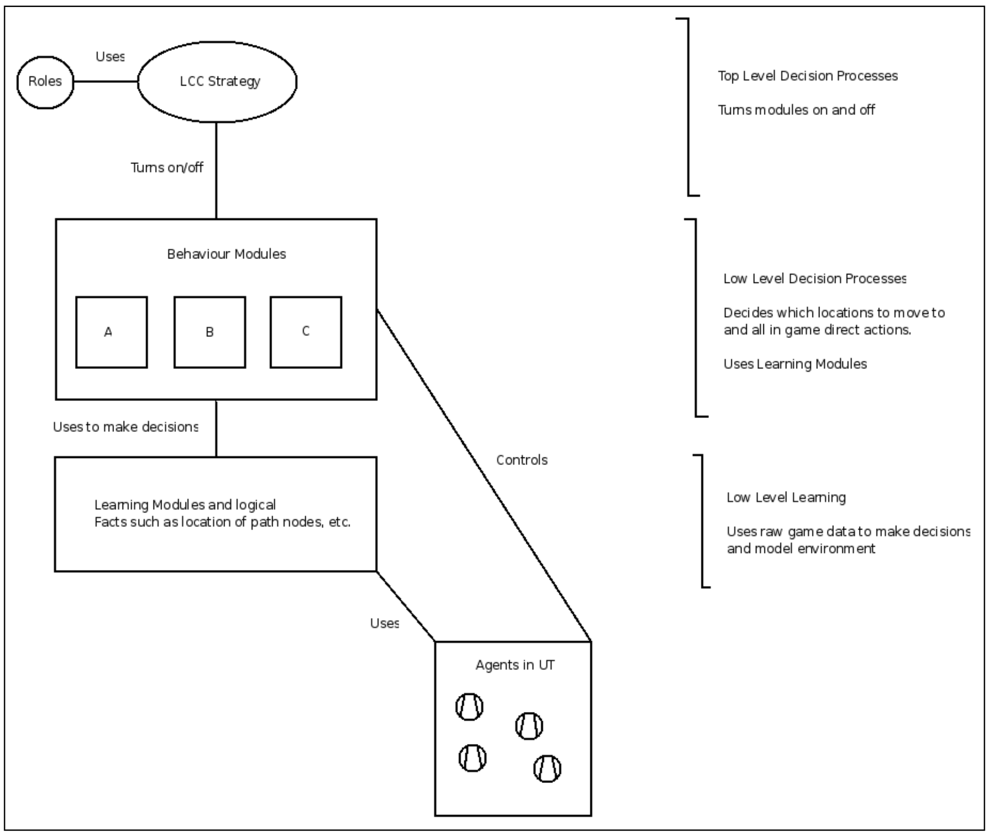

Roles
Uses
LCC Strategy
Turns on/off
Behaviour Modules
A
B
C
Uses to make decisions
Learning Modules and logical
Facts such as location of path nodes, etc.
Controls
Uses
Agents in UT
Top Level Decision Processes
Turns modules on and off
Low Level Decision Processes
Decides which locations to move to
and all in game direct actions.
Uses Learning Modules
Low Level Learning
Uses raw game data to make decisions
and model environment

Figure 5.1: Overall System Design

Altogether this can be thought of as a multi-agent machine learning subsumption architecture but with heterogeneous layers and a fixed size. The link to subsumption architectures is explained in section 5.1.1.

It is also convenient to think of the system as a multi-agent finite state machine with message passing in the upper strategy defining transitions between states (behaviour modules).

5.1.1 Architectural Motivations

Attempting to achieve robustness and stability in any situated, multi-agent, complex, real-time domain requires adaptive agents that learn about the changing environment[105]. There are going to be differences between instances of the domain (levels) and we must have a way of dealing with them. As motivation we can consider the following quote:

> ...complex tasks such as soccer comprise multiple overlapping behaviours, whose diverse demands can only be met by combining the strengths of qualitatively different learning approaches [57].

Soccer is a task which is complex and similar to game-playing in UT in the sense of needing multiple different approaches to behaviour. It involves team work, communication and adaptation as does performing within the UT environment.

Although it may be possible to solve the problem of writing a non-adaptive mechanism to deal with multiple domain instances most research into this area seems to suggest that some level of adaptation would be needed either at the script or lower level[98, 92, 17, 96, 72, 103, 58, 102]. Complex domains tend to present large amounts of observable data. The raw nature of the data generated suggests machine learning and data mining techniques are likely to be most successful. The situated nature of the bots determines that the issues associated with reinforcement learning, namely over-reinforcement of conclusions and adherence to *local maxima*[105], will also apply as actions in the environment affect the environmental state.

Having multiple bots presents the question of whether they should be homogeneous or heterogeneous in respect to the frequency and goal driven nature of learning. If we choose homogeneous then each bot must run an identical learning mechanism whereas heterogeneous leaves more options.

We chose to have heterogeneous bots because of the extra flexibility this offers for reinforcement learning. We also felt that communication could be best utilised if the bots could have different abilities. If they were all performing identical machine learning and running identical modules this would render the communicative element of the system as a purely directive mechanism. A further motivation for the heterogeneous bots is that we know that they will be performing activities other than machine learning. Having team members assigned to different roles in which they perform differing amounts of exploration and exploitation allows this.

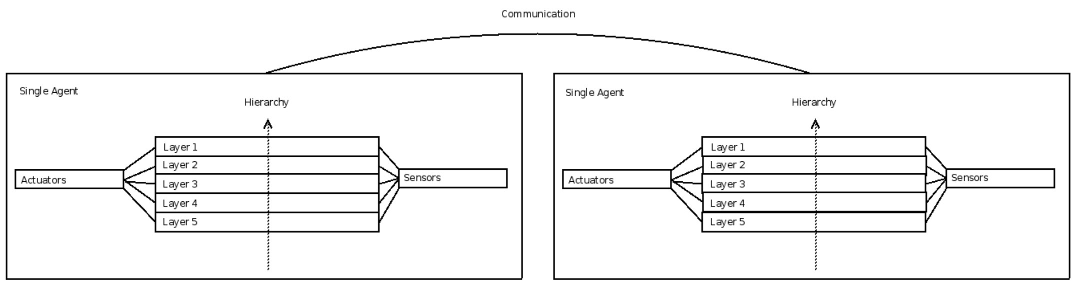
Furthermore if we wish these bots to organise themselves as a team, in a controlled way, they must communicate.

These considerations suggest a system with multiple communicating heterogeneous bots in which adaptation takes place. Each bot must be able to take roles which are more or less explorative and these roles must be able to change in response to in game changes and the quality of data.

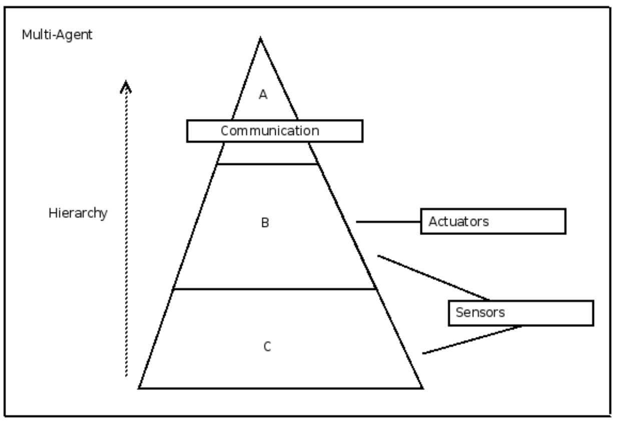
Brooks' Subsumption architectures [20] are common in robotics and one of the key features of these is a layered system with behaviour modules at each layer. These behaviour modules feature varying levels of abstraction with complex behaviours being built upon lower level modules. Our system borrows heavily from this notion of layers building on top of data from each lower layer but ditches one of the assumptions. The layers in our system are heterogeneous as not all layers have access to our virtual sensors and actuators. Our layers are also not all behaviour modules. The layers in a subsumption architecture are built on behaviour modules which combine smaller behaviour modules into more complex behaviours. In our system the only active layer, in terms of actuation in the environment, is the middle layer which contains all the behaviour modules. The other layers are machine learning and organisational/co-coordinative layers.

Another departure from the standard subsumption architectural paradigm is that we have integrated multiple bots into the system via the plans at the top level. Subsumption architectures tend to be for a single agent[19, 70, 117, 25], typically a robot or robotic insect, with multi-agent behaviour being achieved using emergent behavioural designs such as swarm[79, 59] technology or clever solo design. These differences are characterised in figures 5.2(a) and 5.2(b) In our system this is much more easily controlled and mediated via a carefully designed team strategy.

By creating a system in this way team behaviour can be engineered across different levels which is both uniform in gross effect but also unique to domain instance. It also facilitates concentration on agent roles and how the multi-agent part of the system allows communication. To create good plans, which can be executed, we must abstract



(a) A typical subsumption architecture design



(b) Our approach

Figure 5.2: Comparison to subsumption architectures

away all of the instance specific data (Or at least the parts which are believed to change between instances), pushing it down to a lower level mechanism which is more suited to dealing with it, thus allowing the focus of the plan to shift to team dynamics and agent interaction. Later in this chapter we present a motivational example to illustrate this fact and then present the results in chapter 10 as evidence for this claim.

In summary for a domain to be well suited to a solution involving our architecture it must have the following constraints:

Instances Differences Instances of the domain must be substantially different in some way which can be modelled without knowing about other instances *a priori*. Therefore the domain must be known to the extent that an abstract description can be made

Machine Learning Techniques Effective machine learning techniques must be available for different elements of the environment

Team Based The agents acting in the environment must be team based

Complexity The domain must be complex enough as to generate enough data to learn from but not enough data as to render the environment easily predictable without adaptation

Bounds on Learning The agent's actions must have enough effect on the domain environment, and their own performance, so that they cannot simply learn until they have complete data as in [46]. This dictates that the either the environment must be adversarial or there must some time limit on actions.

UT, as a domain, meets these requirements in the following ways:

Instances Differences Each level is of a different shape and size. Pick-ups are also in different locations and enemies spawn in different areas.

Machine Learning Techniques In part 2 we deal with effective machine learning techniques for UT.

Team Based All three games types dealt with are team based.

Complexity With the gamebots modification UT generates enough data to learn from

Bounds on Learning Enemy agents make UT adversarial. The game types also penalise explorative behaviour indirectly. Thus we cannot spend all our time learning until we obtain a complete domain description.

5.1.2 Modularity

One of the key concepts of the architecture is modularity, the idea that individual behaviour modules can be recombined to create desirable behaviour. The concept of modularity also carries through to the adaptation level which utilises a number of machine learning components to build the behaviour modules.

Thus the system can be viewed as having two specific types of modular components, the machine learning techniques and the behaviour modules. These occur at different layers within the system and have slight differences in their properties.

The machine modules take in environment data and compress this into conclusions which are then fed to the behaviour modules. The behaviour modules can thus take data from the environment as well as data from the learning modules and then execute bot actions within the environment. This allows us to think of the learning modules as having only sensors and the behaviour modules as having both sensors and actuators.

The modular nature of the system allows expansion by means of alternative machine learning techniques, than those used for our domain. Similar domains, which have characteristics as described in sub-section 5.1.1, will have specialised learning techniques and behaviour modules which can be slotted into the architecture.

Engineering Box

Adaptability Concerns The least adaptable part of the system is the behaviour modules. These send direct action commands to the game server and are implemented in relatively low level script with many fiddly implementation details. Some suggestions are made in the further work section for scripts which could be used to offer more reusable functionality but in any domain this kind of operation would require significant middle-ware before the modularity desired can be achieved.
It is feasible that an adapter could be written rather than fully building new modules, but this seems slightly counter-productive as it is likely that work in a new domain would require custom behaviour modules anyway.

5.2 Overall Plans and LCC

The team plans for the bots are written in a logic based calculus called LCC (Lightweight Cooperative Calculus)[85]. In principle any communicative calculus could be used but the reactive version of LCC[40] fits with the motivations in sub-section 5.1.1. LCC allows for agents to take multiple different roles and for messages to be sent between agents. Our implementation of LCC was adapted to allow the bots to perform certain actions in response to receiving messages or in response to satisfaction of certain constraints. The following formulation of LCC[40] was used:

```
Framework := {Clause,...}
   Clause := Agent :: Dn
    Agent := agent(Type,Id)
       Dn := Message | Dn then Dn | null <- C
  Message := M => Agent | M => agent <- C | M <= Agent | C <- M <= Agent
        C := Term | C and C | C or C
     Type := Term
        M := term
```

Agents are individual game bots which can take on specific roles. They can send messages to other agents if constraints are satisfied. These are then checked against the bot's internal game state to either test for truth or to perform some task. The intuitive reading of each connective is as follows:

Agent Each bot is considered as an agent with a role (Type) and a name (Id). These two variables define a unique identity for the bot allowing messages to be passed to the correct recipients.

then The `then` connective represents sequence, `Dn` elements are considered sequentially, ordered left to right, when this connective is applied.

null This represents no messages passed, and is usually used to implement default behaviours.

Send => is the syntax for sending messages with the message on the left and the recipient name and constraints on the right.

Receive <= is the syntax for receiving messages with the message on the left, the sending agent on the right and constraints to the left of the message.

Constrained `<-` is syntax for “constrained by”. When this connective is applied to a received or sent message it is read as “Only accept or send messages if constraints are satisfied”

C This is the symbol for constraints

Message Messages are Prolog predicates sent to other bots. The type of message sent is dictated by the `Type` and `Role` in the `Agent`part of the send or receive connective. If the `Id` is unspecified the message will go to all bots in the role `Type`. If the `Type` is unspecified and the `Id` is unspecified the message will go to all bots. In all other cases the message goes to the bot in role `Type` and with name `Id`.

Actions are also encoded as constraints, this might seem unusual but semantically it is read as *an agreement to perform a task and thus endeavour to make it true*. For instance in the following case constraints are used to control movement:

```
  agent(a,player)::helloFrom(a) => a(A,player)<--movementAttempt(nearestHealth)
```

In the example above movementAttempt would act as the constraint upon sending the `helloFrom(a)` message to another agent such that the message could not be sent unless the bot was attempting to move as specified in the clause. The use of an uninstantiated variable means that all agents in the `player`role would receive this message. The following example shows how this could then be responded to by another player to enforce a desired behaviour:

```
  agent(a,player)::helloFrom(a) => a(A,player)<--movementAttempt(nearestHealth)
  agent(b,player)::movementAttempt(nearestAmmo) and ammo(X) and
 prologConstraint(X < 10)
                  <- helloFrom(A) <= a(A,player)
  agent(b,player)::movementAttempt(nearestHealth) <- helloFrom(A) <= a(A,player)
```

In this example agent `b` can respond in one of two different ways depending on its current situation. If it has low ammo and receives the `helloFrom` message then it will go to look for ammo, otherwise it will go to look for health. The predicate `prologConstraint` treats the argument as a Prolog statement and attempts to satisfy it using Sicstus Prolog semantics. Code after a `%` is a comment.

5.3 Path Differentiation - A Motivational Example

Engineering Box

Why? The use of a motivational example here is because above and beyond the body of work suggesting that the problem was a valid one to tackle and the literature surrounding previous works in this area I still felt that there needed to be some concrete example of why the proposed type of approach was likely to work. The example is not such a good motivator of how the specific behaviour modules will be created or even the type of team strategy that will be employed but it remains a good example of the type of thinking which led to the idea of the 3 layers in the architecture.

Dynamic goal/path selection is when a bot within the game could have multiple different goals, of equal importance, and that these could be differentiated between based on some in-game information. In this section this is implemented by a very simple "straw man"[1] approach, differentiating between multiple goals based on the length of the path to each goal point. This serves as a motivational example of why this type of level based modelling is useful.

Multiple goal locations are given to the system and for each a path is returned. These paths are then evaluated by counting the number of path nodes within each.

Next is the question of how to use this data to aid in playing the game. In playing each of the three game types there are questions regarding whether shorter or longer paths should be given higher priority and also what alternatives would be on offer to a bot trying to use this information. The simplest example is a scenario mimicking the human fight or flight response. In humans this reaction is an auto-nervous response which occurs as a result of heightened levels of adrenaline within the body. Given a predator, ready to attack, the human will automatically choose either to fight the attacker or to flee from the area.

To model this the bot is given a strategy which says that in any situation the bot can either go for ammo, health pick-ups or can fight a visible enemy. In the original system, this would entail creating a strategy where movements to health, ammo or enemies would be constrained by in-game conditions such as level of health or ammunition in the gun. The problem with this is that it forces the strategy designer to know the level specific details, such as, for instance, the distance of particular points in the level from

[1]We used the term "straw man" here because we do not expect this approach to work on its own. This is to distance it from our actual techniques, as a motivational tool

the agent. Such a strategy is:-

```
 Strategy 1.1

%% health is low
a(random,Id)::null<--movementAttempt(nearest_health) and health(H)
and prologConstraint(health<50)

%% ammo is low
a(random,Id)::null<--movementAttempt(current_weapon_ammo)
and ammo(A) and prologConstraint(ammo < 10)

%% we can see an enemy
a(random,Id)::null<--visiblePlayer(Location)
and strafeAttempt(Location,Location)

%% none of the above
a(random,Id)::null<--movementAttempt(random_play)
```

Decisions have been made about which goals take precedence within this strategy (Strategies are parsed, when searching for applicable clauses, in standard Sicstus prolog ordering - From top to bottom). The very low level of abstraction away from the game specifics chosen forces this type of strategy to be created. For instance to know what would happen in a situation where ammo was low and health was low a clause such as this would be needed:-

```
a(random,Id)::null<--movementAttempt(CHOICE) and ammo(A)
and prologConstraint(ammo < 10) and health(H)
and prologConstraint(health<50)
```

Here `CHOICE` would represent what to do in this particular situation. Again, this involves coding a strategy for every possible situation, leading to impracticality of plan creation, internalisation, and eventual combinatorial explosion. Even ignoring large theoretical problems there is still an issue of having to make decisions about what to do in these scenarios. Ideally these types of decisions should be handled by the bot, allowing the LCC strategy to remain consistent and verifiable while still being practically executable. This view is echoed in [58] where it is suggested that

> ... it is often infeasible to foresee all the potential situations an agent may encounter and specify an agent behaviour optimally in advance ......

This issue arises because the strategies are parsed in a certain order, forcing the need to check which goals will execute first and last in order to determine bot play. For instance, if health is less than 50, ammo less then 10 and a visible enemy present then the bot would always go for health. But say for instance in many previous cases where the bot had this amount of ammo they had won fights then maybe it would be better to stay and fight with the enemy? Trying to make these kind of decisions by abstracting away from the LCC plan is the main thrust of this thesis.

Thinking of this as a situation where the lengths of paths can be utilised, a strategy like the following allows more dynamic and flexible play without pre-supposing as many constraints on the bots behaviour:-

```
 Strategy 1.2

%% can see a player and health is low and ammo is low
a(random,Id)::null<--movementAttempt(multi,[Location,current_weapon_ammo,
nearest_health])
and visiblePlayer(Location) and health(H)
and prologConstraint(H<50)
and ammo(A) and prologConstraint(A < 10)

%% can see a player and ammo is low
a(random,Id)::null<--movementAttempt(multi,[Location,current_weapon_ammo])
and visiblePlayer(Location) and ammo(A) and prologConstraint(A < 10)

%% can see a player and health is low
a(random,Id)::null<--movementAttempt(multi,[Location,nearest_health])
and visiblePlayer(Location) and and health(H) and prologConstraint(H<50)

%% can only see a player
a(random,Id)::null<--movementAttempt(Location) and visiblePlayer(Location)

%% health is low and ammo is low
a(random,Id)::null<--movementAttempt(multi,[current_weapon_ammo,nearest_health])
and health(H) and prologConstraint(H<50) and ammo(A)
and prologConstraint(A < 10)

%% only ammo is low
a(random,Id)::null<--movementAttempt(current_weapon_ammo) and ammo(A)
and prologConstraint(A < 10)

%% only health is low
a(random,Id)::null<--movementAttempt(nearest_health) and health(H)
and prologConstraint(H<50)

%% none of the above
a(random,Id)::null<--movementAttempt(random_play)
```

Engineering Box

Movement Attempt In this strategy the notion of a multi path movement attempt is introduced. When this is executed the system will pick one of the three different targets based on some measure of path utility. In this section this utility is based on path length.

This strategy has sacrificed both determinism and simplicity (the strategy is still relatively simple but not as easily understood as before) but the bots will play differently based on their position within the level and thus the strategy is more level independent. Using only this measure of goal importance based on path length to goal still creates a strategy which has problems regarding the amount of constraints used. There are still too many repeated constraints implying that we are still having to script what the bot should do in all situations.

This is a straw man attempt, better approaches are discussed later, but it does offer some insight into the power which this type of approach can offer. Eventually this type of play is reduced to a simple constraint `startModule(engage_enemy)` which would then take into account all of the play above at some lower level within the bot. This allows the consideration of more complicated team interactions. Internalising the behaviour within the bot is not unique but affords us the ability to then craft much larger scale strategies without having to worry about the execution details.

Consider observing only the variables of bot health, ammunition count and whether an enemy could be seen within the scope of the bot's view. The old strategy could be used to determine the bot's next move in every case, using the new strategy a much larger quantity of information is needed in order to make this decision. Namely, the information regarding the distance from every point of concern within the 3D space and the bot's current position.

Even though the strategy is more complicated in the abstract, the level of information and game data which it utilises, expressed as a ratio in relation to increase of strategy complexity, is favourable. The strategy is abstract enough to be understood but still covers a large range of different behaviours depending on the path discrimination strategy implemented at the underlying level.

A strategy could be coded which did not use the information regarding path length implicitly but instead made this data accessible to the LCC plan via state variables which could then be checked as constraints. Even with this the size of the strategy and the amount of work involved in creating it is already becoming unwieldy with regards to the expected utility of such a strategy. Take for example the demonstrative strategy 1.3 below:-

```
Strategy 1.3 (warning, this will not work with the system as some of
these constraints are not accessible, it serves only as an illustration
of why this particular technique is a bad idea)

%% can see a player and health is low and ammo is low and player is closest
a(random,Id)::null<--movementAttempt(Location) and visiblePlayer(Location)
and health(H) and prologConstraint(health<50) and ammo(A) and
prologConstraint(ammo < 10)
and distance(Location,D) and distance(nearest_health,He)
and distance(current_weapon_ammo,Cwa) and
prologConstraint(minimum(D,[D,He,Cwa]))

%% can see a player and health is low and ammo is low and health is closest
a(random,Id)::null<--movementAttempt(nearest_health) and
visiblePlayer(Location)
and health(H) and prologConstraint(health<50) and ammo(A)
and prologConstraint(ammo < 10) and distance(Location,D) and
distance(nearest_health,He)
and distance(current_weapon_ammo,Cwa) and
prologConstraint(minimum(He,[D,He,Cwa]))

%% can see a player and health is low and ammo is low and ammo is closest
a(random,Id)::null<--movementAttempt(current_weapon_ammo) and
visiblePlayer(Location)
and health(H) and prologConstraint(health<50) and ammo(A) and
prologConstraint(ammo < 10)
and distance(Location,D) and distance(nearest_health,He)
and distance(current_weapon_ammo,Cwa) and
prologConstraint(minimum(Cwa,[D,He,Cwa]))

%% can see a player and ammo is low and player is closest
a(random,Id)::null<--movementAttempt(Location) and visiblePlayer(Location)
and ammo(A) and prologConstraint(ammo < 10)  and distance(Location,D)
and distance(current_weapon_ammo,Cwa) and prologConstraint(minimum(D,[D,Cwa]))

%% can see a player and ammo is low and ammo is closest
a(random,Id)::null<--movementAttempt(current_weapon_ammo)
and visiblePlayer(Location) and ammo(A) and prologConstraint(ammo < 10)
and distance(Location,D) and distance(current_weapon_ammo,Cwa)
and prologConstraint(minimum(cwa,[D,Cwa]))

%% can see a player and health is low and player is closest
a(random,Id)::null<--movementAttempt(Location) and visiblePlayer(Location)
and and health(H) and prologConstraint(health<50) and distance(Location,D)
and distance(nearest_health,He) and prologConstraint(minimum(D,[D,He]))

%% can see a player and health is low and player is closest
a(random,Id)::null<--movementAttempt(nearest_health) and
```

```
visiblePlayer(Location)
and health(H) and prologConstraint(health<50) and distance(Location,D)
and distance(nearest_health,He) and prologConstraint(minimum(He,[D,He]))

%% can only see a player
a(random,Id)::null<--movementAttempt(Location) and visiblePlayer(Location)
%% health is low and ammo is low and health is closest
a(random,Id)::null<--movementAttempt(nearest_health) and health(H)
and prologConstraint(health<50) and ammo(A) and prologConstraint(ammo < 10)
and distance(current_weapon_ammo,Cwa) and distance(nearest_health,He)
and prologConstraint(minimum(He,[Cwa,He]))

%% health is low and ammo is low and ammo is closest
a(random,Id)::null<--movementAttempt(current_weapon_ammo) and health(H)
and prologConstraint(health<50) and ammo(A) and prologConstraint(ammo < 10)
and distance(current_weapon_ammo,Cwa) and distance(nearest_health,He)
and prologConstraint(minimum(Cwa,[Cwa,He]))

%% only ammo is low
a(random,Id)::null<--movementAttempt(current_weapon_ammo) and ammo(A)
and prologConstraint(ammo < 10)

%% only health is low
a(random,Id)::null<--movementAttempt(nearest_health) and health(H)
and prologConstraint(health<50)

%% none of the above
a(random,Id)::null<--movementAttempt(random_play)
```

Clearly this is not advantageous in creating an abstract plan which can be easily understood and put into action in a dynamic and flexible way. The only real flexibility lies in how the distances are measured. Strategy 1.2 allows a different method of differentiating between these alternative locations to be used in place of the path length measure, without sacrificing the overall feel or intention of the LCC strategy.

With all this added complexity to the way in which the bots perform pathing and choices, and the number of variables which the bot can access, at a level below the strategy programmer's awareness, is there any evidence that the system is working better or differently? Does this extra information actually benefit the bot? To answer these questions involves evaluating these strategies against each other to see how they perform on different levels. Two levels of differing nature were chosen to determine how the strategies affect play. The first is a very small level. The second is a much larger level where it becomes more important to consider length of paths.

To show the importance of level independent plan execution we have included a set of trials in which a further strategy is used which is much simpler as it disregards all data about current ammo and health, concentrating only on whether a player can be seen. When a player can be seen the strategy picks whether to go to the location of the player, the current nearest ammo or the current nearest health, based on path length to each. This is to show two things. Firstly, that even with only a minimal amount of flexibility we can still create system which does something of interest with minimal pre-emptive planning and very situated reactive behaviour. Secondly, it shows that there is still required some level of input into the strategy creation process and we cannot leave all of the decision making to data which is learned at game time.

In further chapters it is demonstrated that a further level of abstraction can be obtained, removing even the constraint about a visible player allowing all of this data to be handled by modules which are turned and off by the main LCC strategy. These modules still need to be created and the behaviour is largely fixed with only smaller elements which vary. This example is meant only as motivation for the kind of things which will vary within the modules. Even the simple abstraction used here, though, is closer to the `engage_enemy` operation that was discussed earlier:-

```
Strategy 1.4

%% can see a player
a(random,Id)::null<--movementAttempt(multi,[Location,current_weapon_ammo,
nearest_health])
and visiblePlayer(Location)

%% cant see a player
a(random,Id)::null<--movementAttempt(random_play)
```

5.3.1 Experimental Data

5.3.1.1 Trial 1 - Level = Gael

This set of trials was played on a very small level called Gael. The purpose of using a small level is to show that there is a difference in performance between the execution of Strategy 1.1 across two levels which can be bridged by strategies 1.2 and 1.4. A larger level is used in the next set of trials to demonstrate this. 5 trials were used for each strategy and then averaged.

The main problem with this level is that there is only one health point in the centre of the level on a platform but this health point is not always available, it takes time to regenerate before the bot can pick it up again. This can lead to situations where the bot has minimal health and is standing on this platform waiting for the pickup to

Strategy	Gael Level Trials, 5 Trial Averages	
	Enemy Score	**Bot Score**
1.1	30	**14.2**
1.2	30	**16.8**
1.4	30	**9.8**

Figure 5.3: Gael Trials - 3 Strategies

re-generate. Ideally the bot would have some way of inducing that the health pack is not where it should be and this is probably a flaw in the way the system, for interaction with UT, has been engineered. Although weak for demonstrating the argument, this shows that even with a very small, well defined strategy, small details of the level can drastically effect performance, altering behaviour. It is these types of plan implementation details which this thesis addresses.

Taking strategy 1.1 it is easy to understand the intention of the strategy and how it should operate in the abstract, however in the environment this is not what happens because of small level details. Occasionally the bot did stop to go get ammunition before returning to a fight. This again worked well because the level is small and very self-contained. On larger levels this could result in the bot having to move away from the fight to go and get ammunition. If this was a significant distance away then there could be problems with whether to go and get ammo or not.

The performance of strategy 1.4 is particularly bad. The bot never fights unless the fight is very close. This leads to the play being invariant of the enemy. This points to weighing ammo and health against the enemy in some way. It is clear that in some situations the basic constraints are useful but setting these is very difficult in advance. The trials also show that because the level is small, the bots often choose not to fight because something else is closer. This is different on larger levels as ammo and health pickups tend to be more spread out. Also the bot will often stand at a point when it has got there because it has no way to tell that its need have been satisfied, leading to it getting killed as it waits.

5.3.1.2 Trial 2 - Level = Deck17

Figure 5.3.1.2 shows the results for the trials on Deck17. Strategy 1.1 is much better than for Gael, with the bot making use of the health points very often instead of engaging in a fight. The problem is, sometimes it would be beneficial for the bot not to

Strategy	Deck17 Level Trials, 5 Trial Averages	
	Enemy Score	**Bot Score**
1.1	30	**4.4**
1.2	30	**9**
1.4	30	**1.4**

Figure 5.4: Deck17 - 3 Strategies

do this and instead keep fighting. They could then pick up the health afterwards when the fight was over. It would be nice to incorporate some more information about experience and the amount of time the bot has been engaged in a fight before considering going for health or ammo.

A strange side effect, observed while watching the play for all these strategies, is that the bots occasionally get stuck in a particular place. This is a side effect of the `random_play` behaviour which is not completely fool-proof. The enemy often gives up searching for the bot when this happens and camps on one of the higher areas in the level waiting to be found. This shows that there are areas of the level in which the enemy are less likely to go. Camping in this manner is quite a frequently observed behaviour of humans playing on on-line servers [87]. This points to the idea that the enemy play be modelled. This idea is detailed in section 7.4.

These trials exhibit some of the same problems as in the previous work [40] with loosing sight of the enemy. This could be solved by adding an internal belief state variable or alternatively using the enemy models which are discussed later in section 7.4. What is interesting is that on the previous levels this was not so much of an issue because the bots were more likely to run into each other as the level was smaller, echoing precisely the results of [40]. This again shows a need for level independent plan execution.

Overall the experiment shows that, although some elements which are learned or dynamic are likely to improve performance, we do need some elements to be set and immutable. The problem is deciding which elements to learn/make dynamic and which elements are specific to the UT domain/game-type rather than to each level.

A final point in this discussion consider table 5.5. Even though the performance of the strategies are all relatively poor the ratios of decrease in performance show that strategy 1.2 exhibits the lowest relative drop in performance with the transition from small to larger level. This shows there is a possibility of creating strategies who's per-

Strategy	Score for Deck17 / Score for Gael
1.1	0.309859155
1.2	0.535714286
1.4	0.142857143

Figure 5.5: Ratio of Decrease in Performance

formance shows low variance over levels but that these mechanisms need to be created carefully if this invariant performance is to be high rather than low. The strategies created here are by no means good models for playing the game. Even with a simple piece of modification to the strategy a slightly lower variance in performance across two different levels can be achieved.

This method of using the path length information is not even particularly dynamic as it is still essentially a decision based on static data about the level but this data changes as the dynamic element, the bot, moves about within the static environment. This represents a motivational starting point from which to progress with more complex work.

Chapter 6

Learning Mechanisms and Techniques: Introduction

Engineering Box

Prototyping Approach Because the architectural layers were pyramidal we employed a bottom up engineering strategy, mainly because the higher layers were based on those lower in the architecture.

Technique Motivations One of the key factors in creating the machine learning techniques was to consider the different environmental factors which could reliably be measured. A lot of the decisions regarding these were based on thinking about what the largest factors concerning performance in the domain were. Typically the biggest three elements are the other bots, the weapons and movement within the domain. Removing knowledge of any of these three results in poor performance.

Time Constraints Some consideration is also taken towards time constraints on learning. This is largely not a problem as extra computational power can always be acquired as long as the learning is within some upper bound, but it must be considered that the bots are acting in a real-time world. As such, sometimes batch learning processes are performed at intervals, rather than when new information is obtained. To deal with this a separate processing thread ran at all times which updated the models, performing calculations which would be time consuming to perform on individual cycles of each bot.

Machine Learning (ML) and Reinforcement Learning (RL) Adaptation Some work is also presented which shows how traditional ML and RL techniques can be adapted to cope with large amounts of changing information which is being gathered in a real time fashion. Many of the techniques presented are not state of the art but the way in which they are being combined and utilised is unique and offers a good model for creating systems of this type.

In this chapter we provide some background discussion concerning the modelling used in the machine learning layer of the architecture. It is this layer of the architecture which provides the adaptibility to changes that happen within a match.

In section 5.1.1 we discussed that one of the criteria for a domain to have, for our architecture to be applicable, was available effective machine learning techniques for modelling environmental factors. In this chapter, and that following it, we show some environmental factors and corresponding learning techniques that can be used for the UT domain, in order to meet this requirement. These chapters cover the development of layer 1 (low-level learning) of the architecture.

As discussed we do not have any interest in these particular techniques themselves, they only represent an application of our architecture to an example domain representative of our applicable environments. As such this chapter, and that following it, can be considered a description of the learning component prototyping process for our domain.

6.1 Learnable Environmental Factors

The 3D environment of UT presents many options for machine learning. These are generally split into two categories: Factors provided by the environment and factors provided by the other bots in the environment. It is also perfectly valid to instead consider the adversarial bots as part of the environment. We either have a neutral environment with adversarial opponents or an altogether adversarial environment. In practice, the distinction makes no difference as we are not considering any properties of interaction such as argumentation.

In practice we should consider the following quote from a paper dealing with research into AI for quake 3 (A similar type of domain viewed from a similar level) when examining which factors are good for modelling:

> “Very good candidates for applying AI learning techniques are the basic decision points of the NPC (Non-Player Character) where it has to select a certain behaviour depending on the environment. Examples are weapon selection, selecting which item to get, selecting a target, etc. The advantage of these types of decision is that they are quite well isolated and therefore the existing methods can easily be replaced by a different one without having to change the rest of the NPC.”[119]

This fits with the observations made in [40] regarding where the strategies used were most effective and the best type of approach to strategy design.

Outside of weapon modelling, the most obvious factor provided by the environment is the levels due to their different shapes and sizes. The levels are arranged into areas such as indoor and outdoor sections as well as individual buildings and structures. They also have different arrangements of the pickups including weapons and health pickups.

The enemy bots in the level have specific areas in which they play. It is assumed that this therefore changes the utility of these areas creating divisions within the level.

Fights with these enemy bots are also theoretically model-able. The outcome of any given fight can be used to guide decisions for future fights.

6.2 Learning Catagories

The unique environment presented by games such as UT, along with the constraints that acting within that environment puts on the learning agents, presents an opportunity to explore avenues of active learning and closed loop machine learning[58]. This is primarily because of the lack of well annotated and categorised data to learn from. Feedback from the environment must be used as input to select data which is most likely to be of use where no clear annotation exists.

6.3 Game Data

It is important to detail the types of data which the game gives us to work with:

Navigation Point Positions 3D Cartesian co-ordinate locations placed around the level by the level designer to allow bots to navigate.

Agent positions 3D Cartesian co-ordinate locations of other bots in the game, for opponents these are only accessible when they are in view, for our team members the location of team members is available to all team members

Health Point Positions 3D Cartesian co-ordinate locations listing all the health point pickups in the level

Weapon and Ammo Positions 3D Cartesian co-ordinate locations listing all the weapon and ammo pickup points in the level

Agent Health Each bot has access to their health value and the rest of their team's health values , this is a number between 0 and 100

Agent Weapon Each bot has access to which weapon it is currently holding and also which weapons it has in its inventory

Length of fight When the bot is involved in a fight it has access to how long the fight took, this is measured in game cycles which are the units of time in UT

Taking Damage The bot is informed whenever it is taking damage and of how much damage is taken

Team Has Flag The bot is informed if its team are in possession of the opponent's flag

Opponent Has Flag The bot is informed if the opposing team are in possession of their flag

Domination Point Status The bot is informed of which team controls the domination points

Game Score The bot is informed of the game score

Kill Achieved The bot is informed if it achieves a kill and which bot it killed

In Game Cycles The bot has access to a timer so that it can parametrise changes in data by a time step similar to the length of fight parameter

These are the raw data elements which the machine learning mechanisms must work from and are the only elements assumed true. Everything else which could be viewed as representational is assumed or learned based on the values of these parameters.

6.4 Static versus Dynamic Learning and Heuristics

During this thesis learning methods are dealt with which are dynamic in the sense that they adapt and learn information based on experience and observations made during the playing of the games. Although online learning is the main focus, some ways are presented of performing static learning based on information which is available before the matches begin. It is also shown how to integrate heuristics and intuitions into the learning process. Most of the time this static learning and heuristic information is used to help the dynamic learning algorithms perform better rather than as a direct learning mechanism. These concepts are defined as follows:-

Static Learning Something which is learned offline prior to other learning or something which is learned from a static environment (Presented in [97] as off-line learning).

Dynamic Learning Something which is learned over multiple occasions or in response to something which changes throughout the life cycle of the domain activity (Presented in [97] as on-line learning).

Heuristic[56, 65, 86] A rule of thumb based on intuition or prior knowledge about the domain. Often used to guide learning processes.

6.5 Evaluation

In order to assess the effectiveness of the modelling techniques, interim strategies are used to gain data from the level. Some of these temporary strategies build on earlier ideas presented in [40] and some are purpose designed to allow the gathering of data to assess how well the models work. A lot of these intermediate strategies are not in keeping with the ideas expressed in section 5.3 but all the chosen modelling techniques are eventually tied together into well abstracted modules in Chapter 3. In many cases it is not the intention to show that a particular technique is the best for a purpose[1], that there isn't a better formulation, or even that the strategy elected for testing is performing something which is ultimately useful. The techniques are evaluated to see in what way they perform and to motivate what they may eventually be useful for. In some cases this amounts to setting a range of parameters to see which model fits the requirement. In others it involves a more complex study of the model design.

Although the system contains many separate working parts and components care has been taken to try to evaluate each of these, where possible, individually, in a modular fashion. In some cases this was not possible. For instance in the case of evaluating the dynamic path-finding algorithm in section 7.10 the enemy model defined in section 7.4 had to be used to provide the data on which the path finder operated. In cases such as this the technique being referenced (the enemy model in this case) had its particular form fixed and chosen from the relevant section. This allows the evaluation of the objective technique (The path-finding algorithm in this example) without introducing

[1]There is no interest in having only the most state-of-the-art machine learning techniques for every purpose. It is the system as a whole which is novel in this work rather than the techniques themselves. This said the way in which some of the techniques have been modified for this domain can be considered novel.

a further variable. In Part 3 an analysis of the modules and strategies which make use of the techniques developed in this section is performed. This is an indirect analysis of the system as a whole and the effectiveness of the individual components. This is also complimented by a basic component level analysis of the system.

This may seem like a strange strategy for assessing the machine learning element of the system but in the deployment environment it is worth noting that there is little actual sense of optimality. To assess the system involves observing the matches and then combining numerical data with observations of the matches played. To assess the individual learning techniques relies on visualisation techniques and subjective assessment of the usefulness of the particular representations. In this way the techniques presented here have elements of both supervised and unsupervised learning with environmental feedback acting as the largest single mediator of the bot's actions, decisions and internal bias. Although not strictly couched in the reinforcement learning area (this is because certain modules may contain elements of RL whereas others will contain very little) there is an element of the bot reinforcing certain decisions based on the state achieved after their execution.

In some cases it is possible to evaluate a technique by applying it to data gained through a game execution, in an off-line manner. This can give good results about how useful a particular technique will be on real-time data. It is also shown how these techniques can then be evaluated in a real-time manner and how this reinforcement learning element of the technique interacts with the machine learning process.

6.6 Potential Problems

It might seem the task of developing these techniques could be very daunting, as it leaves much room for a huge range of different techniques to be used and different manners in which to collect information. Many of the techniques used, and the data which they are based on, dictates what particular style of learning will take place. This limits the number of decisions regarding learning to be considered.

Another problem is sparseness of data. A simple solution to gaining more data for use is to allow the bots to store information about certain levels and games to disk and then read this afterwards to enable better play. Some way of differentiating this data and some decision mechanisms about when to use are then needed. This thesis does not cover this issue, known as knowledge transfer [90], in any depth, although there is an option for furthering the work in this system and there is nothing inherent within the system stopping this approach from being taken. The problem of knowledge transfer is also simpler with numerical statistical data than in more traditional symbolic domains where the semantics can prove a stumbling block.

In terms of the classification of problem suggested in[58] our system has certain constraints on the bot's actions which have a real-time bound on the learning process where the bots are acting as asynchronous processes (although technically they are multi-threaded so they do take turns. This turn taking happens so quickly that the bots are appear to act, to the observer, in real-time). The different techniques also vary between parallel learning and recall at all times and cheap, on-the-fly, learning and off-line, computationally expensive, learning, depending on the particular technique and the complexity and frequency of the input data being used.

6.7 Influence Maps

Most of our modelling work is based on the influence map paradigm[49, 28, 51, 4]. This is the idea of taking a space, which could be an actual space or a theoretical learning space, and then overlaying some form of utility map (occupancy/influence map) to the areas within the space to allow them to be differentiated in some way.

Normally influence maps are used to guide other techniques to better solutions by biassing learning in some way. In our case we utilise influence maps, heavily, directly for online learning. The main reason is that basic influence maps are easy to learn with little data and the results are interpretable quite easily. By heavily processing these maps and exaggerating differences and differentiations in the map we can build wholesale usable techniques based only on these, which are usable straight away.

Our main method for this integration is to feed the influence maps into a path-finding mechanism which is adapted to use the dynamic information.

6.7.1 A Note on density modelling

The work presented in this section of the thesis makes extensive use of kernel based modelling in the context of parzen density estimation, particularly along the lines of the work presented in both [52, 68] and especially [89]. The reason for this is that kernels can offer a good route into learning in areas of sparse data. It is entirely possible to create modules which have more symbolic goals and operate from a more logic based stand-point but the leaning in this document is geared towards a good balance of statistical learning at the low level combined with more symbolic/logic based reasoning at higher levels. This fits with the literature showing that the type of goals of the components of the system are best suited to this type of treatment. In most cases at least two contrasting approaches to the problem are considered. These, however, still tend to be in the same area of learning but often one is simpler, less grounded or more customised for the situation. Again this demonstrates the lack of importance that the individual modules themselves play in the system.

Large use is made of an ML technique known as K-Nearest Neighbours. This is very basic but it is shown that with the particular data generated from the UT problem domain this performs better than more powerful techniques as it begins to generate conclusions from early in the learning process. The mahalanobis distance is presented as an alternative distance measure. Gaussian or radial basis function kernels are not considered as distance measures. This is because they offer no different conclusions in a K-Nearest Neighbours situation. Consider the following diagram showing three points X, Y and Z in two dimensions.

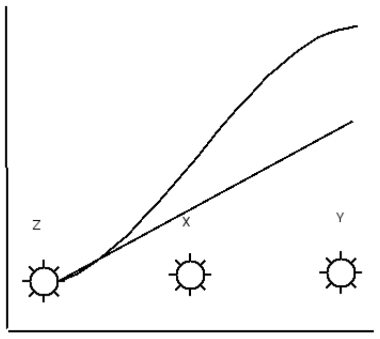


Figure 6.1: Ordinal Situation of Gaussian Kernel

Although the exact distance calculations of points from each other are affected by the Gaussian shown, as opposed to a linear scale, the ordinal ranking is not. As such the kernel model offers no advantage. In general, unless the output function is to be used in a sense other than ordinal ranking, this is true of kernel modelling. With non-smooth kernels which have discontinuous non-ordinal surfaces this fact need not be true, take for instance the situation in figure 6.2.

Here the kernel ranks Z higher than X.

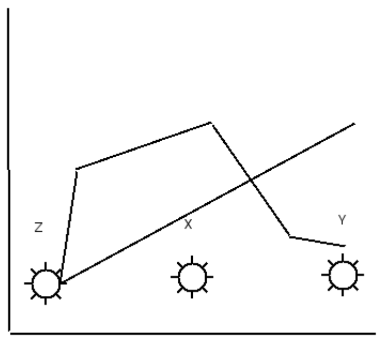


Figure 6.2: Ordinal Situation of Non-Smooth Kernel

Chapter 7

Learning Mechanisms and Techniques

In this chapter we detail the actual mechanics of the techniques used and present our initial analysis of their performance. We also present our final set of chosen techniques and their respective configurations.

This chapter shows ways in which data and information can be abstracted from the level and environment. The techniques utilise information from the bots as a group. Communication is used both directly in the LCC strategies through messages, as in [40], and indirectly, through dynamic model updating in conjunction with other bots. The machine learning mechanisms presented here straddle the traditional definitions of Multiplied Learning, Divided Learning and Interactive Learning [17].

In some of the sections we try to present a typical offline learning technique and show firstly how it could possibly be adapted for online use and then show why it is not particularly good to use it in this manner. This is mainly because the techniques used for parameter estimation here are coming at the task from a different angle to what is normally performed in this field. Our goals are not to find the best technique or the most optimal technique. We are concerned with finding a working technique for our domain, which can perform parameter estimation in an online manner. If we had any optimality constraints they would be highly weighted by how well they handled online learning and the challenges this presents.

Engineering Box

A Personal Perspective My personal reaction to a lot of the offline machine learning mechanisms initially postulated in these sections was that they were too complicated to work in our domain. As the architecture is the main contribution it almost doesn't matter what the domain is as long as it fits certain key assumptions. The big problem with many of these complicated mechanisms is that the actual target function underlying learning is in many cases very simple but with lots of noise. We don't care about the nature of the noise and essentially a lot of these techniques are over-fitting small amounts of very noisy data where as really we want a more general feeling of the data's core concept.

7.1 Learning the Level Size and Shape

One of the key challenges with building the system in this thesis was to pick out which game elements were the best to model. Given that our remit was to deal with the problem of stable performance over a number of different game levels, it seemed pertinent to have some model of these levels. This could then be used to help set other parameters and base other assumptions on. If we knew the type of level we were dealing with, this could possibly make other problems such as path finding and enemy modelling easier to deal with. Therefore it made sense to deal with levels first as a space for other concepts to slot into. We knew that we could only work from the navigation points which we had been given from the gamebots connection layer and so had to extrapolate from these any useful information which we could. Clearly key to this were the size and shape of the levels.

There is much information that can be gained from the size and shape of the levels in the UT domain as they are represented by point clouds, which are similar to the 3D laser scans used in both robotics[60] and city modelling[43]. In both fields a reverse engineered structure is used to gain a better understanding or level of performance. In [60, 115] the extra information is used to guide a robot and improve obstacle avoidance while in [43] the 3D data is used to construct a footprint for buildings for automatic map generation. Our problem domain is a mixture of the two cases. We have autonomous agents whos performance we wish to increase but the nature of our data is more in-line with the 3D laser scans of the city, albeit at a lower resolution, plus we aren't dealing with obstacle avoidance in the environment model. A consequence of the low resolution is that some of the techniques used in [43] are not directly applicable as they rely on having data of a larger density and as such clustering methods are more appropriate than line and curve following algorithms.

Another deviation we make from the city data case is that we are not only trying to find a footprint of the data and are interested in the general structure. As such the work presented in [91] is relevant, although we don't perform classification of sections into different urban features instead only seeking boundaries between them. This also allows us to consider a more general class of algorithms and not have to limit our modelling to specific feature sets.

Most of the data required to fit structure to the point clouds can be extrapolated from the point clouds themselves automatically. This information can then be used to select appropriate strategies and alter some of the machine learning techniques.

Pieter Spronck also provides some notable work into level and environment modelling applied to the area of game AI. In [112] he proposes a system for taking in level features and using these to guide the construction of a decision tree which is then used for deciding agent actions in that particular environment. Again the focus of his work is slightly different as he uses the environmental data to directly construct his strategy. It also differs as [112] assumes that the data used from the level is given in a high level, usable, form.

In the following sections we deal with automatically learning level size and shape using Principal Components Analysis (PCA), Entropy Clouds and polygon fitting.

7.1.1 Level Size

The simplest assessment of a 3D space, relative to similar spaces, is its size. Gaining a general sense of the deviations in size between environments can inform many choices regarding actions and behaviour.

There are several ways in which the size of the level can be estimated from the path node locations. The path nodes in the level are 3D locations which resemble a point cloud, similar to what would be produced from a 3D range scanner [27]. One measure of level size is to pick 3 gradients of direction which are orthogonal to each other and project the level points onto these. The maximal points along the gradients can then be used a measure of level size, biassed by variance of point cloud structure. A good way to pick the gradients to project the level points onto is via PCA (Principle Components

Analysis)[78].

Here we term this method of measuring level size *PCAMagnitude*.

$$PCAMagnitude(set_x) = \frac{\sum_{i=1}^{N} D(X, Closest(X))}{N}$$

This method focuses not just on distance but on distance seen from the perspective of inter-node variance. Simply taking the distance between nodes in the level and assessing the distance from the furthest point in one direction to the furthest point in the other direction is just measuring the size of the level in terms of these maximal points. But it is known, a priori, that the path nodes in the level are spaced in such a way that they can be used to correctly navigate paths and avoid certain obstacles. As such if there are many nodes near each other then it is advisable to factor this into the equation for level size, as it gives a measure of number of compartments as well as size. Projecting onto the axis of most variance factors in both of these ideas.

A simpler measurement of distance is to look at the average distance from every node to its closest neighbour and then multiply this by the number of nodes in the level. This gives a raw value on the size of the level which can be used to rank levels in terms of size. This will be termed the *Magnitude* of the level henceforth.

$$Magnitude(set_x) = \sum_{i=1}^{N} D(X, Closest(X))$$

7.1.2 Natural Level Clustering

Engineering Box

Difficulties Clustering was difficult because in some clustering applications the data being used is expected to be clustered in a specific way so the strategy can be crafted to take advantage of such assumptions. This is the not the case for us as the level data is not clustered in any specific way that is common to all levels. This means that the problem is less about setting the parameters of a known model and fitting it over the data as much as it is about coming up with a model for the data from scratch. As such the mechanism for clustering had to be both automatic and general enough to fit all data in some meaningful way.

7.1.2.1 Background

If we make the assumption that the navigation points in the level are naturally clustered in some way this allows us to perform some modelling on them on this basis.

Because our underlying modelling strategy is to use influence mapping where possible it can be useful to be able to differentiate the objective map into different areas and then model within these.

Although there are many available methods for clustering a point cloud our particular set of circumstances somewhat limits our choices. The biggest limiting factor is the range of expected patterns and total lack of any assumptions regarding expected formations. For instance in medical range scanning for tumours typical scans can offer higher resolution in particular areas based on search priorities. Other factors limiting choices include the lack of an objective evaluation technique and unknown number of points in cloud.

There are many methods not just for fitting structure to point clouds but also for performing automatic clustering. In [43] k-means clustering is used which is essentially a simplified version of Renyi Entropy Clustering with no kernel. In many cases surface growing and planar segmentation are used bot both of these are based on having much more dense data [91, 77] and do not work so well when data is sparse.

There is also much work concerning classification of objects from scanner data, particular if the objects are regular or non-general such as faces and bilateral objects [24] but again these do not offer the required level of generality.

In our case Renyi's Entropy Clustering[52, 35], from information theory, was used. This gives a method of measuring the in-cluster entropy, the between cluster entropy and also an algorithm showing how to use this information to decide the natural clustering of the level. In [52] it is used to cluster unknown data into non-uniform distributions using parzen density window estimators for the purposes of classification.

Extensions such as [36] exist for tasks such as FMRI image clustering but in our domain the basic clustering algorithm performs as required.

7.1.2.2 Renyi Entropy Clustering

The algorithm for Renyi clustering is as follows; start with a much larger number of clusters than expected in the data. Place some initial cluster seeds and then start adding points via a method which minimises within cluster entropy increase in the cluster which it is added to. Having done this, find the worst cluster by removing each one in turn and seeing how this effects the between cluster entropy. Once the cluster is found which, when removed, maximises the between cluster entropy of the remaining clusters remove this cluster, re-assigning its points to the other remaining clusters, and then begin the worst cluster removal process from this new cluster value. Finally look at the differences between the between cluster entropy calculations for each step in the cluster removal process. When a difference is found which creates a large decrease in between cluster entropy, the correct clustering value is the value preceding this drop in entropy.

In Summary:

1. Set max cluster value higher than the expected amount of clusters
2. Set cluster seeds
3. Assign all points to the cluster which minimises within cluster entropy increase
4. Remove cluster which maximises between cluster entropy when removed
5. Re-assign points to other clusters as in step 3.
6. Redo steps 4-5 until single cluster
7. Correct cluster number is the value before sudden drop in between cluster entropy

This outline leaves the following decisions to be made regarding a variety of parameters for the model:-

7.1.2.2.1 The maximum number of clusters used The maximum number of clusters to use is a function of the size of the cluster seeds we expect and the number of data-points to be clustered. In experimentation this was empirically set to 20.

7.1.2.2.2 How to set the initial cluster seeds In [52] initial cluster seeds are set by selecting `x`random points from the dataset, where `x` is the max number of clusters, and then working through the dataset finding the nearest point to any given cluster. The selected point is then assigned to that cluster and the process continues until all cluster seeds have `n_init` number of points. This method is designed to reduce dependence on the initial structure in the dataset. Our simpler version took the `x` random points as the cluster seeds, as the results did not differ greatly from the experiments using the method in [52].

7.1.2.2.3 How to pick the next point to be clustered A more involved method is presented in [52] but for this system we decided to work through the points in order as the relative bias this introduced was minimal.

7.1.2.2.4 How to measure within cluster entropy The method of measuring within cluster entropy is based on a parzen density estimation window[76] with a Gaussian kernel.

$$WCEntropy(Cluster_k) = -\log(\frac{1}{N_k^2}\sum_{i=1}^{N_k}\sum_{j=1}^{N_k} G(x_i - x_j, 2\sigma^2 I)) \tag{7.1}$$

where $G(x,H)$ is a Gaussian kernel defined as

$$G(x,H) = \frac{1}{\sqrt{|2\pi H|}} e^{-\frac{1}{2}(x)^T H^{-1}(x)} \tag{7.2}$$

7.1.2.2.5 How to measure between cluster entropy Measuring between cluster entropy is based on a parzen density estimation window with a Gaussian kernel, but this time only assessed on points which belong to different clusters.

$$BCEntropy(Cluster_{1\ldots k}) = -\log(\frac{1}{2\prod_{k=1}^{K} N_k}\sum_{i=1}^{N_k}\sum_{j=1}^{N_k} M(x_{ij})G(x_i - x_j, 2\sigma^2 I)) \tag{7.3}$$

where

$$M(x_{ij}) = \begin{cases} 1 & \text{if } x_i \text{ and } x_j \text{ are in different clusters} \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

When we perform clustering using Renyi's Entropy a large number of seeds is initially used, more than are expected to be present in the data, and then bring this down to the correct estimated level. This is because the initial placement is random and likely to be sub-optimal. An optimised method, which could be considered, would be to use K means clustering with a high value for K in order to get the initial random seeds. This would possibly allow a smaller number of seeds to be used for the initial placement and lead to a more efficient algorithm.

As the process used Gaussians, the input points were scaled down by a factor of 100 bringing them closer to O(10). O(1) is normally used [10] but O(10) proved more stable for our application.

Engineering Box

Cleaning A process of cleaning was performed which involved finding any clusters which were not of size greater than 4. If these were found they were re-distributed via the same method as step 4 above. This was merely to allow the fitting of 3D polygons in the next session. There is no fundamental reason that we could not have smaller clusters but they break the quickhull algorithm by generating degenerate clusters.

7.1.2.3 Fitting 3D Polygons

One way to assign structure to the clusters in the level data is to fit a 3D polygon round the points, containing them within it. This is a problem from graph theory of finding a convex hull of points, and can be solved using the `quickhull`[9] algorithm.

Quickhull is a recursive, divide and conquer algorithm, that runs in O(NlogN) time. In two dimensions the points are divided into two groups based on a "mid" line calculation which is a linear operation. The mid line is added to both groups and the recursion is performed on both halves. The base case for recursion is when there are only 2 points in a group. In 3D this is very similar except the mid line becomes a mid plane.

Engineering Box

Visualisation Another, practical, reason for fitting polygons to the data is to allow visualisation of the reclaimed 3D structure. This can help us to evaluate the performance of the clustering algorithm.

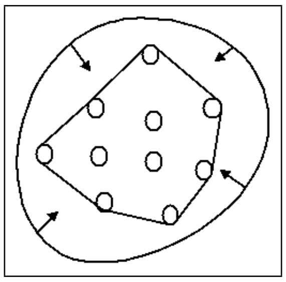


Figure 7.1: Visualising the convex hull

The idea of a convex hull is analogised in 2D by considering an elastic band which is stretched to fit round all of the points , as in Figure 7.1. It will then snap back into the shape of the convex hull. In higher dimensions visualisation becomes tricky but the principle remains the same.

Any clusters which fail to generate a convex hull are disregarded as degenerate[1].

7.1.3 Preliminary Evaluation

The following are basic test cases to demonstrate how the technique works and some of the artefacts that can occur with incorrect clusterings.

Engineering Box

Evaluation Our evaluation was two stage, firstly to test the algorithm on data which we knew should be a very specific shape. This allowed us to determine things that could go wrong even in perfect cases and thus allowed us to more easily evaluate performance on the level data, already knowing some of the problems with our approach. It also allowed us to debug the algorithm more carefully with simple cases rather than trying to spot problems on full levels. This in general is a good idea with systems dealing with this type of data.

7.1.3.1 One Cube

The first test used data points representing a single cube. With parameter settings of variance 0.5 and a maximum of 5 clusters the result shown in figure 7.2 was achieved.

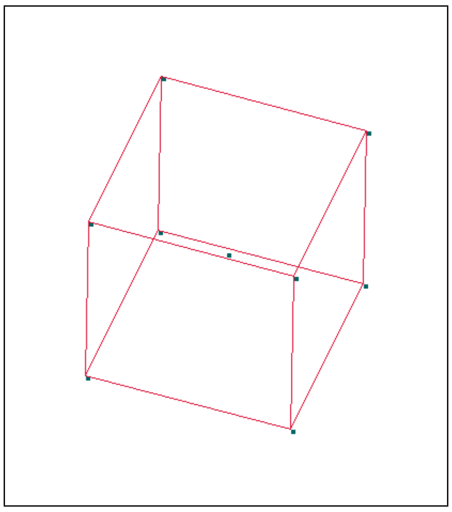


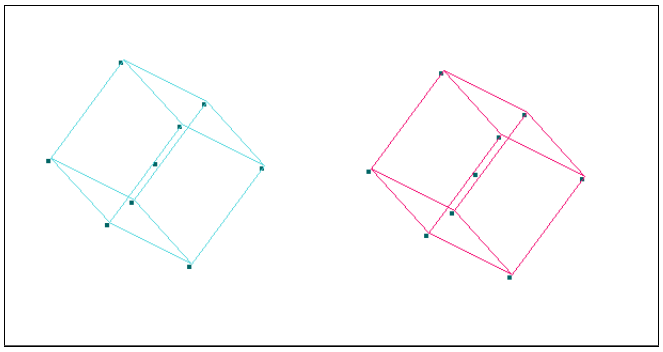
Figure 7.2: Level Model on 1 Cube

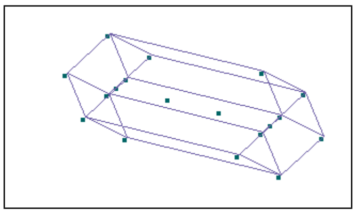
All this serves to show is that the algorithm is capable of correctly finding one cluster and fitting a convex hull.

[1]This can happen if clusters contain co-linear or co-planar points

7.1.3.2 Two Cubes

With the same parameter settings the model sometimes gets the correct result on two cubes as in figure 7.3(a) but also sometimes gets errors such as those shown in figures 7.3(b) and 7.3(c) where one or more nodes from one cluster have become enveloped in the other. The reason for this is that, depending on initial clusterings, there is not a large reason why a point should not be considered part of either cube if the kernel distances are not enough to differentiate between the two. This leads to neither cluster having sufficient entropy to draw the point to them.


(a) Correct Result


(b) Error 1

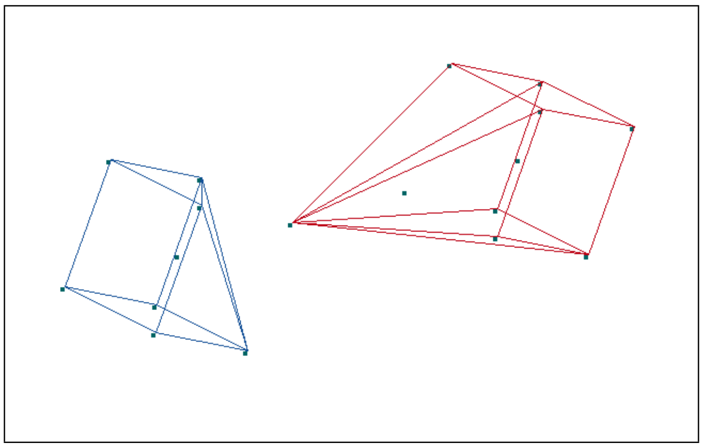

(c) Error 2

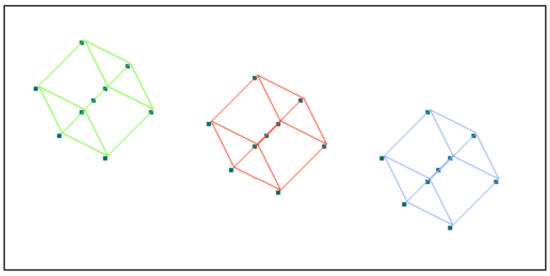
Figure 7.3: Examples involving 2 Cubes

7.1.3.3 Three Cubes

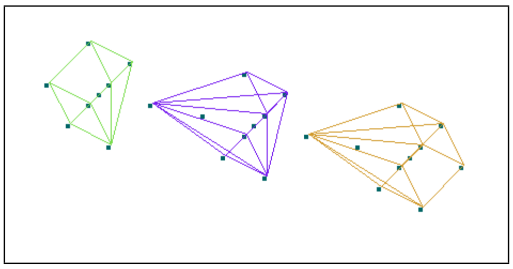
With three cubes the model sometimes generates the perfect case such as in figure 7.4(a) but also sometimes gets errors ranging from those shown in figures 7.4(b) to those in 7.4(c).

Engineering Box

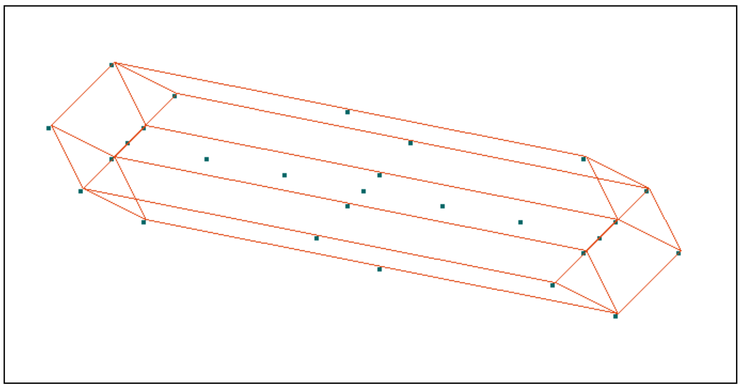
Neither are drastically wrong, and indeed both are practically of use, but they show the range and type of behaviours which can be exhibited by the modelling.



(a) Correct Result



(b) Error 1

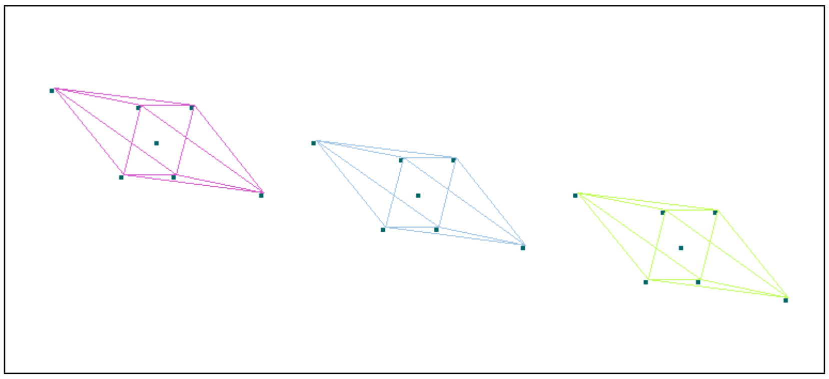


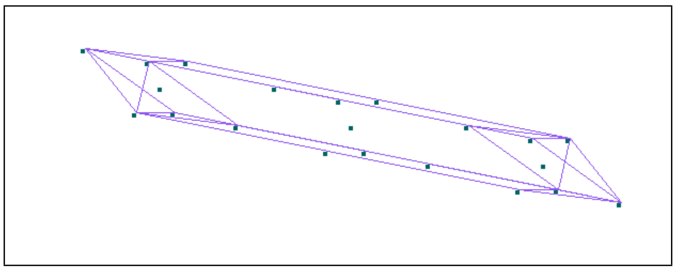
(c) Error 2

Figure 7.4: Examples involving 3 Cubes

7.1.3.4 Three Diamonds

On three diamonds the model sometimes generates the perfect case such as in figure 7.5(a) but also sometimes gets errors ranging from those shown in figures 7.5(b) to those in 7.5(c).


(a) Correct Result


(b) Error 1

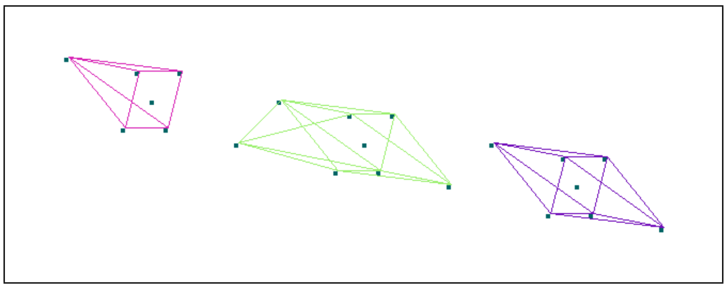

(c) Error 2

Figure 7.5: Examples involving 3 Diamonds

7.1.3.5 Discussion

In most of the cases tested in this section the results were acceptable based on our intuitions about what a diamond should look like. In many cases an argument could even be made that the wrong results were actually not bad enough, theoretically, to be considered wrong as in the case of 2 and 3 cubes.

7.1.4 Game Level Tests

The model was run over some of the game levels. Each parameter setting of the model was given 5 trials to determine performance and variance in the resulting models. In each case a screen-shot of the level was taken and the number of degenerate clusters was recorded along with the size of each cluster and the number of clusters generated. These values were then averaged over the trials (a median was used as it was more relevant than a mean for evaluation of this model). In every experiment the initial clustering seed number was set[2] to 20 and the variance of the Gaussian kernel was tested at the values 80,40,20,10 and 5 (Note that these variance values are relative to the scaling factor used in the level modelling). The results for four of the levels are displayed below. Only the best variance result is shown for each level. The decision to show only these levels was made in the interests of remaining concise. It is also true that the levels shown offer a reasonable span of the size classes, discussed in part 1, covering the categories of small, medium to large and large. The rest of the levels and extra variance visualisations are included in appendix A.

In each case levels are considered in the abstract and the most obvious level features, from the point of view of a human, are annotated. The modelling output is then correlated to these features to determine modelling performance. Numerical measures of performance, such as graphs of average cluster sizes and standard deviations of cluster numbers, are presented.

Lines of differing colours have been used to represent the convex hull of each cluster of points. The colours are not significant, they are simply a tool for showing the different sections, and were generated randomly in the visualiser.

[2] 20 was used because any clustering size larger than 20 was both unlikely given our level intuitions and also becoming computationally unwieldy

Engineering Box

Assumptions The biggest problem we had with evaluating this technique over whole levels was that we had to make assumptions about the levels purely by eye-balling them. Anything above and beyond this was difficult to scientifically justify. As such the evaluation is based on seeing if our polygon fitting matches our intuitions. It is not hard to make arguments against our conclusions drawn from this evaluation method but it is difficult to suggest an alternative method which would be more suited given the modelling requirements.

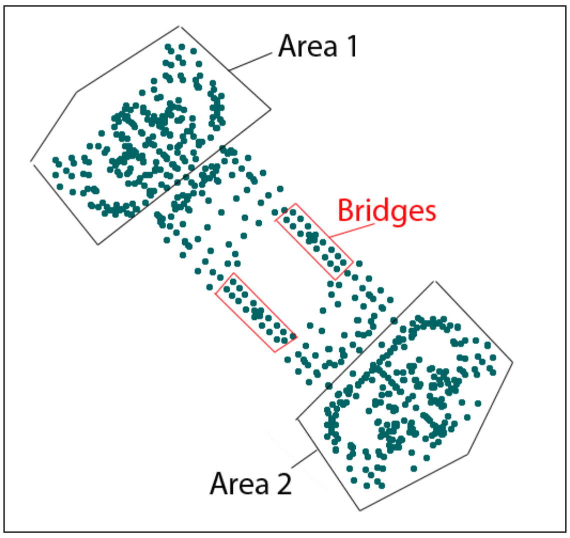
7.1.4.1 Absolute Zero

In AbsoluteZero, one of the larger levels, there are level features as per figure 7.6(a). The main features are two very pronounced areas on either side of the level contained inside large structures along with two bridge sections connecting them. Ideally a model should be able to pick out these features.

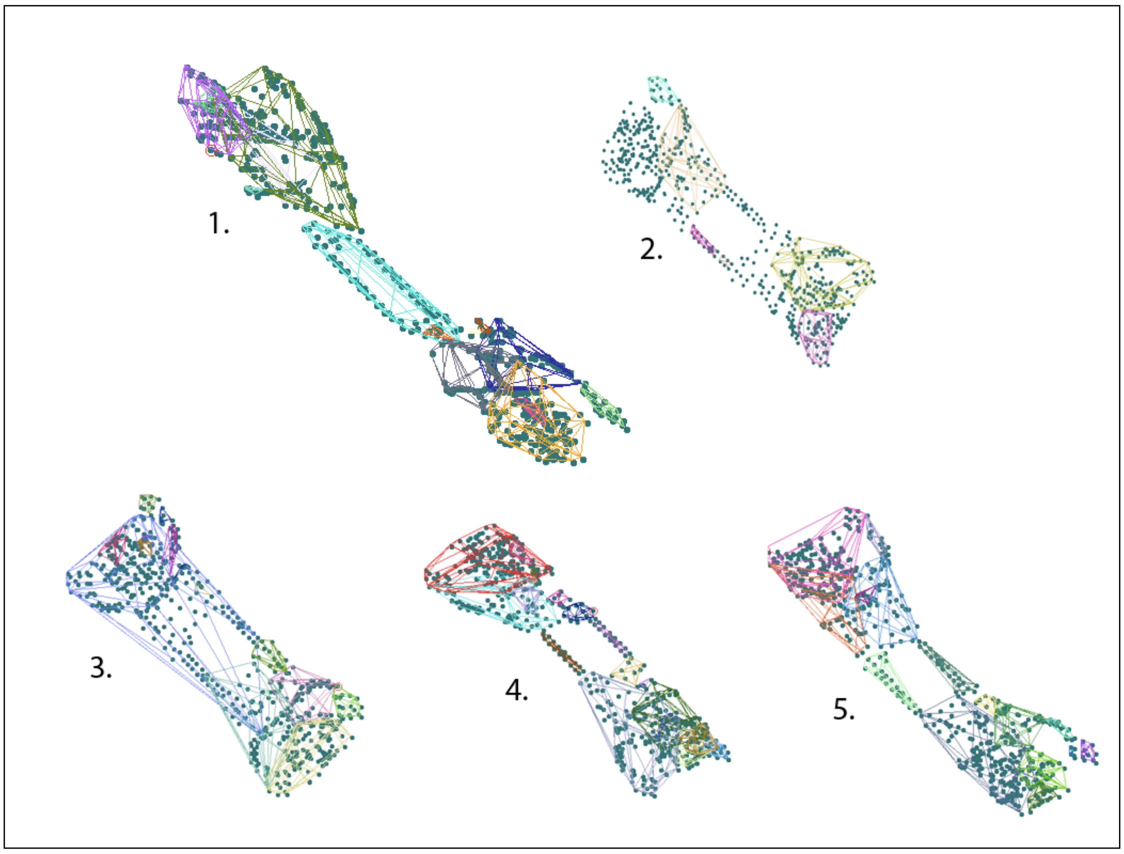
7.1.4.1.1 Variance 5 Visualisation: For most cases the variance 5 model obtains good results with good separation of compartments and definition of the relevant areas of the level. In case 1 the two bridge sections have been considered as one whole but this is still useful and well defined. Case 3 is the worst result obtained. In this case both bridge sections have been absorbed into larger end sections mostly attached to the left hand side of the main structure.

In almost all cases the modelling creates a large structure either side of the bridges and often splits it up into smaller areas. This makes sense as it is expected that the internal structure of these large indoor sections will have some intricate structure so anything which applies structure, which is sensible, to these should give some modelling power at a later stage.

7.1.4.1.2 Larger Variances With larger variances the model ends up accidentally attaching nodes to incorrect polygons. This could be due to the effect of the parzen Gaussian density estimators becoming too thin and not generating good enough information at the more concentrated areas of the clusters. There is often a lot of overlap between sections and bad definition of the bridge elements. The areas tend to bleed into each other and there is a much higher level of granularity in the modelling of the



(a) Expected Features



(b) Polygon Fitting - Variance 5

Figure 7.6: Absolute Zero Visualisations

left hand area than the right.

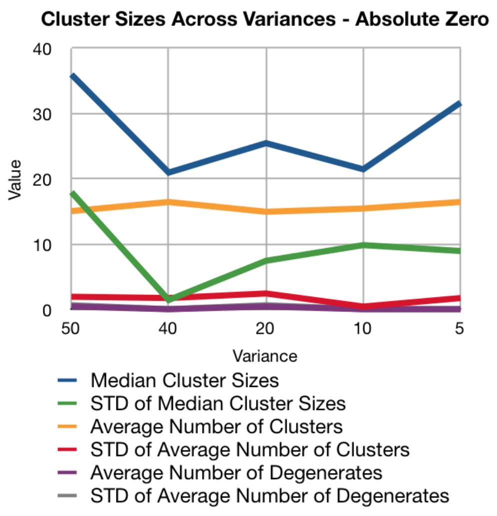
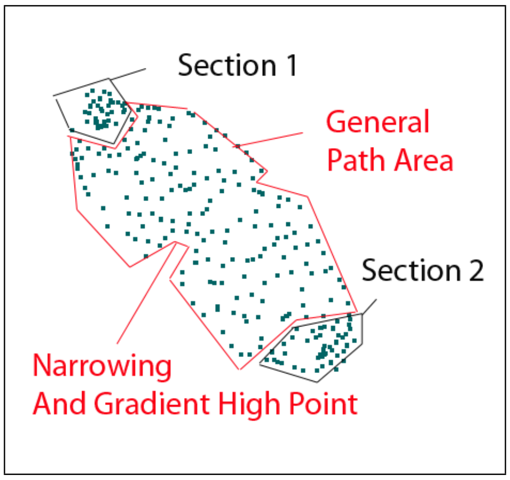


Figure 7.7: Statistics Graph for Absolute Zero

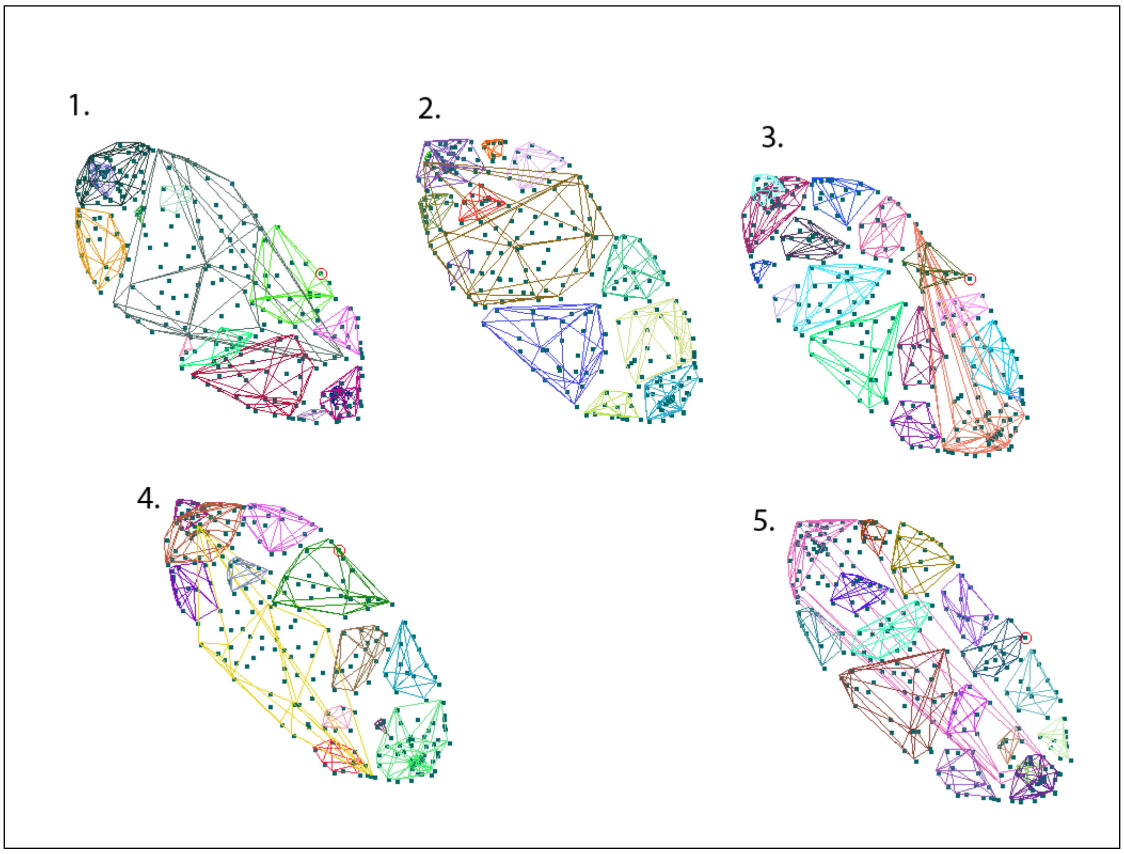
7.1.4.1.3 Conclusions from statistics: Figure 7.7 shows that for all variance values the average number of clusters generated is approximately the same. The standard deviation is relatively low across the board showing that each process generates roughly the same number of clusters for this level with the main difference between each variance value being the size and shape of clusters rather than number of clusters generated, particularly the median size of clusters generated and the standard deviation from this median. The lowest average median cluster size and lowest standard deviation is generated from variance 40 showing a stability point there. Given that in some cases this value obtained a perfect result these statistics may be indicative that this is the best value for this level. Overall a variance of 5 seemed to visually suggest a better modelling performance. In most cases very few degenerate clusters are generated from the process showing that the algorithm is robust.

7.1.4.2 Maul

Maul is one of the smaller levels in the game, displaying the common characteristic of having two well spaced out areas, one either side of the level, separated by a path between them. The path narrows in the centre of the level and is relatively wide in all other places. The level has a height gradient with the centre point of the path being the highest point and the areas either end of the path being the lowest.



(a) Expected Features



(b) Polygon Fitting - Variance 5

Figure 7.8: Maul Visualisations

7.1.4.2.1 Variance 5 Visualisation: The separation of different sections is acceptable. In general the modelling of this level is not great, mainly because the 3D point cloud does not reflect, very accurately, the features of the level. As an observer looking at the point data, it is hard to pick out the key features without prior knowledge of the level. The model still manages to form some sensible polygons which can be used to segregate the model into portions. There are some artefacts contained in the modelling such as certain polygons at one end of the level becoming connected to clusters at the other end of the level creating a spanning structure of no modelling use.

7.1.4.2.2 Conclusions from visualisation: The visualisation suggests that values 5 and 10 are slightly better than the others. The variance value 5 seems to have less, larger, overarching structures which span the whole level and in general the end points of the level are better defined, as such this seems to be the optimal value.

In general the modelling splits the pathway into several smaller sections rather than one large pathway. As long as these sections are separated this will be a useful model, only creating problems in the areas between these sections which will have no direct representation.

Engineering Box
This problem can be avoided by a careful treatment when assigning credit during the section concerning area modelling.

The sections either side of the path are relatively well represented.

7.1.4.2.3 Conclusions from statistics: The statistics show that most of the models are pretty similar in their performance with all displaying generally similar statistics. There is a slight tailing off in the standard deviation of the median sizes in the variance 5 and 10 cases showing that these are the better values to chose.

7.1.4.3 Moon Dragon

Moon dragon is another of the largest levels in the game. It features two tower/temple sections at either end of the level separated by a large number of paths between. It also features an indoor cave section between the two but this is hard to discern from the point data alone. This said the pathways between the sections are feasibly separated and as such would be good candidates for expected level features.

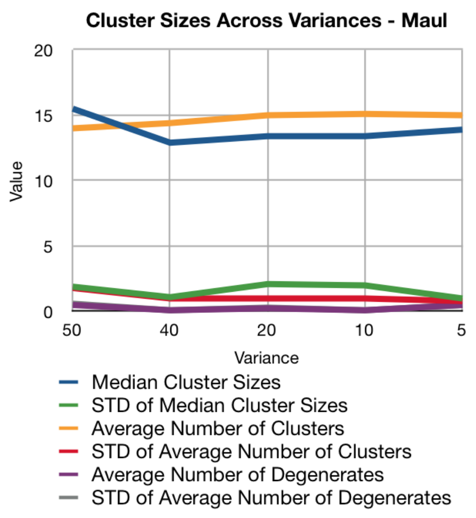


Figure 7.9: Maul - Statistics

7.1.4.3.1 Variance 40 Visualisation: Cases 3 and 5 are very well defined and the cave section is even picked out in each case.

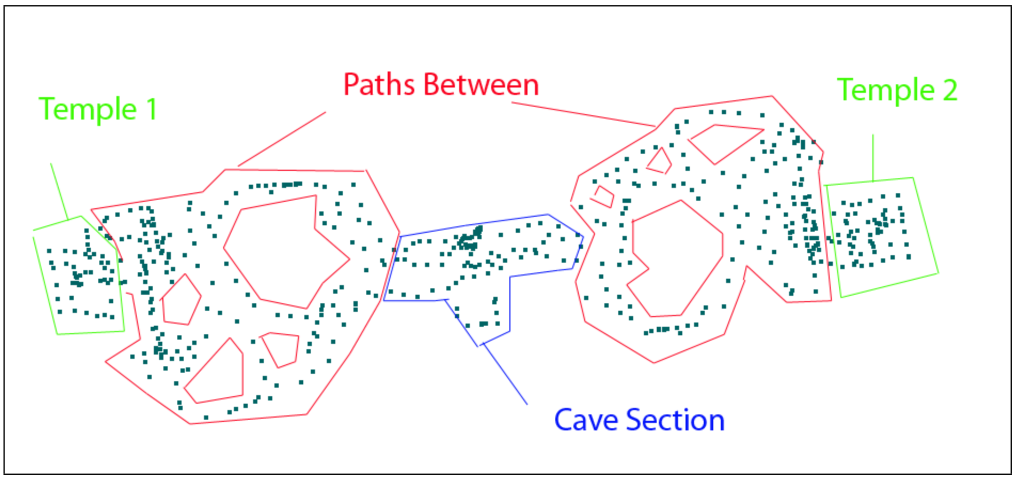
7.1.4.3.2 Conclusions from visualisation: The visualisation shows that 40 was the optimal setting for this level. In most cases the level features were well represented. The paths were mostly well separated.

On the negative side, the large cave section occasionally extends slightly too far towards the temples at either end of the level. For the most part though the modelling looks quite good.

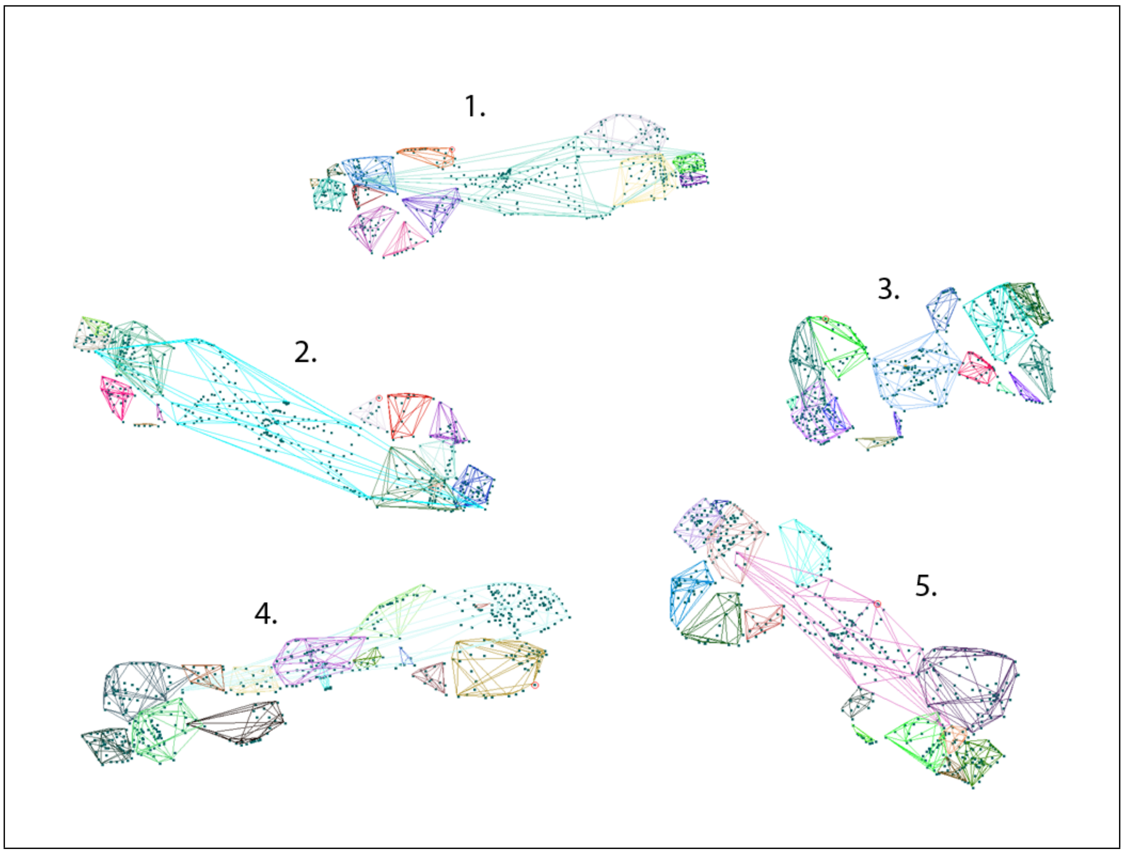
7.1.4.3.3 Conclusions from statistics: The statistics show no real reason why 40 would be an optimal value for this level but they do show that the result for 40 is closer to the result for 5 which seems to suggest that they are exhibiting similar performance. It could be that the standard deviation is higher in these cases because they generate roughly the same output as the other models but also generate a few much better cases which make the deviation in performance larger.

Engineering Box

General SD Assessment This demonstrates a problem with direct analysis of standard deviation in this type of scenario where a population contains mainly low performing individuals with a few strong members.



(a) Expected Features



(b) Polygon Fitting - Variance 40

Figure 7.10: Moon Dragon Visualisations

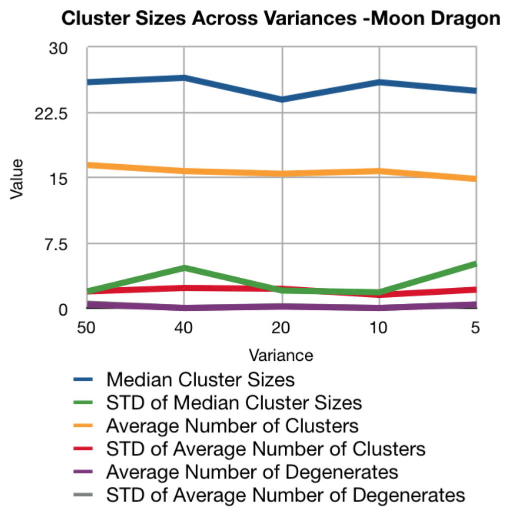


Figure 7.11: Moon Dragon - Statistics

7.1.4.4 Sun Temple

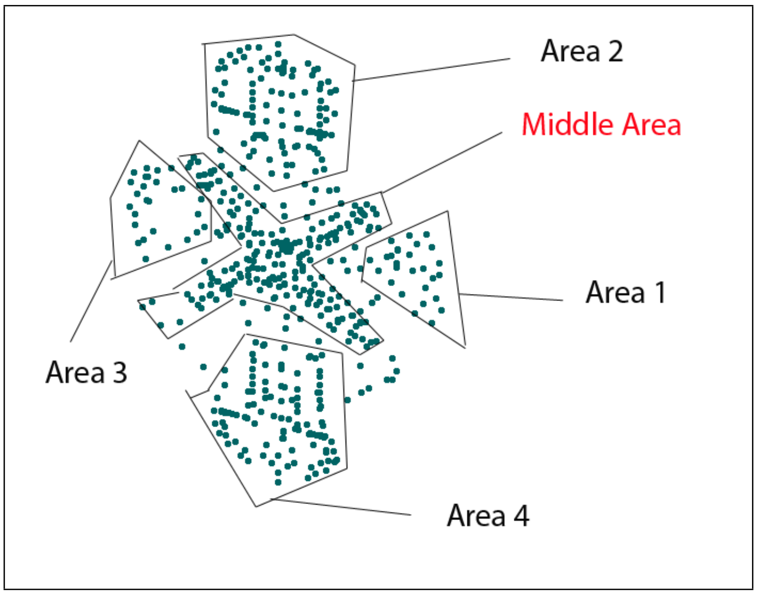
Sun temple is one of the larger levels of the game consisting of 4 angular chambers on the sides of a large middle section where play can occur. These form a compass shape with points on north, east, south and west.

7.1.4.4.1 Variance 10: This setting leads to the best models. There is good definition around the edges and case 5 is close to a good representation of the level.

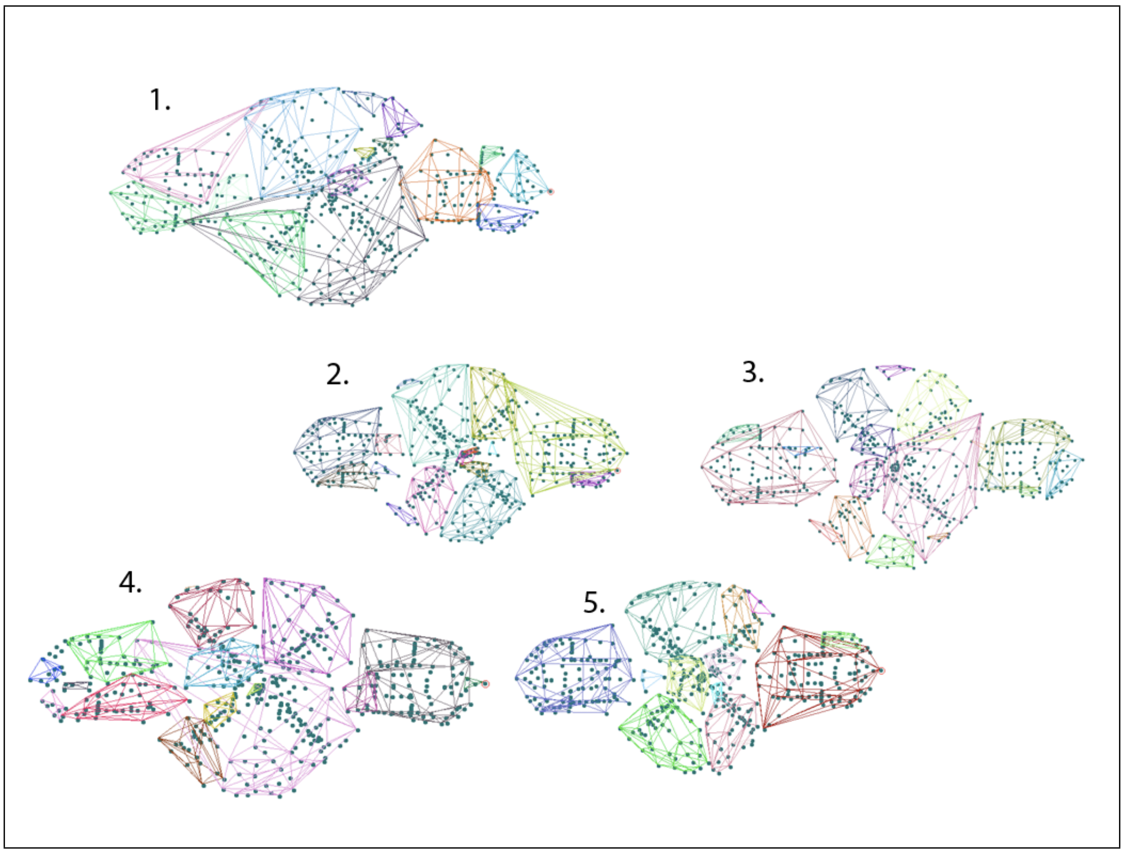
7.1.4.4.2 Conclusions from visualisation: The visualisations show a value of 10 or 5 as being optimal. The middle section is not as well represented as was expected, and there is no defined *star* shape, but the compass point sections in the corners of the level are generally sharp with little overlap. In some cases they are split into smaller sections but this is not necessarily a bad thing as has already been touched on.

