Operationalising the Simulation Theory for Intent Prediction in a Multi-Agent Adversarial Environment

by

Simon Butler

A Thesis submitted for the degree of Doctor of Philosophy of the University of London and Diploma of Imperial College

Department of Electrical and Electronic Engineering
Imperial College London
Exhibition Road, London SW7 2BT

2010

Abstract

Anticipation of an opposing team's actions and the ability to generate advantageous responses is an important feature when advising or controlling a team in an adversarial multi-agent system. Such systems include: complex video games, multi-robot systems, and military scenarios. The first step towards doing this is to observe the opposing team and predict their future actions, so that the system has a chance of selecting or advising the appropriate moves for the observing team to perform.

Theories from psychology and neuroscience provide insights into how the human mind performs such predictions, however, there is a large discontinuity between the theories and an actual implementation in a multi-agent AI system. This thesis bridges the gap between a biologically-inspired cognitive architecture based on the simulation theory of mind, and the implementation in a realistic multi-agent synthetic environment.

The basic simulation-theoretic architecture predicts actions by concurrently executing multiple hypotheses and then by finding the hypothesis that best matches the ongoing observed behaviour. This thesis presents work into operationalising this architecture: firstly by applying it to a custom-made real-time-strategy-style game and overcoming the challenges of this domain, which include, for example, the potentially constantly-changing and concurrently-executing goals, and the adaptation of behaviours to both the synthetic environment and the internal models that perform the simulations; secondly by reducing the computational burden of such a wide action-space through filtering and combining potential hypotheses; and finally by defining an attention mechanism to focus observation resources on the most beneficial areas.

The system is shown to be able to predict multiple manoeuvres, formations and targets involving the opponent's agents, from observations in large-scale adversarial environments. Considering the combination of the scalability of the architecture and the techniques for reduced usage of computational resources, it becomes more feasible to use such a principled approach on the current generation of hardware, moving away from ad-hoc approaches that rely on arbitrary measures of the opponent's intentions.

Acknowledgements

Firstly, I would like to thank my supervisor Yiannis Demiris for his support and guidance throughout my PhD years. Notably, agreeing to read paper drafts within mere hours of a deadline is greatly appreciated. Particular thanks also goes to Bálint Takács for his advice and help through some epic debugging sessions (e.g. "the tanks are inexplicably falling through the ground!", or "how can it be crashing inside *_ZN9ode7quickworldstep8ERSo*?").

Many thanks goes to the past and present members of the BioART lab for their friendship, advice and coffee break discussions. In strict alphabetical order: Tom Carlson, Ant Dearden, Murilo Fernandes Martins, Matt Johnson, KyuHwa Lee, Harold Soh, Paschalis Veskos and Yan Wu. Also, not forgetting the lunchtime regulars: Sunny Bains, Samuel Bayliss, Mercedes Lahnstein and Julia Schaumeier.

Finally, this thesis would not have been possible without the loving support of my parents David and Tricia, and my sisters Claire and Amy.

Declaration

I hereby declare that I composed this thesis entirely by myself and that it describes my own research.

Simon Butler
London
2010

Contents

Abstract **2**

Acknowledgements **3**

Declaration **4**

List of Figures **13**

List of Tables **14**

1 Introduction **15**

1.1 Thesis Aims 15

1.2 Importance of Predicting Intent 16

1.2.1 Military 17

1.2.2 Games 17

1.2.3 Robotics 17

1.3 Thesis Contributions 18

1.4 Thesis Outline 18

2 Background **21**

2.1 Introduction 21

2.2 Theory of Mind 21

2.2.1 Intentions and Actions 21

2.2.2 Definitions 22

2.2.3 Theories of Theory of Mind 22

2.2.4 Theory Theory 22

2.2.5 Simulation Theory 23

2.3 Computational approaches to intention recognition . . . 23
2.3.1 Descriptive Models . . . 23
2.3.2 Generative Models . . . 24
2.4 Multi-agent intent prediction . . . 24

3 Technical Background **27**
3.1 Simulation Theory for Intent Prediction . . . 27
3.1.1 HAMMER . . . 28
3.2 Scenario Domain . . . 29
3.2.1 Multiagent Systems . . . 29
3.2.2 Simulator . . . 30
3.2.3 Environmental Requirements . . . 31

4 Operationalisation **33**
4.1 Introduction . . . 33
4.2 HAMMER in the Multi-Agent Domain . . . 34
4.2.1 Adapting the architecture . . . 34
4.2.2 Synthetic Environment . . . 37
4.2.3 Multi-agent Internal Models . . . 37
4.2.4 Hypothesis generation . . . 39
4.2.5 Evaluation . . . 40
4.3 System Implementation . . . 42
4.3.1 Synthetic Environment . . . 42
4.3.2 Hypothesis Generation . . . 51
4.3.3 Distributed hypothesis clients . . . 54
4.3.4 Evaluation function . . . 57
4.4 Experiments . . . 59
4.4.1 Prototypical Scenario . . . 59
4.4.2 Results . . . 62
4.4.3 Intent Experiment . . . 66
4.5 Discussion . . . 69
4.6 Summary . . . 70

5 Parameterisation **71**

5.1 Introduction . . . 71

5.1.1 Complexity . . . 71

5.2 Inverse Models and Parameterisation . . . 72

5.2.1 Hypothesis sets . . . 73

5.3 Iterative granular parameterisation . . . 74

5.3.1 Experiments . . . 75

5.3.2 Results . . . 79

5.3.3 Discussion . . . 82

5.4 Parameter clustering . . . 83

5.4.1 Path Generation . . . 84

5.4.2 Path distance metric . . . 87

5.4.3 Path clustering . . . 88

5.4.4 Hypothesis sets . . . 89

5.4.5 Experiments . . . 89

5.4.6 Results . . . 92

5.4.7 Discussion . . . 94

5.5 Comparison of Parameterisation Approaches . . . 94

5.5.1 Experiments . . . 95

5.5.2 Results . . . 95

5.5.3 Discussion . . . 99

5.6 Summary . . . 99

6 Attention **100**

6.1 Introduction . . . 100

6.2 Attention mechanisms . . . 101

6.3 Partial Observability . . . 101

6.3.1 Objectives . . . 103

6.4 Implementation . . . 103

6.4.1 Observations . . . 103

6.4.2 System operation . . . 104

6.4.3 Threat-Based Attention . . . 105

6.5 Experiments 107
6.6 Results 109
6.6.1 Comparison between full and partial observability 109
6.6.2 Comparison of attention mechanisms 111
6.7 Discussion 115
6.8 Summary 115

7 Conclusions 117
7.1 Overview of the Thesis and its Contribution 117
7.2 Future Directions 119
7.2.1 Optimisations 119
7.2.2 Deception 119
7.2.3 Autonomous Decision Making 120
7.3 Epilogue 120

References 121

A Publications 128

B Synthetic Environment 129

C Ontology 133
C.1 Unit properties 133
C.1.1 Dynamic properties 133
C.1.2 Static properties 133
C.2 UnitFiring properties 134
C.2.1 Dynamic properties 134
C.2.2 Static properties 134
C.3 UnitTank properties 135
C.3.1 Dynamic properties 135
C.3.2 Static properties 135
C.4 UnitSoldier properties 136
C.4.1 Static properties 136

D Cubic Spline Interpolation 137

E Configuration Files **139**

F Intent Experiment Results **141**

F.1 Introduction 141

F.2 Results 141

G Example Observer Scenario **147**

G.1 Introduction 147

G.2 Example Walkthrough 147

List of Figures

3.1 The HAMMER architecture 28

4.1 The simulationist architecture implemented into a multi-agent framework . 35
4.2 The timing of generating a prediction 36
4.3 System implementation 41
4.4 System event loop 42
4.5 Simulator screenshot 43
4.6 Sequence diagram for starting the server 46
4.7 Sequence diagram for starting the client 47
4.8 Unit class diagram 47
4.9 Confidence for different prediction durations 54
4.10 Sequence diagram for the server 56
4.11 Sequence diagram for the hypothesis clients 56
4.12 Example calculation of the confidence function 57
4.13 Weighted graphs describing each formation 60
4.14 Timing diagram for hypotheses 61
4.15 Unit movements 62
4.16 Timewindowed confidences of each model 63
4.17 Raw confidences of each model 63
4.18 Effect of attacking and retreating hypotheses when moving in a line formation 64
4.19 Comparison of manoeuvre confidences of all the units or just the formation leader 65
4.20 Comparison of formation confidences including and excluding the formation leader 65
4.21 Unit movements 67

4.22 Results of the inverse model test 67

5.1 Example of the iterative granular parameterisation applied to target direction 75
5.2 An illustration of the timings for each of the three different numbers of segments used in the experiment. 77
5.3 A screenshot of the 2D map GUI for the IGP experiment 78
5.4 The percentage of targets correctly predicted by the time the attacker reached each target 80
5.5 The average number of targets in the maximum confidence segment 81
5.6 The percentage of times the maximum confidence segment contains only the correct target, over the whole scenario 81
5.7 The percentage of times the target was reached and the maximum confidence segment at any point contains only the correct target 82
5.8 An example of path clustering vs IGP 84
5.9 Examples for each of the possible path deviation scores 87
5.10 Two examples of the path distance metric being used on pairs of paths . . . 87
5.11 Scenario overview 90
5.12 Predicted paths 91
5.13 Timing diagram for straight-line and path-to-target hypotheses 92
5.14 Target confidence using the straight-line-to-target hypothesis 93
5.15 Target confidence using the path-to-target hypothesis 93
5.16 Timing diagram showing the number of parameters for the path-to-target and the IGP hypotheses 95
5.17 Screenshot of the map interface showing the maze scenario 96
5.18 Screenshot of the 3D interface showing the same scenario 96
5.19 How often the correct target predicted for both the clustering and IGP approaches 97
5.20 The number of clusters chosen throughout a trial 97
5.21 The reduction in the number of parameters that are simulated 98

6.1 Observation-based system operation 104
6.2 Timing diagram of an observation 105
6.3 Observation request scheduling flow diagram 106

6.4 Example scenario trace 108
6.5 Timing diagram showing full observability vs. partial observability 109
6.6 Comparison between correct predictions using full and partial observability 110
6.7 Comparison between latencies using full and partial observability 110
6.8 Predictions with one observer 111
6.9 Timing diagram showing multiple observers using one client 112
6.10 Predictions with 2 observers 113
6.11 Predictions with 3 observers 113
6.12 Threat-based predictions with number of attackers normalised to number of observers 114
6.13 RR predictions with number of attackers normalised to number of observers 114

B.1 A collaboration diagram showing the relation between most of the classes . 130
B.2 The functions that are called from the initialisation method of the Application class 131
B.3 The functions that are called from the preframe method of the Application class 132

F.1 Trial 1 scenario 142
F.2 Trial 1 results 142
F.3 Trial 2 scenario 143
F.4 Trial 2 results 143
F.5 Trial 3 scenario 144
F.6 Trial 3 results 144
F.7 Trial 4 scenario 145
F.8 Trial 4 results 145
F.9 Trial 5 scenario 146
F.10 Trial 5 results 146

G.1 Example observer scenario 148
G.2 Observer movements until unit 7 is reached 149
G.3 Observer movements until unit 12 is reached 150
G.4 Observer movements until unit 9 is reached 150
G.5 Observer movements until unit 8 is reached 151

G.6 Observer movements until unit 4 is reached 151
G.7 Observer movements until unit 11 is reached 152
G.8 Observer movements until unit 6 is reached 152
G.9 Observer movements until unit 10 is reached 153
G.10 Observer movements until unit 3 is reached 153
G.11 Observer movements until unit 5 is reached 154

List of Tables

4.1 Results of the intent accuracy experiment 68

6.1 Correlation coefficient 112

Chapter 1

Introduction

So it is said that if you know your enemies and know yourself, you can win a hundred battles without a single loss.

The Art of War
SUN TZU

As the modern proverb *"know thy enemy"* suggests, knowledge of the opponent is vital in adversarial situations. This knowledge can be used to understand and anticipate the actions of the opponent. However, this skill does not necessarily require years of training—it is an common human trait to be able to recognise and predict the intentions of others purely by observing them. For example, if someone playing football observes an opposing attacking player drawing back his leg when inside the penalty box, they can correctly hypothesise that the player is about attempt a shot on goal; this information can then be used to attempt to block the shot.

Whilst it is natural for humans to attempt to infer the intentions of an opponent in competitive environments, there are cases where it is necessary for a computational approach to this problem. For example, to provide assistance to operators by generating predictions when the problem space is large, or to generate and use the predictions when automatically controlling agents.

1.1 Thesis Aims

The motivation for this work is to take a principled approach into predicting the intent of other agents in an adversarial environment by gaining inspiration from how the human mind is hypothesised to perform such tasks. The obvious way to obtain the objectives of the other agents is to simply communicate with them and ask what they intend to do. However, in many situations this is infeasible, because in an adversarial setting an opponent will be unwilling to hand over its plans and tactics and in co-operative tasks the agents may not share a common language, so they will not be able to understand the request, or, even if they do, when communication fails it is important to have a fallback mechanism to continue the task. Therefore, in all these cases, the intentions of the agents must be predicted based purely on observations.

This thesis investigates the representational and real-time requirements of using a simulation theoretic approach for the aim of intent recognition. The core of this approach is whether the intent of multiple opposing agents can be inferred by matching plausible plans (or hypotheses) to observations of their behaviour. These plans are generated by re-using the internal models of the observing team to plan as if the models applied to the opposing team, i.e., taking a “what would I do in their position” approach. These plans are, as time progresses, simulated to find the closest match with the observations, i.e., prediction through the generation of multiple competing plans. The internal model that was used to create the best-matching plan is then used as an estimate of the intent of the opponent. To do this, models are needed of the actions an agent can perform, and a realistic simulator is needed to perform the predictions.

1.2 Importance of Predicting Intent

Prediction of intent can be formulated as a problem that consists of three main components:

- Identification of applicable goals for multiple agents.
- Generation of predictions from multiple observations of multiple agents.
- Estimation of the intentions of the opponent from the predictions.

These components are important in many different fields, in particular, the military, games, and robotics domains.

1.2.1 Military

The main objective of military scenarios is to defeat an opponent, therefore it is of immense importance to the military to obtain predictions about the enemy, or to anticipate their own force manoeuvres and positions without using communications, given the high cost of failure. Aside from the obvious advantage of being able to anticipate the enemy giving an increased probability of winning, the use of predictions as a decision aid also allows for:

- more efficient use of resources by gaining a more effective assignment of sensors and weapons to targets;
- reduction in target position errors in environments where GPS is not available;
- higher overall accuracy and therefore less collateral damage.

1.2.2 Games

Commercial computer games have the main objective of providing entertainment for the player—they do not necessarily aim to provide the greatest possible challenge or the most realistic experience. However, being able to predict the intentions of a human player means that a game can be made more enjoyable by:

- adapting the computer-controlled opponent's behaviour to the skills of the human player to reduce the need for fixed 'difficulty levels';
- reducing the reliance on rule-based scripting to orchestrate elements of the game—this allows the game to be more dynamic and 'replayable'.

1.2.3 Robotics

There are many situations where it is beneficial to model and predict the behaviour of other robots in the environment. For example:

- if each robot is operating independently then it is important to avoid collisions by predicting each robot's trajectory;
- in an adversarial setting, for example RoboCup soccer (Kitano et al., 1998), it is a competitive advantage to be able anticipate the opponents' actions in order to neutralise and counter them;

- in environments where the robots have a common goal, identifying the sub-tasks being performed by the other agents is usually necessary to aid co-operation, or to avoid duplicating work.

1.3 Thesis Contributions

The main contribution of this thesis is to show how to take a principled approach to adapting a specific theory about how the human mind performs predictions to be implementable and verifiable in a multi-agent system.

This consists of the following features:

- Identification of the limitations of an architecture for intent prediction based on simulation theory in the single-agent domain.
- Adaptation of the architecture to the multi-agent domain whilst retaining the pertinent features of simulation theory.
- An empirical study that validates the approach that an opposing team's intentions can be inferred by matching plausible plans (or hypotheses) to the observed behaviour of multiple agents, tested through the use of a synthetic environment.
- The expansion of the space of actions that can recognised by first parameteristing the models and subsequently showing how to reduce the compuatational cost of the increased number of hypotheses by restricting the number of inverse model parameters to the most pertinent.
- Limitation of the observability of the system to restrict the resources available to perform predictions. This forces the resources to be focussed on the most pertinent areas.

Furthermore, the simulator and corresponding toolkit for executing hypotheses are made available as open-source software from `http://simon-butler.com/Thesis`.

1.4 Thesis Outline

The rest of the thesis is organised as follows:

Chapter 2 In the background chapter, the inspiration for the approach taken in this thesis—*simulation theory of intent prediction*—is introduced. It is shown how this relates to the general *theory of mind*, and how such a cognitive architecture is applicable to the relevant domains.

Chapter 3 This theory needs to be translated into an architecture that can be implemented for the target application. One such single-agent architecture—*HAMMER*—is described in detail in this technical background chapter. Also, the chosen demonstrative domain for this thesis is described, which includes the key building–blocks: definitions of multi-agent systems (MAS) and details on the simulators available to validate this work.

Chapter 4 The *HAMMER* architecture needs operationalisation for use in a simulated multi-agent environment. Work towards this goal includes the implementation of a realistic synthetic environment, construction of relevant inverse models and interfacing with a generic forward model, and investigation of the effects of the simulation stepsize and resource constraints. Work in this chapter is published in Butler and Demiris (2009).

Chapter 5 One of the largest areas for improvement from the initial architecture is to manage the cost of running many models in parallel, and to manage the complexities of using many inverse models. This chapter deals with investigation and analysis of ways of formulating inverse models to reduce their computational cost. Two approaches are shown: a method of iteratively searching the model's parameter space; and a novel path clustering algorithm that combines parameters when their outputs do not differ significantly. Work in this chapter is published in Butler and Demiris (2010a).

Chapter 6 Realistic scenarios will always mean that resources are restricted. In this chapter the problems of partial observability of the opponents movements are investigated. This includes the development of a resource scheduling algorithm to show that the constraints of partial observability can be exploited to reduce the computational cost, without greatly impacting the quality of the results. Work in this chapter is published in Butler and Demiris (2010b).

Chapter 7 The concluding chapter summarises the research contributions and outlines areas where this work can be taken forward.

Chapter 2

Background

2.1 Introduction

The inspiration for the research presented in this thesis is how the human mind performs predictions. In this chapter, *simulation theory* is introduced as a possible mechanism for this, along with its place within the general field of Theory of Mind.

2.2 Theory of Mind

The simulation theoretic approach to predicting the intention of others is a component of *Theory of Mind*. This is a well-researched area in both developmental psychology and cognitive neuroscience. It is mainly concerned with the human capability of understanding that other people can have different mental state than oneself. In this context, mental state is the possible beliefs, intents, desires, knowledge, etc. that the person might have (Premack and Woodruff, 1978). This thesis will, however, focus on a lower level of theory of mind: the ability to understand the intentions of others by observing their actions.

2.2.1 Intentions and Actions

There are signicant difculties in perceiving intentions from observations of actions. The main one is the problem of inversion: an observed action can be the result of more than one intention. Consider the example of someone intentionally pushing you. The immediate goal of the other agent is to displace you from a location, but the underlying intention is not clear until additional information are added into the equation—is the person that pushed me angry at me? Am I in danger in my previous location? The perception of the current context is crucial to correctly infer the intentions of other agents.

2.2.2 Definitions

The example above highlights the close relation of intentions and action goals, and the difficulty in drawing an exact division line between them. The terms goals and intentions are frequently used interchangeably, but in general goals refer to more immediate desirable end-states (e.g., move to a certain position), whereas frequently intentions have a longer term or higher-level connotation (e.g., follow a target) (Tomasello et al., 2005; Bratman, 1990; Cohen and Levesque, 1990). Tomasello et al. (2005) define intentions as “a plan of action the organism chooses and commits itself to the pursuit of a goal—an intention thus includes both a means (action plan) as well as a goal” (p. 676). In this thesis Tomasello’s denition will be used as the working denition.

Likewise, the terms *prediction* and *recognition* can be ambiguous, however, here we will use recognition to refer to behaviour that has happened in the past, and prediction to refer to behaviour that may happen in the future.

2.2.3 Theories of Theory of Mind

Developmental psychology gives us some ideas as to how we, as humans, perform theory of mind. There are two main competing categories of explanations for this cognitive function: *theory theory* and *simulation theory.*

2.2.4 Theory Theory

The theory theory suggests that the adult mind has an body of knowledge, and this knowledge, in particular that of the physical, biological and psychological world, consists of ‘intuitive’ or ‘common-sense’ theories, (e.g. a set of rules), that relates how people act and react to what they think, feel, and believe.

However, within theory theory, there is much debate as to whether the ‘theory’ is tacit or explicit, whether it is innate (Wellman, 2004) or learnt (Gopnik and Blackwell, 2003; Gopnik et al., 2004), and whether it is modular (Leslie, 1987; Leslie et al., 2004, 2005; Baron-Cohen, 1997), or distributed.

2.2.5 Simulation Theory

On the other hand, simulation theory suggests that people do not use theories, rather, people attribute mental states using their own mental processes and resources as manipulable models of other people's minds, taken off-line and used in simulation with states derived from taking the perspective of another person. In other words, people perform a mental simulation of (i.e. imagine) how we would respond if we were put in the situation of the other person and use that to come to a prediction or explanation (Blakemore and Decety, 2001; Nichols and Stich, 2003; Gordon, 1999; Gallese and Goldman, 1998; Michlmayr, 2002).

2.3 Computational approaches to intention recognition

These two possible mechanisms behind theory of mind are broadly analogous to the two main approaches found in the games, agent and robotics domains for the problem of intention recognition and prediction. The first approach matches the game states to generalised templates or rules, which are collectively known as *descriptive* or *discriminative* models, and this is somewhat related to theory theory. The second is the *generative* approach, which is similar to simulation theory. It uses the current state to generate possible future states and performs matching against the generated state.

2.3.1 Descriptive Models

The descriptive approach takes the currently observed state information and, using various transforms, compares subsets of this transformed state with pre-existing descriptions in another state space. In other words, it uses the extraction of low-level features to match against representations created prior to the start of the scenario.

These pre-existing representations can have associated data that label these representations with the goals, beliefs and intentions that underlie their execution. This approach corresponds to the "action–effects associations" method for intention interpretation in the review by Csibra and Gergely (2007), and to the "Theory of Event Coding" approach put forward by Hommel et al. (2001) based on William James' ideomotor principle in which bidirectional action–effects associations are used to predict the goals of an action.

2.3.2 Generative Models

With the generative approach a set of latent (hidden) variables are introduced that encode the causes that *can produce* the observed data. Using these variables for a recognition and prediction task involves modifying the parameters of the generating process until the generated data can be favourably compared against the observed data.

This approach has been shown by using graphical models, such as Hidden Markov Models (HMMs). HMMs are a specialisation of Dynamic Bayesian Networks (DBN) that are particularly appropriate for time sequenced data. There are many variations of the classical HMM approach that attempt to make these models applicable to action and intent recognition, for example Blaylock and Allen (2006) introduces Cascading HMMs to make the inference of Hierarchical HMMs (HHMMs) (Murphy and Paskin, 2002) and the closely related Abstract HMMs (AHMMs) (Bui et al., 2002) become tractable.

A number of architectures have also been directly inspired by simulation theory by using internal models to perform prediction through simulation. These include the HAMMER architecture (Demiris and Khadhouri, 2006), which has an emphasis on perceiving movement as well as producing it, and MOSAIC (Wolpert et al., 2003), which is aimed towards motor control.

2.4 Multi-agent intent prediction

When the same problem is extended into predicting the intent of multiple agents in a team, the problem becomes much more challenging as it is no longer sufficient to recognise the intent of a single agent, but the joint intent of the agents must be considered.

Much of the work can be classified to take the *descriptive* approach. For example, Tambe (1996) constructs explicit team models and tracks them at the team level; as a result the recognition of a large number of individual agent plans is avoided. A similar approach is taken by Soon et al. (2004) to create *Rolegraphs* for each intention and then use graph matching to find the intention of a team, without full knowledge of plans, or complete observations.

Work has also been done to analyse spatio-temporal traces for coordinated motion, whether moving apart or together (Devaney and Ram, 1998), or to simultaneously assign team roles and recognise their actions from similar traces (Sukthankar and Sycara, 2006) by generating team assignments from template matching then filtering out teams that do not follow the parametric model. Similarly, in the domain of a football (soccer) match, player positions are used to infer player roles and explicit rules are extracted from training data (Beetz and Kirchlechner, 2005). Other approaches combine plan recognition, with probabilistic inference. For example temporal logic, high level descriptors and Bayesian networks are used to classify different plays in American football (Intille and Bobick, 1999).

There is relatively little work that can be said to take a purely *generative* approach in the multi-agent domain. Hidden Markov models have been found to be brittle when using real-world data, but different extensions have been tried. For example, the approach of using AHMMs by Bui et al. (2002) has been extended to the multi-agent domain by Saria and Mahadevan (2004), which uses hierarchical decomposition to get to the single agent level where the independence condition of HMMs can be utilised.

Similarly, because HMMs struggle with multiple agents due to the loss of spatial relational data, the CHMMs of Blaylock and Allen (2006) have been extended by clustering and extracting specific team features to get a discretised space that includes semantic information. Probabilistic methods are then used to estimate the policy (White et al., 2009).

Work to make HMMs more robust uses a hybrid approach where a descriptive model is first used to recognise the agent roles, then this is fed into the generative model (Luotsinen and Bölöni, 2008). This role assignment is important because, due to the Markovian assumption, the same order may not be retained on each observation. Similar work by Liu and Chua (2006) uses observed decomposed HMMs (ODHMMs), which separates observations into individual agents following an independence assumption and subsequently assigns them roles. Alternative work by Avrahami-Zilberbrand and Kaminka (2006) combines a symbolic plan recogniser with HHMMs to achieve a similar result.

Work by Vail et al. (2007) has found that the Conditional Random Fields (CRFs) perform better than HMMs in their experiments. These CRFs use the whole set of observations which takes longer but removes the Markovian restriction. They go on to show how CRFs can be converted to HMMs, hence they can be considered to be a discriminative-generative pair.

A similar hybrid approach by Heinze et al. (1999) uses a machine learning algorithm for trajectory matching called CLARET, that incorporates attribute generalisation, graph matching and a Bayesian network to match a relational structure (described in more detail in Pearce et al. (1998)). The matching plan is then applied to a BDI agent to generate a predicted intent.

Chapter 3

Technical Background

3.1 Simulation Theory for Intent Prediction

Of the two theories proposed by psychology (see Section 2.2), it is simulation theory that has a large body of support from neuroscience (Hesslow, 2002). It is this biological-plausibility that has inspired cognitive architectures based on simulation theory, mainly in the robotics domain.

Simulation theory gains credence through research in several fields suggesting that 'thinking' consists of simulated interaction with the environment. The evidence outlined in Hesslow (2002) suggests: that when we think of an action we use the motor structures of the brain but actuation is deactivated; that perceptual activity can be internally generated within the brain; and that these two areas of the brain can be connected internally. Essentially, simulated behaviour causes perceptual activity that is similar to actually performing the behaviour.

It follows that if we try to recognise behaviour of another person, we assume that we are in their position and use this internal simulation mechanism to process possible actions and predict their response (Johnson and Demiris, 2005). Not only does this theory explain how we infer how others will respond to certain situations, it also attempts to explain the experience of an 'inner-world' within our brains where perceptions are not externally triggered. Also, this use of brain structures for both action and recognition is an attractive proposition from an engineering perspective as it allows subsystems to be reused for different purposes.

This theory maps conceptually quite well to a cognitive architecture for intent predic-

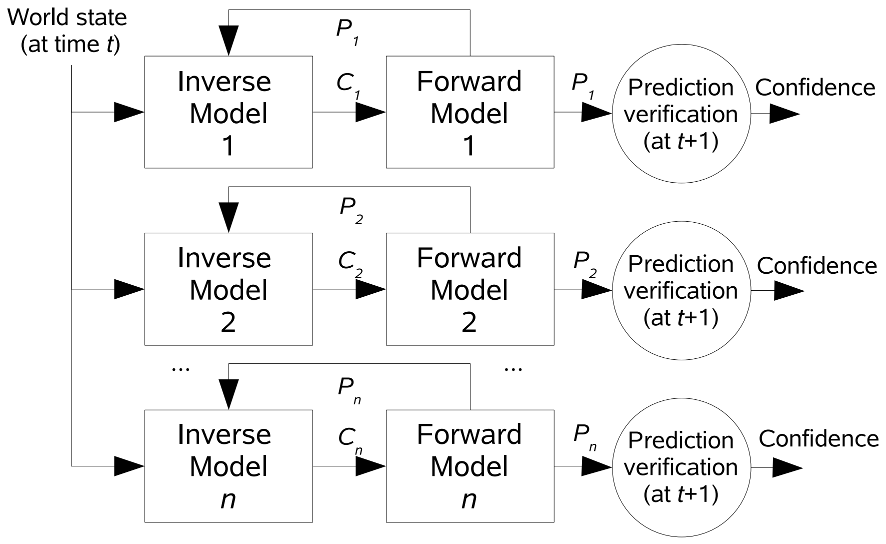


Figure 3.1: **The HAMMER architecture implements the principles of the simulationist approach.** Multiple inverse models receive the world state and suggest possible commands (C_1-C_n), which are formed into predictions of the next world state by the corresponding forward model (P_1-P_n). These predictions are verified on the next time step, resulting in a set of confidence values.

tion: another agent's intentions can be inferred by creating plausible plans (or hypotheses) that are consistent with a range of possible goals. These plans are, as time progresses, simulated to find the closest match with the observed behaviour. Subsequently, the objective that was used to create the best-matching plan is used as an estimate of the intent of the agent. This can be thought of as recognition through the generation of multiple competing plans (Demiris, 2007).

3.1.1 HAMMER

Starting at the lowest level, taking a simulationist approach to recognition and prediction of plans requires a method to generate and evaluate certain primitive actions that can be performed by single agents. The HAMMER (Hierarchical Attentive Multiple Models for Execution and Recognition) architecture provides a starting point for the system (see Figure 3.1). It is comprised of three main components: the **inverse models** (plan generators), the **forward models** (predictors) and the **evaluator** (Demiris and Khadhouri, 2006; Demiris, 2007).

An inverse model (IM) takes the current world state, and, optionally, target goal(s) or other parameters. It outputs the required waypoints or other control signals (a *plan*) that, according to the model, are necessary for the agent to perform so that the implicit or explicit target goal(s) are achieved. Each parallel instance of an inverse model is paired with an instance of the forward model (FM) that provides an estimate of the events that will occur if the generated plan is followed. At each time step this estimate is returned to the inverse model to tune any parameters of the actions to achieve the desired goal(s).

To determine which of these inverse/forward-model pairs most accurately describes the events that are occurring, periodically the output of each forward model is compared with the actual world state. These comparisons result in confidence values that behave as an indicator of how closely the observed events match each particular prediction, and they are subsequently accumulated over time until such a point that one model pair achieves a clear separation from the others.

3.2 Scenario Domain

This thesis presents the first application of the simulationist approach to predicting the intent of multiple agents. It builds upon the successful implementation of the simulationist approach to single agent intent prediction in the robotics domain (Demiris and Khadhouri, 2006), and this thesis investigates how to apply similar principles to predicting agents in an multi-agent adversarial environment.

3.2.1 Multiagent Systems

There are many definitions of an ‘agent’, but a generic definition is as follows:

> An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future. (Franklin and Graesser, 1997)

Using this definition, an agent is a self-contained entity that senses and interacts with an environment with its own goals. A multi-agent system is therefore a system consisting of many agents interacting with each other and the environment. These agents must be reactive, autonomous, goal-oriented and temporally continuous.

3.2.2 Simulator

Given that the intent of these agents needs to be predicted, a platform is needed that allows the observations of agent movements.

Unfortunately, the military is reluctant to give out data, and staging wargames is outside the scope of this thesis. Likewise, games companies jealously guard their data and latest game engines, and access to a large team of robots that are capable of reliably sensing the environment and each other was unavailable at the start of this work. Therefore, the scenario domain for this thesis is limited to a simulated environment, and data that is self-generated.

There are several simulators that could provide the required platform, and these are outlined below.