With higher variance values there tends to be more blurring between the sections with much less definition.

7.1.4.4.3 Conclusions from statistics: The statistics graph suggests that a variance value of 10 generates clusters that have a slightly lower standard deviation from their median pointing to 10 being the best variance value for this level.



(a) Expected Features



(b) Polygon Fitting - Variance 10

Figure 7.12: Sun Temple Visualisations

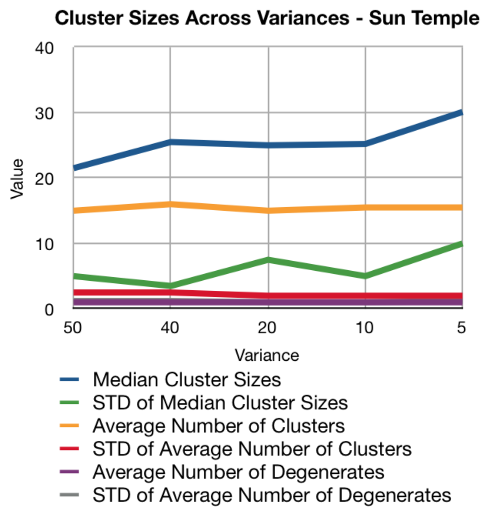


Figure 7.13: Sun Temple - Statistics

7.1.4.5 General Level Modelling Conclusions

In most cases the modelling generates a 3D polygon map, usable as a representation of the level. Key features are shown in the maps but in general the result is a more vague description of the level.

Engineering Box

Stability In all cases the Renyi level modelling procedure generates very few degenerate clusters. This is good because it shows that the algorithm will rarely fail outright; generating data which is unusable.

In a large majority of the cases a smaller setting for the variance value leads to better models, in particular a setting of 10 works well. Scaling this up amounts to a variance of approximately 1000 which ranges between a quarter and a tenth of the level size. Examining the magnitude values alongside the actual values selected for the optimal clustering exposes a pattern. The variance of the gaussians chosen must be around the magnitude of the level. There is some flexibility either side but in general taking the level magnitude as the variance value the results will be acceptable and the modelling relatively stable.

The PCA magnitude value gives a stable measure of the level of variance in node positions in the level. As these node positions are generated to give accurate level navigation they reflect the level and as such can be used as a measure of the variance within the level. Of the results generated in the previous section MoonDragon is the most varied (A result which is backed up by observations of the level itself, showing it to have a vast array of different layout features and variance in the spacing of the path node as a result of this). A good comparison is between this and Gael, one of the smaller levels not tested in this section. Gael is one room and has little variance in the level shape or design. A PCA magnitude of 0.035030820732941476 confirms our conclusions about PCA magnitude in relation to model variance settings.

Level	Standard Magnitude	PCA Magnitude
Absolute Zero	12.192	0.4234
December	20.1869	1.3945
Deck17	5.3382	0.1421
Face3	22.2771	0.1128
Face Classic	16.5066	0.1833
Maul	9.803	0.1068
Moon Dragon	30.6474	2.6968
Sulphur	4.6937	0.0112
Sun Temple	10.52231	0.0732

Figure 7.14: Table of Magnitudes

Figure 7.15 shows the correlation between the two measures of magnitude of the level and the chosen variance value. The PCA magnitude better matches the overall pattern of the chosen variance line, while the standard magnitude matches better the changes in scale.

The algorithm for setting the variance of the model is as follows:

7.1.4.5.1 Level Model Initialisation Algorithm

Begin at 0 with a variance of 5 add 5 to this for every 0.5 benchmark that the level sets on the PCA magnitude scale. For instance if the PCA magnitude value is 1.7 then this has passed the benchmarks for 0.5,1 and 1.5 and thus should have a value of 20.

This method doesn't explain all the data exactly but gives a good rough guide which, given some of the flexibility in the chosen models, is good enough.

7.1.5 Future Work

One possibility for improving the level modelling algorithm is to artificially inflate the polygons to make them less angular and encompass the surrounding area using

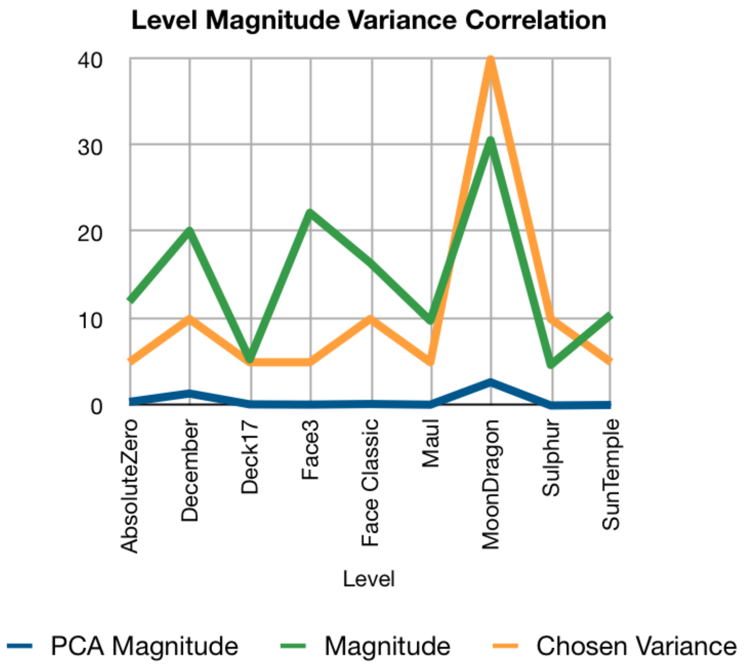


Figure 7.15: Level Magnitude Variance Correlation Graph

bubbling with fixed variance gaussians.

Engineering Box

This helps to capture some of the points of the level space not covered by the polygon fitting but has the disadvantage of possibly creating overlapping polygons. This makes it difficult to attribute responsibility for level points to any polygonal area. One way to deal with this is to treat the polygons as probability distributions and then assess the probability of being in each of the overlapping polygons.

7.2 Area Correlation Model

The level model itself allowed us to gain information about how the level was shaped and sized but this information is useless unless we can gain a performance increase in some way from it. One way to do this is to correlate the number of kills and deaths, within a match, to specific areas/polygons of the level. The game-types are all based around shooting adversaries in order to score more points than them. Even though only one game-type directly rewards kills and not dying, all three intrinsically involve staying alive long enough to achieve certain goals.

The 3D position of kills and deaths can be checked to see if it falls within one of the polygons defined in the level model. By correlating this data we can show which areas are playing host to the most kills and deaths.

Engineering Box

Hierarchical Learning One of the major challenges this particular model posed was that it changed slightly the way we thought about the system. Previously we had conceptualised a very clean 3 layer architecture with lateral modularity between the layers. But having techniques built on other techniques within layers suggested that we were actually dealing with a more complex hierarchy in each layer. In practice this was not a problem but did impose an ordering on the learning techniques which was not ideal. For instance using the area correlation model without first performing the cluster fitting would not work and these models were clearly dependent. Thus the modularity obtained was not quite as clean as first expected.

Point Responsibility Probably the biggest engineering challenge involved with this model was finding a quick way to assign responsibility of particular polygons for different observations. This in itself is not an interesting scientific problem but finding the optimal method for working in a real-time environment made it slightly more interesting. This became particularly interesting considering that we already had setup a separate learning thread during execution offering two options. The first option was to have a quick but sub-optimal measure of distance from each polygon and the second was to have a more intensive option but run it on the delayed learning thread.

The most obvious mechanism for testing which cluster a point is in is to re-generate each convex hull in the level model with and without the new point and see if the hull changes. If it does then the point must not have been contained within the original hull and thus must lie outside the polygon.

Engineering Box

Outliers With this method, it is possible for points to land out-with all of the hulls within the level. A fix would be to take one large convex hull for the entire level set and have this as another cluster.

Efficiency A more efficient method was to associate a cluster number with each navpoint in the level. This was performed at the level modelling stage before any real-time calculations were performed. When an observation was made the cluster number was assigned to that of its nearest nav point. Thus the problem of finding the correct cluster does not involve any re clustering and is achieved in $O(log(n))$.

7.2.1 Experiments and Evaluation

To evaluate the area correlation model tests were run on some of the strategies found to be most effective in [40]. In each case 5 trials were run with 3 bots versus 3 in-game bots. Although not assessed, the scores of the games were recorded so that consistency between trials could be considered. The average number of kills and deaths which occurred in each polygon in the level model along with the standard deviation of these values (For the sake of correlating results the same instance of the level model was used in each of the 5 trials) are presented. The trials were repeated across 3 levels for each of the 3 different game types, however the results are only presented for one level in each game-type [3]. The strategies used are presented for each example. In some cases a brief description of the level is given. In other cases descriptions can be found in section 7.1.2 or in the appendix.

[3]full results can be found in the appendix, along with the clustering diagrams

Engineering Box

Evaluation Strategy When it came to evaluating the model we decided the best strategy was to run some trials and then just observe the results. The point of this was not show that the model could be used for a specific purpose but more to evaluate what kind of information it was telling us. We were trying to assess its typical behaviour and then determine how this could be used. This fits with the general prototypical nature of systems design for this type of architecture. The modular nature of the machine learning layer allows ideas to be tested and subsequently modified or used in an alternative way to the initial design. This is not a failure, it is more of a natural tuning process, the world is full of systems, objects and conceptual designs which are being used for purposes other than their initial design. What tends to happens is that the systems are created for one task, modified in light of working better for another and then refined to the point where the task and solution seem like they were designed for each in the first place. This process is just extending this idea back to the initial design stage.

7.2.2 Team Deathmatch

For each trial a death match was set up with the goal score of 60 kills. The following LCC strategy was used to control each bot.

```
Strategy 2.1

a(random,Id)::null<--visiblePlayer(Location) and
strafeAttempt(Location,Location)
a(random,Id)::null <-- movementAttempt(random_play)
```

This protocol tells each bot to either select a random health or ammo target unless an enemy is visible. If an enemy is visible they will approach the enemy. The bots were set on non-friendly, so they would fire on any visible enemy with whichever weapon they currently held. The idea was to choose a strategy which had a good chance of covering a decent portion of the level space, allowing us to populate the model.

As we know the average score of each game we can compare this with the average kill and death counts to determine how many of the deaths and kills were not being accounted for by the model.

7.2.2.1 Deck17

	Deck17 Area Correlation Trials, 5 Trial Averages	
	Enemy Value	Our Value
Average Score	60	40.6
Standard Deviation from Score	0	7.7

Figure 7.16: Deck17 Area Correlation Model Trials

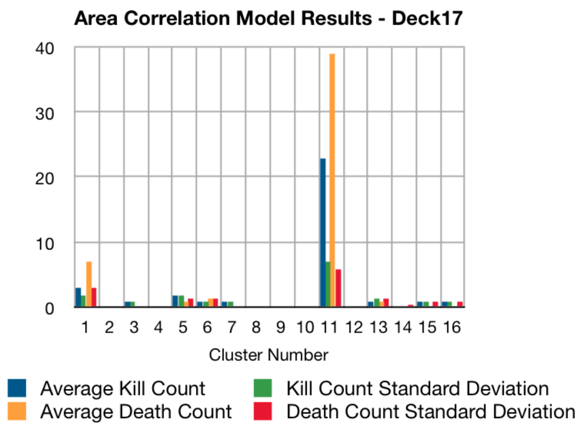


Figure 7.17: Deck17 Area Correlation Model Statistics

Figure 7.17 shows that there is a pattern of play on the level Deck17 which is central to clusters 1 and 11 (The level clustering for Deck17 is not shown but can be found in the appendix). If the bots wished to avoid confrontation they could avoid these clusters. For instance one example behaviour could be to avoid these clusters while collecting health/weapon pickups and return to them once these items had been acquired. Note that we could easily create this strategy without knowing which clusters were selected and as such this model has learned a useful abstraction.

We might have hoped to observe certain clusters being better for kills and others better for deaths because currently the model is just modelling which areas the most play occurs in. This is useful but not of optimum use.

7.2.3 Capture the Flag

The goal for the capture the flag trials was to make 6 successful flag captures. The following LCC strategy was used:

```
Strategy 2.2

%% base_defender , makes sure they don't get our flag to begin with

a(base_defender,Id)::sawTheFlag(L) => a(base_defender,Id) <-- visibleOwnFlag(L)
and strafeAttempt(L,L) and enemyHasFlag(true)
a(base_defender,Id)::strafeAttempt(L,L) and enemyHasFlag(true) <--
sawTheFlag(L) <= a(base_defender,Id)
a(base_defender,Id)::null <-- visiblePlayer(Location)
and strafeAttempt(Location,Location) and enemyHasFlag(false)
a(base_defender,Id)::null <-- currentWeapon(W) and prologConstraint(W =
assault_rifle)
and movementAttempt(nearest_weapon_pickup) and enemyHasFlag(false)
a(base_defender,Id)::null <-- movementAttempt(localised_play(enemy_flag_point))
and enemyHasFlag(true)
a(base_defender,Id)::null <-- movementAttempt(localised_play(own_flag_point))

%% enemy defender , comes into play when they have our flag

a(enemy_defender,Id)::null <-- changeToRole(flag_hunter) and enemyHasFlag(false)
a(enemy_defender,Id)::null <-- changeToRole(flag_carrier) and hasFlag(true)
a(enemy_defender,Id)::sawTheFlag(L) => a(enemy_defender,D) <--
visibleOwnFlag(L)
and strafeAttempt(L,L)
a(enemy_defender,Id)::strafeAttempt(L,L) <-- sawTheFlag(L) <=
a(enemy_defender,D)
a(enemy_defender,Id)::null <-- visiblePlayer(L) and strafeAttempt(L,L)
a(enemy_defender,Id)::null <--
movementAttempt(localised_play(enemy_flag_point))

%% flag_hunter , goes after the flag

a(flag_hunter,Id)::null <-- changeToRole(flag_carrier) and hasFlag(true)
a(flag_hunter,Id)::null <-- changeToRole(enemy_defender) and enemyHasFlag(true)
a(flag_hunter,Id)::canSeeFlag(Location) => a(flag_hunter,F) <--
visibleEnemyFlag(Location)
and strafeAttempt(Location,Location)
a(flag_hunter,Id)::strafeAttempt(Location,Location)
<-- canSeeFlag(Location) <= a(flag_hunter,F)
a(flag_hunter,Id)::movementAttempt(otherBot(F)) <-- gotTheFlag <=
a(flag_carrier,F)
a(flag_hunter,Id)::null <-- movementAttempt(enemy_flag_point)

%% flag carrier , doesn't care about defending

a(flag_carrier,Id)::null <-- changeToRole(flag_hunter) and hasFlag(false)
a(flag_carrier,Id)::gotTheFlag => a(flag_hunter,F)
<-- movementAttempt(own_flag_point) and enemyHasFlag(false)
```

```
a(flag_carrier,Id)::gotTheFlag => a(flag_hunter,F) <--
movementAttempt(localised_play(own_flag_point))
and enemyHasFlag(true)
a(flag_carrier,Id)::gotTheFlag => a(flag_hunter,F)
<-- visiblePlayer(Location)
and strafeAttempt(Location,Location) and enemyHasFlag(true)
```

7.2.3.1 Face3

	Face3 Area Correlation Trials, 5 Trial Averages	
	Enemy Value	Our Value
Average Score	6	0
Standard Deviation from Score	0	0

Figure 7.18: Face3 Area Correlation Model Trials

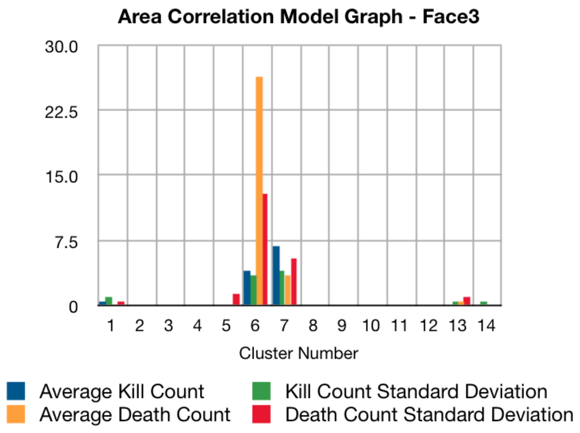


Figure 7.19: Face 3 Area Correlation Model Statistics

Face3 is a level which has two large towers at either end of the level. Each team's flag is contained within their tower area. The area of most deaths correlates to the tower occupied by the enemy. This shows that the area modelling is developing an understanding of the play within the level model. Cluster 7 is the tower occupied by our team and it was no surprise that the largest kill count would occur in this area. Although this fact could easily have been annotated or deduced, the model learned this correlation with no annotation. This means that in levels with different characteristics the model should be able to learn these correlations regardless of level nature. The only threat to this usefulness is that the standard deviation value is relatively larger for the larger averages. This only becomes a large problem if the deviation value becomes larger than half the the average value or high enough that it brings the confidence interval low enough to signal that there is no differentiation between clusters.

It is also interesting that outside of the tower areas there are very few other areas that have any activity. This means there must be a large area of the level in which play does not occur. This could be useful for constructing an avoidance strategy.

7.2.4 Double Domination

The goal for the double domination games was to score 6 points. The LCC strategy used for double domination was as follows:

```
Strategy 2.3

a(extra_support,Id)::movementAttempt(domPoint(P)) and team(T) and
domPointControlledBy(P,T2)
and prologConstraint(\+ T = T2) <-- movingToPoint(P) <= a(extra_support,Id)
a(extra_support,Id)::movingToPoint(P) =>
a(extra_support,Id)<--movementAttempt(domPoint(P))
and team(T) and domPointControlledBy(P,T2) and prologConstraint(\+ T = T2)
a(extra_support,Id)::movementAttempt(localised_play(domPoint(b))) <--
goingWandering <= a(b,AID)
a(extra_support,Id)::movementAttempt(localised_play(domPoint(a))) <--
goingWandering <= a(a,AID)
a(extra_support,Id)::null<--movementAttempt(nearest_weapon_pickup) and
currentWeapon(W)
and prologConstraint(W = assault_rifle)
a(extra_support,Id)::sawThem(L) => a(extra_support,Id) <-- strafeAttempt(L,L)
and visiblePlayer(L)
a(extra_support,Id)::strafeAttempt(L,L) <-- sawThem(L) <= a(extra_support,Id)

a(a,Id)::movementAttempt(localised_play(domPoint(a))) and team(T) and
domPointControlledBy(a,T)
<-- goingWandering <= a(b,AID)
a(a,Id)::movementAttempt(domPoint(a)) <-- goingWandering <= a(b,AID)
a(a,Id)::goingWandering => a(b,BID)<--movementAttempt(nearest_weapon_pickup) and
currentWeapon(W)
and prologConstraint(W = assault_rifle) and team(T) and
domPointControlledBy(a,T)
then goingWandering => a(extra_support,ES)
a(a,Id)::goingWandering => a(b,BID)<--visiblePlayer(L) and strafeAttempt(L,L)
and team(T) and domPointControlledBy(a,T) then goingWandering =>
a(extra_support,ES)
a(a,Id)::null<--movementAttempt(localised_play(domPoint(a)))
and team(T) and domPointControlledBy(a,T)
a(a,Id)::null<--movementAttempt(domPoint(a))
```

```
a(b,Id)::movementAttempt(localised_play(domPoint(b))) and team(T)
and domPointControlledBy(b,T) <-- goingWandering <= a(a,AID)
a(b,Id)::movementAttempt(domPoint(b)) <-- goingWandering <= a(a,AID)
a(b,Id)::goingWandering => a(a,AID)<--movementAttempt(nearest_weapon_pickup)
and currentWeapon(W) and prologConstraint(W = assault_rifle) and team(T)
and domPointControlledBy(b,T) then goingWandering => a(extra_support,ES)
a(b,Id)::goingWandering => a(a,AID)<--visiblePlayer(L) and strafeAttempt(L,L)
and team(T) and domPointControlledBy(b,T) then goingWandering =>
a(extra_support,ES)
a(b,Id)::null<--movementAttempt(localised_play(domPoint(b)))
and team(T) and domPointControlledBy(b,T)
a(b,Id)::null<--movementAttempt(domPoint(b))
```

7.2.4.1 Atlantis

	Atlantis Area Correlation Trials, 5 Trial Averages	
	Enemy Value	Our Value
Average Score	0.6	6
Standard Deviation from Score	0.799	0

Figure 7.20: Atlantis Area Correlation Model Trials

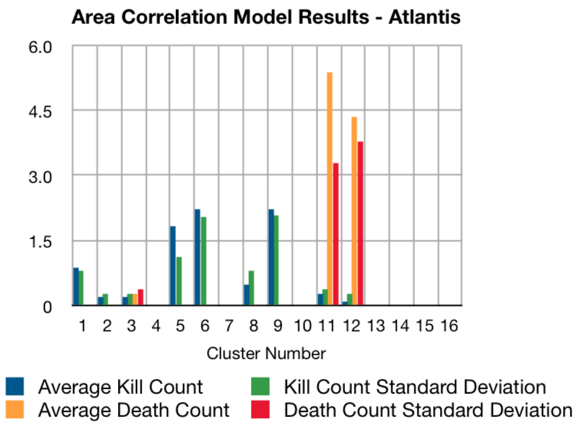


Figure 7.21: Atlantis Area Correlation Model Statistics

Figure 7.21 shows that clusters 11 and 12 have the highest death counts. These correlate to the area near where the enemy spawns and the closest domination point to that area. This suggests that the enemy are employing a strategy based around proximity to the points. Clusters 6 and 9 demonstrate the flip-side of this equation, representing the closest domination point to where the our bots spawn and our spawning location.

The biggest problem with this result is that the standard deviation values are so high that the confidence intervals of the most important clusters are insufficient to place trust in the result. Although the result seems obvious and intuitive from the visualisations we are still not sure whether it can be trusted.

7.2.5 Overall Area Correlation Modelling Conclusions

In almost all cases certain clusters witness significantly more of the action than others. These correlations fit with our expectations regarding play in the level and as such represent a good natural estimator of our target function. This information can be used to guide decision making as discussed briefly above. These models can show which areas of the level are most active, which areas are leading to the most deaths and which the most kills. The models can guide activities such as path finding, allowing the bots to navigate the level using criteria based on dynamic game play data.

In reference to the size of the decision boundary, the utilitarian distinction between areas is often large, this is beneficial as it creates distinct areas and thus distinct decision boundaries.

The team death match trials show that on average the modelling looses 30% of the kills and 38% of the deaths. These figures are not too destructive and relatively low considering that only the navigation points were used to create the level models and it is expected that these could be lowered further using bubbling. A more detailed evaluation could be used to show the nature of these lost points but the expected extra performance, from such an activity, is not large.

7.3 Bot Social Utility

One idea, which might help with machine learning, is to measure a social utility [101, 62] for each bot. In traditional machine learning the information coming into the learning mechanism, for whatever purpose, is often treated as being equal in importance in the hope that the modelling procedure can then automatically pick out which data is useful. Altering which data is taken into account is seen as introducing bias into the model and is often referred to as credit assignment. In many standard machine learning cases it is not realistic to be able to sensibly define this bias, but in some cases there is potential to develop heuristics and methods which can help to improve performance. It is also a useful way to help make some techniques robust with sparse amounts of data.

Credit Assignment[45] Giving data some value which is its expected worth or importance in describing the situation and guiding learning[4].

In a normal single agent situation this can be a difficult activity as it requires having prior knowledge about the distribution and generation procedure which the data originated from. In the multi-agent case[45], however, there is often more of a notion of how data was generated and specifically, the context in which it was generated. More is known about the source due to the observer being an agent situated in some environment with some internal state, affected by those around it. Discrepancies in the knowledge common between agents can lead to information being of larger or lesser importance and agent's experience about each other, through argumentation and information gathering, often gives light as to how to perform credit assignment.

The problem with applying it to this system is that there is no real grounding to measure any kind of social utility amongst the bots given the tasks which they were aiming to model.

In particular it is apparent that the *any bot / any role*, nature of the bots dictates that measuring the performance of a particular bot does not offer any real advantage. Due to the communal learning mechanisms all this data is already available and no extra power is gained from separately modelling these social factors. In a later section the performance of the strategy and bots in particular roles is used, to alter the exploration/exploitation trade-off, which touches on this area, but represents a much better approach to it via role utility rather than bot utility.

7.4 Enemy Modelling

In this section we present a model of the enemy bot's movements and general position within the level. We do this by placing a density model over the level. The databank for this density model is based on observations of the enemy's position. These observations are 3D coordinate positions within the level and as such this gives a full density model over the 3D level. We use this density model to give a probability value to every

[4]This is a slight issue of contention as there as several different meanings of the term credit assignment. This definition is the one which is taken as gospel throughout this thesis

navigation point within the level. These probabilities are then updated with each new observation to give each navigation point a value representing how likely it is that the enemy would be at these points.

Hladky and Bulitko provide, in their detailed evaluation of the field of enemy positional modelling in video games[49], a rich list of approaches to this type problem. Of these our approach is most similar to Isla's occupancy maps approach[50] except we learn our maps online during each match rather than from a set of offline logs. This is in-keeping with the online learning approach throughout our architecture.

7.4.1 Motivation

The most dynamic element of the UT environment is the enemy bots. Although scripted, their play is one of the key elements in defining the outcome of matches and as such represents a very rich avenue for learning. [49] shows this to be a valid avenue for research and plausible modelling task to consider and also that the target function can be modelled probabilistically.

The UT in-built bots often display patterns of play which are signature-like in nature. When humans play the game for long enough they also begin to display particular patterns of play[5][87], to such an extent that experienced players can often pre-empt where a player will be before they arrive, performing effective leading. Statically scripted opponents are even more susceptible to this type of out-manoeuvring.

Modelling this enemy information can allow effective play across a number of levels. Enemy play is a variable within the game which remains measurable, in that it is always measurable in some form, but will change with each level. Therefore modelling this play will give a feature invariant which can be used in order to alter the way in which strategies are executed across different levels. This section deals with some different models which give information about how the enemy play the game.

7.4.1.1 Changing Enemy Behavioural Patterns

Modelling any kind of enemy patterns will always be reliant on what we believe these patterns to be before modelling them. As such the biggest challenge for any system is how it deals with changes in enemy play.

We mentioned above that the in-built UT enemies have patterns of play oweing to their scripted nature. But what if these patterns are sophisticated enough to change

[5]This is true for other first person shooter games as well

during a match? In general any enemy which can change its play substantially at any given point is going to be difficult to model. This said the use of density models does allow for some basic alterations to deal with this scenario. Later observations can be given more weight than earlier cases (discounting). Some form of measure could be used to determine how different a new observation is from the given dataset and thus establish whether this represents a new pattern or is just a statistical outlier.

In general, in our case, we assume the enemy has a basic play pattern which is observable with a certain amount of noise. We also assume that this noise is not substantially large enough to cause us to be unable to model the target function accurately.

Later, in the results part of the enemy modelling section, we present a basic experiment showing how our density models change with some basic changes in the enemy's play.

Engineering Box

Adversarial Elements Enemy modelling in UT is interesting because the enemy are the only adversarial element in the game. Although there are other dangers within the environment itself none of them are active or malicious. Enemies have guns so it makes sense to model effective ways of dealing with weapons but it is almost trivially obvious that we should model the enemy themselves in some way. So then the question becomes what properties of the enemy should we model and how do we go about achieving this.

Dimensions To determine this means thinking about what dimensions do the enemy play, do they make certain movements, do they inhabit certain areas, do they travel in packs? One could also think about modelling their choice of weapon but the interface offers no way to determine the weapon that a bot was shot with.

Location The most obvious choice to make is to model the enemy's location within the map, owing to having observations of their location. This is interesting because we are not really dealing so much with a full 3D space as a partial subspace within it. We have a belief that the enemy will inhabit certain areas of the game maps habitually. It could be that they have no patterns to their play but we would certainly hope this is not the case as a fully random enemy will render any modelling pointless from the offset. Alas we know the enemy are statically scripted and as such we believe they will have patterns to their play.

Density Because we believe the enemy to move in patterns it seemed appropriate to try to fit a density to this movement. Thinking ahead we knew that we were going to have to use the data in this model to determine movement in the level. Thus ideally a model which provided not just a single yes or no to particular areas but instead offered a gradient of variation was required.

7.4.2 The Single Gaussian Model

When dealing with patterned observations in 3D space, density modelling is a common practice used to generate probability distributions and create a probability landscape over an area [13, 11, 68, 52, 10, 12, 76, 121]. It is strange then that there has been so little take-up of density modelling in the video games world. Probabilistic and stochastic processes are prevalent so one might be inclined to assume a probabilistic density treatment should be equally so. It is my belief that the reason for this is the often large amounts of data needed to train such models combined with the complicated training mechanisms involving mathematical optimisation is off-putting. Later in this section we deal with a model which is similar to standard density models but who's training is far simpler.

Viewing the enemy presence as being based in one area of the level, it can be modelled using a Gaussian density estimator based on the following formulation:-

$$P(X) = \frac{1}{\sqrt{|2\pi\Sigma|}} e^{-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)} \tag{7.5}$$

Where x is the new observation, μ is the mean of the data-points and Σ is the covariance matrix of the data-points. The observations of the team, or individual, of enemy sightings are then fed into the model to allow parameter estimation. The parameters are set using Maximum Likelihood Estimation::

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x^i \tag{7.6}$$

$$\Sigma = \frac{1}{N-1}\sum_{i=1}^{N} \left(X^i - \mu\right)\left(X^i - \mu\right)^T \tag{7.7}$$

The observations are based on 3D locations within a space where the values range into the thousands. With this in mind the inputs were rescaled to O(10) in order that the values of the covariance matrix be positive and greater than 1 but also not so large that the covariance becomes extremely spread leading to small Gaussian bumps in the probability landscape. As stated in David Barbers learning from data notes on numerical machine learning techniques[10] it is always a good idea to scale data input to roughly O(1) for statistical analysis. It was found during our initial testing that scaling to O(10) gave better estimator performance.

This model allows the assignment, to each path node, of a value representing the probability of enemy presence. This can then feasibly be used to rank complete paths and areas of the level.

Engineering Box

A single Gaussian is often used to fit a density to a function within a space. In our case we knew this was likely to not work as its too simple and we didn't assume that our enemy would play in patterns drawn (as a group) from a normal distribution. However if they did then this model would be a reasonable approximation and should display good performance.

There is a further issue of the number of data-points to store for training. For instance choosing to only have a store of 100 data-points and straight replacing each of these data-points with new ones, when full, places more importance on recency. Alternatively ranking data-points by how long the bot managed to maintain a fight with an enemy and how similar the points are could be considered. These issues are not really explored here but should be considered by anyone looking to extend work in this area via means of improving the individual machine learning techniques. This falls into the realm of credit assignment and more intricate data mining issues.

7.4.3 The Gaussian Mixture Model

The Gaussian mixture model[10, 63] is based on the Gaussian density estimator but instead of only using one Gaussian to model the data multiple Gaussians are combined. This gives more power and tends to avoid some of the averaging effects which manifest with only a single Gaussian. The combination is performed using mixing components to represent how much of the distribution each Gaussian represents.

Engineering Box

If the assumption that a population would exhibit behaviour loosely Gaussian in nature was too strict then it seemed more plausible that each member of said population would exhibit Gaussian-like behaviour.

The overall model is then represented by the following set of equations:

$$P(X|Model) = \sum_{I=1}^{N} P(Model^i)P(X|Model^i) \tag{7.8}$$

Where N is the number of components of the model and each $Model_i$ is modelled by the Gaussian density estimator formula that was used in the single Gaussian model. The mixture components $P(Model_i)$ represent the responsibility of each model component for the final probability and are all 0..1 subject to the constraint:

$$\sum_{i=1}^{N} P(Model^i) = 1 \tag{7.9}$$

In order to update the parameters of this model the standard $ExpectationMaximisation$ algorithm[11, 10] was used on the maximum likelihood parameter update rules:

$$\mu^i_{new} = \frac{\sum_{n=1}^{P} P(i|x^n)x^n}{\sum_{n=1}^{P} P(i|x^n)} \tag{7.10}$$

$$\Sigma^i_{new} = \frac{\sum_{n=1}^{P} P(i|x^n)(x^n-\mu)(x^n-\mu)^T}{\sum_{n=1}^{P} P(i|x^n)} \tag{7.11}$$

$$P(i)^{new} = \frac{1}{P}\sum_{n=1}^{P} P(i|x^n) \tag{7.12}$$

where

$$P(i|x^n) = \frac{P(x^n|i)p(i)}{\sum_{i=1}^{M} P(x^n|i)} \tag{7.13}$$

At every update the Gaussians were reset so that the means were placed on random data-points in the set, the covariances were set to a small multiple of the identity matrix, and the responsibility values were set to $\frac{1}{N}$ where N is the number of components. The parameter update rules were then performed 50 times. This figure was chosen experimentally using a validation set. Some other update procedures, with more specialised stopping criteria, are presented later in this section.

7.4.4 A Gradual Update Procedure

In the standard Gaussian mixture model, training is performed in a batch manner after a certain number of time steps. This provides adequate performance but, given the dynamic nature of the data, it makes sense to train in an online manner. One option for this is to run the EM update procedure for less amount of time but store the output and use this as input to the next cycle of training. This facilitates a more gradual drift. It also fits more naturally with the derivation of the EM rules as a system of insoluble equations[11]. As a plus it allows information to be stored in the Gaussians, over the course of the match, about previous data without explicitly storing this data in the dataset. This process is analogous to an infinite impulse response filter (IIR) from the digital signal processing domain.

Engineering Box

Limiting Covariance One thing that must be ensured is to limit the amount of shrinkage the gaussians perform over the course of the training. Neglecting this means the model can end up developing very narrow thin gaussians. One way to do this is to stop training adapting the covariance once its determinant has reached a suitably small level. The determinant of a matrix is a measure of the volume of the transformation which that matrix represents (It is a measure of the difference in volume seen when the matrix transformation is applied to a hypercube of unit volume). It is also worth only estimating variance and then applying this to all directions of the covariance so as to ensure that isotropic gaussians are used as this will also limit the convergence seen during the training process[10].

7.4.5 Automatically Choosing the Number of Mixtures

A novel idea, which we propose as a contribution to the Gaussian modelling field, is to automatically pick the number of Gaussian components in the mixture model based on statistics regarding the natural clustering and/or shape and size of the level itself. In the previous models it was arbitrarily chosen to use 3 Gaussian components based on the 3 enemy players but there is nothing to say that this is a sensible thing to do. If each enemy had four different modes of play then a better model for the data would be 12 mixture components with 4 for each enemy.

Using the natural clustering of the level to set this parameter is based on the assumption that if the level has certain separated areas then it makes sense to model play in each one of these areas with a single Gaussian component.

The use of level data to refine the number of mixture components is a novel way to deal with the fact that performing automatic clustering of the data at later points in the level is infeasible. It is not known, a *priori*, the number of clusters that will be present in the observed actor data, but information about the level itself can be used to set this parameter.

7.4.6 Limiting Mean Drift

The previous formulation looked at deciding how many mixture components to use based on the suggested amount of natural clustering within the level. Having done this it also makes some sense to limit the means of these Gaussians to movement within these 3D polygons defining areas of the level[13]. This ensures separation between the Gaussians. It also helps to ensure consistency with the assumption that each Gaussian is representing an area of the level.

This might seem like it will flood the level with Gaussians but it is important to remember that the EM update procedure also updates the mixture responsibilities, gaining information about which areas of the level are most likely to contain enemy play but also, within these areas, which parts of the areas are important.

This process yields a constrained Gaussian mixture model. This is similar to the generative topographic mapping[13] where a mixture of Gaussians is constrained by being placed on the surface of a non-linear manifold. In our case no dimensionality alteration is being performed because the particular geometry being used to define the constraints on the Gaussians is not so well defined as to facilitate this, although in principle it could be feasible to us the PCA method from the level modelling section or some non-linear derivative[89].

Engineering Box

Assumptions Our major assumption with the Gaussian mixture model was that the team play was Gaussian in nature and that this could be modelled with some mixture of Gaussians. The reason for this assumption was that we believe each enemy to follow a script of play. This may include going for weapons in a certain order determined by some condition. When the enemy bots see an opposing bot they are then having to react to this and play around that area. This is element of the system which we believe to be the Gaussian noise term and this is why scripted actions cannot be modelled directly with modelling which does not take into account probability or uncertainty.

Level Data The idea to use the level maps and level model as the constraints for the mixture model is indicative that we expect, at the very least, the enemy play will be constrained to a manifold defined by the level map. Of course this assumption is only as strong as the assumptions made by the level model. If our clustering and polygon fitting aren't good then then constraining data to these has no basis.

Limiting Mean Drift in Practice In order to limit the mean drift it is important to make sure that the mean of each Gaussian does not leave the polygon areas defined in the level modelling section. The following method was used:

1. Find the convex hull of the area clustered using quickhull
2. Insert the new Gaussian mean into the cluster
3. Re-computer the convex hull
4. If the same as the previous convex hull then the point is within cluster
5. Otherwise the point is out-with the cluster area and should be dis-allowed.

The main problem with this method is the inherent inefficiency it introduces due to the complexity of the constant re-computation of convex hulls. This do not effect, directly, the real time performance of the bots due to the afford-mentioned use of a separate thread to run the machine learning calculations. The major effect it has is to change the speed at which the model gets updated. The more work the ML thread has to do the less it can get done at each step it gets on the processor. As such having a more complicated procedure means the model is less likely to contain the most recent data.

7.4.7 One Mixture Per Bot

All of the enemy modelling so far has been based around the assumption that it is possible to model the enemy play with a particular number of Gaussians to represent the points in 3 dimensional space. It was also shown how the system works if each bot in the level has their own single Gaussian model. A particularly nice extension of this is to build a model where each bot is responsible for updating one Gaussian component in a team model. This is based on the assumption the level can be modelled with the number of Gaussians equatable to the number of bots in a team, but given the heuristic approach to setting the number of components in the modelling procedure thus far, it is as likely to succeed as any other method suggested thus far. In the EM update procedure the Gaussians are moved to represent the points for which they offer the most responsibility. Treating the Gaussians as the observations of each individual bot then these can be learned using maximum likelihood estimation as in the single Gaussian case. We are then only required to update the mixing co-efficients to alter the importance of each component in the model. The results presented used mixing co-efficients based on the number of observations in each bot's dataset.

7.4.8 One Mixture Per Opponent

Coupled with the idea of having one mixture component per bot is the idea of having one mixture component per opponent. Of all of the techniques for modelling enemy play, discussed in this section, this technique represents the most obvious in terms of human intuition. Assuming that each enemy bot's play can be modelled with a Gaussian density estimator allows the assigning of one mixture component to each enemy, integrating only observations of that enemy. The mixing co-efficients can then be set by maximum likelihood estimation based on frequency of enemy observations.

7.4.9 A Kernel Model

In Section 7.1.2.2 a parzen density estimator was used as a robust estimator of cluster entropy for modelling level data. This can also be used to perform non-linear non-parametric density estimation over a set of data, with the result resembling a mixture of gaussians, but based on individual samples rather than the entire dataset.

The idea remains the same, to place a Gaussian of fixed covariance on each point, the difference lies in measuring the cumulative distance of a new point from those in the training set using the Gaussian kernel instead of measuring the Gaussian distance within the data, as we did for clustering. The density estimator is defined as follows:

$$P(x_{new}|Model) = (\frac{1}{N}\sum_{i=1}^{N} G(x_{new} - x_i, 2\sigma^2 I)) \tag{7.14}$$

where $G(x,H)$ is a Gaussian Kernel defined as

$$G(x,H) = \frac{1}{\sqrt{|2\pi H|}} e^{-\frac{1}{2}(x)^T H^{-1}(x)} \tag{7.15}$$

In the abstract this method is similar to a summative Nearest Neighbours model but with a Gaussian kernel distance measure. The simplicity of this model is attractive as is the inherent efficiency. The most attractive feature however is the ability to begin generating density estimations with very little data and then incrementally improve these estimations with more data. The modelling procedure is much more stable and predictable. This is supported by the results presented in the situated evaluation.

It also has the advantage of being scalable and easy to understand and interpret. This is the only method which does not have a set number of components and as such requires the smallest amount of input intuition. It mimics the data most closely and gives density functions which best fit the data while pre-supposing as little as possible on the expected density of the data.

Engineering Box

Assumptions We mentioned that the conclusions made with the constrained mixture model were only as strong as the assumptions made in the constraint conditions. With this in mind we wanted to try a model which was more flexible but that also contained the Gaussian noise modelling.

Continuing Assumptions The assumption that the actual play was Gaussian in nature still seemed too strong, given our discussion of why we believed this to be the case. What we were actually saying was that there was a non-linear, non-Gaussian manifold within the map, along which the enemy bots were to be found. This suggested that a parzen density estimator could be useful. The stability and ease of implementation also appealed with this model.

Density Diagrams In this section of the thesis, and those following it, we make large use of density diagrams as a visualisation technique. These are diagrams showing some utility value of each of the level nodes using colours. In general the diagrams show the utility of nodes ranging from 0 to 1 on a scale of blue to red with blue being 0 and 1 being red. Occasionally a thresholding method is used, turning all nodes below a certain utility value yellow, in order to highlight those with higher utility values more clearly.

7.4.10 Basic Evaluation

Some trials were run to test the effect of differing Gaussian models on bot performance. In all the trials in this section, unless otherwise stated, the level used was Trite. This level is relatively large with numerous enclosed sections and rooms. At such it represents a good test bed for enemy modelling.

Engineering Box

Two Perspectives In testing this model we thought it was important to test from two perspectives. That of numerical stability but also from an intuitive aspect. The general approach to these is to use statistics for numerical and visualisation for intuitive. As such our two evaluation sections reflect this.

In order to test the value of the data extrapolated from the model a simple scenario was set up. Two sets of trials were run. In each set of trials the game type was TDM and the game was set up to run for 30 points. Our team was set on *friendly* meaning that they would not fire upon the enemy. For the first trials the bots were set to spend the first 15 deaths exploring the level by picking random points and moving there. This allowed them time in the environment to update the model. For the second 15 deaths they were set to try and hide from the enemy by picking the least likely point of enemy occupation as suggested by the path node utility defined by the enemy model. For both stages of the trial the amount of time between deaths was recorded for each bot. This was achieved using the following LCC strategy:

```
Strategy 2.4

a(hider,ID)::null<--engageModule(hide) and theirTeamScore(S)
and prologConstraint(S > 14)
a(hider,ID)::null<--engageModule(explore)
```

For the second set of trials the set up was almost identical except that the bots spent the second 15 deaths hunting the enemy instead of hiding. This was defined as picking the most likely point for enemy occupation as suggested by the enemy model. It was achieved using the following LCC strategy:

```
 Strategy 2.5

a(hunter,ID)::null<--engageModule(hunt) and theirTeamScore(S) and
prologConstraint(S > 14)
a(hunter,ID)::null<--engageModule(explore)
```

Each scenario was ran 5 times so the results are averaged over 5 trials. In the first case every bot had their own model and updated only it. In the second case every bot updated a single communal model. This was designed to test whether information from the entire team was of more use than each bot only using its own observations.

Because multiple bots were tested the average was taken across the all bots. This is because it is possible that one bot died to make the enemy score 15 and then the others did not die till much later so averaging this out helps to create more stable results.

The table in figure 7.22 describes the types of models tested in evaluation. Each model was tested with two different sizes of dataset to try to show some of the differences this can introduce.

7.4.10.1 Discussion of Results

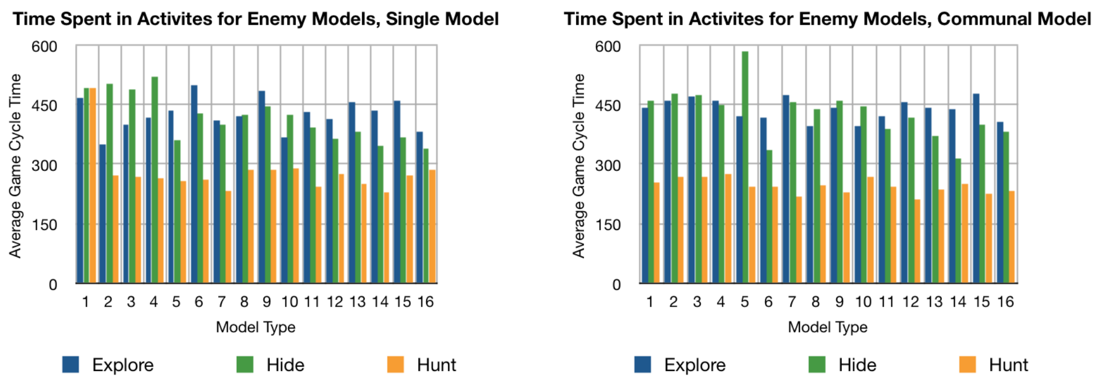
7.4.10.1.1 An Ideal Result The ideal result would be the time spent hunting being substantially less than the time spent exploring and the time spent hiding being substantially more than either of the other two cases. This is because effective hunters will find the enemy quicker and thus die quicker as they are set to not fire upon the enemy. Successful hiders will not find the enemy as quickly and thus die less quickly.

7.4.10.1.2 The Single Gaussian Model There is a definite difference in play when using the single Gaussian model of enemy play for the hiding and hunting purposes. This difference is only present when using the communal model. When using 1000 data-points, a slightly larger difference between the values of hunting hiding and exploring is found when each bot has their own model. The effect of both types of model is closer than when using a 100 point dataset.

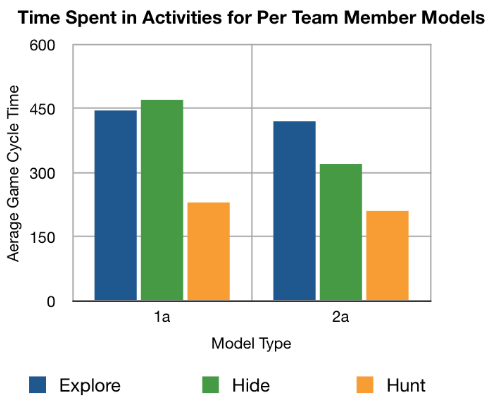
With 1000 or more data-points the single Gaussian model tends to move towards the centre of the level. It is expected that with a larger number of data-points the

	Model Types	
	Model Description	Data Store Size
1	Single Gaussian	100
2	as 1	1000
3	Gaussian Mixture Model 3 Components Standard EM Training Algorithm	100
4	as 3	1000
5	Gaussian Mixture Model 3 Components Gradual EM Training Algorithm Stopping Criteria of 40	100
6	as 5	1000
7	Gaussian Mixture Model 3 Components Gradual EM Training Algorithm Stopping Criteria of 400	100
8	as 7	1000
9	Gaussian Mixture Model No. Components Set by Level Standard EM Training Algorithm Means Drift Limited by Level Early Stopping Criteria of 40	100
10	as 9	1000
11	Gaussian Mixture Model One Component Per Enemy Owner Learn Update Rule	100
12	as 11	1000
13	Parzen Density Estimator Variance 5	100
14	as 13	1000
15	Parzen Density Estimator Variance 1	100
16	as 15	1000
1a	Gaussian Mixture Model One Component Per Team Member Owner Learn Update Rule	100
2a	as 1a	1000

Figure 7.22: Model Types



(a) Basic



(b) Per Agent

Figure 7.23: Evaluation Results Graphs

tendency of the model would be to represent the average enemy presence, particularly with no level data about constrained areas to limit Gaussian movement.

With 100 points the Gaussian models the most recent observations of the enemy. It averages the points but these do not tend to converge towards the centre of the level due to there being less data-points.

The Gaussian density estimator offers some information which is of basic use in modelling enemy play. However, there are better solutions from the other formulations.

7.4.10.1.3 The Basic Gaussian Mixture Model The performance with a communal mixture model is roughly equatable to that of the single Gaussian model. In the non-communal case the mixture model is, however, slightly better with more variation in where the Gaussians are. The convergence towards the centre of the level is avoided showing that this was an artefact of the modelling procedure and not an actual trait in the enemy's play. In the case of the mixture models the results for hunting, relative to exploring, are better using 1000 points than 100 for the single model but slightly worse for the communal model. Typically in machine learning and numerical modelling techniques the number of parameters of the model is linked to the dataset size. Thus it might be expected that with a larger dataset the model with more mixture components is likely to increase performance more than the model with only a single component due to there being more information to estimate the larger number of parameters with.

7.4.10.1.4 The Gradual Update Gaussian Mixture Model In general the gradual update model performs the best of those tested with this evaluation method. In case 5 the model achieves a very high hiding time, however the dangers of reading too much into hiding time are discussed in the overall conclusions. The hunting time for these models is also relatively low showing a stability in the modelling. The communal version of model 7 gives the lowest hunting time.

7.4.10.1.5 The Level Set Mixture Model This model performs better with 100 than 1000 data-points which seems to go against our intuitions based on complexity. The model does not achieve astoundingly better performance than any of the other models (with the exception of the single Gaussian) and as such, due to the complexity and computational overhead, may not represent the best choice of model for use in estimating enemy position.

7.4.10.1.6 The Enemy Component Mixture Model The 100 point model is reasonable here but no better than the gradual update mixture model. Therefore there appears to be no obvious advantage to using it.

7.4.10.1.7 The Team Member Component Mixture Model The team member mixture models offer relatively good performance when using 1000 data-points. The performance is approximately equal to that of model type 7 which so far represents the best choice. This makes this a viable choice as the relative simplicity of the model is quite appealing.

7.4.10.1.8 The Parzen Density Estimator Model The performance of the parzen density estimator models, although reasonable, is not startling. The communal variance 1 model with 100 data-points gives the best performance and isn't too far off model type 7.

7.4.10.1.9 Overall Results Conclusions In general, in the cases of both single and multiple gaussians the figures for hunting are better than those for hiding (signified by the larger difference between the hunting and exploring figures than the hiding and exploring figures). This is representative of the nature of Gaussian modelling as a whole. The gaussians only have useful information about areas which they have data about[10]. In areas outside 3 standard deviations from the Gaussian the data is uniformly un-informative.

Examining the trials from this perspective, and ignoring the information regarding hiding, it becomes clear that communal versions of models 7 and 12 model offer the biggest decrease in time spent hunting to achieve the goal score in reference to distance from the average exploring time. The difference is not particularly drastic and looking beyond the, non-communal, single Gaussian model, the evaluation here doesn't single out a best model for this task as all perform comparatively similarly.

In general the pattern of the communal models is more stable with most of the models giving slightly better play than in the non-communal model cases.

7.4.10.1.10 Further Work One particular problem with mixture models is that there is no guarantee of the exact distance of the data-point from any one Gaussian, only its probability of being generated from the model as a whole. This means that there is even less information about the points which lie away from the model than in the case of the

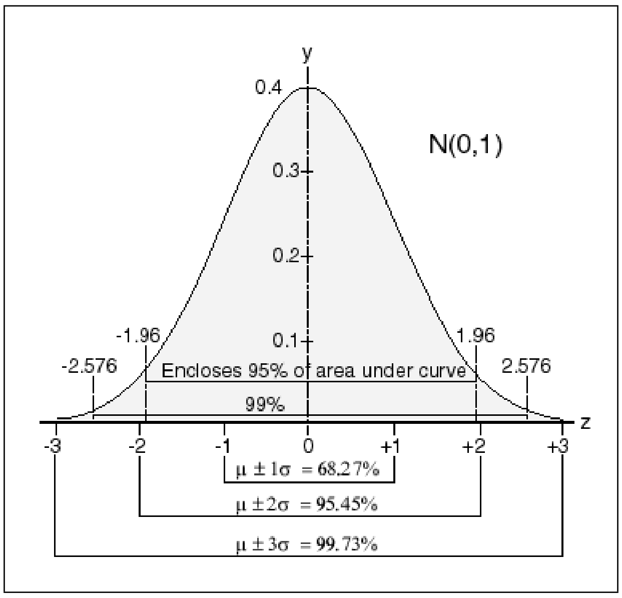


Figure 7.24: Marked Up Gaussian Distribution, image from http://www.stillhq.com

single Gaussian. This points to using multiple models for different purposes[34]. This idea is not explored directly here but left as an interesting avenue for further research.

7.4.10.2 Larger Levels - best model performance

Some tests were set up to show how the best model, from the initial evaluation, performed on some levels which were larger. The levels chosen were Kakori Forest, Osiris, DesertIsle and Junkyard. All of these levels are significantly larger than Trite and fit within the Medium to Large or Large size categories.

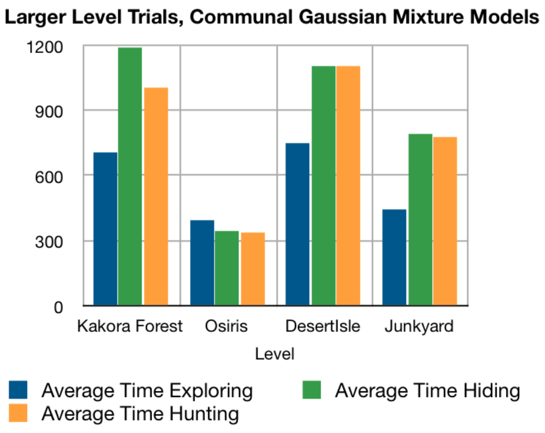


Figure 7.25: 100 points, Communal Mixture Model, Gradual Update Procedure, Early Stopping Criteria of 400, Different Level Tests

Figure 7.25 shows that in all cases the time spent hunting is considerably less than the time spent exploring. The results for hiding are not as clear cut or well defined, sometimes being much more than exploring, others less. Even so it is always a higher value than the hunting figure. The relative ratios between the three activities do not change sharply. This shows that the proportion of the play spent on each activity is stable for larger as well as smaller levels. It also shows the communal Gaussian mixture model is useful for modelling enemy play across multiple different levels.

These tests are by no means exposing the exact model performance, and we are having to add assumption and interpretation to these results to derive any meaning from them. The problem mainly stems from having a large number of un-observable variables in the level which can effect performance. It is true that these simple strategic formulations are not optimal for hiding from and hunting the enemy. When the bot gets to a location it just stands there waiting for the enemy to be in that area. If it were hunting then it should probably try to move around slightly or sample other less likely areas in the hope of finding the enemy. The point of this subsection is to find out if there is any useful information held in these basic machine learning methods and which are the best formulations to use.

7.4.11 Discovering Known Movements

The following section provides some more objective evaluation of the models. A static bot was setup, which would run from one point in the level to another. A team of bots was then be setup to learn this bot's position using the enemy model. This could then be evaluated, objectively via visualisation, because the movements of the bot were known.

Engineering Box

Significance In this section we discuss the notion of how the model reflects play which we know a prior the nature of. This is probably the more important of the two evaluations as it gives a good representation of what the model is actually doing and where the density is concentrating over the course of a match and whether it actually shows what happened in the match. Also our numbers of statistical evaluations weren't substantial enough really to draw any significance conclusions so we are really working on intuition in either case anyway. The idea is to draw conclusions to guide further work and then evaluate the system as a whole in depth later in the process.

Novel Evaluation Part of the engineering process in a system such as this where you are using models in unusual circumstances with unusual constraints in learning and as part of a larger system is that you have to come up with novel ways to evaluate target functions and usability. This often leads to finding ways to modulate parameters while keeping certain factors stable. In normal controlled circumstances this can be very achievable and easily done but when a system is used which is complex in nature it is perhaps not always possible to directly control these parameters directly. Even in the following section where we control a player to move from one location to another and then assess how well the model has represented this we still have some slight issues surrounding the exact routes taken by the bot, fluctuations in their play due to random errors in path finding or movement. This means that even a controlled variable can suffer from random noise and as such it is difficult to be 100% objective in our evaluation strategy. In the context of a system such as ours intuition and observation can be a good guide as to how the rest of the system will fit together and the potential benefits of the technique in a larger context.

7.4.11.1 Test 1 Evaluation

The first test was to have two bots in a level and have them run to two different points. Two points were set as targets and at the beginning of each run a bot selected one of the two points randomly and ran to it. Each bot updated a communal model between the pair and they were set on friendly so they would not shoot at each other. The test shows the power of the model to represent the features of play which are desirable to a bot. Some models from the previous section (all models bar the single gaussians and owner learn versions) were tested to see how well they represented the target concept. This particular method of testing gave a good way to visually assess the chosen sample of the parameter space and set the particular models to be used in later trials.

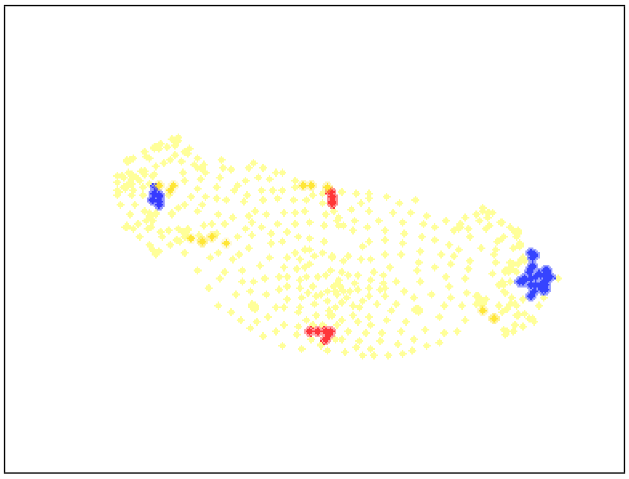


Figure 7.26: The point positions used in the level

Figure 7.26 shows the two starting points and eventual destinations of the bots in the level. The starting points are in blue while the destination points are in red.

A collage of density images displayed in figure 7.27 shows how the Gaussian mixture model performs on the grassyknoll level with 4 components, a data-store size of 100 and standard EM training algorithm with a stopping criteria of 40 iterations. The initial spread is shown in case 1. It then begins to spread to case 2. Cases 3 and 4 show how this then spreads in response to the play across the centre of the level. The problem here is that the model is assigning responsibility to components which take in two areas of the model and is not representing what we want to happen, very well. Case 4 shows more what was required but is not ideal. This model would give a rough estimation of where the enemy are playing but ideally something more predictable in response to play would be better suited to the needs of the system.

Figure 7.28 shows the model's performance when 1000 data-points are used instead of 100. The initial placement in case 1 is good but all other cases show a shrinking of the model's area of influence. This points to a combination of problems common with Gaussian mixture models. The extra data-points mean more runs of the EM model are required to set the parameters of the model. This in turn means there is scope for the Gaussians becoming much thinner and specialised. Thus the early stopping criteria should be higher to combat this.

In figure 7.29 the early stopping criteria is set to 400 to try to combat some of

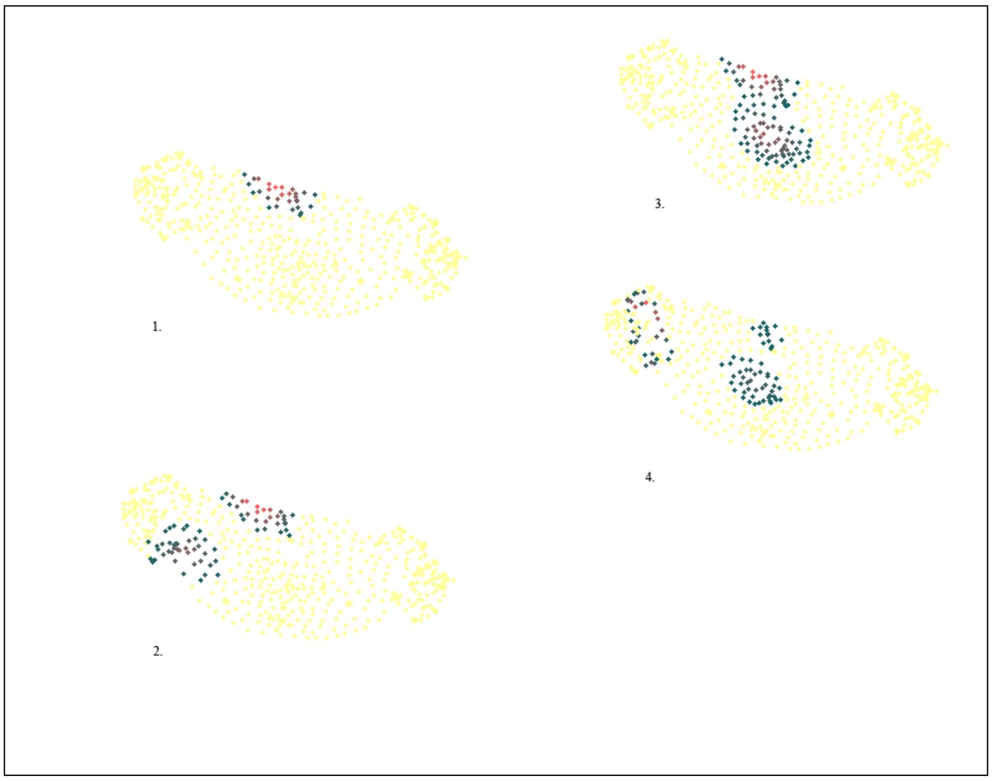


Figure 7.27: Gaussian Mixture Model Density, Standard EM training with stopping criteria of 40 and 100 data-points

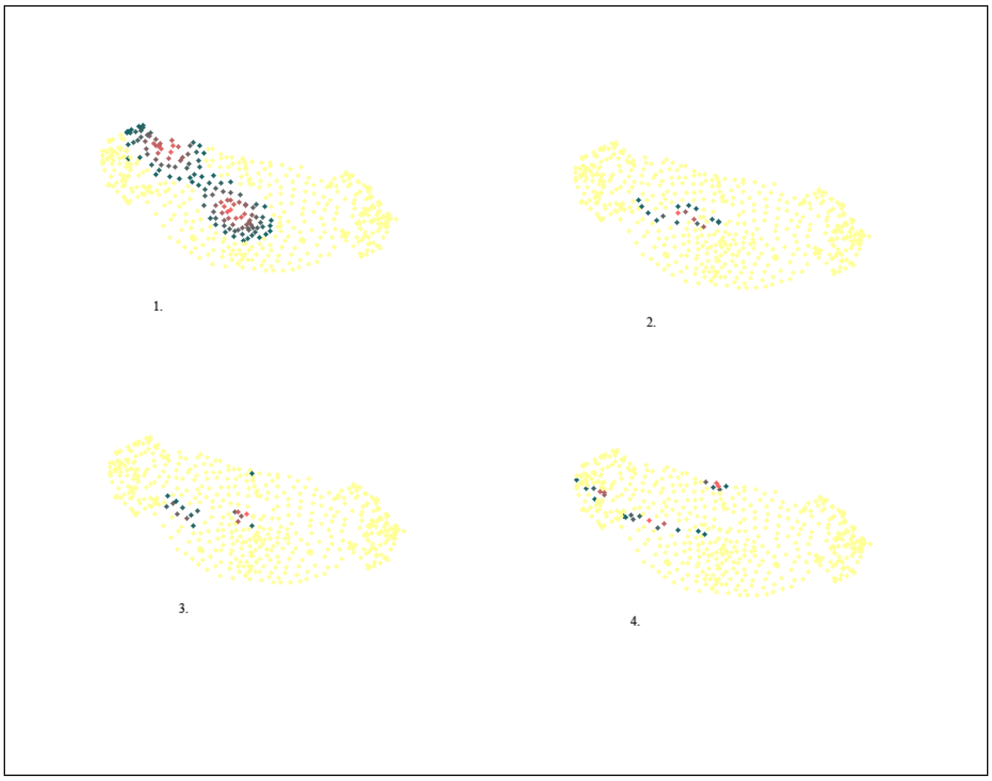


Figure 7.28: Gaussian Mixture Model Density, Standard EM training with stopping criteria of 40 and 1000 data-points

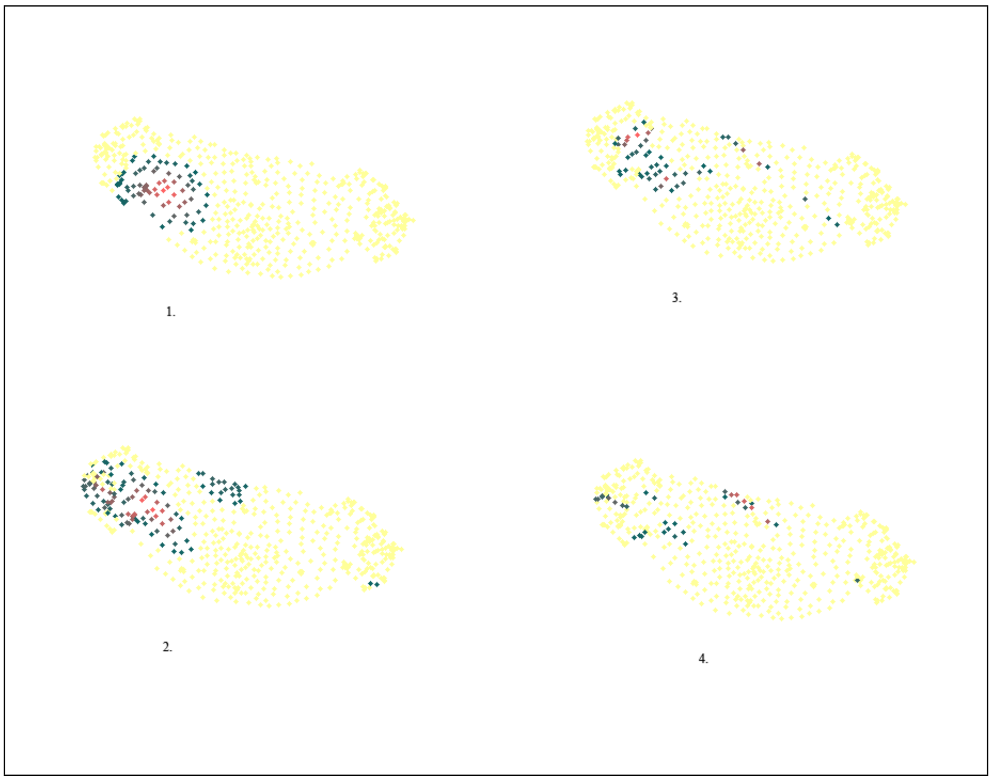


Figure 7.29: Gaussian Mixture Model Density, Standard EM training with stopping criteria of 400 and 1000 data-points

the problems shown at the 40 value, 1000 data-points are again used. The results are slightly better than those for the stopping criteria of 40 but there is still a slimming of the Gaussians to create a narrow density function over the level in cases 3 and 4.

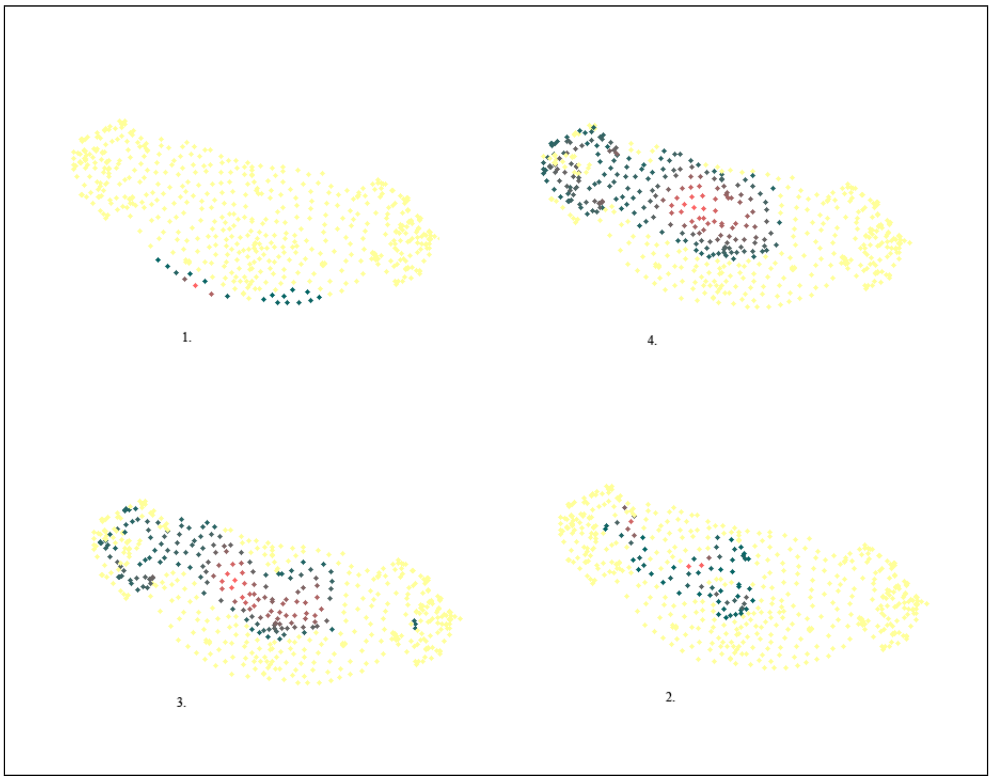


Figure 7.30: Gaussian Mixture Model Density, Gradual EM training with stopping criteria of 400 and 1000 data-points

In figure 7.30 the parameter settings are the same but the gradual update procedure is used. The model is less defined than the standard EM version but again these is no clear definition of the paths taken, which would be be expected if the model was performing as desired. Case 4 shows the eventual settled point of the model at the end of the game only showing a slight movement towards the top of the level which, given the conditions placed on the bots, is not good performance. The gradual update models using stopping criteria of 40 and dataset size 100 showed the same traits as those in the standard EM models.

The next model tested was the version which had the number of components set via the level size and in which the means were limited to the clusters of the level model. This model performed worse than most of the other models in the section. In particular, attention should be drawn to the mixture models' tendency to perform averaging which leads to large areas of density in the middle of the level, a trait normally associated

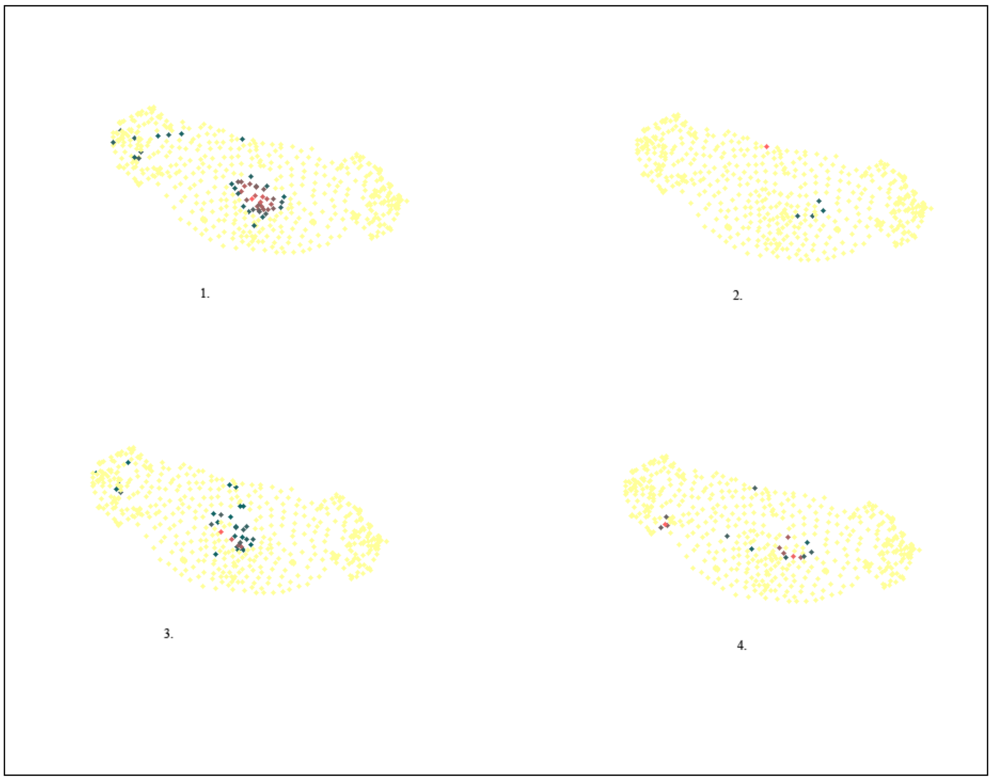


Figure 7.31: Gaussian Mixture Model Density, Standard EM training with component numbers and stopping criteria set by the level, Mean Movement Limited, 100 Datapoints

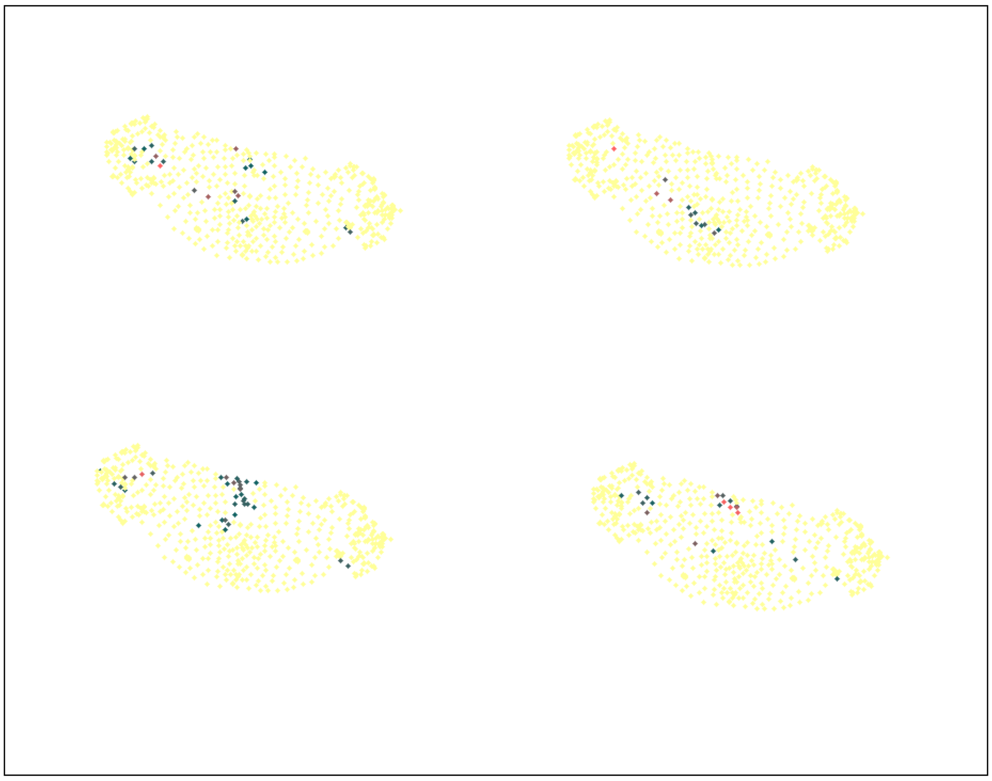


Figure 7.32: Gaussian Mixture Model Density, Standard EM training with component numbers and stopping criteria set by the level, Mean Movement Limited, 1000 Datapoints

with single models. The version with 1000 data-points is slightly less symptomatic, but a different problem is shown which is similar to previous problems regarding over-converged Gaussians. This is more likely to happen in smaller clusters within the level as there are less sampled points within these clusters.

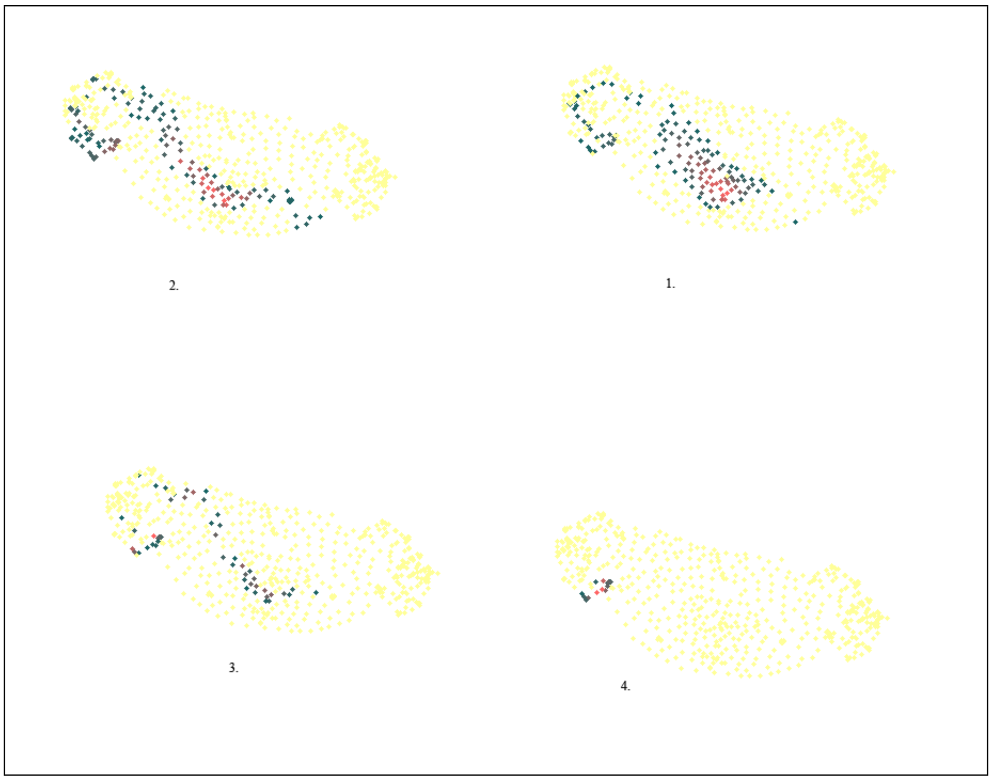


Figure 7.33: Gaussian Mixture Model Density, Gradual EM training with component numbers and stopping criteria set by the level, Mean Movement Limited, 1000 Data-points

The same model was then tested but this time the gradual update procedure was used. The 100 data-point model did not yield any real useful results but the 1000 point model gave a few cases which were, if not useful, at least interesting. The results are shown in figure 7.33. The interesting feature is the path which snakes its way from one side of the level to the other. This path looks very well defined and gives a good estimation of 2 of the possible 4 paths which could be taken by the bots were it not for its gradual disappearance into case 4 throughout the trial.

The parzen density estimator used a Gaussian kernel with a fixed variance of 5. All data-points were given equal weighting. Figure 7.34 shows the 100 data-point model. The results are more predictable, mainly due to the lack of randomness in the training procedure. Case 4 shows the best representation yet of the paths which the bots take.

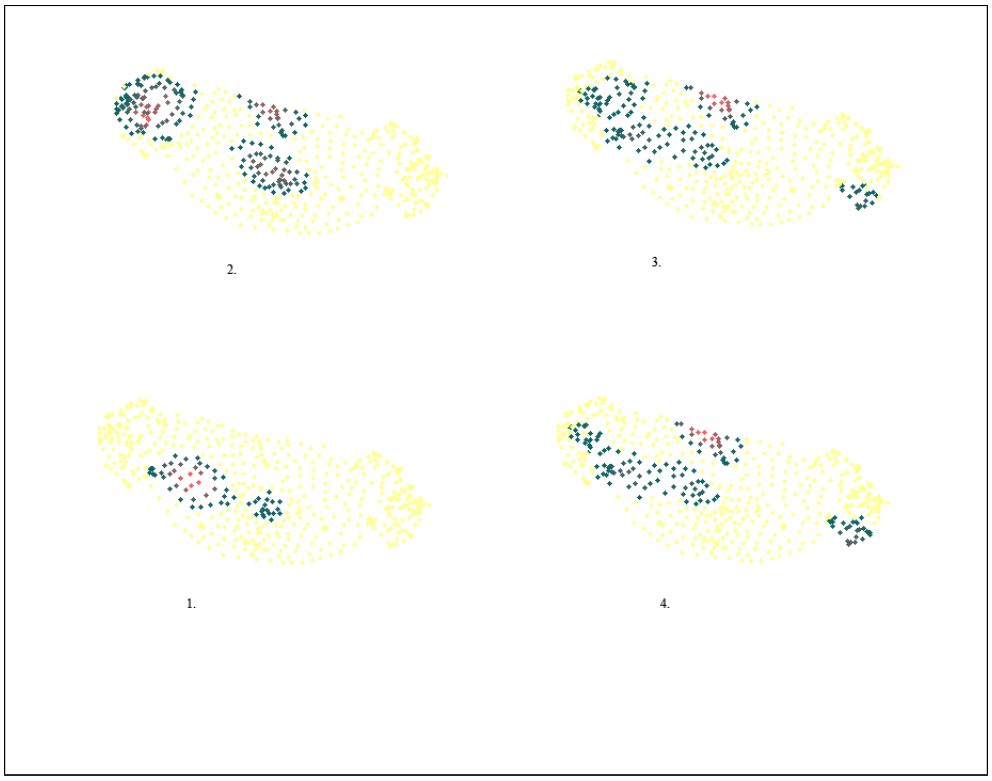


Figure 7.34: Parzen Density Estimator, Gaussian Kernel, 100 Data-points

One of the bots is gaining a slight advantage for seeing the other throughout these trials as the modelling is much more biassed to the left side of the level in nearly all the trials. This model is the first time this feature has been fully visible in the results, to the extent where the density shown can be linked to the bot's behaviour.

Figure 7.35 shows the results for 1000 data-points. The model gives results which, although not a clear indication of the target function are consistent with assumptions about the bots. Case 4 represents all of the target routes with a moderate level of strength.

Figure 7.36 shows the model with a data-store of 2000 data-points with discounting[105]. Discounting is the process whereby points earlier in the data store are given proportionally less influence than those later in the dataset. As the dataset grows in size the proportion of the modelling process afforded to earlier points gets smaller in relativity to the size of the current dataset. So each observation has a multiplier which is defined by the following formula:

$$Multiplier(X) = \frac{Position(X)}{SizeofData - set} \tag{7.16}$$

This result is the best yet. The four main areas of play that the bots were expected

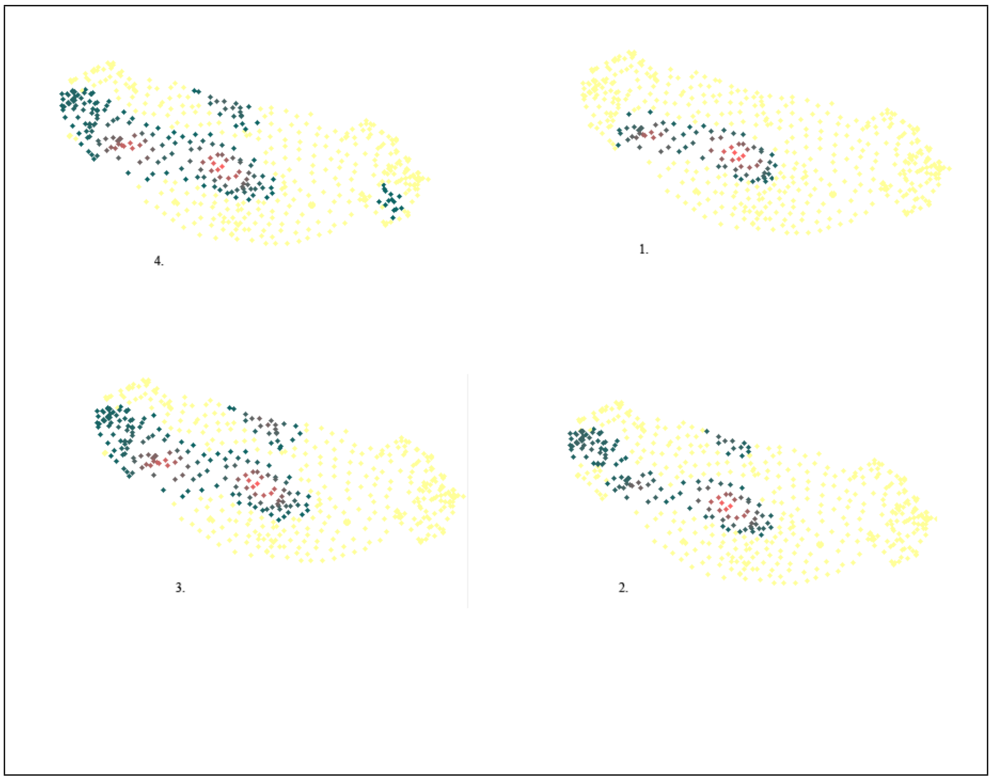


Figure 7.35: Parzen Density Estimator, Gaussian Kernel, 1000 Data-points

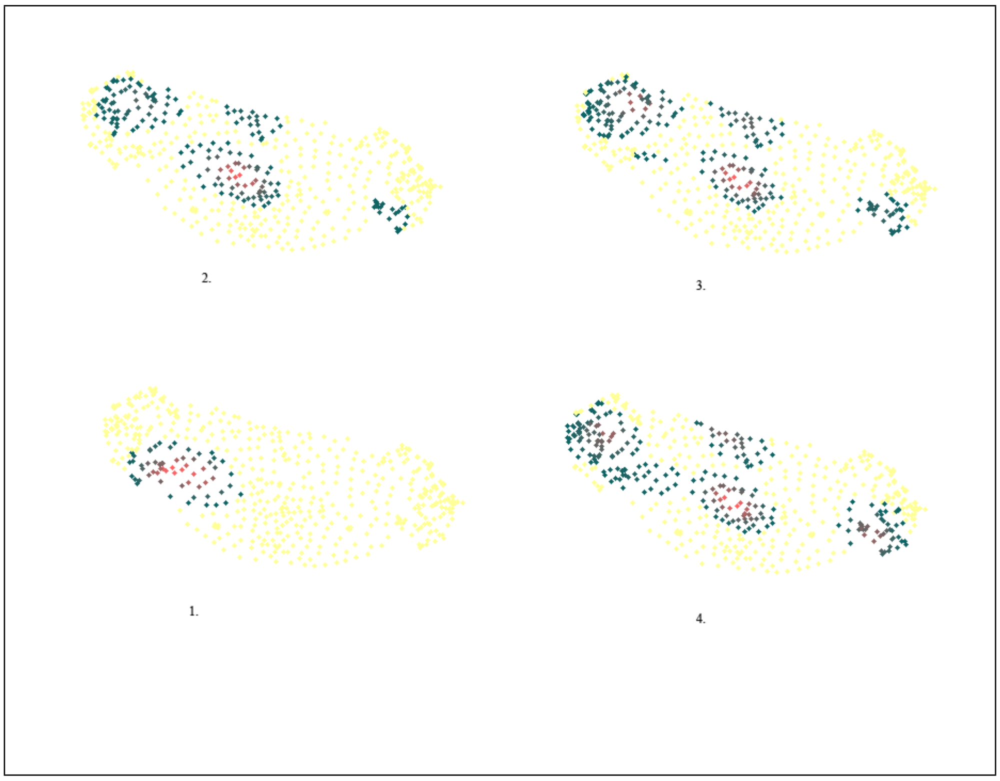


Figure 7.36: Parzen Density Estimator, Gaussian Kernel, 2000 Data-points, Discounting

to be in are shown quite clearly. There is more separation between the areas than is desirable but this is possibly down to the level itself. Considering the raised area in the middle of the level it is easy to see how the paths in between these areas could feasibly not appear in the model.

7.4.11.2 Test 2 Evaluation

The second test had one of the bots playing a game of capture the flag, on the level Chrome, while another bot moved from point A in a level to point B. After a set number of runs the bot changes its pattern to go from point A to a further point C instead. Both bots were set on non-friendly so they would shoot at any visible enemy. Only the bot playing capture the flag updated the model with observations.

This was designed to test how well the model coped with changes in an enemy's play. There is no concern about coping with a non-static or fully random bot, but it is important to show robustness in the model so that if an enemy does change its play during the game then the system will have a way of counter-acting this.

Because of the relative success of the parzen density estimator, in comparison to the mixture model counterparts in the previous section, only the parzen estimator results are presented.

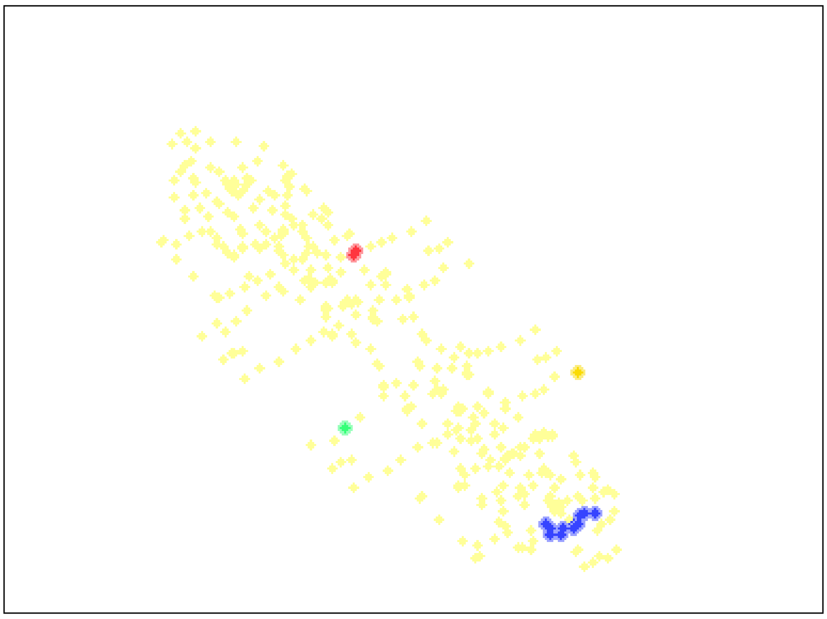


Figure 7.37: The point positions used in the level

Figure 7.37 shows the points used. Point A is blue, B is red and C is green.

The parzen density estimator used a Gaussian kernel and had a variance of 5. All

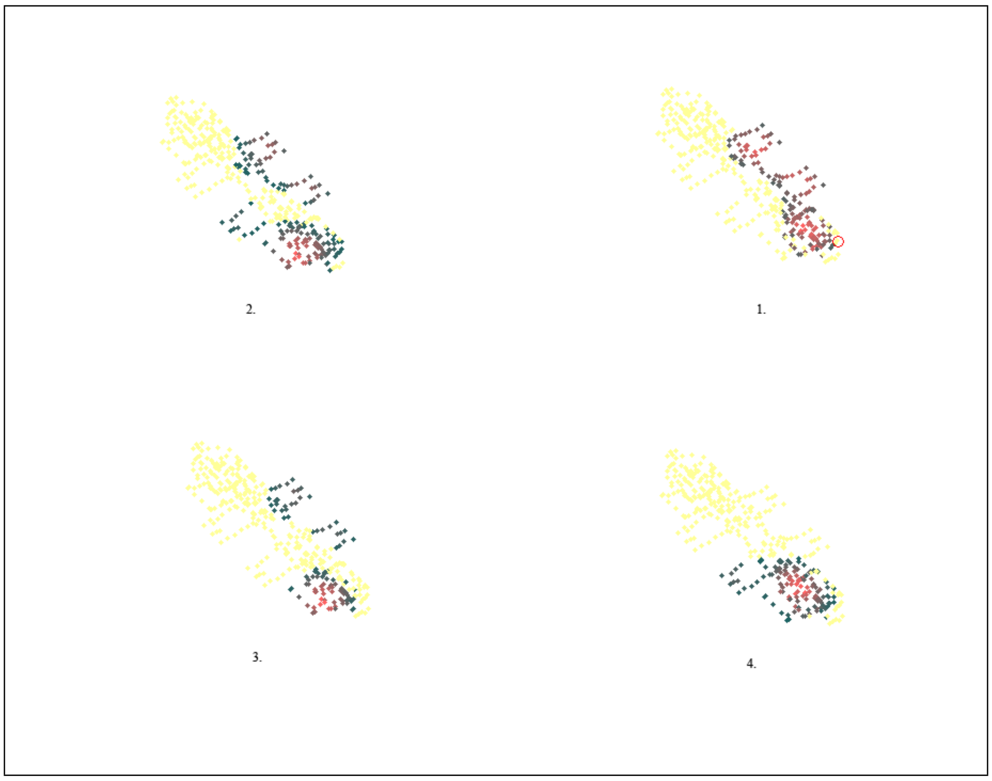


Figure 7.38: Parzen Density Estimator, Gaussian Kernel, 100 Data-points

data-points were given equal weighting. Figure 7.38 shows the 100 data-point version. The model represents the path taken by the bot. It also tracks the changeover from strategy 1 to strategy 2. This is because the dataset begins rewriting older points with new ones as the match progresses. It is interesting that with only the observations made from the bot playing capture the flag the model can still obtain a reasonable estimation of the play of the opponent.

Figure 7.39 shows the results for 1000 data-points. This model doesn't do as well with either the representation of the path or the change of strategy. The model averages the two into a density mass which is a representation of neither one or the other. This is not ideal as, although it does give a logical averaging effect, it would be better to have something more akin to picking one or the other.

Another problem is that there is a common section which occurs in both cases towards the start of the path. This path then gets reinforcement from both strategies, due to the large representation within the dataset, and thus takes the largest part of the density estimator.

Figure 7.40 shows the model with a data-store of 2000 data-points and discounting. The representative power is equally good but the drift is never fully represented, even

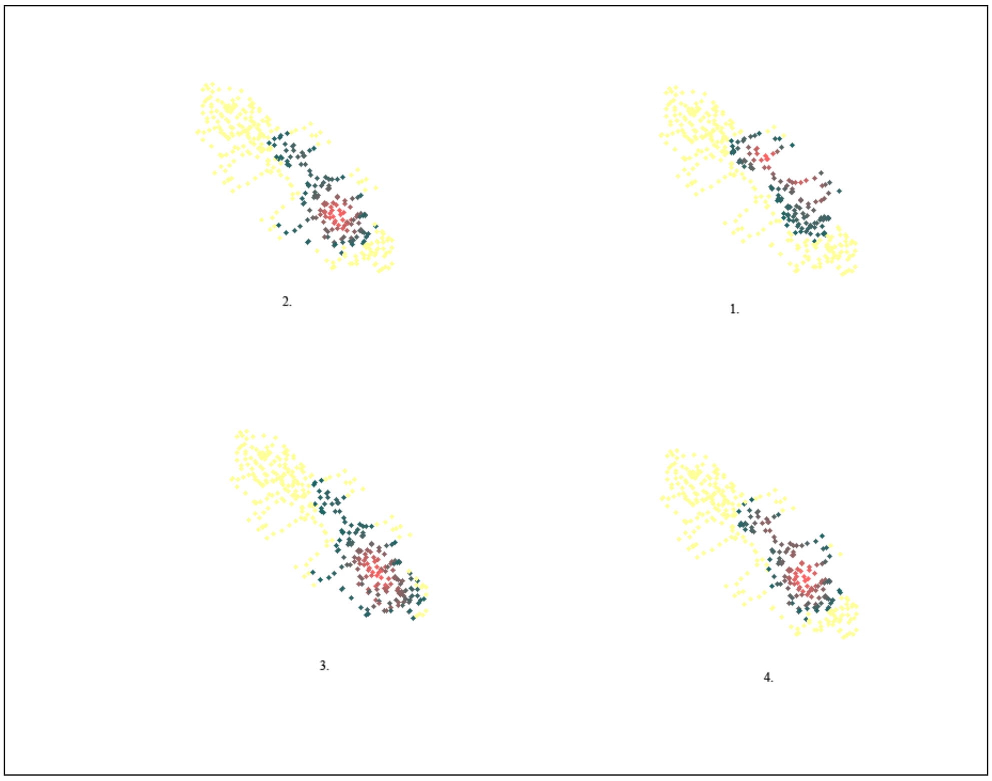


Figure 7.39: Parzen Density Estimator, Gaussian Kernel, 1000 Data-points

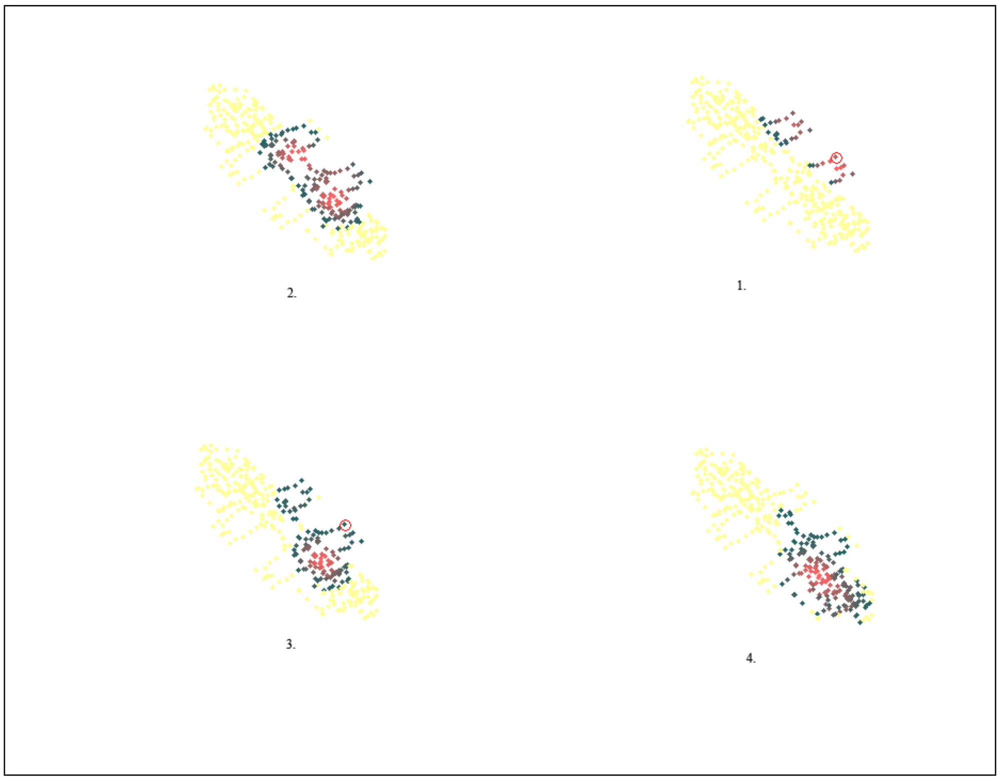


Figure 7.40: Parzen Density Estimator, Gaussian Kernel, 2000 Data-points, Discounting

at the end of the match. That said, this model does have the clearest path representation for strategy 1.

7.4.11.3 Test 3 Evaluation

The third test used was the most robust. The enemy is using the same strategy as that in test 2. Our learning bot was playing differently though. Instead of trying to capture the flag, they followed the enemy bot. If the enemy was seen, the bot would move towards it, other wise they would consult the model for the most likely point of enemy occupancy and go there. Only the `following` bot updated the model and both bots were set on friendly so no shooting occurred.

The point is to show how tracking can be performed using the model as a backup for direct observations of the enemy. Because the paths are known the output of the model can be traced to see if the bot's internal concept of the enemy play matches their known path.

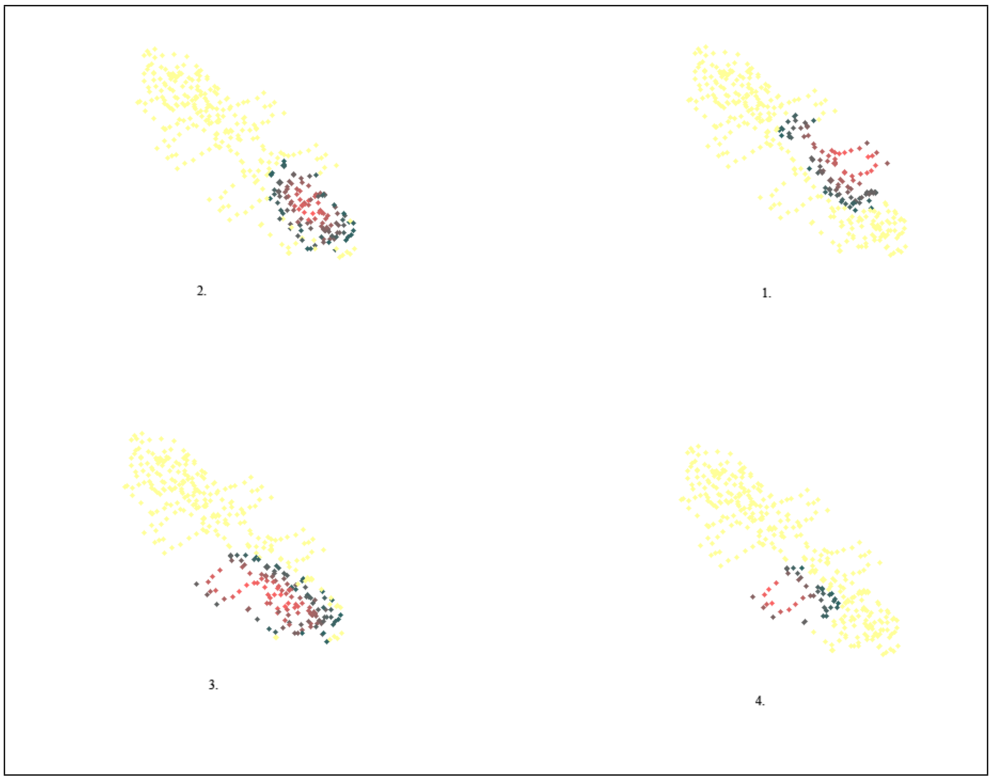


Figure 7.41: Parzen Density Estimator, Gaussian Kernel, 100 Data-points

Figure 7.41 shows the 100 data-point model. This model performs better than in test 2. The paths are clearly represented and the response to change is satisfactory. The only problem is that the first path never reaches full length (by full length we mean the full path taken by the enemy bot). This is mainly down to the sheer number of observations made during a single following run creating a situation where the centre of the path overpowers the end points. It was apparent during the match that the model was doing a good job of directing the bot towards the general area of the opponent's play and aiding in tracking.

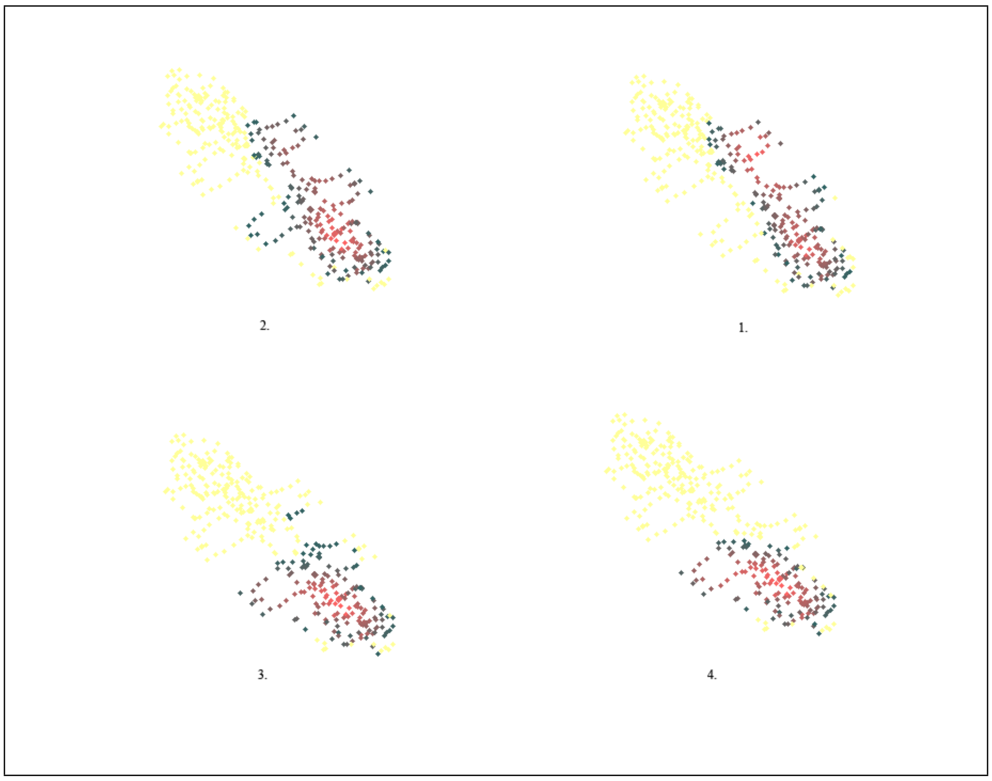


Figure 7.42: Parzen Density Estimator, Gaussian Kernel, 1000 Data-points

Figure 7.42 shows the results for 1000 data-points. This model represents the full path better and also tracks the change. The change here is more of a smooth transition rather than a sudden change showing conservatism in response to temporal changes.

Figure 7.43 shows the model with a data-store of 2000 data-points with discounting. This is the best of the three models tested. It shows the paths in clarity and the transition is smooth but slightly more focused than the 1000 data-point case. There is response to change but the older data-points still have some influence on the density estimation. This represents a good compromise between the two.

7.4.12 Overall Conclusions

From all of the tests performed in the enemy modelling section some observations can be made.

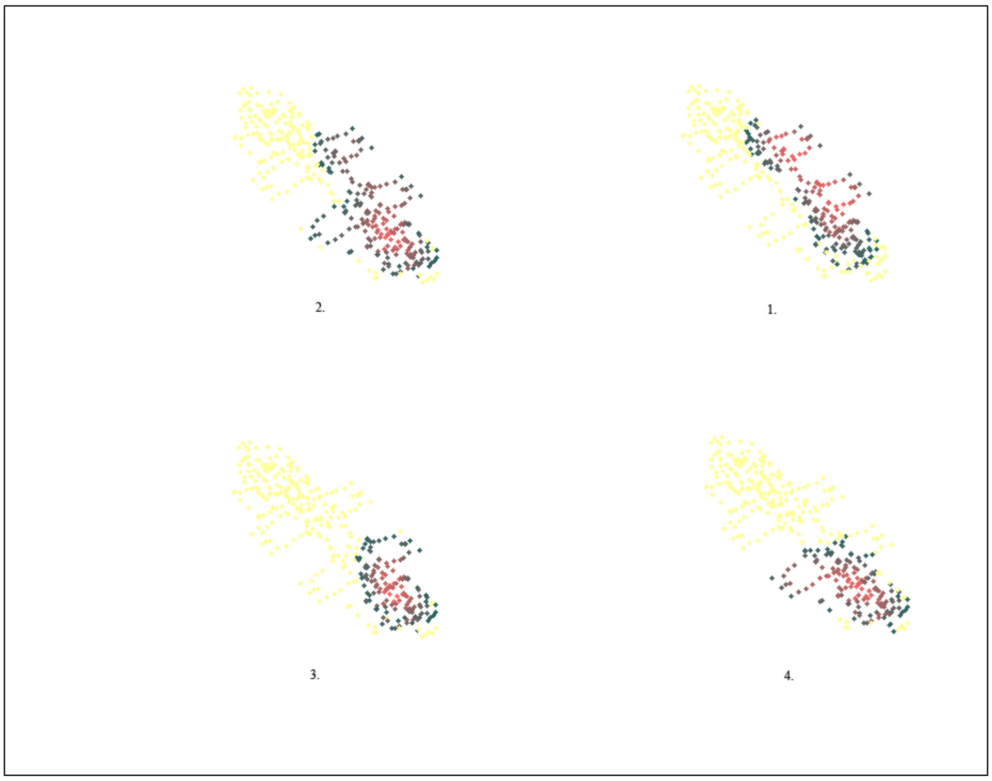


Figure 7.43: Parzen Density Estimator, Gaussian Kernel, 2000 Data-points, Discounting

On the subject of mixture models there is a definite distinction in performance between the standard EM trained versions and those gradually updated. The main problem introduced by the standard trained versions is the inherent unpredictability created by the random elements of the standard EM algorithm. As such the gradual update procedures shown are more predictable, although not great in terms of situated performance.

Overall the parzen density estimation angle presents the best option for modelling the enemy position, based on the known movements trials. The predictability of the model is a desirable quality and its stability and consistency also gives it an advantage over standard mixture models. This model also showed a robustness to basic changes in enemy behaviour which is, again, desirable. Although no results were presented showing how the model dealt with more complicated or subtle changes in enemy behaviour we do not invisage this being a drastic problem for similar video game adversaries. This then raises the question of what other types of adversaries we may face. The most likely in the video games world are humans. But do we expect humans to play with more predictable patterns in their play than the bots used here as a benchmark, or less?

Research in [87] suggests strongly that humans actually have stronger patterns in their play than we might expect. These are normally associated with levels of skill and frequency with game playing and exposure to the particular game. This suggests that drifts in human performance patterns would not be within a single match but over a course of matches. Because our system performs only online learning this means that we would not have a problem with these changing behaviour. Each match we would start modelling again from fresh and model the new human pattern.

With respect to the learning algorithms used, an alternative training algorithm could have been found that had more stability, by eliminating some of the random factors, but the performance of the models does not suggest that this activity would be particularly advantageous.

The chosen model was a parzen density estimator with a data-store of 2000 points and temporal discounting.

7.4.13 Gaussians as a Metaphor for Message Passing - An Alternative Situated Evaluation

The idea of using Gaussians to co-ordinate a team of bots has been touched upon[40] and also the idea that perhaps hiding and hunting are not the best evaluations for understanding the Gaussian models was discussed earlier in this section. Thus, a third test was set up to evaluate the chosen model. The bots were put into a team of 3. When they could see the enemy they would run towards them shooting at them. In the absence of visual enemy contact they would use the enemy model to pick the most likely point of enemy occupation and run to that location. This was achieved with the following LCC strategy:

```
 Strategy 2.6

a(hunter,ID)::null<--visiblePlayer(Location) and
strafeAttempt(Location,Location)
a(hunter,ID)::null<--engageModule(hunt)
```

There is no explicit message passing in this strategy. All the inter-bot communication is performed in the updating up the communal enemy model.

Tests were performed on Sulphur, Idoma and Osiris2. These varied in both size and shape.

A baseline strategy was setup to test any increase in performance due to model use. The baseline would do the same as the above in the presence of enemy bots but would move to random points in the level in the absence of such stimulus.

This was achieved with the following LCC strategy:

```
 Strategy 2.7

a(hunter,ID)::null<--visiblePlayer(Location) and
strafeAttempt(Location,Location)
a(hunter,ID)::null<--engageModule(explore)
```

In addition to this the bots were only allowed to use the assault rifle with which they spawned. This was to try to limit the effects of multiple weapons on the trials in the interests of objective evaluation.

7.4.13.1 Results and Observations

In figure 7.44 the trials using the enemy model are labelled A and those without the model are labelled B. This figure shows that, although not completely even across the 3 levels, and not winning every match, the play is a lot more stable in differing scenarios and level types when using the model. This shows that even with this simple model formulation some level of cross scenario independence can be exhibited. In [40] the best results were almost always achieved on smaller levels. Here the best result is found on the sulphur level which is not one of the smaller levels in the game. This points to not always using the highest level of complexity in modelling if the game levels or scenarios do not warrant this. This is in reference to the fact that a better result can be obtained on smaller levels using a simpler strategy which did not use a model to perform the hunting. This is also based on the assumption that on smaller levels there are no places where the enemy can hide. This information could be obtained from the level modelling procedure discussed in section 7.1.2.3.

Watching the trials allows observations about the enemy model and it use to be made. Firstly, using the enemy model straight is not the best strategy for correctly judging enemy position, due to the inherent problems associated with reinforcement of conclusions. When the bots get to the position of most likely enemy occupancy they will stay there until the enemy, arrive thus further reinforcing the conclusion that this is the most likely enemy occupancy point . This said there is definitely an improvement using the model and this is mainly down to the fact that with multiple bots in a close vicinity they pack more fire power as a unit and as such have more chance of fending off attackers. They steer the bots away from areas of the level which are less likely to have enemy occupancy so in some sense the model does achieve a basic level of performance. Using the model in a strategy, care must be taken to ensure that the correct trade off between exploration and exploitation is achieved.

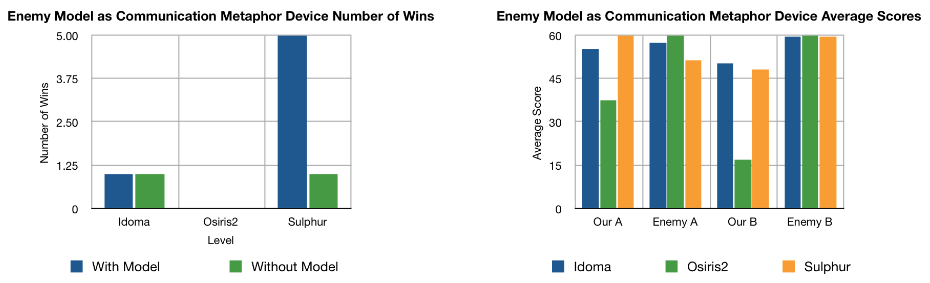


Figure 7.44: Situated Enemy Model Results

7.5 Weapon Utility Estimation

In UT, as in many first person shooter (FPS) type games, a large factor effecting the play within levels is weapon selection. This holds not just for UT and FPS games but also for any situated real-time adversarial environment containing methods, of different levels of effectiveness, for engaging adversaries. In real life military situations weapons are evaluated prior to engagement for effectiveness but one can easily imagine that the utility of weapons in combat would need to be tweaked should the enemy display unexpected characteristics.

Engineering Box

Assumptions A better title for this section would have been "we need to do something about weapons as they play such a big part of the game"' and that pretty much typifies the motivation here. A lot of the machine learning mechanisms used are domain driven. We can retrospectively motivate these by scientific argumentation but the driving force is simply working with the domain and examining what jumps out at us as a good idea for prototyping. Modelling weapons seemed like a good way of boosting performance. It also seemed almost trivially likely that a weapon's effectiveness would change from one level to another and this fits perfectly with our assumptions about useful learn-able environmental factors.
Another factor effecting the decision to model weapons was that it is often the first thing naive players do. With little or no skill one tends to gravitate towards the more powerful weapons to compensate for this, but with a new game this takes time to determine which weapons are more powerful/useful to compensate for a player's particular deficiencies. This idea seemed to fit perfectly with our notion of having certain techniques compensate for others where there were mis-matches in performance.

Interface Limits From the beginning of the project one of the most obvious facts was that weapons were going to be important in the domain. The question then became how do we go about modelling these. The main options were either to model the enemy's choices or how effective our choices were against the enemy. In the end the domain interface made the decision for us as there was no access to the enemy's current weapon. Again this is another area where the actual system being engineered effects our approach to modelling.

Problems Ideally the bots should be able to learn their own weapon preferences based on their own experience. The main problem with doing this is abstracting the information from the relatively sparse data throughout the course of the matches. Regardless of the technique chosen care must be taken not to allow one particular weapon to run away with all of the probability via self-reinforcement. For instance if the bot uses a flak cannon to kill an enemy in a very quick fight then this weapon has a higher probability of remaining selected. As such more data will be gathered for it. Using simple probabilistic selection based entirely on frequency values gained from automatic naive estimation of probability values, a reinforcement loop[105] could easily be created. With this in mind, fights with negative results should result in negative weighting within the selection mechanism. Therefore any parameter space must be rescaled to O(1) but also to be in the range -1..1.

7.5.1 Previous Weapon Modelling Work

Although there is no work concerning weapon modelling for exactly our set of circumstance or situation there do exist papers dealing with the idea of weapon selection for FPS games in general.

Galli et al [37] detail the creation of an automatic weapon selection policy which learns to imitate certain styles of play from a bank of data. The learning is offline and the primary purpose of the system is to aid a human player in a way similar to assisted aiming but the paper presents an analysis of a range of weka techniques as applied to the pre-processed dataset which is useful as a guide to aid our choice of technique.

Burkey et al [22] describe the use of a both a neural net and a fuzzy logic system for online weapon selection in Quake 3. They then place this inside a larger hybrid architecture in a way which is quite similar to ours ideologically if not in final execution. The most interesting thing about the work in this paper is that it suggests possible factors which could be used as input features for a specific choice of machine learning technique, the most pertinent of which are “distance to enemy” and “ammo”.

In the next subsection we discuss a well defined mathematical model and show why for our domain it is not ideal. In doing so we demonstrate some of the problems with online learning for weapon modelling.

7.5.2 Belief Network Weapon Modelling

One possible model for weapon selection is the Bayesian Belief Network(BBN)[12]. This type of model uses probabilistic dependencies based on observations which can be used to perform inference in a constructive and probabilistically sound way. With a fixed structure the conditional probabilities must be learned or estimated. The standard method used is maximum likelihood estimation but unfortunately for this situation it is not ideal. Simple counting based estimation requires discrete continuous variables. This leads to several problems. Firstly it creates numerous problems with the conditional independence assumptions in the model and secondly it offers no way to allow for a gradient of probabilistic influence. For instance discretising a variable such as health into 10 categories creates a need for data in each of these categories to get meaningful data for any given weapon option. This is the problem of having sparse data but needing to estimate a model with more parameters than the data supports.

A better option is to leave the variables as continuous and then try to estimate their density parameters. This is along the same lines as the Gaussians used in the enemy model play except that said Gaussians are combined in a conditionally probabilistic manner by expressing children as linear transformations of their parents. Informally any estimation function can be used as long as this is then normalised against the other possible values [6]. Therefore different estimators can be used for different types of conditional independence calculations. Due to still having an overall probabilistic treatment of the inference in the BBN the sensible option is a frequentest approach to the parameter estimation.

An initial BBN was setup to reflect intuitions about how the various parameters interact with each other.

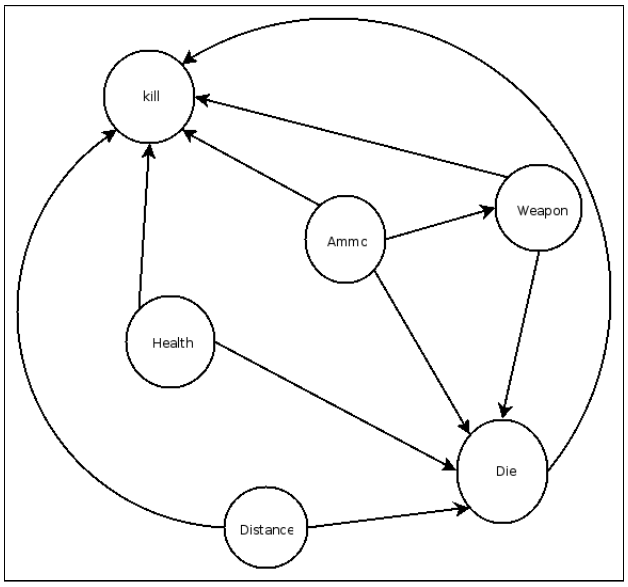


Figure 7.45: Initial Bayesian Network

From initial informal testing it became obvious that this idea was not going to work. There were several problems but mainly there were two critical points of failure within the model. The first was the sparseness of the data. It is mentioned throughout this thesis that this is a problem, due to ignoring knowledge transfer, but it proved to be fatal for this particular model. Related to this problem was the more critical issue of the nature of data. The system could only sample positive cases i.e. something becomes more probable if it happens a lot. There was no way of negatively reinforcing a conclusion so that it became much less likely. What was needed was something utility based which tallied performance based on intuitions about weapon performance.

[6]with continuous variables the area underneath the density function integrates to 1

A further issue with this formulation was that the bots tended not to have very many weapons in their possession at any time. This meant that the information gained was not very useful or informative and rarely could the optimum weapon be selected.

The experimentation suggested that one option could be to estimate which weapon was better overall and then use this, in the behaviour modules, to decide which weapons to go for before engaging the enemy. It also suggested that there may not be enough data collected during the match to estimate the parameters of something as complex as a BBN (Which typically requires a larger amount of data to estimate parameters). The system needed to get more from the sparse data with more initial human input.

7.5.3 The Heuristic Function Model

Our second model was a heuristic utility estimation function[86]. This approach is not as well grounded as the BBN model but is more robust for this system [7].

A heuristic model allows both collection and refinement in light of new data and also integration of naive prior bias. The system used the following heuristic:

$$\text{initialise: previous utility = 0} \tag{7.17}$$

$$f(weapon_x) = \begin{cases} \text{previous utility - 200} & \text{if } death = 1 \\ \text{previous utility + 100 - health difference} & \text{if } kill = 1 \\ \text{previous utility} & \text{otherwise} \end{cases} \tag{7.18}$$

At each time-step this value was re-calculated based on taking utility over all different observations of fights in the dataset and the weapons are ranked in order based on this value. The threshold points were also set so that when a weapon reaches a certain level of utility the bots decide it is worth going for. For experimentation with this function the bots were set to select the last weapon which they picked up. This allowed for sampling of all the weapons without putting a bias on weapon selection.

In addition to the above model a discounting scheme was also tested. This rated newer observations higher than older observations. This was performed by multiplying the utility update by a multiplier which reflected the observation's temporal positioning in the trial in a similar way to that used in the discounted parzen density estimator in the level modelling section.

[7]Even with a non-frequentest Bayesian treatment of the parameter estimation of the Gaussians there still wasn't anything that suggested the BBN was going to work in any particularly useful way

Two other models were set up alongside the weapon model described above. One only considered observations where the distance from the enemy was over 1000 positional units away and the other where the distance was under this value. This gave models which only dealt with close range and long range weapon utility estimation. Discounted results are presented for each of these models.

The initial evaluation procedure was to run 5 trials and take the state of the model at the end of each trial. These were then averaged to give a set of values. The standard deviation of each set of 5 trials is also reported to show how the model was performing with regards to stability.

The results graphs in this section display the average utilities of each weapon at the end of the match. All weapons began the matches at a utility of 0.

7.5.4 Weapons for Team Death Match

In this section we deal with the results for the TDM game-type.

7.5.4.1 Idoma

The Idoma level results show that the assault rifle performs best in all categories for this level. One of the main reasons that this result cannot be trusted is that the bots are spawned automatically with the assault rifle and no other weapons. This leads to a larger representative sample for this weapon than others because of the experimental process. As such the result is not so much a representation of how well the assault rifle does against other weapons on the level but is a reflection of how well the bots do overall.

Looking beyond the assault rifle, the minigun performs best in the *overall* and *near* categories. The only category in which it is beaten is *far*. At longer range the shock rifle is the preferred choice from the non-assault rifle weapons. At close range the minigun is best with the bio rifle or rocket launcher, as secondary options.

The standard deviation results show that of the weapons which performed best the assault rifle has the highest standard deviation followed by the minigun and the shock rifle. It seems to be a pattern that the more extreme the performance of a weapon the larger its standard deviation. This shows that because of the small amount of samples used in the training set, the modelling is working on a dataset which is not stable. This said there are still patterns, even considering the small amount of data used.

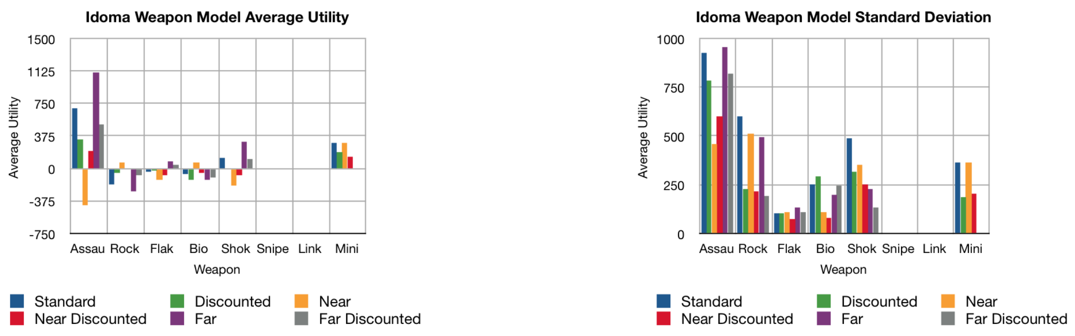


Figure 7.46: Weapon Results Graphs for Idoma

7.5.4.2 Desert Isle

The Desert Isle level results show that the shock rifle is performing much better than the other weapons in the basic and far categories. At close range the rocket launcher is deemed most useful. Again the standard deviation graph shows that the weapons with the most extreme values have higher rates of standard deviation. In the assault rifle's case the utility deviation runs above 1000 showing a huge swing in the typical utility values.

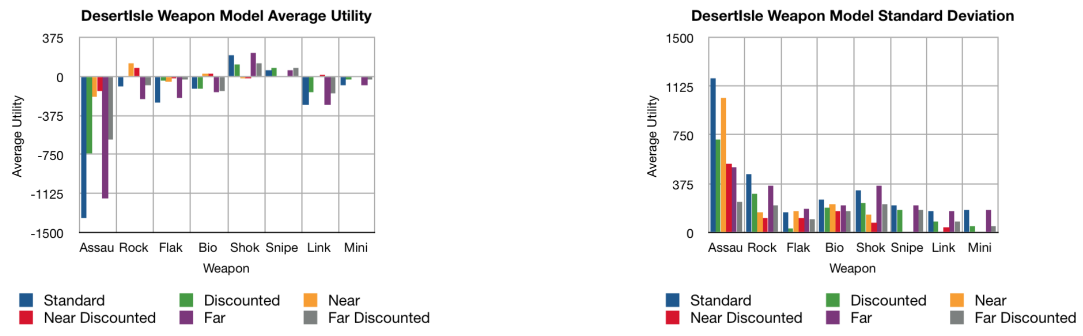


Figure 7.47: Weapon Results Graph for Desert Isle

7.5.5 Weapons for Capture The Flag

In this section we deal with the results for the CTF game-type.

7.5.5.1 Orbital2

The shock rifle, again, dominates the weapon utilities with an accompanying high standard deviation value.

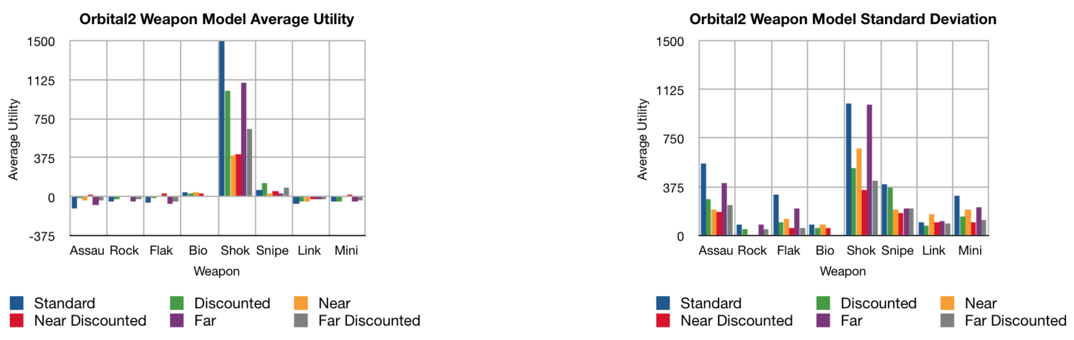


Figure 7.48: Weapon Results Graph for Orbital2

7.5.5.2 Twin Tombs

All the weapons perform markedly badly on this level. The minigun, shock rifle and rocket launcher achieve small positive utility values in some categories but overall no weapon is the best choice in this situation. This is a reflection of the fact that the bots don't perform well. The model does still give a slight gradient which, with proper sampling, could be useful for exploring weapon preferences.

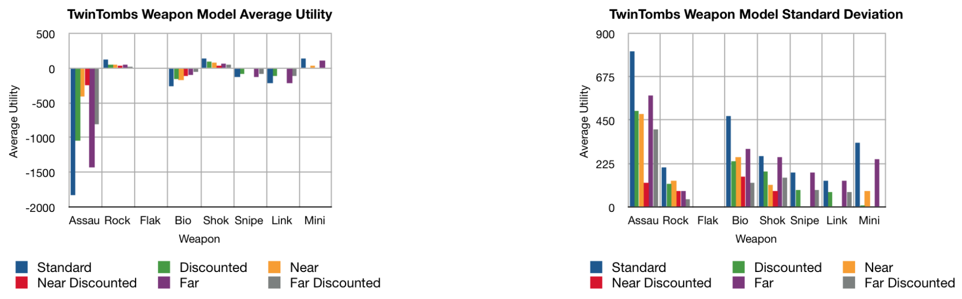


Figure 7.49: Weapon Results Graph for Twin Tombs

7.5.6 Weapons for Double Domination

In this section we deal with the results for the DD game-type.

7.5.6.1 Sun Temple

Figure 7.50 shows a bias towards long range weapons. This is because leading up to each domination point there is a long tunnel, which the attacking bot must run along. As such long range weapons are likely to lead to a lot of kills in this area for either the defending or attacking bot.

Engineering Box

Caution When sampling play for weapon modelling it must be ensured, firstly, that models are not used in a greedy fashion, i.e. instance always getting the best weapon and overly reinforcing it. On top of this it must be ensured that the sampling is representative of the play within the level. Thus the bot who is making the modelling observations must engage in activities representative of the play which is likely to occur by a bot using the model to better its play.

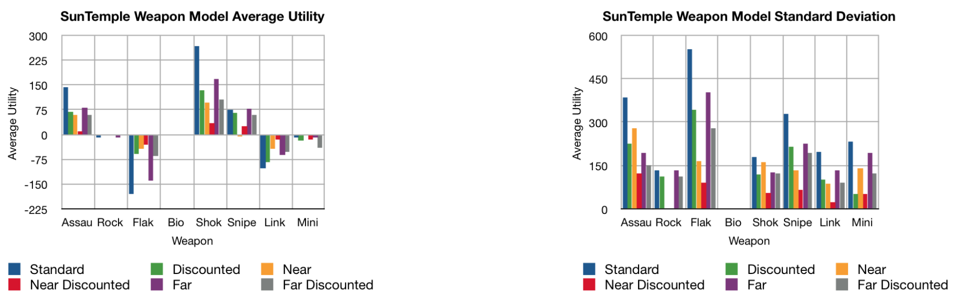


Figure 7.50: Weapon Results Graph for Sun Temple

7.5.6.2 Conduit

The graph shows that the minigun has the highest utility in all categories. The minigun and the bio-rifle are the only weapons with a positive utility for any categories. The standard deviation values do not reflect this fact but its worth noting that the difference in utilities between these and any of the other weapons modelled is minimal.

7.5.7 Weapon Modelling Evaluation Conclusions

There are trends on each level which can be used to model factors which vary between levels. There is discrimination between each weapon such that a hierarchy can be

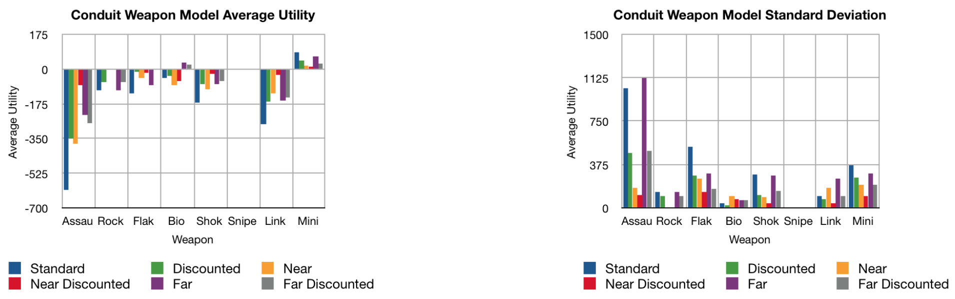


Figure 7.51: Weapon Results Graph for Conduit

established for individual game instances. On many levels the best weapon comes out as the shock rifle because it is as accurate at short distance as long distance, therefore the bots can kill the enemy when they enter line of sight regardless of distance (within reason). The sniper rifle also has this characteristic but takes a longer time to reload and as such is not as effective. On some levels the shock rifle is no better than others, with weapons such as the flak cannon and minigun performing much better, particularly at close range. This can be down to factors such as the position of the weapon pickups in the level and also the chosen paths of the bots and level design. In general the environment can override objective differences in acquired resource effectiveness.

These models contain a feedback loop. The bot's choice of active weapon can severely effect modelling. If the bot always sticks with the default assault rifle then no other weapon will get reinforced. This is why it is important to perform two tasks when using this model:

- This model must not be used in a greedy manner as this could lead to adherence to bad *local minima* within the learned selection policy. This point is dealt with much more deeply at the strategy level when the exploitation versus exploration trade-off is considered.

- The model must be able to go into the negative range as well as the positive. This is to prevent the scenario where one weapon is always selected, even if it performs badly.

Individual results can be explained using intuitions about the weapon and level. These intuitions could be used to create static weapon selection policies. The benefit of the model approach is that it allows the bot to smooth out some of the level details regarding which weapons are best based on its own experience. For example at long range the shock rifle performs well overall but there may be situations where the bot does not posses a shock rifle, at a large distance from the enemy. The model still gives a way to discriminate between the other weapons, without having to pre-empt all situations.

Engineering Box

Biassing Assumptions can be encoded into the model to prevent some basic situations known to be bad. One example is placing all weapons which are not the assault rifle at a slightly higher utility initially. This encourages exploration as the bot always spawns with an assault rifle. Our biassing must counter-act the natural bias for this weapon.

A possible adaptation In many of the trials the model ends up giving an average representation of the bot performance rather than a measure of particular weapons. It might be a good plan to normalise the model's conclusions against bot performance for the purposes of system integration. This is left as further work for anyone wishing to extend the system.

7.5.8 Situated Trials

Some situated tests were performed to show how the model worked in a semi-realistic scenario. The issue of strategy design in relation to model use is touched upon.

Engineering Box

Situated Evaluation Purpose In most of our modelling the point of situated evaluation is to provide rigour to the process. In some cases it is used, instead, to allow more a intuition based evaluation about how a particular technique's use might change when applied to actual realistic data from expected domain use. Here it is used to highlight one of the pitfalls of using the model in a greedy fashion.

Two of the models were tested and the following two modules were set up:

7.5.8.0.1 The baseline module The baseline module instructed the bot to move towards any visible enemies whilst shooting at them. The weapon chosen was decided by the game's in built static metric. If no enemy was visible the bot would go on a weapon tour, choosing any weapon at random and going to pick it up.

7.5.8.0.2 The Weapon Hunter Module The second module was similar, except the weapon chosen in enemy presence was decided using the weapon model. In the absence of enemy presence the bot would choose the best weapon according to the weapon model and go to pick it up.

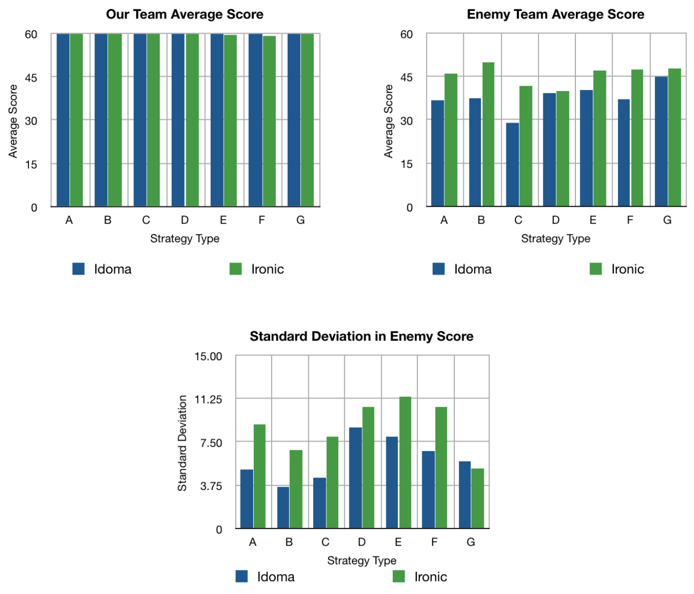


Figure 7.52: Situated Weapon Model Results

The results shown in figure 7.52 show the performance of 4 types of overall strategy. Strategy A is 4 baseline players, B is 2 baseline players and 2 weapon hunters. Strategy C is 1 baseline player and 3 weapon hunters. Strategy D is 4 weapon hunters. In strategy D the weapon hunters were allowed to update the model with observations but in B and C they were not. This is to highlight the importance of exploitation versus exploration issues surrounding modelling in multi-agent domains. Strategies E, F and G are the same as D, C and D except that they use the version of the weapon model which uses the information regarding near and far data to determine the best weapon given enemy presence.

The results for the bot's average team score aren't particularly different for each model but the difference lies in the enemy's average score. Strategy C offers the best result in most categories, limiting the enemy's score (Although the raw result for ironic is only slightly lower than D the standard deviation is lower to the extent that strategy C can be judged to be performing better).

Applying the model straight with all bots, as in strategy D, the results are favourable to that of strategy A where no model was used but not to case C where a mixture of modelling bots and baseliners was used. Case D has a slightly lower average enemy score on ironic pointing to the idea that a combination of the two strategies could be useful. This is dealt with in much further depth in the full strategies section regarding automatic adaptation of the exploitation versus exploration trade-off.

The results show that care must be taken as to how the models are used. They illustrate the importance of tackling the exploration/exploitation problem, from a situated reactive multi-agent perspective. A mixture of explorer bots and exploiters is beneficial to the team play as a whole over both the static and naive learning stances.

7.5.9 A Further Extension

An extension to the weapon model is to tie it into the regions found in the level modelling section. This is a combination of the weapon model with the area correlation model to evaluate the utility for each weapon in each cluster of the level. The evaluation performed was the same as the initial non-situated testing of the weapon model, but only over 3 levels and standard deviation values are not presented. The point of this activity was to show that the weapon utility values can be correlated to clusters to give a weapon hierarchy for each area of the level. Only the discounted models were tested.

7.5.9.1 Further Evaluation

In each section only the weapons and clusters which registered a utility, positive or negative, are shown.

The results present weapon hierarchies for the areas of each level. The problem is that the sparseness of data and average 30% loss of data over levels means that important data is being lost about certain weapons falling out-with clusters. This issue may be less problematic with the use of the alternative level modelling cluster selection methods suggested in the level modelling section to deal with these issues but there is no certainty of this without further testing. Another argument against this type of model is the inherent inefficiency introduced to the weapon utility estimation process. A trade-off must be setup between power and real-time use, and given the down-sides to the model compared with the simpler formulations, this model is not a good choice. It is an interesting avenue of research but may not have too much to offer for this particular system.

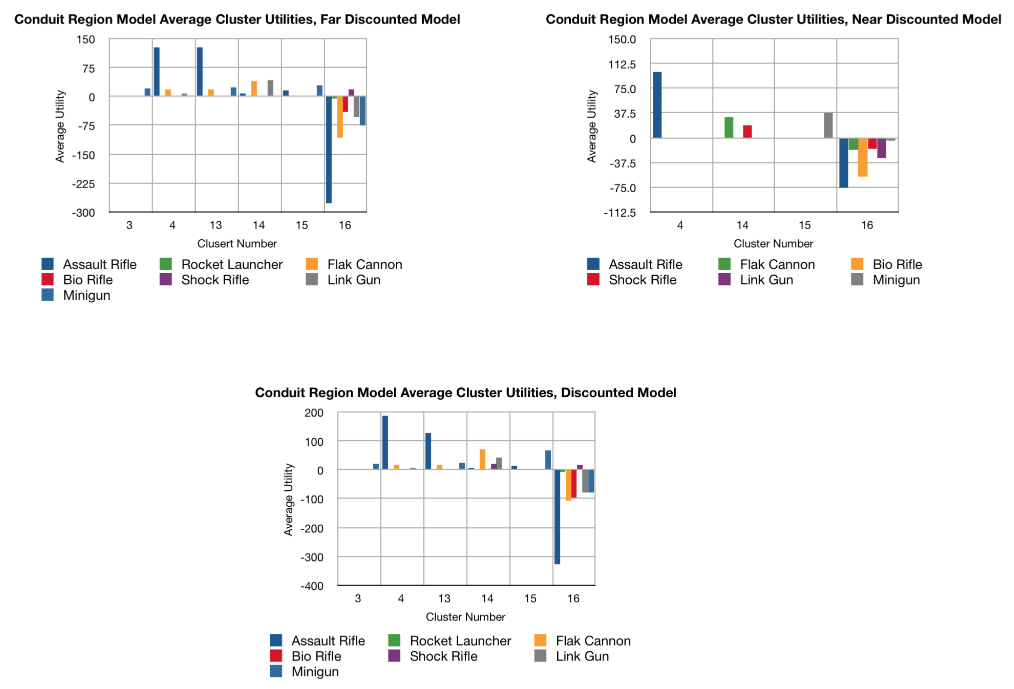


Figure 7.53: Region Weapon Model Results Graphs for Conduit

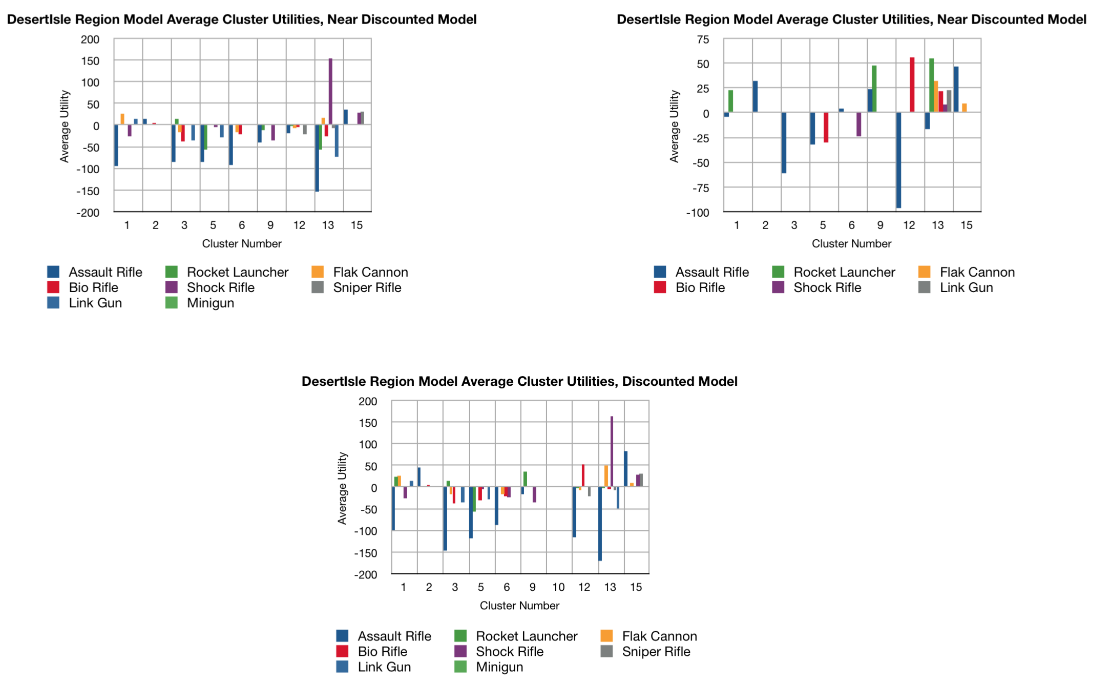


Figure 7.54: Region Weapon Model Results Graphs for Desert Isle

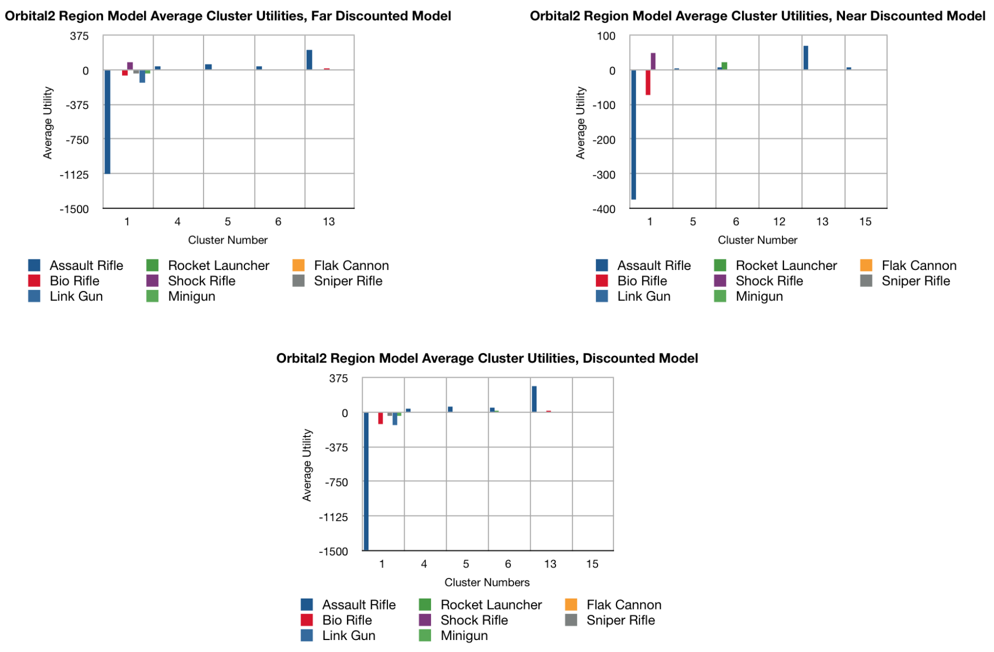


Figure 7.55: Region Weapon Model Results Graphs for Orbital2

Engineering Box

Engineering Issues This was an idea which was easy enough to conceptualise and also easy to defend as a viable performance booster but in practice it just didn't quite pan out as it was just a little too complicated to work with the dataset. One can easily imagine target functions for which this type of approach would be more effective though.

7.6 To Fight or Not To Fight

When dealing with adversaries in a situated environment the question of when to engage is important. For some environments we could imagine that never engaging unless absolutely needed could be an optimal strategy, e.g. infiltrating some form of high security building. In others this balance is tipped much more towards always engaging, such as in a crowded battlefield. In general though, for the wider range of adversarial environments, it makes sense to consider situation modelling in order to decide on a per example basis. Each confrontation should be treat differently, dependent on circumstances. Always engaging visual enemies is not an optimal, or effective, way to behave in our domain primarily because our bots have multiple goals to decide between. As such a fight model was developed to inform bots of when to engage a visible enemy.

Because the fight model makes a decision regarding the bots overall goals it is important that this decision be well grounded and that some level of confidence can be afforded to it. It should also be noted that at this stage the fight model does not directly consider other goals which the agent may have. This is achieved at the behaviour modules layer of the architecture.

Engineering Box

Prototyping One of the nice things about building a system over which you have total control is that you can explore any activity which you feel might bear fruit. This is precisely what happened when it occurred that we had defaulted to the idea of always confronting and attacking a visible enemy. I cannot think of a single predatory animal which always attacks any given opponent in exactly the same manner so why should an artificial agent be any different.

7.6.1 Game Theory

There is much literature surrounding the field of game theory which at first looks applicable to our problem but on further consideration is not really suited to the particular type of problem being dealt with [41, 81, 5]. Most of the game theoretic approaches concentrate on forming complete models of the game in question to then allow probabilistic rules and models to be applied allowing the prediction of further outcomes. These are often based on finding some form of equilibrium into which the game settles at a given point. In our case confrontations between engaging bots have too many independent variables to allow such a conclusive and thorough model to be constructed. The variables which we know will be present are the choice of weapon, bot health, distance from enemy and ammo. On top of this we then have all the level specific factors. It is these level specific factors which determine that we must learn on a per level basis and add credence to our arguments against the use of knowledge transfer. Where we to try to correlate data about fights over several levels the knowledge transfer becomes less trivial as the exact correlations are not known. Even though our conclusions from our fight model in the following sections are only based on health, ammo, distance and weapon choice we also indirectly model the level nuances. For instance having health of less than 50 with an assault rifle on one level may have been learned as a viable situation in which to enter into confrontation but a change of level may alter this drastically as the level design dictates that such confrontations are likely to end badly.

7.6.2 Neural Net Estimation

In order to model fight outcomes we must have some way of adapting from experiences. Data must be integrated about the current weapon and issues such health and ammo. Our original plan was to use the belief network constructed for weapon selection with a different inference pattern to gain a probability of death and of killing the enemy. Based on observations from the section on weapon utility estimation concerning the problems of embedded belief networks for our purpose, we decided to try a technique typically used for offline learning, a feed forward neural net[48, 10, 12].

It is assumed that there is some correlation between the bot's experience of situations and the outcome of said situations. e.g. certain weapons are better if you are a certain distance away from the enemy, others at better at other distances. Constructing a belief network with assumptions about what the connections were puts too much bias into the problem and focuses parameter estimation on the wrong area. Instead of estimating the probability of assumed connections we should be trying to find out what the connections are.

Viewing the neural network as being a set of fully connected nodes the weights of the connections between these nodes can then be estimated. Thus learning in the neural net is approximately equatable to learning in the belief network if the number of perceptron units is large enough to allow fully connected belief networks to be estimated.

Engineering Box

Prototyping Speed Because of their nature as general function approximators, neural nets will always be considered for prediction and classification tasks. They are also generally well understood and training algorithms are readily available so this speeded up the task of prototyping.

This comparison to belief networks and the proposed graphical model from section 7.5 suggested the model needed at least as many nodes as were in the belief net (6). Hence a net architecture as in figure 7.56 was chosen.

The experimentation procedure was to collect a large amount of data from various

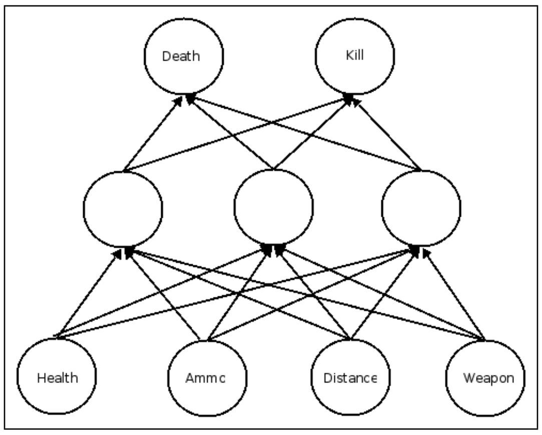


Figure 7.56: Neural net based on Bayesian network

levels of the game. TDM was initially used because this offered an easy entry point into a simple scenario which is well understood.

The data was then split into training data and test data for each level/trial and this was then run through all of the net models tested in a 2-fold trial (first the net was trained with the training data and assessed on the test data, then it was retrained from scratch using the test data and tested on the training data). A variety of different models were tested over the data to see which achieved the best overall performance across multiple levels and on each individual level across 5 different trials. The results were then averaged over these 5 trials to give an average ratio for each level of correct predictions over total number of data points. Each net took in fight data which was collected from the levels. The fight data stored information about each of the following:

- Distance of enemy
- Health at beginning of fight
- Weapon at beginning of fight
- Ammo at beginning of fight

The data was also scaled in the following way in order to ensure that the input to the network and the network weights were O(10).

- Distance = Distance / 1000
- Health = Health / 100
- Ammo = Ammo / 100

The weapon item was encoded using 1-of-M encoding to ensure that there were no interdependencies introduced to the data. Each weapon had its own input to the net which was either on or off. The input and output of the net is shown in figure 7.57.

Different numbers of variably sized hidden layers were used. The net was then trained using the scaled conjugate gradients algorithm [67] as this offered the fastest way to get reliably trained nets. The nets were assessed on how many instances they got correct. Because each net predicted two different outputs the prediction rates were assessed for the cases where both outputs were correct. The predictions for death and kill were assessed using the following function:

$$prediction(output) = \begin{cases} 1 & \text{if } output > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{7.19}$$

This prediction was then assessed against the real outcome of the trial to determine if the prediction was correct or not. The following function expresses this more succinctly:

$$utility(output) = \begin{cases} 1 & \text{if } prediction(output) = actualoutcome \\ 0 & \text{otherwise} \end{cases} \tag{7.20}$$

Trying to predict the outcome of two binomial random variables yields an expectation of getting either of the variables correct 50% of the time and an expectation to get both the variables correct 25 % of the time guessing randomly.

Some more trials were conducted where, instead of allowing the data to be temporally ordered as in the previous case, all the data of one bot was taken then all the data of another and so on, adding each to the dataset thus destroying any temporal ordering. This meant that the training data was all the data of one bot mixed in with some of the data of another bot. The same 2-fold trials were performed with this data, but only used the communal model (This is because if the bot only took its own data then there would be no point as this is exactly the same setup as before). The same trials were than run to determine whether this effected performance.

For the next set of trials the condition of being correct was changed slightly. Instead of looking at when the net was correctly predicting given all the data the results only took into account the cases in which the net was certain of its prediction, i.e. only consider cases where the output was greater than 0.95 or less than 0.05. This tested how the net was performing in the area of the test cases in which it generated strong conclusions.

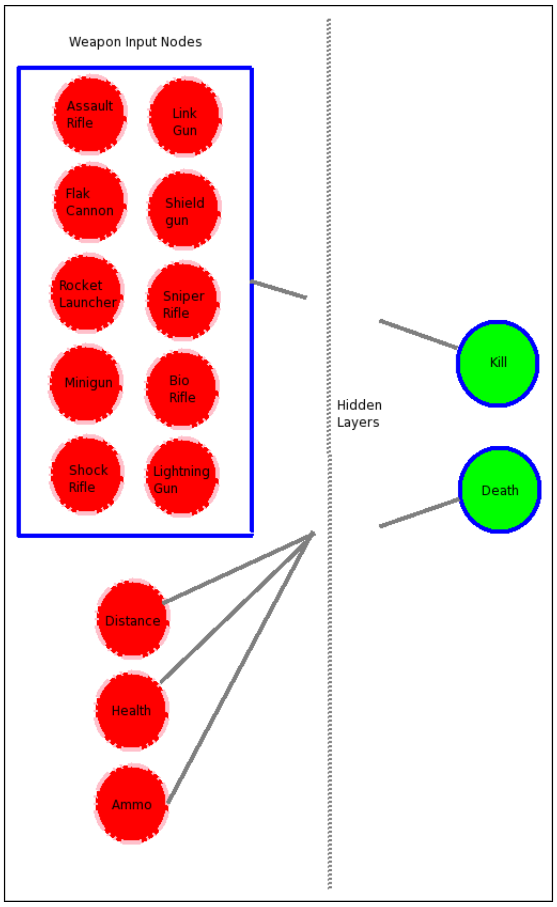


Figure 7.57: Neural Net Structure Diagram

To further investigate this effect another set of trials was run where each bot was trained on only its own dataset. These trained nets were then tested using the datasets of the other bots to see how effectively prediction of outcomes could be performed on other bot's data from one's own. The experimental procedure was slightly different as the whole training set was used to train the net and the whole training sets of the other bots was used to test the trained net. All other experimental details remained the same.

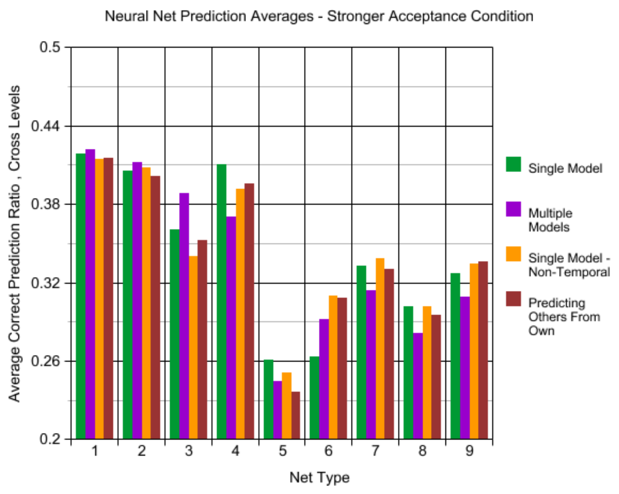
The nets used had the following parameter settings:

	Neural Net Parameter Settings			
	No. Hidden Layers	Layer Size	Training Threshold	Training Tolerance
1	4	4	0.08	0.5
2	4	10	0.1	0.5
3	4	10	0.25	0.5
4	4	12	0.02	0.5
5	1	10	0.9	1
6	4	10	0.25	0.5
7	4	10	0.2	0.45
8	1	10	0.25	0.5
9	1	10	0.2	0.45

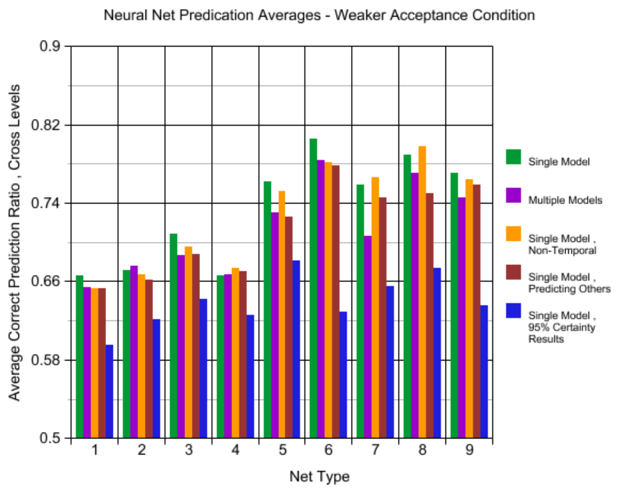
Figure 7.58: Neural Net Parameter Settings

Figure 7.59(a) shows that the performance of the neural net for predictions of both death or kill variables is not particularly good. In some cases there is improvement on the expected 25% accuracy for predicting both variables but this is not significant. It is important, however, to remember that the domain is very noisy and that there is no intention to use this information as the sole decision procedure. The post processing on the neural net outputs will allow these outputs to be placed in the context of a larger strategy for behaviour. There is useful information in the cases where the net only correctly predicts one of the two outputs.

All of the previous trials were run again but this time the acceptance condition was changed to define a correct prediction as getting either the death or the kill prediction outputs correct. This then meant that the condition became a logical *or* rather than a logical *and*. Therefore this is expected to yield a correct result 75% of the time, randomly guessing.



(a) Strong Acceptance Condition



(b) Weak Acceptance Condition

Figure 7.59: Neural Net Trials

7.6.2.1 Discussion

In the case of predicting both outputs the model achieves performance which is negligibly better than guessing at random. In general using the single model is better than using multiple models but the difference in performance is, again, virtually negligible.

Trying to predict one output from the two it is shown that the single model is almost always better than the multiple model, achieving 5% over the random average. When the net is 95% sure of its prediction it is possible to get the correct prediction approximately 60% of the time.

One interesting result is that most of the bots can predict the other bot's fight results to the same level of accuracy as they can predict their own if they have a single model. This is dependent on them having similar capabilities, however the fact that, for similar bots, the situations they encounter are similar to each other is encouraging for us being able to model these situations.

In Summary:

- The overall performance, although not brilliant, does point to the neural net being useful for making predictions about one of the inputs.
- It is possible for one bot to predict other similar bot's data from only their own, showing that data may be useful to other bots for increasing performance. This suggests that the use of communal models will be advantageous
- The exact performance increase gained from the model will be largely dependent on how the outputs of the net are processed

7.6.3 Improvements Using Heuristics

In our naive model we have chosen to select all data as admissible for modelling purposes. This is not ideal as we only have minimal amounts of data for training purposes, and are using some data which may or may not be good for understanding the underlying generator function. One possible improvement of this model is to use heuristics based on some of our intuitions. The idea is that we want the modelling to tweak its performance in certain areas, while not learning much about areas in which we are already fairly certain of what outputs should be. The general idea of heuristics for neural nets is not new. The research presented in [30] concerning the use of neural nets with output heuristics for forecasting loads on an energy company shows this.

With the neural net model in our system there are two contrasting ways to use heuristics:

Data Tampering Altering the data which goes into the training set for the model.

Output Tampering Editing the output of the net in response to known inputs.

Engineering Box

Heuristics The nature of the heuristics chosen is guided by the fact that neural nets are traditionally described as being input, output and hidden layered devices as such this really only gives two conceptual areas to alter. The hidden layer units don't really lend themselves massively to focussed adaptation so this only really leaves the other two options.

7.6.3.1 Data Tampering

In order to tamper with the data we must have a notion of what type of heuristic would be useful and how to represent it.

One viewpoint is that we could add items to the dataset, biassing it towards certain conclusions. This is very similar to the ideas expressed by Denzinger et al[32] concerning the use of a human skeleton of manually set state-action pairs to improve the performance of a genetic process. Neither the skeleton nor the genetic process alone can solve the problem being tackled but when combined they yield a high performance solution. The process could also be compared to the case based reasoning presented by Miles et al[66] albeit in a much simpler form and without the dynamic case base.

One option for achieving this first type of data tampering is to insert, proportional to the dataset size, *idealised* cases which reflected our beliefs about heuristic rules which should be correct in a game situation. This means the output of the net is still a continuous value which is biassed towards certain conclusions. Cases which are not similar to these fall into the area of learned correlations.

This option also gives useful data about modelling in areas where we may not have any data about a certain weapon. If there are no data-points in the training set with settings for these parameters the net is likely to learn that these parameters do not effect the outcome of fights. This is not ideal and we would like to inject *defaults* about these parameters into the modelling process.

7.6.3.2 Evaluation

The same trials as before were performed but this time a data injection of ideal cases was used which represented our beliefs and prior knowledge about the domain:

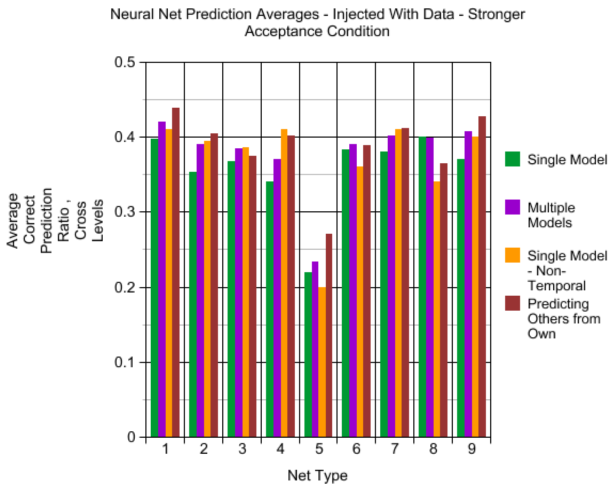
1. If the bot has a flak cannon or minigun with large health, and the enemy is close, then they should be safe
2. At a distance the bot should be safe with a shock cannon and high health
3. With low health the bot should have a high probability of dying
4. With low ammo the bot should have a high probability of dying

Further to this a set of trials were run where the injected data was used but no data was added to training set from the matches. This was to asses the effect of the injected data on the model in the absence of other data.

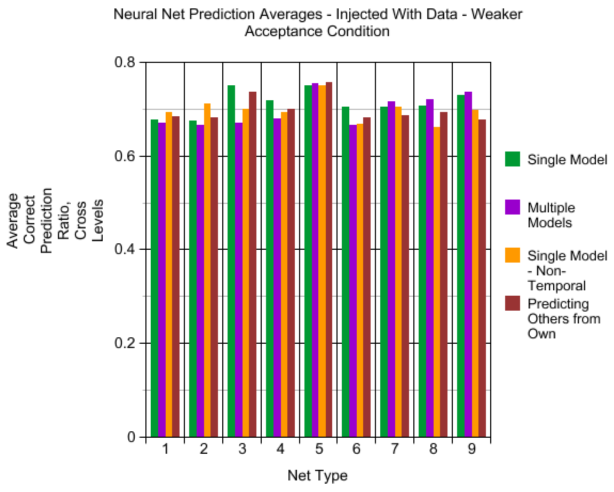
From the strong acceptance condition results shown in Figure 7.60(a) we can see the data tampered model achieves better performance in general. Nets 6-9 are all higher than those in the non-injected case. This said, the best result for any net in any individual category is not significantly better than that of the non-injected case. The injected case is better at predicting other bot's data from ones own model and performs relatively well in the non-temporal case. The best single model and multiple model cases, however, belong to the non-injected set.

The weaker acceptance condition trials present the same story as the strong. The injected data models perform better in general but the best cases are roughly equatable to those in the non-injected model. In the injected case nets 3 and 5 give the best results but in the non-injected case the best results are obtained from model 6.

Another factor to consider is how important the general better performance actually is. Each of these neural nets has a different set of parameters, if they all achieve similar performance this tells us that the initial parameter space is not affecting, to a large extent, the performance of the net. This could mean the target concept is so strong that any parameter setting of the model is capable of finding it. It looks more likely, though, that the model is actually only able to learn to a certain extent with the given data and that adding the ideal data to the set has simply pushed it towards that limitation. The key factor here is that the limitation level appears to be roughly 5% over the expected rate when guessing, for the best model, and we have simply made sure that the parameter settings for the net achieve that level.

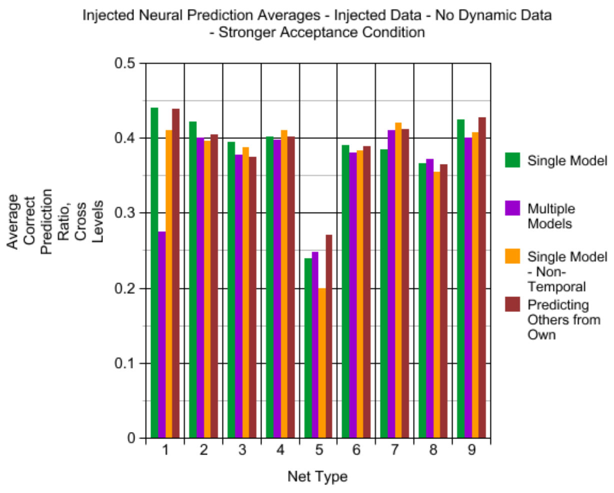


(a) Strong Acceptance Condition

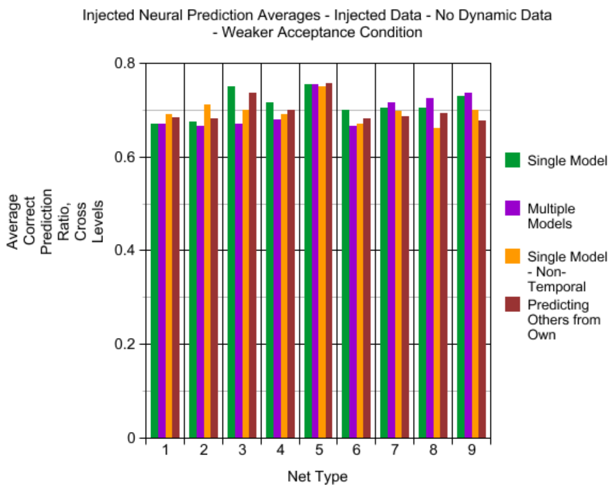


(b) Weak Acceptance Condition

Figure 7.60: Neural Net Trials, with Data Tampering



(a) Strong Acceptance Condition



(b) Weak Acceptance Condition

Figure 7.61: Neural Net Trials, with Data Tampering and No Learned Data

Interestingly the models which do not use the dynamic data do better than those which do in some cases (primarily the strong acceptance condition cases, the weaker acceptance condition cases are almost identical). There are two possible explanations for this. The first is that our predicted data is much better than that which we observe for predicting future data. This seems highly unlikely and it is more plausible that the neural net model is not performing very well for this domain and this data. The model is theoretically well grounded but the patterns in the data may not be pronounced enough for it to be a good predictor.

In the case of the weaker acceptance condition models, with no dynamic data, the similar performance may be caused by the perfect data being very similar to the actual data in the level. It also fits with the assumption of a bound on the learning of the net being a limiting factor on performance.

7.6.3.2.1 An Alternative Viewpoint on data tampering

An alternative viewpoint is that we could attempt to *clean* the dataset by examining each data-point with respect to our heuristic and then changing values which we feel are strongly wrong. This approach is nice in cases where we do not wish to add extra data to our set but can also lead to problems if it is performed religiously. With small datasets this can lead to re-ordained perfect outcomes which is not what we are looking for. The first viewpoint also allows the dataset to take over when we get more data about the situation and obtain a clearer picture of situation which is being dealt with. In some sense the influence of the heuristic *ideal* cases gets minimalised as we add more data.

7.6.3.3 Output Tampering

Although we have not presented any results for output tampering here, as it was felt that the neural net performance was not sufficiently high enough to warrant such tests, it is still worth discussing how this would be achieved.

There are two different viewpoints on output tampering. The first viewpoint is that we could apply a set of heuristic cases to input data which we knew to be correct and thus pre-define outputs for these cases.

A second, more considered approach, is to use the heuristic rules in conjunction with the output of the net to bias the predictions for certain cases into line with what should be happening. This allows the net's prediction to have baring on the result but allows us to specify that the net must be certain that we are wrong in our heuristic estimation before an over-rule of the decision can occur.

A simple example of this behaviour could be as follows:

$$Output(input_x, d, w) = \begin{cases} \text{(net output + 1)/2} & \text{if } d < 0.2 \text{ and } w = \text{flak cannon} \\ \text{net output} & \text{otherwise} \end{cases} \tag{7.21}$$

Where d is a decisiveness factor.

Additions to this could allow the prediction strength of the net output to be considered to allow weighting between net outputs and heuristics to be used.

7.6.4 Nearest Neighbour Estimation

Having tried a relatively complex, although not outlandish, technique we decided to try working from the opposite end of the complexity spectrum with probably the simplest known pattern recognition technique. This was largely based on the fact that the general data pattern didn't seem to support a complex conclusion.

The simplest modelling framework given the situation is a nearest neighbour model[10]. The nearest neighbours model is based on the assumption that the data space which we are trying to model is inherently smooth and that points in this space, which are similar in location, should be similar in classification.

7.6.4.1 Single Nearest Neighbour

In the single nearest neighbour framework novel examples are given the classification of the nearest training set data-point. There are a number of different distance measures. In our case we scaled the data as in section 7.6.2 and then used Squared Euclidean Distance calculation:

$$SquaredEuclideanDistance(X,Y) = (X-Y)^T(X-Y) \tag{7.22}$$

Because we have two different outputs, we consider all examples which have the same classification on both outputs as having the same classification.

Engineering Box
An alternative method could be to consider each separately with entirely different models but given that we hold a prior belief that they are linked together, and are not independent, this would not be the best choice.

7.6.4.2 K Nearest Neighbours

K-Nearest Neighbours [10, 26, 7] is an adaptation of the Nearest Neighbours algorithm which, instead of selecting only the nearest neighbour in the dataset, selects the K nearest Neighbours and then picks the majority classification from this group. This change is designed to avoid outliers in the dataset.

Engineering Box
In our case we took the K nearest neighbours and summed their output values for each output. We then divided this by K and took this is our output classification:

$$Output(X^i) = \frac{\sum_{n=1}^{K} X_n^i}{K} \tag{7.23}$$

This has the same effect as standard K-Nearest Neighbours if we apply a step function to the output but gives a continuous output value.
A key part of the nearest neighbours algorithm is how to decide what to do in situations of ties. In our case a tie is defined as:-

$$Output(X^i) = 0.5 \tag{7.24}$$

Our initial condition for dealing with ties was to take the cumulative distance for each output of the K nearest neighbours and then assign the output as the lesser of the two distances. This was biassed strongly towards a negative rather than a positive prediction by adding 1 to the cumulative distance of the positive summation. It was considered less committal to make a negative prediction. Less commitment is good because we are in a situation where the standard method has lead to a tie with a value of 0.5 showing that there is enough insecurity to justify not jumping to a positive conclusion.

7.6.4.3 The Mahalanobis Distance

An alternative distance measure is the Mahalanobis distance[10, 80]. This distance takes into account the covariance within the particular class of data in the dataset in order to further avoid outliers. The distance measure is altered to the following:

$$MahalanobisDistance(X,Y) = (X-Y)^T \Sigma_{class=class_y}^{-1} (X-Y) \tag{7.25}$$

Where Σ_{class} is the covariance matrix of all examples having the same classification as Y.

This helps to obtain correct classifications in situations such as those shown in figure 7.62 where the nearest neighbours may lie on some hyper-plane within the data such that there is no real variance beyond this hyper-plane. Thus as a class they are distant from the data-point despite being individually close.

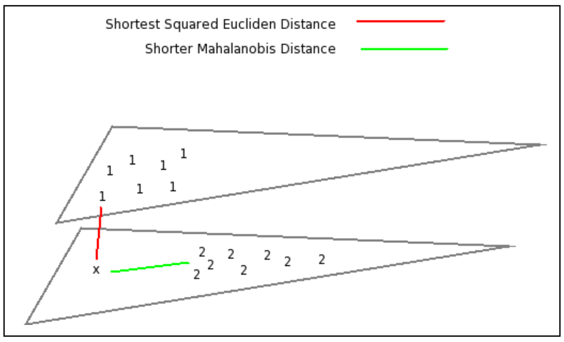

Shortest Squared Eucliden Distance
Shorter Mahalanobis Distance
1
2
x

Figure 7.62: Mahalanobis Example

Engineering Box
The problem with this was that we did not have enough data in our dataset for any given trial to have estimations for the weapon input parameters for any given class. This led to always having a singular covariance matrix due to the determinant being 0, hence for the trials performed in this section we only performed the mahalanobis distance calculations over the ammo, health, and distance parameters.

7.6.4.4 Results

For ease of comparison we have tried to make the results presentation as consistent with those in the neural net section as possible. We have also treated standard nearest neighbours as a special case of K-nearest neighbours where $k = 1$. For higher K values we applied a step function to each output as in the neural net case.

From the strong acceptance conditions there are three important points to be made. Firstly the results are more even across the model selections than in the neural net case. The model is more stable, predictable and reliable. This is a trait of the nearest neighbours algorithm in comparison to neural nets in general and is strongly emphasised in our domain. The performance is also more stable across the individual categories tested. The single model usually slightly out-performs the multiple model case.

The second point is that the performance for any given model is not significantly higher than the best results for the neural net case. The difference is that the categories such as non-temporal data and predicting other bot's data from our own match up with

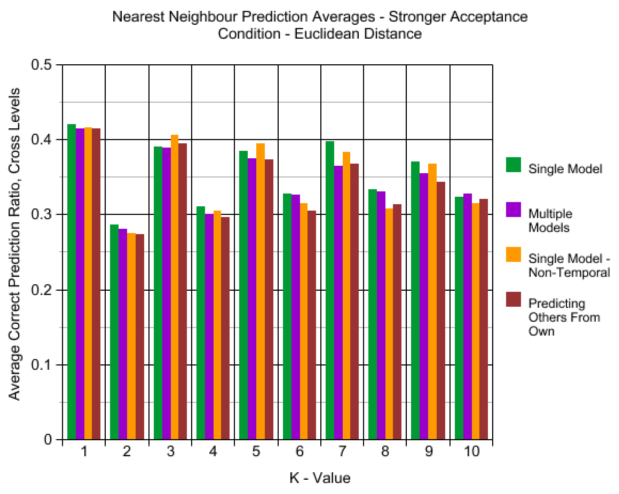


Figure 7.63: Nearest Neighbour Trials, Strong Acceptance Condition, Euclidean Distance

the single model evaluation showing that the model is less susceptible to changes in the temporal nature of the data source.

The final point is that there is an alternating pattern in the evaluation data corresponding to odd and even K values. Odd K values are much higher than lower k-values in general. This is due to the tie condition.

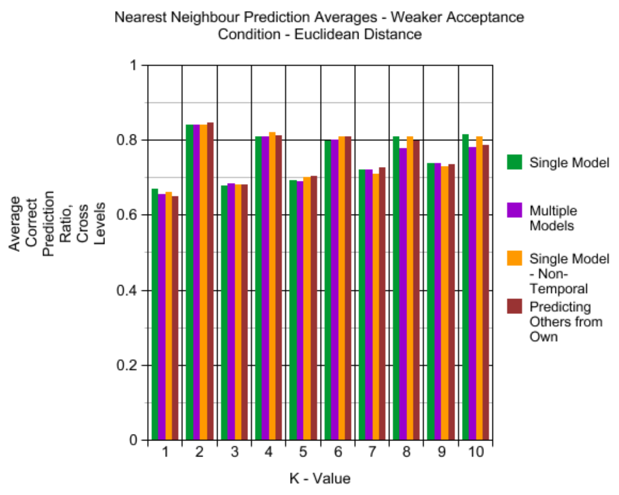


Figure 7.64: Nearest Neighbour Trials, Weak Acceptance Condition, Euclidean Distance

The weak acceptance condition shows a reversal of the pattern observed in the strong acceptance condition case. This occurs due to a combination of the tie condition chosen and patterns within the data. In the strong acceptance condition case the sceptical approach to prediction values led to lower performance but here it does better as the even cases perform better due to this sceptical approach.

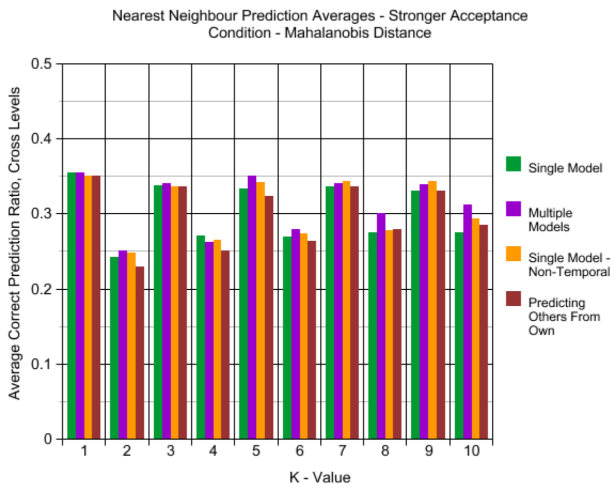


Figure 7.65: Nearest Neighbour Trials, Strong Acceptance Condition, Mahalanobis Distance

With the strong acceptance condition the mahalanobis distance model does worse than the standard euclidean distance model in all categories of evaluation.

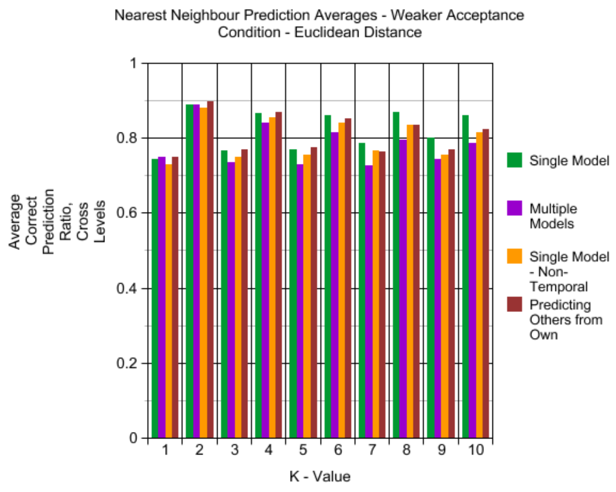


Figure 7.66: Nearest Neighbour Trials, Weak Acceptance Condition, Mahalanobis Distance

In the weak acceptance condition the mahalanobis model is considerably better than the euclidean model showing that it has better qualities for single output optimisation.

7.6.5 Nearest Neighbour Conclusions

The neural nets do better than the nearest neighbour models at predicting both outputs jointly. This is primarily because they are wired to optimise over both outputs and are disadvantaged at predicting individual outputs as a result. This said, the outputs have interpretable meaning and as such, if they are prioritised, can be used in a decision procedure.

The best model was the Mahalanobis distance version of the K-Nearest Neighbours formulation which only used data regarding distance of enemy, current health and current ammo levels. This shows that there is a correlation between these factors and the fight outcome which is clouded by the current weapon selection. This does not mean that the weapons do not effect the actual outcome but rather they effect it in such a way which is difficult to observe from the data and particular type of model considered here. Therefore the decision to model weapons separately seems to be pertinent.

7.6.5.1 Improvements Via Heuristics

Although the performance of K-Nearest Neighbours is better than neural nets we still seek to improve it using a heuristic. For nearest neighbours the easiest way to do this is to alter the way in which we rescale the data. Instead of scaling all data parameters to be within the same range we can alter the scaling on certain variables to be larger than on others. Because the scale of these variables will have much more affect on the distance calculations this will place more importance on these factors. For instance if we believed that health was a larger factor than ammo we might use the following scaling factors rather than those defined in section 7.6.2 :

- Distance = Distance / 1000
- Health = Health / 50
- Ammo = Ammo / 150

This should bias the modelling to consider the health before the ammo. We can choose any rescaling but it should reflect our intuitions about the game and leave enough room for the model to adapt to the data in areas where our intuitions are not strong.

We can also try to perform the same type of data tampering and output tampering as in section 7.6.3 in order to compare the effect that this has. The approach presented in section 7.6.3 will be termed injection while the type of tampering presented in this section is termed rescaling.

The following rescaling factors were used :

Rescaling 1

- Distance = Distance / 1000
- Health = Health / 50
- Ammo = Ammo / 150

Rescaling 2

- Distance = Distance / 2000
- Health = Health / 10
- Ammo = Ammo / 50

Rescaling 3

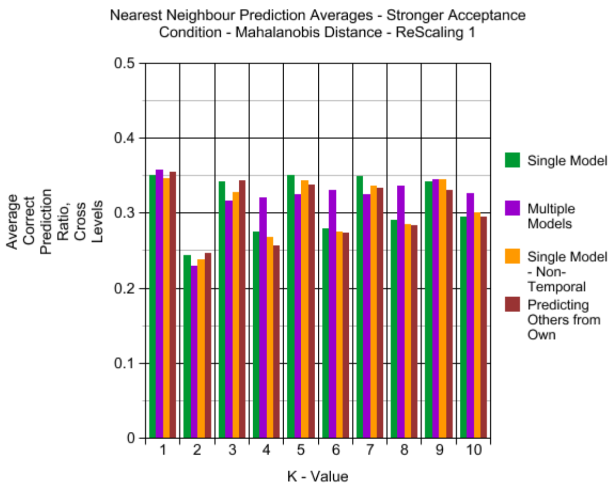
- Distance = Distance / 100
- Health = Health / 200
- Ammo = Ammo / 100

We also tested a second injection set which represented the same set of conclusions as injection set 1 but with slightly less sure numerical values. This was to test the sensitivity of the system to initial injection set specifics.

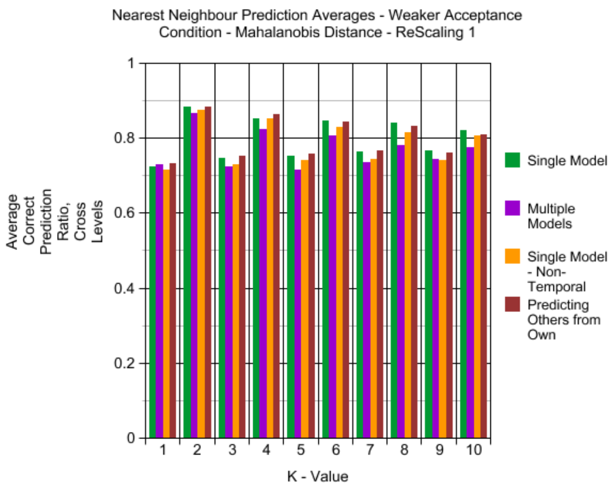
7.6.5.2 Results

The results presented follow the same format as those in the rest of this section. For all trials the mahalanobis distance nearest neighbour model was used as this gave the best results from the previous trials.