Games

There are several open-source game engines that enable simulation of agents within a specific environment. In the real-time strategy game genre, the Spring engine (Johansson, 2010) is a powerful free engine, with a full 3D engine and physics simulation engine and support for a scripting interface. Unfortunately this engine only recently became stable and feature complete, so was unavailable for serious use at the start of this project. The other freely available RTS game engines are Stratagus, and the Broodwar API for Starcraft, however, these are simple 2D grid-world engines that are not very realistic.

The Unreal engine is modern 3D engine, mainly used for first-person shooter games. It is has an API that allows access to parts of the game state, however, it may take up to 100ms to obtain a state, which is too slow to autonomously control agents.

Military Synthetic Environments

In the military domain there are synthetic environments that realistically simulate military engagements. Virtual Battlespace 2 (VBS2) is the only one available to academic institutions, however it has a rather limited scripting API.

Robotic Simulators

In the robotics domain, the Player/Stage project provides the Gazebo simulation environment. It is a 3D environment that simulates robots with a high fidelity. However, it is only designed to simulate a small number of robots, so is unsuitable for simulating teams containing many agents.

3D Engines

The alternative to the engines listed above is to build a 3D application from scratch. Programs can be constructed at the polygon level, interfacing directly with a 3D API such as OpenGL or DirectX, or it can make use of a higher level 3D engine which provides abstractions to the 3D API being used. Given the complexity of 3D environments required by the current generation of simulators, interfacing directly with a 3D API is impractical. Therefore a 3D engine is required to manage the system.

There are three popular engines, all written in C++: Delta3D, Irrlicht and Ogre3D.

Delta3D is an integration of many other open source projects, such as OpenSceneGraph (OSG) for rendering the scene, based on OpenGL, Open Dynamics Engine (ODE) for simulating physics, OpenAL for sound support, CEGUI for graphical user interface widgets, etc.

Irrlicht is a fast 3D engine based on OpenGL or DirectX. It contains many state-of-the-art features, such as pixel shading and particle effects.

Ogre3D is a well designed engine, again supporting OpenGL or DirectX, and contains many of the same features as Irrlicht.

3.2.3 Environmental Requirements

To create the most generic platform that covers the common elements of the military, games and robot domains the environment must be outdoor, large-scale and continuous (i.e., not a grid world), containing two teams of agents. Each agent within the environment must be free to move around, albeit constrained by realistic physics, and when given a command will always execute it.

The reason for this environment is that it can easily be modified to apply to:

- simulated military scenarios, with the inclusion of specialist military knowledge for the operation of the agents and the actions they can perform;
- real-time strategy games, given the player has a method of collecting resources and building new units;

- multi-robot scenarios, if models of the robots' actions can be produced.

Agents

The agents have the basic goal of defeating the opposing team and are capable of moving, and firing at the opponents agents. Throughout, the system is trying to predict the intentions of the opponent based on observing the movements of the agents. These predictions can be applied as an aid in choosing a counter-strategy if the observing team is computer-controlled, otherwise they can be presented graphically.

Using the definition of an agent above, the actual entities that would be described as an agent are straightforward in the games and multi-robot domains—each agent is a controllable unit that can be of many different types. In the military domain an agent would be a human soldier, however, we shall also consider that vehicles containing multiple soldiers can be considered to be a single agent.

It could be argued that the individual units are only sub-agents because they are just carrying out the orders of a commander. However, the units do fulfill the requirements of being reactive, autonomous, goal-oriented and temporally continuous: they react to the surrounding units to perceive enemies and to stay in a formation; they autonomously decide to engage the enemy; they will achieve an assigned goal; and they continue until they are destroyed.

Assumptions

Due to the scale of the problem, some simplifying assumptions need to be made. The first is that there is a complete and detailed map available of the area within which the scenario takes place. Secondly, initially, the world is fully observable—the position of all the units within this area are known without uncertainty. Thirdly, there are no communication delays between units, i.e., the units report their position and receive orders instantaneously.

However, whilst the availability of a detailed map is a crucial and a realistic assumption, full-observability is not possible in the robotics and military domains with current technology. Therefore this assumption is relaxed in Chapter 6. The instantaneous communication assumption is also not realistic in real-world conditions, however, current communications technologies are near-instantaneous and therefore it is anticipated that relaxing this condition is not worth pursuing.

Chapter 4

Operationalisation

4.1 Introduction

To investigate the simulationist approach to intent prediction in multi-agent environments, the theory first has to be operationalised into a system that can be implemented on current hardware. Firstly an abstract architecture that strictly adheres to the principles of the theory has to be designed. Fortunately the HAMMER architecture described in the previous chapter does this by introducing the concept of parallel instantiations of many inverse and forward model pairs. This concept forms the basis for the system presented in this thesis. Subsequently, operationalisation requires this architecture to be adapted and extended to account for the challenges inherent in the multi-agent domain. It is not simply a case of implementing HAMMER in a new domain because there are several key areas that are different or under-specified, and new approaches need to be considered to retain the principles of simulation theory.

This chapter identifies the key limitations of the HAMMER architecture that need to be overcome when operating in the multi-agent domain as described in Section 3.2. First, the architecture is adapted to the target scenario and details of the required inverse and forward models are presented. Subsequently, the implementation details of the realistic synthetic environment developed for this thesis are described. Finally, experiments are performed to test the limits of the system and identify areas for improvement.

The work in this chapter was published in Butler and Demiris (2009).

4.2 HAMMER in the Multi-Agent Domain

The HAMMER architecture was conceived to operate in the single agent domain, with the system identifying the actions of a single demonstrator within well defined trials, i.e., each trial consists of a demonstrator performing a single high-level action. When considering utilising this approach within the multi-agent domain a number of issues become apparent:

Actions involve multiple agents It may take more than one agent to perform a task, hence models need to be applied to multiple agents.

Multiple concurrent actions There can be many agents all performing different actions and the system must be able to recognise each of the actions.

Wider space of possible actions As there may be many heterogeneous agents the set of all possible actions increases with each agent type, and, more importantly, actions may involve more than one agent, which exponentially increases the number of actions.

Variable timestep The original architecture implies the duration of a prediction (a *timestep*) is fixed according to the agent involved. However, with multiple heterogeneous agents this reliance on an implicit timestep needs to be abstracted away.

Additionally, not because of the move to multiple agents but due to the chosen scenario, several other factors have to be considered when adapting the HAMMER architecture:

Changing agent actions HAMMER assumes that there is a single result for a trial. This is not necessarily the case in this scenario because the actions that each agent performs may change throughout the trial due to the agents updating their goals.

Inverse and forward models The internal models must be created to correspond to the range of actions anticipated for the scenario.

4.2.1 Adapting the architecture

To help incorporate solutions to these issues into an implementable system, the term *hypothesis* will be introduced to encapsulate the basic building block of HAMMER—the inverse and forward model pair (as previously described in Section 3.1.1 and shown in Figure 3.1).

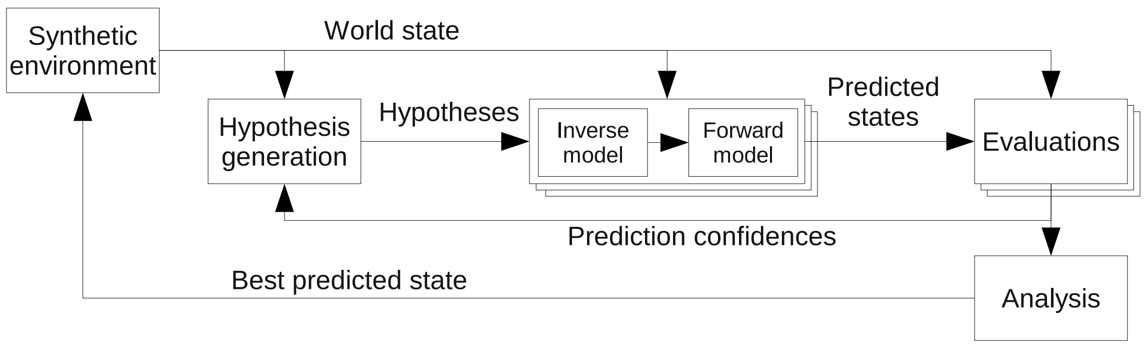


Figure 4.1: **The simulationist architecture implemented into a multi-agent framework (assuming full observability of the world state)**. The world state is used to generate plausible hypotheses, and these hypotheses are simulated in parallel to get a set of future predicted states. Each prediction is evaluated (compared against the observed state) to gain a confidence level that the hypothesis is currently being executed. This information is sent back to the hypotheses generation state to be used to tune which hypotheses to execute. The best performing hypotheses are then returned to the user to be used as a predicted state.

An hypothesis encapsulates the following information:

Controlled agents — these are the agents that the inverse model will control.

Inverse Model — the model to use that performs the desired action.

Forward Model — the corresponding forward model to use.

Duration — the amount of time the internal models will run before returning a predicted state.

Time scale — the speed up over real-time that controls how fast the forward model runs.

Using this approach, the HAMMER architecture can be wrapped with modules that perform hypothesis generation and evaluation, as shown in Figure 4.1. Here a scenario is performed using two teams—a team utilising this system and an opposing team—in the *synthetic environment* block. The world state and confidence of the currently executing hypotheses are used to generate new hypotheses (the *hypothesis generation* block) for the opponent's units. Each hypothesis is assigned a specific inverse model, a set of units to apply it to, and a duration t_p. These units are then simulated by the forward model with control signals generated by the inverse model (the *inverse-forward model pairs* block). The predicted positions of the units and the actual positions are compared (the *evaluations* block) after the specified duration—the result of which is used to calculate confidence

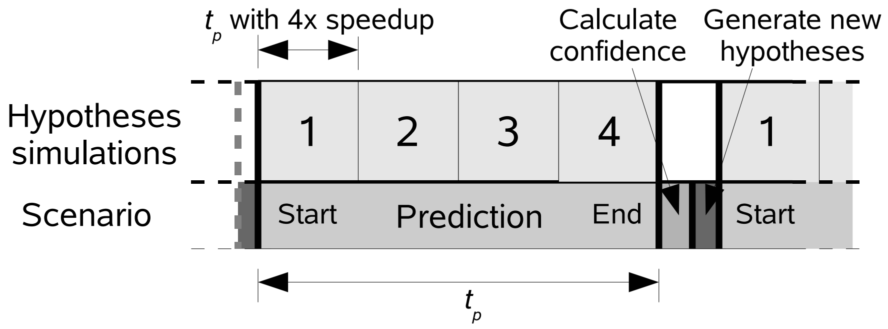


Figure 4.2: **The timing of generating a prediction.** First the hypotheses are generated then the world state is recorded and sent to the internal simulations to execute the hypotheses in faster than real-time. When the prediction duration t_p has elapsed the world state is recorded again and evaluated against the predictions to generate the confidence. Subsequently the loop is repeated.

that those units are achieving the goal. The best performing hypotheses are analysed (the *analysis* block) and a future predicted state is sent presented to the user.

Intent

The best performing hypotheses are also used to obtain the 'intentions' of the agents. Referring back to the definition of an intention in Section 2.2.2, an 'action plan' is defined in the chosen scenario domain as the series of waypoints that an agent has to follow. These waypoints are generated by the inverse models, and an intention is defined to be the overall goal of the inverse model. Hence, the predicted intent of the agents is considered to correspond to the goal of the inverse models that generated the predictions that best match the observed actions.

Timing

The timing of performing a prediction is shown in Figure 4.2, and illustrates how many hypotheses can be evaluated in faster than real-time within the same prediction duration t_p. Assuming the speedup s parameter is 4, then if there are less than 4 hypotheses, the confidence will still be calculated after t_p has elapsed. If there are more than 4 hypotheses then the confidence will be calculated after all the hypotheses have finished executing, which would be $t_p + n\frac{t_p}{s}$, where n is the extra number of hypotheses.

An overview of main blocks in Figure 4.1 will be discussed in the following sections, and the implementation details discussed in Section 4.3.

4.2.2 Synthetic Environment

The synthetic environment is responsible for hosting the agents that perform the scenario thereby simulating the 'real-world' interaction both between the agents themselves and between the agents and the environment. It provides positional and other properties about each of the agents to act as the input for the predictive system.

A synthetic environment is utilised to collect data for the scenario used in this work, however, this is not a requirement of this approach—for example, data could be obtained from teams of robots, or from sensors attached to humans or vehicles.

In this work, teams of agents populate the environment and can be controlled by human operators. Typically there would be two teams of agents and, correspondingly, two human commanders, each controlling one of the teams. Additionally, only one of the commanders utilises the predictive system, and the other acts as the opponent—from the perspective of the predictive system. Human users provide the input to control the teams of agents through a typical game-like interface.

Alternatively, the system can operate without human operators, for instance, the opposing team can automatically be controlled by inverse models to achieve certain goals and the observing team can either be static or the agents can also be controlled by the inverse models chosen by a higher-level autonomous decision making system, based on the predictions. This is useful for objectively testing parts of the system without needing subjective inputs from human controllers.

4.2.3 Multi-agent Internal Models

The inverse and forward models used in this architecture are application-dependent. The inverse models will depend on the type of plans and actions each of the units available in the scenario, and the forward model will depend on the dynamics of the synthetic environment.

There are approaches that try to learn the appropriate models, for example, in the robotics domain, the inverse models can be learnt by using motor babbling (Dearden and Demiris, 2005), and the forward models are defined by the physical characteristics of the robot.

Alternatively, in the games domain, it is very likely that the inverse models already exist as a library of actions a computer-controlled player can perform, and the forward model is defined by the way game units behave, which is encoded by the game engine itself.

Similarly, in the military domain, there is a well defined set of rules of engagement that cover the actions that the agent's can perform, which can be hand–coded into inverse models. Also, like robotics, the forward model involves the real-life physics of moving the different unit types through the environment.

In this work, the aim is to use a custom synthetic environment, and the focus is not how to automatically generate the internal and forward models, but how to best use them in a multi-agent environment. Therefore it is justified to hand–code the inverse models, and the best approach for the forward model is to use separate instances of the synthetic environment engine.

Synthetic Environment as a Forward Model

The main reason for using the full synthetics environment engine to simulate the outcome of the inverse models is that it is easier than developing plausible forward models for each agent type. However, it also has the added benefit that it has a greater prediction accuracy, at the cost of being more computationally expensive.

The reason for the greater accuracy is that the inverse models only output an ideal next state, and that may take several seconds to be achieved, whereas the physics rules and reactive behaviours in the engine actually determine the observed movement of the units. For example, the inverse model will output a series of waypoints to route a unit to a specific goal position. However, when run in the 'real-world', the actual trajectory that the unit will take will be dependent on the interpolation used between the waypoints, the type and gradient of the terrain, and any use of local obstacle avoidance; or there may be some dynamic team behaviour, such as the unit may have to maintain a position in a formation.

The easiest way to encode all this behaviour is to use an engine that already exists as the environment hosting the 'real-world' scenario. However, this is only justified as being applicable to non-simulated domains, such as robotics, if the engine is non-deterministic, i.e., it includes a noise component. This means that the outcome of the forward model should never be exactly the same as that produced in the 'real-world', even if exactly the same inverse model is used to control the agents in both cases.

To use the engine as a forward model, it has to support a few features not usually exposed with an external API. It needs to be able to quickly read the complete current state of the environment, transfer and instantiate that state in a new instance of the engine. This involves extracting the pertinent position, orientation and motion properties of each unit, along with dynamic and static attributes such as health and firing range. This information then needs to be serialised and sent to an independent instance of the engine, where the state information is initialised and executed for the specified duration, after which the resultant unit positions are returned to be evaluated.

4.2.4 Hypothesis generation

Once the information about each of the agents is obtained from the synthetic environment, it is used to generate hypotheses about the possible actions each of the agents are able to perform in the current conditions. Each action is represented in this system as an inverse model, hence this means the applicable inverse models for each agent are identified, based on, for example, the agent's location and capabilities, and incorporated into an hypothesis.

This is similar to the basic HAMMER approach, where all of the inverse models are evaluated on each timestep. However, for multiple agents, there exists inverse models that only apply to more than one agent, and others can be combined so that several inverse models apply to the same agent at once. This means that the applicability of each inverse model to an agent is defined on a per–model basis, and models are grouped into categories so that models from different categories can apply to the same agent.