The figure for the weaker trials of injection set 1 shows that the single model case outperforms the non-injected version. The single model also outperforms the multiple model version. This shows two things, that the injected data is making a difference and that this difference is more severe for a single model. The explanation is most likely that the extra data is complementing the learned data in the single model case

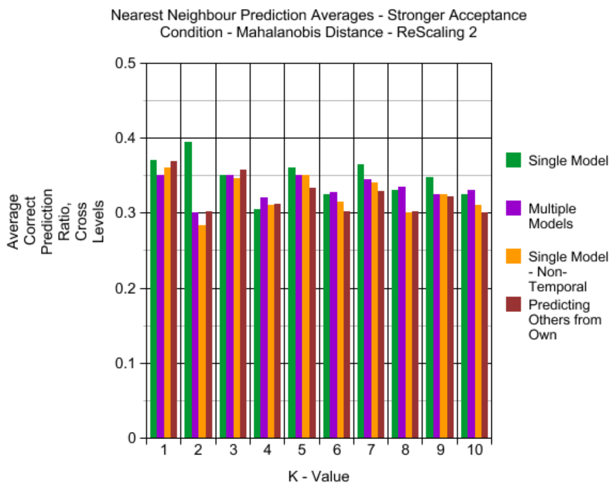


(a) Strong Acceptance Condition

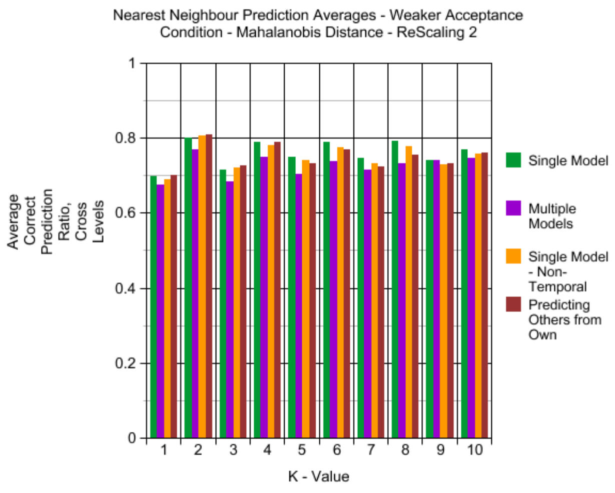


(b) Weak Acceptance Condition

Figure 7.67: Rescaling 1

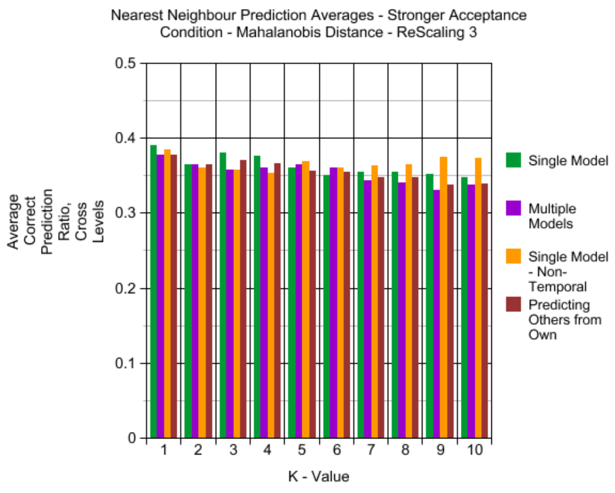


(a) Strong Acceptance Condition

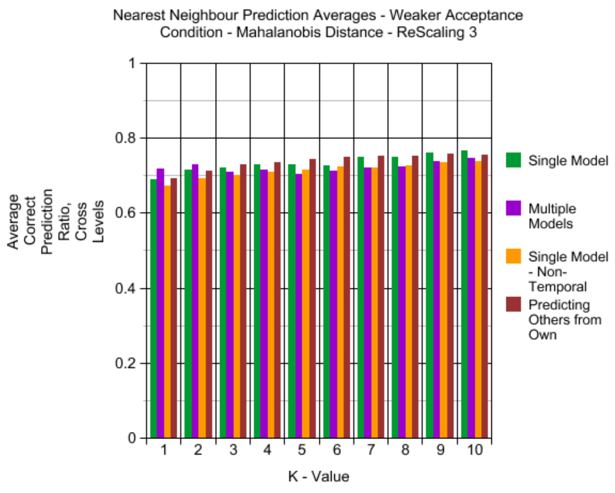


(b) Weak Acceptance Condition

Figure 7.68: Rescaling 2

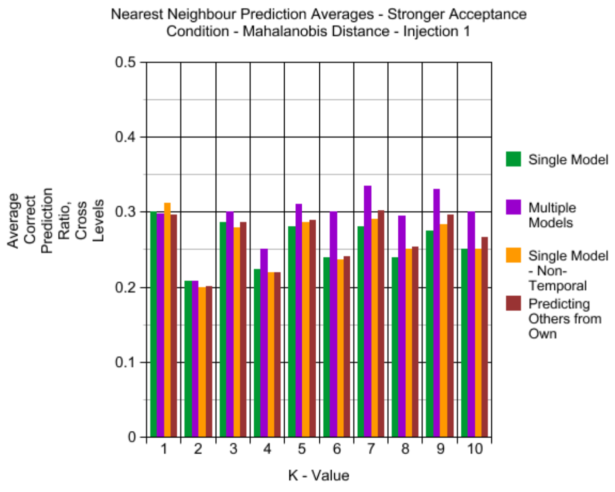


(a) Strong Acceptance Condition

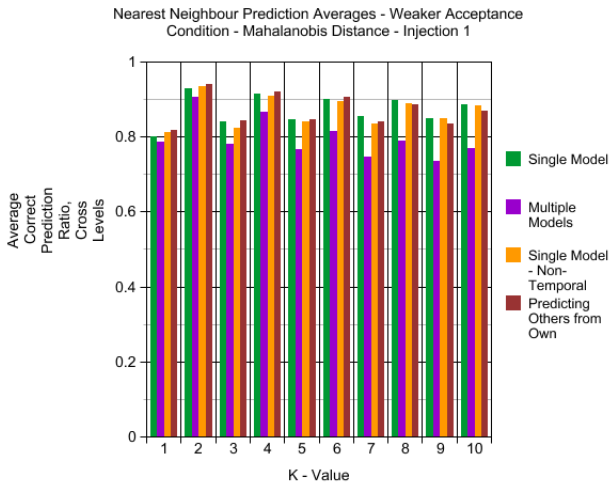


(b) Weak Acceptance Condition

Figure 7.69: Rescaling 3

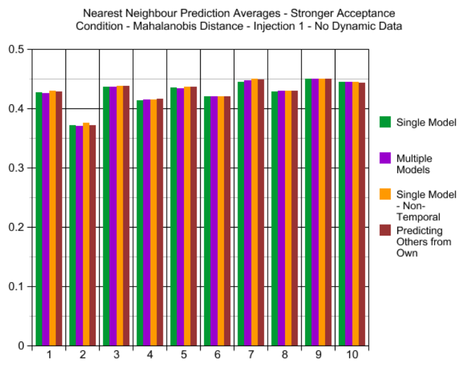


(a) Strong Acceptance Condition

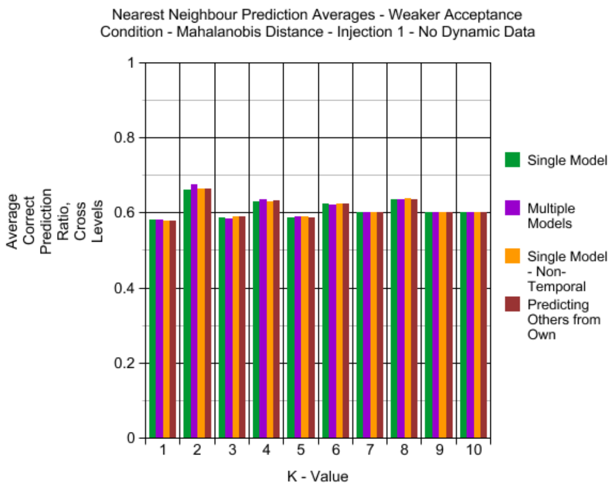


(b) Weak Acceptance Condition

Figure 7.70: Injection 1



(a) Strong Acceptance Condition

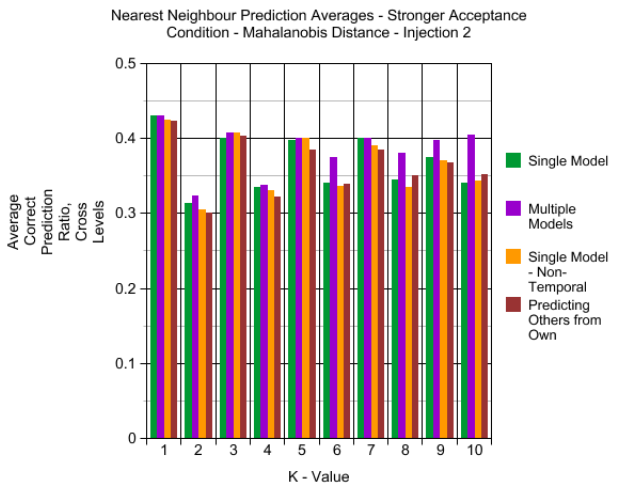


(b) Weak Acceptance Condition

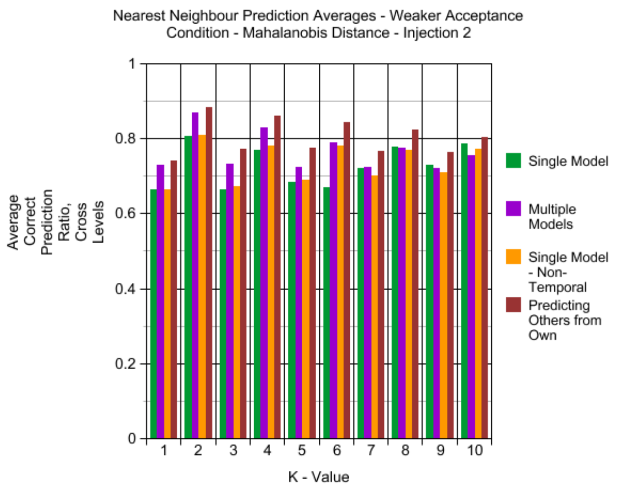
Figure 7.71: Injection Set 1 and No Dynamic Data

but is overwhelming it for the multiple models case, with each single model not having enough data to form a beneficial data set.

In Figures 7.71(a) and 7.71(b) the results are shown for the model with only the injection data and no data learned from the level. The purpose of this is to judge the effects of the injection data. Here we see that the model performs much worse with only the injection data. It also performed worse with only the learned dynamic data yet the combination of the pair creates a model which performed better. This shows that the learned data can alter supplementary intuitions in a meaningful way leading to better modelling.



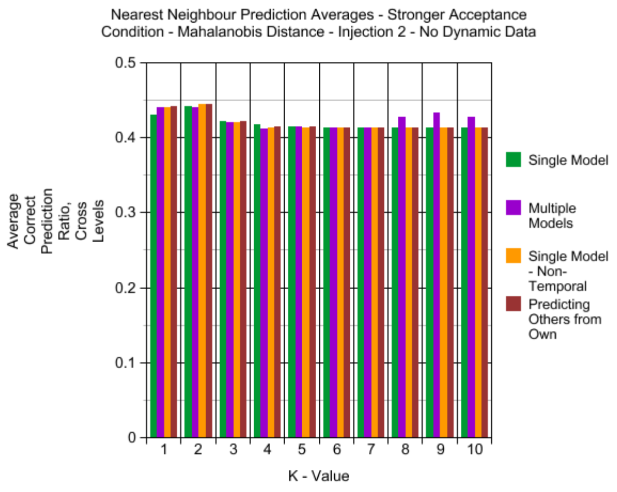
(a) Strong Acceptance Condition



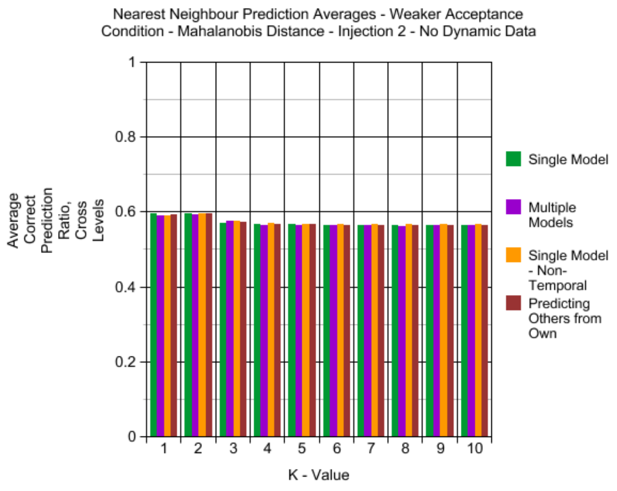
(b) Weak Acceptance Condition

Figure 7.72: Injection Set 2

The results for injection set 2 are not as good as for injection set 1 and the pattern across K values is different showing that the performance increase seen can be dependent on the initial injection data set. The whole process of data tampering with injection sets seems to be hinged quite strongly on the particular data used and how well grounded the intuitions are. One of the benefits of working with a domain where some expert knowledge is present is that this type of model alteration is possible but it would a dangerous result to generalise to domains where such knowledge was not readily available.



(a) Strong Acceptance Condition



(b) Weak Acceptance Condition

Figure 7.73: Injection Set 2 and No Dynamic Data

Even though the second injection set is not as good, the effect of removing the dynamic data remained roughly the same showing that the performance increase is due to the way in which the supplementary data interacts with the learned data and is not simply in the data itself.

7.6.6 Heuristics Conclusions

Heuristics, if used in a subtle and careful manner, can increase model performance but even slight changes can turn this in the opposite direction. Of the two heuristics rescaling was outperformed by data injection. This reflects the fact that with smaller data sets adding more data is likely to improve performance if the added data is sensible and representative of the target concept.

There is room for further work in this area, in particular a more careful and thorough analysis of injection sets and how these effect performance in a variety of domains towards developing a framework or algorithm for the automatic creation of such sets for given domains.

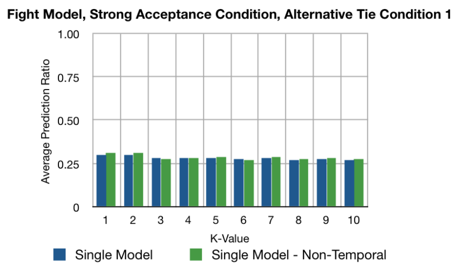
Engineering Box
Care must be taken not to bias the training set to the specific test set being used by involving it in the training process. This is, however, true for all machine learning and is good practice.

7.6.7 Alternative Tie Conditions

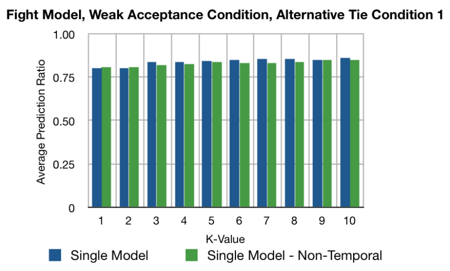
In the previous section we presented results using a biassed condition for cases of ties. In this section we present a few simple alternatives and show how they effect our optimal result from the previous section. This is to show two things. Firstly it helps to motivate our choice of tie condition. Secondly it highlights some of the effects that the effect of the tie condition can have on the model and why it should be carefully considered.

The first alternative tie condition tested was to look at the total distance of all the examples for each output value and then take that with the smallest distance as the output from the set of K. Figures 7.74(b) and 7.74(a) show how the model performs. The results are not an improvement on the normal tie condition and both are worse.

The second alternative condition was to always predict 1 for outputs in tie conditions. Figure 7.74(a) shows that this was not an improvement on the standard tie condition.



(a) Strong Acceptance Condition



(b) Weak Acceptance Condition

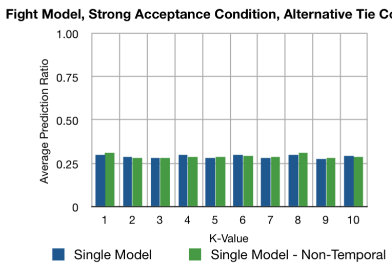
Figure 7.74: Prediction Scores for Alternative Tie Condition 1

The explanation for this lies in conservatism of prediction. A death and a kill are both events which can happen or not. In situations where the model is not sure then the best option is to hedge bets and not predict either rather than jumping into one court without sufficient evidence.

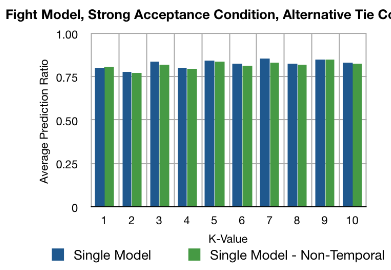
In both cases the step result, with either the odd or the even result being higher than the other is not present. In the case of taking both outputs and looking at the distance this is obvious as we are not introducing any particular disjointed behaviour. In alternative tie condition 2 we still do this though and this is more down to the bad choice of always committing to a positive outcome despite the evidence.

7.6.8 Situated Evaluation

As with the enemy and weapon models a situated evaluation was performed. The trials were set up on the game level Idoma. Strategy A was a baseline, the same as that used in the enemy modelling section. The bots in Strategy A were limited to assault rifles. This was a valid thing to do because the mahalanobis distance model used did not take into account weapon preferences.



(a) Strong Acceptance Condition



(b) Weak Acceptance Condition

Figure 7.75: Prediction Scores for Alternative Tie Condition 2

Strategy B was the first attempt to use the model in a naive way. In the event of a visible enemy player the bots would query the fight model regarding whether they should shoot at the enemy or not. If the model predicted a kill they would shoot at the enemy and approach them, otherwise they would continue to play randomly as in the case of no visible enemy. The weapons were limited to assault rifles.

Strategy C was a slightly better attempt to use the model. In the presence of a visible enemy the bots would query the model and only approach the player if a kill was predicted. If the model predicted a death then the bot would execute an evasive manoeuvre.

Engineering Box
The evasive manoeuvre was performed by calculating the parametric equation of the line between the bot and the enemy and then using this to project to a point roughly 20 times this distance behind the bot in the opposite direction of the line. The nearest navpoint in the level to this point was then calculated and set as the destination for the bot. Although not the optimal avoidance strategy (alternatives include ideas such as artificial potential fields[21, 42]), it generated acceptable behaviour.

The bots would always shoot at the enemy even if not approaching them.

Strategy D was almost identical to the baseline strategy except the bots were allowed to use all the weapons. Thus when a player was visible the bot would choose the best weapon based on the built in static weapon selection strategy.

Strategy E was identical to strategy C except all weapons were allowed. A different model was also used to take advantage of the extra information offered by the weapons. Even though the model achieved slightly worse results in the clinical evaluation, the extra data offered by situated use could mean it still proves better in testing. The model used was the euclidean distance model with K = 2 and injection set 1.

Strategy F was a mixture of exploitative players and exploring players. There were 2 bots using strategy E and 1 using strategy D. Only the strategy D player was updating the model.

Figure 7.76 shows that using the strategy in a naive way is not optimal. It offers the worst level of performance possible as it did not achieve a single kill. The problem is that even if you choose not to approach an enemy in the game, shooting them should always be performed as a defence mechanism. With better situation modelling it may be possible to determine more about the scenario and whether a stealth retreat could be made but in our case it is better to shoot.

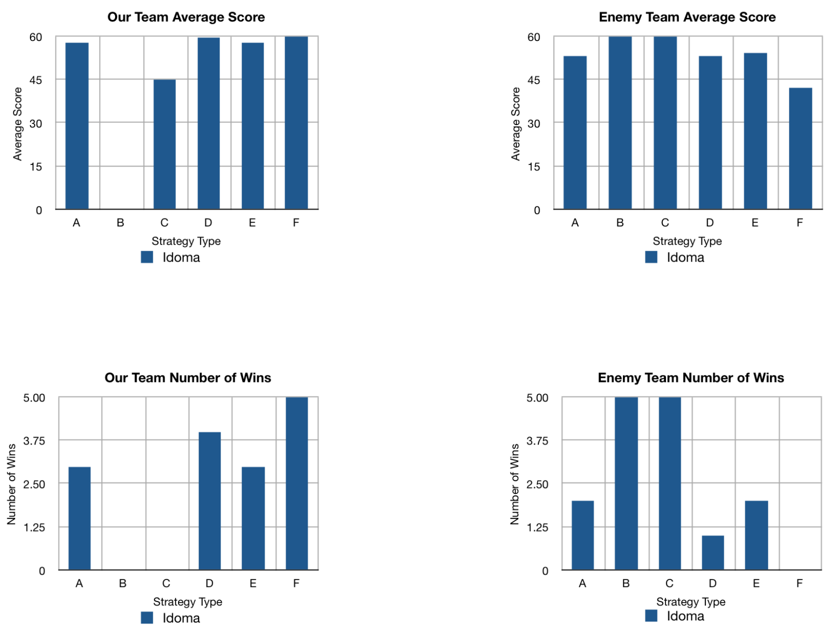


Figure 7.76: Situated Fight Modelling Results

This is exemplified by strategy C which gives performance which, although still worse than the baseline, is at least offering some form of resistance.

When we move to strategies involving weapons there is a very interesting situation. Having all the members of the team using the model and updating it leads to a worse result than the baseline. Having one member of the team updating the model while exploring leads to a drastic improvement in performance giving the only model set which achieves 5 wins in 5 matches.

7.6.8.0.1 Conclusions Just using the model naively is bad and doesn't give good performance. Also having the entire team using the model at once while updating it and using the information in a greedy manner does not give good performance.

The best approach is to balance the team with some exploring agents and some exploiting agents. The performance of this is considerably better than either the baseline or the naive use of the model. This gives the best motivation for our proposed solution to the exploration versus exploitation problem. This is discussed further in chapter 13.

7.6.9 Ensemble Networks

From the above results we can see that even a simple modelling technique such as the nearest neighbour model can offer some acceptable level of performance. This then begs the question of which examples the nearest neighbour network gets correct and which the neural network framework gets correct. If these differ drastically a combination of the two might exhibit better performance by specialising certain cases to each model. This concept of combining multiple models into one with expected better performance is sometimes termed an *ensemble model*[34].

Experimentation with an ensemble model showed that there was no statistically differing pattern to show that the either the neural net or the nearest neighbours model got particular examples correct where the other didn't. As such there was no viable way to combine the two into one model based on any scientific assumptions or heuristic intuitions. The same can be said of a combination of models from either of the two sections internally (for instance combining a model with a mahalanobis distance measure and no injection with a euclidean model with rescaling and injection), as again, no pattern was displayed to suggest that any increase in performance could be achieved.

7.6.10 Alternatives

When dealing with decision boundaries there is an almost endless list of alternative machine learning techniques which could be considered. The most notable are support vector machines (SVM) and Bayesian methods. Of the two SVM's[88], when applied with the correct kernel, are generally seen as the more powerful technique. The main problem is that they can take a large amount of time to train and again are often difficult to use with small datasets. There is also more work in deciding on the appropriate kernel to use.

Engineering Box
If these could be trained in an online manner which was comparable computationally with heuristic models then these might be a more viable alternative.

7.7 Game-Type Specific Route Modelling - CTF

In any adversarial environment situated agents with goals are going to have to move around in a way which enables them to deal with said adversaries. If certain routes are good for achieving goals then we would like the agent to be able to learn to take these more often and conversely avoid bad routes. If the environment changes between instances and no knowledge transfer is possible we would like the agent to learn these routes quickly during trial execution.

In the enemy modelling section it was shown how a density model could be used to rank path nodes, by giving them a utility which reflects the probability of enemy presence. In this section we show how a density model can be used to rank nodes for approaching and returning a flag in CTF. We also present a more specialised model of flag return play.

Engineering Box

Parzen Density Window Flexibility Having already established the relative good performance of the density model approach to node ranking it seemed efficient to use this for problems having a similar set of parameters and potential modelling target functions. This flexibility is largely a product of the overall versatility of parzen density window estimation techniques.

The mechanism for applying the standard density estimator is as follows:

1. When a bot is spawned they begin storing nodes on the path they take.
2. When they reach the flag (go to 3) or die (return to 1) they update the approach model with the path they took.
3. They begin storing the nodes on the path taken.
4. When they reach the home base with the flag or die they update the return model with the path they took (return 1).

In order for the mechanism to perform modelling updates, nodes in the level are given a flag approach utility value. This is then updated in the event of an update from a bot. In our case we used a parzen density estimator based on Gaussian distance from the path as a whole.

$$P(x|Model) = \left(\frac{1}{N}\sum_{i=1}^{N} G(x - x_i, 2\sigma^2 I)\right) \tag{7.26}$$

where $G(x,H)$ is a Gaussian Kernel defined as

$$G(x,H) = \frac{1}{\sqrt{|2\pi H|}} e^{-\frac{1}{2}(x)^T H^{-1}(x)} \tag{7.27}$$

and each x_i is a node in the path.

Engineering Box

Updates With each update the utility of any given node was augmented with the point's cumulative probability of having been on the path (as calculated above) and normalised so that all node probabilities were between 0 and 1.

A separate model was used for the capture and the return portions of the flag run.

7.7.1 Evaluation

To evaluate this model several tests were run and the resulting updates were performed upon flag capture and return. The effect on the nodes in the level was then graphed in the same format as in the enemy modelling section. The approach and return models are shown separately.

Each model was tested on the levels Citadel, Grassy Knoll, Absolute Zero and Lost Faith over the variance values 250, 500 and 1000.

Only two of the levels are shown. The other two can be found in the appendix as the results were similar to those shown.

For the first set of tests one bot was set to run to the enemy flag and then return to base again.

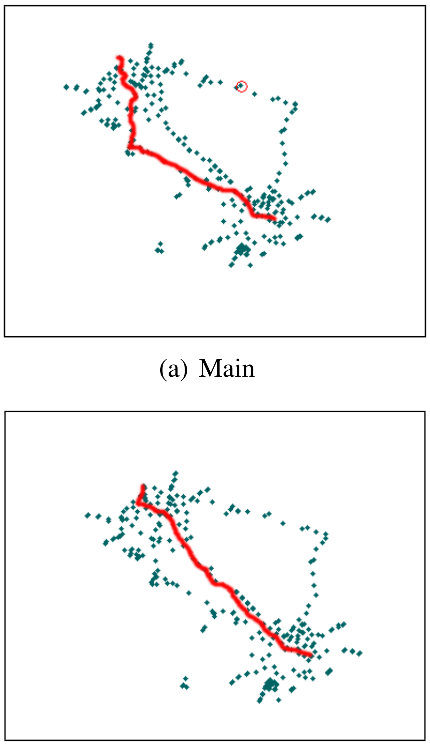
Engineering Box

This was a simple test to evaluate the power of the model to represent a known path and how these representations reflected what happened. This doesn't test the element of the model which handled path failures, which is dealt with in the next set of evaluations.
It was also a basic test to determine if the model would work similarly as it did in the enemy modelling section for a different target function with different parameter type.

In the diagrams in this section, the numbers represent different time points within the match.

7.7.1.1 Citadel

Figure 7.77(a) shows the main path the bot takes to and from the enemy flag back to base. Figure 7.77(b) shows an alternative path that is sometimes taken based on the bot's starting position (The internal path finder is optimised for distance so this can alter the path taken if the bot's starting position is nearer the goal via the alternative path). Both paths are shown as sometimes both are reflected in the model.

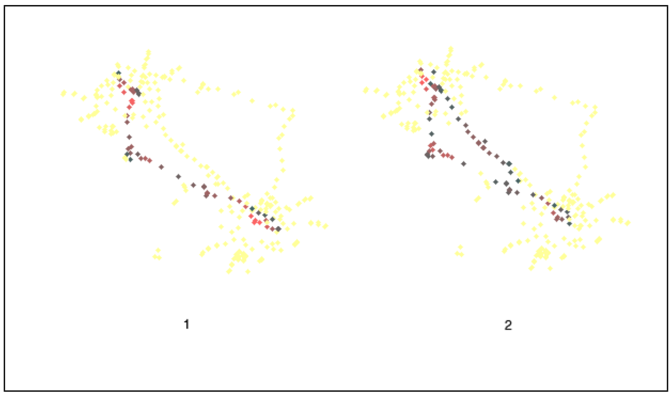

(a) Main

(b) Alternative

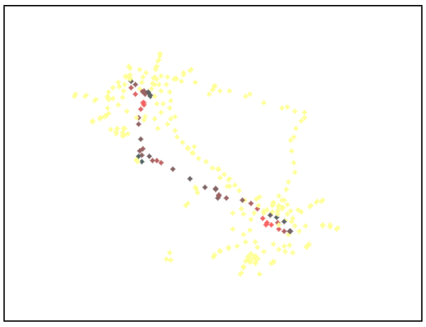
Figure 7.77: Citadel Approach and Return Paths

7.7.1.1.1 Variance: 250 From figure 7.78 we can see that the model with variance 250 creates very narrow density influences why only effect very specific areas. This is good for a basic representation of the path taken but lacks power in terms of influencing the nodes around the path. For path finding, this can be visualised as walking along a ridge on a mountain. If the ridge is not particularly steep then you can use alternative routes around you by walking down the edges of the ridge and continuing along these paths. However if the ridge is ridiculously steep then there is no hope of getting off it and you must slavishly follow it or risk taking paths of unknown origin. This model is just memorising the path taken and offers little extra power over simply storing all the nodes taken and only reinforcing their utility by some set amount. The use of this model would lead to steep ridges like those in our analogy.

In one case we see the model representing two different paths but both are extremely tight to the paths taken rather than being a smooth influence throughout the level nodes which surround the paths.



(a) Approach



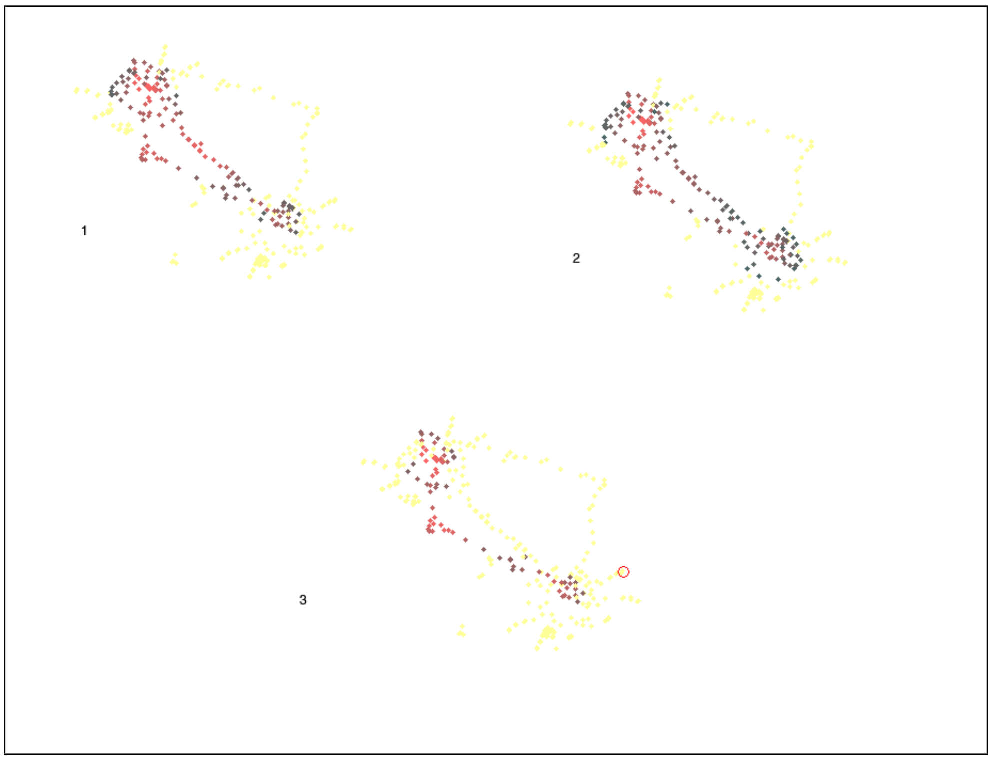
(b) Return

Figure 7.78: Flag Route Density Models for Citadel with variance 250

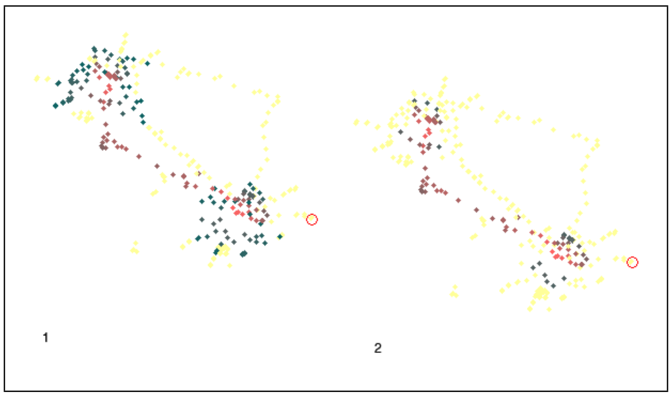
7.7.1.1.2 Variance: 500 The variance 500 model in figure 7.79 has much more density spread than the variance 250 version. Both paths taken are represented but the left hand path is more strongly shown. This is what we would expect as this path is more prevalent in the bot's play. There is an even distribution of utility of reinforcement across the nodes without reducing the density to being non-descriptive and uniform.

A uniform density would provide the opposite problem whereby all the nodes in the level would be equally reinforced and no evaluation of potential paths could be performed.

7.7.1.1.3 Variance: 1000 The spread in figure 7.80 here is too uniform across the level. It was discussed that the model should not create too steep a ridge but we also want a definite gradient of utility throughout the level to guide any path finding mechanism. Reinforcement of one path in this level is influencing both paths too heavily.



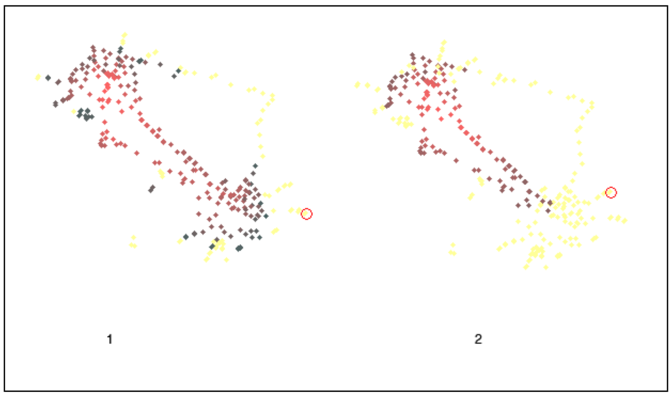
(a) Approach



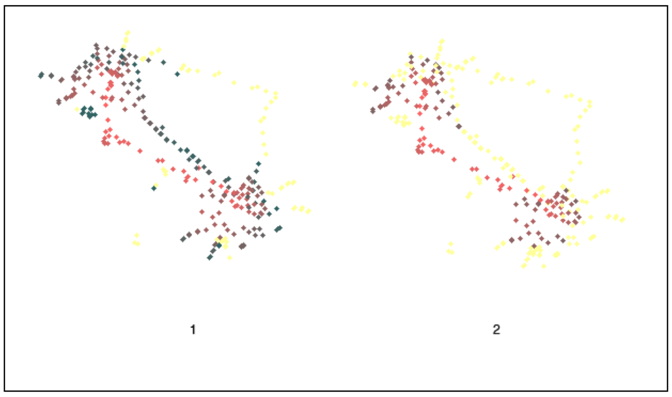
(b) Return

Figure 7.79: Flag Route Density Models for Citadel with variance 500

This is likely to lead to the false conclusion that the middle section of the level is equally probable to the left hand path. In this case the middle path should have less utility.



(a) Approach



(b) Return

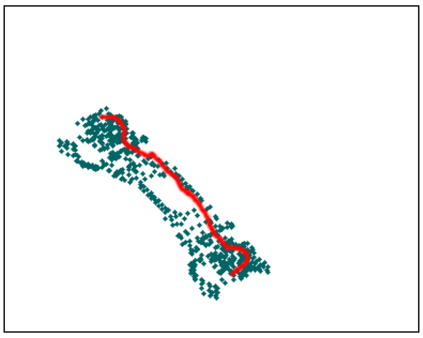
Figure 7.80: Flag Route Density Models for Citadel with variance 1000

7.7.1.2 AbsoluteZero

Figure 7.81(a) shows the main path the bot takes to the enemy flag. Figure 7.81(b) shows the route back from the enemy flag to home base.

7.7.1.2.1 Variance: 250 As shown in figure 7.82 In the approach case both paths are strongly represented but the ridges are, again, steep with too low a variance value in the directions perpendicular to the path.

7.7.1.2.2 Variance: 500 Figure 7.83 shows that there is a smooth density gradient throughout both the *return* and *approach* stages but the path is clearly picked out at


(a) Main

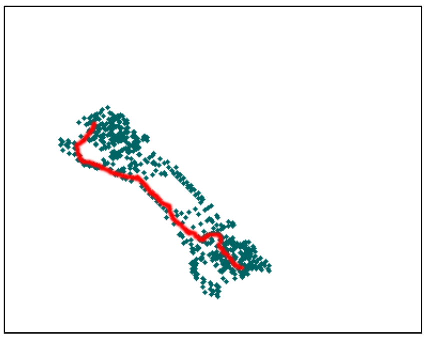
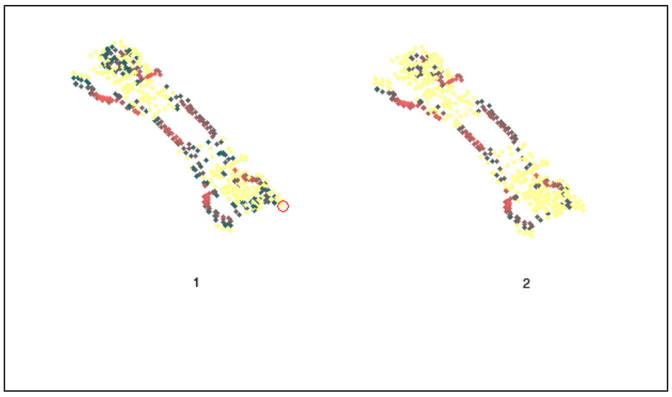
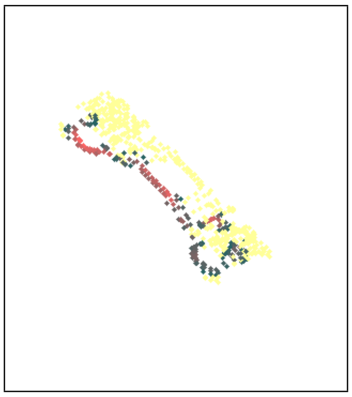

(b) Alternative

Figure 7.81: Absolute Zero Approach and Return Paths



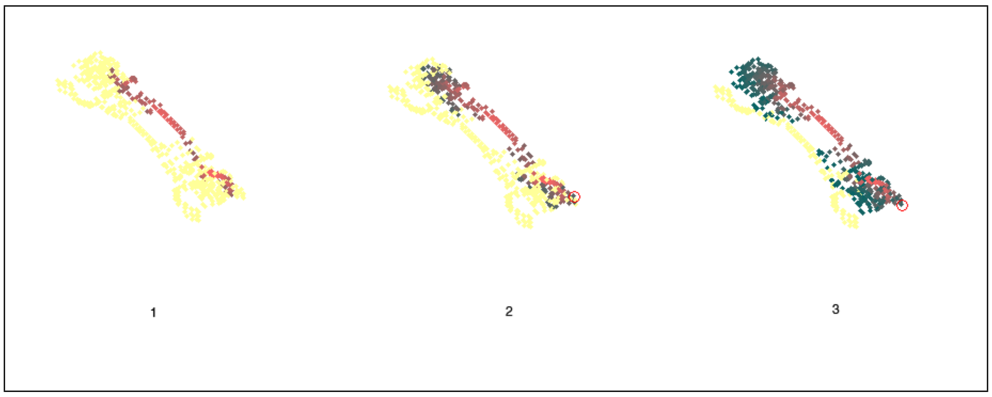
(a) Approach



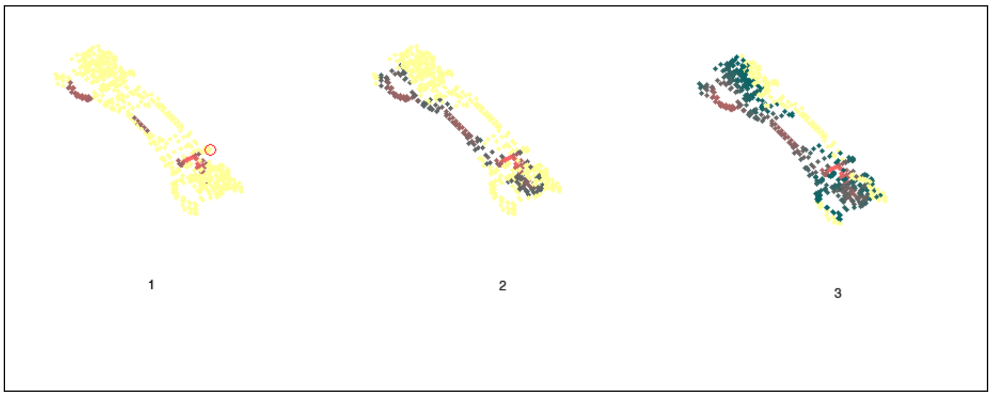
(b) Return

Figure 7.82: Flag Route Density Models for Absolute Zero with variance 250

high threshold levels. The influence of the reinforcement is also well represented as it does not extend over the bridge section to the other pathway. This is partly due to the correct variance value and is also a feature of parzen density estimation in general ; being able to generate density functions which are, in some sense, non Gaussian.



(a) Approach

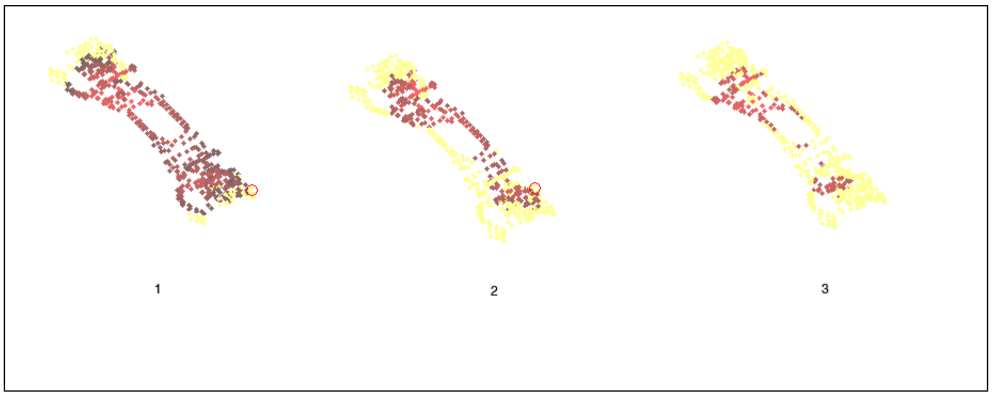


(b) Return

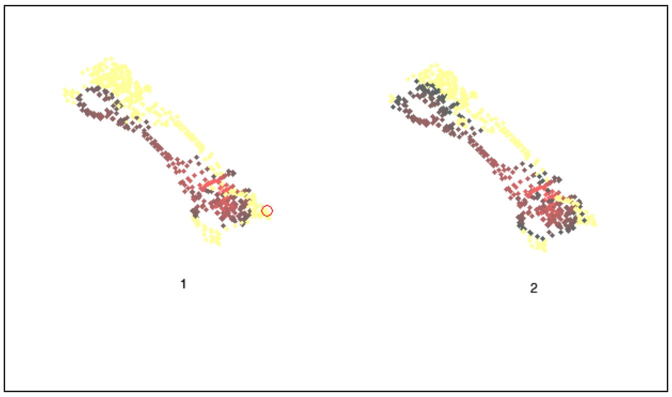
Figure 7.83: Flag Route Density Models for Absolute Zero with variance 250

7.7.1.2.3 Variance: 1000 All of the results presented in figure 7.84 are at high threshold levels as at low threshold levels the whole level was being swamped by the density function. The paths are well represented but note there is little change in the gradient, or shade of red, in the nodes signifying a lack of usable gradient of influence throughout the level.

This is showing too much uniformity in the way the model is effecting the nodes throughout the level.



(a) Approach



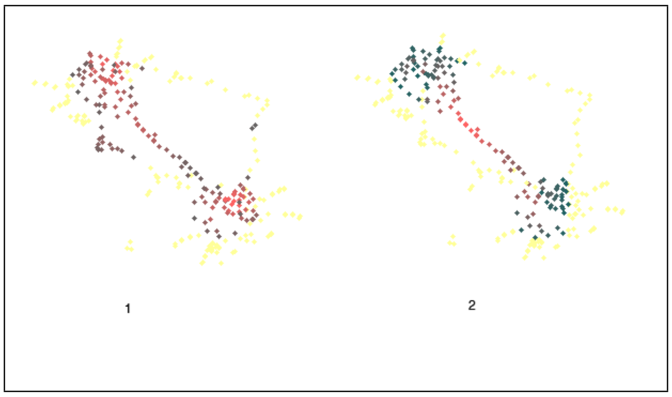
(b) Return

Figure 7.84: Flag Route Density Models for Absolute Zero with variance 500

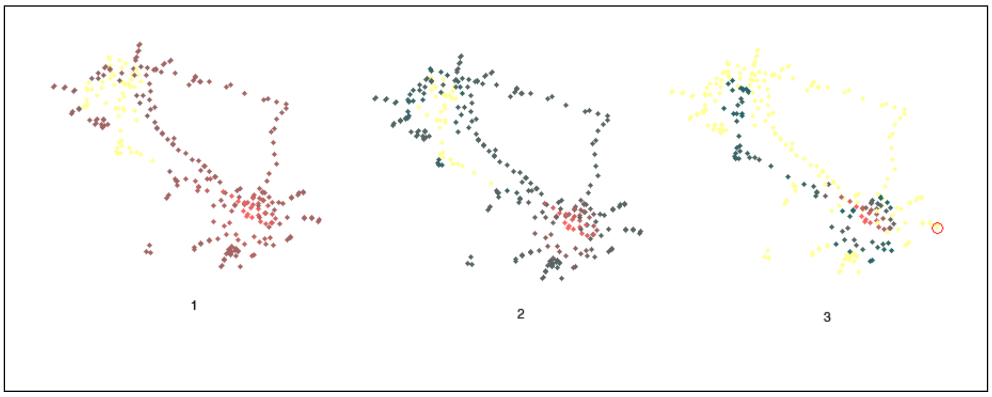
7.7.1.3 Conclusions from evaluation

The modelling procedure shows that the path nodes taken can be used to reinforce a set of node utilities over the level which generate a continuous surface or influence map. The model with variance 500 performs best (of those which we tested. It was beyond the scope of this thesis to perform a full scale variance test.) and does not suffer from the artefacts of the other two variance values tested in these trials. This section only tested the positive reinforcement element of the model but showed that it worked as expected with our specifications for a useful model.

7.7.1.3.1 Further Evaluation 1 In the following section the experimental setup was the same as before except that an enemy bot would now be present playing against our single bot. Based on the results presented in the basic evaluation only the model with variance 500 was tested as this showed the best performance.

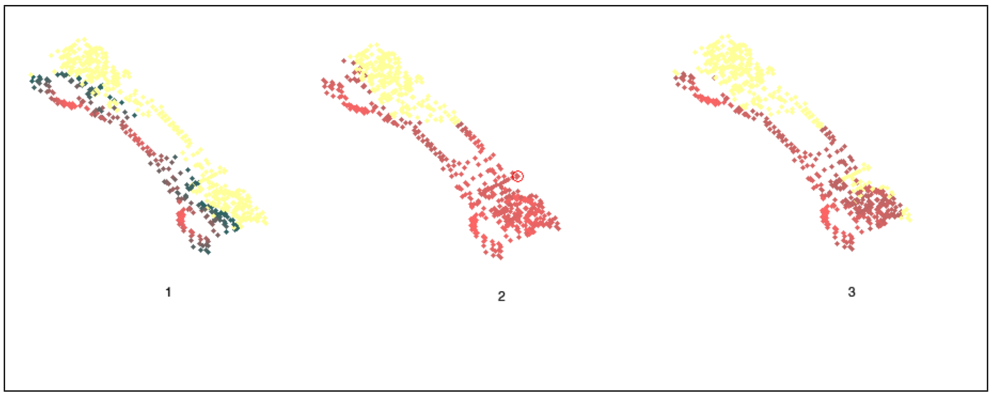


(a) Approach

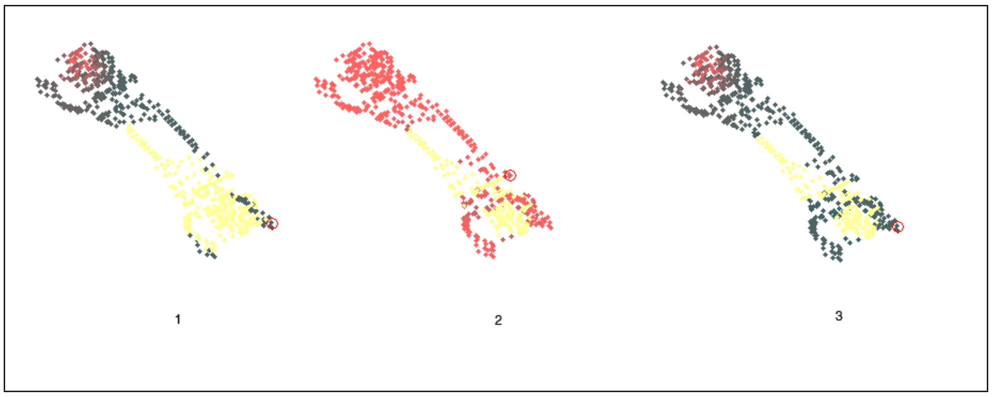


(b) Return

Figure 7.85: Further Models for Citadel



(a) Approach



(b) Return

Figure 7.86: Further Models for Absolute Zero

In both the Citadel (Figure 7.85) and AbsoluteZero (Figure 7.86) cases we see that for the most part the approach route is well picked out. The return route is sometimes represented but generally speaking the other bot's influence is forcing negative reinforcement of the main path. The visualisation displays the nodes not affected by the density estimator as being more likely, which would fit with expectations regarding how this should work. In the section on weapon modelling we touched upon the idea that having a negative reinforcement element of the model helped to encourage exploration in response to negative data and this would appear to be the case here. All nodes becomes more likely than those on the failed attempted path.

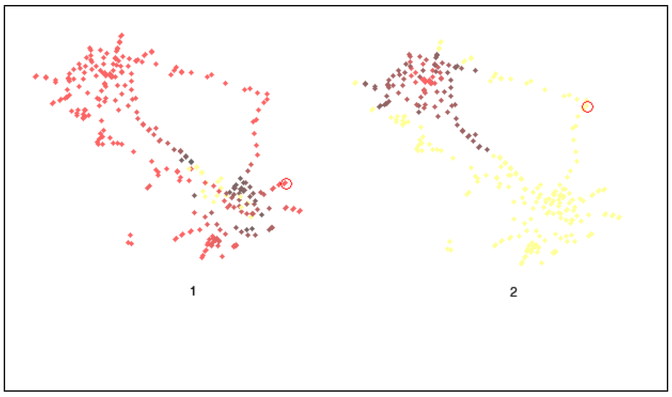
7.7.1.3.2 Further Evaluation 2 In the following section the experimental setup was the same as before except that there were 3 enemy bots trying to stop our 3 bots.

7.7.1.4 Citadel

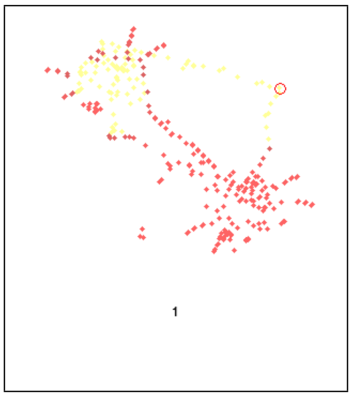
In figure 7.87 we begin to see a pattern developing. The approach model becomes dominated by the negative reinforcement. This is because the enemy bots are getting more kills so we are generating more failed capture attempts. The return model still picks out the paths taken but is much less defined. The general areas which the model ranks as good are easily picked out but they have little discrimination within them. This is also a reflection that the matches had less successful return runs. If the same route is taken several times, sometimes generating a good result and others a bad, then we don't expect strong reinforcement of this route. We also see that the middle route is taken with more frequency.

7.7.1.5 AbsoluteZero

In figure 7.88 both the approach and return models show similar characteristics and pick out similar paths in the level. One of the most interesting points about this is that the model can pick out which of the two bridges were used to cross over to the enemy side. This is preserved in the scenario where there are 3 enemy bots present. This kind of feature of the route modelling is important. If we are to use this model to guide us through the level then negative reinforcement of a path taken could lead to negative reinforcement of a bridge within this level allowing the bot to decide, sensibly, to avoid the bridge on the next run. This represents emergent behaviour at a low level of data analysis and it is not a big step to envisage how a decision procedure could be based

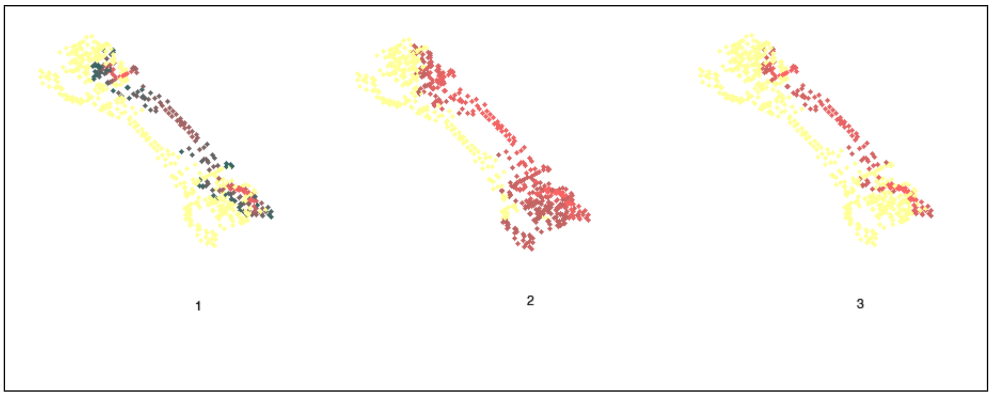


(a) Approach

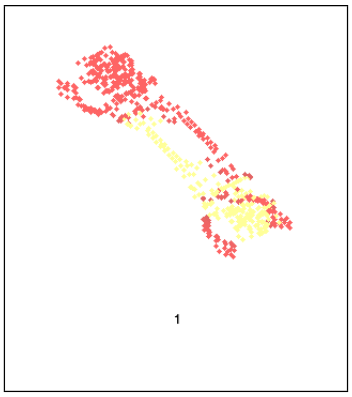


(b) Return

Figure 7.87: Further Models 2 for Citadel



(a) Approach



(b) Return

Figure 7.88: Further Models 2 for Absolute Zero

on such a system.

Engineering Box

Discounting During very early testing a model was used which had discounting to place more importance on points in the path based on intuitions about path nodes and levels of importance. For the approach model the tapering placed more importance on nodes towards the capture end of the path. For the return case it placed more importance on node towards both the capture and return ends of the path.

The discounted models didn't perform very well, generating paths which were quite non-descript. In most cases they generated an area of density over each flag point without reinforcing the particular path taken.

One possible explanation is that there was nothing to say that the tapering method chosen was correct based on the intuitions we had about path nodes. Why should nodes near the flag points be more important than those in the middle of the path? There is still the possibility of improving the model, via biassing the update rules to certain areas of the path, but more study would need to be put into the biassing function used for this to be effective.

An Alternative Kernel The Gaussian kernel function used is not the only possibility for this type of work. Other common options include triangle, quartic and Epanechnikov[61]. On top of these customised kernel designs could also be included. Probably the biggest problem is that there is no negative drop-off as you leave the area of a Gaussian kernel function. One approach to dealing with this and getting more out of the reinforcements could be to use a kernel which, after you leave the area of the Gaussian, begins to have a negative influence on nodes. This would then lead to a greater influence of each update on every node within the level. One suggestion for the kernel (in the same basic format as the standard kernel) could be:

$$f(X,Y,\sigma) = \begin{cases} G(X-Y, 2\sigma^2 I) & \text{if } \sqrt{X-Y^2} > 2\sqrt{\sigma} \\ -\sqrt{X-Y^2} & \text{otherwise} \end{cases} \tag{7.28}$$

where $G(x,H)$ is a Gaussian Kernel defined as

$$G(x,H) = \frac{1}{\sqrt{|2\pi H|}} e^{-\frac{1}{2}(x)^T H^{-1}(x)} \tag{7.29}$$

We have not tested this and so this remains future work for anyone extending this thesis.

7.8 Game-Type Specific Route Modelling - DD

The domination point model was identical to the flag approach route model with a few small changes. Firstly there is only one phase of the model as once the bot gets to the domination point there is no returning to base. There are also two different domination points so the point which was to be updated was only decided when a bot got to that point.

Engineering Box

Repeated Nodes One interesting engineering problem was how to deal with repeated nodes. Our eventual solution was to have the model only reset the path and take fresh nodes when a bot died. This was to stop the situation where a bot gets to the domination point and simply moves backwards and forwards by a node, thus reinforcing that node drastically over all other nodes.
Some other alternatives were pondered but never implemented such as strongly negatively reinforcing the areas surrounding the domination points or creating areas of no reinforcement at these points. Neither solution was particularly elegant but the chosen solution was not exactly optimal either.
This problem mirrors a general problem with this type of route modelling, that of dealing with path updates when the goals and intentions of a bot are not known in advance. A process of pre-empting has to be undertaken concerning the eventual bot's movements and specifically their likely use of the model within a larger decision process. This can be easier if the same designer is responsible for all the parts of the system but in general this is not the case with full multi-layered behavioural systems. This kind of bisector of the layers in the hierarchical architecture is also not good for the modularity of the system, creating problems for further revisions and adaptation.

7.8.1 Results

Only the results for variance 500 are presented here. This is in accordance with the results in the section on flag approach route modelling. The target function and model formulation are almost identical so we expect that the same variance setting should be equally useful for domination point approach modelling.

The model was tested over 3 levels, Sepukku Gorge, Ruination and Outrigger (spanning medium to large and large) but only the first two results are presented[8]. Two bots were used against two in-built bots. The bots used a strategy which said that one bot should go to domination point A while the other went to domination point B. When they got there they would play randomly around the area of the point.

7.8.1.1 Sepukku Gorge

The collage shown in figure 7.89(a) does not show a bias for any part of the level outside of the area around the domination point. This is largely useless for any meaningful purpose.

The collage for point B is much more interesting. To begin with a path from the top left hand corner extends down to the flag point A in response to an arrival of the bot at the domination point from a spawning position near point A. The largest influence throughout the level appears to be the negative reinforcement.

This model generates densities which are slightly too generic. This would hopefully change in response to more situated active data but this assumption cannot be guaranteed.

7.8.1.2 Ruination

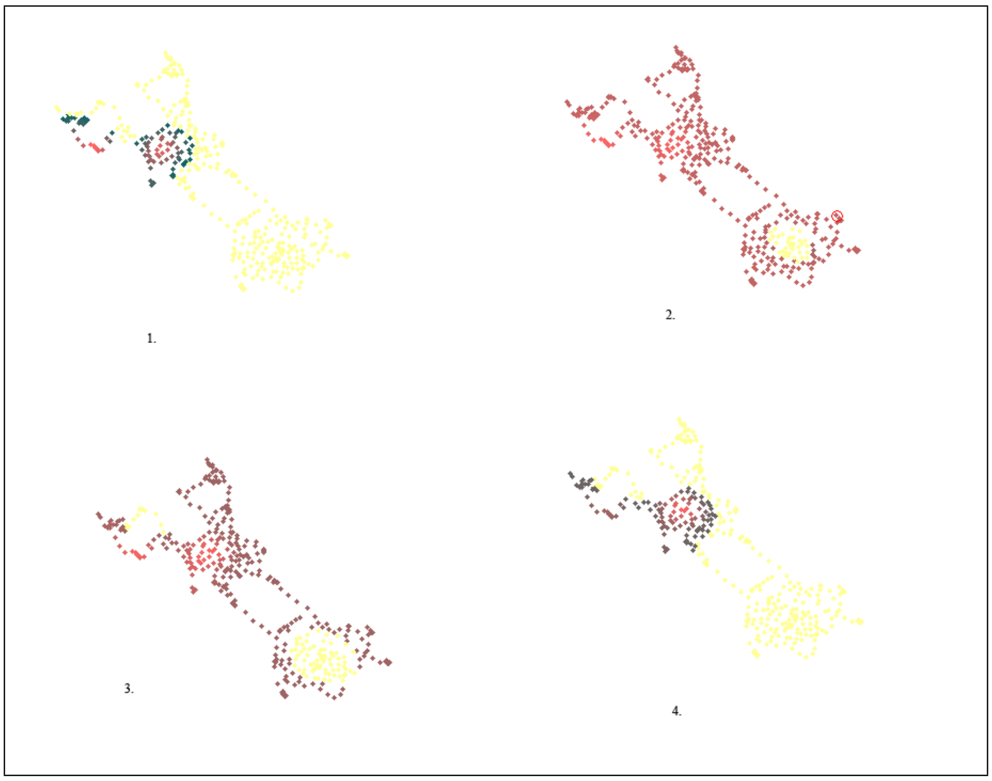
Ruination is a larger level then Sepukku Gorge and as such there is much more possibility to be specific about the actual routes. Figure 7.90(a) shows stronger definition of the paths. There are less problems with negative reinforcement of the entire level and the level of granularity of the paths is acceptable.

The routes of domination point B take a slightly longer time to develop. They develop no go areas of the level, which had low success rates for bot performance.

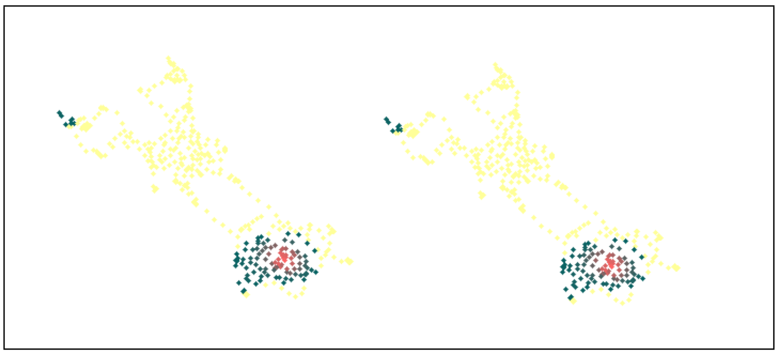
7.8.1.3 Overall Dom Approach Route Conclusions

The evaluation in this section shows little more than we already knew about the model, from the flag route modelling section, other than it working for the domination routes as well. A better evaluation of both the flag and domination approach models requires generating paths from these models to test if they improve bot performance or not. In a later section we construct a custom path finder and show how it can be used in

[8]Outrigger is in the appendix

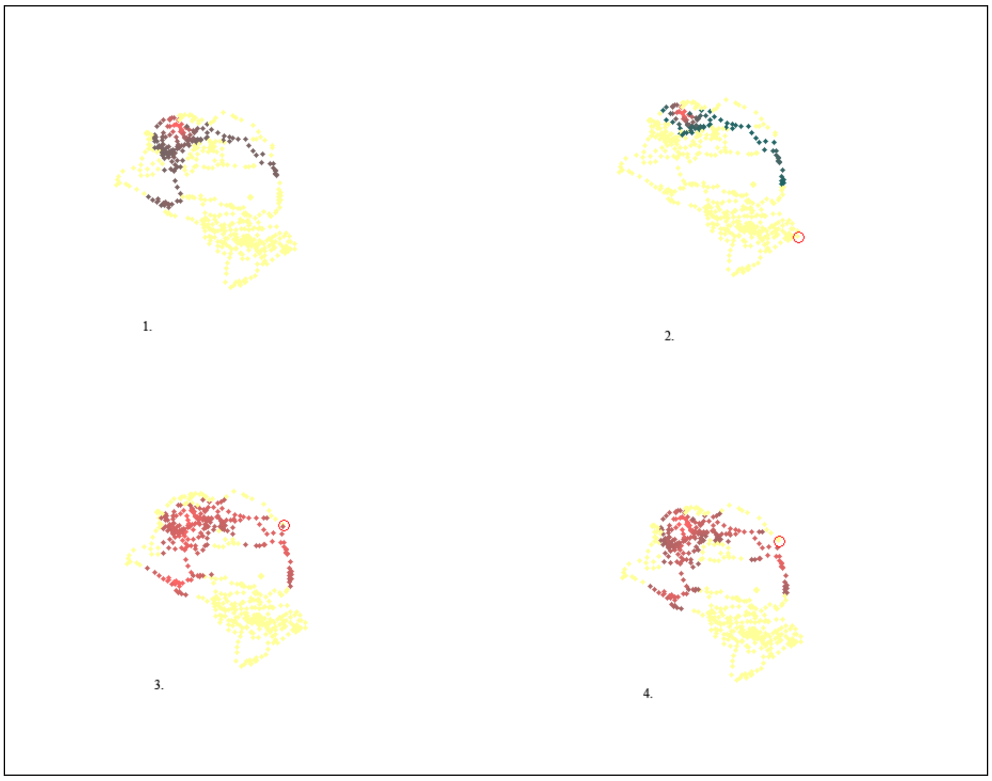


(a) Point A

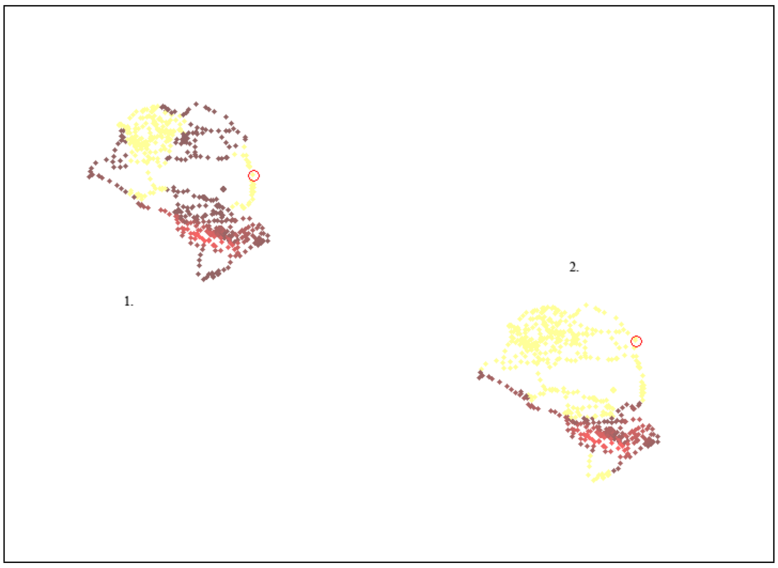


(b) Point B

Figure 7.89: Domination Point Approach Routes, Sepukku Gorge



(a) Point A



(b) Point B

Figure 7.90: Domination Point Approach Routes, Ruination

conjunction with the models. The situated evaluation of these models is contained within that section as it requires the path finder.

7.9 Flag Sighting Modelling

The bots are aware of flags in their field of view. Often when a flag is in play and not situated at home it can be difficult to determine where is the best place to look for it. Our intuition is that by incorporating our observations of the flag into a model we can use this to give better areas to begin looking for the flag. This is more important for retrieving our flag than it is for the finding the enemy flag. Generally when the enemy flag is dropped it is picked up almost immediately by an opponent and returned to the enemy base.

The model we have chosen is a parzen density estimator, with a Gaussian kernel, based on flag data observations. The model has two different estimators, one for each flag. No observations of the flag in its home state are incorporated, to avoid the biassing of modelling to the flag points. The point of the model is to give information about the flag in transit.

Engineering Box

Motivations The flag sight model's development is fairly typical of a lot of the ideas in this thesis in such much as it was born from observing game matches and spotting common patterns and possible avenues of modelling. It was obvious that when our bot's flag was taken they were struggling to retrieve it. This was mainly because they had no concept of the paths which the enemy might be inclined to take when taking the flag back to their own base. This lead to them just following the same paths regardless of how the enemy were playing.

7.9.1 Evaluation

To test this model a team of 3 bots were played against the in-built bots on three different levels. The visualisations of the densities generated were then examined in reference to the game-play in order to determine model performance.

Variance settings of 5 and 10 were used[9] over three levels. These settings were chosen because in previous sections, using the parzen density estimator to measure

[9]with rescaling these equate to 500 and 1000 in terms of the estimator used for the flag path route

game based observations, they proved most effective.

7.9.1.1 Citadel

In the own flag test, of the variance 5 model, what is being developed is a model of the enemy escape route from our base after a flag capture. The strategy used to play the game sees our flag defender attempt to follow the flag capturer while they remain in vision, keeping track of the last known sighting. Therefore they chase any visible enemy holding the flag. The results are reasonable but the densities are slightly too thin, narrow and focused. Once again this comes down to not creating too high peaks in the space of the density function. The information gained in this example is not doing much more than memorising the points along the path taken.

The enemy flag model displays similar traits to the own flag model but it is interesting to note that the observations of the enemy flag are made by members of our team other than the flag carrier. Thus the model will be very much determined by the movements of the supporting team members.

The variance 10 models are better and give a more spread influence map. The paths are not as obviously defined but the influence does not extend beyond sensible boundaries, which would be detrimental.

7.9.1.2 Geothermal

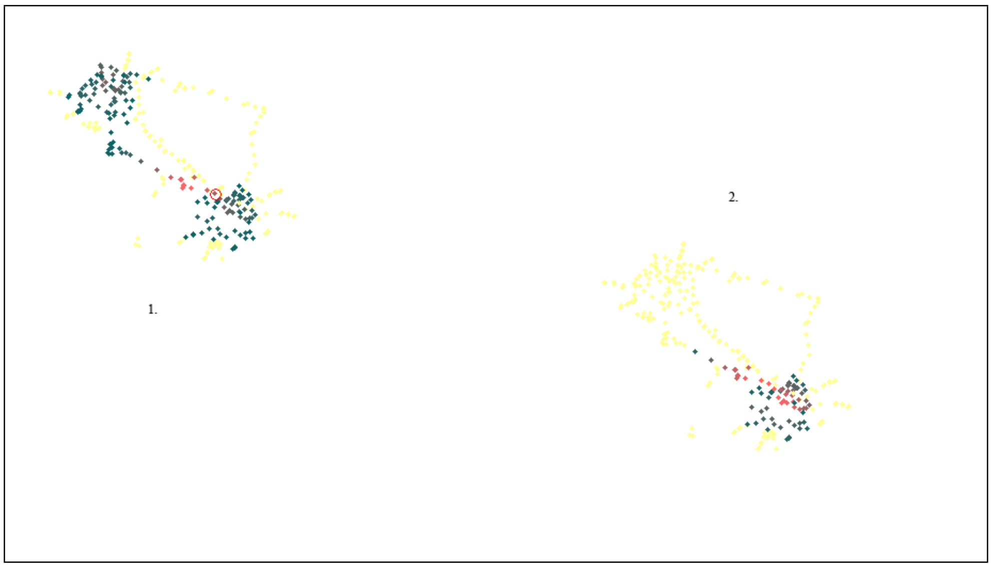
Geothermal is a slightly larger level but follows the CTF format of having two large base areas with a section linking them.

Both the own and enemy models in the variance 5 section have problems with narrow influence maps. The problem is less exaggerated than in the Citadel level but it is still visible. The enemy model develops a path which cuts through the level and clearly shows an enemy attack route bias via a half figure of 8 pattern.

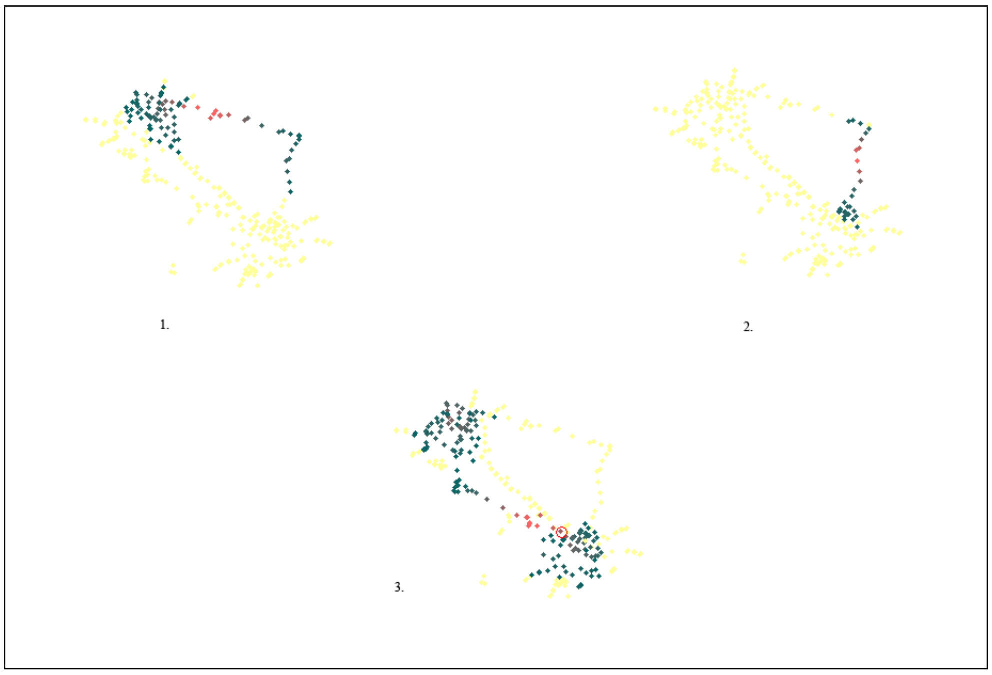
The variance 10 visualisations display this path much more accurately with a wider spread of influence. The path taken is more succinctly picked out in red while areas around it are given a gradient of red which is useful. Steep ridges in the decision surface are avoided.

7.9.1.3 MoonDragon

The bigger level highlights the problems with the variance 5 model. Islands of influence are very common as the effect of sightings has not setup a large enough bias

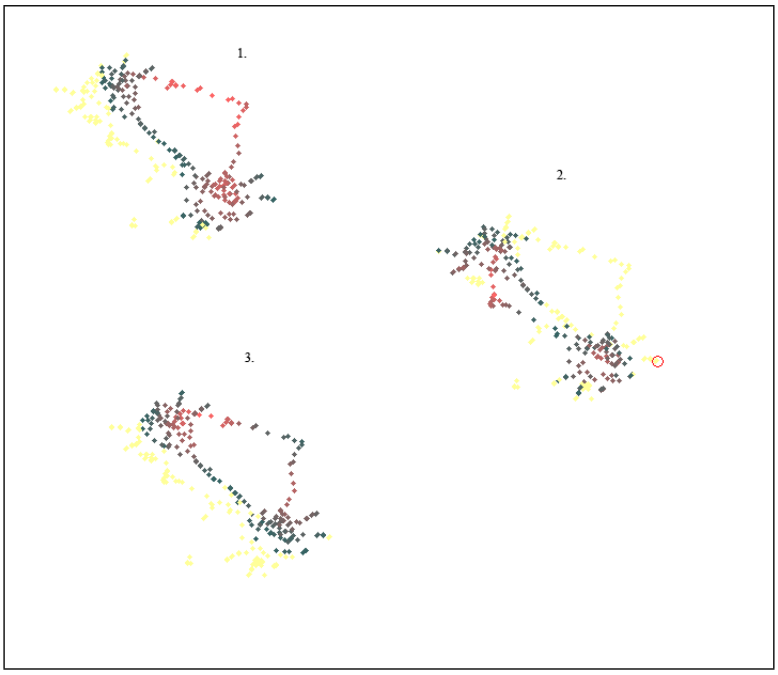


(a) Own Flag

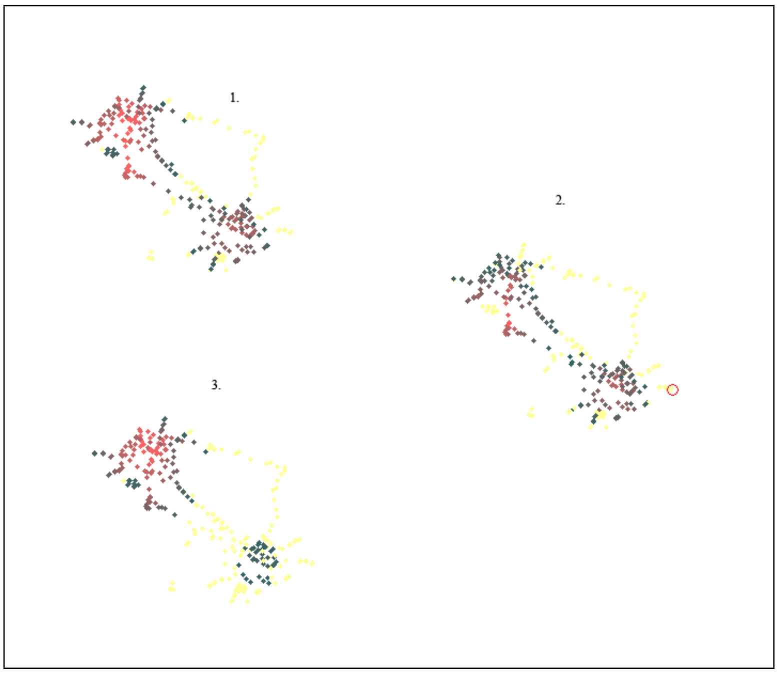


(b) Enemy Flag

Figure 7.91: Flag Sighting Models, Citadel, Variance 5

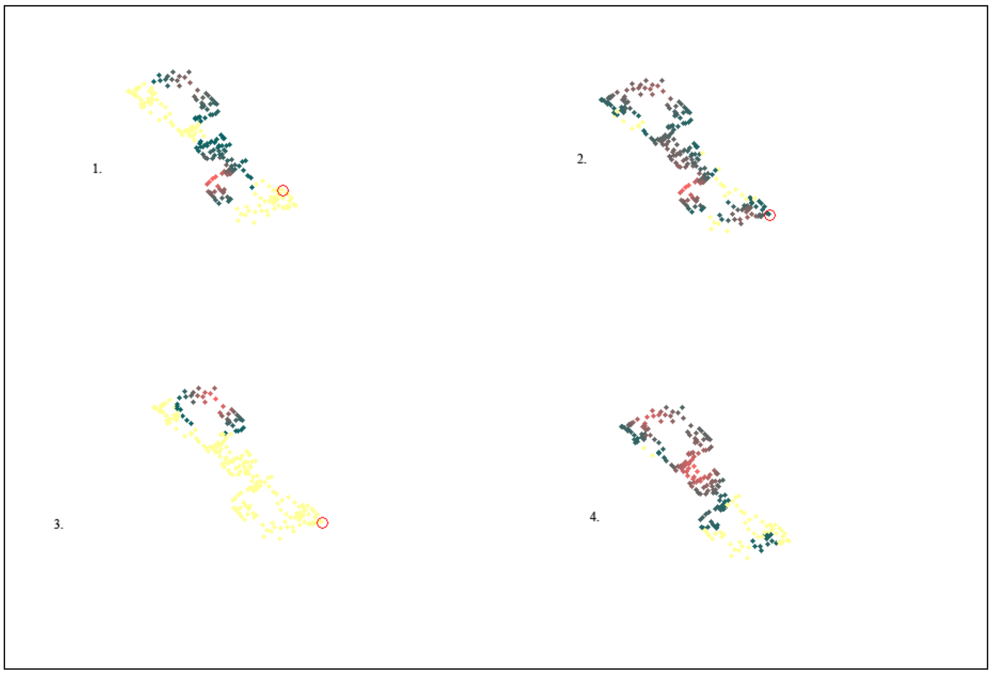


(a) Own Flag

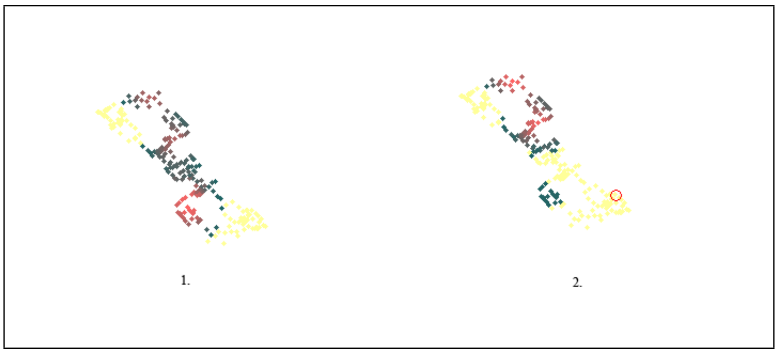


(b) Enemy Flag

Figure 7.92: Flag Sighting Models, Citadel, Variance 10

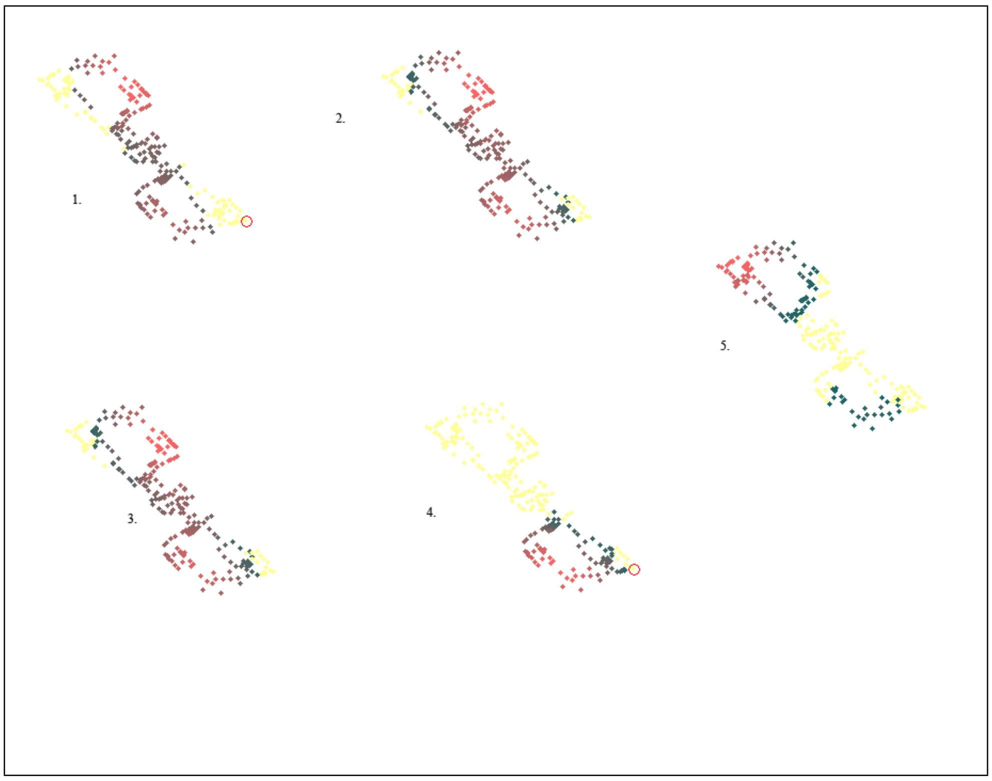


(a) Own Flag

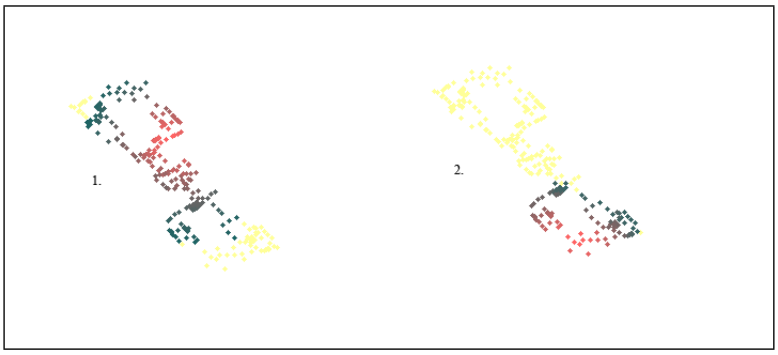


(b) Enemy Flag

Figure 7.93: Flag Sighting Models, Geothermal, Variance 5

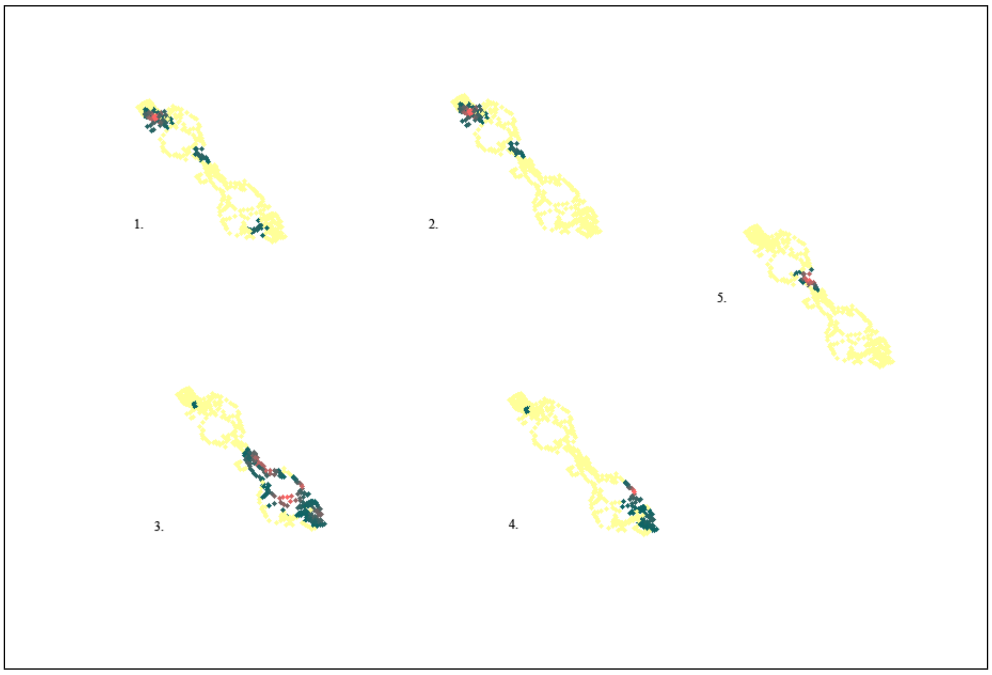


(a) Own Flag

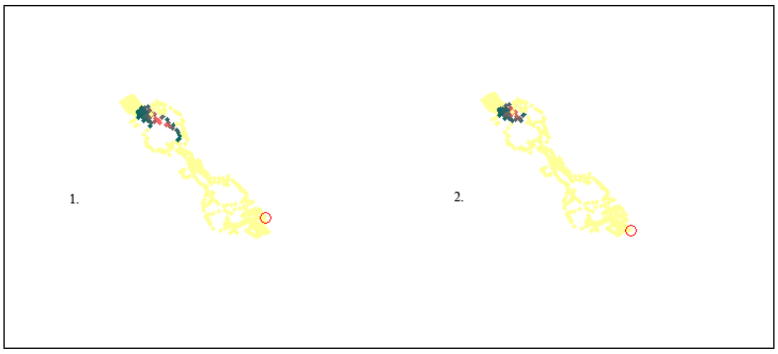


(b) Enemy Flag

Figure 7.94: Flag Sighting Models, Geothermal, Variance 10



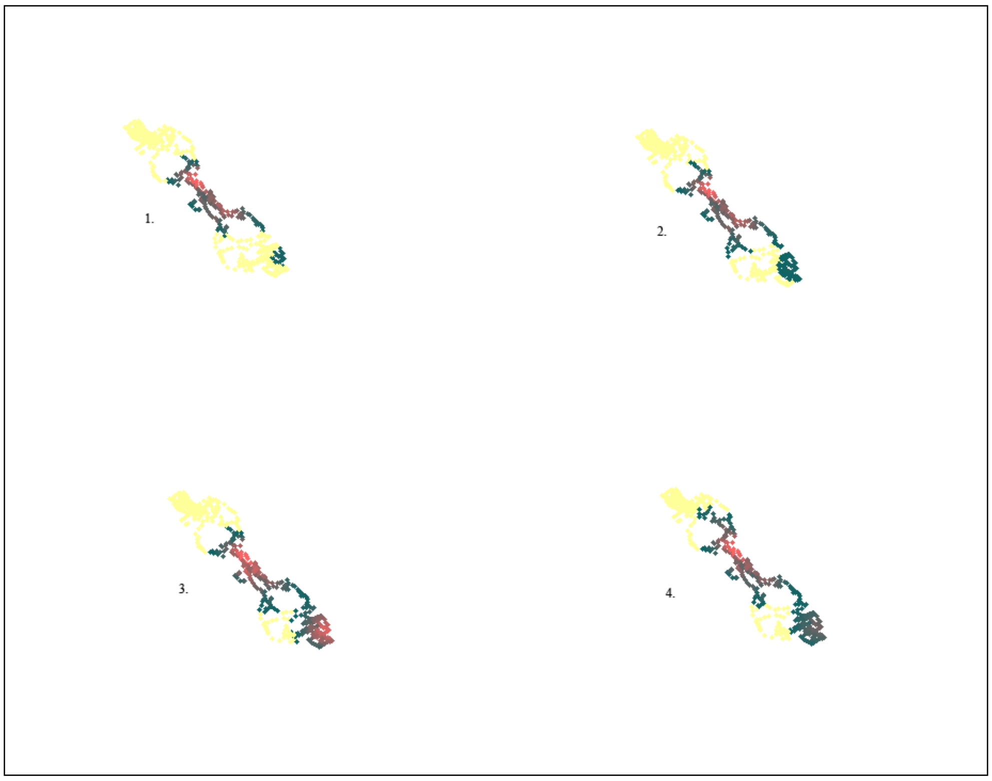
(a) Own Flag



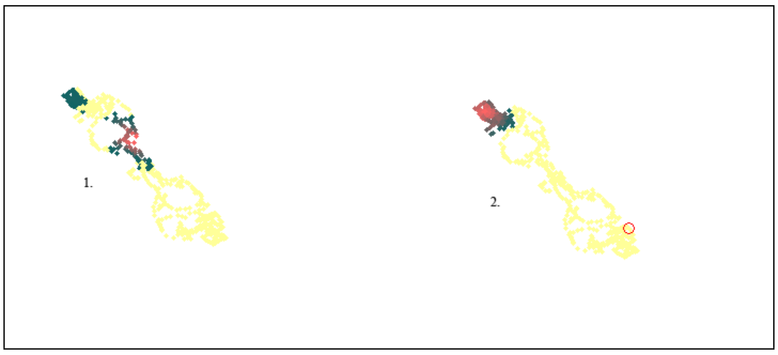
(b) Enemy Flag

Figure 7.95: Flag Sighting Models, Moon Dragon, Variance 5

to give a solid path or surface throughout the level. The larger variability of node placement on Moon Dragon, as shown via the PCA magnitude calculations in the level modelling section, means a larger modelling variance is needed to correctly model the flag sightings within the level.



(a) Own Flag



(b) Enemy Flag

Figure 7.96: Flag Sighting Models, Moon Dragon, Variance 10

Interestingly the enemy flag model has no real pathway of any length, mimicking the fact that the bots don't manage to carry the enemy flag very far, and thus reflecting the bot's poor performance on this level.

In contrast to the islands of the variance 5 models, the variance 10 models show a path through the bulk of the level in the own flag case. The main concentration is on the central portion of the level but there are signs of this extending further out.

7.9.2 Conclusions

From the results we can see that the performance of variance 10 model is much better than that of variance 5. The area of influence is much greater and there is less chance of creating small areas of high density, which is good, as these create peaks in the density surface which are not very informative for the surrounding area.

The most interesting observation about the model is its ability to suggest the opponent's escape path from our base when they have the flag. This is useful as it allows us to counter-act this behaviour in advance with the whole team converging on the area.

Engineering Box

An Alternative Model One alternative to the parzen density estimator in this situation is a model similar to the area correlation model. Observations of the flag presence could be tallied for each cluster in the level and then used to give a utility value to points in these clusters. The reason this wasn't explored is that the density estimation method gives a generative angle into the problem which can be used to give a smoother surface for a range of decision procedures to manoeuvre in. This said, it is still a viable alternative method.

7.10 Path Finding

In this section we examine the idea of path-finding. This may seem strange in a chapter concerning modelling and machine learning mechanisms but we show how the dynamic data from the models discussed in this chapter can be integrated to create an intelligent reactive path-finder. In this way it represents not only a path-finder but a continuous decision mechanism for controlling conclusions based on dynamic data.

7.10.1 Current Approaches to Path-Finding

Path-finding in general is a well established practice and is central to NPC performance in any environment driven video game. The ability to negotiate levels and obstacles is one of the most basic of skills required for an intelligent situated agent.

> "Undoubtedly, the way units move is important for winning a game, because an intelligent moving behaviour might reduce casualties of own units"[28]

Current approaches to path-finding tend to fall into one of two categories, static and dynamic, as detailed below.

7.10.1.1 Static Path-Finding

Static path-finding (by our definition) is path-finding which does not base its algorithm any real time data from the game other than the agent's current location. Thus methods for performing static path-finding are mostly optimised for path length only.

The most common method for path finding is the A* algorithm[31]. This is a well known path-finding algorithm for game AI and is a good starting point for any work in this type of area.

> " A* and IDA* (and their variants) are the algorithms of choice for single-agent optimisation search problems[14]"

The main steps in A* are as follows:

1. Initialise Open and Closed list of nodes to be empty lists
2. Define a start and goal position
3. Find all nodes directly reachable from the start position and add them to the open list
4. Maintain open list
5. Maintain closed list
6. Assign these nodes a G-Score, H-Score and F-Score[10]
7. Pick the node with lowest F-Score as new target node
8. Add parent node to closed list
9. repeat process from new node

[10]Where F = G + H

7.10.1.2 Dynamic Path-Finding

In contrast to static path-finding, dynamic path finding utilises in-game data, of a dynamic nature, to optimise paths over selective goal criteria. This is typically more powerful for scenarios where the dynamic data is rich, but also trades off against the optimality of path length.

It is also true that less work has been performed on dynamic path-finding in comparison to static path-finding as the following quote summarises:

> "Less attention has been paid to the problem of incorporating rich global conditions that require more than simply reaching a specific location."[118]

One solution to achieving dynamic path-finding is to retrofit a dynamic mechanism to the existing A* algorithm by modifying certain parameters within the mechanism[4]. This is sometimes called annoted A* or AA*[44]. The main idea is to create influence maps which we believe to represent desirable target parameters. We then use these maps to change the utility of any given node in the search space so that calculations are not based purely on distance but rather a combination of distance and the influence maps.

Modifying A* is a nice solution because it is very clean and simple. Other more complex solutions do exist which offer more performance but they are typically much more prescribed and domain specific. One example is Wang et al's system [118] for performing multi-unit tactical path planning. Rather than being a path-finder it is more an integrated planner with path-finding execution capabilities. The problem with it is that the specific actions available are very domain specific and the exact domain used is not widely similar to ours.

7.10.2 The UT Path-Finder

The current UT engine provides us with paths to any location, generated on a static basis. The engine does not use any in-game information to create these paths, performing path-finding based on bot skill[11] while optimising for length of path. It is a clear example of a static path-finder.

[11]bot skill is a parameter set at level startup

7.10.3 Our Approach

Our approach was to adopt a dynamic method based on the aforementioned modification of the A* algorithm to optimise over dynamic data as well as path length.

In section 5.3 we stated that rich information may be contained within the dynamic modelling data which can be obtained from some simple machine learning methods. To test this idea a system was designed which allowed paths to be generated which optimised length, as per those from the unreal engine, but also took into account some of the data from modelling. We also show that all of the information detailed in this section can be used to create paths which are optimised for certain sets of goals which the bot may have.

Engineering Box

Current State This functionality is not at the present time available within the Unreal engine or the Gamebots modification. This is because the path finding call is something which EPIC implemented in native-C and as such there is no source for this part of the game engine which is publicly available. The path finding presented in this section has been implemented within the Java/Prolog part of the system.

7.10.4 Intuitions Behind this Concept

Our key assumption is that the modelling conclusions are rich enough to allow it to be of more importance for finding better paths than the information regarding path length.

They needs to be better rather than just good because the path finding algorithm in UT is likely to be better optimised over length than our vanilla version. Thus it will be the dynamic data which will tip the scales in our favour.

Play on larger levels is likely to be more effected by this technique than on smaller levels, where the level is too small to avoid confrontation on any paths.

7.10.5 Methodology

Our starting point for path-finding was the A* algorithm as discussed in the existing approaches to static path-finding.

In our case the G-Score was the distance from the start node, the H-Score was the distance to the target node and the F-Score is the sum of the two. Maintaining the open and closed lists involves removing duplicates from the lists and also changing adjacent nodes to be the version in the open list with the smallest G-Score. This is to ensure that we always pick the shortest path.

Engineering Box

Adjacency To use this we first had to define adjacency within our 3-dimensional node structure. Adjacency was defined as being within a certain distance from the current node. This meant that we were not limiting ourselves to one particular direction of movement.