Another addition to the HAMMER architecture is that, due to the constantly changing agent goals, it is important to only evaluate the hypotheses that are relevant. This can be achieved because after the initial run of the hypothesis generation stage, there is past performance information for each hypothesis available, i.e. the confidences calculated by the evaluator. This is used, for example, for deciding whether to stop or pause badly performing hypotheses or whether to split groups if the confidences of the agents vary considerably within a group.

Forward model parameters

Once an hypothesis has an assigned action, it also needs parameters to pass to the synthetic environment acting as a forward model. The main parameter is the simulation *duration.*

The HAMMER architecture uses a fixed single timestep for the forward model simulation, ostensibly derived from the underlying perception update loop of the agents being observed and/or controlled, however it is not explicitly defined in the architecture. In contrast, when working with multiple heterogeneous agents, discrete timesteps need to be abstracted away. Instead, variable-sized prediction blocks that are not coupled to the timesteps of the agents need to be used. This also accounts for the possibility of changing agent actions requiring different prediction durations as the scenario progresses.

The length of a prediction has to be carefully chosen to balance the latency of an action change versus the accuracy of a prediction and the overhead of calculating confidence. For example, very short predictions means that the system is very responsive to generating new predictions when the observed action changes, however, they have not been simulated for very long so the predictions may be noisy.

Choosing the prediction length is therefore a function of the dynamics of the internal simulation and the importance of recognising an action change as quickly as possible, so this time length can be specified for each hypothesis individually.

Additionally, the forward model takes a *speedup* parameter that defines how much faster than real-time to perform the simulation. There is the possibility to trade-off simulation accuracy against throughput, so this parameter can also be specified for each hypothesis.

4.2.5 Evaluation

When the same duration that was used to generate the predictions has elapsed in the synthetic environment playing out the scenario (which is running in real-time), the current unit positions are obtained and are compared against the predicted unit positions. These comparisons result in a confidence that each of the opponent's agents are following the assigned inverse model. Therefore, for the lifetime of an hypothesis, there are confidences recorded at each of the finishing times of the prediction intervals. Additionally, as inverse models can apply to groups of agents, the confidence is averaged over all the agents in a group to get a single confidence for the group.

Also, HAMMER specifies that the confidence of an hypothesis should be accumulated to see which one performs the best over a trial where a single task is performed. However, accumulation would not perform well in this case because the start and end points of

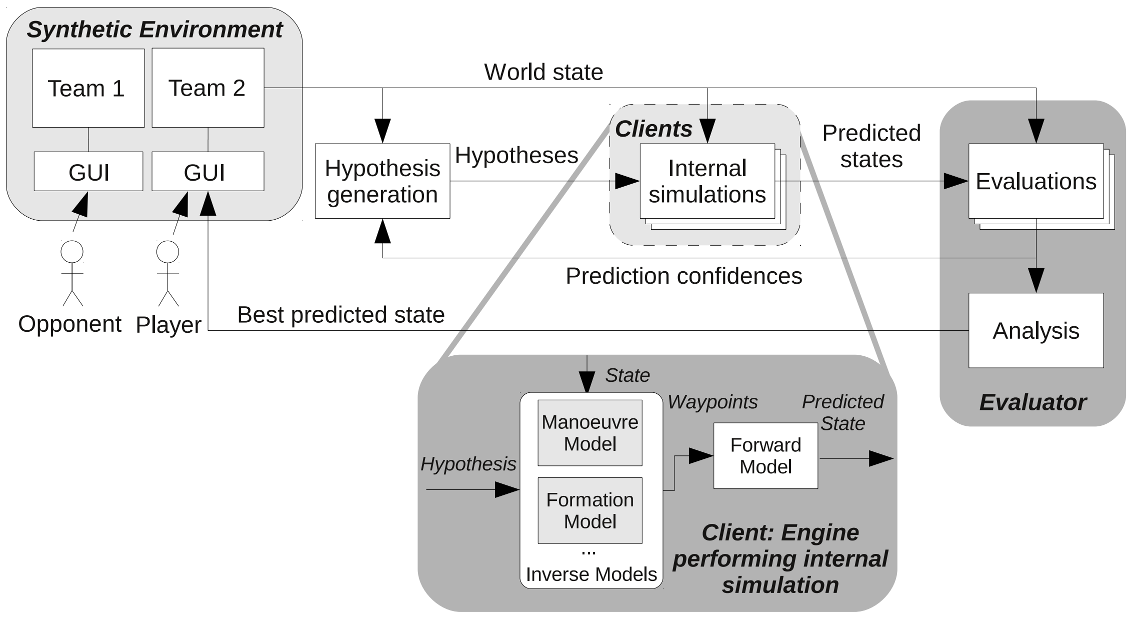


Figure 4.3: **System implementation.** Expanding from Fig. 4.1, this shows the human-controlled synthetic environment instance at the top-left of the diagram, where the two commanders control their respective team through a GUI. The internal simulations are performed on separate clients that host the inverse and forward models. The resulting best performing state is then returned to be displayed as predicted unit traces on the GUI.

a 'trial' are not well defined—the player's intentions could change at any moment. For example, if an hypothesis performs well at the start of a scenario it would accumulate a large confidence, and it would take a long time after the best matching hypothesis changes for a new one to overtake it. This is overcome by not accumulating, but averaging the available confidences over a time window. The method of assuming the hypothesis with the highest averaged confidence wins still applies, but the effects of an hypothesis earlier in the scenario are negated, whilst also smoothing the effects of a noisy prediction.

It should be noted that the confidence returned by the current prediction before the averaging can be used immediately for short, quick responses, for example the AI might respond to a predicted attack and it doesn't matter if the highest confidence hypothesis fluctuates or is noisy. Whereas this time-windowed averaging of the confidence creates a more stable winning hypothesis to provide a overall confidence for a particular agent or group of agents. This can be acted on by humans as they have slower reactions, or a high-level intention analysis.

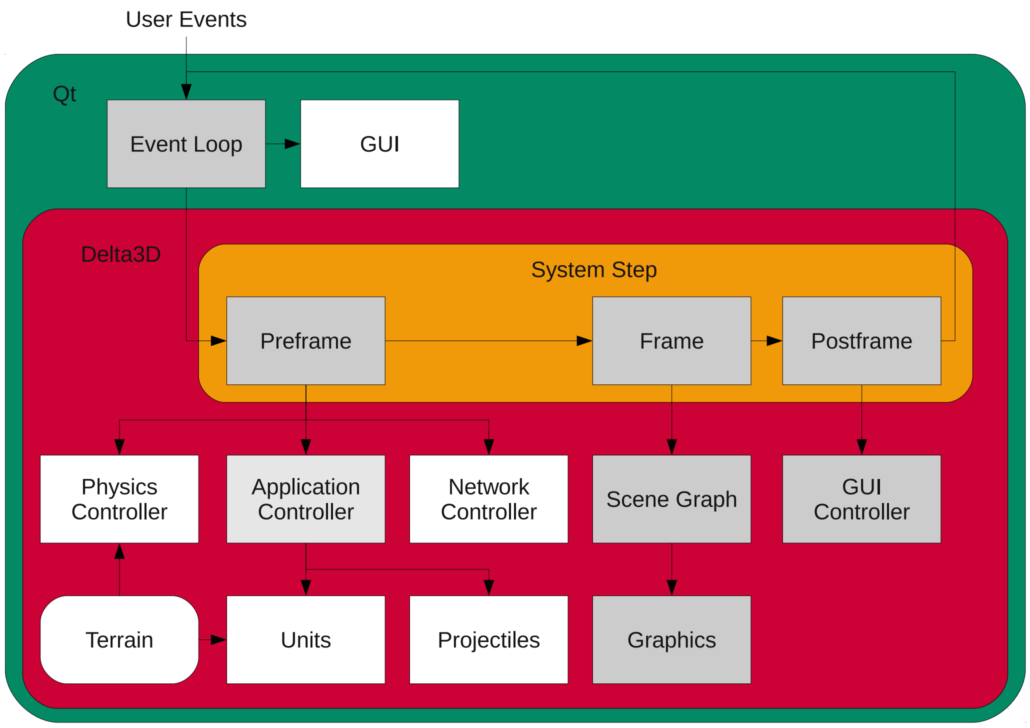


Figure 4.4: **The system event loop.** The main event loop of the simulator and the components that are updated on each timestep.

4.3 System Implementation

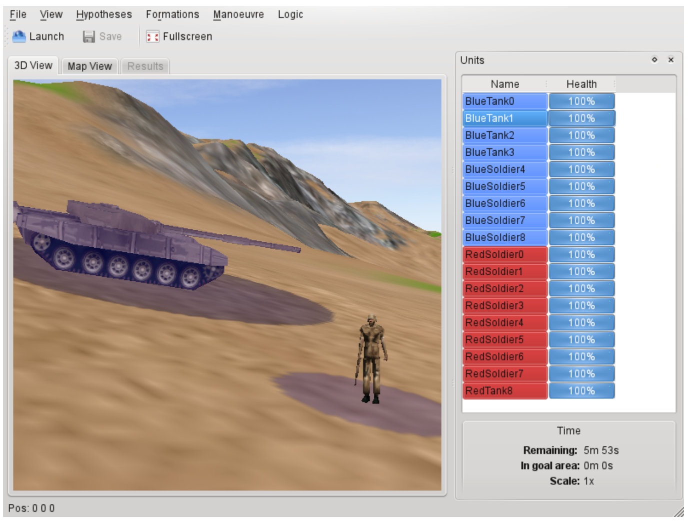
The implementation of the system is divided into four main sections (as shown in Figure 4.3): the 'real-world' synthetic environment that host each team of agents (user-controllable units); the generation of hypotheses; the clients running the game engine in 'hypothesis' mode, executing the requested internal models; and the evaluator that generates confidence values and analyses the predictions to presents the results to the user.

In this system, the set of units U is divided into opposing units U^o and units from the observing team acting as targets for the opposing units U^t, so $U = U^o \cup U^t$.

4.3.1 Synthetic Environment

The engine of the game is based on Delta3D (Darken et al., 2005), which is an open-source project to integrate various software libraries, such as Open Scene Graph (OSG), Open Dynamics Engine (ODE), Character Animation Library 3D (Cal3D), Game Networking Engine (GNE), etc., into a coherent platform for simulation and games.

The synthetic environment is comprised of six main components: the units, the projectiles, the terrain, the physics, the networking interface and the user interface.



Name	Health
BlueTank0	100%
BlueTank1	100%
BlueTank2	100%
BlueTank3	100%
BlueSoldier4	100%
BlueSoldier5	100%
BlueSoldier6	100%
BlueSoldier7	100%
BlueSoldier8	100%
RedSoldier0	100%
RedSoldier1	100%
RedSoldier2	100%
RedSoldier3	100%
RedSoldier4	100%
RedSoldier5	100%
RedSoldier6	100%
RedSoldier7	100%
RedTank8	100%

Time

Remaining: 5m 53s
In goal area: 0m 0s
Scale: 1x

Figure 4.5: **Simulator screenshot.** This shows the interface for the human-controlled teams. The main 3D view is used to issue commands to the units. The status of each unit is shown in a panel on the right of the screen. The tabs allow the user to access a 2D map of the unit positions and their predicted movements.

The event loop is controlled by the GUI framework (Qt), the time step is controlled by the simulator framework (Delta3D), the graphics are rendered by Open Scene Graph (OSG), and the physics are simulated by the Open Dynamics Engine (ODE), as shown in 4.4.

The 3D engine (OSG) was used to model a large outdoor terrain (see the screenshot in Figure 4.5), with the height and other features (texture, trees, buildings, etc) of the terrain being displayed based on 2D feature maps. The physics engine (ODE) was used to accurately model the movement of the various agents and vehicles, with additional control of their aiming and firing mechanisms.

When using the synthetic environment, the human commanders are responsible for choosing goal positions for their units, so they are free to perform manoeuvres and formations as they see fit.

Terrain

The terrain is rendered from a 16-bit greyscale image heightmap. In this thesis L3DT[1] was used to generate a heightmap and texture map. Transparent PNG images are used to control the features that appear on the terrain. These include the position of roads, forests, cities and defensive positions. An XML file (see Listing E.3 for a sample) controls

[1]http://www.bundysoft.com/L3DT/

how the features are generated from the image maps. For example it controls the object(s) used to represent the feature (e.g. a tree mesh), the mean and sigma of how the objects are distributed across the specified areas, the shell/bullet yields, and whether the feature affects the range of a unit's sensors.

Physics

The physics engine (Open Dynamics Engine) is integrated into Delta3D so that the geometry used to represent the units in the 3D scene is synchronised with the bodies that are moved according to their physical properties on every physics step update.

Each body is a separate entity and by default is only subject to the force of gravity. Bodies can be connected using joints to simulate more complex objects. When bodies that are part of different entities get close, collision joints are created and parameters such as friction control the result of the collision. The number and placement of collision joints that are created within the contact area on each step are random. This creates a stochastic element to each collision, and, for example, affects the movement of vehicles across the terrain. This, ultimately, is the main reason for the differences between the predictions and the 'real-world' simulations.

To avoid numerical instability in the physics engine, the step size of each physics step update should be fixed at t_π. However, the the system does not have a fixed frame rate, so the number of physics steps needs to be calculated one each frame. On each call to `System::PreFrame` the time delta since the last call to `PreFrame` is recorded as Δt. This is multiplied by the speedup parameter s to get $s\Delta t$ (if the simulation is not running in real-time), and passed to the physics controller. The physics step is run $\lfloor \frac{s\Delta t}{t_\pi} \rfloor$ times and any remaining time is left until the next call to `PreFrame`.

If there is a large time delay between frames then calling the physics step many times without getting to the control stage can cause instability within the physics simulation. To avoid these effects there is a maximum physics step $\Delta t_{\max}$ and if $s\Delta t$ is greater than $\Delta t_{\max}$ then it is capped to $\Delta t_{\max}$.

GUI

The graphical user interface (GUI) for the simulator uses the Qt framework and embeds the 3D engine within an OpenGL widget. The main event loop is controlled by the Qt application class, so to render the 3D scene the update method is called on the QGLWidget which calls the updateGL method from the event loop. This forwards the request to Delta3D to perform a timestep. Subsequently, the update method is called again before leaving the updateGL method, so the every time around the event loop the next timestep is performed.

From the QGLWidget viewport the user is able view the 3D scene and can select a unit by clicking on it, or to select multiple units by clicking and dragging a box over the required units. This was implemented by determining the points on the terrain that were clicked on, through the use of a projected line starting at the 3D camera centre and the 2D coordinates of the point on the projected viewport, and seeing where this line intersected the terrain. All of the units that fell within the area on the terrain were selected.

To aid in the identification of units and their goals (particularly when the camera is zoomed out) there are graphical elements composited onto the terrain texture at the required positions. For example, a circle is painted on the terrain texture around each unit, and lines over the terrain indicate the units path to their goal position. This was achieved by rendering 3D objects to a special OSG `OverlayNode`, the result of which was then projected onto the terrain texture.

The GUI also has a 2D view of the simulation where a top-down view of the positions of the units and their groupings is shown. This view also shows the results of the predictions by displaying traces of the future movements of each unit.

Networking

To create a realistic scenario, each commander needs to control their own team of units separately from the others. This means that the synthetic environment needs to be run on multiple computers, one for each team, and the resulting state of each simulation transferred over the network to the other client.

One computer acts as the server, which also hosts the predictive system, and the other instances connect to it as clients. Each instance—whether server or client—simulates its own units. Positional events, and other events such as projectile detonations, are sent on each frame of the simulation over the network to the opponent, and representations of those events are shown on the opponent's instance. For example, the position of a 3D mesh of an opponent (using the `UnitRemote` class) is updated when a position packet is

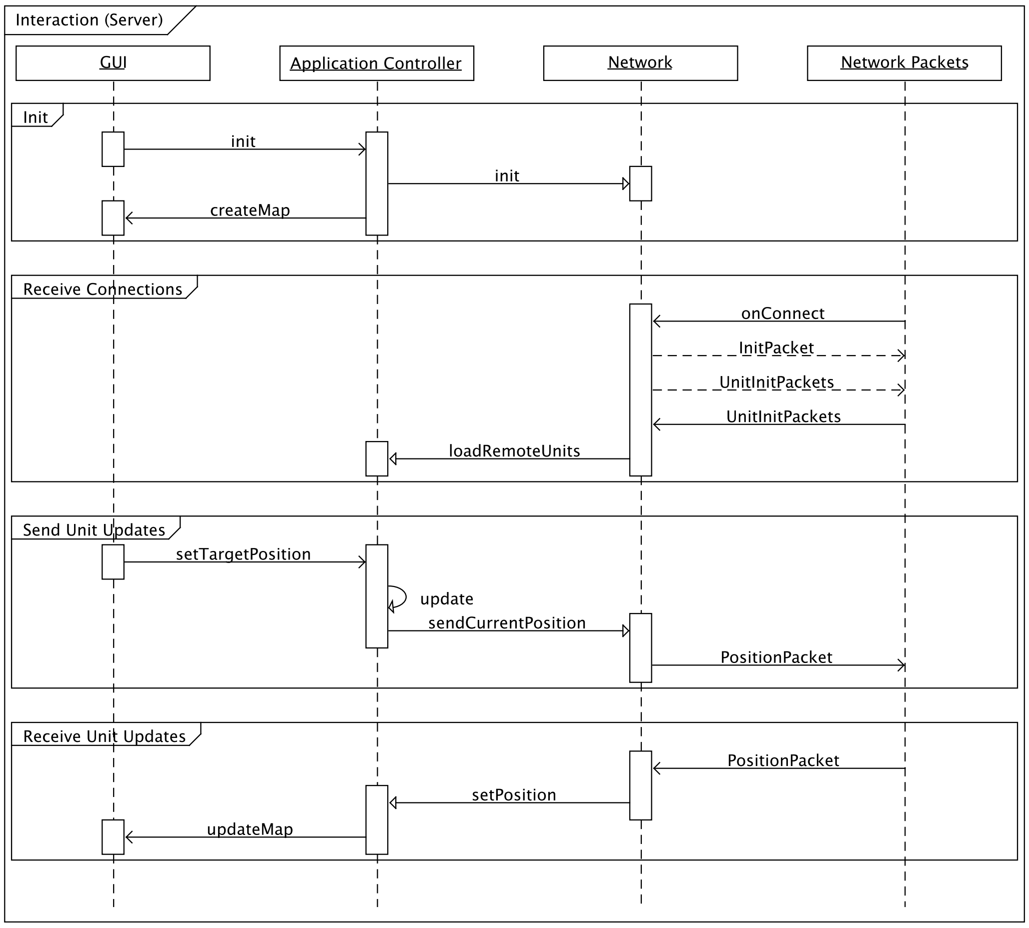


Figure 4.6: **Sequence diagram for starting the server.** First, the start of the GUI triggers the initialisation of the application controller, which in turn initialises the networking to start listening on the server port. Subsequently clients connect and are immediately sent the initialisation parameters and the units that the server hosts. Then the client sends its own units and these are loaded as remote units on the server. When the units move in the application update function, the new unit positions are sent over the network to the client. Similarly, when a unit position packet is received the position of the unit is updated.

received from the opponent's client as a result of their simulation.

A number of messages are sent when initialising and sending or receiving positions. These are passed between the GUI, the application controller and the network, and are shown for the server and client in Figures 4.6 and 4.7 respectively. All of the messages passed between the GUI and the application are asynchronous events passed using Qt's signal and slot mechanism. Messages between the application and the network layer are synchronous, but are in different threads, so the network resources are protected by a mutex.

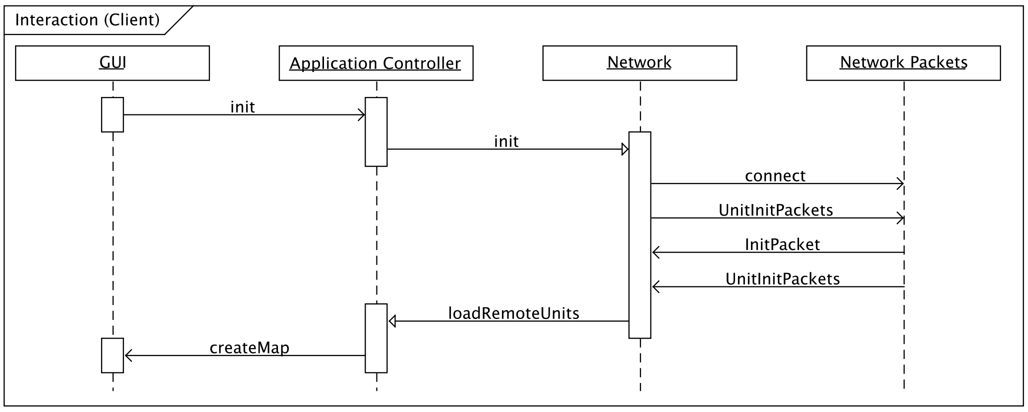


Figure 4.7: **Sequence diagram for starting the client.** Similar to the server sequence diagram, except the clients immediately connect to the server, rather than having to listen for connections. The sending and receiving of position packets is identical to that of the server.

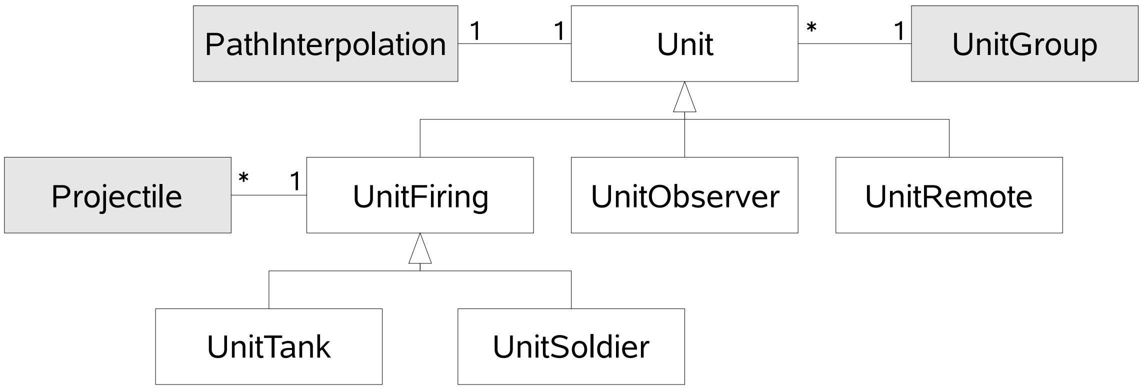


Figure 4.8: Class hierarchy for unit-based classes.

Agent architecture

All of the agents in the synthetic environment inherit from the same `Unit` class that contains the default behaviour of all agents (see Figure 4.8 for details of the `Unit` inheritance hierarchy). This includes the ability to plot a series of waypoints to reach a specified goal position, and the ability to calculate the necessary vectors to reach said waypoints. Various other common properties are defined and listed below.

The path following behaviour is performed by generating piecewise cubic spline (see Appendix D) through each of the waypoints. An index is maintained of the unit's progress along the spline which corresponds to an interpolated point on the spline that the unit steers towards. When the unit is within a certain range of the point, the index is incremented and the unit steers to the next interpolated point on the spline. This continues until the unit reaches the goal position at the end of the spline.

The architecture is designed so that the `init` method is called when the unit is created, and the `update` method is called on each frame of the simulation. Subclasses of the `Unit` class then override methods that are called by default from these methods, such as `loadMeshes` and `initPhysics` from `init` and `updatePhysics` from `update`.

Units can be displayed using any number of 3D meshes, which represent the unit in the 3D view. They can be in any format that OSG can read in, and in this case the meshes are all in *IVE* format, which is the native binary format for OSG. These meshes can be animated using the Character Animation Library 3D to perform realistic movement without using the physics engine. A fragment shader written in GLSL (OpenGL Shader Language) is used to colour the units to either the *red* or *blue* sides so they can easily be identified. The colour is blended with the texture of the unit with an alpha transparency of 0.8. A separate shader is applied when a unit is selected by the user which overlays an animated texture on the unit.

Firing Units

The `UnitFiring` class inherits from `Unit` and provides generic methods and properties for aiming and firing a projectile.

Algorithm 1 Auto-fire behaviour

```
Require: $U$
  loop
    $u$ = selectTarget(this, $U$)
    ($x$, $y$) = predictTargetPosition($u$)
    ($\theta_h$, $\theta_p$) = calculateAim($x$, $y$)
    performAim(this, $\theta_h$, $\theta_p$)
    if isAimed(this, $\theta_h$, $\theta_p$) and isLoaded(this) then
      fire(this)
    end if
  end loop
```

The main algorithm for automatically firing at the opposing units is shown in Algorithm 1. A short description of each of the functions used in that algorithm is given below.

Select target: Find all visible targets within sensor range of the firing unit. The sensor range is also affected by the visibility of the terrain type of the unit being observed. Select the closest of the visible targets found.

Predict target position: Assuming that the velocity of the target unit is known, and the approximate time the projectile is in the air can be calculated, the approximate position of the target unit when the projectile reaches it becomes the target position to aim for.

Calculate aim:

1. **Get heading to position:** Calculate normalised 2D vector to target and calculate normalised 2D vector of the unit's heading. Use these vectors to find their dot product. Calculating the inverse cosine of the result to gets the (magnitude of the) angle (θ_h needed to rotate to head to the target. To find direction of rotation, first find the dot product of target vector and unit heading rotated by 90 degrees. Then if the result is positive negate the angle to turn the opposite way.

2. **Get firing angle:**

$$\theta_p = \tan^{-1}\left(\frac{v^2 \pm \sqrt{v^4 - g(gx^2 + 2yv^2)}}{gx}\right) \tag{4.1}$$

   where v is the projectile velocity, g is acceleration due to gravity, x is the horizontal distance to the target and y is the vertical distance to the target.

3. **Correct for unit pitch:** The pitch (θ_p) has to be corrected for the current pitch of the unit. Rotate the heading vector of the unit to the target vector about the unit's normal, and calculate the angle with the z-axis. Add to this the previously calculated firing angle to get the required pitch.

Perform aim: Overridden by subclasses to begin to rotate the unit firing joints to the correct firing heading and pitch.

Check aim is within thresholds: Overridden by subclasses.

Check loaded: Check that the reload time has elapsed since the last launch.

Fire:

1. **Get projectile launch position:** Overridden by subclasses. Defaults to unit centre.

2. **Get projectile launch vector:** From the heading and pitch angles rotate the heading vector for the unit.

3. **Launch:** Create a new `Projectile` object and assign it a direction and velocity.

Soldier Units

The `UnitSoldier` class is a specialisation of the `UnitFiring` class. Movement is performed by translating the character to a new position corresponding to moving at the unit's specified average speed on each frame, multiplied by the delta frame time. This means the unit will always move as expected, even if the frame rate is not constant. Movement is accompanied by rendering the soldier performing a running action using the character animation framework. When the unit is within the slowing distance of its goal, the unit moves at a slower speed and is shown to be walking. When within the stopping distance of the goal the unit halts and the idle animation is played. Turning is performed by rotating the unit at the specified turning rate on each frame to align the unit's heading with the specified angle.

To fire the soldier must be stationary and the unit's heading must correspond to the aim heading θ_h. Rotating the gun to the required aim pitch θ_p is assumed to be performed instantaneously as it is usually a very small movement.

Tank Units

The `UnitTank` class is a specialisation of the `UnitFiring` class. Movement is performed through the use of the Open Dynamics Engine physics library. The tank uses four wheels, placing one at each corner of the chassis. These are attached to the main body using ODE's hinge-2 joint that simulates a vehicle steering and suspension. The unit moves by applying a target velocity and a maximum force that can be applied to the joint to achieve the desired velocity. Tanks are powerful vehicles, so the maximum force is set quite high—this enables the tank to traverse steep inclines. Steering is achieved by differential drive of the left and right side wheels.

To correctly simulate the tank tracks, the properties of the collision joint that is created when the wheels interact with the terrain are modified so that the coefficient of friction is reduced in the forward direction, but is increased in the perpendicular direction. This stops the tank from sliding sideways down slopes.

Firing is performed by moving the turret and barrel that are attached using ODE's hinge joints. They have one degree of freedom each, which correspond to the firing heading θ_h and pitch θ_p.

Configuration

To avoid recompiling the synthetic environment every time a parameter changes, the starting positions of the units and various other parameters that affect the simulation are loaded at runtime from an XML configuration file. The unit positions are grouped into two sides, (the *red* and *blue* teams) and the units are specified with `unit` tags that specifies the type (either *soldier* or *tank*), the number of units, and the instance number that specifies the client that will control the unit. There are two ways of specifying the positions within the `unit` tag, either a multiple `pos` tags with `x` and `y` attributes, or with `startpos` and `endpos` tags that linearly spaces the specified number of units between the start and end positions.

All of the parameters for each unit type are also read from XML configuration files. This allows quick adjustment of the behaviour of the unit, for example the maximum speed, or the yield of a projectile. Sample XML files are included in Appendix E.

Logging and Replay

All of the events that get sent across the network are also logged to a file. This means the same scenario can be replayed just by parsing the log file and executing the commands as if they were being input by the human commander.

A separate log contains the output of the predictive system, where the confidence of each hypothesis is logged against the time it was calculated.

Targets

It is useful to keep track of the units that are in the vicinity of each other, so the synthetic environment will group units that are within a specified range of each other. These are kept track of by using the `UnitGroup` class.

There are special features on the terrain, such as defensive and city regions. The synthetic environment parses the information from the terrain layers and can provide the closest position of such features for a unit.

4.3.2 Hypothesis Generation

Once the synthetic environment has started and the any clients have connected, the scenario can begin. This first involves creating hypotheses h_i to form H, the set of all active hypotheses. For each agent, hypotheses are created for each applicable inverse model combination. These combinations are formed by taking one model from each category of inverse models that can apply to an agent, as can bee seen from Algorithm 2, where u is the unit and M is a list of lists containing inverse models—where the length of the first dimension is the number of categories $|M|$ and the second is the number of models in each category $|M^i|$. For example, the categories are *formation* or *manoeuvre* models, and each category contains relevant models, e.g. the *formation* category contains *wedge* and *column* inverse models.

Algorithm 2 Generation of hypotheses for a unit

```
Require: $u$, $m^i_j : i \in 1..|M|, j \in 1..|M^i|$
  $x_i = 1, i \in 1..|M|$
  repeat
    $h :=$ new Hypothesis($u$)
    for $i := 1$ to $|M|$ do
      $h$.addIM($m^i_{x_i}$)
    end for
    next := false
    for $i := |M|$ downto 1 do
      if $x_i + 1 \leq |M^i|$ then
        $x_i := x_i + 1$
        next := true
        break
      else
        $x_i := 1$
      end if
    end for$H$.add($h$)
  until next = false
  return $H$
```

Some categories of inverse models only apply to aggregations of agents (e.g., formations apply to agents in a group). Therefore, it is at this point that agents are grouped into hypotheses groups h^g that contain compatible hypotheses (i.e., are assigned the same inverse models) which contain units that are within a certain distance of one another. This is done by pruning all of the hypotheses in H, so that when an hypothesis contains a unit that is within range of a group, the hypothesis is removed from H and added to the group, for details see Algorithm 3. This means more agents can be simulated at the same time as they belong to the same hypothesis group with the same inverse models.

Subsequently, after the hypotheses have been grouped, the set of unique groups of units U^g is extracted from H^g. These unit groups $u^g_i \forall i \in U^g$ contain all of the hypothesis groups h^g that contain the same units, but may differ in their inverse models.

Algorithm 3 Grouping of hypotheses

```
Require: $H$ the list of all hypotheses, $|H|$ the number of hypotheses in $H$
  for $i := 1$ to $|H|$ do
    $h^g$ = new HypothesisGroup()
    $h^g$.add($h_i$)
    $H$.remove($h_j$)
    children.push($h_i$)
    range = $h_i$.getRange()
    while not children.empty() do
      $h$ := children.pop()
      for $j := i$ to $|H|$ do
        if $h$.getUnit().inRange($h_j$.getUnit(), range) and $h$.isCompatible($h_j$) then
          $h^g$.add($h_j$)
          $H$.remove($h_j$)
          children.push($h_j$)
        end if
      end for
    end while
    $H^g$.push($h^g$)
  end for
  return $H^g$ the list of hypothesis groups
```

After each simulation of an hypothesis group, it is evaluated to see if the applicable conditions still hold, i.e., if all the agents are still within range. If not, then all the hypotheses are separated back into H and the same grouping algorithm is executed again.