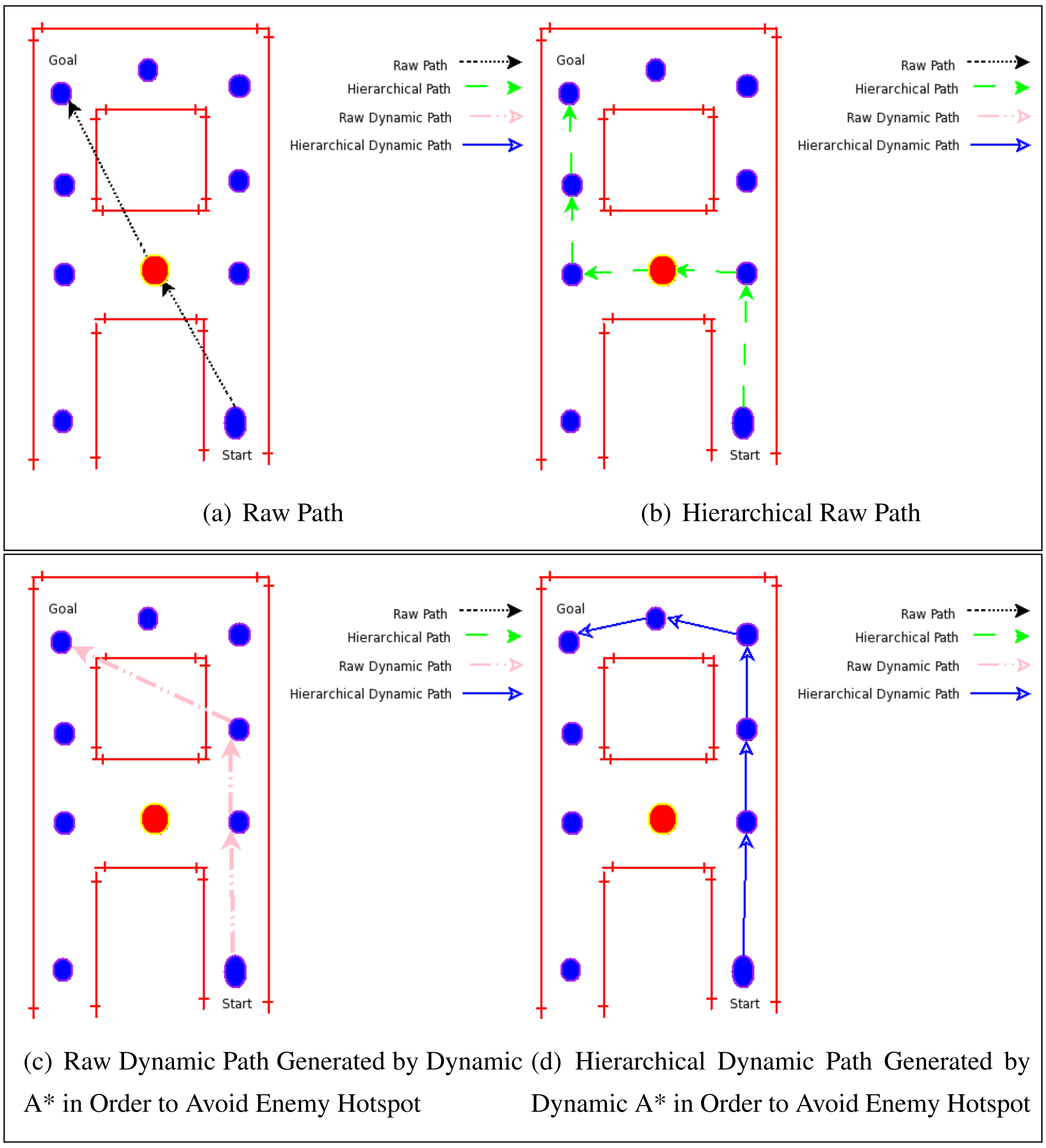
It became apparent that due to not having any information about the structure of the level this simple path finder was not going to generate directly executable paths. The problem, as illustrated in figure 7.97(a), is that without knowing which elements of the level are reachable from any location it is impossible to generate paths which factored this into the act of path finding. Many times there would be obstacles in the way of the path generated which meant it could not be executed. This is obvious when you consider that, without any other information about obstacles, the paths generated would be as close as possible to a straight line to the goal location.

The solution to the obstacle problem of figure 7.97(a) was to create a hierarchical path finding system using the A* path finder to generate overall paths which were not executable. We then passed these to the UT path finder so that a new path was generated for getting from every point on the overall path to the next point.

```
twoLevelPlan(Location,Goal){
        HighLevelPath = aStar(Location,Goal)
        for each move in HighLevelPath {
                LowLevelPath[MoveNumber] =
StandardPathFind(HighLevelPath[MoveNumber])
        }
        return LowLevelPath}
```

This idea is similar in concept to both HPA* and PRA* [16] in which graphs of searchable nodes are separated into abstracted areas which are searched through by a high level path-finder.

The diagrams in figure 7.97 represent how the path-finder is expected to perform on well behaved data. The paths generated from this system resembled those in figure 7.97(b). We already stated that this is not much use because it is going to generate straight line paths which are worse than those generated by the in-game path finding system.



(a) Raw Path

(b) Hierarchical Raw Path

(c) Raw Dynamic Path Generated by Dynamic A* in Order to Avoid Enemy Hotspot

(d) Hierarchical Dynamic Path Generated by Dynamic A* in Order to Avoid Enemy Hotspot

Figure 7.97: Theoretical Path Finder Diagrams

To make this system dynamic we altered the algorithm slightly to take into account a node utility value. This could be pre-determined based on any relevant data. When considering the F-Score nodes for selection the algorithm would be set to either add or subtract the node utility value from the F-Score. The result is that the algorithm no longer optimised solely for length of path, instead optimising over path length and either positive or negative node utility.

A scaling factor allowed careful tapering of the importance that the path finding placed on the node utility values by altering the scale of the utility in comparison to the scale of the path node values. We have called this scaling multiplier the fear/attraction factor.

Because the algorithm is no longer optimising for distance the theoretical results of A* path-finding no longer apply. In our case though, if we base the node utility values on our models from in game play, optimality takes a new meaning as our optimisation target is the survival and successfulness of the bot.

This parameter was tested over a variety of different values. The G-Score value for the A* path finding was then set to be the distance travelled so far in the paths and the HScore was set to be the euclidean distance from the goal location plus the additional node utility. In this way nodes which have a higher likelihood of enemy presence were given a higher F-score and thus were less likely to be selected for paths. The easiest way to visualise this situation is by considering the input as a 4 dimensional vector rather than 3.

Engineering Box

Scaling Factor The scaling factor is important here because if the node utility is on the same scale as the other 3 components then it will be considered as important as one of them. If the node utility grows larger it dominates the euclidean co-ordinates in the F-Score generation procedure.

Figure 7.97(c) shows how the path would then be generated to avoid the *enemy hotspot* in the central area of the environment. Figure 7.97(d) shows how this would be further refined by the hierarchical path finding system to give an executable path.

7.10.6 Evaluation Data

The first diagram shows the path finder working on an implementation of the theoretical problem shown in figure 7.97(d). It shows the path for different values of the fear factor. For the sake of demonstrating how the scaling relates to the actual distance between the nodes we also include here a list of the nodes used in this example alongside their positions (Co-ordinates are standard euclidean):

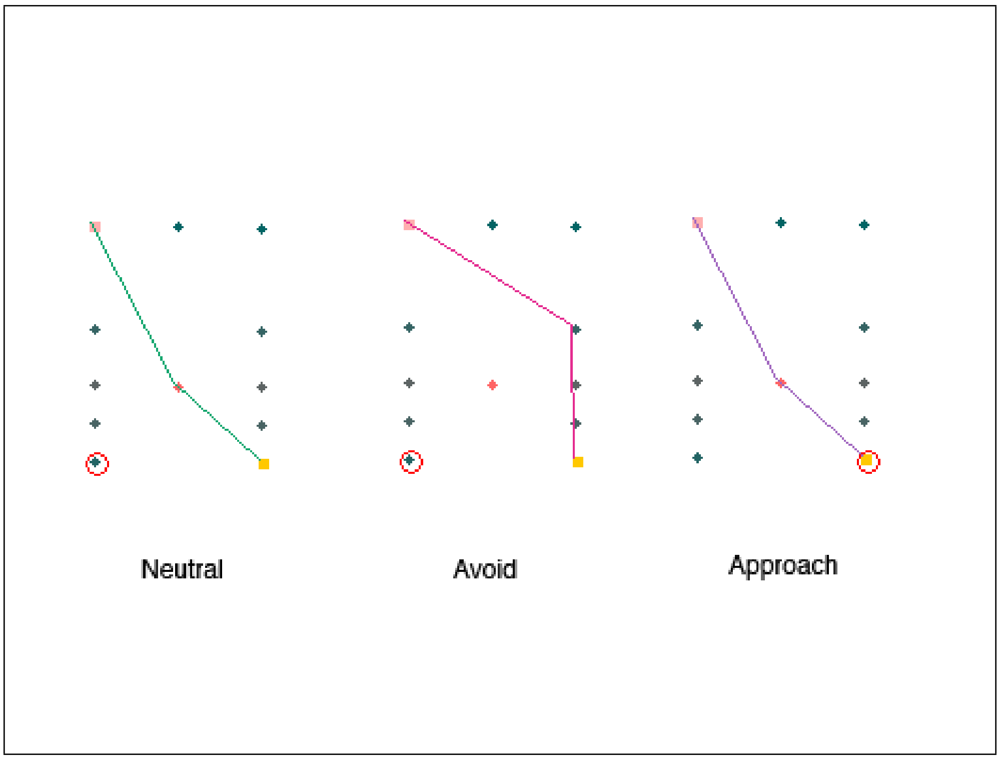


Figure 7.98: Path Generated by Dynamic A* in Order to Avoid Enemy Hotspot

The start of the path is marked in yellow and the end in pink. A Gaussian of variance 1 was placed over the hotspot and each node was given a utility value in accordance with the probability density of the Gaussian. In the first experiment shown in Figure 7.98 the fear factor was set to 1000. The max distance parameter was set to 5. This parameter controls the distance which the path finder uses to define its adjacency condition. If nodes were within a distance of 5 then the path finder would deem them adjacent to the current node. The average distance of any node to its nearest neighbour in this example is 1.416 .

The diagram shows that our high level path finder performs as expected. The approach and avoid diagrams show the path when set to favour nodes of higher utility or lower utility. The neutral result is the path finder applied without the node utilities. Interestingly when we lower the fear factor to 100 the path generated by the avoid case remains unchanged. We do, however, generate different paths if we lower the maximum distance value from 5 to 3.

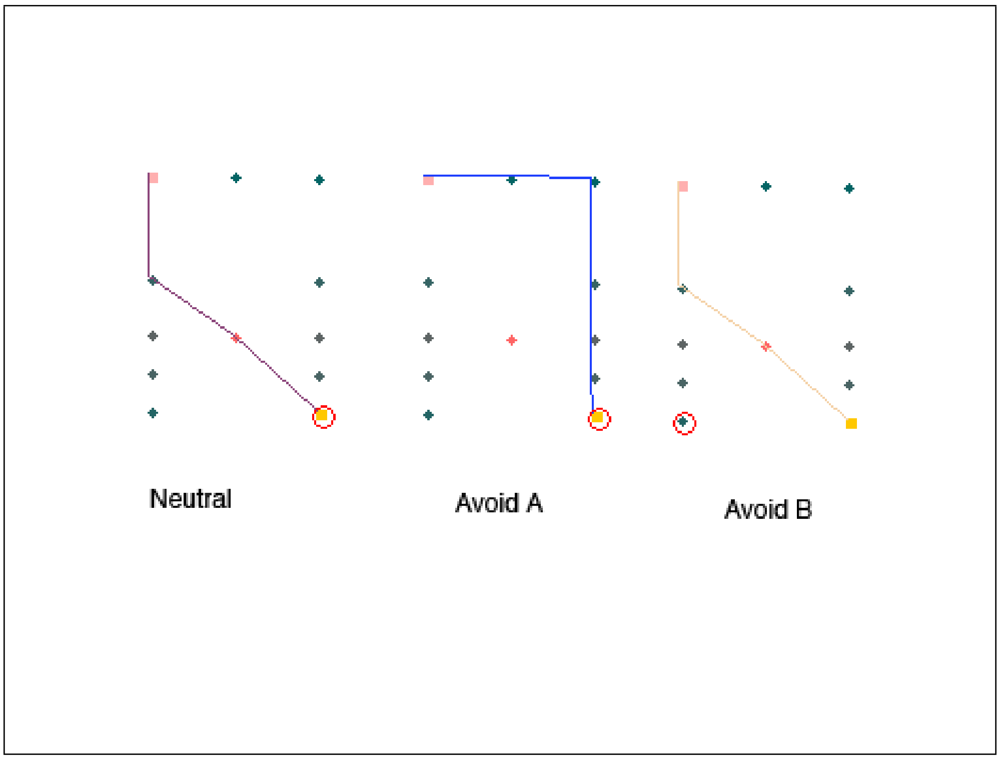


Figure 7.99: Path Generated by Dynamic A* in Order to Avoid Enemy Hotspot, Lower Adjacency Condition

Point	Test Points		
	X	Y	Z
A	2.5	3.5	1
B	2.5	4.5	1
C	2.5	5.5	1
D	2.5	7	1
E	2.5	9.7	1
F	4.5	5.5	1
G	4.5	9.7	1
H	6.5	3.5	1
I	6.5	4.5	1
J	6.5	5.5	1
K	6.5	7	1
L	6.5	9.7	1

Figure 7.100: Neural Net Parameter Settings

In Figure 7.99 the neutral path has to take a few extra steps to get to its destination due to the diagonal from the hotspot to the goal being unavailable with the smaller adjacency condition. We also see that changing the fear factor has a larger effect in this case. For instance when we set the value to 100 as in Avoid A the path goes straight through the hotspot. If we set the value to 1000 as in Avoid B it veers more strongly away from the hotspot. The explanation for this is simple, in the case where a short cut across the top right hand corner was available the algorithm could find a short path going both through the hotspot and around it. As such both paths were virtually equal length and the node utility was the main factor affecting the choice between the two. Removing this branch, with lower max distance condition, effectively increases the length of the path that is needed to be taken to avoid the hotspot, so in turn we needed to increase the fear factor to compensate for this.

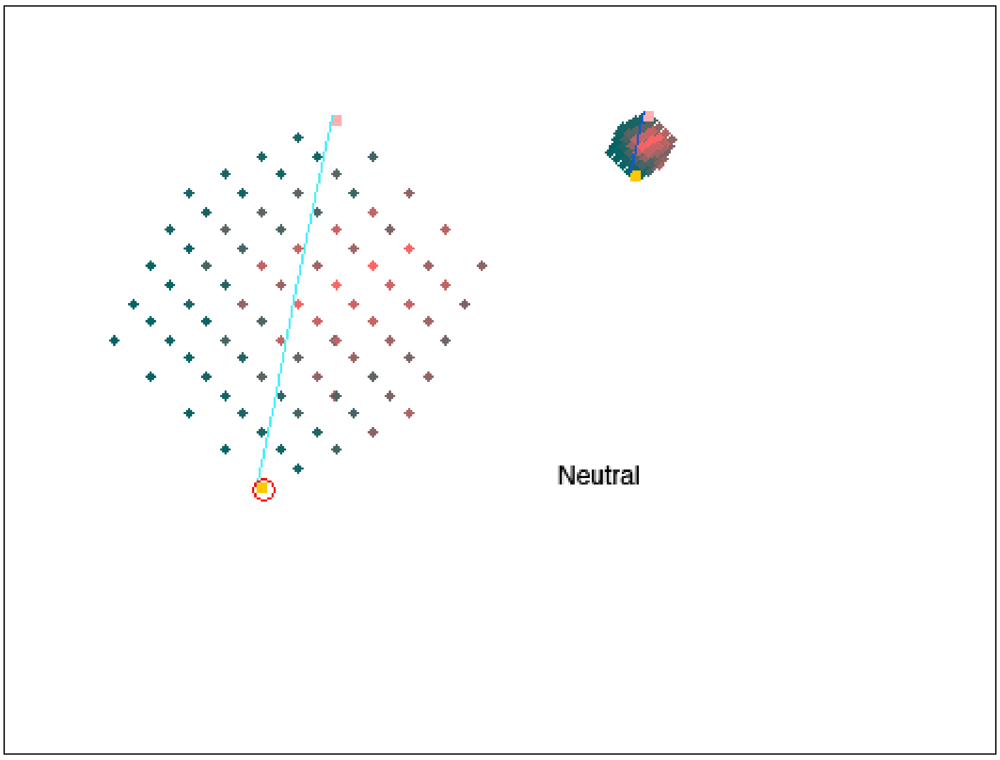


Figure 7.101: Path Generated by A* in Order to Navigate a Cube

Engineering Box

Small Scale Changes The example in figure 7.99 is affected by these small changes as there is no other pathways around the hotspot and as such this example is slightly unrealistic. This said, it highlights the considerations which must be taken when setting up the path finder and taking into account the scale of the input data.

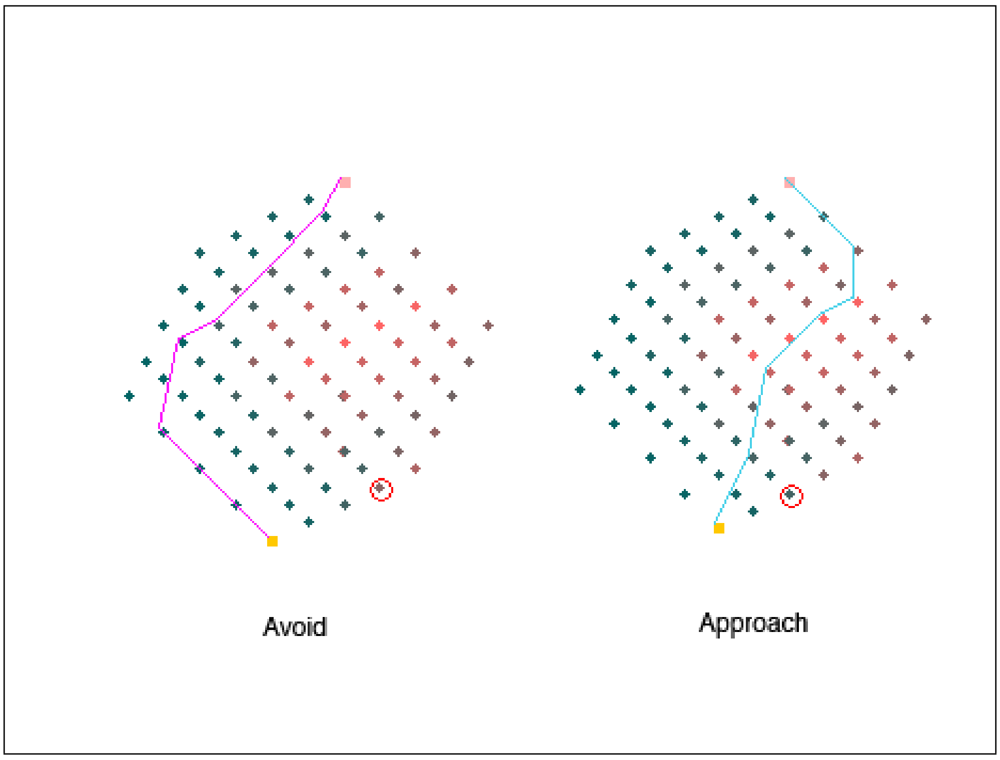


Figure 7.102: Path Generated by Dynamic A* in Order to Navigate Gaussians in a Cube

We now show how the path finder performed on a larger example involving a cube. The first cube was size 5 meaning that the cube had 125 nodes. The average distance to the nearest node was 1. Three 3D gaussians were placed in the cube area, each of variance 1, to create the node utility values. The maximum distance was set to 2 and the fear factor was 1000.

In Figure 7.101 we show two views on the normal path generated by vanilla A* through the cube, to show the colouration of the Gaussians. Figure 7.102 shows the paths taken by the avoid and approach modes. Both reflect what we might expect with the *avoid* path veering to the left to avoid the rightmost Gaussian mass and the *approach* doing the opposite. It is interesting that the dual optimisation of distance and avoidance is more obvious in this example as both paths have traded off the two goals against each other.

Engineering Box

Efficiency considerations As we are replacing the in-built path finder, which runs in almost real-time, we most consider the run-time of our path finder. In some larger cases, e.g. a cube of 1000 points, we found that the path finder took several seconds to run. In order to improve performance the option to only use a fraction of the level nodes was added. The nodes removed were randomly chosen but with partial ordering to ensure that the removal process was at least well separated.

The result was that, in general, paths could be generated, which were not substantially different from the paths generated with all the nodes, if the node count was between 100 and 300. This means that for levels of about 1000 nodes usually taking approximately 1/3 is acceptable. In these cases, however, the acceptable maximum distance also had to be set slightly higher to compensate for the fact that less nodes were being considered, thus increasing the average distance between nodes.

The idea of not searching all the nodes in a graph is not a new one as the following quote demonstrates:

> "A very effective method for the efficient computation of path planning solutions is to make the original problem more tractable by creating and searching within a smaller approximate abstract space"[44]

7.10.7 Level Based Situated Evaluation

We now show how the path finder performs on some real examples taken from levels in the game. Various different node utility calculations are presented to show how the paths generated can be altered to reflect the particular type of bias we wish the bot to have.

Engineering Box

Normalisation In order to get the path finder to consider data from all of of our models we needed to find a viable way to combine their outputs into a single value which could be used to give the nodes an overall utility. The idea was to take every model which could rate individual nodes and then use this raw value as a starting point. The outputs from each of the models was likely to be in a different range as they all used slightly different numerical methods. Therefore if we wished the models to be of equal importance we had to normalise these values to a common scale. Even if we do wish to bias towards a particular model this is easier if the starting point is normalised. The following ratio was used to normalise the values before they were combined:

$$Normalise(X) = \frac{X - Min}{Max - Min} \tag{7.30}$$

Lift and Jump Spots On top of this we also had the opportunity to deal with some issues, which have been present throughout our work with UT and particularly the gamebots protocol, regarding the use of lifts and jump spots. The protocol doesn't relay any information regarding when the lifts reach their destination. Therefore any attempt to get the bots to deal with the lifts so far has been to use a timer within the bot which handles the amount of time spent on lifts. A better solution is give nodes around lift nodes a very negative utility. This discourages the high level path finder from taking these routes but still allows the low level path finder to take them if no other routes are available. The approach is to place a parzen density estimator over the level, with the lift and jump spot nodes as its data store. Every other node in the level is then given a utility based on the output of this parzen density estimator.
One might be inclined to ask why the weapon model outputs were not given a similar system. The reason is that we want the lift nodes to be avoided but we know that because we are using a high level path finder the likeliness is that if we simply gave the lift and jump spots a very high *bad* utility this would lead to the path finder narrowly avoiding them. When this is passed to the low level path finder this is likely to still result in paths which use the lifts. Conversely when we use the weapon nodes we want them to be given a utility which will either steer the path finder through them or not depending on the weapon model. Therefore reinforcing the area around them would not give good performance because there is no reason that a path moving just past a rocket launcher is any better than one which avoids it entirely.

7.10.7.1 Citadel

The first level chosen was citadel. The data was taken from a capture the flag game of 3 versus 3 with our bots using the following LCC strategy[40]:-

```
Strategy 2.8

%% base_defender , makes sure they don't get our flag to begin with

a(base_defender,Id)::sawTheFlag(L) => a(base_defender,Id)
<-- visibleOwnFlag(L) and strafeAttempt(L,L) and enemyHasFlag(true)

a(base_defender,Id)::strafeAttempt(L,L) and enemyHasFlag(true)
<-- sawTheFlag(L) <= a(base_defender,Id)

a(base_defender,Id)::null <-- visiblePlayer(Location) and
strafeAttempt(Location,Location) and enemyHasFlag(false)

a(base_defender,Id)::null <-- currentWeapon(W) and
prologConstraint(W = assault_rifle) and movementAttempt(nearest_weapon_pickup)
and enemyHasFlag(false)

a(base_defender,Id)::null <-- movementAttempt(localised_play(enemy_flag_point))
and enemyHasFlag(true)

a(base_defender,Id)::null <-- movementAttempt(localised_play(own_flag_point))

%% enemy defender , comes into play when they have our flag

a(enemy_defender,Id)::null <-- changeToRole(flag_hunter) and enemyHasFlag(false)

a(enemy_defender,Id)::null <-- changeToRole(flag_carrier) and hasFlag(true)

a(enemy_defender,Id)::sawTheFlag(L) => a(enemy_defender,D)
<-- visibleOwnFlag(L) and strafeAttempt(L,L)

a(enemy_defender,Id)::strafeAttempt(L,L) <-- sawTheFlag(L) <=
a(enemy_defender,D)

a(enemy_defender,Id)::null <-- visiblePlayer(L) and strafeAttempt(L,L)

a(enemy_defender,Id)::null <--
movementAttempt(localised_play(enemy_flag_point))


%% flag_hunter , goes after the flag


a(flag_hunter,Id)::null <-- changeToRole(flag_carrier) and hasFlag(true)

a(flag_hunter,Id)::null <-- changeToRole(enemy_defender)
and enemyHasFlag(true)
```

```
a(flag_hunter,Id)::canSeeFlag(Location) => a(flag_hunter,F)
<-- visibleEnemyFlag(Location) and strafeAttempt(Location,Location)

a(flag_hunter,Id)::strafeAttempt(Location,Location)
<-- canSeeFlag(Location) <= a(flag_hunter,F)

a(flag_hunter,Id)::movementAttempt(otherBot(F)) <-- gotTheFlag
<= a(flag_carrier,F) a(flag_hunter,Id)::null
<-- movementAttempt(enemy_flag_point)


%% flag carrier , doesn't care about defending

a(flag_carrier,Id)::null <-- changeToRole(flag_hunter) and hasFlag(false)

a(flag_carrier,Id)::gotTheFlag => a(flag_hunter,F)
<-- movementAttempt(own_flag_point) and enemyHasFlag(false)

a(flag_carrier,Id)::gotTheFlag => a(flag_hunter,F) <--
movementAttempt(localised_play(own_flag_point)) and enemyHasFlag(true)

a(flag_carrier,Id)::gotTheFlag => a(flag_hunter,F) <-- visiblePlayer(Location)
and strafeAttempt(Location,Location) and enemyHasFlag(true)
```

The strategy has 4 roles, `base_defender`, `enemy_defender`, `flag_hunter` and `flag_carrier`.

The `base_defender` defends our base against attackers. They get the nearest weapon and then patrol the base area responding only to calls from other team members who have seen the flag.

The `enemy_defender`'s main role is to move to the enemy flag point when they have out flag. The idea behind this is that if we can defend their flag base then they cannot return our flag to it. These players change back to a `flag_hunter` when the flag is returned or dropped.

The `flag_hunter` bots try to obtain he flag from the enemy's flag base. When they get it they become `flag_carrier` bots.

The `flag_carrier` bots try to bring the flag back to our base to score a point.

Figure 7.103 shows the path finder generating a path from our base to the enemy base set to optimise for nodes of high utility. The only model factored into the node utility is the flag approach route model. Here the fear factor is set at 1000, maximum distance is 2 and 1/2 of the level nodes were used.

Figure 7.104 shows the paths generated for both the avoid and approach modes. The avoid path is shown in yellow while the approach path is shown in purple. It is

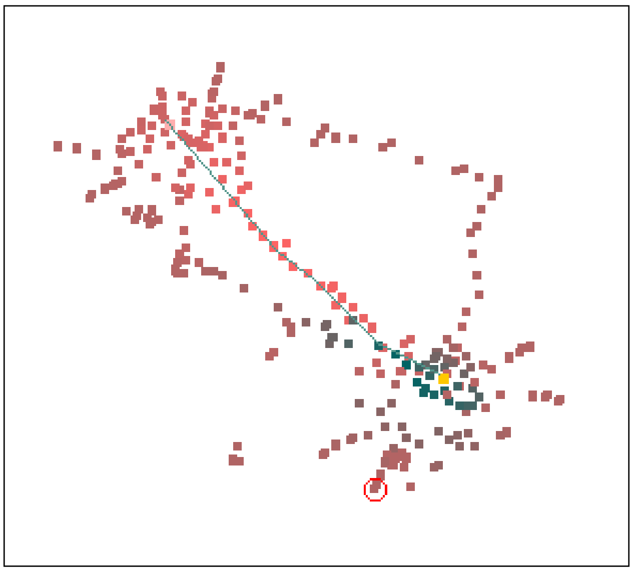


Figure 7.103: Citadel Path generated using only the flag route model

clear that the avoid path is significantly different to the approach path showing how the generated path is responsive to the modelling.

In figure 7.106 the maximum distance is reduced to 1.5 and the generated path no longer contains the large jumps. These artefacts are a result, mainly, of the level design in combination with the excessive distance allowed for the path to move at any point.

Engineering Box

Setting This Value It is quite a difficult challenge to know how large to make this value and this is one of the goals of testing and validation. The lower the value for the maximum distance the faster the path finder runs, as it has to consider less nodes. If the value is too high then paths through the level will be missed, even though they contain valid moves.

The interesting thing about Citadel is that it has three main bridge sections between the two ends of the level. This leads to the path finder selecting a bridge as the play tends to be across each of the bridges. As such anything above and beyond this is merely tweaking small details of the path. This said it should be obvious that this type of level will still benefit from this choice of bridge section under the assumption that one of the bridges will be better for approach or return play.

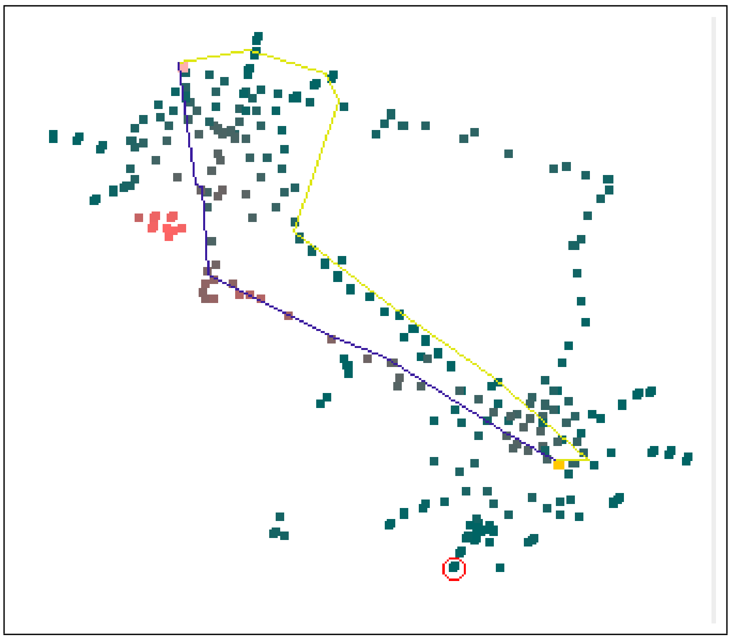


Figure 7.104: Citadel Path generated using only the enemy model for both approach and avoid modes

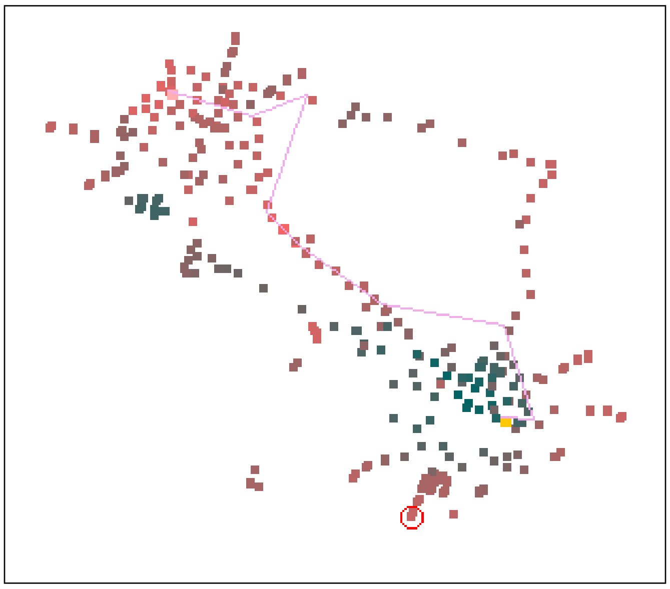


Figure 7.105: Citadel Path Generated with all models contributing

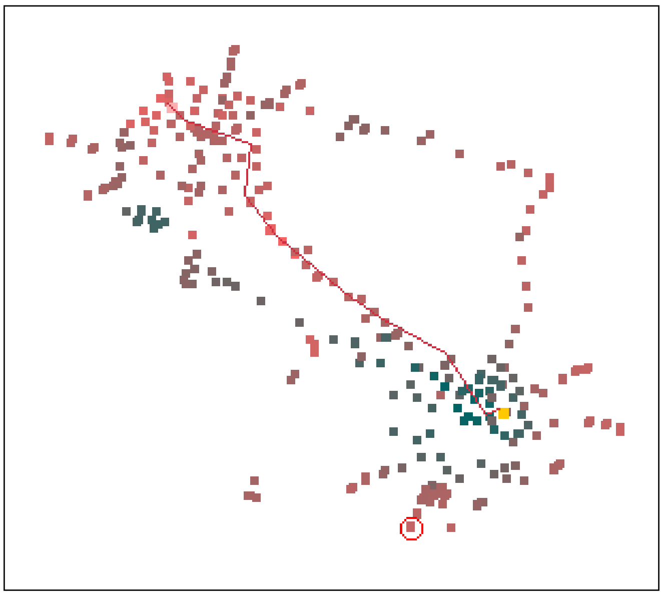


Figure 7.106: Citadel Path Generated with all models contributing but maximum distance of 1.5

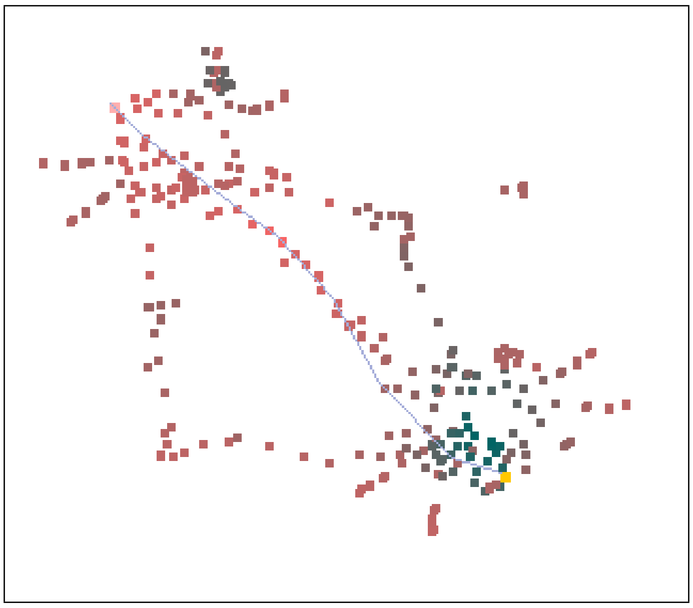


Figure 7.107: Citadel Path Generated with all models contributing and the flag approach utility doubled

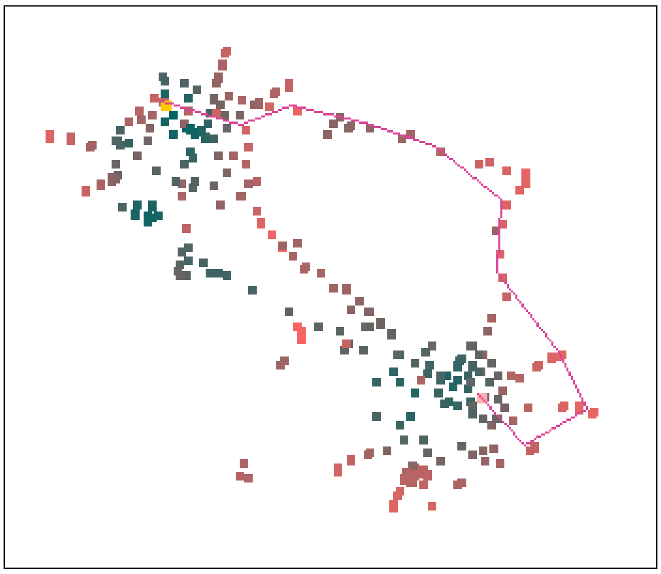


Figure 7.108: Citadel Path Generated with all models contributing for return

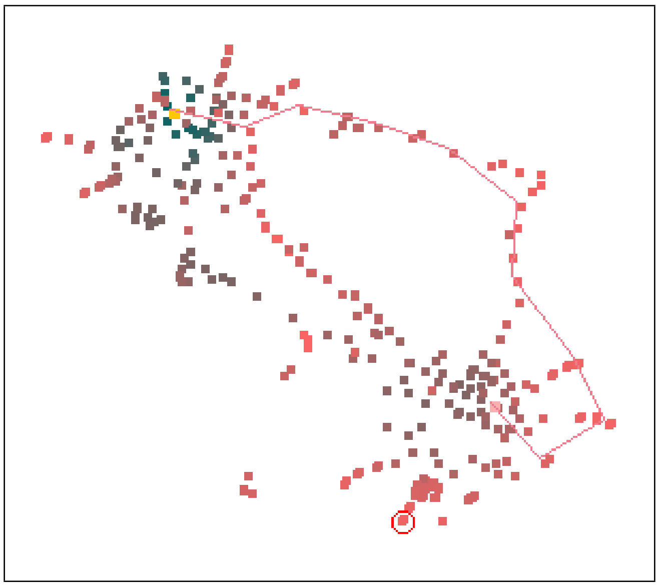


Figure 7.109: Citadel Path Generated with all models contributing for return and the flag return utility doubled

7.10.7.2 Ruination

Ruination is one of the DD levels, as such some results regarding the domination point route models can be presented in conjunction with the path finder. In response to the previous section the initial value for the maximum distance was set at 1.5 to try to avoid some of the artefacts shown to be prevalent in larger maximum value cases.

Again only half the nodes were used as this gave the best results. When we moved up-to 1/3 of the nodes problems began to seep into the paths generated which could only be solved by increasing the maximum distance allowed. This caused a degeneration in the quality of the paths.

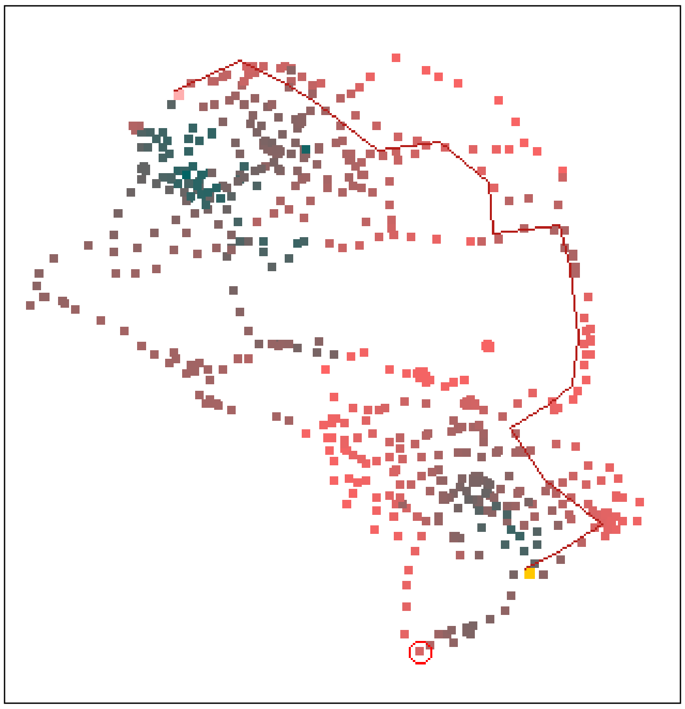


Figure 7.110: Ruination Path Generated with all models contributing for dom point A

7.10.7.3 Trite

Trite is a TDM map. The TDM results show how the enemy model can be factored into the equation but in general are not as interesting as those for CTF and DD. This is because the game of TDM does not lend itself as well to the idea of routes to points as there are less defined goals.

The diagrams here show a variety of different routes. Some approaching areas of high density and others low. There are also a variety of maximum values shown to illustrate how this can effect the paths taken. The situated evaluation in the following

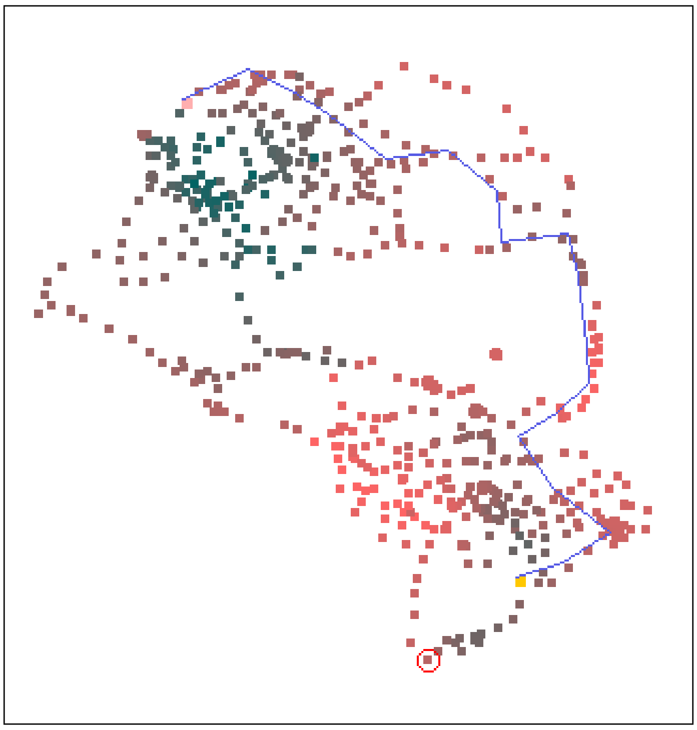


Figure 7.111: Ruination Path Generated with all models contributing for dom point B

section is better for showing the actual effect of the path finder but it is beneficial to motivate why the results are as they are.

Figure 7.114 shows the avoid and approach paths on one visualisation. The paths are different and we can imagine how a bot playing the game would go a different route in each case.

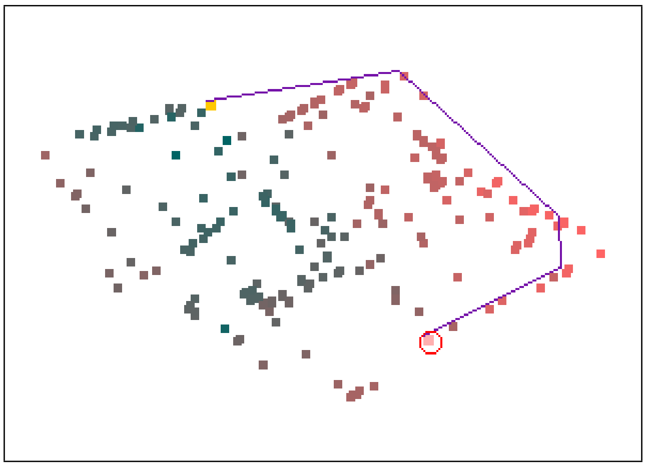


Figure 7.112: Trite Approach, Adjacency 1.5

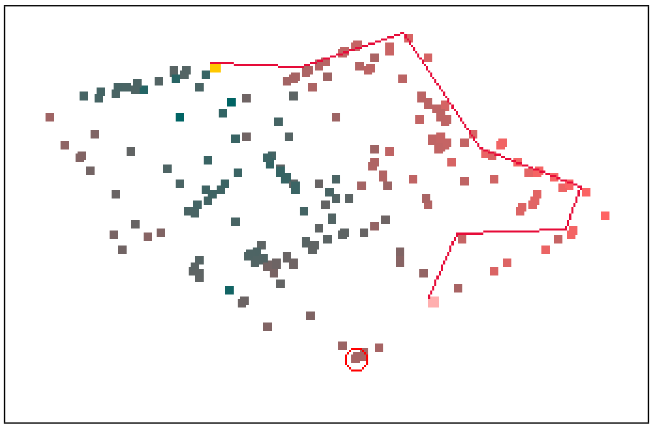


Figure 7.113: Trite Approach, Adjacency 1.5 1

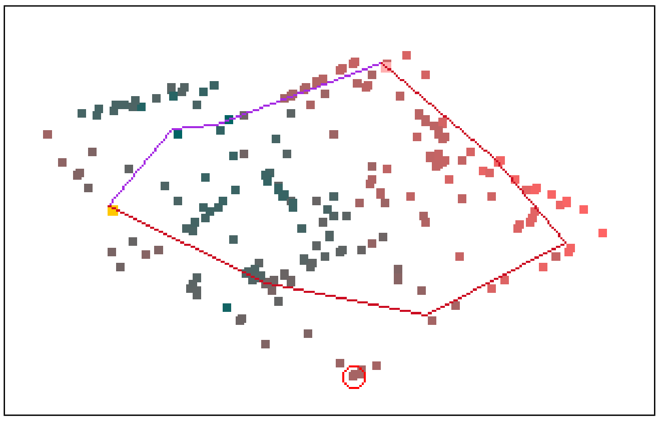


Figure 7.114: Trite Approach and Avoid, Adjacency 1.5

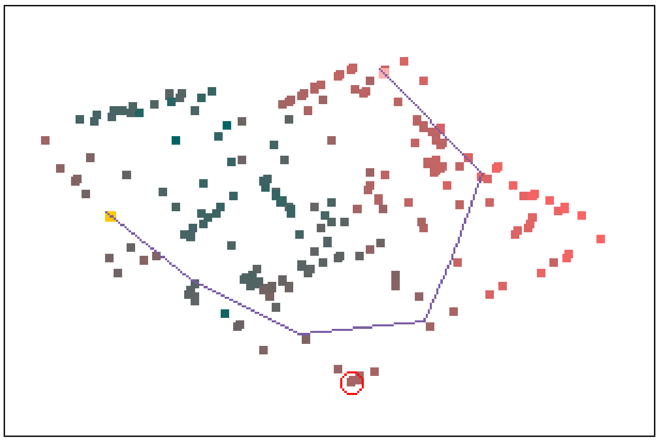


Figure 7.115: Trite Approach, Adjacency 2-1

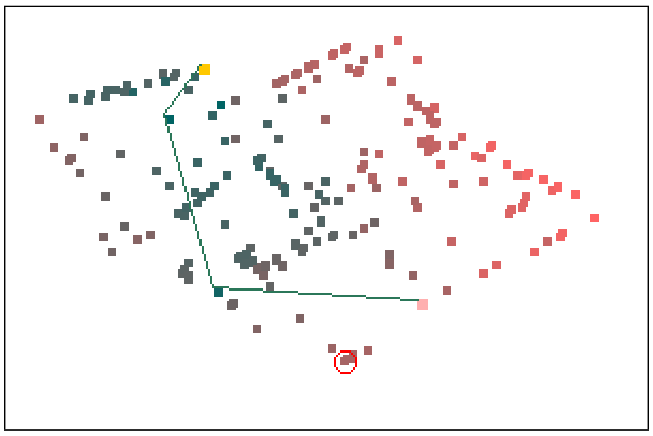


Figure 7.116: Trite Avoid, Adjacency 1.5

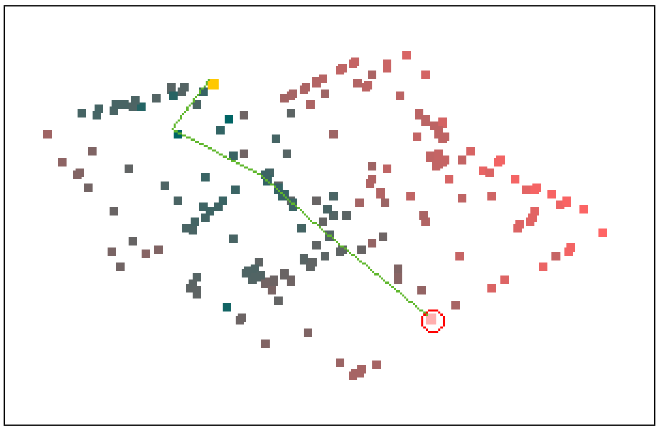


Figure 7.117: Trite Avoid, Adjacency 1

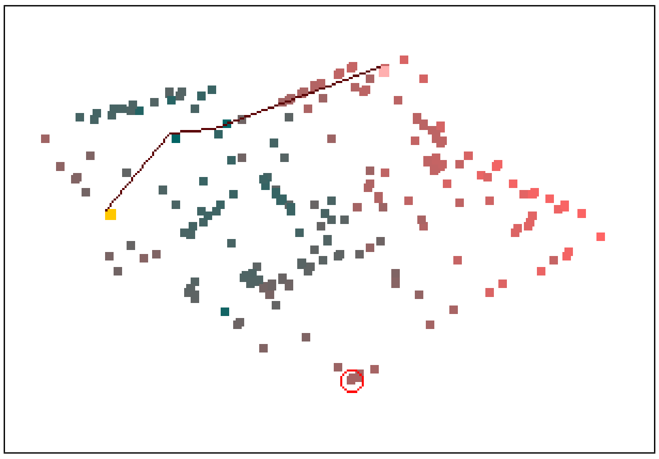


Figure 7.118: Trite Avoid 2, Adjacency 1.5

7.10.8 Situated Evaluation

In the situated evaluation we plugged the path finder into the system and ran it in some games to show the difference to team scores with or without the path finder.

The first thing that we discovered was that although the theory of having the hierarchical path finding system was well grounded, in practice it needed some tweaking. If any of the following conditions occurred the next node in the top level path was removed from the path:

1. If the path to the next node in the overall path was more than twice the length to the goal destination or the overall path.
2. If the next node in the overall path was *behind* the player.
3. If over 50% of the path to the next overall path node was *behind* the player.
4. If the time taken to get to the next overall node had exceeded the reasonable threshold.
5. If we were within 1250 game units of the next overall path node

By *behind* here we refer to the point being located behind a plane which is generated by taking the vector from the bot's location the goal location as a normal to the plane.

The reasonable threshold was set to be the distance to the next overall path node divided by the bot's acceleration (approximately 150 game units per game cycle).

Engineering Box

Motivation These modifications helped to avoid some of the problems introduced by non-accessible nodes and other problems such as walls in the way of the path. They also left the important parts of the path intact and gave good results.

7.10.8.1 TDM

For the TDM section the path finder was used to show how the enemy model could be factored into a module presented earlier. In a previous section the weapon model was used to allow the bots to gain an improved performance by retrieving the best weapon, as gauged by the weapon model, and then engaging the enemy. In the following trials this strategy was used as a baseline against a dynamic strategy which used the path

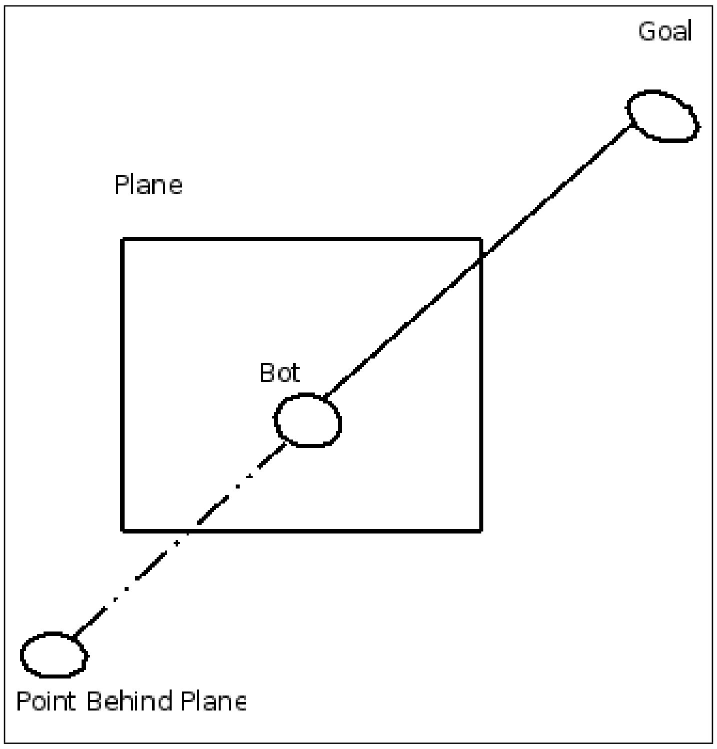


Figure 7.119: Definition of behind

finder to give a path to these weapons. The path finder was set to avoid the enemy by giving a negative value to the level points via the enemy model. Three bots used the weapon model while a 4th updated it and performed explorative duties as per the level modelling section.

From figure 7.120 it is clear the model using the dynamic path finder performs worse. This is primarily down to the smaller size of the level (ironic) and the similarity of the paths generated to those generated by the in-built path finder. Following the in-built path finder is more efficient than using the custom built one so if the paths generated are not significantly better no real advantage is gained. Even so the result is not significantly worse.

7.10.8.2 Double Domination

The double domination trials were played on Scorched Earth and Sun Temple. The strategy used was for one bot to run to dom point A and the other to dom point B. The baseline was for the bots to use the in-built path finder while the dynamic model used the dynamic path finder. The models used were the dom approach route model, the enemy model and the lift and jump node avoidance models. On Scorched Earth all of the level nodes were used with a max distance of 1 and on sun temple the max distance

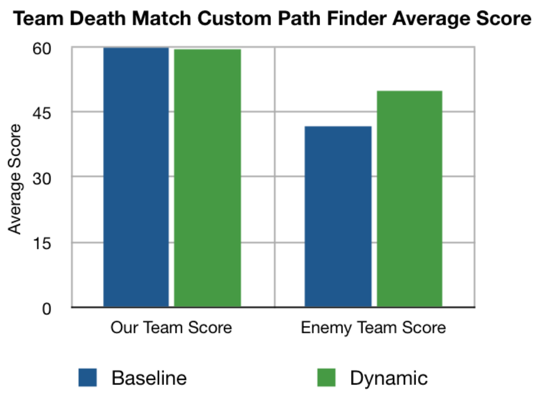


Figure 7.120: Situated Team Death Match Custom Path Finder Results

was 1.5 using only half of the nodes.

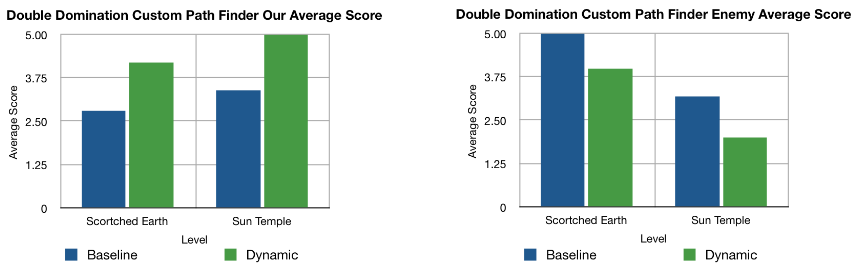


Figure 7.121: Situated Double Domination Custom Path Finder Results

Figure 7.121 shows that a substantial increase in performance is gained from using the dynamic model. In particular the sun temple trials were very interesting to watch. Surrounding one of the domination points is a three tunnel entrance to the point. One entrance is very exposed to anyone standing at the domination point while the other two, located either side of this, are less exposed and offer a safer approach. During play the bot approaching this point would often begin by going the obvious route down the central tunnel. After a few failed approaches it would then begin trying the side tunnels, which often fared better. This is an interesting emergent behaviour which is a consequence of the dynamic path finder in conjunction with the approach route model.

7.10.8.3 Capture the Flag

For the capture the flag trials two levels were used, Citadel and Magma. Citadel is the smaller of the two levels. The strategy had the bots run to the enemy flag point when they didn't have the flag and run back to home base if they did. They also selected the best weapon via the in-built game metric.

Two types of strategy were used, one which used the in-built path finder and one which used the dynamic. The dynamic was set to follow the best flag approach route nodes from the flag approach route model, on the way to the flag, and similarly from the return route models on the way back. It also avoided high enemy presence nodes and avoided the lift and jump nodes in the level.

For Citadel all of the level nodes were used and the max distance was set to 1. For Magma $\frac{1}{3}$rd of the nodes were used and the maximum distance was set to 2.5.

The trials performed had the bots used a strategy analogous to the *last* strategy from [40] which would select the last weapon picked up as the current weapon of choice.

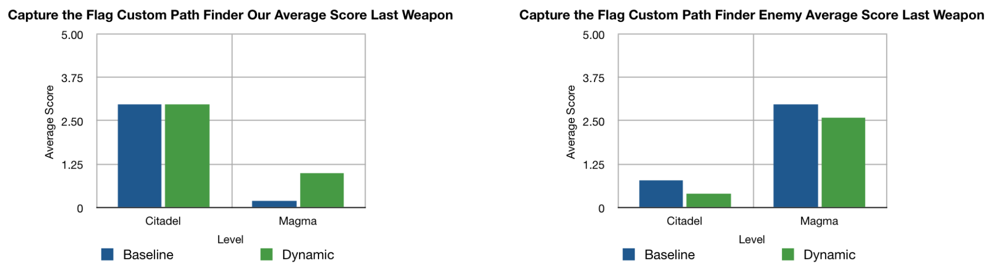


Figure 7.122: Situated Capture the Flag Custom Path Finder Results, Last Weapon Selection

We can see that the custom path finder performs better than the standard version. As per the double domination models, this shows that both the path finder and the models used are giving the bot improved performance. The performance is not even across the two levels but that is a goal of the entire system. One small part which gives a slight increase is moving in the right direction.

7.10.9 Path Finding Conclusions

One of the most important things shown in this section is that with only a 3rd of the path nodes in a level, considering only nodes which are on the direction towards our goal, we can generate paths which reflect closely the optimal paths generated by full dynamic A*. Because the underlying hierarchical path system will pass these paths onto the in-built path finder then, as long as these paths are relatively similar, the system will not fail.

In a way it is almost advantageous for the top level paths to not take into account all the nodes in the level. We wish to be able to see that the paths generated correctly reflect our intuitions about avoiding certain nodes in the level. Generating a higher level path which picks out hotspot points in the level, of higher utility, gives the bot a more general path which moves around areas rather than meeting specific point targets.

Another interesting observation is that the fear/attraction factor can be used to balance the trade off between distance from the goal of the paths and the node utilities. This is important because it gives a lot of control over the bot's movement.

When the path finder is working it is important that an attraction landscape is created which the path finder can follow to its goal. The path finder works best when all the nodes in the level have a utility forming a gradient which can be used to guide the generation of a decent path.

We also saw that there was good information in both the domination point and flag approach point models which could be used to guide play in these game types. We expect that a more careful treatment of these models in terms of exploration/exploitation will yield more consistent results.

In conclusion the path finder does give improved performance but only when the models used are good. It is only as strong as the modelling utility information which is fed into it. Feed in bad, or non-relevant, information and it will under-perform. This is true to the point that even standard paths which are virtually the same as those in the game will under-perform due to inefficiencies in the path following mechanism.

7.10.10 Further Adaptations

An interesting further adaptation of this path finder is to use the polygons in the level model as general navigation points for the path finder. The start and end points are set in certain polygons. These polygons are then rated via the cumulative node utilities of all the points in them, giving a polygon utility value. These polygons are then fed into the path finder as nodes which give a very high level path. This is not perfect because in some levels there are minimal amounts of polygons.

Engineering Box

General Conclusion In general with path finding in a situation where an efficient low level path finder is available it is best to keep high level paths fairly abstract. Activities such as identifying level zones and communal areas can give good results.

7.11 Overall Conclusions from Modelling

Having presented the results regarding the modelling portion of the system we now draw some conclusions.

A lot of the time the more mathematically grounded, and complex, models generate results which are not particularly beneficial. This ranged from not much better than guessing to cases where the model output generated was completely useless. Very often this was to do with sparseness of data, which negates the benifits to our system of using techniques with such a high level of mathematical rigour. A good example is in estimating the parameters of a Gaussian mixture model. If the number of data items is not greater than N^2, where N is the number of items in the covariance matrix, then this yields poor models [12, 11].

There is an analogy to *big-OH* notation where the constants within calculations are often overlooked. In our low numerical data set cases the constants can make all the difference as the constants are what contain the information.[12].

In comparison to mixture models the simpler parzen density estimator, which is non-parametric, can begin generating useful density models from a smaller amount of input data as it is designed to work with samples of a distribution. The same can be said of the k-nearest neighbours algorithm. For instance if we only have one example of a positive and negative case then this may be sufficient if they were significantly good representations of the input/output function and were well separated.

We also saw that it was possible to use a probability density function over a space to assign utility to static nodes using samples of a dynamic target concept. This allowed us to map out an area to define a probability space which could be used to guide movements within the area.

We saw that in most cases we could successfully model the concepts which we felt were desirable to the bots and performed some form of evaluation of these techniques to suggest that they would be useful at a later point. The purpose of the evaluation was to show that these models offered something to the bot and to what extent a decision procedure could be based on them. The evaluation also offered a chance to test and set a variety of parameters for each model and to examine how the models worked with respect to the domain and data types. We showed how some models could be altered to perform better with augmented data sets and also drew attention to the bias

[12]This is not to say we are concerned with run-times, in our situation the constants are related to the low numbers of data-points

this introduced.

Having created a set of models and tested them we then tied the more geometric of these together with a modification of the classic A* algorithm for path finding and showed how this could be used to generate high level paths which could be passed to a low level path finder to implement.

This path finder also showed that while no individual model result was particularly startling, when they were put together they created a situation which was advantageous to the bot. The key is not in finding one model which immediately makes performance perfect, but in showing that each model can give a small improvement. We can then integrate these to gain an overall performance stability increase.

Engineering Box

Caution One very strong message from this chapter was that no matter how good a model we have, the way in which we use it can severely alter the performance of the bot. Naive use can lead to situations which are worse then not using the model at all.

7.11.1 Eventual Chosen Model Set

The following table summarises which models were chosen from the chapter :

All models chosen were used in a communal mode with one model which all bots in learning roles updated.

Engineering Box

Other Models Originally there were two further models in the tested model set, flag and domination point decision models. These resembled the fight decision model. Neither of these models were used beyond initial testing. This was because early tests with these models in practice suggested that, although they generated good performance on off-line data, the game-type characteristics were such that always approaching either the flag or domination point always lead to better performance. This is because the game-types are time limited. The fight model works better because not choosing to fight does not mean loosing a fight where as not approaching a currently occupied domination point often will lead to loosing a point.

Model Name	Chosen Model	Parameter Settings
Enemy Model	Parzen Window Density Estimator	Variance 500, No Discounting
Fight Model	K-Nearest Neighbours	Mahalanobis Distance, Injection Set 1
Level Model	Renyi Entropy Clustering	Quickhull Polygon Fitting
Area Correlation Model	Kill and Death Counts Over Polygons	
Flag Approach Route Model	Parzen Window Density Estimator	Variance 500, No Discounting
Domination Point Approach Route Model	Parzen Window Density Estimator	Variance 500, No Discounting
Flag Sighting Model	Parzen Window Density Estimator	Variance 10, No Discounting
Dynamic Path Finding	Dynamic A*	

Figure 7.123: Chosen Models

7.11.2 Alternatives

Throughout each section in this chapter we have presented alternatives for the methods chosen and tried to discuss in brief the applicability of these to the domain. The machine learning techniques presented here are by no means the most powerful or the best but they show performance which is largely acceptable and as such are good enough for our purposes.

Chapter 8

Behaviour Modules

In the previous chapter a number of modelling techniques were used to gather dynamic data about the levels used in the game. These formed layer 1 of the overall system architecture which we present in this thesis. In this chapter we construct behaviour modules based on layer 1, forming layer 2 of the architecture. These perform the role of combining the various modelling techniques of section 7 and are then combined themselves into larger strategies in layer 3, as per Figure 5.1. These modules can be considered as separate.

The modules in this section do not perform adaptation and are fixed and hard coded. They are simply switched on or off at any given time by a top level strategy. Their purpose is to provide stable building blocks of behaviour of which the exact details, for any given decision, are controlled by the machine learning modules in the lower layer.

In this chapter we will deal with only the modules which are used in the larger scale strategies described in chapter 9.

8.1 Exploitation vs. Exploration

As discussed earlier an important issue from reinforcement learning is the notion of exploration vs. exploitation. This is the idea that for a lot of adaptive entities, which must act in their environment, there is a trade-off between acting on what has been learned and selecting a different action which may lead to another option not considered. Although most of the machine learning mechanisms which we use are not reinforcement learning based the way in which we use them, situated in a real-time environment, forces us to consider many of the same considerations and constraints.

One of the concentrations of chapter 7 was how the machine learning mechanisms should be used in practice. Much care was taken to point out that naive use of the machine learning techniques was not a good idea. Different roles which were more or less explorative and exploitative were used to show how this kind of issue could be dealt with. This idea is expanded in this chapter by showing which modules are more or less exploitative and concluded in the next chapter with LCC strategies containing the eventual trade-off dynamics. There is also a chapter exploring the benefits of this approach to the problem.

8.2 The Module Descriptions

Because the modules were written in Java code it does not serve us best to simply include the code. Instead for each module a list is given of the models which it uses and those which it updates. A brief description is then given of the behaviour of the module along with pseudo-code and notes explaining anything extra.

8.3 TDM

In this section we present the TDM game-type modules.

8.3.1 TDM Weapon User V6

Module A	
Models Used	Weapon,Fight,Enemy,Area Correlation
Models Updated	None

Figure 8.1: TDM Weapon User V6

8.3.1.1 Description

This module is the main TDM exploiter. It uses all available models and dynamic path finding to guide play. The idea is to obtain the best weapon via the distance/utility metric and then engage enemies only when the fight model stipulates. In other cases, move to the most likely area of enemy occupancy.

8.3.1.1.1 Pseudo-code

```
1.Switch to current best weapon held using weapon model
2.IF Enemy Visible and fight model predicts a kill and no death{
3.        Move towards enemy shooting at them
4.}
5.ELSE{
6.    use distance/utility correlation to assess most attractive weapon
7.    IF most attractive weapon not held and no higher utility weapon held {
8.        Use dynamic path finding, with enemy avoid and area avoid flags,
          to move to most attractive weapon pickup location
9.    }
10.   ELSE{
11.       Use dynamic path finding, with enemy approach and area approach
flags,
          to move to sampled likely enemy location
12.   }
13.}
```

8.3.1.2 Notes

8.3.1.2.1 Distance Correlation The distance/utility correlation is a method of determining the attractiveness of different weapons based on the criteria of weapon model utility and distance from current location.

$$Attractiveness(Weapon_x) = \frac{\text{utility}_x}{50} - \frac{\text{Distance from current location}}{1000}$$

8.3.1.2.2 Enemy Engagement Engaging the enemy consists of moving towards them shooting. If the enemy are seen but not engaged then they will still be shot at but not moved towards.

8.3.1.2.3 Enemy Location Sampling The method of sampling the enemy model for the most likely location was to use roulette wheel selection [8], over the navigation points, with roulette wheel proportions based on enemy model probability values.

8.3.1.2.4 Path Finding Flags The path finding flags used in the dynamic path finding were as follows :

Enemy Avoid Use the enemy model to bias paths towards avoiding enemy location

Area Avoid Use the area correlation model to bias paths towards avoiding areas of high death probability

Area Approach Use the area correlation model to bias paths towards area of high kill probability

8.3.2 TDM Weapon User V6 Follower Learning

Module A	
Models Used	Weapon,Fight,Enemy,Area Correlation
Models Updated	Weapon

Figure 8.2: TDM Weapon User V6 Follower Learning

8.3.2.1 Description

This module is roughly the same as TDM Weapon User V6 except that in the cases where the fight model says not to fight the bot will move to the the input bot's location rather than sampling the enemy model for the most likely enemy position. The purpose of this is to allow the bots to group together to increase their fire-power. This module also updates the weapon model with fight outcomes, allowing the weapon model to collect data about which weapons are most effective for follower bots.

8.3.2.1.1 Pseudo-code

```
Input(Other Bot's Name)

1.Switch to current best weapon held using weapon model
2.IF Enemy Visible and fight model predicts a kill and no death{
3.         Move towards enemy shooting at them
4.         Update Weapon Model with result
5.}
6.ELSE{
7.     use distance/utility correlation to assess most attractive weapon
8.     IF most attractive weapon not held and no higher utility weapon held {
9.         Use dynamic path finding, with enemy avoid and area avoid flags,
           to move to most attractive weapon pickup location
10.    }
11.    ELSE{
12.        Use dynamic path finding, with enemy approach and area approach
flags,
           to move to other bot's location
13.    }
14.}
```

8.3.3 TDM Weapon User V5 Learning

Module A	
Models Used	Weapon,Fight,Enemy,Area Correlation
Models Updated	Weapon

Figure 8.3: TDM Weapon User V5 Learning

8.3.3.1 Description

This is the default module for this game-type as it sits between exploiting and exploring. It is similar to TDM Weapon User V6 but it also updates the weapon model.

8.3.3.1.1 Pseudo-code

```
1.Switch to current best weapon held using weapon model
2.IF Enemy Visible and fight model predicts a kill and no death{
3.         Move towards enemy shooting at them
4.         Update Weapon Model with result
5.}
6.ELSE{
7.     use distance/utility correlation to assess most attractive weapon
8.     IF most attractive weapon not held and no higher utility weapon held {
9.         Use dynamic path finding, with enemy avoid and area avoid flags,
           to move to most attractive weapon pickup location
10.    }
11.    ELSE{
12.        Use dynamic path finding, with enemy approach and area approach
flags,
           to move to most likely enemy location
13.    }
14.}
```

8.3.4 Dedicated Weapon Fight Enemy Area Sampler

Module A	
Models Used	Weapon
Models Updated	Weapon,Fight,Enemy,Area Correlation

Figure 8.4: Dedicated Weapon Fight Enemy Area Sampler

8.3.4.1 Description

This module is the main exploring module for TDM. Its purpose it to explore the level sampling the weapons and routes and determining which of these are most beneficial to play.

8.3.4.1.1 Pseudo-code

```
1.Select weapon with least magnitude of utility
2.IF Weapon not held {
3.        Use dynamic path finding, with random flag, to run to weapon pickup
4.        Select It
5.}
6.ELSE{
7.    IF visible enemy {
8.        Engage Enemy
9.        update weapon, fight, area correlation and enemy models
10.   }
11.   ELSE{
12.       play randomly
13.   }
14.}
```

8.3.4.2 Notes

8.3.4.2.1 Magnitude of Utility The magnitude of utility is the absolute value of the weapon utility

8.3.4.2.2 Path Finding Flags The path finding flags used in the dynamic path finding were as follows :

Random All nodes have a random utility

8.3.4.2.3 Random Play Random plays consists of picking a random health or ammo pickup and running there

8.3.5 Dedicated Weapon Fight Enemy Area Sampler Last

8.3.5.1 Description

This is similar to the first explorer module except that it uses the last weapon that was picked up. This is designed to sample which weapons are most often found on throughout the level. Thus if routes are common we wont the most sampling data

Module A	
Models Used	Weapon
Models Updated	Weapon,Fight,Enemy,Area Correlation

Figure 8.5: Dedicated Weapon Fight Enemy Area Sampler Last

about the weapons on those routes are the bots are likely to posses said weapons most often.

8.3.5.1.1 Pseudo-code

```
1.Select weapon with least magnitude of utility
2.IF Weapon not held {
3.         Use dynamic path finding, with random flag, to run to weapon pickup
4.}
5.Select Last Weapon Picked Up
6.ELSE{
7.     IF visible enemy {
8.         Engage Enemy
9.         update weapon, fight, area correlation and enemy models
10.    }
11.    ELSE{
12.        play randomly
13.    }
14.}
```

8.4 CTF

In this section we present the CTF game-type modules.

8.4.1 Flag Exploiter V1 Best

Module A	
Models Used	Weapon, Fight, Flag approach route, Enemy
Models Updated	Flag approach route, Flag sighting

Figure 8.6: Flag Exploiter V1 Best

8.4.1.1 Description

This is the main exploiter module for CTF. Because CTF is a more complicated game type with more opportunities for adaptation there are now fully exploitative modules used. The main goals of the module are to make assaults on the enemy flag and return it to base using the weapon model to select the best model and the fight model to determine when best to engage enemies. The dynamic path finder is used with a variety of learning flags to select the best paths to make assaults on the enemy flag.

8.4.1.1.1 Pseudo-code

```
Select weapon held according to weapon model
IF Got flag {
    IF at own flag base {
        Play randomly around own flag point
        Update Flag Route Model
    }
    ELSE{
        Use dynamic path-finding, with flag return route, enemy avoid,
        death area avoid and kill area approach flags, to run to own
        flag point
    }
}
ELSE{
    IF Can see own flag and its not at home base {
        Use basic path-finding to run to flag
        Update flag- sighting model
    }
    ELSE{
        IF Visible enemy and fight model predicts kill {
            Run at enemy shooting
        }
        ELSE{
            assess best weapon with distance utility measure
            IF Weapon not held and no higher utility weapon held {
                use dynamic path-finding, with death area avoid and enemy hide
                flags, to run to weapon pickup location
            }
            ELSE {
                IF Can see enemy flag and its not at enemy base {
                    Use basic path-finding to run to it
                }
                ELSE {
                    IF At enemy flag base {
                        play randomly around enemy flag point
                        Update Flag Route Model
                    }
                    ELSE {
                        Use dynamic path-finding, with flag
approach route,
                                     enemy avoid, death area avoid and kill area approach
                                     flags, to run to enemy flag point
                              }
                        }
                  }
            }
      }
}
```

8.4.2 Flag Explorer V2

Module A	
Models Used	Weapon,Flag approach route, Enemy
Models Updated	Flag approach route, Weapon, Enemy, Fight, Flag sighting

Figure 8.7: Flag Explorer V2 Best

8.4.2.1 Description

This is the main explorative module for CTF. Its purpose is to make random flag runs to determine effective flag routes. In the course of this it will also sample the weapons by confronting enemies to judge weapon effectiveness.

8.4.2.1.1 Pseudo-code

```
Select the last weapon picked up
Determine weapon of least magnitude
IF Visible Enemy and holding weapon of least magnitude {
        Run at enemy shooting them
        Update weapon and fight models
}
ELSE {
    IF not got weapon of least magnitude {
           Use basic path-finding to run to weapon pickup location
    }
    ELSE {
          IF got flag {
                 IF at own flag point {
                      Play randomly around own flag point
                      Update flag route model
                 }
                 ELSE {
```

```
18.                  Use dynamic path-finding, with random flag,
                     to run to own flag point
19.               }
20.         }
21.         ELSE {
22.               IF Visible enemy flag not at enemy base {
23.                       Use basic path-finding to run to flag
24.               }
25.               ELSE {
26.                   IF at enemy flag point {
27.                        play randomly around enemy flag point
28.                        Update flag route model
29.                   }
30.                   ELSE {
31.                        Use dynamic path finding, with random flag,
                           to run to enemy flag point
32.                   }
33.               }
34.         }
35.  }
36.}
```

8.4.3 Protector(Location)

Module A	
Models Used	Weapon
Models Updated	Flag sighting

Figure 8.8: Protector(Location)

8.4.3.1 Description

The protector module offers support to bots in certain locations by movoing there to add extra fire power. Should it see our flag on route then it will return this before continuing.

8.4.3.1.1 Pseudo-code

```
1.Select best weapon according to weapon model
2.IF can see own flag, not at own base {
3.    Use basic path-finding to run to it
4.    Update flag-sighting model
5.}
```

```
6.ELSE {
7.    IF can see enemy flag not at enemy base {
8.          Use basic path-finding to run to it
9.    }
10.    ELSE {
11.        Use basic path-finding to run to Location
12.    }
13.}
```

8.4.3.2 Notes

8.4.3.2.1 Location `Location` is a 3D location passed into the module from the LCC strategy. This module then operates a following function.

8.4.4 Retriever V1 Non-Learning

Module A	
Models Used	Flag sighting, Weapon
Models Updated	Flag sighting

Figure 8.9: Retriever V1 Non-Learning

8.4.4.1 Description

The retriever module's sole purpose to get back our flag when the enemy steal it. The flag sighting module is used to guide a route to the enemy base to stop them returning it.

8.4.4.1.1 Pseudo-code

```
1.Select best weapon according to weapon model
2.IF can see own flag not at own base {
3.    Use basic path-finding to run to it
4.    Update flag sighting model
5.{
6.ELSE {
7.          Use dynamic path-finding, with flag sighting flag,
            to run to enemy base
8.}
```

8.5 DD

In this section we present the DD game-type modules.

8.5.1 Dom A Exploiter

Module A	
Models Used	Weapon,Fight,Dom Approach Route
Models Updated	Dom Approach Route

Figure 8.10: Dom A Exploiter

8.5.1.1 Description

The domination point modules are generally split into two categories, those for point A and those for point B. These normally mirror each other in functionality only differing in the use of the correct models for the point in question.

This module is the main exploiter module for DD. Its purpose is to get to dom point A and defend it. To do this it uses the weapon and fight models in conjunction with the domination point model and updates the dom route as it plays.

8.5.1.1.1 Pseudo-code

```
1.Switch to current best weapon held using weapon model
2.IF Enemy Visible and fight model predicts a kill and defending{
3.         Move towards enemy shooting a them
4.}
5.ELSE{
6.     use distance/utility correlation to assess most attractive weapon
7.     IF most attractive weapon not held and no higher utility weapon held {
8.         Use dynamic path finding, with dom route A flags
           to move to most attractive weapon pickup location
9.     }
10.    ELSE{
11.             IF not defending{
12.                     IF not reached dom point A{
13.                           Use dynamic path finding, with dom route A flag,
14.                           to run to dom point A
15.                     }
16.                     ELSE{
17.                         defending = true
18.                         update dom approach route model
19.                     }
20.            }
22.            ELSE{
22.                 Play randomly around dom point A
23.            }
24.      }
25.}
```

8.5.1.2 Notes

8.5.1.2.1 Distance Correlation Instead of using the current location to correlation the distance utility value, the distance from the target point is used.

8.5.1.2.2 Localised Random Play Line 22 uses the concept of localised random play. This is where the bot picks random navigation points within a small radius of a location and runs to them and back.

8.5.1.2.3 Path Finding Flags The dom approach route A flag provides data to the path finder about the best domination point approach route, using the dom approach route model.

8.5.2 Dom B Exploiter

Module A	
Models Used	Weapon,Fight,Dom Approach Route
Models Updated	Dom Approach Route

Figure 8.11: Dom B Exploiter

8.5.2.1 Description

This is the mirror of Dom A Exploiter but for point B.

8.5.2.1.1 Pseudo-code

```
1.Switch to current best weapon held using weapon model
2.IF Enemy Visible and fight model predicts a kill and defending{
3.         Move towards enemy shooting a them
4.}
5.ELSE{
```

```
6.     use distance/utility correlation to assess most attractive weapon
7.     IF most attractive weapon not held and no higher utility weapon held {
8.         Use dynamic path finding, with dom route B flags
           to move to most attractive weapon pickup location
9.     }
10.    ELSE{
11.             IF not defending{
12.                     IF not reached dom point B{
13.                           Use dynamic path finding, with dom route B flag,
14.                           to run to dom point B
15.                     }
16.                     ELSE{
17.                         defending = true
18.                         update dom approach route model
19.                     }
20.             }
22.             ELSE{
22.                  Play randomly around dom point B
23.             }
24.       }
25.}
```

8.5.3 Dom A Weapon Approach Updater

Module A	
Models Used	none
Models Updated	Weapon, Dom Approach Route

Figure 8.12: Dom A Weapon Approach Updater

8.5.3.1 Description

This module is designed to test random approach routes to domination point A using the last picked up weapon.

8.5.3.1.1 Pseudo-code

```
1.Switch to last weapon picked up
2.              IF not defending{
2.                      IF not reached dom point A{
2.                            Use dynamic path finding, with random flag,
5.                            to run to dom point A
6.                      }
7.                      ELSE{
```

```
            defending = true
            update dom approach route model
        }
    }
    ELSE{
        Play randomly around dom point A
    }
  }
}
```

8.5.4 Dom B Weapon Approach Updater

Module A	
Models Used	none
Models Updated	Weapon, Dom Approach Route

Figure 8.13: Dom B Weapon Approach Updater

8.5.4.1 Description

This is the mirror of Dom A Weapon Approach Updater but for point B.

8.5.4.1.1 Pseudo-code

```
Switch to last weapon picked up
    IF not defending{
        IF not reached dom point B{
            Use dynamic path finding, with random flag,
            to run to dom point B
        }
        ELSE{
            defending = true
            update dom approach route model
        }
    }
    ELSE{
        Play randomly around dom point B
    }
  }
}
```

8.5.5 Dom A Dedicated Weapon Fight Sampler

Module A	
Models Used	Weapon, Dom Approach Route
Models Updated	Weapon, Fight

Figure 8.14: Dom A Dedicated Weapon Fight Sampler

8.5.5.1 Description

This module samples the weapons and then makes an assault on dom point A to test their effectiveness.

8.5.5.1.1 Pseudo-code

```
1.Select weapon with least magnitude of utility
2.IF Enemy Visible and selected weapon held{
3.        Move towards enemy shooting a them
4.}
5.ELSE{
6.    IF selected weapon not held {
7.        Use basic path finding to move to
          most attractive weapon pickup location
8.    }
9.   ELSE{
10.           IF not defending{
11.                  IF not reached dom point A{
12.                       Use dynamic path finding, with dom route A flag,
13.                       to run to dom point A
14.                  }
15.                  ELSE{
16.                      defending = true
17.                      update dom approach route model
18.                  }
19.           }
20.           ELSE{
21.               Play randomly around dom point A
22.           }
23.      }
24.}
```

Module A	
Models Used	Weapon, Dom Approach Route
Models Updated	Weapon, Fight

Figure 8.15: Dom B Dedicated Weapon Fight Sampler

8.5.6 Dom B Dedicated Weapon Fight Sampler

8.5.6.1 Description

This is the mirror of Dom A Dedicate Weapon Fight Sampler but for point B

8.5.6.1.1 Pseudo-code

```
1.Select weapon with least magnitude of utility
2.IF Enemy Visible and selected weapon held{
3.         Move towards enemy shooting a them
4.}
5.ELSE{
6.    IF selected weapon not held {
7.         Use basic path finding to move to
           most attractive weapon pickup location
8.    }
9.   ELSE{
10.             IF not defending{
11.                     IF not reached dom point B{
12.                           Use dynamic path finding, with dom route B flag,
13.                           to run to dom point B
14.                     }
15.                     ELSE{
16.                         defending = true
17.                         update dom approach route model
18.                     }
19.             }
20.             ELSE{
21.                  Play randomly around dom point B
22.             }
23.       }
24.}
```

8.5.7 Dom V2 Weapon Approach Updater

8.5.7.1 Description

This is the same as the Dom A and Dom B Exploiter modules except that it picks the point which it is closest to and then plays as per that point's corresponding module.

Module A	
Models Used	none
Models Updated	Weapon, Dom Approach Route

Figure 8.16: Dom V2 Weapon Approach Updater

8.5.7.1.1 Pseudo-code

```
Switch to last weapon picked up
        IF not defending{
                IF not reached nearest dom point{
                      Use dynamic path finding, with random flag,
                      to run to nearest dom point
                }
                ELSE{
                      defending = true
                      update dom approach route model
                 }
        }
        ELSE{
              Play randomly around nearest dom point
        }
    }
}
```

8.5.8 Dom V2 Dedicated Weapon Fight Sampler

Module A	
Models Used	Weapon, Dom Approach Route
Models Updated	Weapon, Fight

Figure 8.17: Dom V2 Dedicated Weapon Fight Sampler

8.5.8.1 Description

This is the same as the dedicated samplers for points A and B but will pick the nearest point and choose that as its target.

8.5.8.1.1 Pseudo-code

```
Select weapon with least magnitude of utility
IF Enemy Visible and selected weapon held{
        Move towards enemy shooting at them
}
ELSE{
    IF selected weapon not held {
        Use basic path finding to move to
        most attractive weapon pickup location
    }
    ELSE{
        IF not defending{
            IF not reached nearest dom point{
                Use dynamic path finding, with dom route A and B flags,
                to run to nearest dom point
            }
            ELSE{
                defending = true
                update dom approach route model
            }
        }
        ELSE{
            Play randomly around nearest dom point
        }
    }
}
```

8.6 Adapting this Layer

Although we chose not to adapt this layer of the architecture during the course of a match it is not particularly difficult to imagine a situation where this could be allowed. One example could be to take each module as not a single module but a collection of scripting elements and then apply a method such as Spronck's dynamic scripting [96] to this script set in an on-line manner.

Our argument is that although this might provide more power and adaptability it does not yield the same level of controllability or stability. Our goal is to be able to verify that the overall desired behaviour of our team strategy has been met. The use of an adaptable behaviour module layer makes this difficult unless the exact scripts used are very tightly regimented. This regimentation is likely to lead to a similar situation to that which we are already in: known scripts with slight details adapted with machine learning.

Chapter 9

Team Strategies

In this chapter we detail the team strategies that make up layer 3 in the architecture. These are used to control the behaviour modules, of layer 2, designed in the previous chapter. They do this by turning particular behaviour modules on and off. Again no adaptation is performed in this layer but it is plausible to again imagine that dynamic scripting could be used to allow a reformulation of the strategies at real-time. Our reasons for not performing this are the same as in the behaviour modules case.

Agents in the team strategies are given particular roles. These roles correspond to behaviour modules in a 1 role to many behaviour modules relationship. The idea is to allow particular roles to be given certain sets of behaviours which constitute a desired, stable behaviour set. Each agent's particular role, and the current mode of behaviour within that role, are changed during matches in response to changes in performance or communications from other agents.

The strategies are presented in LCC along with a description and the starting roles of the bots used in the trials.

9.1 Exploration vs. Exploitation Revisited

One of the most interesting parts of the system is the way in which explorations and exploitation are dealt with. In particular an unusual stance is presented showing that multiple bots in the environment can be given specific behaviours which have more or less of an explorative or exploitative feel. Bots can even be sacrificed fully to the task of exploration while others exploit. The strategy can then be assessed and utilities assigned to roles over a certain time-scale.

Engineering Box

Meta LCC This concept opens up the notion of Meta level LCC in much the same way that an interpreter such as Sicstus opens up control possibilities in Prolog using predicates such as var(X). In the testing chapters we show that the LCC strategy can be used to control and monitor the bots and allow them to adjust their performance in response to changes in role utility.

9.2 Reactive, Deliberative and Power Structures

One of the most hotly debated topics in agent design is the issue of reactive versus deliberative and how this choice effects the notion of a plan. In [33] Devigne et al have the following to say of reactive systems:

> "This approach has limitations as soon as complex behaviours are desired."

In short they feel that complex planned behaviour is difficult to achieve with a reactive system. They also argue that it is difficult to allow the individual agents in a deliberative system any level of autonomy. This attitude of associating reactivity with chaos and deliberation with complex planning ability and no autonomy seems prevalent throughout the field of intelligent agent design and goes hand in hand with previous discussions of multi-agent control.

Our argument is that by having a coherent multi-agent plan and allowing single agents to carry out their particular section of that plan with autonomy these issues can be dealt with quite simply. Planning need not follow the deliberative path to afford the dealing with changes in the details of the plan using our system and the advantages associated with a reactive system are not lost.

9.3 Team Death Match

This is our main strategy for the TDM game-type.

9.3.1 Description

This strategy has explorers and exploiters. The balance of numbers of each is changed in response to a performance increase or decrease in the exploiter role. This realises a performance centric approach to the team exploitation versus exploration trade-off, allowing tapering of the team learning rate based on an estimation of the value of the current state.

There are 4 abstract roles with 2 LCC roles, the `hunter` and the `base`. Within each role is the concept of a *stuck* player, denoted by the names `sh` and `sb` for *stuck* `hunter` and *stuck* `base` respectively. They are referred to as *stuck* because their roles do not change throughout the game.

The *stuck* `hunter`'s primary purpose is to perform exploitative duties. As a second purpose they assign roles to the *auxiliary* players based on `hunter` performance. The *auxiliary* players start out as `base` players. In this role they perform exploratively. When they kill an opponent they send a message to the *stuck* `hunter` player requesting a role change. The *stuck* `hunter` then assesses the last five encounters of the `hunters`. If over 50% of these were a kill , they send back a role change message to the *aux* player who in turn becomes a `hunter`[1]. The *stuck* `hunter` also sends out messages to other `hunters` telling them to follow and support. They in return follow the *stuck* `hunter` while updating weapon data.

The *stuck* `baseline` player performs explorative duties such as sampling weapons, sampling the arena area, and engaging enemies to judge effectiveness of weapons, and expected fight outcomes.

As the game progresses, the models accumulate more information and the performance of the `hunter`bots increases, forcing the *aux* players to focusing on exploitative rather than explorative play. This achieves convergence of the machine learning techniques at a team level.

9.3.2 The LCC

```
Strategy 4.1

%%The stuck hunter, reassigns members to the hunter module based on performance
a(hunter,sh)::engageModule(tdm_weapon_user) <-- gotAKill <= a(base,ID)
then change => a(base,ID) <-- performance(hunter,kill)

a(hunter,sh)::followMe(sh) => a(hunter,_) <-- engageModule(tdm_weapon_user)

a(hunter,sh)::null<--engageModule(tdm_weapon_user)


%%The standard hunter
a(hunter,Id)::engageModule(tdm_weapon_user_follower_learning(ID)) <--
```

[1]Upon death the *aux* players return to a base role

```
followMe(ID) <= a(hunter,sh)

a(hunter,Id)::null<--engageModule(tdm_weapon_user_learning)

%%The stuck baseliner module
a(base,sb)::null<-- engageModule(dedicated_weapon_fight_enemy_area_sampler)

%%The baseliner module
a(base,Id)::engageModule(dedicated_weapon_fight_enemy_area_sampler_last) and
changeToRole(hunter) <-- change <= a(hunter,ID)

a(base,Id)::gotAKill => a(hunter,sh) <--
engageModule(dedicated_weapon_fight_enemy_area_sampler_last) and kill(true)

a(base,Id)::null <--
engageModule(dedicated_weapon_fight_enemy_area_sampler_last)
```

9.3.3 The Starting Team

Team Member	Starting Role
sb	base
aux1	base
aux2	base
sh	hunter

Figure 9.1: Starting TDM team

9.4 Capture The Flag

This is our main strategy for the CTF game-type.

9.4.1 Description

The strategy features two main roles, the `explorer` and the `exploiter`. The two roles are almost identical in their operation with the exception that one uses an exploiter module while the other uses an explorer module.

If a bot receives a protection request it must move to the location specified in the request and follow the bot making the request. Requests are generated from bots when they have the enemy flag.

When the enemy have our flag we run the *flag retriever* module to try to regain it.

There is also a rule which states that `explorers` can become `attackers` when the performance of the `exploiters` is rated as kill.

9.4.2 The LCC

```
Strategy 4.2

% This changes the roles of explorers in response to positive exploiter play
a(attacker_explore_v2,Id)::null<--changeToRole(attacker_exploit) and
performance(attacker_exploit,kill)

%% The weapon fight sampling attacker module
a(attacker_explore_v2,Id)::engageModule(protector(Location)) <--
protect(Location) <= a(_,_)

a(attacker_explore_v2,Id)::protect(Location) => a(attacker_exploit,_)
<-- engageModule(flag_explorer_v2) and hasFlag(true) and selfLocation(Location)
then protect(Location) => a(attacker_explore_v2,_)

a(attacker_explore_v2,Id)::null<--engageModule(retriever_v1_non_learning) and
enemyHasFlag(true)

a(attacker_explore_v2,Id)::null <-- engageModule(flag_explorer_v2)

%% The fully dynamic attacker module
a(attacker_exploit,Id)::engageModule(protector(Location)) <-- protect(Location)
<= a(_,_)

a(attacker_exploit,Id)::protect(Location) => a(attacker_exploit,_)
<-- engageModule(flag_exploiter_v1_best) and hasFlag(true) and
selfLocation(Location)
then protect(Location) => a(attacker_explore_v2,_)

a(attacker_exploit,Id)::null<--engageModule(retriever_v1_non_learning) and
enemyHasFlag(true)

a(attacker_exploit,Id)::null<--engageModule(flag_exploiter_v1_best)
```

9.4.3 The Starting Team

Team Member	Starting Role
exploit1	attacker-exploit
explore1	attacker-explore-v2
explore2	attacker-explore-v2
explore3	attacker-explore-v2

Figure 9.2: Starting CTF team

9.5 Double Domination

9.5.1 Description

The strategy consists of two main roles, the `exploiter` and the `explorer`. The `exploiter` role is centred around sending messages to other bots in the exploiter roles detailing your current distance from the domination points. Upon receiving these messages the other bots test whether their current distance from either point is closer or further than that of the message sender. They engage the exploiter module for the point which they are relatively closer to.

The `explorer` role is split into 4 main phases. Each phase (except phase 1) has the same method as the `exploiter` role for determining which point to approach.

Phase 1 states that if the `exploiter` role has a performance rating of kill and we were the last team to score a domination point then we should play as an `exploiter` for the appropriate point.

Phase 2 states that if we were the last team to score but the `exploiter`role is not at a kill rating then we should play as a `focus explorer` role (using the weapon approach updater). This module explores the weapons on route to the domination point.

Phase 3 states that if the `exploiter` is not at a kill rating and we were not the last team to score we should play using a weapon fight sampler module. This module explores all the weapons in the level.

In all the cases above a message is received from other bots with the distance to each of the domination points and another in turn send out. In the initial cases these messages are not present and so Phase 4 consists of the sending these messages out and playing as point independent explorative modules.