Prediction Time Length and Quality

As described in Section 4.2.4, the forward model parameters concerning the prediction time length (t_p) and the speed up from real-time (s) need to be set before the hypotheses can be simulated.

The speedup parameter (s) affects the quality of the forward model simulation because the faster the speedup, the greater amount of time is spent performing the physics steps before returning the new positions to the application controller (see Figure 4.4). This means the feedback to correct the control signals gets delayed, reducing the accuracy of the predictions. In this implementation there is a maximum number of physics steps in a single frame to avoid instability due to the delayed feedback. This means the speedup is constrained by the computational power of the computer to simulate the physics step within the allotted time. In this case this speedup was found to be 4 to stay within the maximum physics timestep of 0.2 seconds.

To investigate the effects of changing the prediction time length (t_p) a simple scenario

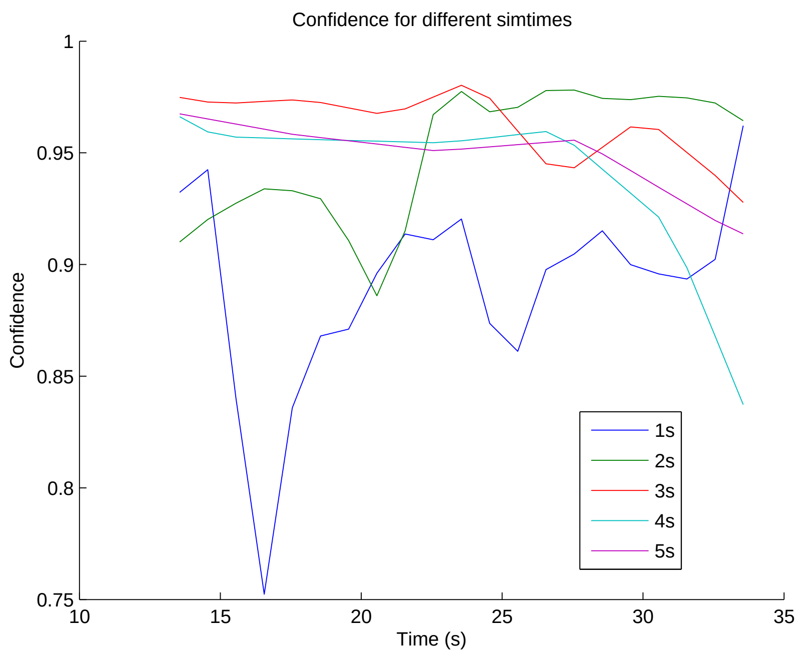


Figure 4.9: **Confidence for different prediction durations.**

was devised where the same inverse model was used to control agents in both the 'real-world' and in simulation. This means that, after accounting for noise, any deviation of the confidence would be due to the effects of changing t_p.

The results in Figure 4.9 show the confidences for different prediction durations averaged over 5 trials. From this graph, a large difference in stability of the predictions can be seen. With just a one second prediction length, the confidence is too volatile. At five seconds and above the confidence appears very stable, but is beginning to introduce a lot of latency between starting a simulation and getting a prediction. In conclusion a t_p of 5 seconds of provides the best trade-off between stability and latency and this value is used for all of the hypotheses throughout the experiments.

4.3.3 Distributed hypothesis clients

Once the hypotheses have been generated, they need to be executed using the internal models. The HAMMER architecture specifies that many of the inverse/forward model pairs should be run in parallel, one for each hypothesis that needs testing. This is challenging because the forward model used here is computationally intensive and only a limited number can be run on a single CPU. To overcome this, the same networking infrastructure to support multiple commanders was utilised to setup the inverse/forward model pairs as *hypothesis clients*, connecting to the server on a different port.

The hosts (K) available to run these clients are specified in an XML file, with a parameter specifying the number of clients m_k to use per host k, giving a total number of clients $d = \sum_{i \in K} m_i$. During the initialisation phase, d clients are launched over SSH and subsequently connect back to the server over a TCP connection. Throughout the scenario, if there are more hypotheses than currently available clients then the hypotheses are queued until a client finishes a prediction and becomes available. Ideally, there would be at most $d \times s$ hypotheses at any one time (where s is the speedup of the forward model), so that all hypotheses can be evaluated at every prediction time interval t_p. However, if this is not the case, then there is a longer wait between subsequent evaluations of an hypothesis.

At the start of each prediction, the current state of the local team's units (U^t)—including any goal parameters—and, as specified by the relevant hypothesis group, the state of a subset of the opposing units (in U^o), are sent to an hypothesis client over the network. This is done by using the Boost serialization library to serialise the state of every unit into a number of packets (one packet per unit, because the maximum packet size is 496 bytes and the average size of a serialised `UnitTank` is 300 bytes). This is implemented by overriding `Unit::load` and `Unit::save` member functions for each `Unit` subclass to serialise the relevant variables for each unit type. See Appendix C for details on the properties that are serialised.

The synthetic environment engine then deserialises the packets so that it can simulate the units starting from the specified state and using the requested inverse model. This is done in a faster than real-time mode with the graphical output disabled. Subsequently the predictions are sent back to the server when the simulation ends using the same serialisation and deserialisation routines.

The interaction between the GUI, the application controller and the network for starting clients and performing an hypotheis is shown in Figures 4.10 and 4.11.

Inverse Models

The inverse models have a simple interface, just the units to control. They also have access to the global information about the terrain and the opposing agents. They output control signals for each of the units under their control, which usually is waypoints to follow, but

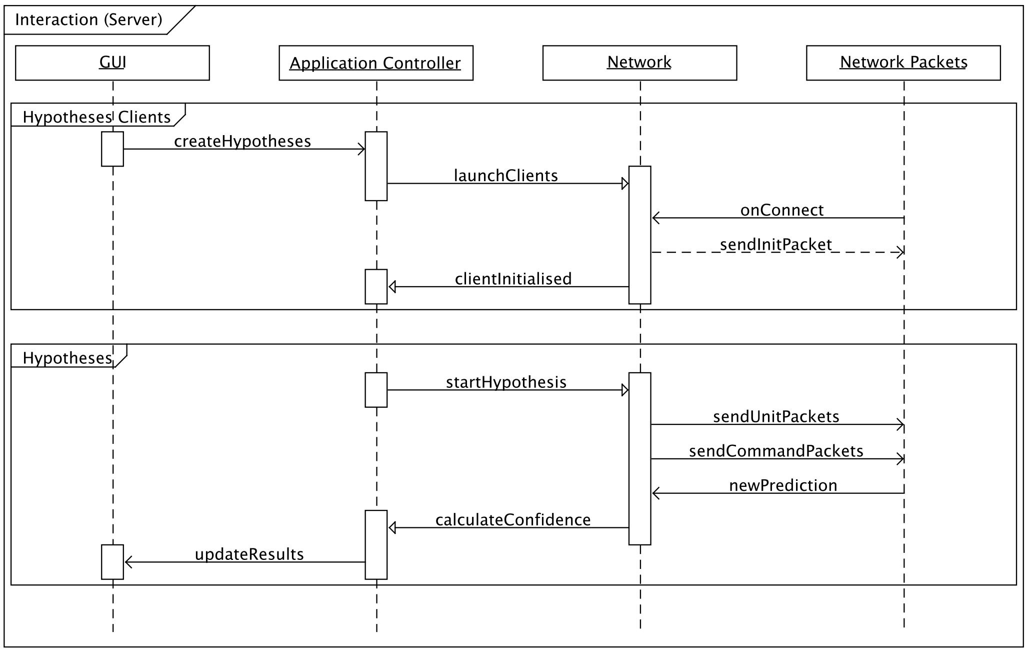


Figure 4.10: **Sequence diagram for the server.** The launch of the hypothesis clients is triggered by the user in the GUI, then the server waits for the clients to connect and it sends out initialising packets. When there is an available client, an hypothesis is started, which involves sending the state of all the units, and the new commands to execute. When the prediction is returned it is used to calculate the confidence, which is displayed to the user.

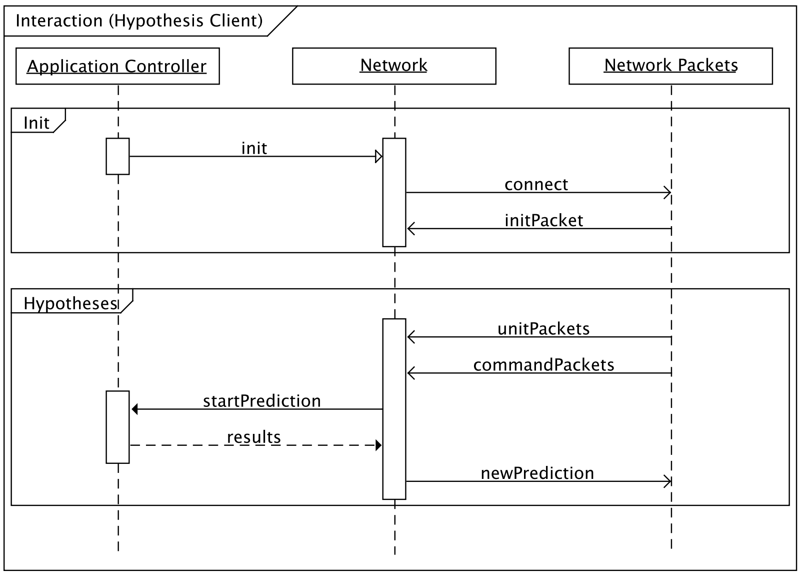


Figure 4.11: **Sequence diagram for the hypothesis clients.** As soon as the client is started it connects back to the server. It then waits to receive units and commands, and sends back predictions.

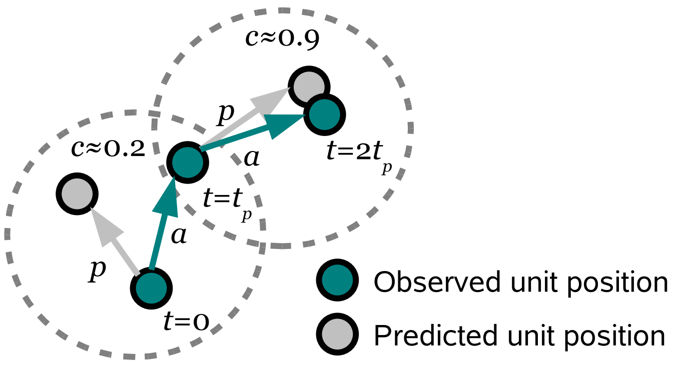


Figure 4.12: **Example calculation of the confidence function.** This diagram shows the result of two iterations of the prediction loop, where the actual (a) and predicted (p) vectors are compared to get the confidence c.

could be other parameters, such as firing target or speed.

The inverse models are grouped into categories that define their exclusivity, unit applicability and mulitiplicity. Exclusivity is controlled so that models in the same category applied to the same unit cannot be simulated concurrently within the same environment (e.g., a unit cannot be attacking and retreating at the same time); unit applicability means there may be a category that only applies to just one type of unit (e.g., tanks or soldiers); multiplicity states whether the model can apply to just one unit, one or more units, or more than one units. Also, if the model can apply to more than one unit then it must also specify the range within which a unit is considered to be in the same group.

4.3.4 Evaluation function

After the requisite prediction time t_p for a hypothesis h there will be an observed ending position for the unit u. Likewise, after the hypothesis client has finished its simulation of duration $\frac{t_p}{s}$ there will be a predicted position for the unit. These positions are then compared with the observed position of the unit at the start of the current prediction iteration. This results in the difference between the actual and predicted positions $c(h)$ (this is known as the error signal in the HAMMER architecture), and is used to calculate the confidence $c(h)$ of the hypothesis, see Figure 4.12 for an example. This can be then be used to calculate an overall confidence of a group $c(h^g)$ that the model that produced the predicted positions actually describes the observed behaviour.

This confidence function is defined as:

$$c(h) = \begin{cases} 1 & \text{if } |\vec{a}| < \epsilon \text{ and } |\vec{p}| < \epsilon, \\ \hat{\vec{a}} \cdot \hat{\vec{p}} \frac{\min(|\vec{a}|,|\vec{p}|)}{\max(|\vec{a}|,|\vec{p}|)} & \text{otherwise.} \end{cases} \tag{4.2}$$

where u is the unit assigned to hypothesis h, $\vec{a}$ is the vector from the start position to the *actual* position for unit u and $\vec{p}$ is the vector from the start position to the *predicted* position for unit u in hypothesis h, $\hat{\vec{a}}$ and $\hat{\vec{p}}$ are the normalised vectors, $|\vec{a}|$ and $|\vec{p}|$ are the length of the vectors, and ϵ is the dead-zone parameter.

This confidence function is able to distinguish whether the unit is moving towards or away the predicted position, or in some other direction. This is accomplished by normalising $\vec{a}$ and $\vec{p}$ (to get $\hat{\vec{p}}$ and $\hat{\vec{a}}$) and taking their dot product. This has the desired characteristics that if the unit moves towards or away from the predicted position then the confidence approaches 1 or -1 respectively, or if it moves perpendicular to the predicted position then the error is zero. Therefore 1 means high confidence, 0 means uncertain confidence, and -1 means no confidence.

The dot product term determines what direction the unit travelled, but it is also useful to know how close it got to the predicted position. However this needs to be invariant to the different speeds of the units, so that the error of different unit types can be compared. Therefore the result of the dot product is scaled by the length of the shortest vector ($\min(|\vec{a}|, |\vec{p}|)$), as a proportion of the length of the longest vector ($\max(|\vec{a}|, |\vec{p}|)$).

An additional condition was added to mitigate the effects of noise within the dynamics engine, so that if the magnitudes of both vectors are within the dead-zone (ϵ) then the confidence is 1. This is to reduce results based on the heading when $|\vec{a}|$ and $|\vec{p}|$ are both very small, but $|\vec{a}|$ is non-zero, which occurs where the unit drifts or slides slightly. Therefore, in this case a dead-zone with a radius of 1m was used, within which the unit is counted as stationary, i.e. if no movement is predicted and there is only a small movement, then we are confident of our prediction, regardless of heading.

This gives the confidence for each unit, however each hypothesis is usually contained within an hypothesis group h^g. Therefore we need to average across all units contained in the hypotheses in the group to get an hypothesis-group level confidence, for a specific time-indexed execution of an hypothesis:

$$c(h^g, t) = \sum_{h \in h^g} \frac{c(h)}{n} \tag{4.3}$$

where h is an hypothesis assigned to the hypothesis group h^g, and n is the total number of hypotheses in h^g. Only when the hypothesis has finished executing in the time between $t-1$ and t does $c(h^g, t)$ return a confidence, otherwise it returns zero.

These confidences are then averaged over a time window to get the current overall confidence of an hypothesis:

$$c(h^g) = \sum_{t=1 \ldots \tau} \frac{c(h^g, t)}{m} \tag{4.4}$$

where τ is the time window, which is 20 seconds in our implementation, and m is the number of times the hypothesis group h^g has been executed in the time window.

The best matching hypothesis group for a particular group of units is:

$$c(u^g) = \max_{h^g \in u^g} (c(h^g)) \tag{4.5}$$

where h^g is an hypothesis group from the set of hypothesis groups that apply to the group of units u^g.

4.4 Experiments

4.4.1 Prototypical Scenario

To test the feasibility of the generative approach, an prototypical scenario was performed using human operators. The experimental setup consists of two humans, each running an instance of the simulator and controlling their (red or blue) team. In this case the predictive system was applied against the blue team's units (the *opponent*).

For this experiment several inverse models were created. Each unit can have both a *manoeuvre* model and a *formation* model applied to it. The *manoeuvre* inverse models that are available for execution are either *attack* or *retreat*, these are defined as follows:

Attack The unit finds the nearest enemy unit and moves towards it until it is in "weapons" range and has clear line-of-sight. At this point it stops and "fires at the enemy".

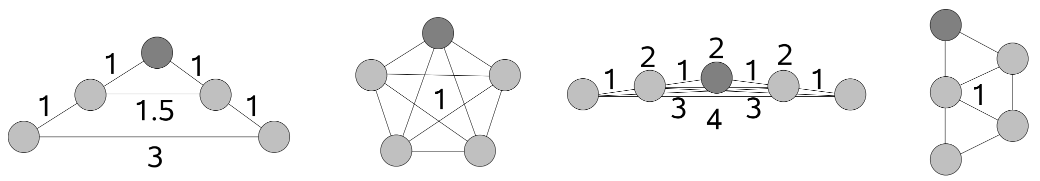


Figure 4.13: **Weighted graphs describing each formation.** From left to right: *wedge*, *circle*, *line* and *column*. The numbers indicate the weight of the connection, higher numbers mean increasing repulsive forces. Those with just one weight specified indicates that all the weights take this value. The darker grey circle indicates the unit that acts as the leader of the formation.

Retreat The unit finds the nearest enemy unit, and if it has clear line-of-sight then the unit moves in the opposite direction until hidden from its sight.

The *formation* inverse models are either *wedge*, *circle*, *line* or *column* and are shown in figure 4.13. Each formation is described by a weighted graph (as defined by Ji and Egerstedt (2007)), so each unit must keep a distance (scaled by the weight) from their neighbours and this distance is assumed to be a constant 50m for this experiment. Therefore their target position is a linear combination of the offsets to each of the positions required to maintain the distance to each of the connected neighbours. Each formation requires a unit to act as the leader and this was implemented by using a heuristic measure to select the unit that is nearest to the target position. This then allows the other positions to be assigned by nearest neighbour, starting with the leader.

For ease of analysis, there are three units on the blue team and one unit on the red team. There are two hypothesis clients and each one is initialised with the world state then run for a pre-specified duration ($t_p = 5$ seconds) at an increased time scale ($s = 4$) and the positions of the units are returned within at most 1.5 seconds. The client is then reset and the process repeated with the next model in the queue, upon the receipt of the next world state. Therefore a result for each model is obtained about every 5 seconds. The start of the scenario is assumed to be at time 0. See Figure 4.14 for the timing—note that the time shown to calculate the confidences and generate the hypotheses is for illustrative purposes only, the actual time is negligable.

As can be seen from the trace of each unit's positions for this experiment (shown in figure 4.15), the red team's objective is to at first to stay hidden behind a ridge, then once the opponent's units are close enough, move over the ridge and attack. The blue team's units move together in an attacking wedge formation, towards the hidden red unit. When

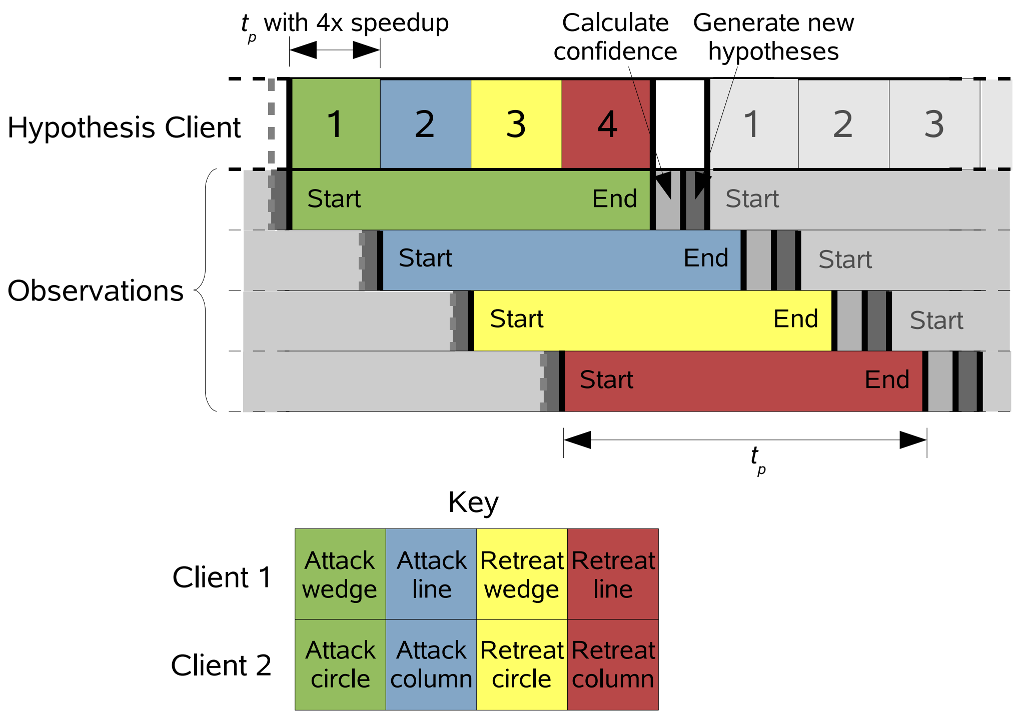


Figure 4.14: **Timing diagram for hypotheses executing on a client.** In this experiment the 8 hypotheses (corresponding to the 8 inverse models) are executed on two clients to maximise the utility of the clients when running at a 4x time scale. When a client finishes a prediction, then next hypothesis in the queue is started and the corresponding start and end observations are recorded at the appropriate point. The key shows the order in which the hypotheses are executed, with the two clients operating in parallel.

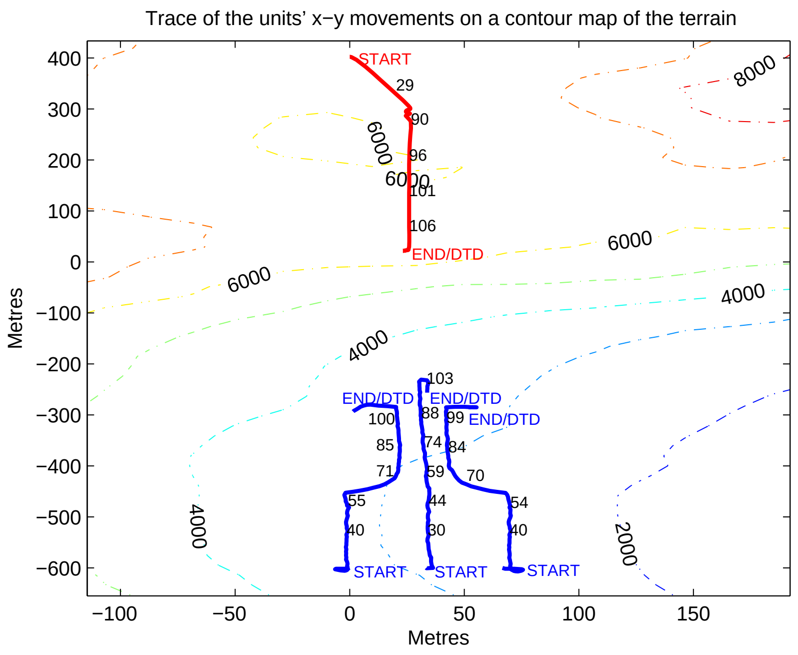


Figure 4.15: **Unit movements.** A contour map (dashed lines), and a trace (thick lines) of the movements of blue units (starting at the bottom, in a valley), and the red unit (starting at the top, on a ridge). The numbers on the traces represents the time step at that point, and the numbers on the contours represent the height of the terrain.

they move closer to the red unit, they move into a line formation (at around time 60). Then when the red unit comes into range they stop and fire at it, until it is destroyed.

4.4.2 Results

The overall confidence of each combination of inverse models is calculated by averaging the confidence of each individual unit within the team, averaged over a 20 second time window using Equation 4.4 and repeated for each hypothesis group, as shown in Figure 4.16. From this it can be seen that it takes about 20 seconds for the the noisy first predictions to settle down and after that an attacking wedge formation becomes the clear winning hypothesis. At about 75 seconds the switch in formation becomes noticable and the attacking line formation becomes the winning hypothesis until the end of the scenario.

The effect of averaging over the 20 second time window can be seen by comparing the raw confidences as obtained by Equation 4.3 in Figure 4.17. The confidences become far more noisy and there are large spikes where erroneous hypotheses are given high confidences, usually when the units are changing formation. However, the change in formation is visible at about 70 seconds, thereby reducing the latency between performing

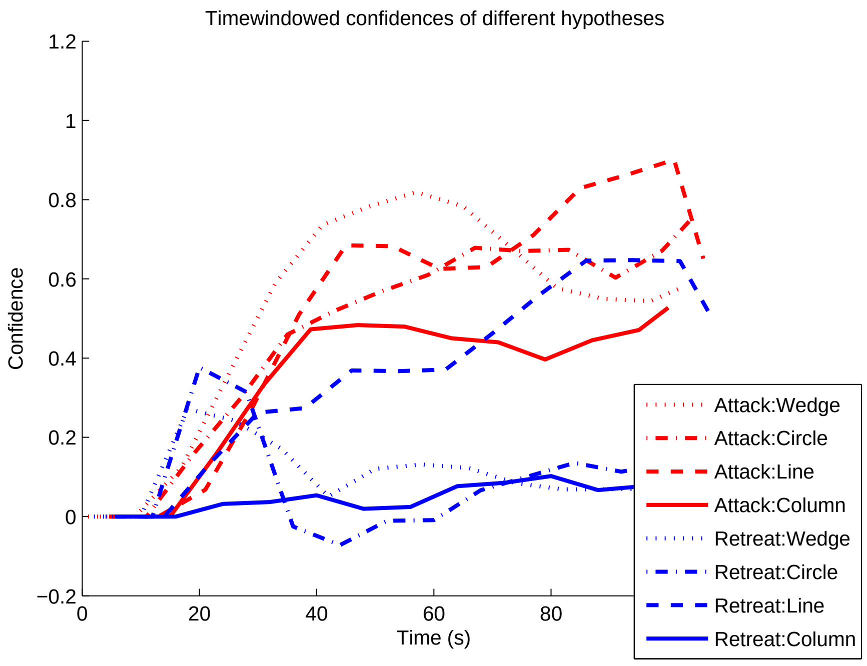


Figure 4.16: **Timewindowed confidences of each model.** Showing the *attack* and *retreat* inverse models, and the formations *wedge*, *circle*, *line* and *column* applied to each. The change in formation can be see at about 75 secs.

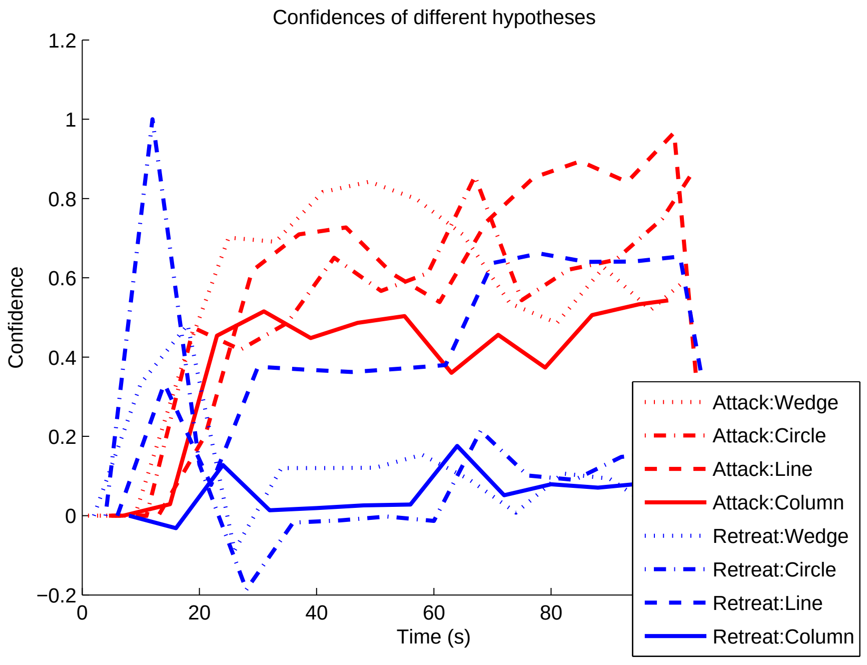


Figure 4.17: **Raw confidences of each model.** Without the averaging over the timewindow, the confidences become considerably more noisy and the winning model is not always clear.

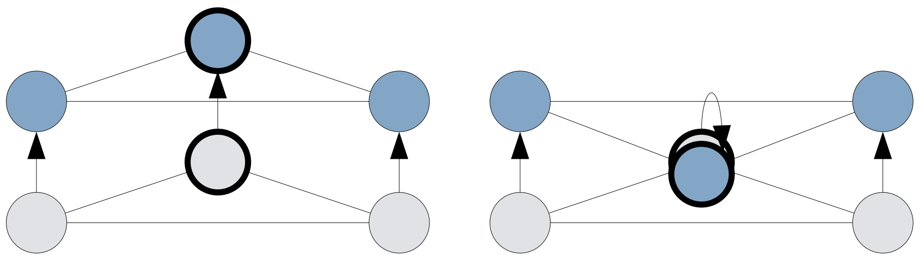


Figure 4.18: **Effect of attacking and retreating hypotheses when moving in a line formation.** The non-leader units end up in similar positions regardless of the hypothesis.

a change and it being recognised by the predictive system. Therefore the raw confidences are advantageous in terms of latency, but at the cost of not filtering out noise.

The relative confidences of each of the different formation hypotheses is broadly as one might expect. The line formation is quite similar to the wedge formation so it has only a slightly lower confidence. The column confidence is a quite different formation to line and wedge, so it has a low confidence. The circle formation is also usually a fairly dissimilar formation to line and wedge, however, with only three units it actually looks quite similar to both, which is why the confidence is not any lower than it would be with more units in the formation.

The one hypotheses that has an unexpectedly high confidence is the retreat with line formation. However, this can be explained by the combination of how the line formation is implemented and the use of just three units in the formation. If the units move forward in an attacking line formation then the units form the same shape translated towards the target, as shown in the left side of Figure 4.18. Whereas, if the retreating line formation is applied to a team already performing an attacking line formation, the leader stops and starts moving in the opposite direction, however due to the repelling forces affecting the other units this pushes them into similar positions as if they had continued as an attacking line formation. This is a rather unique case involving the dynamics of the movement of the units, the weighting on the formation graph, and the prediction duration t_p, and can be mitigated by changing any of those factors.

The leader of the formation will always have the same confidence regardless of the formation being performed. Therefore the effect this has on the confidences can be seen by separating out the different inverse model categories by averaging the confidence values of the leader in all the formations for each *manoeuvre* model (shown in figure 4.19) and

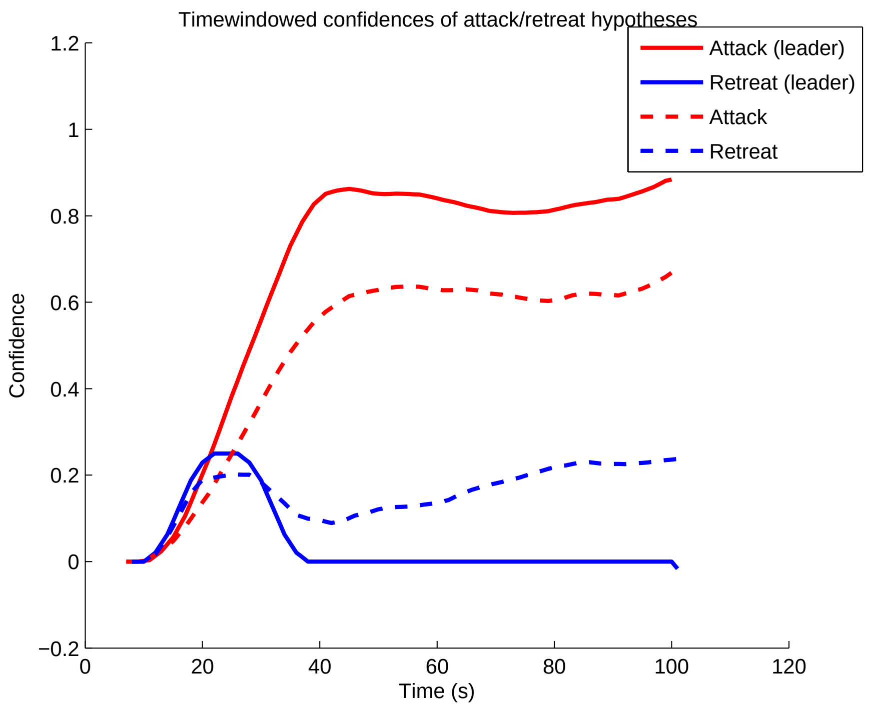


Figure 4.19: **Comparison of manoeuvre confidences of all the units or just the formation leader.** Note the increased separation from the winning hypothesis when just considering the leader. The confidence at each second is the average of the linear interpolated formation hypotheses for the manoeuvre