9.5.2 The LCC

```
Strategy 4.3

%There must always be two exploiters and two explorers for this to work.

a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y])
and engageModule(dom_a_exploiter) <-- domDetails(A,B) <= a(exploiter,_)
then domDetails(Da,Db) => a(exploiter,_)

a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y]) and
engageModule(dom_b_exploiter)
```

```
<-- domDetails(A,B) <= a(exploiter,_) then domDetails(Da,Db) => a(exploiter,_)

a(exploiter,_)::domDetails(Da,Db) => a(exploiter,_)<--squaredDistance(a,Da) and
squaredDistance(b,Db)

%Phase 1

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_exploiter)
and performance(exploiter,kill) and lastTeamToScore(T) and team(T)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y])
and engageModule(dom_b_exploiter) and performance(exploiter,kill)
and lastTeamToScore(T) and team(T) <-- domDetails(A,B) <= a(_,_)
then domDetails(Da,Db) => a(explorer,_)

%Phase 2

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_weapon_approach_updater)
and performance(exploiter,kill) <-- domDetails(A,B) <= a(_,_)
then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y])
and engageModule(dom_b_weapon_approach_updater)
and performance(exploiter,kill)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

%Phase 3

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y])
and engageModule(dom_a_dedicated_weapon_fight_sampler)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y])
and engageModule(dom_b_dedicated_weapon_fight_sampler)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

%Phase 4

a(explorer,_)::domDetails(Da,Db) => a(explorer,_)<--squaredDistance(a,Da)
and squaredDistance(b,Db) and performance(exploiter,kill)
and lastTeamToScore(T) and team(T)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_)<--squaredDistance(a,Da)
and squaredDistance(b,Db) and performance(exploiter,kill)
and engageModule(dom_v2_weapon_approach_updater)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_)<--squaredDistance(a,Da)
and squaredDistance(b,Db)
and engageModule(dom_v2_dedicated_weapon_fight_sampler)
```

9.5.3 The Starting Team

Team Member	Starting Role
explore1	explorer
exploit1	exploiter
explore2	explorer
exploit2	exploiter

Figure 9.3: Starting DD team

Engineering Box

Lack of Stability in Numbers Because of the nature of the message sending and distance calculations this strategy becomes largely unpredictable with different starting numbers of players in each of the two roles. An ideal number is 2 and 2. The reason this doesn't work with larger numbers is best explained with an example as in figure 9.4. Consider a team of 3 bots trying to assign roles. Bot 1 sends their current position and bot 2 receives this. They choose the point which they are closer to and begin moving towards it. Bot 3 then sends their co-ordinates to bot 1 and they are actually closer to the point so they change their target. This then continues and the situation becomes cyclic.

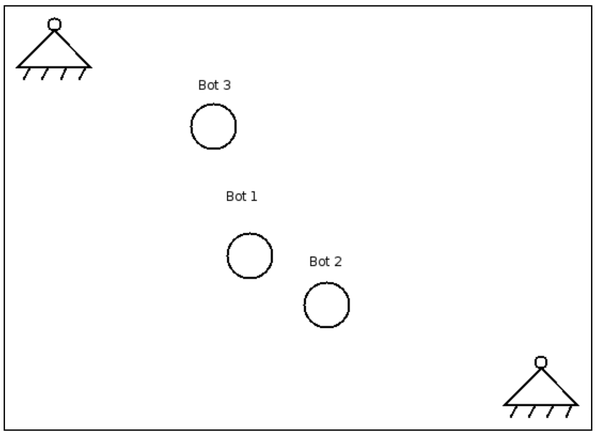


Figure 9.4: Three Bot Situation

Part III

Evaluation and Analysis

Chapter 10

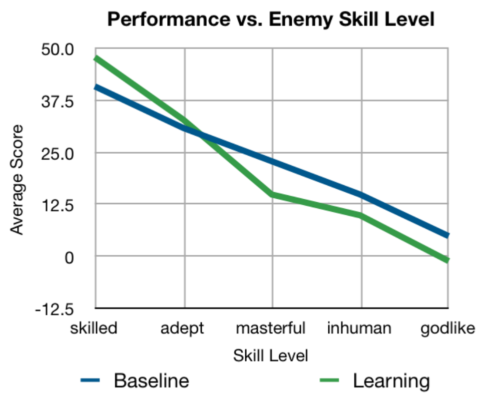
Testing

In this chapter we present the results from testing the architecture. This is our main evaluation of the system and is our main evidence to support the claims made in Chapter 1. We tested over the three game-types with the strategies described in chapter 9.

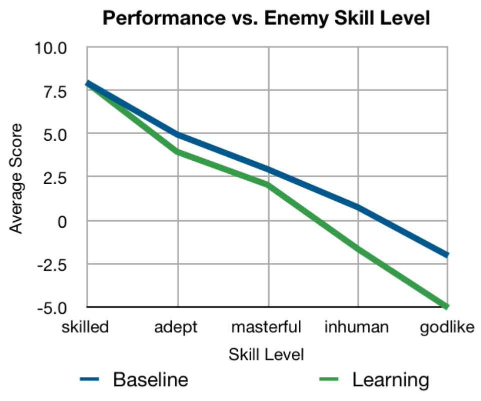
10.1 Different Enemy Skill Levels

As performance is a mediating factor in our learning rate, enemy skill has an effect on our score. The enemy skill ratings determine factors such as the enemy's speed and shooting accuracy and speed of evasive manouvering. They do not mark any changes in reasoning, thinking or intelligent behaviour. The following figures show the space of enemy skill versus performance increase and show the level of non-linearity of the correlation for the three different game types. The result was obtained by taking an average score over 5 trials on one level in each case. This is not a robust method of showing the exact space but it can be used as a guide to show which areas of the space are likely to give the best results.

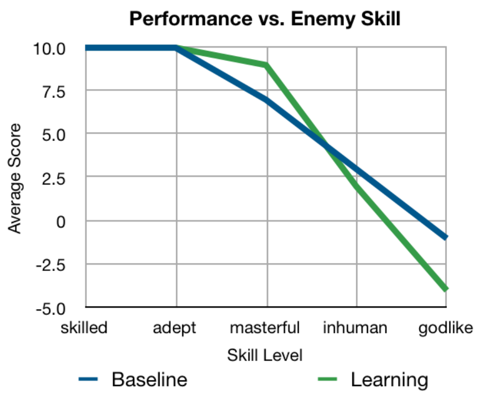
Figure 10.1(a) shows that the largest positive gap between the learning strategy and baseline (The baseline strategy is one in which all the models are turned off. Both strategies are played against the in-built bots and the difference in scores is measured. This is our general evaluation strategy in this chapter and is better explained in the proceeding pages) for TDM occurs when the *skilled* skill level is tested. Although the degradation in the baseline strategy is largely linear, in response to enemy skill changes, the learning strategy has an optimal zone for performance increase and sub-optimal zones elsewhere.



(a) TDM



(b) CTF



(c) DD

Figure 10.1: Skill Levels Charts

Figure 10.1(b) shows that for CTF there is no skill level where there is a positive gap between learning and baseline strategies. This suggests that potentially there isn't any performance increase at any skill level. The best testing option is the *skilled* skill level.

Figure 10.1(c) shows that the largest positive gap between the learning strategy and baseline for DD occurs when the *masterful* skill level is tested.

In all game-types there is non-linearity in the learning strategy's increased performance in response to changes in enemy skill.

10.2 Methodology

Our evaluation method was to play 5 matches against the in-built bots and measure the difference in scores. We then did the same for a baseline. The baseline was identical to the main strategy except that is did not use any of the learning mechanisms. This was performed over several levels for each game type and the results were assessed both on a per level and cross-levels basis to give an overall performance measure. We also performed these tests over two enemy skill levels. This was to show the highest possible performance achievable and also to highlight how drastically the nature of the enemy can effect learning rate which is mediated by a performance rating. The skill levels were chosen in accordance with the results of section 10.1.

10.3 Results

The figures in this section display the results of the trials. The level based results have three graphs showing the average `baseline` and `learning` strategy scores for each level, the standard deviations in these scores and the P-values of the T-tests between the strategies. The data obtained in the level based trials passed a KS normality test and welch's un-paired T-Test variant was used to compensate for un-equal variances.

Thoughout this section line graphs are used when perhaps bar graphs would be more conventional. This is because we wish to show how stable the play is across levels. Therefore, it makes more sense to consider this as being as close to a straight line as possible.

10.3.1 TDM

10.3.1.1 Level Based Results - Inhuman Skill Level

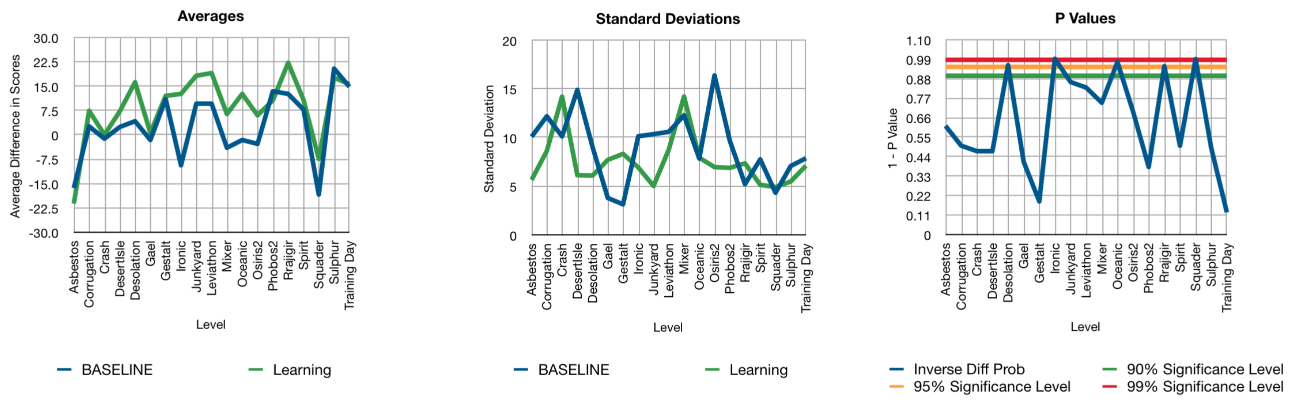


Figure 10.2: TDM Level Based Results - Inhuman

Figure 10.2 shows the results of the level based analysis. The first graph shows the average differences in the score for both the `learning` strategy and the `baseline` strategy. It shows that the learning strategy is ahead in most levels and maintains a more stable line of average scores. In the cases where the `baseline` performance drops substantially the `learning` strategy does not drop as substantially. Observing the graph of standard deviations we can see that neither the `learning`nor the `baseline` strategy are particularly stable. Standard deviation values are high in both cases and show much fluctuation. The final graph shows the inverse P-values for the T-tests for each level. In the cases where the performance of the `baseline` and the `learning` strategy are not drastically different there is no statistically significant increase. In the cases where the `baseline` drops severely in performance the `learning`strategy is statistically higher in performance and closer to the cross-level average. There are no cases where the `baseline` is statistically significantly higher in average than the `learning` strategy. This is a desirable result.

10.3.1.2 Cross-Level Based Results - Inhuman Skill Level

The cross level analysis took the individual difference scores for each match and then assessed them as one 95 trial dataset. In some cases the data in this section did not pass a normality test as such we have reported the conclusions of the Gaussianility tests used as well as the results of a wilcoxon paired median test. Because each set of 5 trials represents a separated section a paired T-Test was used. Fig 10.3 shows the analysis.

	Analysis	
	Baseline	Learning
Average Score	2.98	8.95
Median Score	3	12
KS Normality Test	Pass	Fail
D'Agostino And Pearson Omnibus Normality Test	Pass	Fail
Shapiro-Wilk Normality Test	Pass	Fail
Standard Deviation	13.444	12.419
Size of set	95	95
99% Confidence Interval	2.4799	2.2909
Paired T-Value	**4.289**	
Paired P-Value	<**0.0001**	
Wilcoxon P-Value	**0.0002**	

Figure 10.3: Cross Level Results Analysis - Inhuman

Figure 10.3 shows that the average score for the `learning` strategy is significantly higher than that of the `baseline`in both the T-Test and Wilcoxon cases (Median difference is larger than average difference here). The probability of this difference in averages being down to chance is less than 1%. The standard deviation of the `learning` strategy is slightly lower although not as much as we would have hoped. This said, the average score is significantly higher.

10.3.1.3 Level Based Results - Skilled Skill Level

Both the `baseline` and `learning` strategies are more stable at this skill level but this is to be expected as the enemies are posing less of a threat. The `learning` strategy average scores are slightly higher and the standard deviation values are much more stable and lower. In the Junkyard and DesertIsle cases the `baseline` strategy's standard deviation value peaks sharply while the `learning` strategy's does not. This shows a more stable performance across these levels which is more in line with performance on other levels. The T-test scores show that the `learning` strategy is significantly better in more levels than in the *inhuman* case, signifying a gap in performance.

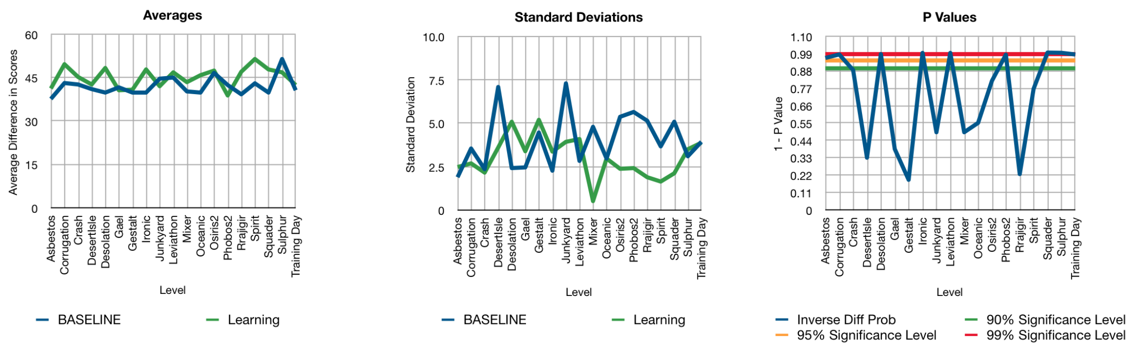


Figure 10.4: TDM Level Based Results - Skilled

10.3.1.4 Cross-Level Based Results - Skilled Skill Level

The cross level analysis took the individual difference scores for each match and then assessed them as one 95 trial dataset. Fig 10.3 shows the analysis.

	Analysis	
	Baseline	Learning
Average Score	43.41	45.16
Median Score	43	45.00
KS Normality Test	Pass	Pass
D'Agostino And Pearson Omnibus Normality Test	Pass	Pass
Shapiro-Wilk Normality Test	Pass	Pass
Standard Deviation	5.16	4.35
Size of set	95	95
99% Confidence Interval	1.4874	1.2544
Paired T-Value	**2.674**	
Paired P-Value	**0.0091**	
Wilcoxon P-Value	**0.0130**	

Figure 10.5: Cross Level Results Analysis - Skilled

Although the `learning` strategy is significantly better, over all the level data, the margin of this significance is less than for the *inhuman* case. The difference between the averages and medians is also less than it was in the *inhuman* case.

10.3.2 Capture The Flag

10.3.2.1 Level Based Results - Inhuman Skill Level

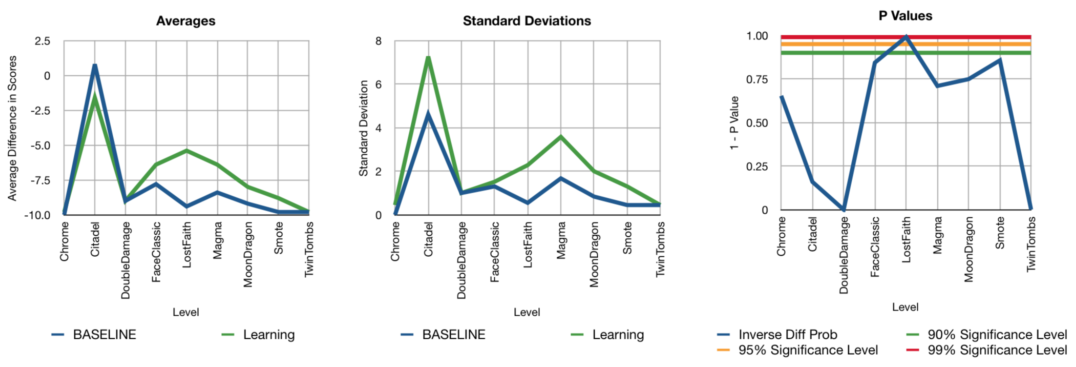


Figure 10.6: CTF Level Based Results - Inhuman

Fig 10.6 shows the level based results for the CTF game type. These are not as good as the TDM type results because there is less stability. In most levels the standard deviation of the `learning` strategy is higher than that of the `baseline` which is largely undesirable. The T-tests hold some hope as the cases where the `baseline` is above the `learning` have very low inverse P-values and the cases where the `learning` strategy score is higher than the `baseline` have much higher inverse P-values. This shows that in cases where the `learning` strategy is ahead it is significantly ahead but in cases where the `baseline` is ahead the result is not significant.

One of the more interesting elements of the results here is that the standard deviation value fluctuates with level choice in a way which was not observed in the TDM trials. As average score increases so does standard deviation. This makes it difficult to achieve a positive T-Test result as T-Tests factor in standard deviation. It points towards an inherent instability in the learning mechanisms for CTF used as they are achieving very varied results depending on the particular trial.

10.3.2.2 Cross-Level Based Results - Inhuman Skill Level

The cross level analysis took the individual difference scores for each match and then assessed them as one 50 trial dataset. Fig 10.7 shows the analysis.

The table shows that using the paired T-Test there is no significant improvement in the average scores, cross levels, from using the learning based strategies for CTF. The

	Analysis	
	Baseline	Learning
Average Score	-8.1	-7.28
Median Score	-9	-8
KS Normality Test	Fail	Fail
D'Agostino And Pearson Omnibus Normality Test	Fail	Fail
Shapiro-Wilk Normality Test	Fail	Fail
Standard Deviation	3.4796	3.5458
Size of set	50	50
99% Confidence Interval	1.2675	1.2916
Paired T-Value	**1.645**	
Paired P-Value	**0.1063**	
Wilcoxon P-Value	**0.0227**	

Figure 10.7: Cross Level Results Analysis - Inhuman

standard deviation is higher and even with a dataset size of 50 there is roughly a 90% probability that the average difference in scores is explained by chance.

This said we can see that the data fails all Gaussianility tests and as such the assumption of the T-Test no longer applies. The difference in the medians of 1, however, is significant via the paired Wilcoxon test.

10.3.2.3 Level Based Results - Skilled Skill Level

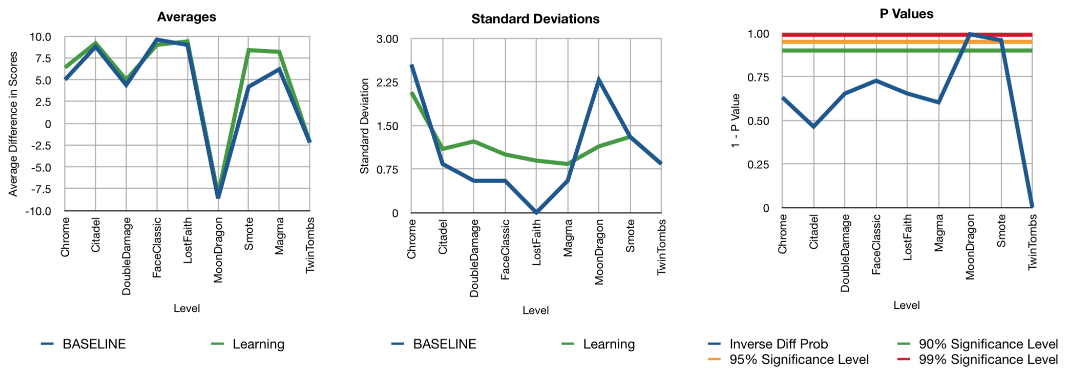


Figure 10.8: CTF Level Based Results - Skilled

The figure 10.8 results are quite different from those in the *inhuman* case. The levels in which the `learning`performs better than the `baseline` are not the same and the T-test results are completely different. The averages for the `learning` case are almost identical to the `baseline` cases showing no drastic increase. The learning standard deviation values are much more stable than the `baseline` but the T-tests show that there are only two levels where a statistically significant result was achieved.

10.3.2.4 Cross-Level Based Results - Skilled Skill Level

The cross level analysis took the individual difference scores for each match and then assessed them as a 50 trial dataset. Fig 10.9 shows the analysis.

	Analysis	
	Baseline	Learning
Average Score	4.52	5.08
Median Score	6.5	7.5
KS Normality Test	Fail	Fail
D'Agostino And Pearson Omnibus Normality Test	Fail	Fail
Shapiro-Wilk Normality Test	Fail	Fail
Standard Deviation	5.5924	5.6994
Size of set	50	50
99% Confidence Interval	2.0372	2.0761
Paired T-Value	**2.082**	
Paired P-Value	**0.0426**	
Wilcoxon P-Value	**0.0578**	

Figure 10.9: Cross Level Results Analysis - Skilled

The cross-level results show no significant improvement in the medians using the `learning` strategy at the *skilled* skill level via the Wilcoxon test. The T-Test P-Value is significant but both datasets fail all three Gaussianality tests so no conclusions about the averages can be drawn from this.

10.3.3 Double Domination

10.3.3.1 Level Based Results - Inhuman Skill Level

Fig 10.10 shows the level based results for the DD game type. They are worse than those for CTF. The standard deviation is higher for the `learning` strategies in comparison with the `baseline` and the average scores are generally worse. The only case which passed a T-Test was the case where the `baseline` was significantly higher than the `learning` strategy which is the polar opposite of our intention.

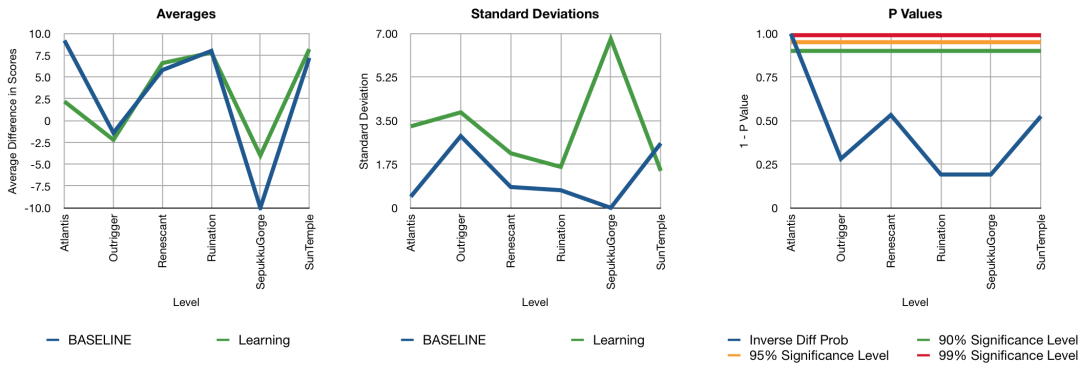


Figure 10.10: DD Level Based Results - Inhuman

10.3.3.2 Cross-Level Based Results - Inhuman Skill Level

The cross level analysis took the individual difference scores for each match and then assessed them as a 30 trial dataset. Fig 10.11 shows the analysis.

	Analysis	
	Baseline	Learning
Average Score	3.133	3.1
Median Score	6	5
KS Normality Test	Fail	Fail
D'Agostino And Pearson Omnibus Normality Test	Pass	Pass
Shapiro-Wilk Normality Test	Fail	Fail
Standard Deviation	7.0746	5.9384
Size of set	30	30
99% Confidence Interval	3.327	2.7927
Paired T-Value	**0.03436**	
Paired P-Value	**0.9728**	
Wilcoxon P-Value	**0.8282**	

Figure 10.11: Cross Level Results Analysis - Inhuman

10.3.3.2.1 Figure 10.11 shows that no significant improvement is gained from using `learning` strategies for the DD game type at *inhuman* skill level. The `learning` strategies achieve a slightly lower standard deviation over the levels, showing slightly more stability, but they are not averaging any better than the `baseline`.

10.3.3.3 Level Based Results - Masterful Skill Level

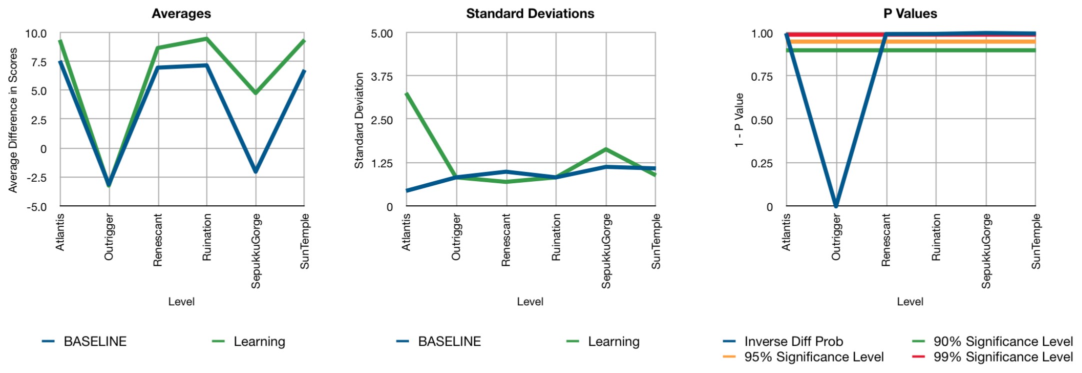


Figure 10.12: DD Level Based Results - Skilled

The average scores are better for the *skilled* case then the *inhuman* case. They match the pattern closely of the `baseline` but are better overall. The standard deviation of the `learning` strategy is much lower than the `baseline` and the T-tests show that in all levels bar Outrigger the `learning` strategy is significantly better than the `baseline`. These results all show that the `learning` strategy is working as expected for the *skilled* case.

10.3.3.4 Cross-Level Based Results - Masterful Skill Level

The cross level analysis took the individual difference scores for each match and then assessed them as a 30 trial dataset. Fig 10.13 shows the analysis.

The cross level analysis backs up the level based results showing a higher average score and lower standard deviation for the learning strategy at the *masterful* skill level. The data fails the Gaussianality tests but the median values, which have a greater difference than the averages, are backed up by the Wilcoxon result.

10.3.4 Conclusions

10.3.4.1 Inhuman Skill Level

The TDM (T-Test) and CTF (Wilcoxon) game-types yielded positive results while DD did not. To understand this, an ideal result, such as that shown in fig 10.14, must be

	Analysis	
	Baseline	Learning
Average Score	3.433	6.16666
Median Score	6	9
KS Normality Test	Fail	Fail
D'Agostino And Pearson Omnibus Normality Test	Fail	Fail
Shapiro-Wilk Normality Test	Fail	Fail
Standard Deviation	5.066	4.9763
Size of set	30	30
99% Confidence Interval	2.5896	2.3402
Paired T-Value	**7.836**	
Paired P-Value	<**0.0001**	
Wilcoxon P-Value	<**0.0001**	

Figure 10.13: Cross Level Results Analysis - Masterful

considered. It shows the graph of the learning strategy performing evenly across the levels (lower cross level standard deviation). Even though the baseline is occasionally higher the even performance of the learning strategy is preferable as its performance could be more reliably predicted. The TDM result isn't as good as this because it deviates at roughly the same times as the baseline model showing that is suffering from some of the same issues as the baseline model regarding weapon placement and level specific problems. This said it is relatively close to the ideal in comparison to the other game types and the statistics show that it is performing better and showing a greater stability.

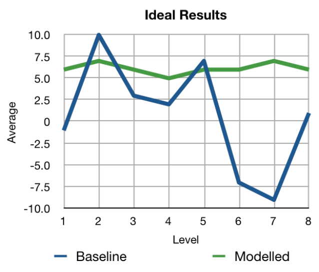


Figure 10.14: Ideal Model Results

10.3.4.2 Skilled/Masterful Skill Level

The *skilled* results are not better for CTF or DD but the *masterful* results are much better for DD showing a stability in the learning strategies that fits with what we wanted and is closer to the ideal result in fig 10.14.

10.3.4.3 Overall System Results

The results show that the system works as expected in the three game-types but that there are factors which can drastically affect performance, particularly in the CTF and DD cases. The `learning` strategies are more stable across the levels than the `baseline` but are not perfect. The unstable performance shows that there are elements of the system which do not work as expected but overall the results are positive and significant showing the validity of the architecture and its ability to facilitate multi-agent machine learning in an effective way.

Engineering Box

Skills Level Abstraction from UT If we treat enemy skill as a measure of adversarialness of the environment then it is possible to abstract some guidelines for systems using current performance as a mediator of learning rate.

- The space of performance increase to adversarial value is likely to be non-linear for all systems of this type
- Performing a basic analysis of the different levels of adversarialness can be used as a guide if the chosen instances are representative of the full testing domain

Chapter 11

Component Testing

In this chapter we detail some further experimentation on the system to try to find out which elements were most responsible for overall performance. This takes the form of several knock-out experiments where each model or strategic component is removed from the system alongside some experiments where they are tested in isolation.

Although interesting, this is by no means a conclusive analysis and the results are largely inconclusive. Ideally a full scale sensitivity analysis should be performed to show the changes in performance on a continuous basis rather than the simple experimental setup performed here but this was outwith the scale of this thesis oweing to the time taken to perform individual experiments.

Engineering Box

Attributing Performance To properly determine the merit of a system or architecture we cannot simply just present results and then claim that the architecture as a whole is responsible for the success. This is particularly prevalent in architectures involving a large number of modular interacting components which are capable of adaptivity. As such we need a way of separating components into what they contribute to the overall performance.

11.1 Methodology

The methodology used had two distinct parts. Part 1 was to remove each machine learning component in turn and assess fluctuations in system's performance. Part 2 was to test each component in the absence of all other components. This allowed the measurement of both the detrimental effects of the removal of a component and the singular effect it was giving to the whole. This allowed performance to be attributed to either single components or a combination of components. It also allowed cumulative assessment of whether the multiple components were combining in a useful way. Because of time constraints this could only be performed on one game level. The level chosen was that which produced a difference between the baseline and the learning strategy. For TDM matches this meant choosing a level with a large positive result and for the other game types this meant taking levels which had negative results as we were trying to assess what the failing components were. This was in line with the fact that we wanted to assess both failures and successes and that TDM gave a better performance than CTF and DD.

The components were tested at two layers of the architecture. The first layer was the module/machine learning level. This involved testing certain machine learning elements and key parts of of the basic movement architecture.

The second layer was the strategy level. At this level certain modules were tested along with strategy and protocol components such as message passing and roles. It was not possible to test components at strategy level on their own and so only component removal testing was performed there.

The CTF game-type was tested over two levels to show a level where we the system performed well and one where it did not to try to determine the cause of the failure.

The graphs presented are relative to the performance difference between the baseline strategy and the fully learning strategy.

11.2 Layer 1 Testing

11.2.1 TDM

For TDM strategies the following components were tested separately in the module layer analysis:

1. Weapon Model
2. Fight Model
3. Enemy Model
4. Area Correlation Model
5. Dynamic Path Finder

6. Distance/Utility Weapon Correlation

The Pseudo-code for the new modules, in the same format as the modules chapter, is available in the document appendix but in the interests of readability is not included here.

11.2.1.1 Inhuman Skill

Figure 11.1 shows the performance changes when removing certain modelling components at the *inhuman* skill level.

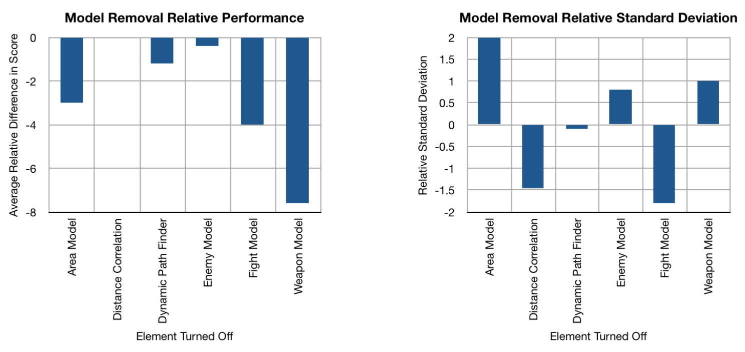


Figure 11.1: Layer 1, Models Off, Component Analysis Results, Inhuman

Figure 11.1 shows that removal of the distance correlation and dynamic path finder elements did not drastically negatively effect model performance. They lowered the standard deviation although the value was less than 2.5 % which is not significant. Removal of the area model lowers performance and increases standard deviation. As the path finder module is based on information from the enemy model and area models, and removal of the enemy model does not create a significant decrease in performance, we can conclude that any increase in performance gained from the path finder is based on information from the area model. The model which gives the largest performance decrease when removed is the weapon model. This makes sense as we would expect that, given the nature of the game type, having the correct weapon would be an important performance factor.

Figure 11.2 shows the performance changes when using only single models in isolation.

The results here are interesting because they show that working with any model on its own provides an increase in performance over no modelling, which is roughly equatable to using all the models, with the exception of the area model which has a

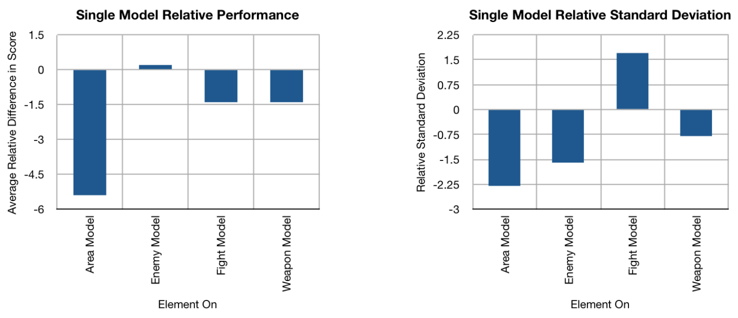


Figure 11.2: Layer 1, Single Models On, Component Analysis Results, Inhuman

slightly lower performance level than the others. This shows that modelling areas in this context is less important then knowing when to fight, which weapon to have and some structured notion of where the enemy are likely to be. This result also points to the actual combination part of the system not performing quite as expected.

11.2.1.2 Skilled Skill

Figure 11.3 shows the performance changes when removing certain modelling components at the *skilled* skill level.

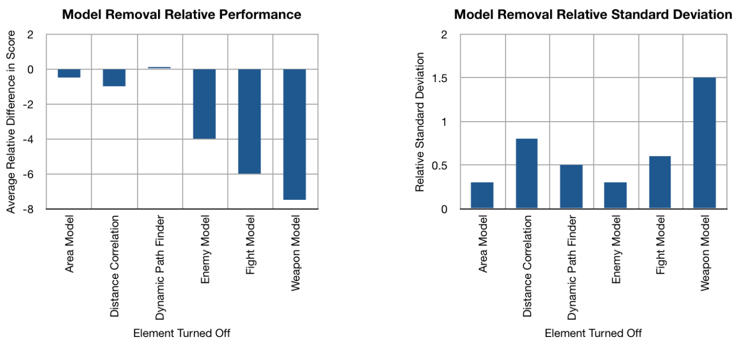


Figure 11.3: Layer 1, Models Off, Component Analysis Results, Skilled

Figure 11.3 shows the results are almost the same for the *skilled* case as the *inhuman* case. The area model removal has less of an effect but the weapon and fight models are still the most important.

Figure 11.4 shows the performance changes when using only single models in isolation for the *skilled* skill level.

The results show that in the *skilled* case the weapon and fight models are the only single models which lead to a large increase in performance. This reinforces the con-

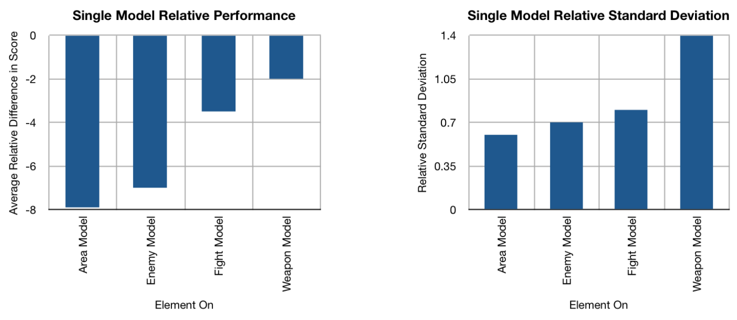


Figure 11.4: Layer 1, Single Models On, Component Analysis Results, Skilled

clusions from the *models off* case which is something which was not so obvious for the *inhuman* skill level.

11.2.1.3 TDM Layer 1 Results Conclusions

The results from the layer 1 analysis show that no particular model is of principle importance to the modelling. We saw that removal of the weapon and fight models resulted in a decrease in performance but also that the other models on their own could gain acceptable levels of performance for the *inhuman* case. As such we conclude that certain models must have a detrimental effect on each others.

Using the path finder dictates using the area model but not necessarily the enemy model.

It was also shown that in general having multiple models tends to result in a lower standard deviation for *inhuman* if not the absolute optimum (In all cases the full learning strategy was within a difference of score value of 1 of the highest performing single model, or single removed model version). The resulting interaction can therefore be described as complex and not easily observed or explained. We could draw the conclusion that the weapon model is responsible for the largest part of the performance of the system but it is more likely that the removal of this component altered the trade-off of the other components against each other, decreasing their overall performance.

11.2.2 CTF

For CTF strategies the following components were tested separately in the top level analysis:

1. Weapon Model

2. Fight Model

3. Enemy Model

4. Area Correlation Model

5. Dynamic Path Finder

6. Distance/Utility Weapon Correlation

7. Flag Approach Route Model

8. Flag Sighting Model

11.2.2.1 Inhuman Skill Level

Figure 11.5 shows the performance changes when removing certain modelling components at the *inhuman* skill level.

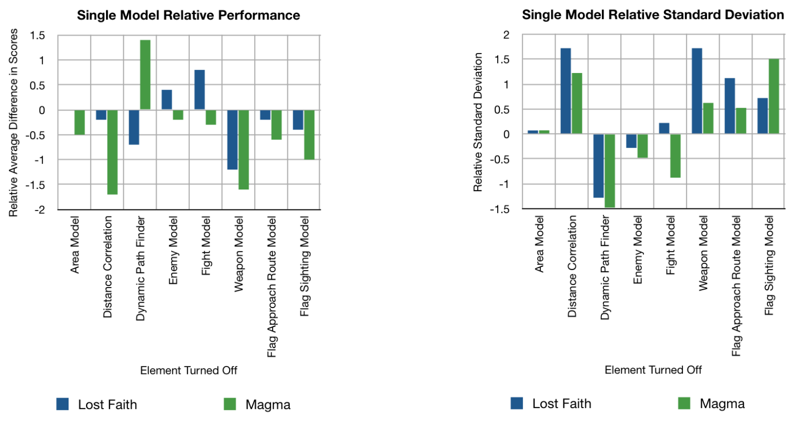


Figure 11.5: Layer 1, Models Off, Component Analysis Results, Inhuman

The most interesting fact about the results shown in figure 11.5 is that removal of certain models causes an increase in performance on each of the two levels. On magma (A level representing a failure in testing) the removal of the path-finder component causes an increase in performance and a lowering of the standard deviation. On lost faith removal of the enemy and fight models caused an increase in performance but with less of a decrease in standard deviation values. This suggests that the path finder was the critical failure point within the system for this game-type. It is also worth noting that removal of the components upon which the path-finder was based did not lead to a reduction in performance, suggesting that the path-finder itself was causing the problem and not its data input.

The conclusion to be drawn from this is that using the path-finder causes a decrease in performance but having made the decision to use it, the models chosen are good enough to base it on.

One possible explanation for the decrease in performance is that CTF is a largely time-critical game. When the flag is being returned it is essential that paths back to base are as efficient for length as possible. The enemy bots have a larger skill at moving on a set path and the benefits obtained from the custom path-finder are not outweighing this ability to move quicker and more efficiently across a given path. As such it is better to optimise for length rather than other goals when carrying the flag.

Figure 11.6 shows the performance changes when using only single models in isolation.

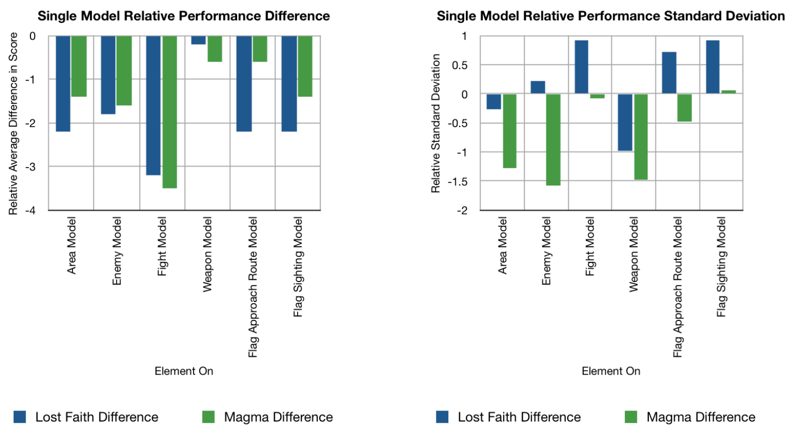


Figure 11.6: Layer 1, Single Models On, Component Analysis Results, Inhuman

The highest average performing single models are the flag route modelling and the weapon modelling mirroring the two mains goals in the game-type, capture flags and kill the enemy.

Overall singular models lead to a decreased standard deviation value. This suggests that it is the combination of multiple models which causes problems for CTF. The fact that the flag approach route modelling is one of the higher single models shows that the path-finder is not behaving satisfactorally when combined with other models as the flag approach route model uses it.

11.2.2.2 Skilled Skill Level

Figure 11.7 shows the performance changes when removing certain modelling components at the ’skilled’ skill level.

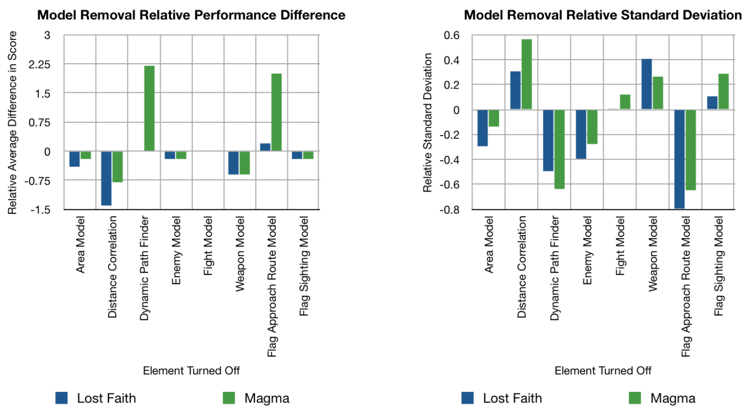


Figure 11.7: Layer 1, Models Off, Component Analysis Results, Skilled

Removal of the path-finder and the flag approach route model both offer an increase in average performance and decreased standard deviations showing these models to be the weak points at this skill rating. This adds further credence to our conclusion that these are the overall weak points for CTF.

Removal of the distance correlation element created the largest drop in performance accompanied with the largest increase in standard deviation pointing to this being one of the stronger elements alongside the weapon model.

Figure 11.8 shows the performance changes when using only single models in isolation.

In single mode the weapon model achieves slightly better performance than the full learning strategy with a slight decrease in standard deviation. The area and flag approach route models are both seen to be the weak points in this section and again both rely on the path-finder suggesting it is weak also.

11.2.2.3 CTF Layer 1 Results Conclusions

The CTF results show that the path-finder and associated models are the weakest points in the system. When we remove these components this causes an increase in performance. When the stronger components such as the weapon model, fight model, enemy model and flag sighting model are isolated they do not achieve the expected level of performance suggesting that, even when removing the path-finder from the system,

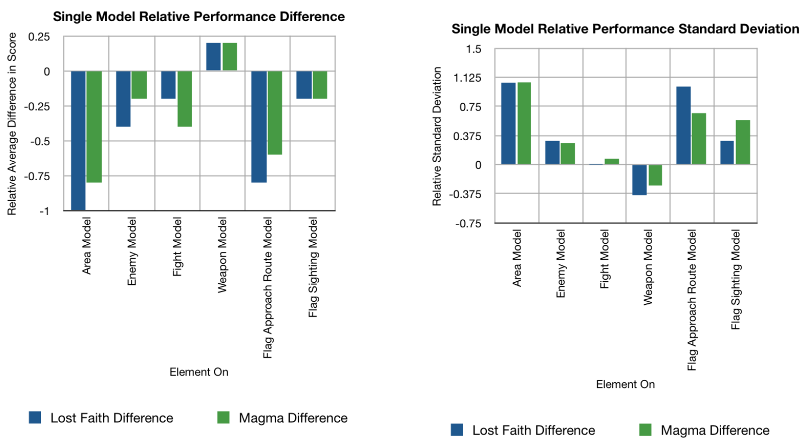


Figure 11.8: Layer 1, Single Models On, Component Analysis Results, Skilled

their relationship is one of complex interaction. The weapon model is the strongest component but seems to require the other strong components to obtain a high performance level.

Trials without the path-finding elements would be interesting for further study.

11.2.3 DD

For DD strategies the following components were tested separately in the top level analysis:

1. Weapon Model
2. Fight Model
3. Dynamic Path Finder
4. Distance/Utility Weapon Correlation
5. Dom Point Approach Route Model

11.2.3.1 Inhuman Skill Level

Figure 11.9 shows the performance changes when removing certain modelling components at the *inhuman* skill level.

At *inhuman* skill the removal of the path-finder, fight model and domination point approach route model leads to both an increase in average performance and a decrease

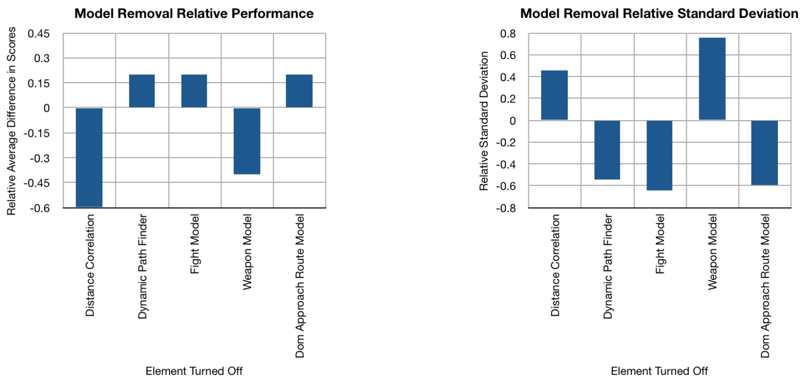


Figure 11.9: Layer 1, Models Off, Component Analysis Results, Inhuman

in standard deviation. This points heavily to the weapon model being the strongest component of the system.

Figure 11.10 shows the performance changes when using only single models in isolation.

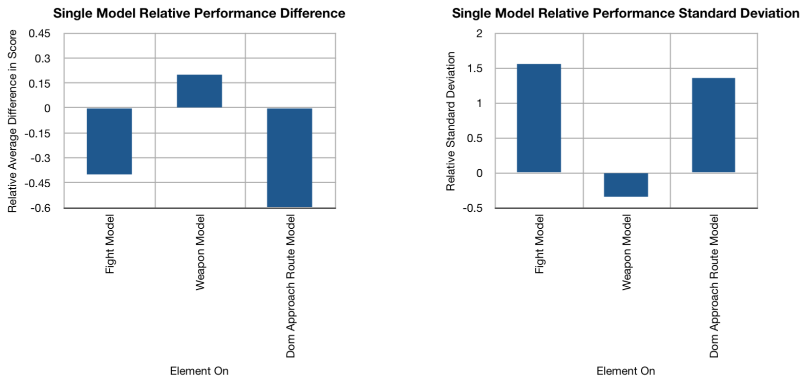


Figure 11.10: Layer 1, Single Models On, Component Analysis Results, Inhuman

The single model results back up the conclusion that the weapon model is the strongest component with the fight model and dom approach models being weak points.

11.2.3.2 Masterful Skill Level

Figure 11.11 shows the performance changes when removing certain modelling components at the *masterful* skill level.

At *masterful* skill removal of any given model leads to a decrease in average performance. In particular removal of the distance correlation component causes an increase

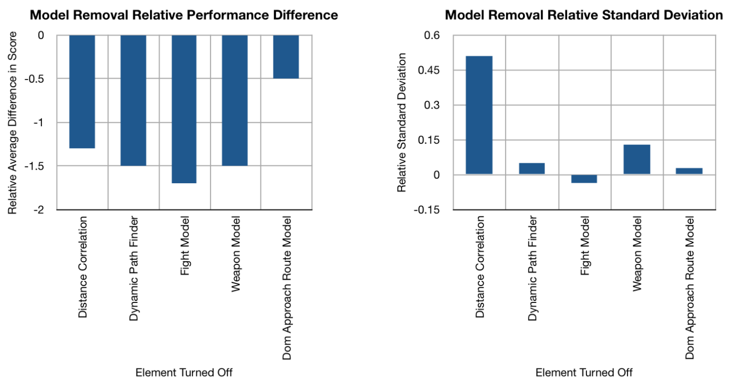


Figure 11.11: Layer 1, Models Off, Component Analysis Results, Masterful

in standard deviation showing that the weapon model is important but must be accompanied by a correct usage policy.

Figure 11.12 shows the performance changes when using only single models in isolation.

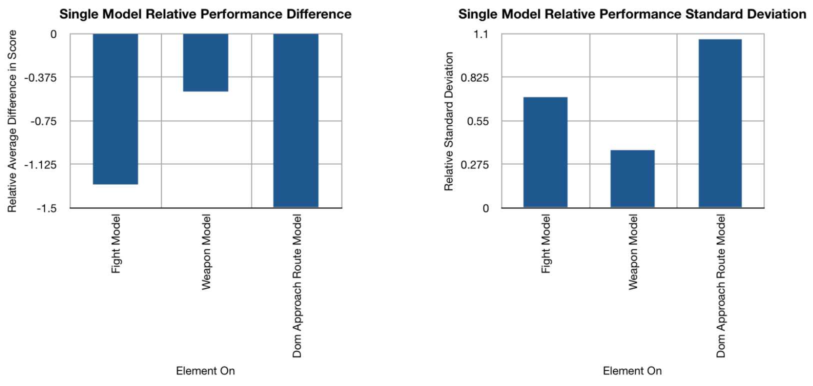


Figure 11.12: Layer 1, Single Models On, Component Analysis Results, Masterful

In isolation the weapon model is the best performer but in all cases a decrease in performance and increase in standard deviation is observed.

11.2.3.3 DD Layer 1 Results Conclusions

DD provides the best result for the arguing that the architecture allows the components to be combined in a modular fashion effectively. When singular components are removed performance decreases and components in isolation do not achieve good performance.

Another interesting result from this game-type is that the change of skill level results in different models being more or less effective. At *masterful* all models are useful but at *skilled* certain models are ineffective. This shows that not only is full system performance mediated by how well the bots do in general but single model performance is also affected.

11.3 Layer 2 testing

Level two of the analysis involved the removal of components from the LCC strategy.

11.3.1 TDM

11.3.1.1 Removal of the dynamic trade-off element

One of the main components of the LCC strategy is the dynamic trade-off of the numbers of exploiting and exploring agents. The best way to test the removal of this is to evaluate the performance of static approaches to the trade-off value. Fig 11.16 shows the numbers of these roles.

Strategy	Explorative	Exploitative
4 Base	4	0
3 Base 1 Hunt	3	1
2 Base 2 Hunt	2	2
1 Base 3 Hunt	1	3
4 Hunt	0	4

Figure 11.13: Static Trade-Off Values

11.3.1.2 Removal of the request change components

In this strategy we removed the element of the LCC which forces the `base`players to request a role change to `hunter` players and had them change whenever they obtained a kill.

```
%%The stuck hunter module, reassigns members to the hunter module
a(hunter,sh)::followMe(sh) => a(hunter,_) <-- engageModule(tdm_weapon_user_v6)
```

```
a(hunter,sh)::null<--engageModule(tdm_weapon_user_v6)

a(hunter,Id)::engageModule(tdm_weapon_user_v6_follower_learning(ID))
<-- followMe(ID) <= a(hunter,sh)

a(hunter,Id)::null<--engageModule(tdm_weapon_user_v5_learning)

%% The baseliner module
a(base,sb)::null<-- engageModule(dedicated_weapon_fight_enemy_area_sampler)

a(base,Id)::null <--
engageModule(dedicated_weapon_fight_enemy_area_sampler_last)
and kill(true) and changeToRole(hunter)

a(base,Id)::null <--
engageModule(dedicated_weapon_fight_enemy_area_sampler_last)
```

11.3.1.3 Removal of the stuck baseline player

Here we removed the *stuck* `baseline` player and reassigned their role to a further *auxiliary* player.

```
  %%The stuck hunter module, reassigns members to the hunter module based on
performance

a(hunter,sh)::engageModule(tdm_weapon_user_v6) <-- gotAKill
<= a(base,ID) then change => a(base,ID) <-- performance(hunter,kill)

a(hunter,sh)::followMe(sh) => a(hunter,_) <-- engageModule(tdm_weapon_user_v6)

a(hunter,sh)::null<--engageModule(tdm_weapon_user_v6)

a(hunter,Id)::engageModule(tdm_weapon_user_v6_follower_learning(ID))
<-- followMe(ID) <= a(hunter,sh)

a(hunter,Id)::null<--engageModule(tdm_weapon_user_v5_learning)

%% The baseliner module
a(base,Id)::engageModule(dedicated_weapon_fight_enemy_area_sampler_last)
and changeToRole(hunter) <-- change <= a(hunter,ID)

a(base,Id)::gotAKill => a(hunter,sh)
<-- engageModule(dedicated_weapon_fight_enemy_area_sampler_last) and kill(true)

a(base,Id)::null <--
engageModule(dedicated_weapon_fight_enemy_area_sampler_last)
```

11.3.1.4 Removal of the follower component

For the final test we removed the component of the strategy which made the hunters follow the *stuck* `hunter` in response to `followMe(sh)` messages.

```
  %%The stuck hunter module, reassigns members to the hunter module based on
performance

a(hunter,sh)::engageModule(tdm_weapon_user_v6) <-- gotAKill
<= a(base,ID) then change => a(base,ID) <-- performance(hunter,kill)

a(hunter,sh)::null<--engageModule(tdm_weapon_user_v6)

a(hunter,Id)::null<--engageModule(tdm_weapon_user_v6)

a(hunter,Id)::null<--engageModule(tdm_weapon_user_v5_learning)

%% The baseliner module
a(base,sb)::null<-- engageModule(dedicated_weapon_fight_enemy_area_sampler)

a(base,Id)::engageModule(dedicated_weapon_fight_enemy_area_sampler_last)
and changeToRole(hunter) <-- change <= a(hunter,ID)

a(base,Id)::gotAKill => a(hunter,sh)
<-- engageModule(dedicated_weapon_fight_enemy_area_sampler_last) and kill(true)

a(base,Id)::null <--
engageModule(dedicated_weapon_fight_enemy_area_sampler_last)
```

11.3.1.5 Results

The graphs below shows the results of the removal of strategic elements.

11.3.1.6 Inhuman Skill Level

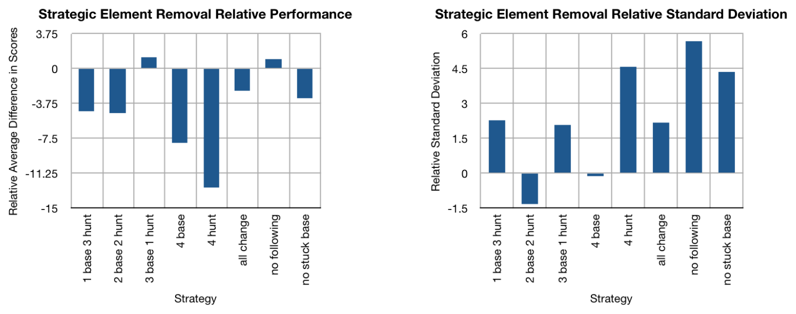


Figure 11.14: Layer 2 Component Analysis Results, Inhuman

We can see from figure 11.14 that strategies with static numbers of different members have varying performance for this level. The 3 base 1 hunt strategy actually achieves higher performance than the standard strategy used but the standard deviation is higher showing a greater level of instability. Having either all explorers or all exploiters does not work. This result is trivial as all exploiters don't learn anything and all explorers is equal to the baseline strategy.

One counter intuitive result is that the `no following`strategy gains a slightly better result than the standard version, though the standard deviation is higher meaning that the strategy is less stable.

In conclusion there is no strategy which obtains a higher positive average difference in score while also obtaining a lower standard deviation, than the standard strategy, via removal of protocol elements.

11.3.1.7 Skilled Skill Level

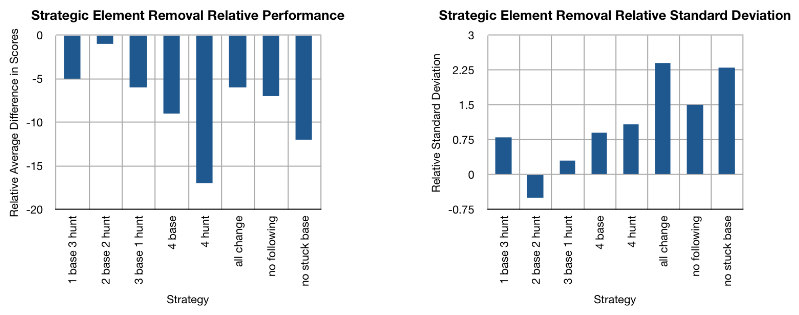


Figure 11.15: Layer 2 Component Analysis Results, Skilled

Most of the patterns from the *inhuman* case remain but none of the static strategies achieve better performance than the `all modelson` strategy. The *stuck* `baseline` player is shown to be largely important but this is expected as they are the solitary fully explorative element. The results also show that removal of either the following component or the environmental feedback mechanism for changing roles leads to increases in standard deviation, supporting our decision to use them.

11.3.2 CTF

11.3.2.1 Removal of the dynamic trade-off element

As for the TDM case, it was possible to alter the balance of the exploitation exploration trade-off and employ some static strategies. Fig 11.16 shows the numbers of these roles.

Strategy	Explorative	Exploitative
4 Base	4	0
3 Base 1 Hunt	3	1
2 Base 2 Hunt	2	2
1 Base 3 Hunt	1	3
4 Hunt	0	4

Figure 11.16: Static Trade-Off Values

11.3.2.2 Removal of the protection component

For the second test we removed the component of the strategy which allowed bots to call for protection when they had the flag.

```
a(attacker_explore_v2,Id)::null<--changeToRole(attacker_exploit)
and performance(attacker_exploit,kill)

%% The weapon fight sampling attacker module

a(attacker_explore_v2,Id)::null<--engageModule(retriever_v1_non_learning)
and enemyHasFlag(true)

a(attacker_explore_v2,Id)::null <-- engageModule(flag_explorer_v2)

%% The fully dynamic attacker module

a(attacker_exploit,Id)::null<--engageModule(retriever_v1_non_learning)
and enemyHasFlag(true)

a(attacker_exploit,Id)::null<--engageModule(flag_exploiter_v1_best)
```

11.3.2.3 Removal of the retriever element

For the final test we removed the strategy component responsible for retrieving the flag when it was taken by the enemy.

```
a(attacker_explore_v2,Id)::null<--changeToRole(attacker_exploit)
and performance(attacker_exploit,kill)

%% The weapon fight sampling attacker module
a(attacker_explore_v2,Id)::engageModule(protector(Location))
<-- protect(Location) <= a(_,_)

a(attacker_explore_v2,Id)::protect(Location) => a(attacker_exploit,_)
<-- engageModule(flag_explorer_v2) and selfLocation(Location)
then protect(Location) => a(attacker_explore_v2,_)<--hasFlag(true)

a(attacker_explore_v2,Id)::null <-- engageModule(flag_explorer_v2)

%% The fully dynamic attacker module
a(attacker_exploit,Id)::engageModule(protector(Location))
<-- protect(Location) <= a(_,_)

a(attacker_exploit,Id)::protect(Location) => a(attacker_exploit,_)
<-- engageModule(flag_exploiter_v1_best) and selfLocation(Location)
then protect(Location) => a(attacker_explore_v2,_) <-- hasFlag(true)

a(attacker_exploit,Id)::null<--engageModule(flag_exploiter_v1_best)
```

11.3.2.4 Results

The graphs below show the results of the removal of strategic elements.

11.3.2.5 Inhuman Skill Level

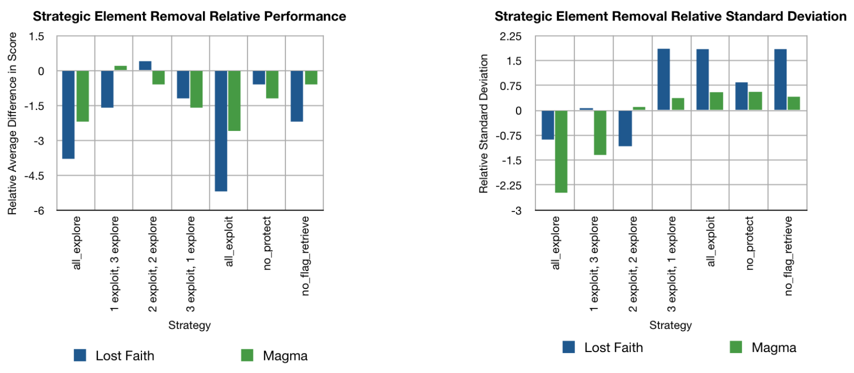


Figure 11.17: Layer 2 Component Analysis Results, Inhuman

The results for *inhuman* are similar to those observed for the TDM. There are static strategies which achieve higher performance for single levels but not for both our tested cases. Removal of the protection and flag retriever components leads to a decrease in performance and increase in standard deviation.

11.3.2.6 Skilled Skill Level

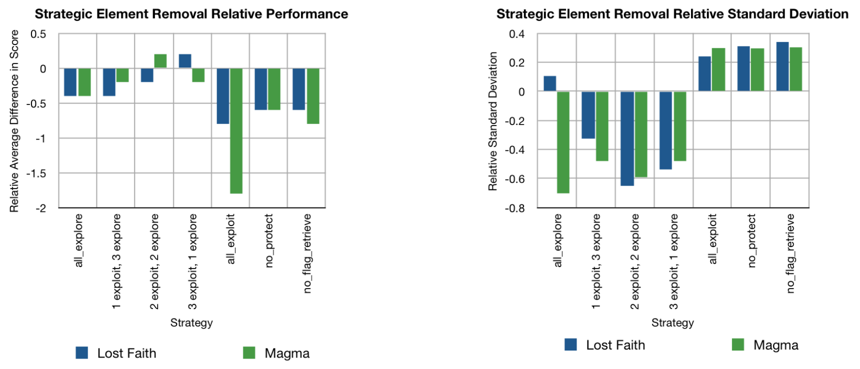


Figure 11.18: Layer 2 Component Analysis Results, Skilled

At *skilled* the standard deviation values are much more stable over the two levels but the average performance results are roughly the same, if again more stable across the two levels. Thus we can conclude that even though the performance is not substantially better using the lower skill level, the performance is more stable.

11.3.3 DD

11.3.3.1 Removal of the distance messaging

One of the key elements of the strategy was the ability to assign roles to bots based on not just their proximity to the domination points but of their team-mates. For the first test we removed this and assigned them static roles regardless of position.

```
a(exploiter,exploit_a)::null <-- engageModule(dom_a_exploiter)
a(exploiter,exploit_b)::null <-- and engageModule(dom_b_exploiter)

a(explorer,explore_a)::null <-- engageModule(dom_a_exploiter)
a(explorer,explore_b)::null <-- engageModule(dom_b_exploiter)
a(explorer,explore_a)::null <-- engageModule(dom_a_weapon_approach_updater)
a(explorer,explore_b)::null <-- engageModule(dom_b_weapon_approach_updater)
a(explorer,explore_a)::null <--
engageModule(dom_a_dedicated_weapon_fight_sampler)
a(explorer,explore_b)::null <--
engageModule(dom_b_dedicated_weapon_fight_sampler)
```

11.3.3.2 Removal of the relative movement

Instead of having the bots assign their role based on messages from others, in this test they used only their own distance to each point as the decision mechanism.

```
a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([Da < Db]) and engageModule(dom_a_exploiter)

a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([Da > Db]) and engageModule(dom_b_exploiter)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([Da < Db]) and engageModule(dom_a_exploiter)
and performance(exploiter,kill) and lastTeamToScore(T) and team(T)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([Da > Db]) and engageModule(dom_b_exploiter)
and performance(exploiter,kill) and lastTeamToScore(T) and team(T)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([Da < Db])
and engageModule(dom_a_weapon_approach_updater)
and performance(exploiter,kill)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([Da > Db])
and engageModule(dom_b_weapon_approach_updater)
and performance(exploiter,kill)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([Da < Db])
and engageModule(dom_a_dedicated_weapon_fight_sampler)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([Da > Db])
and engageModule(dom_b_dedicated_weapon_fight_sampler)
```

11.3.3.3 All exploit

In this test the dynamic trade-off element of the strategy was removed. All bots were set to exploit.

```
a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_exploiter)
<-- domDetails(A,B) <= a(exploiter,_) then domDetails(Da,Db) => a(exploiter,_)
```

```
a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y]) and
engageModule(dom_b_exploiter)
<-- domDetails(A,B) <= a(exploiter,_) then domDetails(Da,Db) => a(exploiter,_)

a(exploiter,_)::domDetails(Da,Db) => a(exploiter,_)<--squaredDistance(a,Da)
and squaredDistance(b,Db)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_exploiter)
and team(T) <-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) =>
a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y]) and
engageModule(dom_b_exploiter)
and team(T) <-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) =>
a(explorer,_)
a(explorer,_)::domDetails(Da,Db) => a(explorer,_)<--squaredDistance(a,Da)
and squaredDistance(b,Db)
```

11.3.3.4 No Dynamic Change

In this test the dynamic trade-off element of the strategy was removed. The explorers could not migrate to exploiters.

```
a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_exploiter)
<-- domDetails(A,B) <= a(exploiter,_) then domDetails(Da,Db) => a(exploiter,_)

a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y]) and
engageModule(dom_b_exploiter)
<-- domDetails(A,B) <= a(exploiter,_) then domDetails(Da,Db) => a(exploiter,_)

a(exploiter,_)::domDetails(Da,Db) => a(exploiter,_)
<--squaredDistance(a,Da) and squaredDistance(b,Db)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)

and prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_weapon_approach_updater)
and performance(exploiter,kill) <-- domDetails(A,B) <= a(_,_)
then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y])
and engageModule(dom_b_weapon_approach_updater) and performance(exploiter,kill)
```

```
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y])
and engageModule(dom_a_dedicated_weapon_fight_sampler)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y])
and engageModule(dom_b_dedicated_weapon_fight_sampler)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_)
<--squaredDistance(a,Da) and squaredDistance(b,Db)
and performance(exploiter,kill) and lastTeamToScore(T) and team(T)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_)
<--squaredDistance(a,Da) and squaredDistance(b,Db) and
performance(exploiter,kill)
and engageModule(dom_v2_weapon_approach_updater)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_)
<--squaredDistance(a,Da) and squaredDistance(b,Db)
and engageModule(dom_v2_dedicated_weapon_fight_sampler)
```

11.3.3.5 All explore

In this test the dynamic trade-off element of the strategy was removed. All bots were set to explore.

```
a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y])
and engageModule(dom_a_weapon_approach_updater)
<-- domDetails(A,B) <= a(exploiter,_) then domDetails(Da,Db) => a(exploiter,_)

a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y])
and engageModule(dom_b_weapon_approach_updater)
<-- domDetails(A,B) <= a(exploiter,_) then domDetails(Da,Db) => a(exploiter,_)

a(exploiter,_)::domDetails(Da,Db) => a(exploiter,_)<--squaredDistance(a,Da)
and squaredDistance(b,Db)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_weapon_approach_updater)
and performance(exploiter,kill) <-- domDetails(A,B) <= a(_,_)
then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
```

```
and prologConstraint([X is A-Da,Y is B-Db, X < Y]) and
engageModule(dom_b_weapon_approach_updater)
and performance(exploiter,kill) <-- domDetails(A,B) <= a(_,_)
then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y])
and engageModule(dom_a_dedicated_weapon_fight_sampler)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y])
and engageModule(dom_b_dedicated_weapon_fight_sampler)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_)
<--squaredDistance(a,Da) and squaredDistance(b,Db) and
performance(exploiter,kill)
and lastTeamToScore(T) and team(T)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_)<--squaredDistance(a,Da) and
squaredDistance(b,Db)
and performance(exploiter,kill) and engageModule(dom_v2_weapon_approach_updater)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_)<--squaredDistance(a,Da)
and squaredDistance(b,Db) and
engageModule(dom_v2_dedicated_weapon_fight_sampler)
```

11.3.3.6 Removal of the staggered layers

In the exploration role there are 3 staggered stages ranging from fully explorative to exploiter. For this test we removed the middle layer.

```
a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_exploiter)
<-- domDetails(A,B) <= a(exploiter,_) then domDetails(Da,Db) => a(exploiter,_)

a(exploiter,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y]) and
engageModule(dom_b_exploiter)
<-- domDetails(A,B) <= a(exploiter,_) then domDetails(Da,Db) => a(exploiter,_)

a(exploiter,_)::domDetails(Da,Db) => a(exploiter,_)<--squaredDistance(a,Da) and
squaredDistance(b,Db)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_exploiter)
```

```
and performance(exploiter,kill) and lastTeamToScore(T) and team(T)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y]) and
engageModule(dom_b_exploiter)
and performance(exploiter,kill) and lastTeamToScore(T) and team(T)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X > Y])
and engageModule(dom_a_dedicated_weapon_fight_sampler)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db)
and prologConstraint([X is A-Da,Y is B-Db, X < Y])
and engageModule(dom_b_dedicated_weapon_fight_sampler)
<-- domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_) <--squaredDistance(a,Da)
and squaredDistance(b,Db) and performance(exploiter,kill) and
lastTeamToScore(T)
and team(T)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_) <--squaredDistance(a,Da)
and squaredDistance(b,Db) and performance(exploiter,kill)
and engageModule(dom_v2_weapon_approach_updater)

a(explorer,_)::domDetails(Da,Db) => a(explorer,_) <--squaredDistance(a,Da)
and squaredDistance(b,Db) and
engageModule(dom_v2_dedicated_weapon_fight_sampler)
```

11.3.3.7 Results

The graphs below shows the results of the removal of strategic elements.

11.3.3.8 Inhuman Skill Level

At *inhuman* the only significant results were those which were trivially expected to be so. Removal of the elements concerning the decision about which point to go to did lead to a higher standard deviation but no change in the actual performance showing that they were responsible somewhat for stability of performance rather than actual performance.

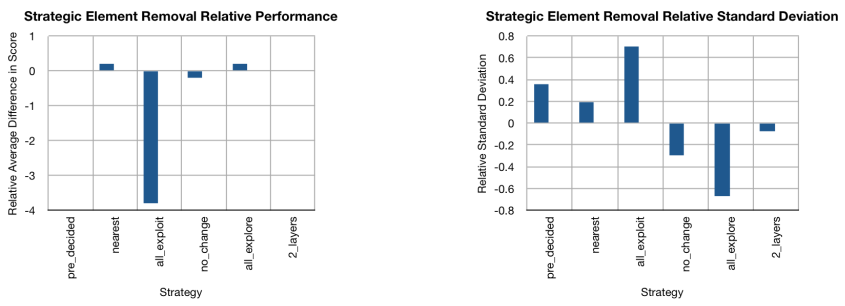


Figure 11.19: Layer 2 Component Analysis Results, Inhuman

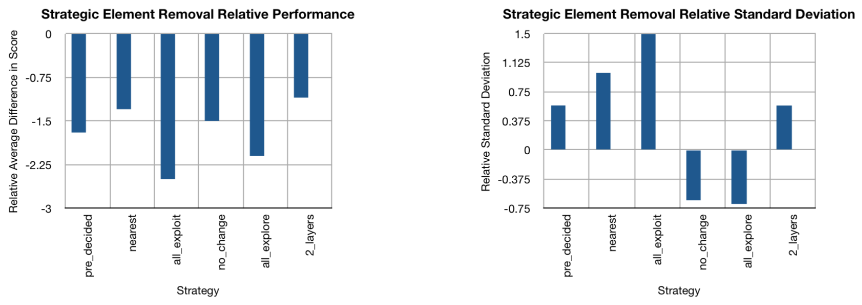


Figure 11.20: Layer 2 Component Analysis Results, Masterful

11.3.3.9 Masterful Skill Level

At *masterful* removal of any single strategic component led to a decrease in performance showing that all were working well at this skill point and all were partly responsible for performance. This is good as this was exactly what we were aiming for.

Removal of the ability of the agents to change from explorers to exploiters led to a decrease in standard deviation but this was negated by the decreased performance. This shows that although we gain a performance increase from this adaptive mechanism there is a cost in stability.

11.4 Overall Conclusions

The following is a brief summary discussion of the two layers of analysis.

11.4.1 Machine Learning Techniques

One of the most startling conclusions from the analysis is that the models which are the most general and contribute to performance across all three games types perform consistently highest. The weapon model in particular seems strongest. These stronger elements also seem to combine best in the architecture but the combination is complex and not easily evaluated.

We also saw that certain models are more or less effective at different skill settings. This is expected as certain models have different prerequisite amounts of learning required to function and mediating rates of learning with performance will change the amount of learning that is performed. Running a model with bad information is likely to have a negative effect even if the overall performance is satisfactory.

We saw that for CTF the cost of taking an inefficient route with regards to path length is not acceptable to achieve successful performance and that removing these non-optimal routes increased performance.

Overall we can conclude that in general elements concerned with path finding contributed least to performance and elements concerned with confrontation (fight,enemy,weapon) contributed most showing that our decision making models were of higher standard than our geometric approaches.

11.4.2 Strategy Components

The strategic component results are slightly different from those tested in layer 1. The more general components such as the reinforcement learning control elements seem to be responsible for play stability rather than specific performance, which fits with our intuitions about what they should be doing.

The more specific game-type components are responsible for increase or decrease in average score.

Again all components are mediated by enemy skill level.

Chapter 12

Efficiency Testing

In this chapter we present an experiment to show the efficiency of the system. This is to suppliment the results we have already presented, allowing some insight into the time taken for useful adaptivity to occur rather than simply whether it occurs or not.

Our notion of effeciency is based on how fast the system converges upon a state where the maximum amount of exploitation is occuring. Because this exploitative state is mediated by performance, this gives a measure of when the system's performance has reached its most effective level.

This also allows us to determine how effective and stable the online learning process is. This adds credence to our claims that the system straddles the competing goals of stability across game matches and adaptability to different conditions.

12.1 Experimental Details

The experiment we perform is to run the system over 10 trials on 5 levels and measure, on average, at what point the fully exploitative state (3 exploiting bots, 1 explorer) is reached and the average percentage of the trial which was spent in this state. We also present the average difference in scores (as per the full evaluation trials) in order to show some comparison measures with the results presented in the full system evaluation. The enemy used were the standard in-built bots set at the "skilled" skill level. The game-type chosen was TDM and all of the models were enabled. The levels chosen were, Ironic, Leviathon, Squader and Desolation. These were chosen because they were the levels in which a significant increase in performance was observed, in the full system evaluation, using the learning strategy over the baseline. This decision was made because these are the most likely to show convergence, oweing to our convergence criteria being performance mediated. We also present a result for the level Sulphur because this was a level in which we performed significantly below the baseline in the full system evaluation. As such this gives us an insight into what happens to system efficiency when we fail to perform effective learning. The strategy used was our chosen strategy from the full scale evaluation. Only the learning strategy was tested because the baseline did not give any information about convergence and also, we already have the baseline results from the full system testing, as a control.

This is not designed to be an exhaustive experiment but instead to show roughly how efficient the system is on average.

12.2 Results

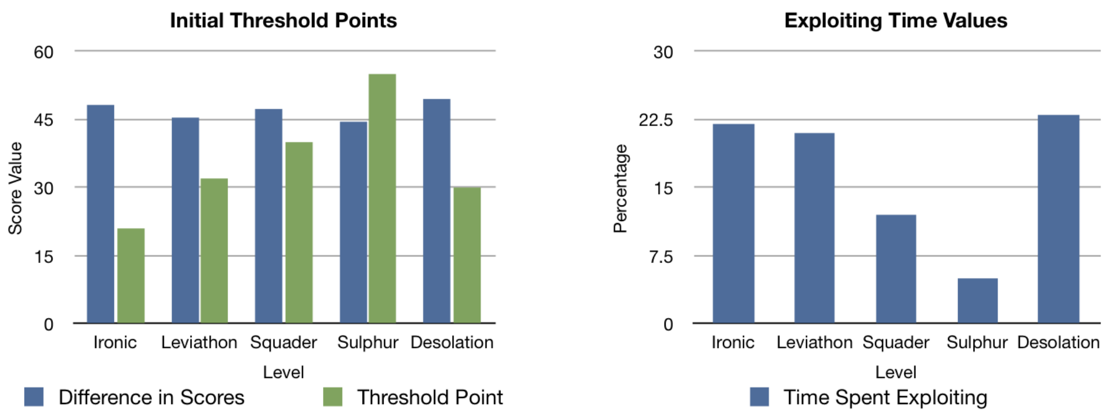


Figure 12.1: Effeciency Results

Figure 12.1 shows the initial score, in the match, at which the fully convergent state was reached (on average) in our trials alongside the difference in scores between our learning bots and the in-built bots. It also shows, in the right hand graph, the percentage of the match time spent in this exploiting state.

The results show that in most cases convergence is not realised until late in the match (the goal score in these matches was 60 and the overall average across the levels is between 45 and 50). Generalising we can conclude that the initial threshold point is normally over half the winning score for this skill level. This said, it is the percentage result which is most interesting.

In cases where our learning strategy performed statistically higher than the baseline strategy, in the full system evaluation (Ironic,Leviathon, Squader, Desolation), we observe that the system spent roughly 20-25% of the match in a fully exploitative state. Correlating this with the initial thresholding points, this means that once the fully exploitative state is acheived, it tends to be stable for the remainder of the match.

In the case where the learning stategy was statically worse than the baseline strategy (Sulphur), this value drops to 5% . This is indicative of the fact that our system is very reliant on convergence. If we spend roughly 20-25% of the match in an exploitative state then we are likely to win the match.

12.3 Conclusions

The results show that although our system is not the most effecient in terms of time spent getting to an exploitative state, it is stable when it gets to that point. They also show that being in the exploitative state is a position powerful enough advantage to win the match and that not being in this state for at least 20% signifies that the learning is not likely to be effective.

Our system is reliant on convergence, if the majority of our team do not convergence on an exploitative state then this leads to bad performance. This is no real surprise as the convergence triggers in our strategy are based on performance. Convergence leads to good performance; if we converge on a steady exploiting state then we win the match. Again this is nothing profound due to the nature of our system convergence criteria but it gives a bi-implication showing a useful indicator for reference.

Combining these results with those from the component analysis would be an interesting future experiment to show the interaction between individual models and the initial thresholds, but this is beyond the scope of this thesis.

Chapter 13

Exploration Versus Exploitation

Above and beyond simply having a respectable performance in a complex domain we also wanted to show that the architecture could tackle a more concrete task in an interesting way. Because our system was inherantly multi-agent in nature we could tackle the reinforcement learning problem of exploration versus exploitation from a multi-agent perspective. Our approach was to allow the roles of individual members of the team to be more or less exploitative or explorative. This meant that we did not have to split the time spent performing actions within any single agent.

13.1 Background

Reinforcement learning[105] is the study of learning systems in which an agent is situated in an environment, allowing for feedback from its actions to guide its further learning. Often this notion of feedback can be utilised for better learning if its effects are both informed and controlled. One particular concern for reinforcement learning is the idea of exploration versus exploitation. Exploration is time spent performing actions, mainly at random, to judge their effect and introduce randomness and variation to the learning. Exploitation is time spent performing actions as set out in a learned policy. Typically this process is performed during large numbers of off-line prior trials. Different trials are either exploration or exploitation trials. In some cases off-line learning is not a possibility, typically where the domain is time or resource limited in some way or large numbers of trials are infeasible.

Traditional numeric approaches to reinforcement learning[105, 12] include options such as Q-learning and temporal difference learning with ε-greedy being a typical basic approach to exploitation versus exploration. These usually revolve around predicting the utility of certain state-action pairs and as such are applicable only to problems where the domain lends itself to such a representation.

13.1.0.1 Multi-Agent Reinforcement Learning

The reinforcement learning idea becomes even more interesting when we consider multi-agent domains. Sadly there is not much concentration on this issue within the machine learning for video games world.

> "Multi-agent reinforcement learning has not often been used for computer game non-player character development"[18]

Bradley and Hayes' paper on group utility functions[18] provides some idea of the types of approaches to multi-agent reinforcement learning but even then this focuses on game theoretic notions and the traditional reinforcement learning notion of state-action pairings.

In general the approach to multi-agent reinforcement learning is to have multiple agents provide separate observance points for a policy learning algorithm such as Q-learning. The idea is that with multiple agents learning is faster, but the general approach of off-line learning remains the same. Because our system deals with on-line learning we take a different stance on the issue of exploration versus exploitation. Our idea is that instead of having many trials and setting all the agents to be explorative or exploitative for any given trial, we split the team to be more or less exploitative in nature at the individual's level. Each agent has their own role and as such performances a different task. This allows for more or less explorative strategies for each agent and also allows the strategy designer to pick which models to exploit or explore with in each bot. This then presents a real-time trade-off between exploration and exploitation[105, 54, 110] which we show can be altered in real-time and tapered in response to current performance to provide better performance.

The idea of multi-agent reinforcement learning as found in our work is similar to the way in which reinforcement learning ideas of exploration verses exploitation are performed at real-time, during a match, found in [82]. In this paper they are used to taper an evolutionary strategy towards better performance, instead we use them to deal with the problems found with on-line learning in general for unspecified ML techniques.

Some key work in this area has been performed by Peter Stone. Stone's work is largely from the perspective of ways of combining reinforcement learning techniques with machine learning to create more robust algorithms [53]. In his review paper he details a number of learning techniques and how he feels that these could be combined into a system for controlling multiple agents.

> "Such agents will need to be able to learn, both in order to correct and circumvent their imperfections, and to keep up with a dynamically changing world."[53]

He states that systems dealing with complex problems will need to provide algorithms which are applicable to alternative domains. The work presented here provides not an algorithm but an architecture with this adaptability in mind.

Stone's work also draws on the importance of learning quickly in domains where time is critical when considering sample efficiency [46]. His, single-agent, approach is to build a model of the domain early in the life cycle of the instance and then simulate runs in this model to increase the quality of the RL policy, beginning exploiting only when the policy is of sufficient quality. This is interesting mostly because it involves building a model of the environment and then applying reinforcement learning over that. The difference between this and our work is in two areas, number of agents and then what the model is used for. Our multiple models are assumed to each be incomplete and we continue to tweak them and use them directly. Stone uses his model for simulation purposes assuming it to be an accurate model of the complete environment sufficient for creating an RL policy from.

We mirror Stone with our approach to exploitation versus exploration but instead adapt this idea to a team situation and show how measurement of policy effectiveness can be achieved using heterogeneous agents with more or less exploitative roles.

13.1.0.2 Reinforcement Learning for FPS Games

The following quote sums up the rough state of play regarding reinforcement learning for domains such as ours:

> "To our knowledge only Vasta et al[116] has investigated RL in FPS game"[64].

The reason for this is that people are still largely concerned with reinforcement learning formulations explicetly based on state-action pair representations, and policy learning. Our argument is that by considering the learning process as a black box and only modulating the on-line trade-off of team roles in the abstract, the need for a state-action pair representation can be avoided. We show how our architecture can help to achieve this.

In chapter 1 we made the following two claims:

- Communication can be used to control the learning rates of a team of multiple bots while they act in the domain environment
- It is more effective to have bots in multiple different, adaptive, roles with varying degrees of learning and exploitation then to have single homogeneous bots learning locally using a strategy such as ε-greedy

We have presented a system which which meets the first claim but to what extent does it deal with the second? The following is an analysis of the approach to exploitation versus exploration used which aims to determine to what extent claim 1 was met and also whether claim 2 is met.

The exploration versus exploitation trade-off [105, 23] is tested against a single approach to the problem. This consisted of testing this approach against the in-built bots and comparing this with results from our learning strategy. TDM was used because this offered the highest positive significance margin from testing. Observations from the trials during testing also suggested that it was offering the best environment for the basic learning techniques to operate with the largest number of confrontations to work from. We also saw from the component analysis that the weapon model proved the most detrimental to performance when removed. As such these trials used strategies involving only that model to try to test how effective our approach was versus a simple classical approach with only the one, albeit simple, model. The tests were performed over 5 levels.

13.2 Reinforcement Learning and the Architecture

Although it is necessary to deal with the issue of reinforcement learning, with any learning agent situated in an environment, it also serves as a good test of the architecture's ability to deal with a problem that is likely to occur in similar domains. As such it is an example of how the architecture not only allows us to produce a novel solution to a known problem, but also how we can also produce an effective one.

13.3 Formalising our approach

Our approach was two-fold. Firstly to allow certain team members to be more exploitative or explorative in relation to their role and secondly to allow these roles to change in response to the performance of the exploitation agents. Note that this necessitates that performance is also mediated by the enemy skill, indirectly.

Our approach can be categorised as a bootstrapping[105] approach as the current state's value is often assessed based on the estimated value of neighbouring/future states. It is not a backup strategy as no information about eventual end state values between trials was used to reinforce current state value.

13.4 Single Agent Reinforcement Learning

The classical approach to reinforcement learning is to have a single agent spending a certain percentage of their time exploiting and the rest exploring. In our case, because our bots were formulated as reactive agents, we had to work on a per death basis. The way we achieved this was to assign the agent an exploitative or explorative role after each death, based on maintaining a ratio of game lives spent in either pursuit.

Because we only used one model in this section it was easier to design a homogeneous module for all the bots which performed a single agent policy.

13.4.0.2.1 Description

```
1.IF died on previous cycle{
2. Set A to random between 1 and 100
3.}
4.IF A < 26 {
5. Select weapon with least magnitude of utility
6. IF Weapon not held {
7.          Use standard path finding to run to weapon pickup
8.          Select It
9. }
10. ELSE{
11.     IF visible enemy {
12.          Engage Enemy
13.          update weapon model
14.     }
15.    ELSE{
16.         play randomly
17.    }
18. }
19.}
20.ELSE{
```

```
Switch to current best weapon held using weapon model
IF Enemy Visible{
         Move towards enemy shooting a them
}
ELSE{
     use distance/utility correlation to assess most attractive weapon
    IF most attractive weapon not held{
        Use standard path finding
        to move to most attractive weapon pickup location
    }
   ELSE{
        play randomly
   }
}
}
```

13.5 Testing

To test the overall effectiveness of our approach we tested it against our single agent strategy, not only because it was the simplest of the reinforcement learning approaches, and easiest to explain and therefore analyse, but also because other approaches such as temporal-difference learning and q-learning did not lend themselves easily to the game environment chosen. Most rely on offline learning of parameters over multiple runs which, as has already been discussed, was not our intention. We could have performed a similar change as in the single agent case and performed the updates in an online fashion treating a single run as one life but this did not yield enough runs for q-learning to be effective in this domain.

13.6 Results

13.6.1 Individual Level Analysis

At an *inhuman* skill level there is no significant difference between the two methods. The single agent case is slightly higher on average but the P-values show this is likely due to chance.

The *skilled* results show our strategy performing significantly better than the single agent case. In 3 of 5 levels there is a statistically significant increase in performance. The overall pattern of P-values is similar to the *inhuman* skill level but this time they are relatively higher.

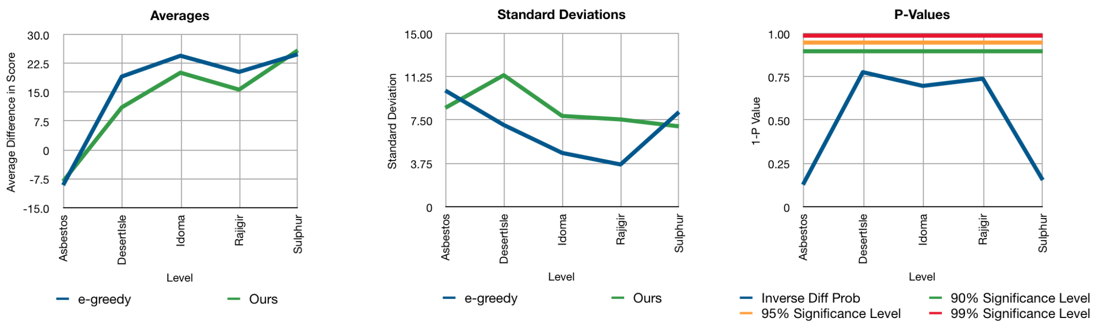


Figure 13.1: Inhuman Skill Individual Level Comparisons

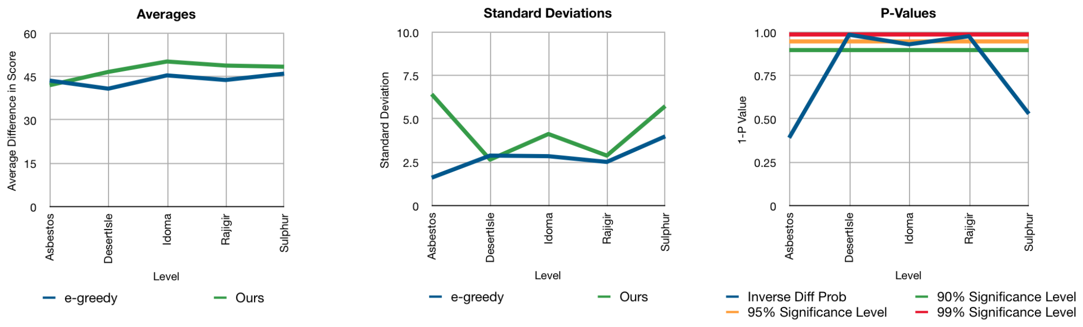


Figure 13.2: Skilled Individual Level Comparisons

13.6.2 Cross level Analysis

	Analysis	
	Single Agent	Ours
Average Difference in Score	16.04	12.8
Standard Deviation	14.54	14.47
Size of set	25	25
99% Confidence Interval	7.494	7.455
T-Value	**0.789**	
P-Value	**0.4337**	

Figure 13.3: Inhuman Skill Cross Level Results Analysis

The cross level analysis tables back up the conclusions from the individual level sections that at *inhuman* skill level there is no significant increase in performance using our method but at the *skilled* skill level there is. In general the standard deviation values are much less in the *skilled* case, for both methods, suggesting that convergence is occurring more readily.

	Analysis	
	Single Agent	Ours
Average Difference in Score	44.12	46.56
Standard Deviation	3.23	4.99
Size of set	25	25
99% Confidence Interval	1.66	2.57
T-Value	**2.051**	
P-Value	**0.046**	

Figure 13.4: Skilled Skill Cross Level Results Analysis

13.7 Conclusions

The results show that in the *skilled* case our learning approach is significantly better than that of a basic single agent model. The results also add further evidence that our approach has a narrow enemy skill level setting at which it performs highest.

We have shown that a team of heterogeneous bots can perform adequately the learning functions of a team of homogeneous single bots without the need to split every bot's time between exploring or exploiting.

It is viable that this approach could be extended to other video game domains where the machine learning techniques for those domains are similarly located at the lowest layer in the architecture. It is possible that in other domains with a longer life-span for agents that q-learning or other offline approaches could also be tested against our approach.

Part IV

Conclusions

Chapter 14

Alternatives

Our approach to the problem is not the only method available. To show this we discuss here a few alternatives. We did not implement any of these methods but they are viable alternatives and some of the potential advantages and disadvantages are discussed.

14.1 Knowledge Transfer

As discussed early in the thesis a common way of dealing with machine learning in a situated complex multi-agent environment is to employ the use of knowledge transfer[90] and use information learned from trials to influence further trials. This increases the likelihood that the problem would end up being formulated as a classic reinforcement learning problem.

It could possibly allow a larger learned map of the levels and their particular quirks, to be obtained by comparing issues such as level design and weapon placement. This could be used, for instance, to tweak our level model to better refine the expected polygons. It could also be used to discover known walls and improve the path finding model so that the two-level hierarchy was less important.

This cumulative learned data could also be used to construct a model of the domain on which further trials could be simulated in much the same way as in [46]. The improvement would be that we wouldn't have to rely on the model and could instead build it in the background over time and trials, gradually improving performance with the hope of hitting an exponential peak performance level.

It may even be the case that the levels as a whole would form a hyper-dimensional space for exploration of a better generic strategy which takes into account all of the levels rather than allowing environmentally mediated learning elements at each part of the strategy. I believe this is unlikely as it would have reduced the problem to a search space over strategies and the complexity of the domain does not seem to suggest this kind of observability.

14.2 Machine Learning of Full System Parameters

One particularly nice way to deal with the problems of removing additional constraints, and further abstracting plan formation to lower levels, is take all of the system parameters such as health, ammunition levels, etc and plug these into a neural net or other such machine learning mechanism. We then use the level of attainment of the goals of the bots as a feedback mechanism for training within the game environment and try to learn a mapping directly from scenario to decisions and movement commands.

This idea is extremely unlikely to work unless there exists some simple scripted behaviour which would be obvious from the outset as being acceptable play, unless the net were very complicated but also simple to train. Even the in-built game bots, who's play is largely static, require a very large amount of UTScript to achieve performance which is regarded as acceptable to the player (most people still prefer to play against other players if possible[87]).

This idea of a single machine learning mechanism for the entire decisions process fits with a lot of the literature on multi-agent machine learning and adaptations to reinforcement learning techniques for complex domains but authors are now beginning to focus on more multi-technique approaches [57].

14.3 Plan Re-Organisation

Early in the thesis we showed some of the problems that result from trying to specify all possible scenarios within a logical strategy with an inflexible resolution based inference procedure. One possible way to tackle this problem could be to re-organise the LCC clause order to work better for individual levels based on dynamically learned information. We might then hope to be able to read off the strategy back out from the end of the level to see what the eventual outcome was, gaining a better understanding of the priorities for that scenario. A simple formulation would amount to nothing more than plan re-ordering which would of course make it very easy to read the result. The main problem with this is that any plan would be dictated by the original input plan and would be highly constrained by this. The work in [98] gives a detailed example of how this can be achieved with small pieces of modular script.

14.4 Genetic Programming for fixed constraint plans

Another option is to have a more constrained lexicon and run a genetic programming process over it. There is always an upper bound on what can be achieved with genetic programming when the lexicon becomes big enough to write things which make sense syntactically but not semantically (This is also comparable to the point at which tree based GP yields solutions which break the key properties of parent unity and respect). The reactive LCC presented is small enough that it may be possible to formulate this in such a way that rules could be built into the genetic process to ensure coherence and usefulness. This would require a large amount of constraints on the way in which the LCC is organised and what is a valid strategy. It is possible this reorganisation and constraint tagging could be achieved automatically by intelligent parsing.

Chapter 15

Further Work

In this chapter we close by discussing a few possible further work avenues generated by the work presented in this thesis.

15.1 More Complex Techniques

It would be interesting to see how the over-arching reinforcement learning issues apply when using more complicated machine learning techniques such as support vector machines and more focused Gaussian modelling procedures. It would also help to provide some evidence of what level of architecture modularity is present within the system and how flexible our paradigm is for other domains.

Further to this, the results presented in this thesis were not optimal. The exact level of improvement which could be attained via more powerful modelling techniques is not suggested from the conclusions drawn as the results only pertain to the included models. As such it would be advantageous to gain an understanding of how much system improvement could be gained from techniques which were significantly better than those used.

One option for performing this could be to incorporate the bank of techniques used in a system such as WEKA[120]. Our intention in using simple techniques was to allow a system to be prototyped and evaluate the architecture and show that it was effective. Having done this we can then add these more complex techniques to improve performance and a system such as WEKA should not be overly difficult to integrate as it is written in Java.

15.2 A Universal Module Script

One of the weakest elements of the architecture, in terms of modularity, was the behaviour modules, due to their lack of portability to other domains outside of UT. One option to deal with this is the development of a universal module script. As discussed previously this would require some middle-ware to work correctly in other domains but with a careful script specification this is achievable. One option is something akin to the notion of code snippets used in Ken Kahn's work on educational systems[55].

It may be beneficial to consider typical scripting languages used in subsumption architectures given the comparisons to them which our work draws[47].

In relation to self defining middle-ware an adapted ontology matching strategy might allow matching to other domains but this would made easier if a universal module script was developed which was domain independent while having latching predicates. Latching predicates are key elements of the code, resembling pseudo-code keywords, which have to be defined in order for a universal script to become instantiated to a particular domain. At present we have shown one approach to the problem but it would be more useful if a generalised framework could have been created and shown to have plug and play characteristics. This would definitely have made the claims of modularity and the abstract contribution of the architecture more compelling.

15.3 More Efficient Code

The tested system code wasn't optimised (with the exception of a few problem instances where un-optimised code was a large bottle-neck). This meant that performing larger numbers of tests was infeasible as a respectably powerful computational system was needed to run the bots and UT in their current state. It would be advantageous to have an implementation which was portable to smaller scale platforms, and also distributed, allowing multiple platforms and machines to be utilised to perform more in depth testing.

15.4 LCC Enhancements - Code Repetition

In most modern programming and scripting languages there are methods of packaging large parts of code, where used frequently, into methods, procedures or predicates which are called instead of repeating local code in our case. This is made slightly more difficult because of the particular method of interpretation used in the LCC implementation, along with the unificative nature of the of the reactive system engine, but there should be a way to package code up as per the following example:

```
a(explorer,_)::squaredDistance(a,Da) and squaredDistance(b,Db) and
prologConstraint([X is A-Da,Y is B-Db, X > Y]) and
engageModule(dom_a_dedicated_weapon_fight_sampler) <--
domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)
```

Would be repackaged as :

```
a(explorer,_)::dist(Da,Db) and
engageModule(dom_a_dedicated_weapon_fight_sampler) <--
domDetails(A,B) <= a(_,_) then domDetails(Da,Db) => a(explorer,_)

dist(Da,Db):-
squaredDistance(a,Da),
squaredDistance(b,Db),
prologConstraint([X is A-Da,Y is B-Db, X > Y]).
```

This could be achieved in principle by passing any commands not found in the gamestate to the Prolog LCC file and including code definitions. This does assume that Prolog is used as an interpretation engine but this assumption is made with the `prologConstaint` constraint anyway.

15.5 Vehicles

At present none of the standard gamebots game-types have vehicles in them. The assault levels in the single player mode do. Our hope is that the bots could determine how they can perform better using these vehicles during game matches. This is difficult but seems viable giving some of the results obtained with simple models for other environment centric concepts.

15.6 Manual Operators - The Human Touch (A fun aside)

It is theoretically possible to include a human operator in a number of simple roles in a team of bots. These external agents can then engage in team activities if their inputs can be formatted in a bot readable way. This shows a branching of this work into experimental human-computer interaction, which was not dealt with within this thesis.

The easiest way to implement this is to have the human operator take a commanding role, issuing commands to the bots such as target locations and localised weapon preferences.

And alternative method could be something similar to that presented in [32] whereby a human operator presents a skeleton policy or strategy for an adaptive agent who then flesh out the details of this with their chosen learning method. In [32] the method chosen is a genetic evolutionary process but this need not be the case. We saw in our section dealing with fight modelling that even simple rules can provide an effective compliment for a well defined learning process. Our rules there were general cases that we felt should influence the learning but it is not difficult to visualise that a human operator with detailed knowledge on a particular scenario or situation, in which the bots are likely to play, could input this data as a policy as an injection case for the learning mechanism.

Chapter 16

Conclusions and Contributions

In Chapter 1 we made the following claims:

1. Stable cross instance performance of a (statistically significantly) higher than baseline standard, in a complex, real-time, adversarial domain can be achieved using combinations of machine learning techniques, to learn about dynamic environmental factors within the domain.

2. A layered heterogeneous architecture, based on behaviour modules, is an effective way to combine these learning techniques to increase performance

3. Communication can be used to control the learning rates of a team of multiple agents while they act in the domain environment

4. Communication can be used to control the behaviours of a team of agents using a simple, easily understandable, fixed LCC plan

5. It is more effective to have agents in multiple different, adaptive, roles with varying degrees of learning and exploitation then to have single homogeneous agents learning locally

This chapter draws to a close all of the results in this thesis and ties together the evidence for these claims. It also draws to attention any contributions which have been made along with the wider context for these.

16.1 Main Contributions

The following are the two largest contributions:

The Architecture A layered architecture for integrating multiple machine learning methods into a system for controlling teams of agents with communication and adaptive behaviour in a complex virtual environment

The Reinforcement Learning Approach A method for performing exploration versus exploitation on a team level, directly considering the environment and environmental feedback, without time-sharing the activities of any individual team member

16.1.1 The Architecture

Evidence for claims 1 through 4 is in the form of a functioning architecture and the performance which it achieves. The architecture presented integrated machine learning techniques with multi-agent strategies in such a way that certain fundamental reinforcement learning problems could be approached without the need to sacrifice control and predictability at an agent level. This was achieved using a layered heterogeneous subsumption based fixed architecture based on behaviour modules. It was shown how this architecture could be used over multiple game-types, with multiple interacting machine learning techniques and how it could be adapted to other domains. The architecture presented acheived online learning with an observable predictability due to its fixed plan/architecture.

The modularity within the system permitted the use of multiple different learning techniques and behaviour modules which could be switched on and off easily. These could also be replaced with alternative domain specific techniques if required.

The co-ordination side of the system gave an effective way of assigning roles to individual bots and a direct approach to the reinforcement learning issues dealt with. It also permitted configuration of the behavioural modules to create teams of bots which adapted. In the DD scenario it was used to give a team of un-assigned bots a set of roles and responsibilities based only on their relative location and the communication of this information.

The weak points in the system were the particular machine learning techniques used and the strategy/protocol design used rather than a failing at any point in the architecture. Both of these issues could be dealt with by a larger scale development team alongside better knowledge of the specific target domain. A more worrying issue is that the machine learning techniques did not complement each other as well as expected. The building block effect of steady increase with added machine learning techniques did not occur in all game-types and as such this does point to a failing of the architecture.

Because the machine learning techniques were geared towards certain goals which were, in some cases, counter-balanced against each other, making decisions about which goals to prioritise had a drastic effect on the overall performance. The choice of learning elements could have benefited from much better decision procedures and goal selection mechanisms. This could possibly be achieved using an automatic design produced within the behaviour module script but this sacrifices some of the predictability that was sought with the architecture.

On top of the architecture itself we showed how to go about performing basic component analysis on it to analyse failings and weaknesses yielding the following strategy:

- Isolate machine learning components and assess the effects of them, both in isolation, and the effects of their removal;
- Isolate strategy components and assess the effects of their removal.

16.1.2 The Reinforcement Learning Approach

Claim 5 is met by our approach to the reinforcement learning problem and the results obtained using it. Having multiple-team members offers more flexibility to the problem of reinforcement learning. Team based adaptive reinforcement learning can be used to bypass some of the issues that normally dictate a formalised machine learning treatment of complex domains. This approach can also be used in an on-line manner during game execution, without the need for multiple trials, to achieve acceptable results.

Treating the team as a single entity and giving each role a more or less exploitation nature is novel for first person shooter video games. The traditional approaches to this problem usually have a split within a single agent and there is no sense of the wider roles of a team of players learning in a way as structured or explicit as we have presented. The dynamic trade-off of this ratio of exploiters to explorers is also a new way of dealing with this problem and a powerful extension to the split team concept. We showed that that this way of treating the problem is amenable to a well known technique and also demonstrated the ease with which our approach allows intuitions about the domain to be integrated into the team learning dynamic.

This also demonstrated the use of the architecture to provide a more meaningful, performance mediated, solution to a known problem in a specific environment.

16.1.3 Dynamic Plan Execution

It has been shown that dynamic plan execution can be approached not only from the traditional avenues such as replanning and plan refinement as presented in the planning world but also from the angle of allowing flexible execution at the individual agent level while maintaining a fixed plan. Treating the system as a heterogeneous subsumption architecture where the plan is only the top level allows the lower levels to use machine learning to alter plan execution on a local agent based scale.

This approach allows differences between multiple instances within a domain to be smoothed over with simple machine learning techniques.

16.2 Narrower Contributions

On top of those above we also make the following, less significant, contributions to narrower fields.

The Constrained Gaussian Model A method of constraining a Gaussian mixture model to specific areas using an adaptation to the EM learning algorithm

Utility Weighted Dynamic Path-finding An adaptation to the standard A* path finding algorithm which allows it to take into account the utility of specific areas of 3D terrain.

16.2.1 The Constrained Gaussian Model

The constrained Gaussian mixture model gives an intuitive way to force means and covariances to be, not only within certain bounds, but within specific areas of the space assumed to be useful. Although not entirely novel, this idea is novel in its application to the domain of levels and multi-agent learning.

Engineering Box

Ultimately it proved too complicated to be useful in our domain. This was mainly down to our desire to learn on a single trial basis rather than facilitating knowledge transfer between trials. With this it is feasible that the model could be more useful.

16.2.2 Utility Weighted Dynamic Path-finding

Although the path finding algorithm presented was not one of the more important elements of the system the individual results presented in the modelling section suggested it could be used to advantage in other systems. Admittedly this argument would be more compelling if we could have presented a more detailed analysis of how changing the optimisation criteria for A* altered its optimality but the results are still suggestive of a useful idea.

16.2.3 Early Learning with Sparse Data

We dealt with the issue of sparse data, and how to learn from it early in the game cycle, in chapter 2 and discovered that many of the more grounded machine learning techniques were of no use as these required large amounts of training data and learning time. We presented simple techniques which could generate results from very small sample amounts of data and showed that for our particular situation these performed better. By doing this comparison and describing the properties of the type of data used we allow these conclusions to be used to guide learning from similar data types in time constrained domains where early learning from sparse data is vital to performance.

16.2.4 Adversarial Domains

In the UT domain we found that at different skill levels there were differing levels of performance increase achieved using our system and architecture. This was largely because of our use of tapered learning in response to the measured "goodness" of the current information. We allowed the learning rate (expressed as *number of learning units* within a team) to change if the exploiting performance was believed to be good. This, almost by definition, leads to differing performance as the enemy have different levels of skill.

This result can be applied in general to adversarial domains where performance mediated learning is used to tackle reinforcement learning trade-offs. It will not be the case that the space of performance increase versus severity of the adversarial nature is always the same but the non-linear nature of the space is believed to be a general result with this architecture.

16.3 The Broader Context

16.3.1 Biological Context

Above and beyond the specifics of the previous work there are less obvious reasons for this research. There is a theme of combining methods from machine learning with those of a more structured logical approach to Artificial Intelligence in order to try to bridge a gap between the two. Many authors are concerned with trying to prove that either sub-symbolic or symbolic methods provide the best, more biologically grounded, approach to creating a system which could be described as in some way intelligent. The ideas proposed within this thesis provide a basic starting point for how to straddle this gap using a layered architecture as proposed.

Some research[111, 104], within the machine learning and neural computation communities, suggests that a successful sub-symbolic/connectionist approach to AI would have to eventually develop some form of symbolic representation, via evolution or otherwise, in order to deal with the vast amount of symbolic systems present in every day life. From the contrasting perspective there is research into finding a biological basis for claims that we should work backwards from the symbolic angle[100]. This argues that the reason sub-symbolic engineering has made so little progress is that it doesn't have an over-riding vantage point or framework for the system to fit into.

My personal viewpoint is that learning mechanisms are important but that these must fit into some overall structure, which may or may not be learned. The work of Bednar et al[84] on the visual processing systems of the human brain shows that there are clearly areas of the brain which offer plasticity towards how they are organised and how they represent each individual part of the input space. Even so, these systems are embedded within an overall structure of the brain about which more is known due to its more obvious gross anatomical structure. This parallels (albeit in a complex manner) the work presented here where the modelling of enemy behaviour is handled through traditional machine learning methods and models, while the overall plan structure is fixed. This is not to say that our system is any way applicable to models of any of the audio or visual processing systems contained within the brain. We merely draw an analogy to this, as an example of a system where adaptation of low level behaviours is allowed within a rigid overall structure, as a means of highlighting this property of our system.

16.3.2 Real World Conclusions

As well as showing that this particular architecture is useful in the UT domain it is important to discuss how this work could apply to other areas. Probably the most obvious area of use for this type of work is in the military domain where there is a direct analogy with weapons and health points for units. In this domain typically a larger number of units is used but in this case we can make simple analogies to health as number of units and single viewpoints as squad leaders. The idea of pushing down implementation details to lower architectural levels would translate to allowing the squad leaders to organise their squads at a local level and implement movement details. In this sense the learning mechanisms would be the squads themselves and the behaviour modules would amount to the squad leader's commands. As such the most portable element of the system would be the LCC Strategic element and the approach to the reinforcement learning.

The work presented here also ties in to the real world OpenKnowledge (OK) project[2]. In OK project plans are formulated in LCC to allow multiple different agents to perform in scenarios such as search and rescue. The main problem with these plans is that when they are used the multiple agents often fail to execute their own part of the plan or execute it in such a way that the goals do not actually get satisfied or are left in a state where no verification of such properties is possible. By showing how to approach this problem in UT this work makes a contribution towards this problem and how, by carefully considering the possible environments and the level of abstraction which the plan contains, this issue could be resolved. The solutions and plans here are of a lower complexity than those for real world scenarios such as search and rescue but the mechanisms used are still of relevance.

A further connection concerns the issue of global optimality over local optimality. In real world scenarios agents are expected to act in a way which maximises the global utility of the team but which is not considered maximal in regards to the agent's local measure of performance. This is very apparent in the chosen approach to the exploration versus exploitation problem, where the central theme is global optimality with sub-optimal local bot strategies. Maybe the only way to tackle such real world problems is to consider having certain real world elements or agents act in a known sub-optimal manner.

A nice side effect of the abstraction of the LCC strategies is that what we are left with as LCC plans are simple enough that they may more easily lend themselves to auto-generation via standard planning techniques and understanding to human elements. If humans are to work in some way with robotic agents, or pieces of autonomous software controlling certain elements of a large scale operational equation, then this must somehow be factored in. If we can give each element a certain level of autonomy whilst making the overall multi-agent plan human readable, this is likely to help us achieve this goal.

Bibliography

[1] Black and White Official Website - http://www.lionhead.com/bw/.

[2] The Open Knowledge Project official website - http://www.okfn.org/.

[3] Andrade, G. and Ramalho, G. and Santana, H. and Corruble, V. Automatic Computer Game Balancing: A Reinforcement Learning Approach. *AAMAS*, pages 25–29, 2005.

[4] P. Avery, S. Louis, and B. Avery. Evolving Coordinated Spatial Tactics for Autonomous Entities using Influence Maps. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[5] Avinash,D. and Skeath,S. *Games of Strategy*. W.W. Norton And Company, 2004.

[6] Bahceci, E. and Miikkulainen, R. Transfer of Evolved Pattern-Based Heuristics in Games. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[7] Bailey, T. and Jain, A. A Note on Distance-Weighted K-Nearest Neighbor Rules. *IEEE Trans. Systems, Man, Cybernetics*, 8:311–313, 1978.

[8] Baker, J.E. Reducing Bias and Inefficiency in the Selection Algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 14–21. Hillsdale, 1987.

[9] Barber, C. Bradford and Dobkin, David P. and Huhdanpaa, Hannu. The Quickhull Algorithm for Convex Hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[10] Barber, D. *Learning From Data*. 2006.

[11] Bilmes , J.A. *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mexture and Hidden Markov Models*. International Computer Science Institute, U.C. Berkeley, April 1998.

[12] Bishop, C.M. *Pattern Recognition and Machine Learning*. Springer, 2006.

[13] Bishop, C.M. and Svensen, M. and Williams, C.K.I. The Generative Topographic Mapping. Technical report, Neural Computing Research Group, Aston University, Birmingham, April 1997.

[14] Y. Bjornsson, M. Enzenberger, R.C. Holte, and J. Schaeffer. Fringe Search: Beating A* at Pathfinding on Game Maps. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[15] Blackburn, P. and O'Sullican, B. Building Reactive Characters for Dynamic Gaming Environments. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[16] A. Botea, M. Muller, and J. Schaeffer. Near Optimal Hierarchical Path-Finding. *Journal of Game Development*, 1(1):7–28, 2004.

[17] Bowling, M. and Furnkranz , J. and Graepel, T. and Musick, R. Machine Learning and Games. *Machine Learning*, 63:211–215, 2006.

[18] Bradley, J. and Hayes, G. Adapting Reinforcement Learning for Computer Games: Using Group Utility Functions. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[19] Brooks, A.R. Elephants Dont Play Chess. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, 1990.

[20] Brooks, R.A. Intelligence Without Representation. *Artificial Intelligence*, 47:139–159, 1991.

[21] Bum Hee Lee. An Analytic Approach to Moving Abstacle Avoidance Using an Artificial Potential Field. In *IROS '95: Proceedings of the International Conference on Intelligent Robots and Systems-Volume 2*, page 2482, Washington, DC, USA, 1995. IEEE Computer Society.

[22] Burkey, M. and El Rhalibi, A. A Hybrid AI System for Agent Adaptation in a First Person Shooter. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[23] Champandard, A.J. *AI Game Development*. New Riders, 2003.

[24] Combes, C. and Hennessy, R. and Waddington, J. and Roberts, N. and Prima, S. . An Algorithm To Map Asymmetries of Bilateral Point Clouds.

[25] Connell, Jonathan H. Creature Design With the Subsumption Architecture. In *IJCAI'87: Proceedings of the 10th international joint conference on Artificial intelligence*, pages 1124–1126, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[26] Cover, T.M. and Hart, P. Nearest Neighbour Pattern Classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[27] Curless, B. From Range Scans to 3D Models. *ACM SIGGRAPH Computer Graphics*, 33(4):38–41, 2000.

[28] Danielsiek, H. and Stuer, R. and Thom, A. and Deume, N. and Naujoks, B. and Preuss, M. Intelligent Moving of Groups in Real-Time Strategy Games. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[29] Dawson, C. *AI Game Programming Wisdom*, chapter Formations, pages 272–281. Charles River Media, 2002.

[30] de Aquino, R.R.B. and Ferreira, A.A. and Carvalho Jr, M.A and Lira, M.M.S. and Silva, G.B and Neto, O.N. . Combining Artificial Neural Networks and Heuristic Rules in a Hybrid Intelligent Load Forecast System. In *Artificial Neural Networks ICANN 2006*, volume 4132/2006 of *Lecture Notes in Computer Science*, pages 757–766. Springer Berlin / Heidelberg, September 2006.

[31] Dechter, D. and Pearl, J. Generalized Best-First Search Strategies and the Optimality of A*. *Journal of the ACM*, 32:505–536, 1985.

[32] Denzinger, J. and Winder, C. Combining Coaching and Learning to Create Cooperative Character Behaviour. In *IEEE Symposium on Computational Intelligence and Games*, 2007.

[33] D. Devigne, P. Mathieu, and J.C. Routier. Teams of Cognitive Agents with Leader: How To Let Them Some Autonomy. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[34] Dietterich, T.G. *Multiple Classified Systems, First International Workshop, MCS 2000 Cagliari, Italy*, chapter Ensemble Methods in Machine Learning, pages 1–15. Springer Berlin / Heidelberg, June 200.

[35] E. Gokcay and J.C. Principe. A New Clustering Evaluation Function Using Renyi's Information Potential. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 6:3490–3493, 2000.

[36] Erhan Gokcay and Jose C. Principe. Information Theoretic Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):158–171, 2002.

[37] Galli, L. and Loiacono, D. and Lanzi, P.L. Learning a Context-Aware Weapon Selection Policy for Unreal Tournament III. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[38] Gat, E. Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Mobile Robots. In *SIGART Bulletin 2*, pages 70–74, 1991.

[39] Gold, A. Academic AI and Video Games: A Case Study of Incorperating Innovative Academic Research into a Video Game Prototype. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[40] Graham, P. Using LCC with Unreal Tournament. Undergraduate Dissertation, May 2007.

[41] Green, K.C. Forecasting Decisions in Conflict Situations: a Comparison of Game Theory, Role-Playing, and Unaided Judgement. .

[42] J. Hagelback and S.J. Johansson. Dealing with Fog of War in a Real Time Strategy Game Environment. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[43] Hammoudi, K. and Dornaika, F. and Paparoditis, N. Extracting Building Footprints Fom 3D Point Clouds Using Terrestrial Laser Scanning At Street Level. In Stilla U and Rottensteiner F and Paparoditis N, editor, *CMRT09*, volume XXXVIII, 2009.

[44] D. Harabor and A. Botea. Hierarchical Path Plannig for Multi-Size Agents in Heterogeneous Environments. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[45] Harati, A. and Ahmadabadi, M.N. and Araabi, B.N. Knowledge-Based Multiagent Credit Assignment: A Study on Task Type and Critic Information. *Systems Journal, IEEE*, 1(1):55–67, Sept 2007.

[46] Hester, T. and Stone, P. Generalized Model Learning for Reinforcement Learning in Factored Domains. In *The Eighth International Conference on Autonomous Agents and Multiagent Systems*, 2007.

[47] Heudin, J.C. *Virtual Systems and Multimedia*, volume 4820/2008, chapter Evolutionary Virtual Agent at an Exhibition, pages 154–165. Springer Berlin / Heidelberg, March 2008.

[48] Hill, T. and Marquez, L. and Remus, W. Artificial Neural Network Models for Forecasting and Decision Making . , Sept.

[49] Hladky, S. and Bulitko, V. An Evaluation of Models for Predicting Opponent Positions in First-Person Shooter Video Games. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[50] Isla, D. *AI Game Programming Wisdom 3*, chapter Probabilistic Target Tracking and Search Using Occupancy Maps, pages 379–387. Charles River Media, 2006.

[51] S. Jang and S. Cho. Evolving Neural NPCs with Layered Influence Map in the Real-time Simulation Game 'Conqueror'. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[52] Jenssen, R. and Hild II, K.E. and Erdogmus, D. and Principe, J.C. and Eltoft, T. . Clustering Using Renyi's Entropy. *IEEE*, 2003.

[53] Jong, N.K. and Stone, P. Compositional Models for Reinforcement Learning. In *The European Conference on Machine Learning*, 2009.

[54] Kaelbling, L.P. and Littman, M.L. and Moore, A.W. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[55] Kahn, K. Comparing Multi-Agent Models Composed from Micro-Behaviours. In *Third International Model-to-Model Workshop*, 2007.

[56] Kahneman, D. and Tversky, A. and Slovic, P. *Judgment under Uncertainty, Heuristics And Biases*. Cambridge University Press, 1982.

[57] Kalyanakrishnan, S. and Stone, P. Learning Complementary Multiagent Behaviors:A Case Study. In *Proceedings of the RoboCup International Symposium*, 2009.

[58] Kasakov, D. and Kudenko, D. Machine Learning and Inductive Logic Programming for Multi-agent Systems . *ACAI ,LNAI 2086, pp. 246270*, 2001.

[59] Kennedy, J. and Eberhart, R.C. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[60] Lalond, J.F. and Vandapel, N. and Huber, D. and Hebert, M. Natural Terrain Classification Using 3-D Ladar Data for Ground Robot Mobility. *Journal of Field Robotics*, Volume 23(10), 2006.

[61] Li, Q.; Racine, J.S. *Nonparametric Econometrics: Theory and Practice*. Princeton University Press, 2007.

[62] Mao, W. and Gratch, J. The Social Assignment Problem. In *the 4th International Working Conference on Intelligent Virtual Agents, Kloster Irsee , Germany 2003*.

[63] McLachlan, G.J. and Basford, K.E. Mixture Models: Inference and Applications to Clustering, 1988.

[64] McPartland, M. and Gallagher, M. Creating a Multi-Purpose First Person Shooter Bot with Reinforcement Learning. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[65] Michalewicz, Z. and Fogel, D.B. *How To Solve It: Modern Heuristics*. Springer Verlag, 2000.

[66] C. Miles and S.J. Louis. Case-Injection Improves Response Time for a Real-Time Strategy Game. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[67] Miller, M.F. A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. November 13 1990.

[68] Negio, Y., Vincent, P. . Locally Weighted Full Covariance Gaussian Density Estimation. Technical report, Cirano, University of Montreal, 2004.

[69] Nicolai, G. and Hilderman, R.J. No-Limit Texas Hold'em Poker Agents Created with Evolutionary Neural Networks. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[70] Noel J. Rode and Andrew P. Paplinski and Andrew P. Papli'nski. Simulation of a Subsumption Architecture Robot: Genghis, .

[71] Onieva, E. and Pelta, D.A. and Alonso, J. and Milanes, V. and Perez, J. A Modular Parametric Architecture for the TORCS Racing Engine. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[72] Panait, L. Learning Team Behaviours with Adaptive Heterogeneity. *AAMAS*, pages 25–29, July 2005.

[73] Panait, L. and Luke, Sean. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, 11(2):387–434, 2005.

[74] Parker, M. and Bryant, B.D. Visual Control in Quake II with a Cyclic Controller. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[75] Parker, M. and Bryant, B.D. Backpropagation without Human Supervision for Visual Control in Quake II. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[76] Parzen, E. On the Estimation of a Probability Density Function and the Mode. *Annals of Math. Stats.*, 33:1065–1076, 1962.

[77] Paul, G. and Liu, D.K. and Kirchner, N. . *Robotic Welding, Intelligence and Automation*, volume Volume 362, chapter An Algorithm for Surface Growing from Laser Scan Generated Point Clouds, pages 481–491. Springer Berlin / Heidelberg, 2007.

[78] Pearson, K. On Lines and Planes of Closest Fit to Systems of Points in Space. 1901.

[79] Pugh, J. and Martinoli, A. Multi-Robot Learning with Particle Swarm Optimization. *AAMAS*, May 2006.

[80] R. De Maesschalck, D. Jouan-Rimbaud and D. L. Massart*. *The Mahalanobis distance*. ChemoAC, Pharmaceutical Institute, Department of Pharmacology and Biomedical Analysis, Vrije Universiteit Brussel, Laarbeeklaan 103, B-1090 Brussels, Belgium, January 200.

[81] Rasmusen, E. *Game and Information: an Introduction to Game Theory*. Wiley-Blackwell, 1989.

[82] Reeder, J. and Miguez, R. and Sparks, J. and Georgiopoulos, M. and Anagnostopoulos, G. Interactively Evolved Neural Networks for Game Agent Control. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[83] Riemsdijk, M.B.V. and Dastani, M. and Meyer, J-J.Ch and de Boer, F.S. Goal-Orientated Modularity in Agent Programming. *AAMAS*, pages 8–12, 2006.

[84] Risto Miikkulainen and James A. Bednar and Yoonsuck Choe and Joseph Sirosh. *Self-Organization, Plasticity, and Low-Level Visual Phenomena in a Laterally Connected Map Model of the Primary Visual Cortex*. 1997.

[85] Robertson, D. Lightweight Co-ordination Calculus. In *Proceedings of the International Conference on Logic Programming, Sant-Malo*, 2004.

[86] Russel, J.S. and Norvig, P. *Artificial Intelligence: A Modern Approach*, pages 97–104. Prentice Hall, 2003.

[87] Schaeffer, J. A Gamut of Games. *Artificial Intelligence Magazine*, 22(3):29–46, 2001.

[88] Scholkopf, B. and Smola, A.J. *Learning With Kernels*. The MIT Press, 2002.

[89] Schlkopf, B. and Smola, Am and Muller, K.R. *Advances in Kernel Methods-Support Vector Learning*, chapter Kernel Principal Component Analysis, pages 327–352. MIT Press Cambridge, 1999.

[90] Sharma, M. and Holmes, M. and Santamaria, J. and Irani, A. and Isbell, C. and Ram, A. Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. *IJCAI*, 2007.

[91] Sithole, G. and Vosselman, G. Automatic Structure Detection in a Point-Cloud of an Urban Landscape .

[92] Spronck, P. *Adaptive Game AI.* PhD thesis, Maastricht University, 2005.

[93] Spronck, P. and Bakkes, S. Symbiotic Learning in Commercial Computer Games. *7th International Conference on Computer Games*, pages 116–120, 2005.

[94] Spronck, P. and Bakkes, S. and Postma, E. TEAM: The Team-oriented Evolutionary Adaptability Mechanism. *Entertainment Computing - ICEC*, pages 273–282, 2004.

[95] Spronck, P. and Bakkes, S. and Postma, E. Best-Response Learning of Team Behaviour in Quake III. *Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 13–18, 2005.

[96] Spronck, P. and Ponsen, M. and Sprinkhuizen-Kuyper, I. and Postma, E. Adaptive Game AI with Dynamic Scripting. *Machine Learning*, 63:217–248, 2006.

[97] Spronck, P. and Sprinkhuizen-Kuyper, I. and Postma, E. . Improving Opponent Intelligence through Machine Learning. *Proceedings of the Fourteenth Belgium-Netherlands Conference on Artificial Intelligence*, pages 299–306, 2002.

[98] Spronck, P. and Sprinkhuizen-Kuyper, I. and Postma, E. Online Adaptation of Computer Game Opponent AI. *Proceedings of the 15th Belgium-Netherlands Conference on Artificial Intelligence*, pages 291–298, 2003.

[99] Stanley, K.O. and Bryant, B.D. and Miikkulainen, R. Evolving Neural Network Agents in the NERO Video Game. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[100] Stenning, K. and Lambalgen, M.V. *Human Reasoning and Cognitive Science*. MIT Press, 2008.

[101] Stirling, W.C. Social utility Functions-part I: theory. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions*, 35(4):522–532, Nov 2005.

[102] Stone, P. Learning and Multiagent Reasoning for Autonomous Agents. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007.

[103] Stone, P. and Veloso, M. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*, 8:345–383, 2000.

[104] Su, Mu-Chun and Chen, Hsin-Hua and Cheng, Wan-Chi. A Neural-Network-Based Approach to Optical Symbol Recognition. *Neural Process. Lett.*, 15(2):117–135, 2002.

[105] Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[106] Thangarajah, J. and Padgham, L. and Sardina, S. Modelling Situations in Intelligent Agents. *AAMAS*, pages 8–12, May 2006.

[107] Thompson, T. and Levine J. Scaling-up Behaviours in EvoTanks: Applying Subsumption Principles to Artificial Neural Networks. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[108] Thompson, T. and Levine, J. Realtime Execution of Automated Plans using Evolutionary Robotics. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[109] Thompson, T. and Milne, F. and Andrew, A. and Levine, J. Improving Control Through Subsumption in the EvoTanks Domain. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[110] Touzet, C. and Santos, J.F. Q-learning and Robotics. In *IJCNN'99*. European Simulation Symposium, Marseille.

[111] Ultsch, A. Emergence in Self-Organizing Feature Maps. In *Proceedings Workshop on Self-Organizing Maps (WSOM '07).*

[112] van der Blom Spronck, L. and P. and Bakkes, S. Map-Adaptive Artificial Intelligence for Video Games. *8th International Conference on Intelligent Games and Simulation*, pages 53–60, 2007.

[113] van der Heijden, M. and Bakkes, S. and Spronck, P. Dynamic Formations in Real-Time Strategy Games. In *IEEE Symposium on Computational Intelligence and Games*, 2008.

[114] van Hoorn, N. and Togelius, J. and Schmidhuber, J. Hierarchical Controller Learning in a First-Person Shooter. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[115] Vandapel, N. and Huber, D. and Kapuria, A. and Hebert, M. Natural Terrain Classification using 3-D Ladar Data. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)*, 2004.

[116] Vasta, M. and Lee-Urban, S. and Munoz-Avila, H. RETALIATE: Learning Winning Policies in First-Person Shooter Games. In *Proceedings of the Seventeenth Innovative Applications of Artificial Intelligence Conference (IAAI-07)*, 2007.

[117] Vastianos G. SCROB - Subsumption Controlled Robotic Bug, 2001.

[118] H. Wang, O.N. Malik, and A. Nareyel. Multi-Unit Tactical Pathplanning. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[119] Westra, J. and Dignum, F. Evolutionary Neural Networks for Non-Player Characters in Quake III. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[120] Witten, I.H. And Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)* . Morgan Kaufmann, June 2005.

[121] Zelen, M. and Severo, N.C. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, chapter 26. National Bureau of Standards, 1964.

Appendix A

Appendix A

A.1 Level Modelling Extra Data

A.1.1 Absolute Zero

A.1.1.0.2 Variance 80: When using variance 80 as in figure A.1 we see that in cases 1, 3 and 5 the model picks out 1 or both of the bridge sections quite well. The problem is that sometimes it also adds points from one of the large sections either side of the bridges to the opposite side creating a large protruding polygonal element. This is not ideal as given that this creates a shape where a large area of it is not actually contained within an accessible portion of the level we don't expect to able to gain much information from modelling it. The worst cases of this occur in cases 2 and 3. In all cases the modelling creates a large structure either side of the bridges but often spits it up into smaller areas. This makes sense as we expect that the internal structure of this large indoor sections will have some intricate structure so really anything which applies structure which is sensible to these should give us some modelling power.

A.1.1.0.3 Variance 40: When switching to variance 40, figure A.2, we see that in some cases the model delivers very good results ,particularly case 4 which is pretty much exactly what we would look for. In other cases however the results are very bad. Case 5 in particular has a lot of overlap between sections and not definition of the bridge elements. There is still some level of separation of the two large areas either side of the level but for the most part this result is not great.

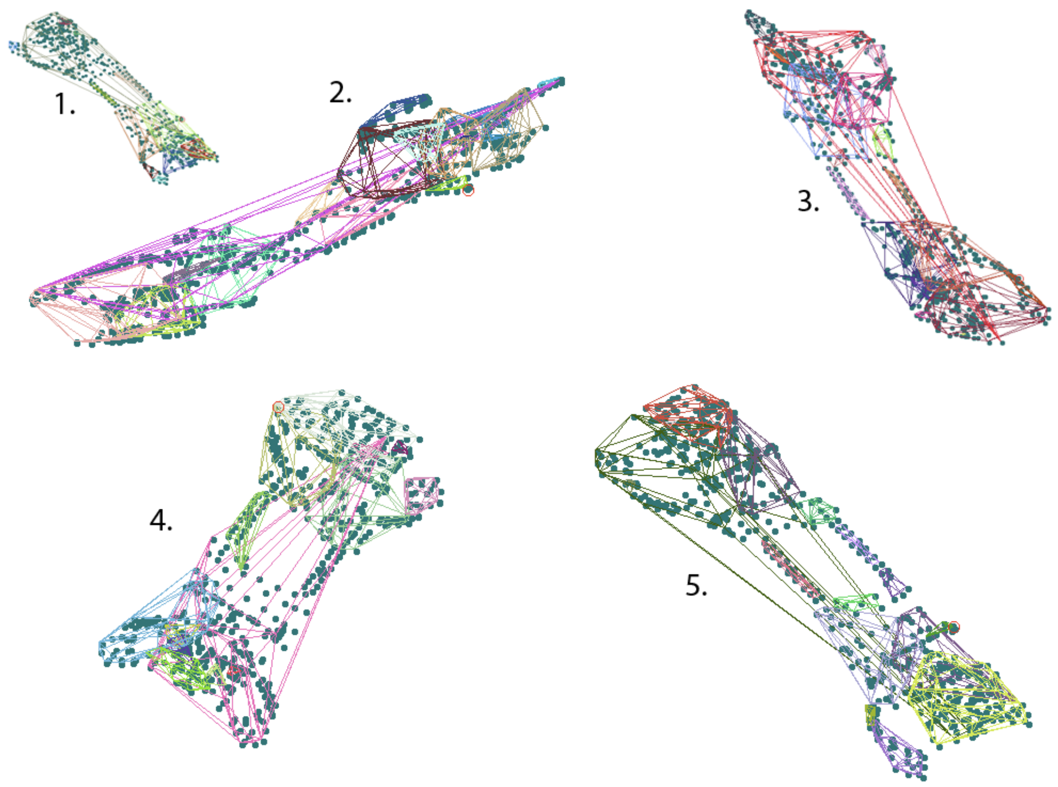


Figure A.1: Absolute Zero polygon fitting - Variance 80

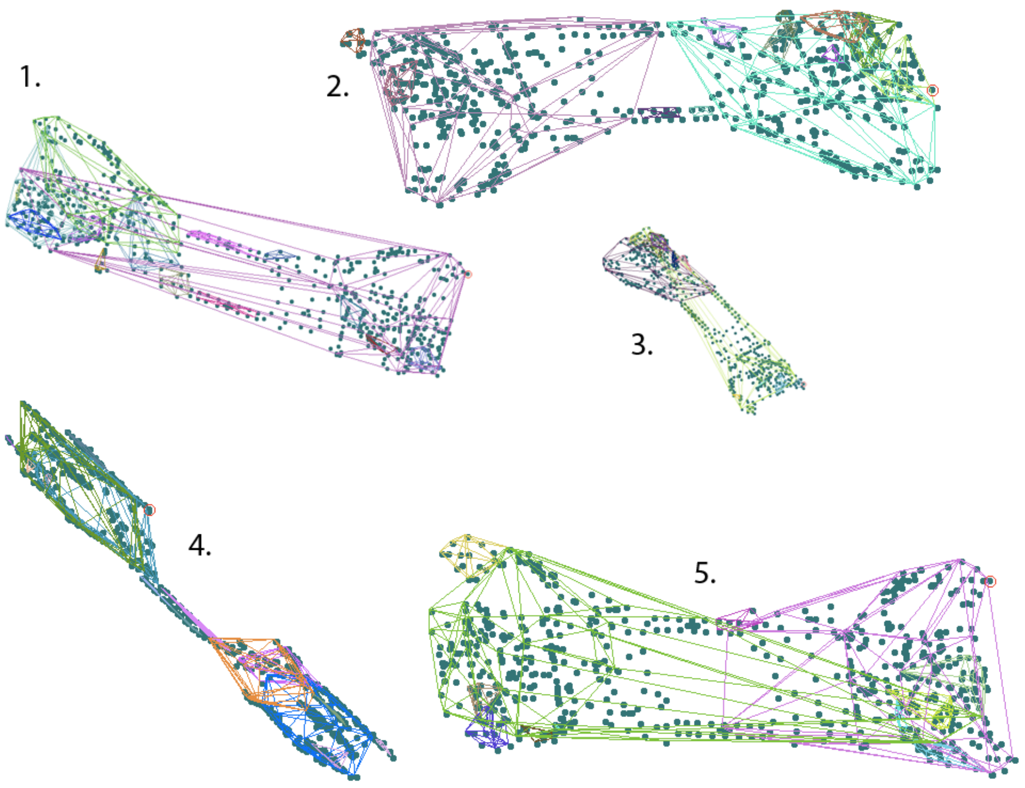


Figure A.2: Absolute Zero polygon fitting - Variance 40

A.1.1.0.4 Variance 20: The variance 20 model resembles the variance 40 model in a lot of ways, with cases 3 and 5 again being very good. This said however again cases 1, 2 and 4 are not great with a lack of separation and some modelling of sections which are mostly inaccessible. The modelling is probably slightly better here though as in two cases rather than 1 a "Perfect"[1] result is achieved.

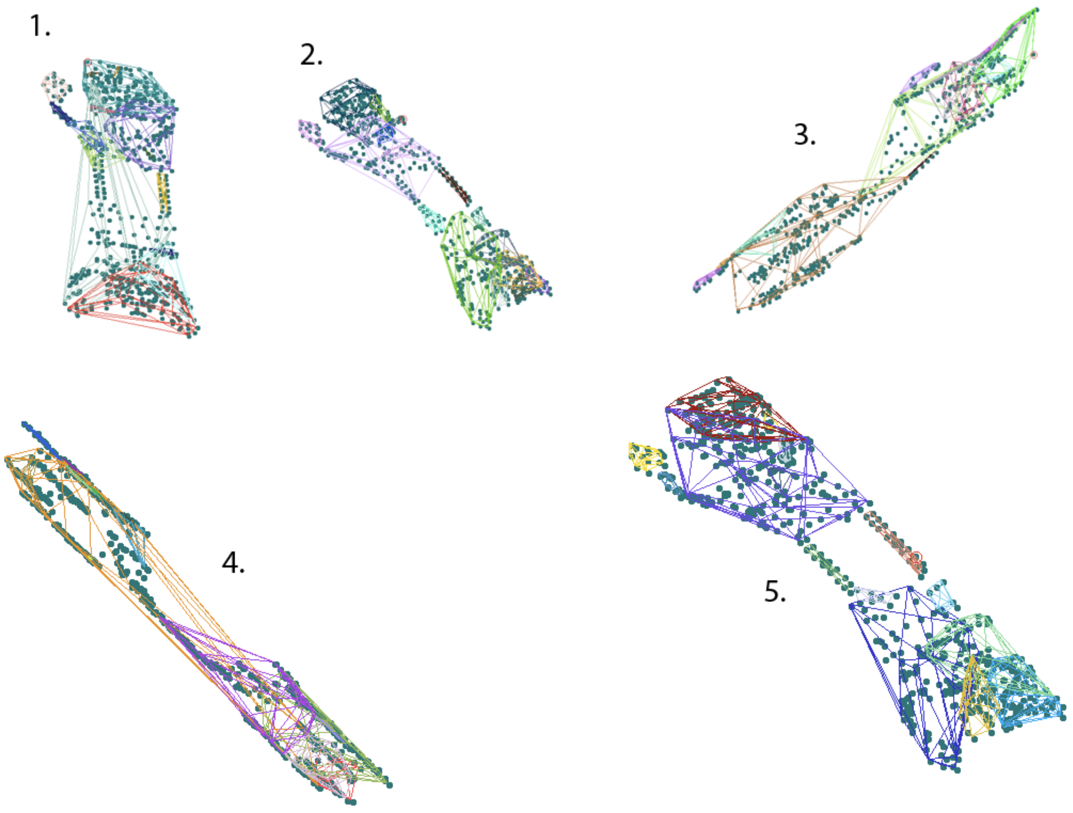


Figure A.3: Absolute Zero polygon fitting - Variance 20

A.1.1.0.5 Variance 10: When we move to variance 10 the bridge sections are relatively well defined in 3 of the cases (in case 1 the bridge has become part of the end area however it is still relatively well defined. In case 2 the two end sections are still well separated but in case 5 this is not so good. The areas bleed into each other and there is a much higher level of granularity in the modelling of the left hand area than the right.

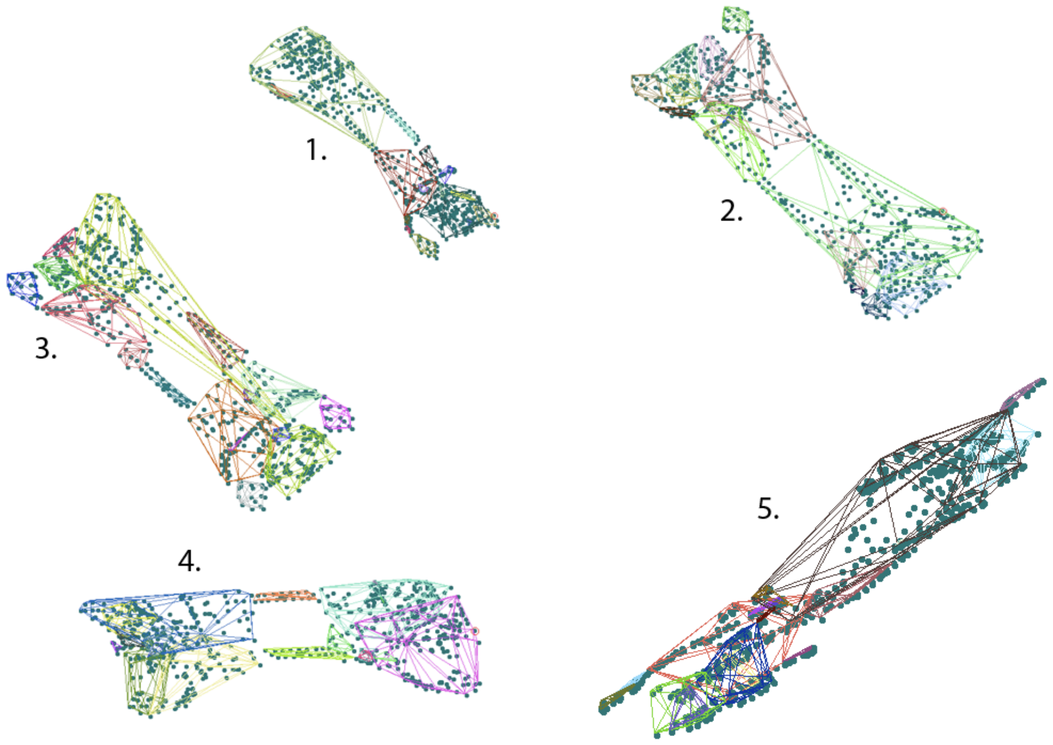


Figure A.4: Absolute Zero polygon fitting - Variance 10

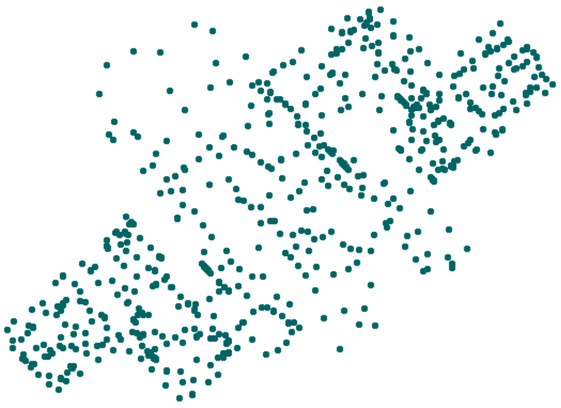


Figure A.5: December - 3D Point Data

A.1.2 December

December has two main sections either side of a large area in the middle of the level which contains a boat. The boat is not accessible but it can be used as a platform to link the two main sections.

Magnitude = 20.1869 PCA Magnitude of level = 1.3945869990510722

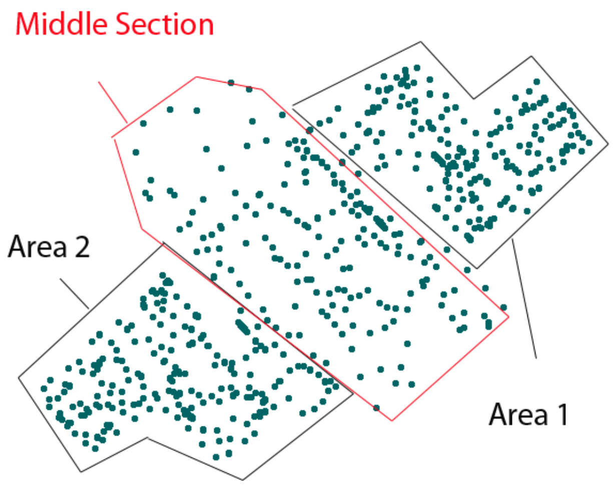


Figure A.6: December - Expected Polygons

A.1.2.0.6 Variance 80: In some cases the large middle section is at least obvious but it seems to span a lot more area than would be expected meaning that the two sections on either side of it are not very well defined.

A.1.2.0.7 Variance 40: This has some of the same problems as the previous setting. Occasionally the middle section is also split into two but this as more acceptable than when it is simply far too large

A.1.2.0.8 Variance 20: Once again this has similar problems. If we view the middle section from the view in case 3 we can see that the nodes making up the 'roof' of this middle section do actually expand out quite far into the other sections so our initial assumption that it is small may be quite difficult to ground on the basis purely of the 3D points which the algorithm is working from.

A.1.2.0.9 Variance 10:

[1]By perfect here we simply refer to the result as the being acceptable to a human observer with some knowledge of the level layout.

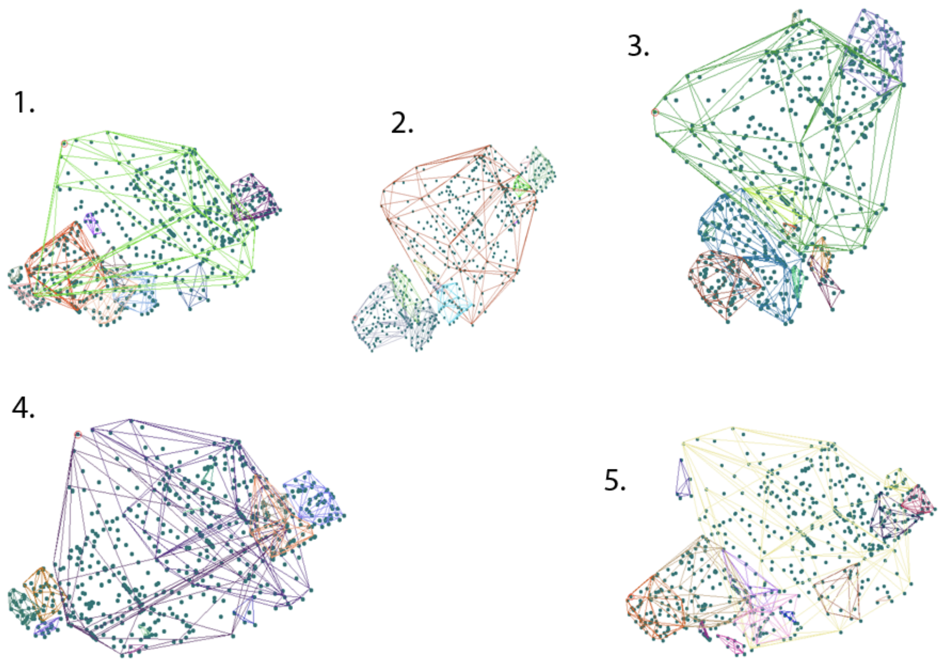


Figure A.7: December polygon fitting - Variance 80

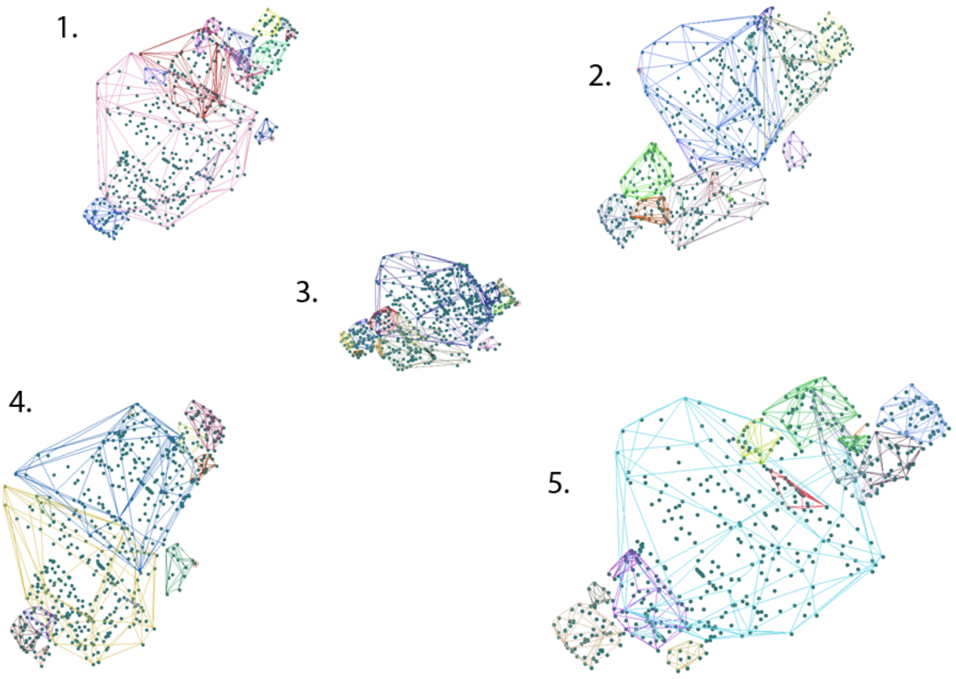


Figure A.8: December polygon fitting - Variance 40

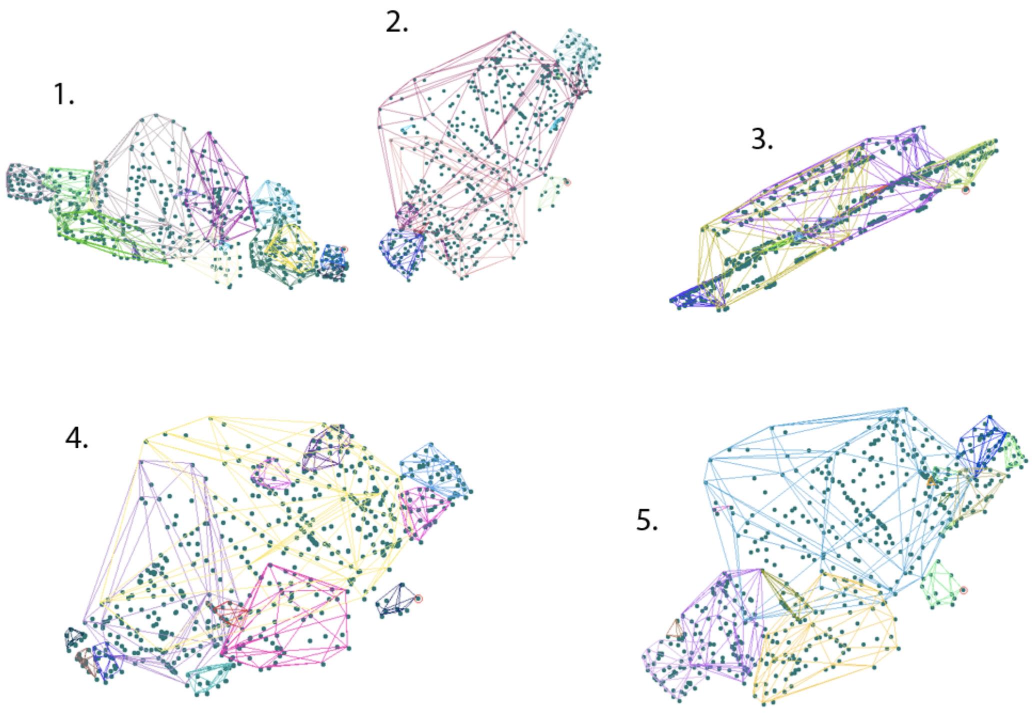


Figure A.9: December polygon fitting - Variance 20

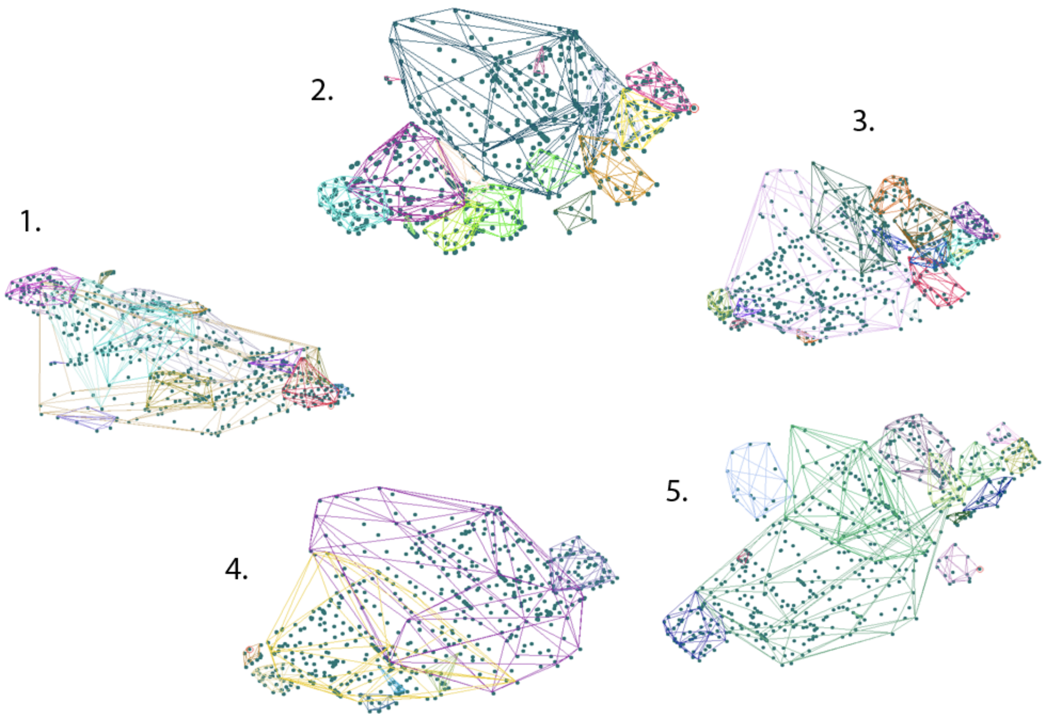


Figure A.10: December polygon fitting - Variance 10

A.1.2.0.10 Variance 5: This is not particularly good. It suffers from the same problem as the previous variances.

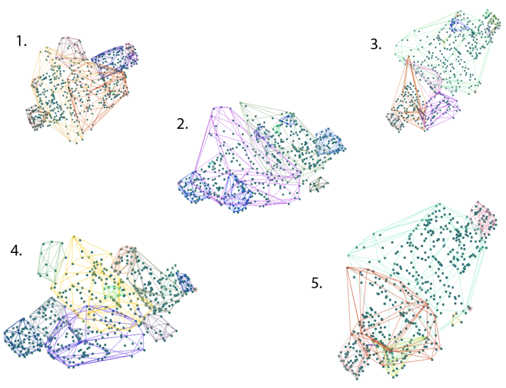


Figure A.11: December polygon fitting - Variance 5

A.1.2.0.11 Conclusions from visualisation: The visualisations offer no clear cut winner here. Variance 10 had one acceptable case but on the basis that the nodes expand out quite far into the other sections anyway. It may be acceptable to choose either 20 or 40 as an acceptable setting on the basis of visualisation.

In some cases the large middle section is at least obvious but it seems to span a lot more area than would be expected meaning that the two sections on either side of it are not very well defined. Occasionally the middle section is also split into two but this is more acceptable than when it is simply too large.

A.1.2.0.12 Conclusions from statistics: The statistics show that variance 10 has the lowest median cluster size and the second lowest standard deviation. Given that this was the only variance setting which generated a visually acceptable case and the fact that we were trying to make the section in the middle smaller it seems that this might be a good choice.

Observing figure 7.7

A.1.3 Deck17

Deck 17 is a largely indoor level with the main feature being in the middle section of the level where there is a large bridge going from one side of the level to the other with smaller bridge going over the top of the central bridge. This area should show up in the modelling, I don't expect the modelling of the bridges to be very good but I do

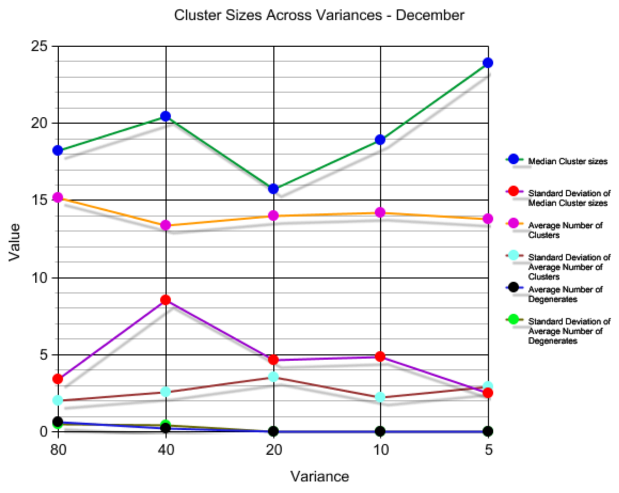


Figure A.12: Statistics Graph for December

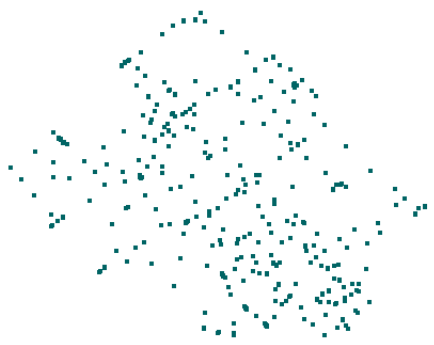


Figure A.13: Deck 17 - 3D Point Data

expect this area to at least be separated form the rest of the level which is essentially just tunnels. The rest of the level is not terribly well defined in the point cloud so I expect this just to generate random areas which are at least semi-sensible.

Magnitude = 5.33824

PCA Magnitude of level = 0.1421278849668724

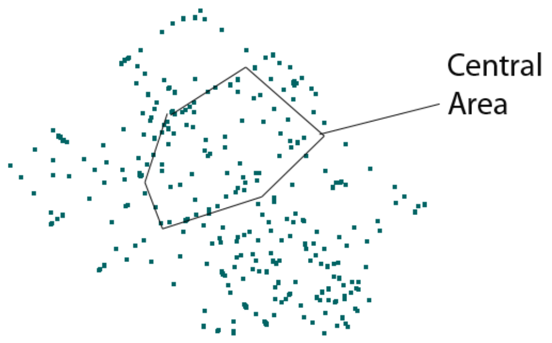


Figure A.14: Deck 17 - Expected Polygons

A.1.3.0.13 Variance 80: There is no clear definition of the central area but the actual polygons generated all seem well separated and give a concise representation of the 3D point cloud.

A.1.3.0.14 Variance 40: Here there is less separation between the sections but in cases 3, 4 and 5 the middle section is represented and well defined.

A.1.3.0.15 Variance 20: The results here are variable , in some cases the middle section is well represented but in others it is not and a large amount of overlap is seen.

A.1.3.0.16 Variance 10 Visualisation: In all cases the middle section is well represented. There is still overlap but it is less severe and makes more sense with reference to the level.

A.1.3.0.17 Variance 5: The middle section is relatively well represented here and there is little overlapping as in variance 10.

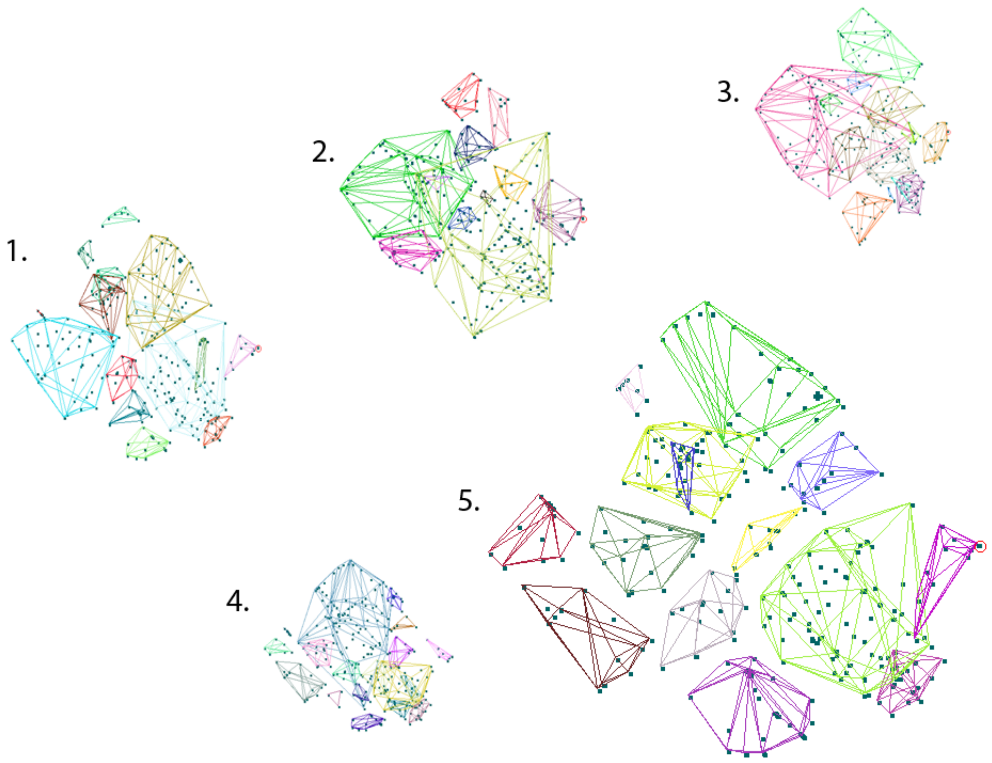


Figure A.15: Deck17 polygon fitting - Variance 80

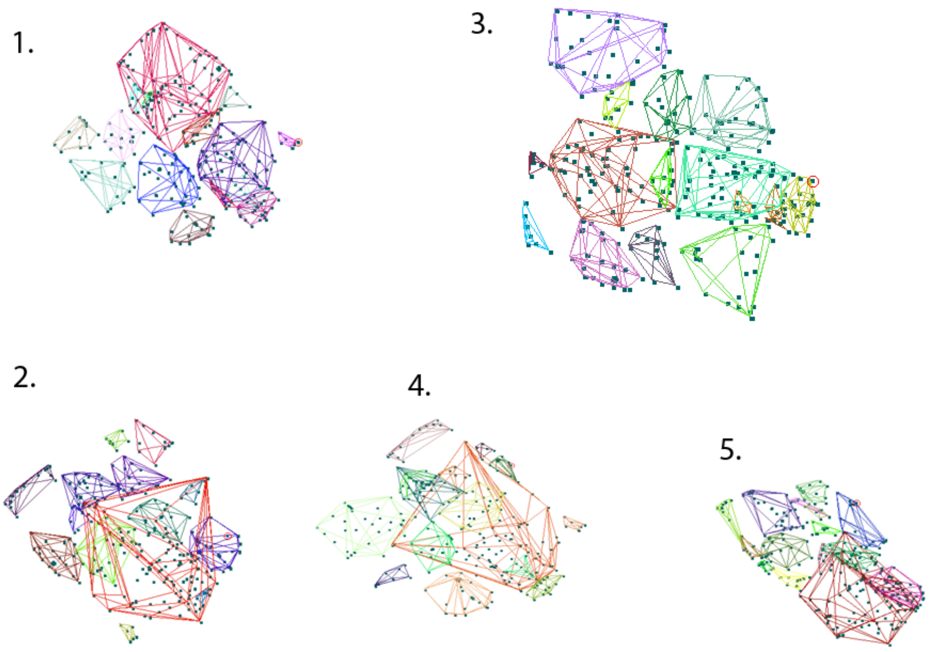


Figure A.16: Deck17 polygon fitting - Variance 40

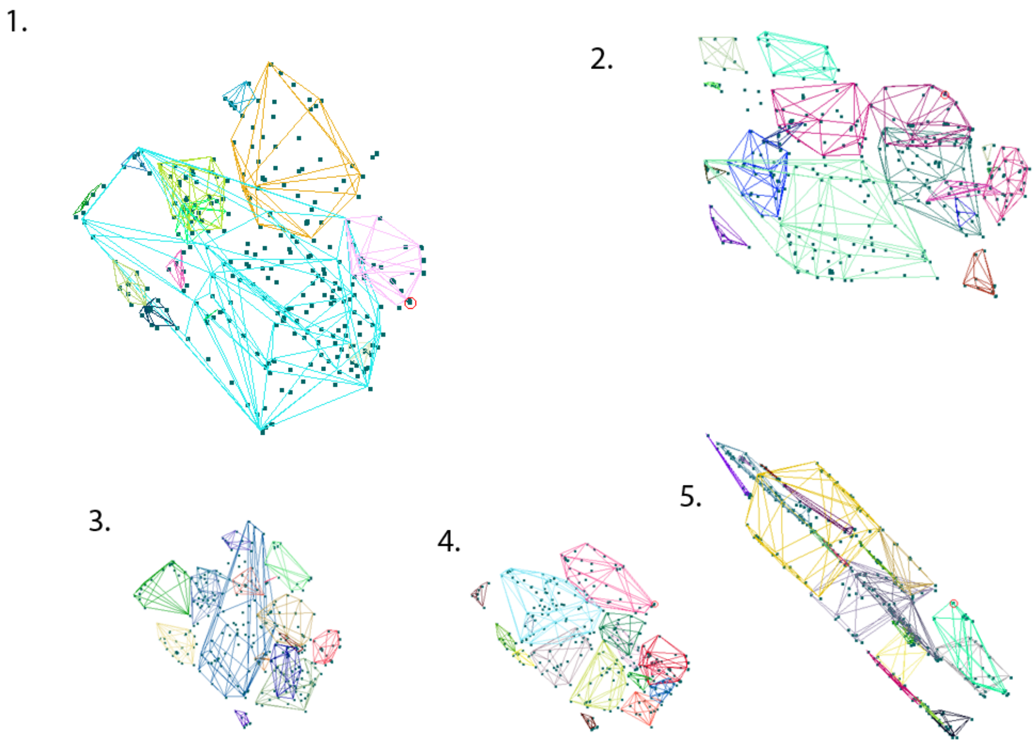


Figure A.17: Deck17 polygon fitting - Variance 20

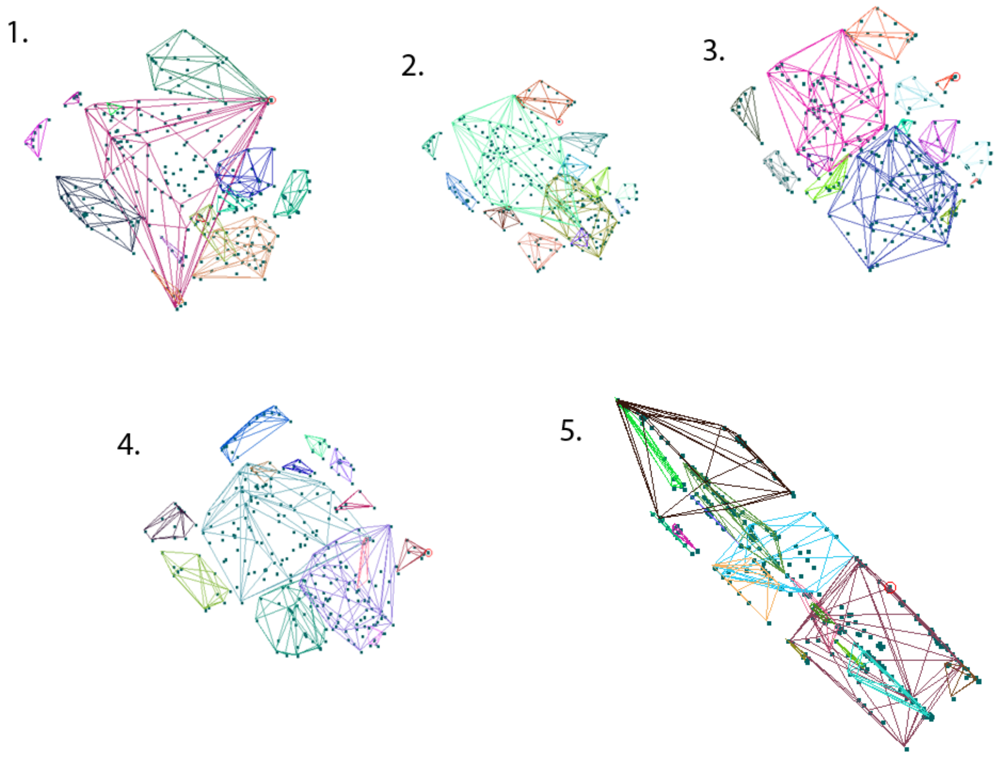


Figure A.18: Deck17 polygon fitting - Variance 10

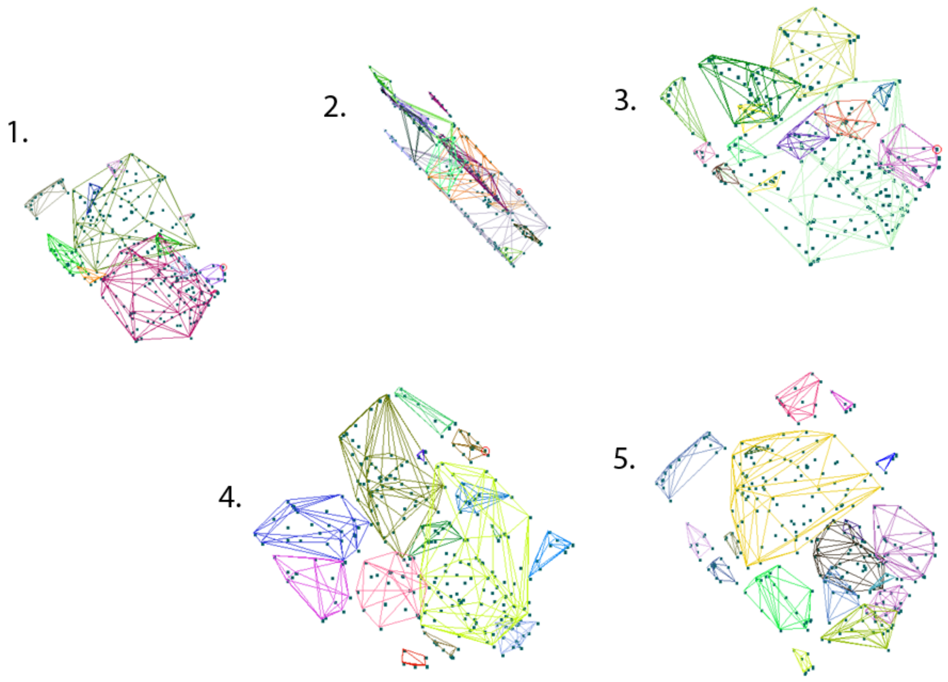


Figure A.19: Deck17 polygon fitting - Variance 5

A.1.3.0.18 Conclusions from statistics: The statistics show that variance 10 has the lowest standard deviation in the median cluster size but also that variance 5 generates a lower number of clusters. This suggests that if we wish larger clusters then we should probably use variance 10 but if we wished larger numbers of clusters then variance 5 would be a better idea. I personally feel that with reference to the visualisations the results in variance 10 are slightly better than those of 5.

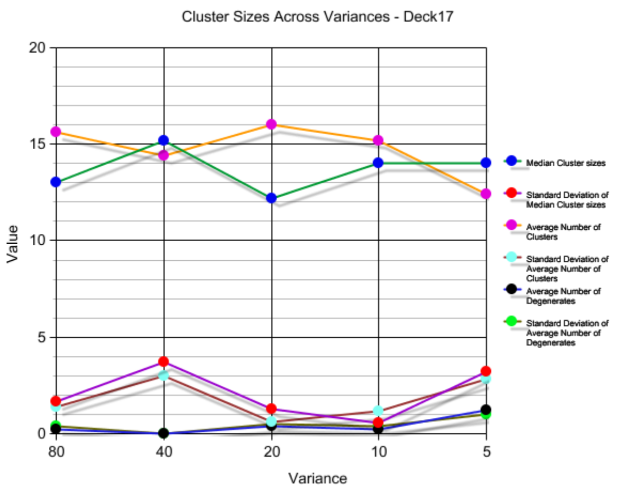


Figure A.20: Statistics Graph for Deck17

A.1.4 Face3

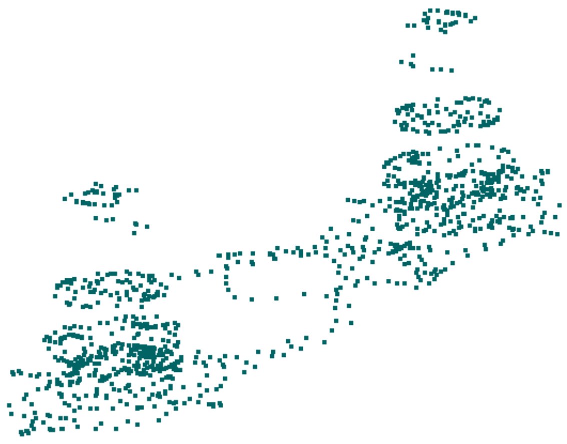


Figure A.21: Face 3 - raw 3D point data

Face 3 is a map similar in style to Absolute Zero. It has two large towers at either end of the level which are connected by a dual pathway between them. Again these represent the features that we wish to work with. It also worth noting that each tower has a small section towards the top which appears, in the navigation point set, to be separated from the lower section but in actual fact is an internal area within the upper region of the towers.

Magnitude = 22.27714 PCA Magnitude = 0.11281022398279406

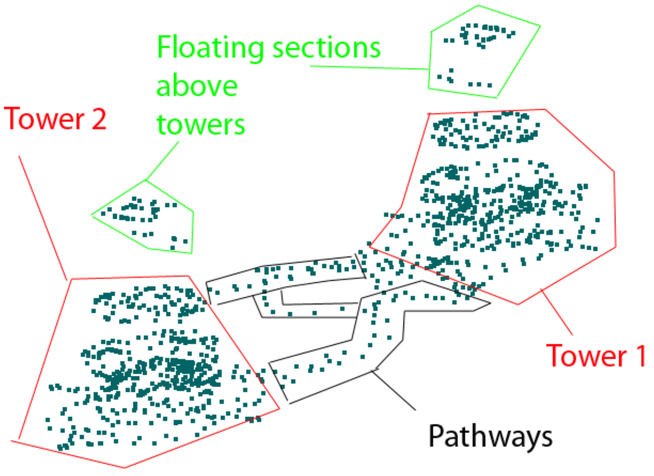


Figure A.22: Face 3 - Expected Polygons

A.1.4.0.19 Variance 80: We can see from figure A.23 that at variance 80 the pathways are not particularly well defined. The model does usually reflect relatively well the two tower areas but in general they tend to to envelope the paths in them aswell which reduces the level of granularity of the modelling in these two areas. In some cases one of the floating areas above the towers are seen and in cases 4 and 5 the modelling of one tower is much better than the other.

A.1.4.0.20 Variance 40: The modelling is pretty similar to the variance 80 model except in case 2 where the two path ways are represented much better and the floating sections above the towers are both reasonably well defined. These floating sections are also relatively well represented in case 1.

A.1.4.0.21 Variance 20: At variance 20 the paths are more defined and the separation of the two towers in general better. In cases 4 and 5 there is still a point from one tower getting caught in the other's hull which isn't so good.

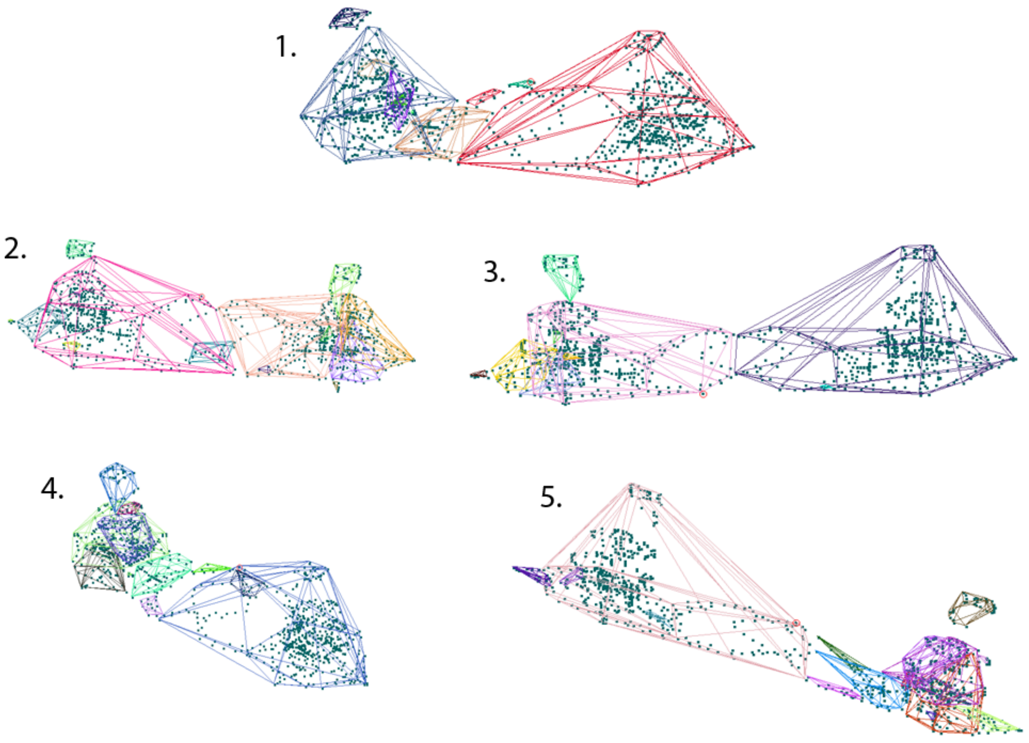


Figure A.23: Face 3 polygon fitting - Variance 80

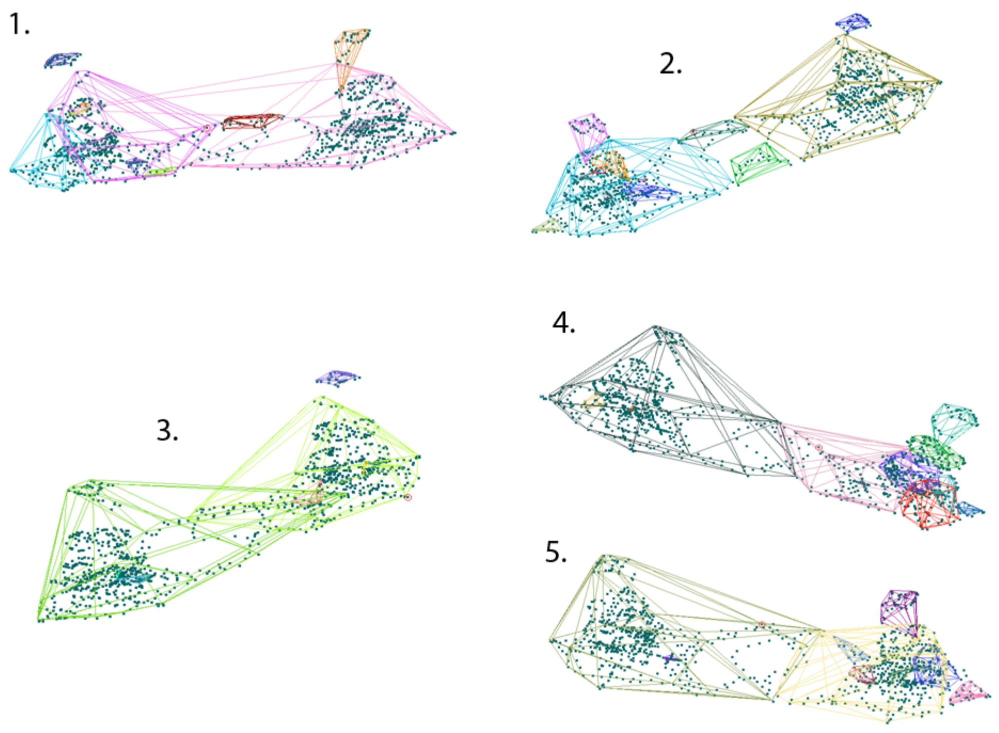


Figure A.24: Face 3 polygon fitting - Variance 40

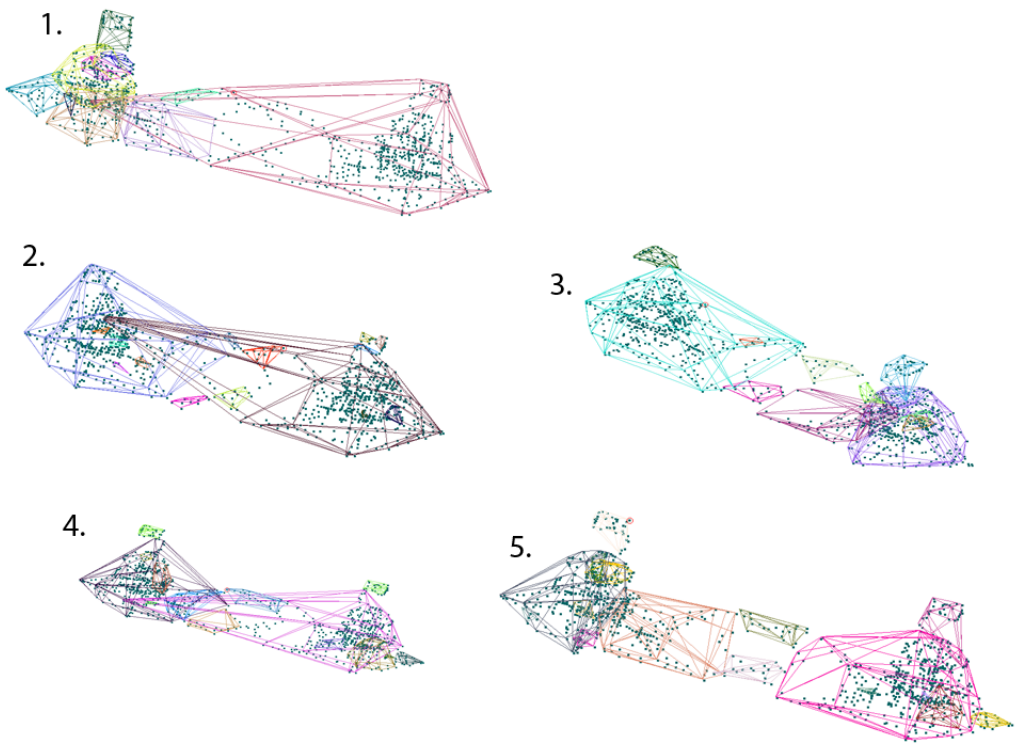


Figure A.25: Face 3 polygon fitting - Variance 20

A.1.4.0.22 Variance 10: This is similar to the 20 case but there is only one case where there is a significant overlap between the two towers sections. The two floating sections above the towers are slightly less well defined in this model.

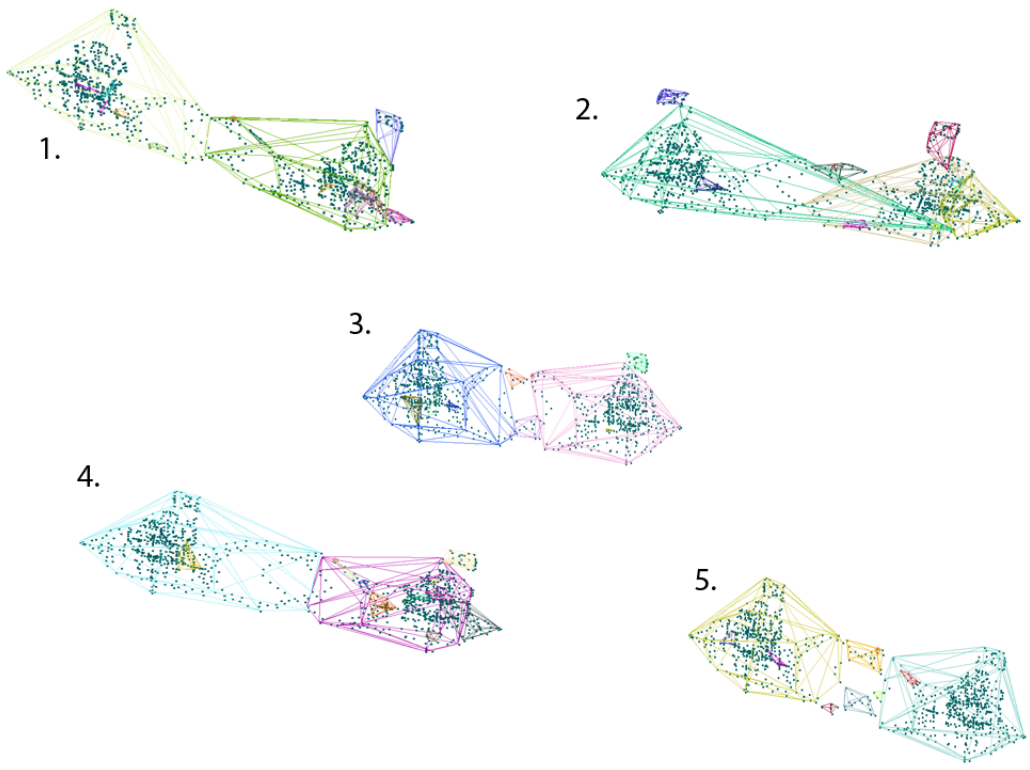


Figure A.26: Face 3 polygon fitting - Variance 10

A.1.4.0.23 Variance 5 Visualisation: This model seems to be the most stable. The separation of the two towers is well preserved and in most cases the two floating sections above the towers are quite well represented. Case 4 has the best representation of the paths of any of the variance values. We are still seeing the phenomenon however that usually one tower is better represented than the other. This is probably a product of the random placement of the initial seeds. It is difficult to think of a way of placing these initial seeds which would reduce dependency on where they were placed without prior knowledge of the level arrangement.

A.1.4.0.24 Conclusions from visualisation: If we take the results into consideration then the modelling using variance 5 seems to be the best selection for this level.

Larger values don't seem to have the level of granularity of modelling needed to differentiate the path sections. In these cases the paths become enveloped within the

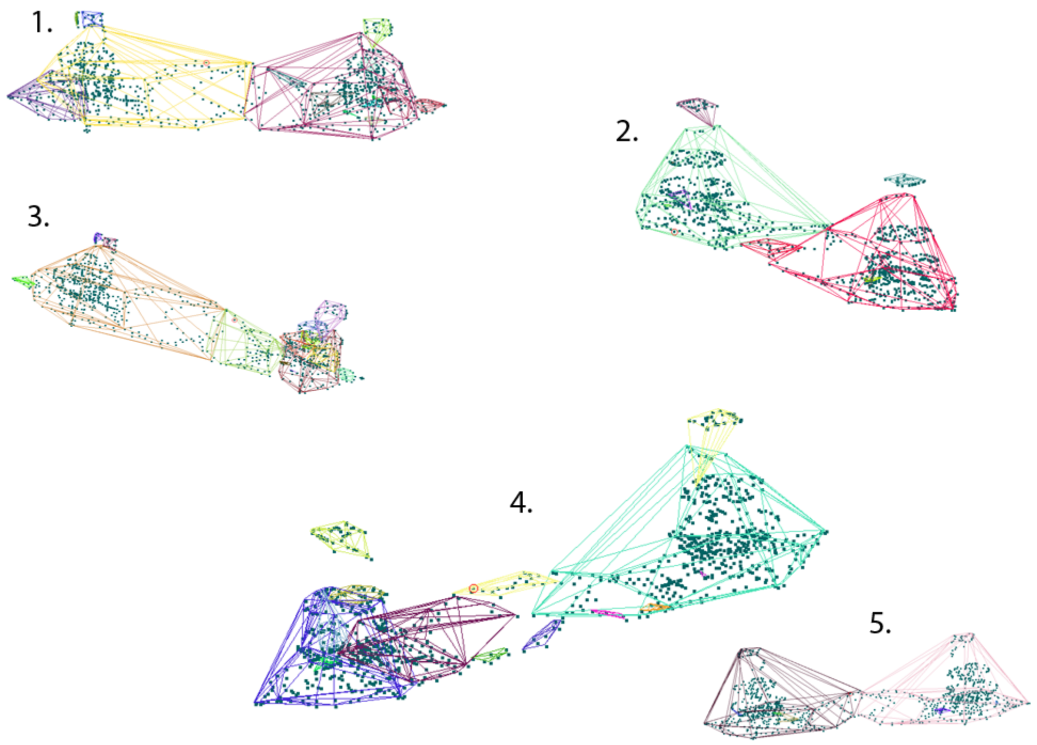


Figure A.27: Face 3 polygon fitting - Variance 5

tower sections whole level begins to look like to blobs and not much else. Occasionally a particular artefact also occurs where one point within a tower gets caught in the other's hull. These causes a large bridge section across the level. In actual fact this will not cause too many real problems in practice for us but it symbolises a small inadequacy in the modelling procedure at these variance values that is not ideal.

A.1.4.0.25 Conclusions from statistics: Observing figure 7.7 we see that for all variance values the average number of clusters generated is roughly the same. The standard deviation is also relatively low across the board so we can conclude that each process does in fact generate roughly the same number of clusters for this level with the main difference between each variance value being the size and shape of clusters rather than number of clusers generated We also see that in most cases very few degenerate clusters are generated from the process showing that our algorithm is relatively stable in the sense of not failing. The standard deviation lulls as we go down the variances from 40 but peaks again at 5 so this may suggest that 10 or 20 are better values as they appear more stable. The visualisations, do not, however seem to bare this out. Variance 5 definitely seems to be the best value for this level.

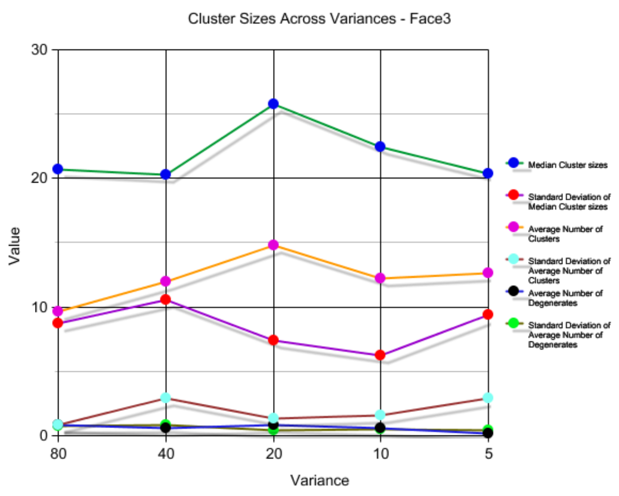


Figure A.28: Face 3 - Statistics

A.1.5 Face Classic

Face classic is very similar to face 3 except that it has many less path nodes and is slightly simpler. It was the original version of face contained in the original Unreal

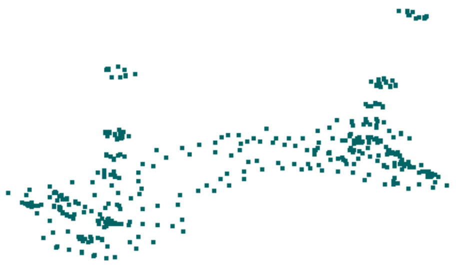


Figure A.29: Face Classic - 3D point data

Tournament game. As such it is interesting to see how our algorithm works on this version of the same level.

Magnitude = 16.5066 PCA Magnitude = 0.18334338164360908

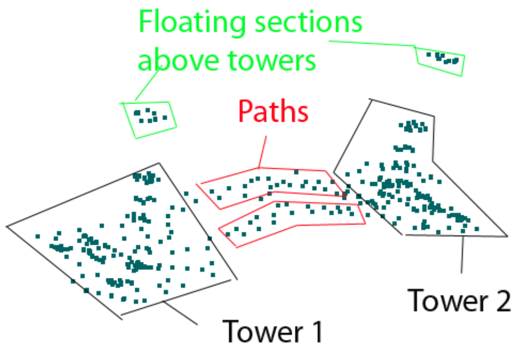


Figure A.30: Face Classic - Expected Polygons

A.1.5.0.26 Variance 80: The separation of the different sections here is much better than for face3. The path ways are better defined. The main problem that is still apparent is one of the towers having a point in the other leading to modelling of parts of the level which cannot have play in them.

A.1.5.0.27 Variance 40: This is better than for the 80 variance as the separation of the towers is again better. The path modelling is also better here than for 80. The floating sections above the towers are also more consistently picked up upon.

A.1.5.0.28 Variance 20: These are again showing more improvement with cases 4 and 5 in particular being very well defined and representing all the key features very sharply.

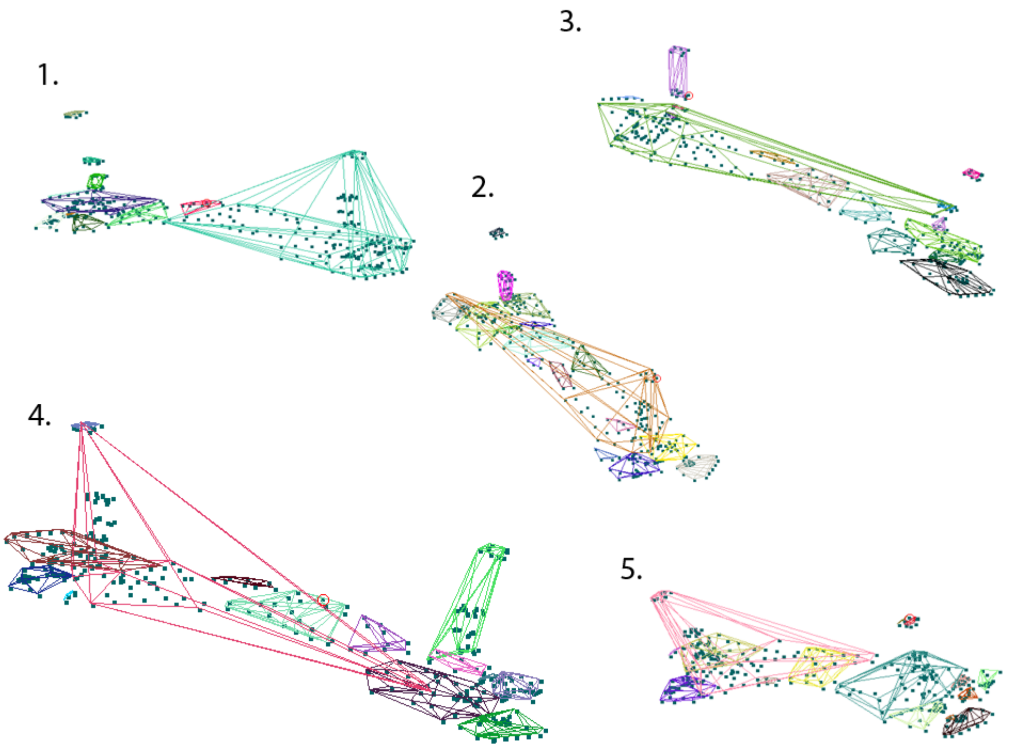


Figure A.31: Face Classic polygon fitting - Variance 80

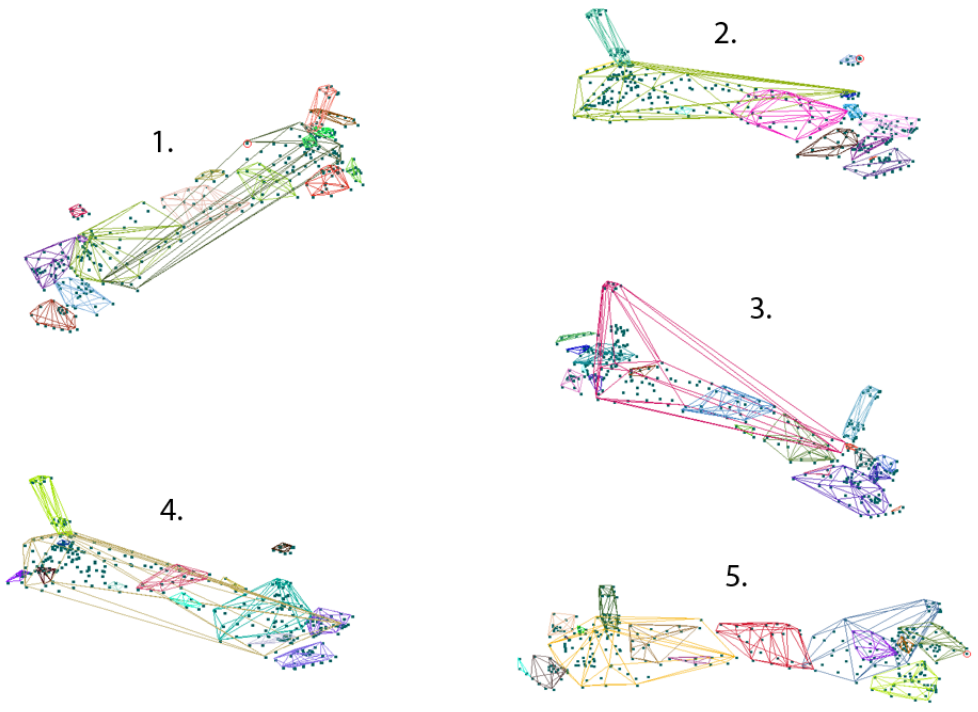


Figure A.32: Face Classic polygon fitting - Variance 40

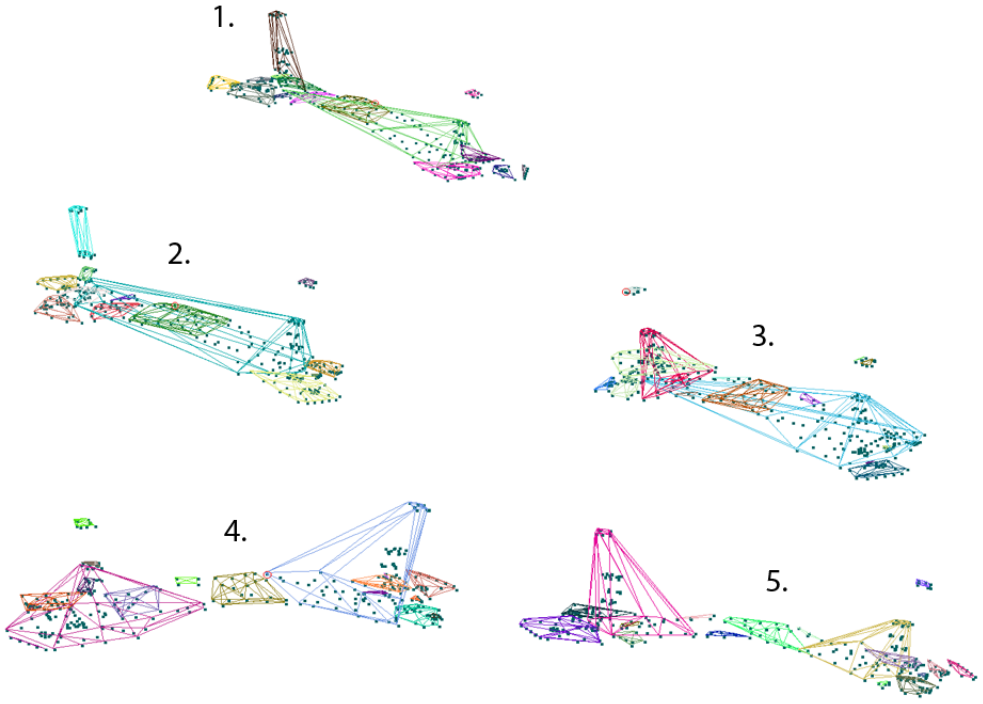


Figure A.33: Face Classic polygon fitting - Variance 20

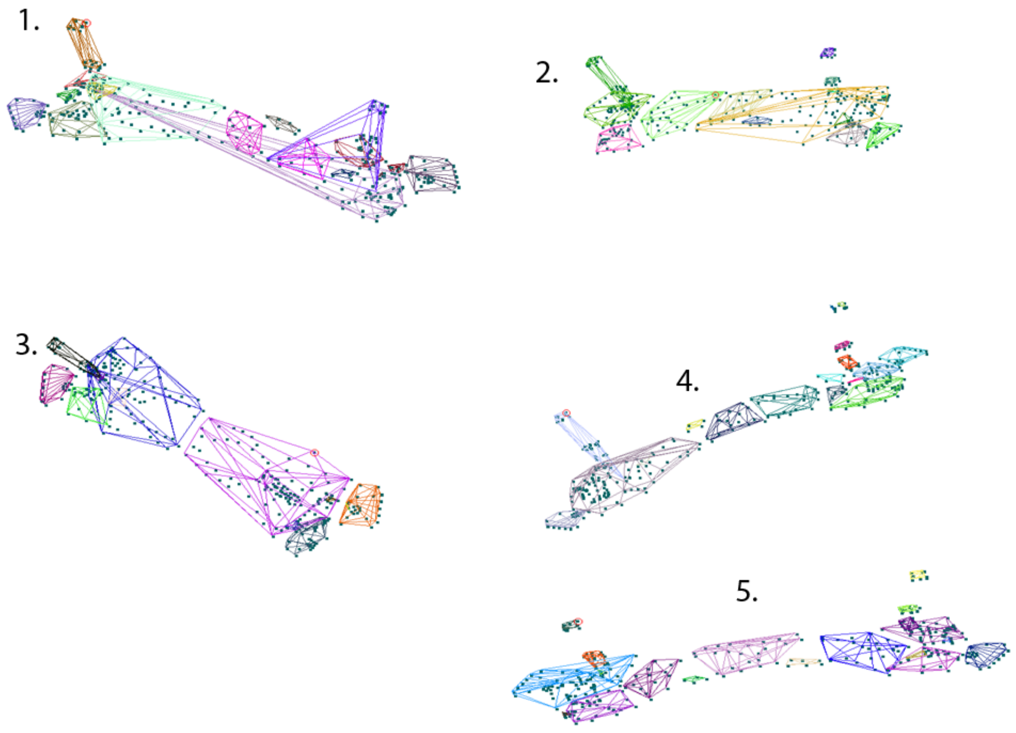


Figure A.34: Face Classic polygon fitting - Variance 10

A.1.5.0.29 Variance 10:

A.1.5.0.30 Variance 5: There is less separation here than in the previous variance value. There is also more evidence of the rogue point in one of the towers being connected to the other tower, which we are trying to avoid.

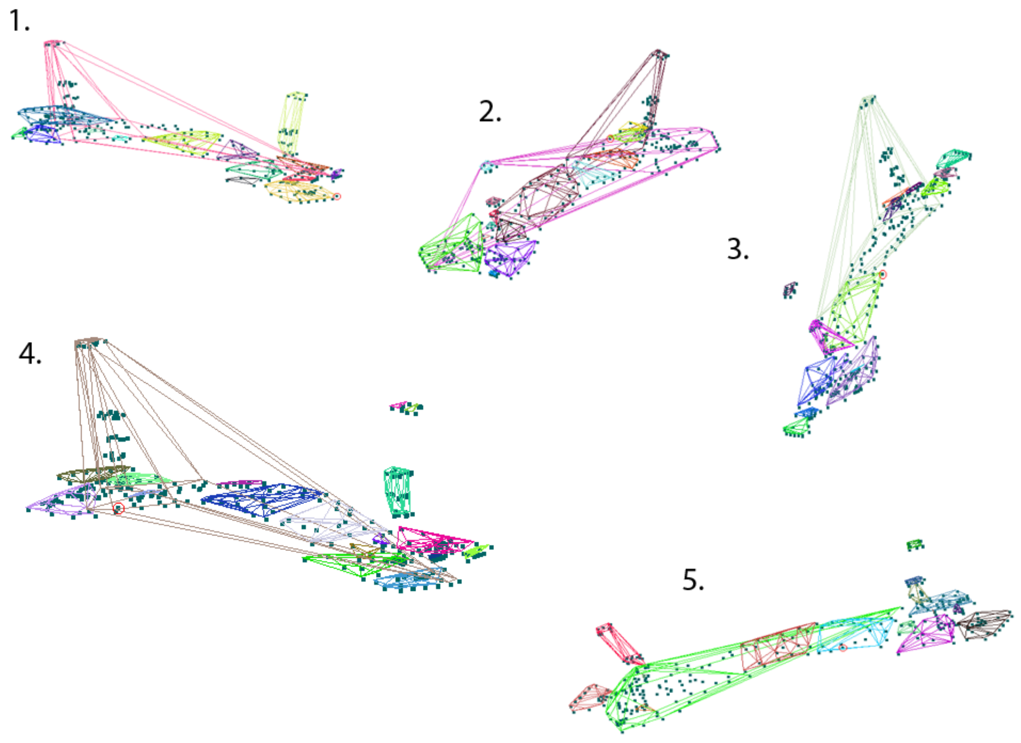


Figure A.35: Face Classic polygon fitting - Variance 5

A.1.5.0.31 Conclusions from visualisation: From the visualisation it is relatively obvious that the best value for the variance is 10. This has the best set of feature representations. Overall the separation of the key sections is much better than for face3. The pathways are better defined.

When we move down to a variance value of 5 some of the less desirable characteristics of the modelling procedure begin leaking back into the visualisations. Most notably one of the points in one tower becoming attached to the cluster for the other tower.

A.1.5.0.32 Conclusions from statistics: The statistics graph seems to suggest that there is very little to choose between any of the variance values. With this in mind it seems best to go with a variance value of 10 to match the conclusions from the visualisation data. Again there are very few degenerate clusters generated.

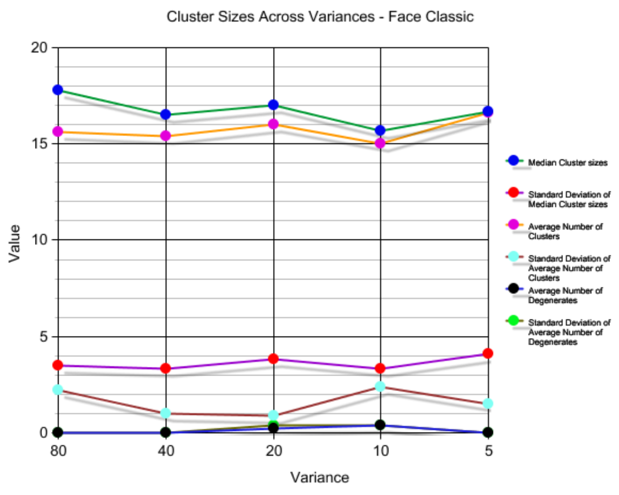


Figure A.36: Face Classic - Statistics

A.1.6 Maul

A.1.6.0.33 Variance 80: The model here tends to separate the path into several smaller sections. The end sections are at least quite well separated from each other. Occasionally the usual modelling artefact occurs with one point from one end section becoming absorbed into the other end's polygon.

A.1.6.0.34 Variance 40: The results here look pretty much the same as for the variance 80 section and the same modelling artefacts present themselves regularly. Case 3 is a good example of what might be an acceptable modelling output for this level. This is also the only case so far which has captured correctly the narrowing of the path in the centre area.

A.1.6.0.35 Variance 20: Apart from case 1 these results look worse as a whole. The sections are beginning to develop a large amount of overlap and ill-defined separation.

A.1.6.0.36 Variance 10: In this model the artefact is present in most cases but the actual separation of the polygons is generally pretty good. Case 1 also manages to represent the narrowing of the path.

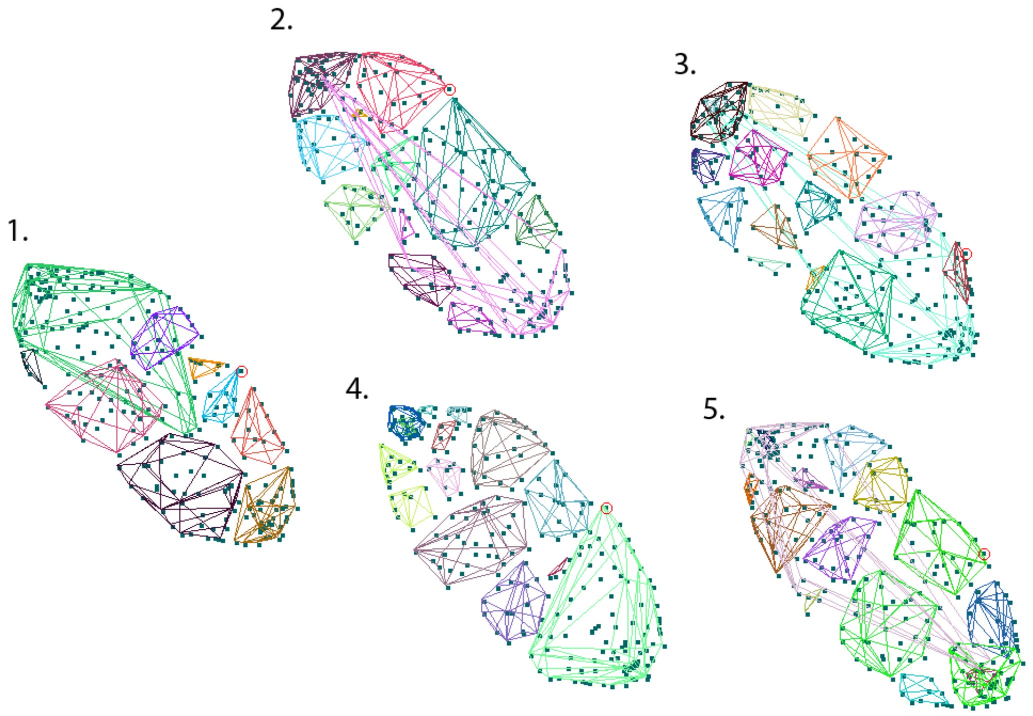


Figure A.37: Maul polygon fitting - Variance 80

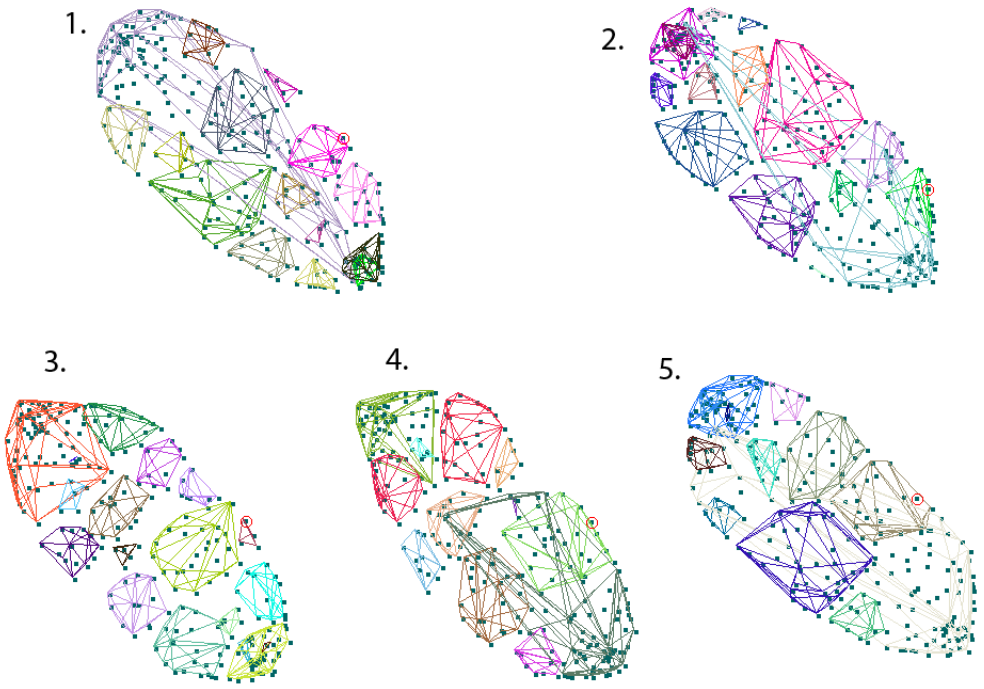


Figure A.38: Maul polygon fitting - Variance 40

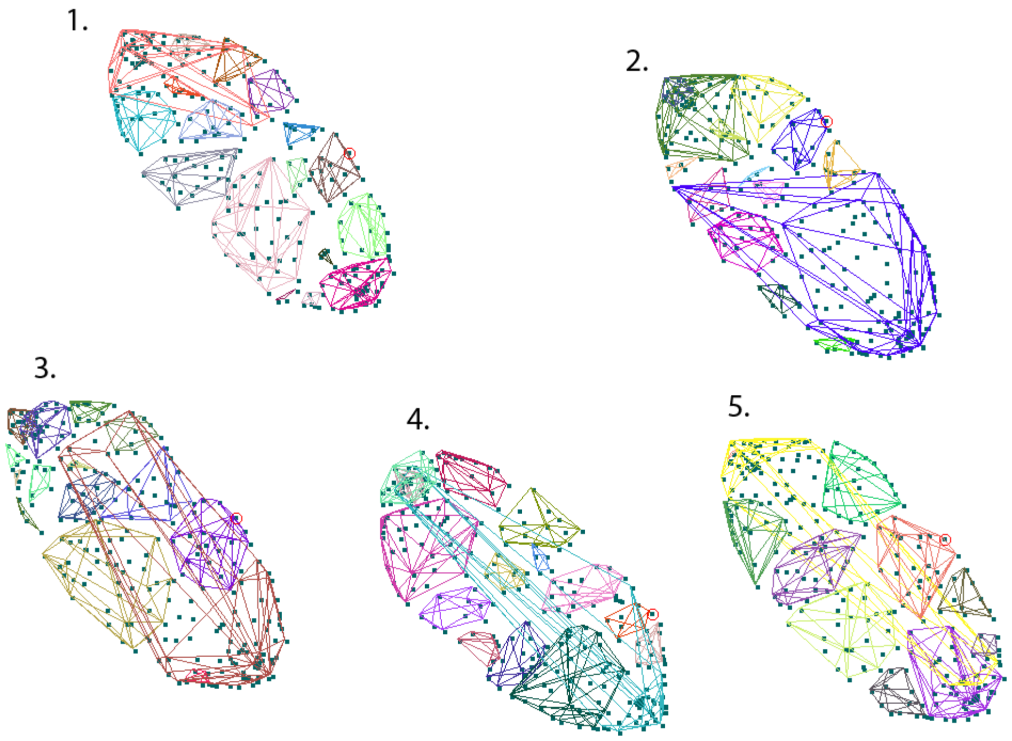


Figure A.39: Maul polygon fitting - Variance 20

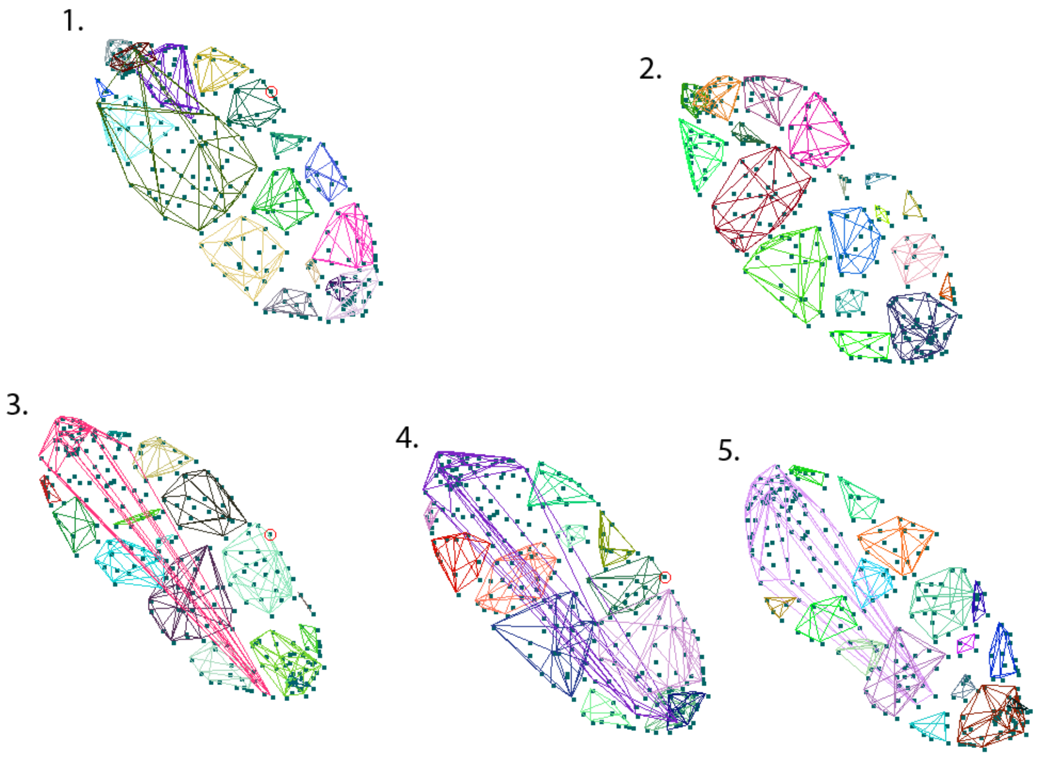


Figure A.40: Maul polygon fitting - Variance 10

A.1.7 Moondragon

A.1.7.0.37 Variance 80: Overall the modelling here isn't too bad. Some sections are ill defined but the temples at either end of the level are relatively well defined. The path sections often overlap or are badly defined but in general the models here don't look too bad.

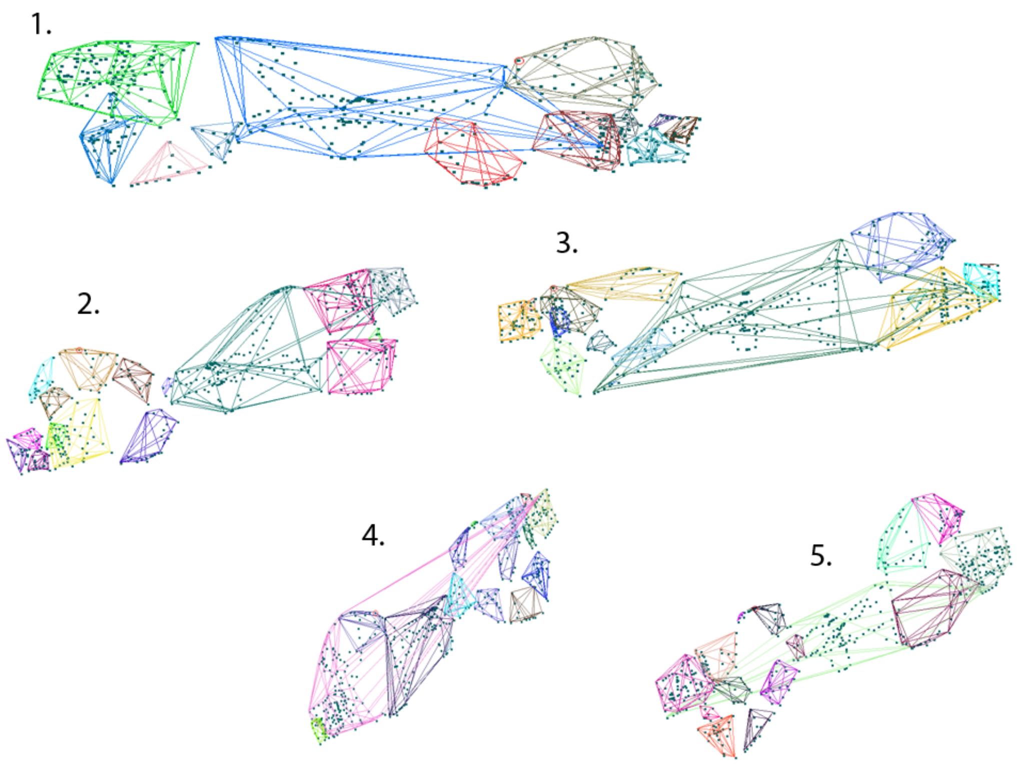


Figure A.41: Moon Dragon polygon fitting - Variance 80

A.1.7.0.38 Variance 20: This variance settings it noticeably worse than the 40 setting. The sections generated are much less well defined and very often are very large and over-reaching with little representation of the key features.

A.1.7.0.39 Variance 10: Again this not so good, this setting is very similar to that of 20.

A.1.7.0.40 Variance 5: This section gives slightly better results than for 10 and 20 but still not as good the 40 setting. The sections are better separated here but there are more artefacts of the modelling than for variance 40.

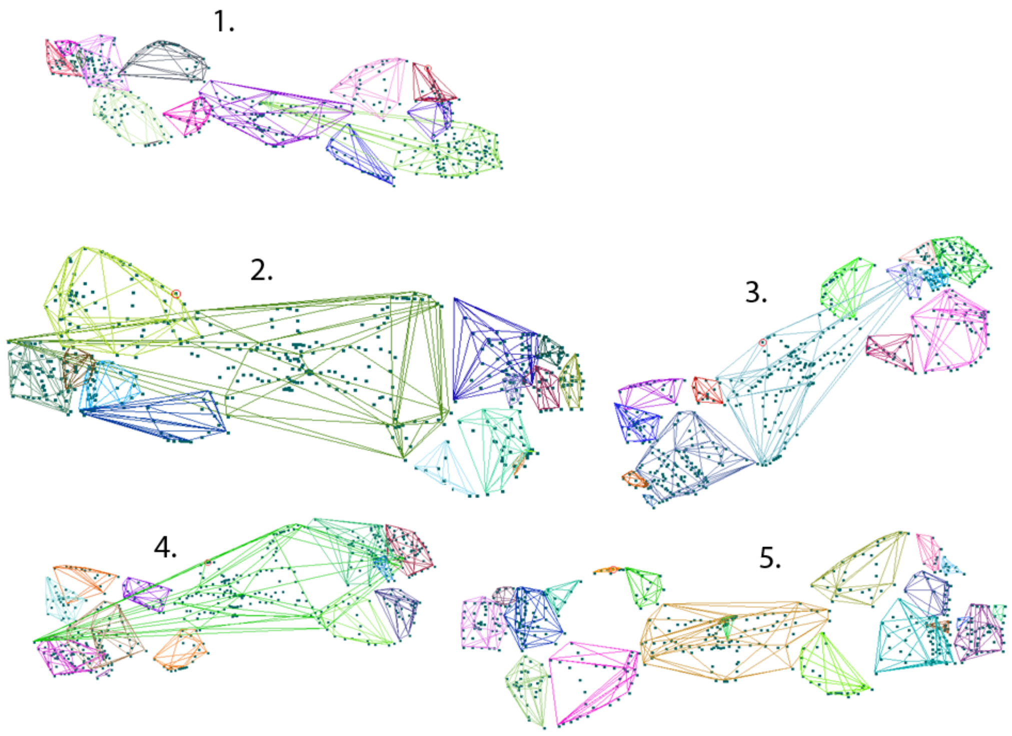


Figure A.42: Moon Dragon polygon fitting - Variance 20

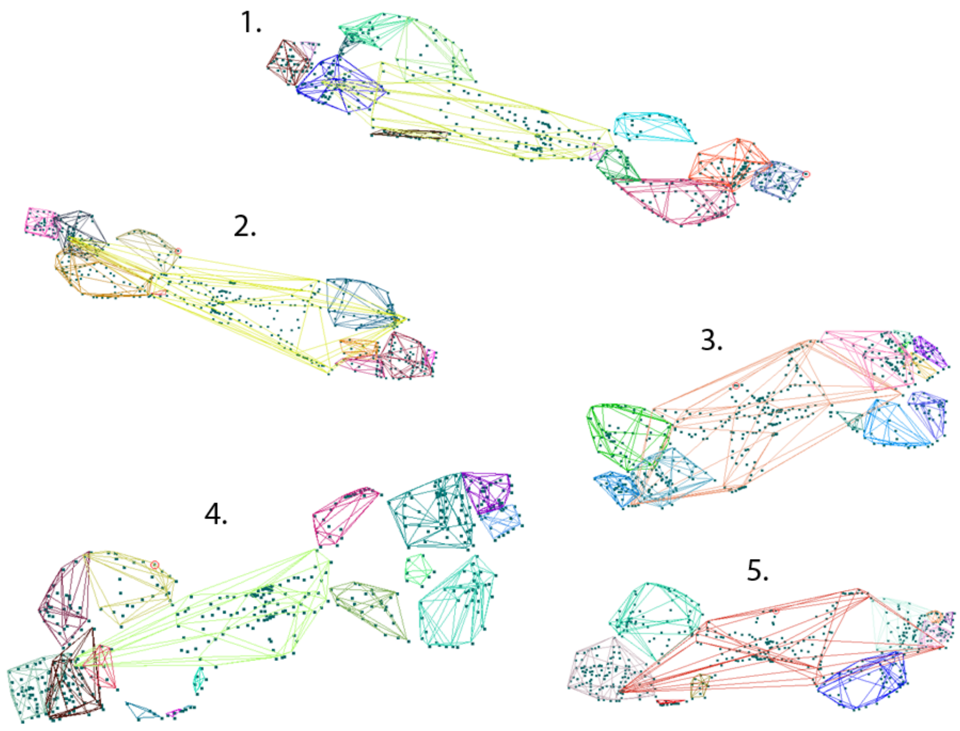


Figure A.43: Moon Dragon polygon fitting - Variance 10

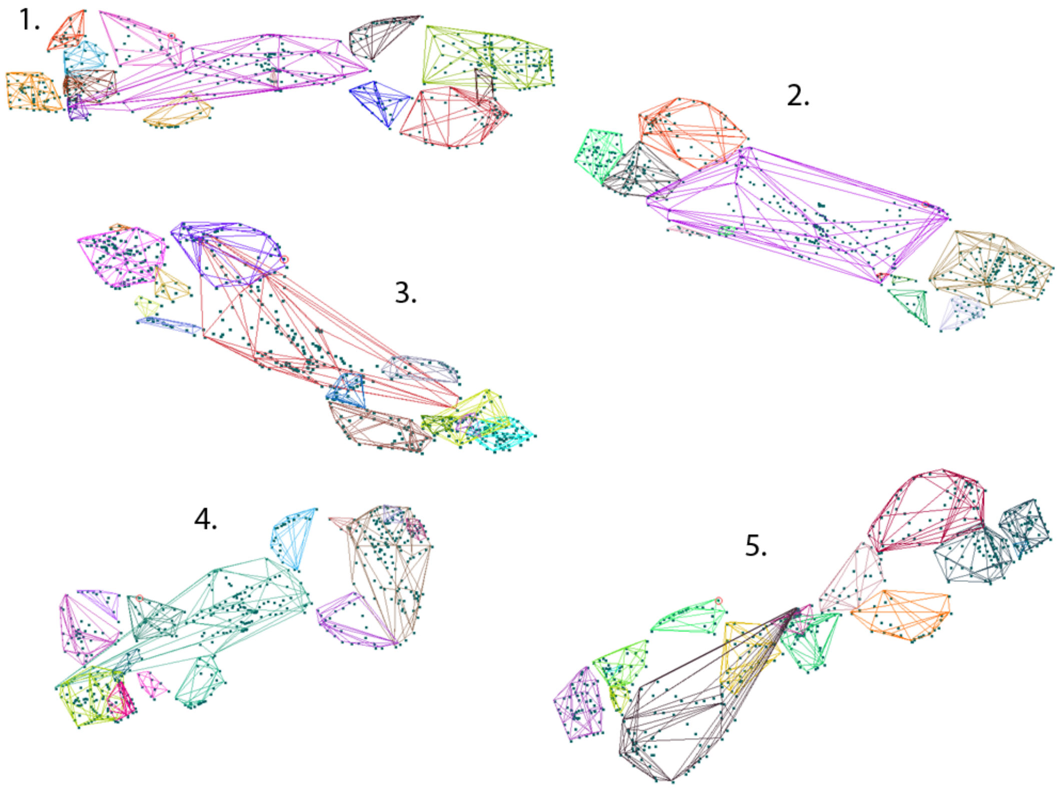


Figure A.44: Moon Dragon polygon fitting - Variance 5

A.1.8 Sulphur

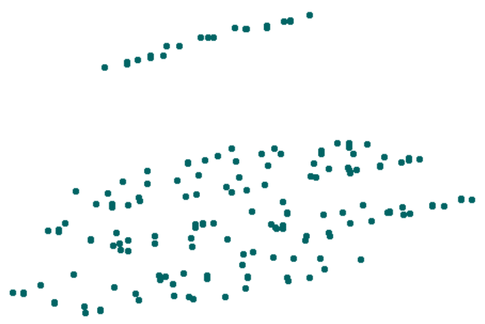


Figure A.45: Sulphur - 3D Navigation Point Data

Sulphur is a level set on an off-shore oil rig with 4 levels. The final and highest level floats above the main platform and is only accessible from jump pads on the level.

Magnitude = 4.69375 PCA Magnitude = 0.01128480057741909

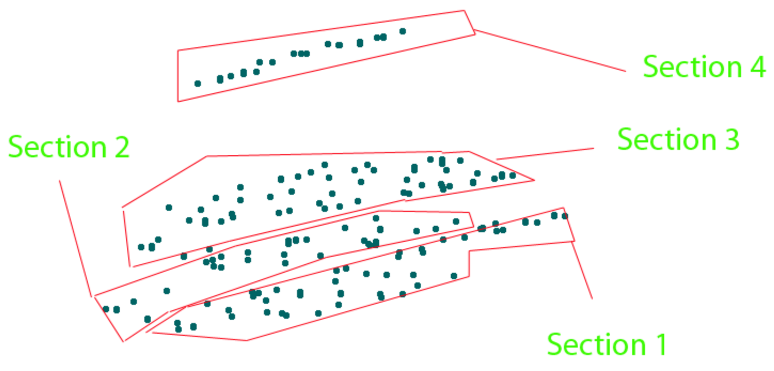


Figure A.46: Sulphur - Expected Polygons

A.1.8.0.41 Variance 80: In general the modelling here is not great. The problem is that, form a modelling perspective, there is more deviation between the nav points on a lateral level than there is between the different levels. This leads to polygons which span various levels. In some cases the section which floats above the level has been separated form the rest of the level. There is then a large amount of polygons which are separated from each other but which span the other 3 areas of level. These do seem relatively well space though to represent the actual level so these may still be useful.

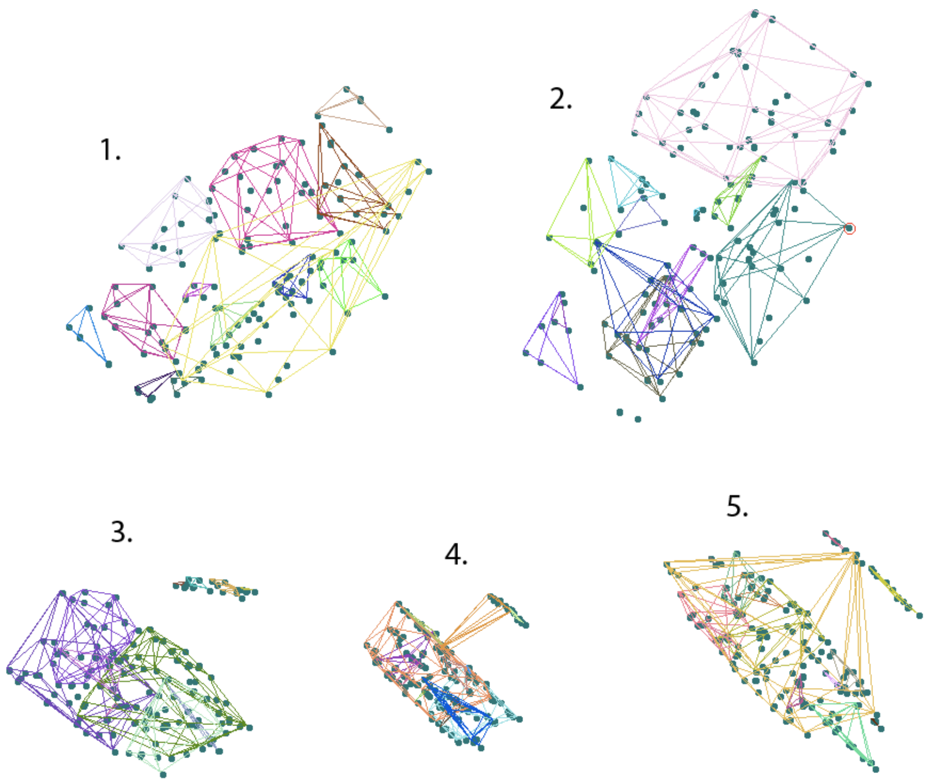


Figure A.47: Sulphur polygon fitting - Variance 80

A.1.8.0.42 Variance 40: The section above the level is slightly better separated form the level but in general the polygonal modelling in the actual main part of the level is pretty much the same.

A.1.8.0.43 Variance 20: this is the same as the variance 40 section except that the polygons in the level look slightly less well separated with one larger polygon representing most of the rest of the level.

A.1.8.0.44 Variance 10 Visualisation: This model is probably the best as the floating section is present in most cases and the polygons in the lower part of the level are spaced out well and representative of the level with very few points belonging to degenerate clusters.

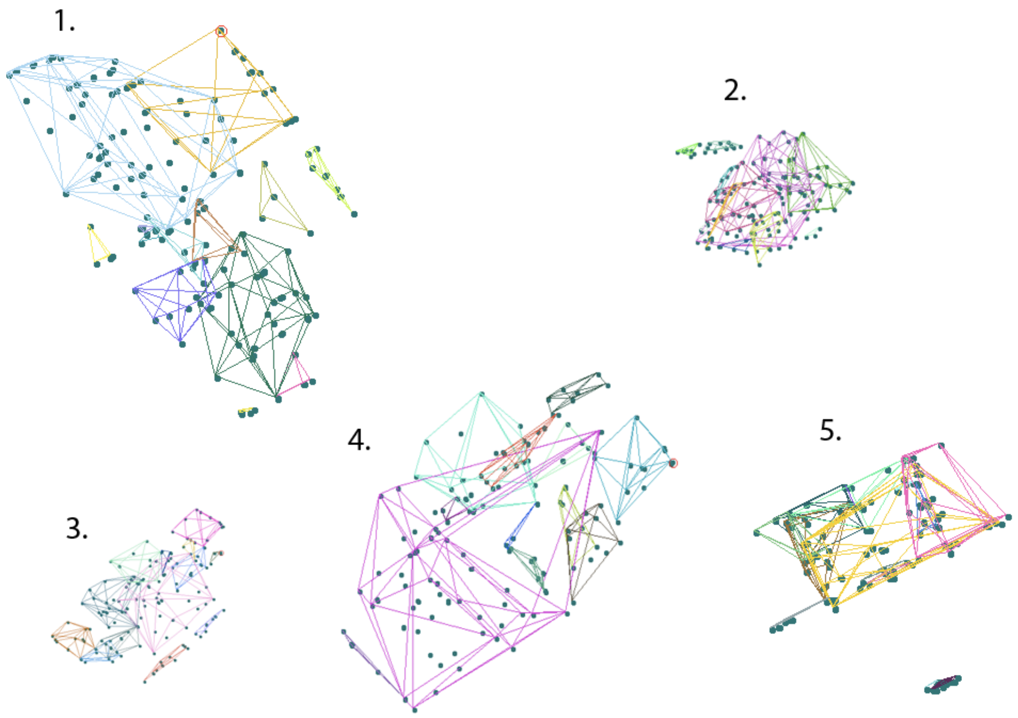


Figure A.48: Sulphur polygon fitting - Variance 40

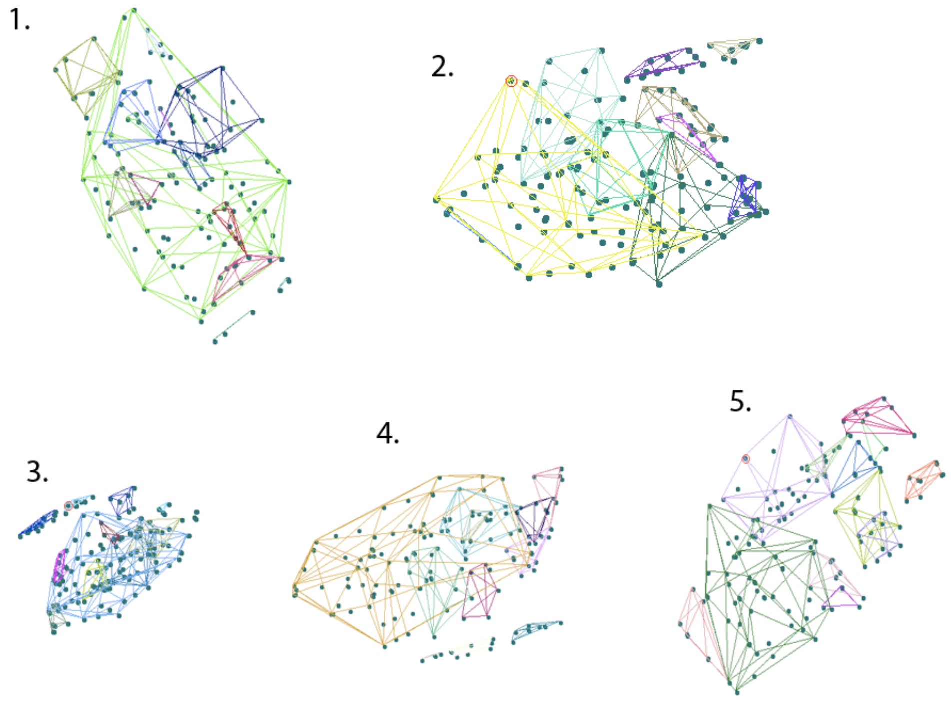


Figure A.49: Sulphur polygon fitting - Variance 20

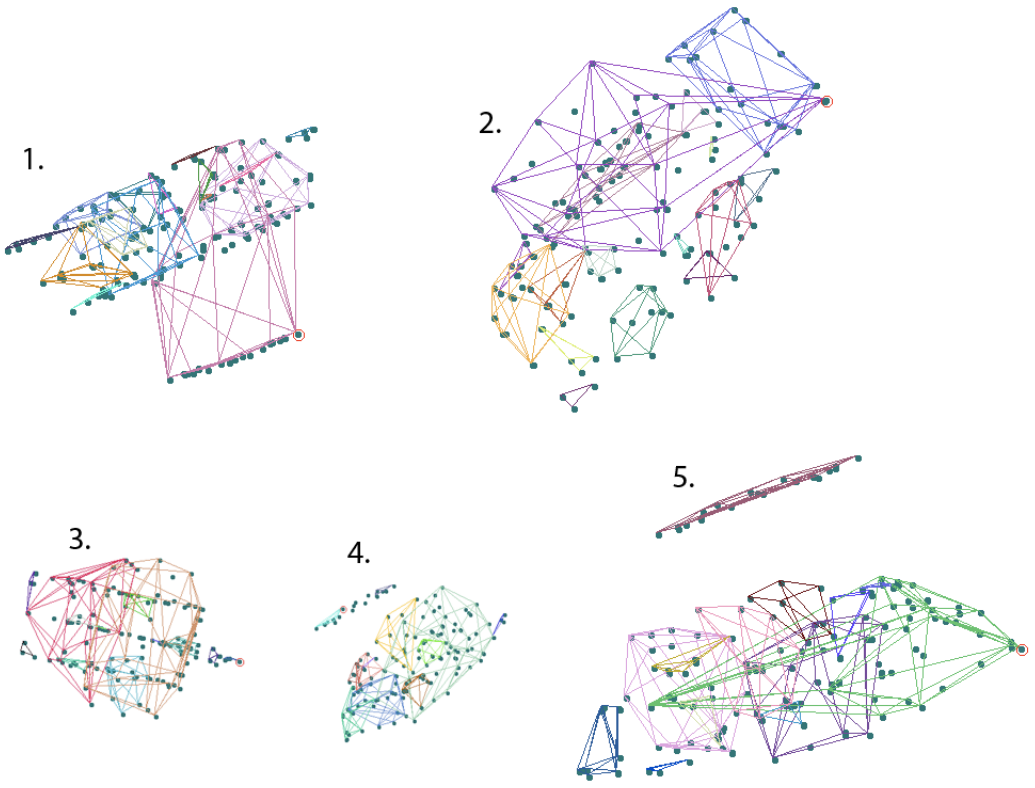


Figure A.50: Sulphur polygon fitting - Variance 10

A.1.8.0.45 Variance 5: This model is worse than the others as here is no real separation between the the top section and the rest of the level and the lower areas of the level area not well separated from each other.

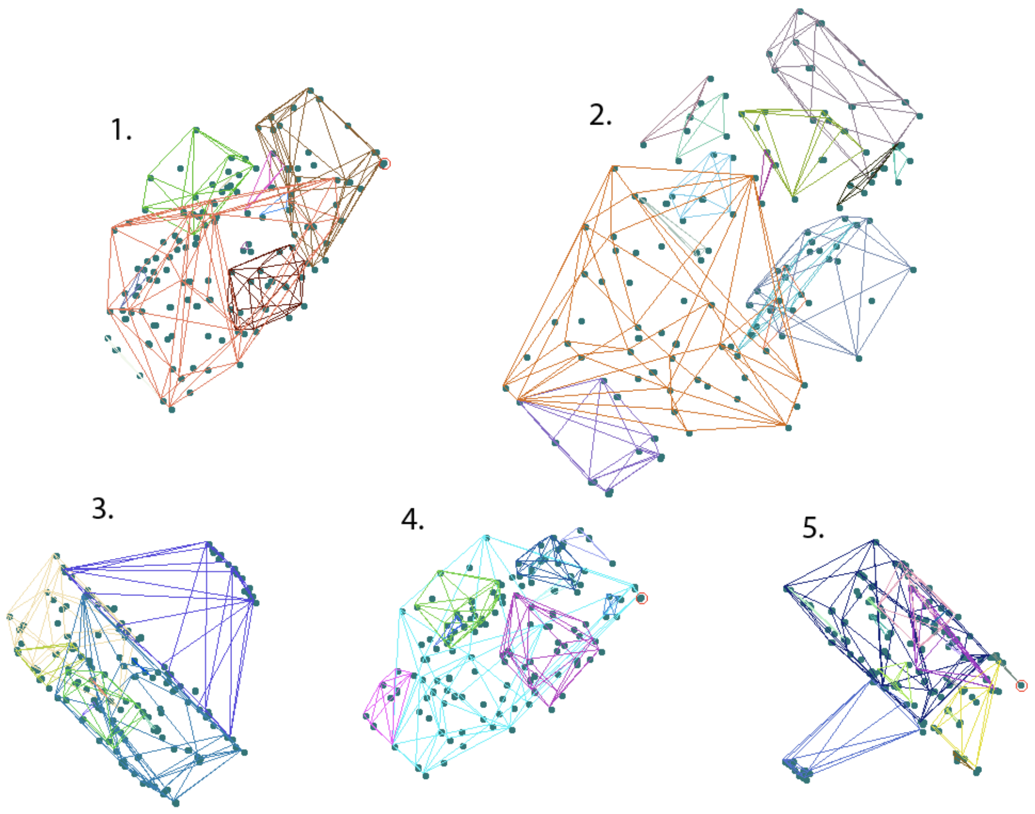


Figure A.51: Sulphur polygon fitting - Variance 5

A.1.8.0.46 Conclusions from visualisation: Again the visualisations seem to show a value of 10 as being optimal for modelling purposes. Although not perfect with regards to the difference sections of the level this variance still yields some polygons which seem to be sensibly separated enough to aid modelling.

The variance 10 model does slightly better at picking out section 4 than the others, this suggests it is best as section 4 is probably the most notable feature of the level.

A.1.8.0.47 Conclusions from statistics: The statistics graph also suggests that variance of 10 generates the smallest average number of clusters which may be indicative of a stable point in the modelling process.

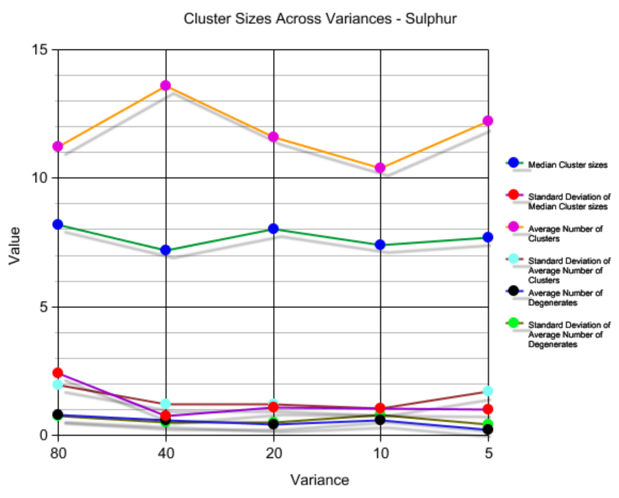


Figure A.52: Sulphur - Statistics

A.1.9 Sun Temple

A.1.9.0.48 Variance 80: The end two sections are very well picked up but the whole area in the middle is not terribly well defined and tends to become a bit messy with no distinct pattern to the actual sections and a lot of overlapping.

A.1.9.0.49 Variance 40: Again the middle sections are not perfect but the blurring is less severe and the generated areas seem more useful. In case 5 the end section has become attached to the polar opposite section which is not so good.

A.1.9.0.50 Variance 20: This variance setting leads to models which are roughly similar to those in the variance 40 section

A.1.9.0.51 Variance 5: This is very similar to the variance 10 models but the separation is probably slightly more so this model probably represents the best choice so far.

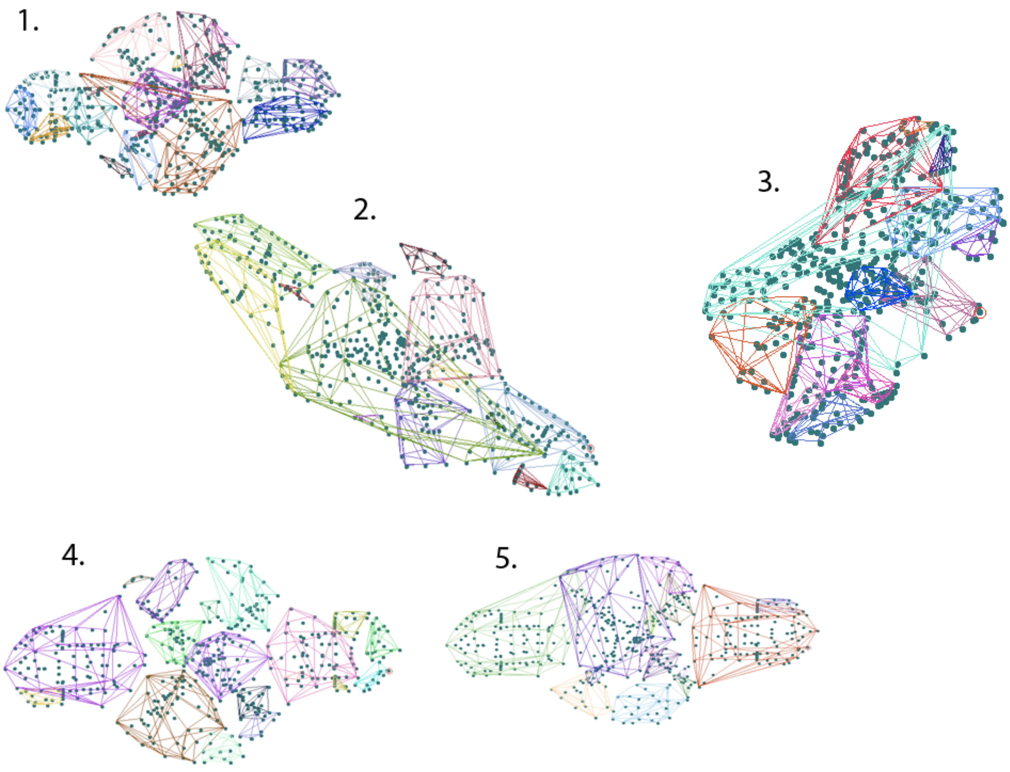


Figure A.53: Sun Temple polygon fitting - Variance 80

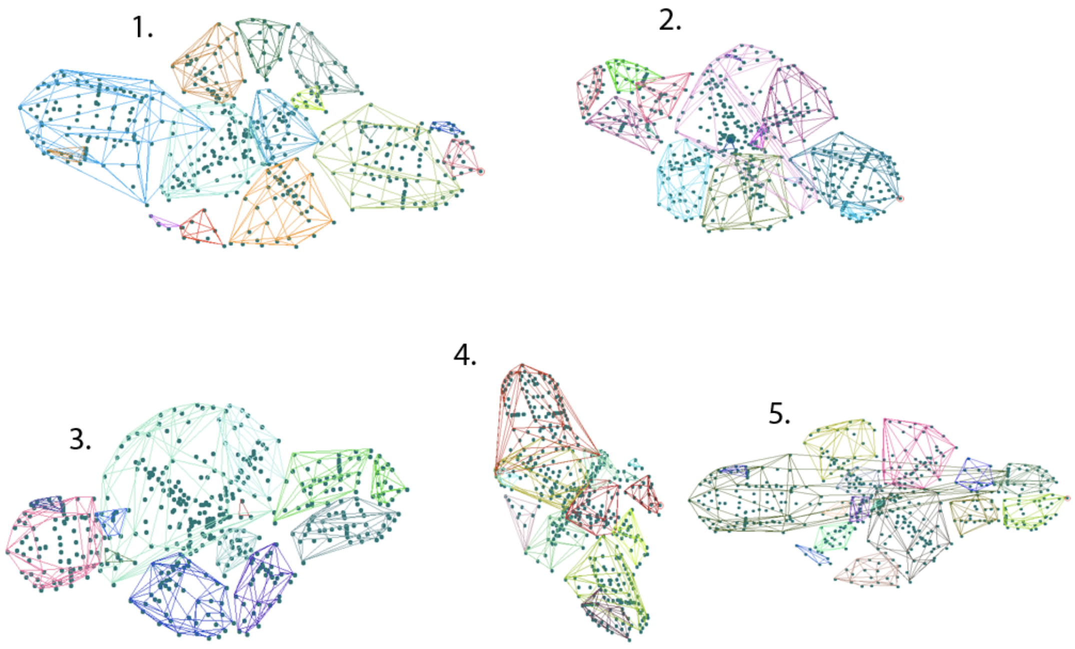


Figure A.54: Sun Temple polygon fitting - Variance 40

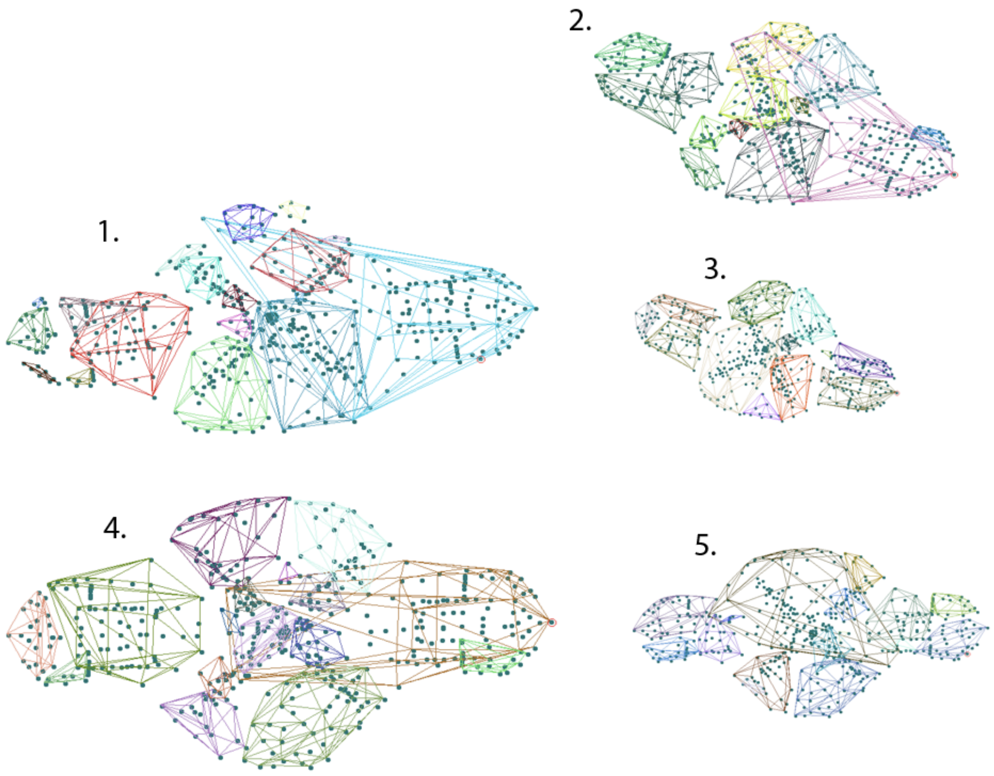


Figure A.55: Sun Temple polygon fitting - Variance 20

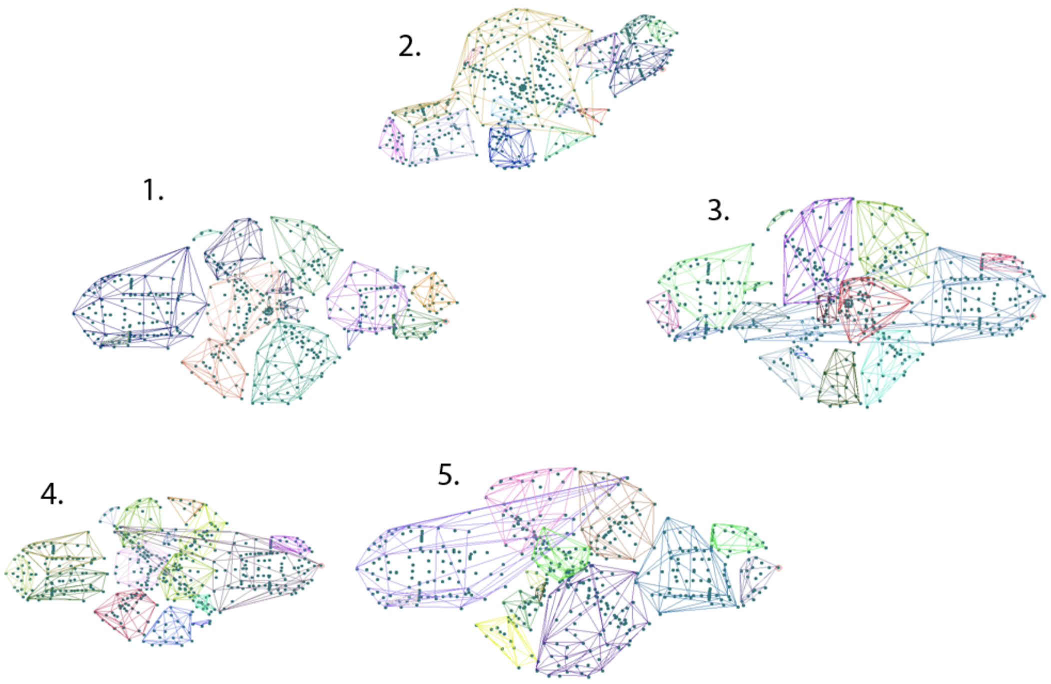


Figure A.56: Sun Temple polygon fitting - Variance 5

Appendix B

Appendix B

B.1 Area Correlation Model Extra Data

B.1.1 TDM

B.1.1.1 Sulphur

	Sulphur Area Correlation Trials, 5 Trial Averages	
	Enemy Value	Our Value
Average Score	60	43.8
Standard Deviation from Score	0	5.03

Figure B.1: Sulphur Area Correlation Model Trials

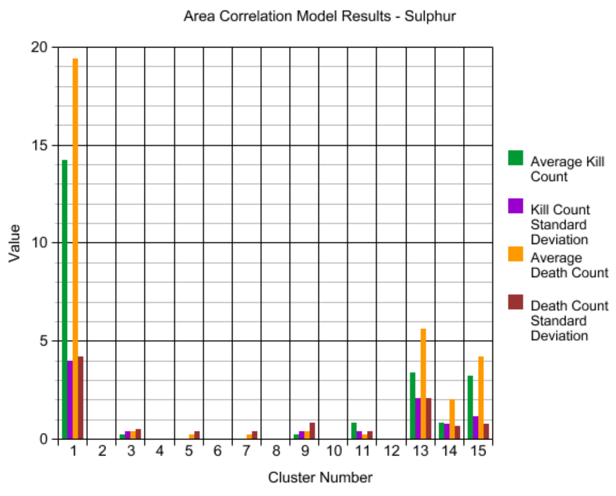


Figure B.2: Sulphur Area Correlation Model Statistics

B.1.1.2 Antalus

	Antalus Area Correlation Trials, 5 Trial Averages	
	Enemy Value	Our Value
Average Score	60	17.6
Standard Deviation from Score	0	4.27

Figure B.3: Antalus Area Correlation Model Trials

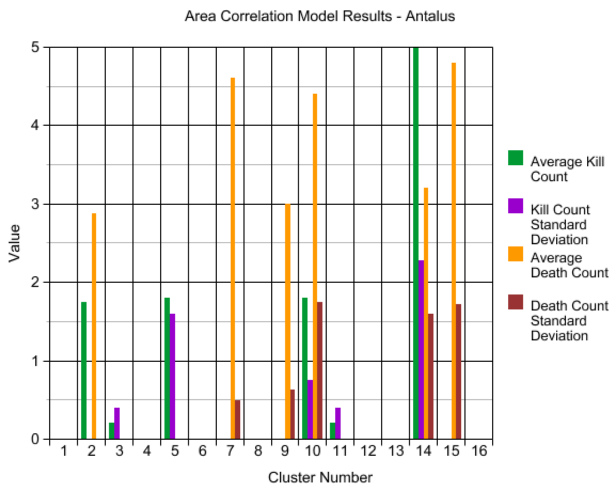


Figure B.4: Antalus Area Correlation Model Statistics

B.1.2 CTF

B.1.2.1 MoonDragon

	Moon Dragon Area Correlation Trials, 5 Trial Averages	
	Enemy Value	Our Value
Average Score	6	0
Standard Deviation from Score	0	0

Figure B.5: Moon Dragon Area Correlation Model Trials

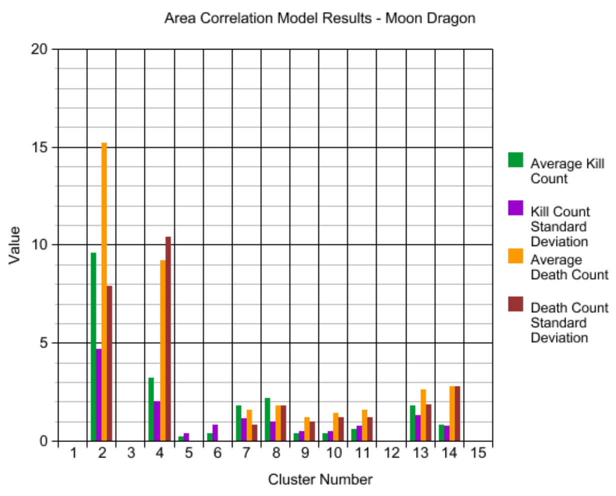


Figure B.6: Moon Dragon Area Correlation Model Statistics

	Grassy Knoll Area Correlation Trials, 5 Trial Averages	
	Enemy Value	Our Value
Average Score	6	0.2
Standard Deviation from Score	0	0.4

Figure B.7: Grassy Knoll Area Correlation Model Trials

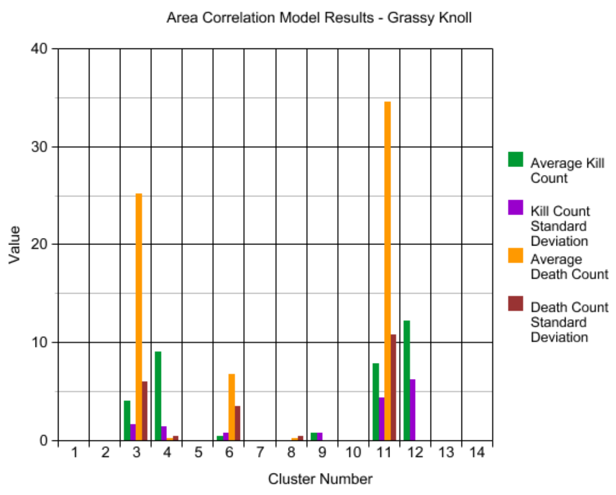


Figure B.8: Grassy Knoll Area Correlation Model Statistics

B.1.2.2 GrassyKnoll

B.1.3 DD

B.1.3.1 SunTemple

	Sun Temple Area Correlation Trials, 5 Trial Averages	
	Enemy Value	Our Value
Average Score	0.4	6
Standard Deviation from Score	0.48	0

Figure B.9: Sun Temple Area Correlation Model Trials

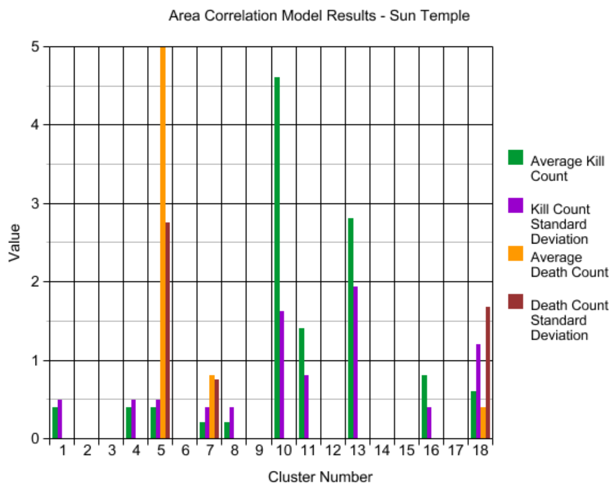


Figure B.10: Sun Temple Area Correlation Model Statistics

B.1.3.2 ScortchedEarth

	Scortched Earth Area Correlation Trials, 5 Trial Averages	
	Enemy Value	Our Value
Average Score	2.2	6
Standard Deviation from Score	1.72	0

Figure B.11: Scortched Earth Area Correlation Model Trials

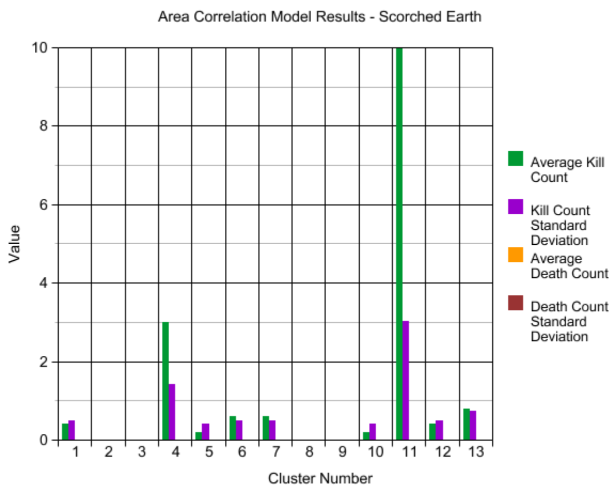


Figure B.12: Scortched Earth Area Correlation Model Statistics

Appendix C

Appendix C

C.1 Flag Approach Route Modelling

C.1.1 Evaluation 1

C.1.1.1 GrassyKnoll

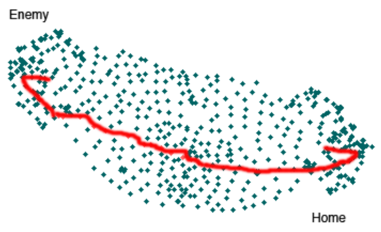


Figure C.1: Approach Path For Grassyknoll

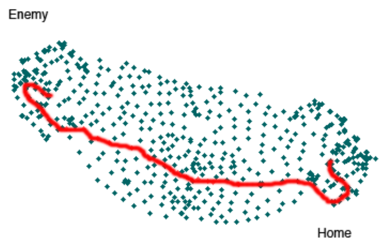


Figure C.2: Return Path For Grassyknoll

C.1.1.1.1 Figure C.1 shows the main path the bot takes to the enemy flag. Figure C.2 shows the route back from the enemy flag to home base.

C.1.1.1.2 Variance: 250 In both the approach and return cases the path actually starts out relatively well defined but as more reinforcements occur the path begins to get very narrow and a steep ridge develops. This is bad as we would like the smooth influence to be shown as in the case of the 500 model for citadel.

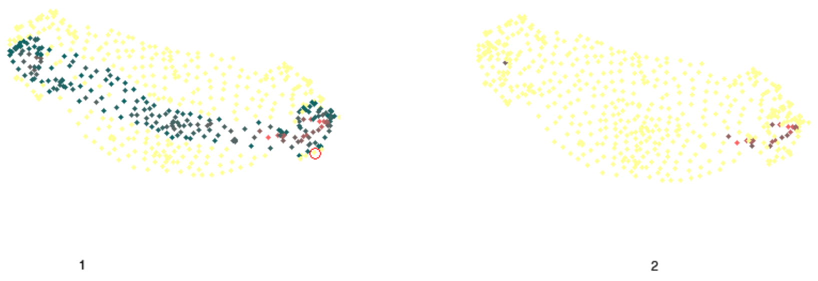


Figure C.3: Approach Density Model, with Variance 250, For Grassyknoll

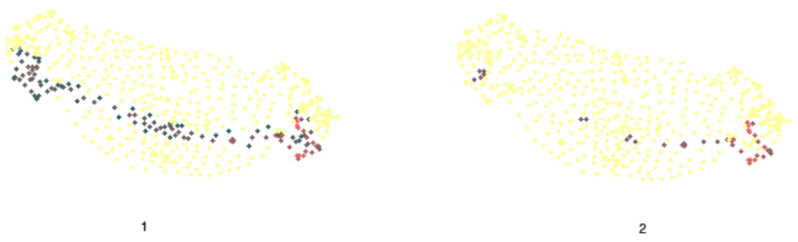


Figure C.4: Return Density Model, with Variance 250, For Grassyknoll

C.1.1.1.3 Variance: 500 Again the variance 500 model is much better with a smoother influence over the whole level. The return path is possibly slightly narrower than would be liked as the level progresses but there is still some level of influence. It should also be noted that the threshold has been put to a higher level for the right hand visualisation in both the 250 and 500 cases showing that the 500 model maintains its spread at a higher level of threshold.

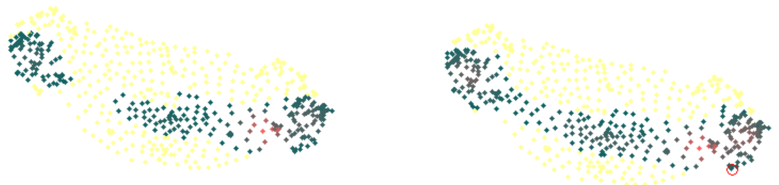


Figure C.5: Approach Density Model, with Variance 500, For Grassyknoll

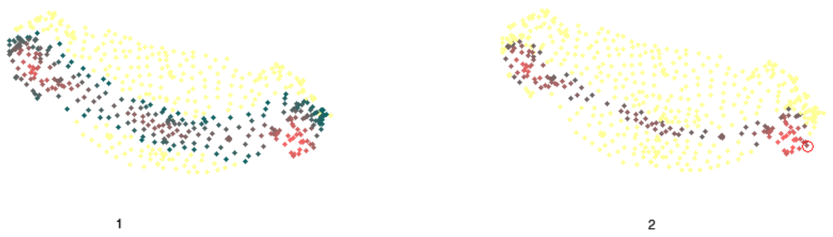


Figure C.6: Return Density Model, with Variance 500, For Grassyknoll

C.1.1.1.4 Variance: 1000 The model does give very good smooth influence over the nodes in both the low and high threshold cases but in the approach model it does seem to loose a portion of the path at one point. This is again down to the path re-inforcing too much the nodes around the path. As such if the path curves round a corner it can strongly reinforce the nodes which are close to both edges of the middle of the curve. The rescaling for visualisation purposes then ends up possibly loosing the section of the path which was on the actual path.

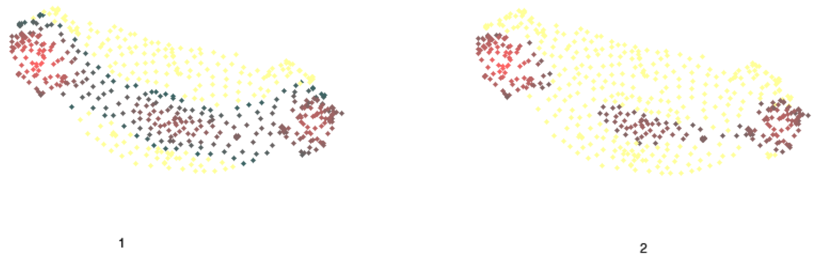


Figure C.7: Approach Density Model, with Variance 1000, For Grassyknoll

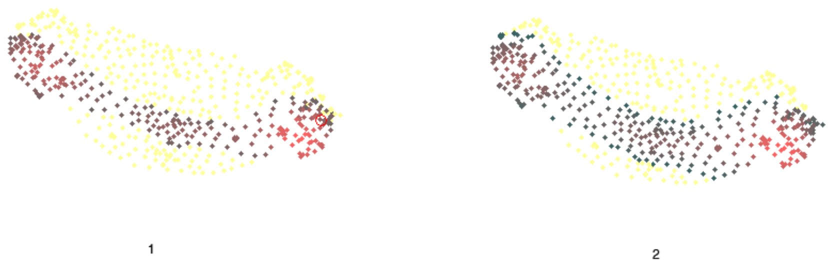


Figure C.8: Return Density Model, with Variance 1000, For grassyknoll

C.1.1.2 LostFaith

C.1.1.2.1 Figure C.9 shows the main path the bot takes to the enemy flag and back.

C.1.1.2.2 Variance: 250 The model here is suffering from the most severe case of the steep ridge syndrome discussed throughout this section. At high thresholds almost no reinforcement bar a few specific nodes is seen and any gradient is very narrow.

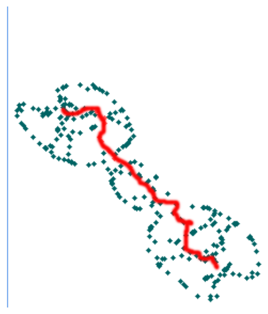


Figure C.9: Approach Path For LostFaith

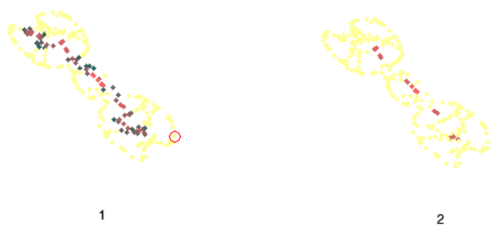


Figure C.10: Approach Density Model, with Variance 250, For LostFaith

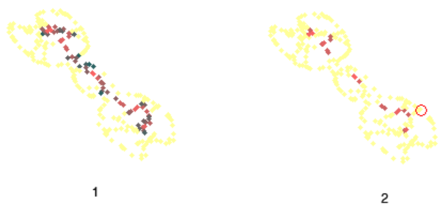


Figure C.11: Return Density Model, with Variance 250, For LostFaith

C.1.1.2.3 Variance: 500 This case is much better as the influence is still biassed to one side of the section in the middle of the level but not so biassed that it doesn't pass over some influence to the nodes around the area.

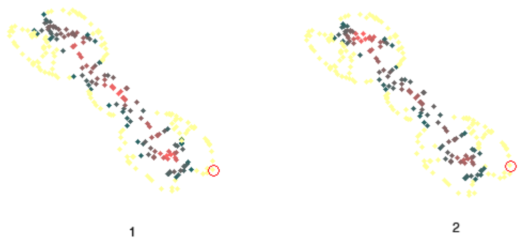


Figure C.12: Approach Density Model, with Variance 500, For LostFaith

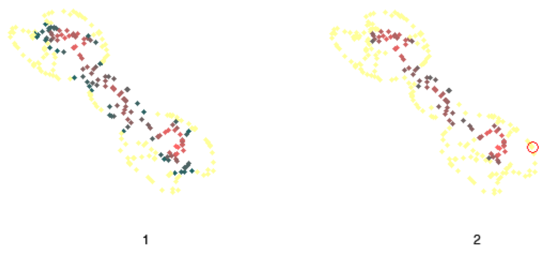


Figure C.13: Return Density Model, with Variance 500, For LostFaith

C.1.1.2.4 Variance: 1000 In the variance 1000 case all of the threshold levels had to be set very high in order to see any influence or spread across the level. Even when the influence was visible it is, again, to non-discriminative. The path either side of the middle section is highly reinforced when in fact only one side should be as this is the path taken.

C.1.2 Evaluation 2

C.1.2.1 GrassyKnoll

The GrassyKnoll results are particularly interesting. In both of the models the path taken is initially picked out as bad via negative reinforcement but then reverses at a

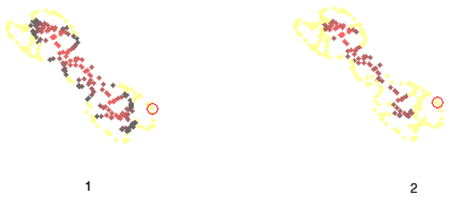


Figure C.14: Approach Density Model, with Variance 1000, For LostFaith

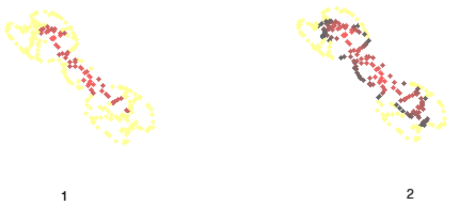


Figure C.15: Return Density Model, with Variance 1000, For LostFaith

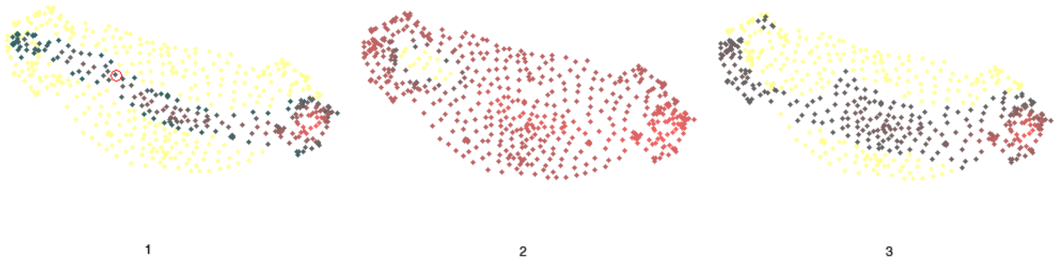


Figure C.16: Approach Route Model For GrassyKnoll

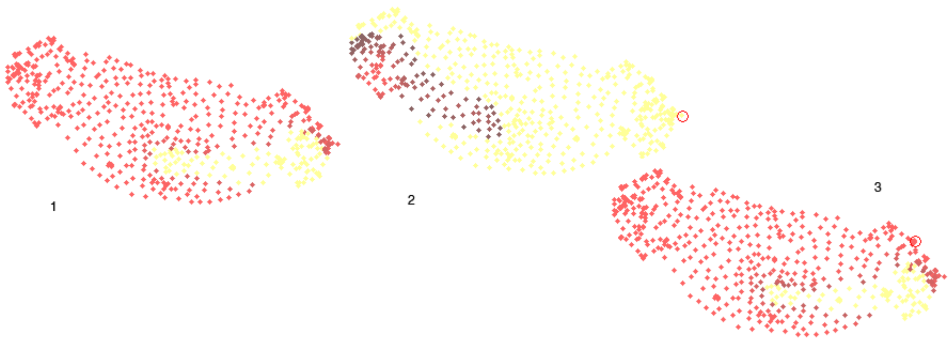


Figure C.17: Return Route Model For GrassyKnoll

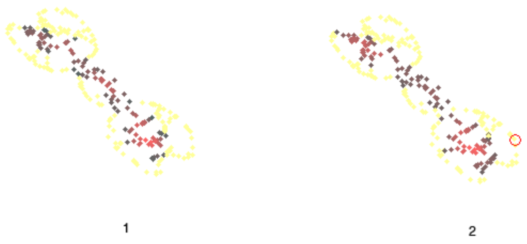


Figure C.18: Approach Route Model For LostFaith

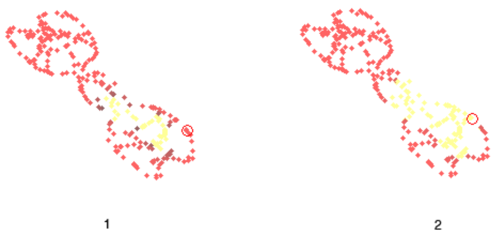


Figure C.19: Return Route Model For LostFaith

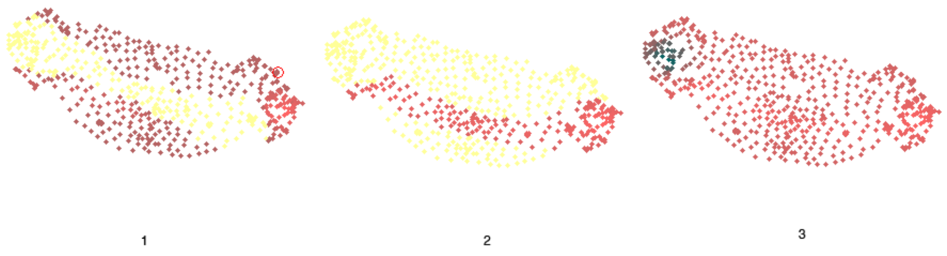


Figure C.20: Approach Route Model For GrassyKnoll

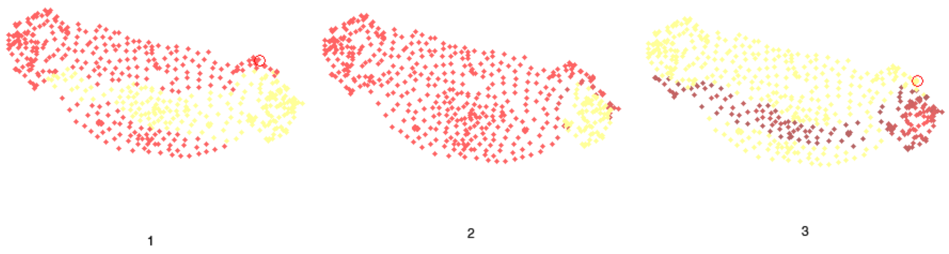


Figure C.21: Return Route Model For GrassyKnoll

later point in the match to reflect a successful capture. Towards the end of the match the approach model becomes very general and has little deviation across the level except a notable bad point within the enemy base. This makes sense as we would expect the most deaths to occur round the enemy flag point. This effect is somewhat mirrored in the return model as the home flag base is picked out as bad due to the most number of deaths occurring there. Even so his model eventually settles down into a solid model of the path taken.

C.1.2.2 LostFaith

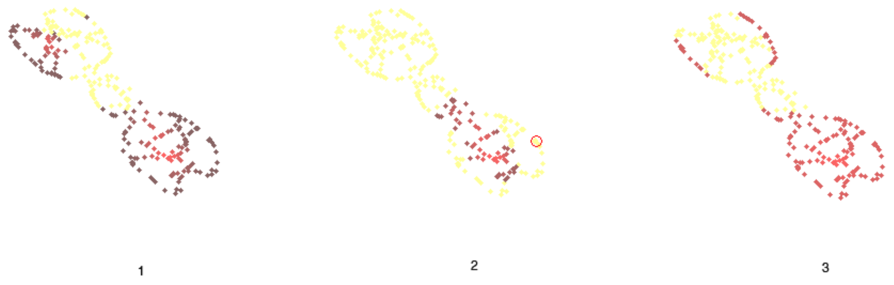


Figure C.22: Approach Route Model For LostFaith

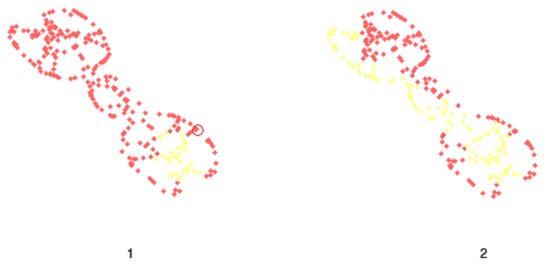


Figure C.23: Return Route Model For LostFaith

The approach model for Lost Faith is not the best. In particular no route through the central area is strongly reinforced. The side of the enemy base is slightly favoured on one side but the bias is not that strong. The home base is strongly reinforced but with the starting point of almost all paths beginning in that area this does not surprise.

The return model fairs slightly better. It start out quite general but quickly settles more into the situation shown in case 2. This picks out a side of the route through the central area of the level as being better. This is important because the central area of this level is an underground cave section separated by walls. Taking the most reinforced side of this cave is very important to navigating this level.

Appendix D

Appendix D

D.1 Domination Point Approach Route Modelling

D.1.1 Outrigger

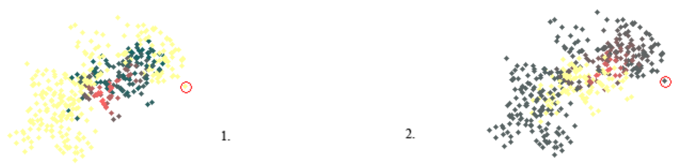


Figure D.1: Domination Point Approach Route for Point A, Outrigger Level

D.1.1.0.1 It is much harder to discern any real information regarding performance on this level. The route is reinforced but the nodes are so tight together that determining any particular routes is quite difficult. Situated results for this level would tell a much more interesting story as the bot's response to negative reinforcement would allow the other areas of the level to be explored.

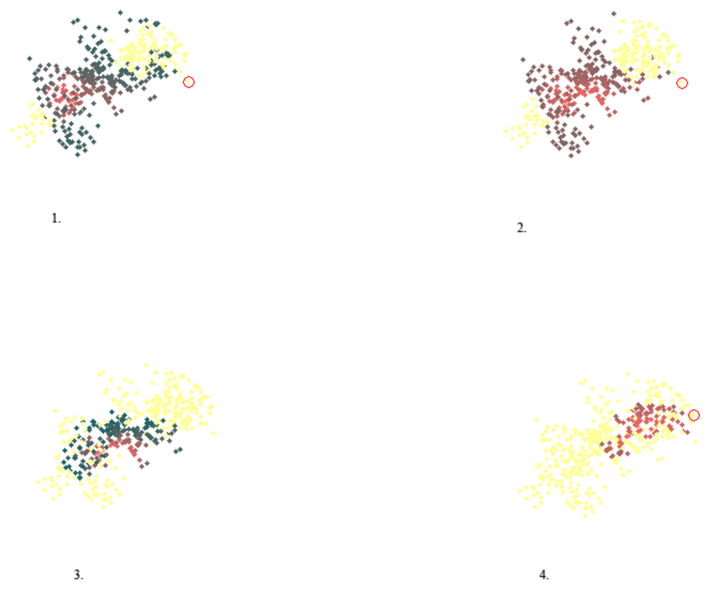


Figure D.2: Domination Point Approach Route for Point B, Outrigger Level