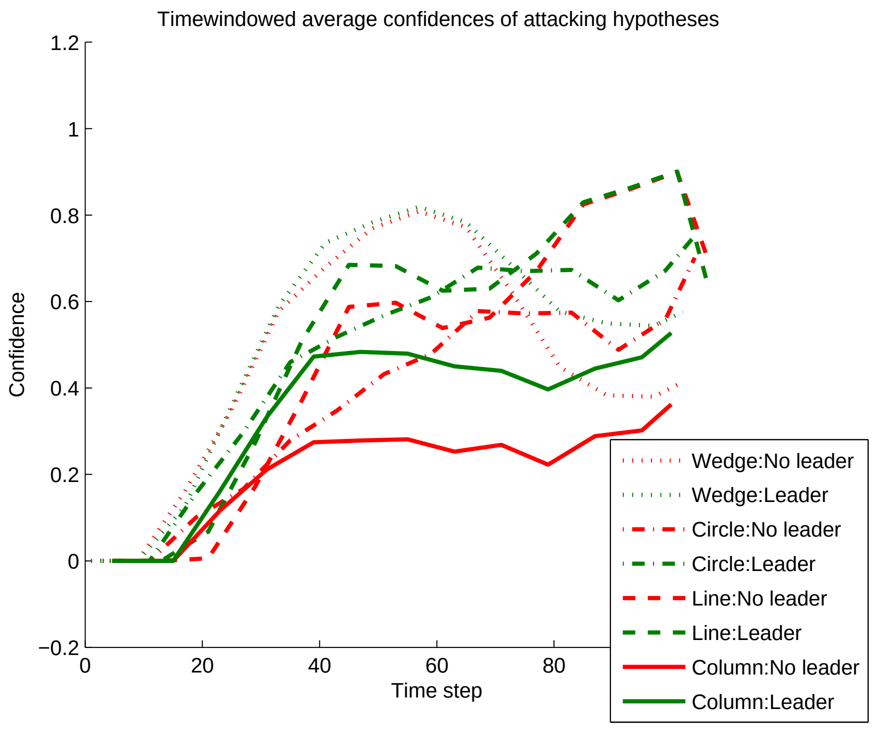


Figure 4.20: **Comparison of confidences including and excluding the formation leader.** Note the increased separation from the winning hypothesis when excluding the leader

by averaging the confidences of the remaining units each of the formations (i.e., excluding the leader) that apply to the winning *manoeuvre* model (shown in figure 4.20). For the attack/retreat hypotheses, separating out the leader doesn't significantly affect the decision between the models, it just increases the separation to the winning hypothesis, so just averaging across all of the models with the same manoeuvre model provides a good result. Likewise, with the formations, removing the leader from the confidence calculation just reduces the average confidence—increasing the separation to the winning hypothesis—but does not affect the overall result.

4.4.3 Intent Experiment

The previous experiment shows the system working in a typical scenario, however the setup does not provide a way to extensively test the different inverse models. To test the capability of the system to correctly identify the intent of the opponent over longer runs, the simulator was modified to not model damage, so that the scenario would not end prematurely if the units came under sustained heavy fire.

Inverse Model Test

A test scenario (shown in Figure 4.21) was performed to establish the correct operation of each of the inverse models.

In the scenario, a team of 5 unit retreat from the opponent's target unit in a wedge formation, then switch to a circle formation. Subsequently the units reverse and start attacking the opposing unit, moving in a line formation, then changing into a column. The experimental setup is maintained from the previous experiment, with two clients, each running 4 hypotheses. However, the target unit remains stationary, and just one human operator controls the movements of the blue units.

The results in Figure 4.22 show that the predictive system correctly recognises the intent of the blue units in this experiment—the hypothesis with the highest confidence corresponds to the actions being performed for nearly all of the duration of the scenario. The only slight anomaly in the results is the relatively high confidence of a retreating line hypothesis when a retreating circle hypothesis is being performed. This highlights a slight drawback in the implementation of the formation–following units, in that the units always travel at their maximum speed, hence when changing from one formation to another, the

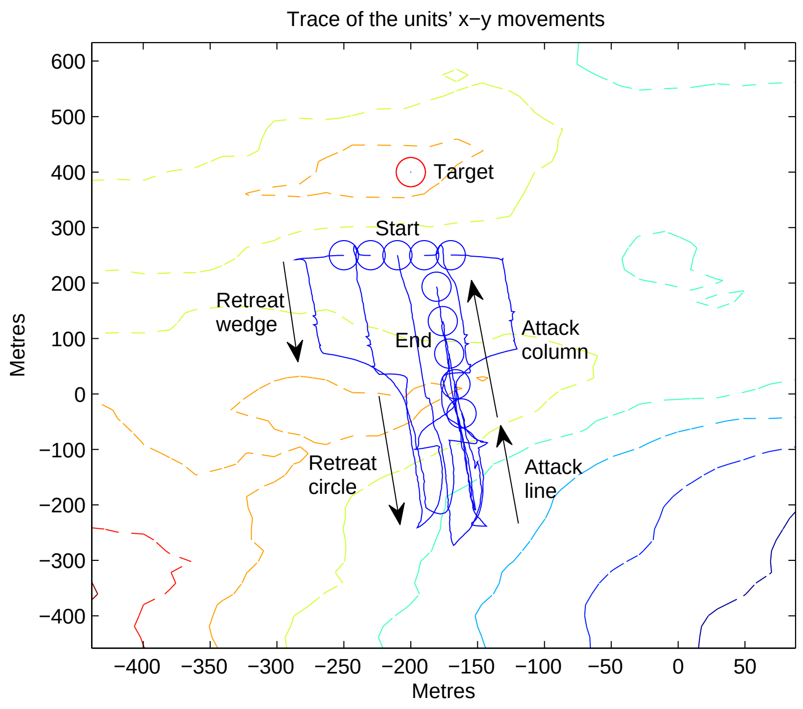


Figure 4.21: **Unit movements.** A contour map (dashed lines), and a trace (thick lines) of the movements of blue units (starting at the bottom), and the red unit (static at the top).

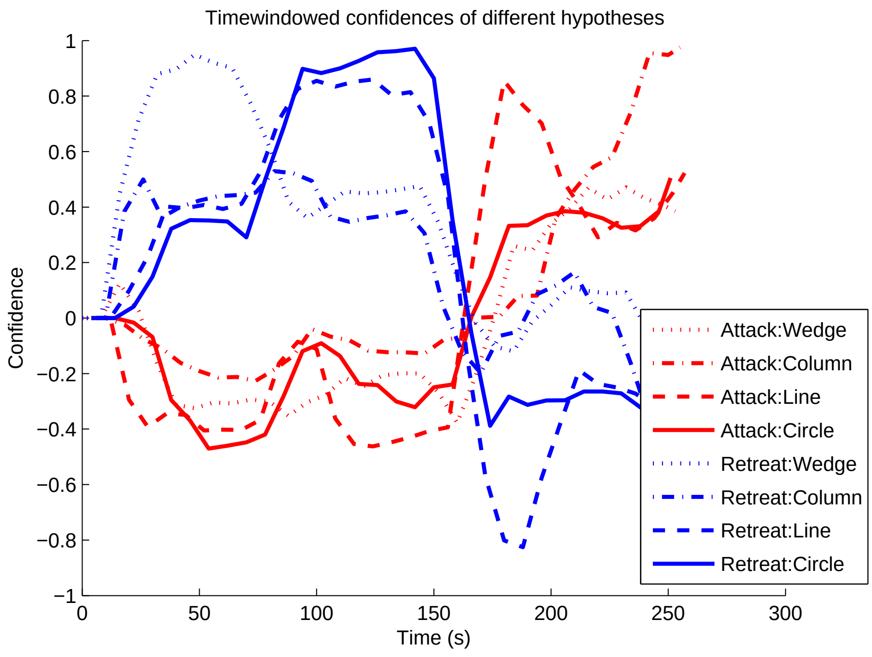


Figure 4.22: **Results of the inverse model test.** The correct hypothesis always has the highest confidence.

Trial	Attack Formation	Accuracy	Retreat Formation	Accuracy
1	Wedge	74.8%	Column	42.0%
2	Column	91.9%	Circle	79.2%
3	Line	92.8%	Wedge	88.3%
4	Circle	93.7%	Wedge	92.8%
5	Line	85.6%	Line	98.7%
Average Accuracy 84.0%				

Table 4.1: Results of the intent accuracy experiment

units cannot move fast enough to get into the correct position for the formation unless the leader slows. Given this limitation, when the units are in the circle formation, they do not differ significantly from when in the line formation, hence the confidences are similar. This problem could be remedied if the formation–following algorithm was reworked to either allow units to speed up to get into position faster, or to slow the leader to achieve the same effect.

It also shows that when using five units in the formation, the side-effect of the line formation being the same in both attack and retreat (for the non-leader units, as shown in Figure 4.18) is mitigated.

Intent Accuracy Experiment

To measure how accurately the system is able to recognise the actions being performed, several trials of the following experiment were performed, so that the percentage of the time the correct hypothesis has the highest confidence can be recorded.

In the experiment there are five units in the centre of the environment, and a choice of 3 opposing units as targets. These targets are all the same distance away from the centre, but in different directions. The instructions given to the operator are to head towards (attack) one of the targets with a randomly chosen formation, then, after 2 minutes, turn around and retreat back to the starting position with another randomly chosen formation.

The same experimental setup is used as in the previous experiment, with 2 clients, 4 hypotheses each, and one human operator.

Results

Five trials were performed and the accuracy of each of the recognised hypotheses are shown in Table 4.1. For full details of each of the trials and the confidences of each hypothesis produced by the system, see Appendix F.

These results show that the system is very good at recognising the actions being performed, with the mean accuracy being 84%. The only apparently anomalous result is the low accuracy of retreat column formation in trial 1. The reason for this is that in this case the human operator decided to stop and change formation, rather than choosing a retreat goal position first, and then changing formation. This meant that the heuristics used to pick the leader of a formation used different information in the hypothesis clients than in the simulator, hence the predictions are different until the units reorganised themselves into the new formation so that the heuristics chose the same leader. This is not a inherent drawback of the system, but a problem with this particular implementation of the heuristics to choose the leader. This could easily be solved if predictions were generated for each of the units being the leader, however it would increase the computational load.

4.5 Discussion

Overall, the experiments show that the system is capable of correctly predicting the intention of a human-operated team of agents in an adversarial environment. The use of averaging over a timewindow does increase the latency of detecting a change in action slightly, but also provides a good way of filtering out the noisy predictions. Additionally, the ability to discriminate between the different inverse models is helped by separating the different categories of models by either isolating or excluding the formation leader. However, in these experiments, this was not needed in order to correctly identify the opponent's intent.

The inverse models (manoeuvres and formations) created to perform these experiments are dependent on this domain, and, particularly with the formation models, some compromises had to be made to the simulation theoretic principles to generate predictions with a manageable number of simulations. For instance, the leader selection heuristics and the fixed weight parameters mean that only one simulation of each formation needed to be performed, rather than a whole series of them. If more computational resources were available, then these restrictions could be relaxed. Furthermore, the system assumes that units travelling within a certain distance of each other are involved in the same action. Again, this was due to the resource limitations—not every combination of units being grouped can be simulated–but it is a reasonable assumption in order to reduce the complexity in these experiments.

The key limitation to acknowledge from these experiments is that, here, all of the combinations of the inverse models are simulated, which introduces a high computational load because only 4 hypotheses can be executed within one t_p without reducing the quality of the predictions. For instance, to accommodate more hypotheses, either the time scale parameter (s) can be increased (which runs the risk of exceeding the maximum time step and causing the physics engine to return unrealistic results), or t_p can be increased (which reduces the frequency that predictions are generated and increases the latency to detect a change in intent).

If the number of opposing groups were to be increased significantly, then there would need to be a corresponding increase in the number of hosts available to run the distributed prediction clients in parallel because the clients are single threaded and utilise the maximum of a single CPU core (there is no upper bound to the frame rate). Additionally, the inverse models are all bespoke, and so the range of behaviour that can be recognised is limited.

4.6 Summary

In summary, this chapter has shown how to adapt HAMMER to the multi-agent domain. This involves: generating and grouping hypotheses for constantly changing goals; finding the inverse model duration and time scale parameters; evaluating the predictions; and the use of a distributed cluster of simulators so that the internal models can be executed in parallel.

The creation of a complex and realistic synthetic environment, and corresponding inverse models, was a significant undertaking in order to perform experiments to validate the approach. These experiments show system is capable of recognising the actions of the opponent in a human vs human scenario. However, the system is limited by high computational requirements and the limited range of behaviour that can be simulated. Approaches to mitigate these limitations are investigated in the rest of this thesis.

Chapter 5

Parameterisation

5.1 Introduction

The inverse models used in the previous chapter were predefined to automatically choose their target parameter, for example, *attack the nearest opponent unit*, or *head to the nearest defensive position.* This covers the most likely behaviour, however, if the agent's were to choose their target on other criteria than shortest distance, then one would expect that the predictions would fail to match the observed behaviour.

This chapter is concerned with deciding how to expand the range of behaviour able to be generated with the inverse models, without increasing the number of models to an unmanageable level.

5.1.1 Complexity

There are two main ways to increase the number of behaviours capable of being produced by the inverse models:

1. Create many more models, each with their own methods to decide on the target, in order to cover the range of expected behaviour.
2. Expand existing models to take a range of parameters, with the parameters being chosen prior to the model being executed.

In either case, the space of possible behaviour that the models need to cover can be large. If a very basic RTS-style game is considered, that has an environment that can be divided into 512x512 segments, has 10 homogeneous units on each team which can rotate (in 1 degree increments) and have properties such as firing and health (as an integer percentage), the state space is 1.9×10^{12}. Obviously, for any real game or simulation, the space is much larger than this, and so any brute force approach to searching this space would be difficult.

Fortunately, the simulation theoretic approach only has to search through the set of possible *actions* rather than the whole state space, so the inverse models have a much reduced space to cover.

This means it could be possible to take the first approach of creating more models in order to increase the range of behaviour of the inverse models—if the actions cover the expected behaviour. For example, there could be several different *attack* inverse models: attack-nearest, attack-weakest, attack-most-threatening, etc. However, in this case, each model would need to be considered as a 'black-box', meaning that all of the inverse models would need to be instantiated concurrently. When considering a large action-space, this could lead to a large number of models both needing to be created and executed.

The alternative approach is to have fewer inverse models, but to parameterise them so that the generated behaviour can be covered by altering the parameters given to the model. This leaves more scope for intelligently searching the parameter-space of the model, thereby reducing the total amount of models needing to be created and executed.

In the rest of this chapter, this second approach of parameterising inverse models will be discussed, along with ways to perform the parameter search.

5.2 Inverse Models and Parameterisation

Given the need to parmeterise the inverse models, below is a list of the possible inverse models for the scenario used in this thesis, and their corresponding parameters.

Single unit u:

Go-to (position = $\{$*enemy, ally, defensive, vantage*$\}$, speed = $\{x < \max\}$, range = $\{\min < x < \max\}$, stealth = $\{$*direct, hidden*$\}$)

Fire-at (target = $\{$*enemy*$\}$)

Group g:

Formation (type = $\{$*line, column, wedge, circle*$\}$, leader = $\{\forall x \in \text{group}\}$, separation = $\{\min < x < \max\}$)

Multiple (one or more) groups g^n:

Manoeuvre (type = {*engage, retreat, surround, ambush*}, target = {*enemy*})

Scout

Rendez-vous (position = {*ally, defensive, vantage*})

Most of the parameters in the inverse models are real numbers (e.g., positions, speeds, distances), but in many cases only a subset of these numbers need to be considered, which greatly reduces the number of inverse models to execute. For example, the position parameter is used by the *go-to* inverse model and can be any point on the terrain. However, even if the possible target positions are reduced to a lower resolution it is not necessary to test every position because certain targets are more likely than others. I.e., units are likely to be heading either to attack a group of opposing units, a good defensive position, or to rendez-vous with another group of units. Another example is the *formation* inverse model that takes a formation type and unit separation parameter. If template-matching is first use to get a set of plausible parameters then the set of possible hypotheses to test is vastly reduced.

This can be taken further by exploiting relationships between sets of parameters so only those that are sufficiently different need to be executed. As an example, optimisation of the target position parameter for the *go-to* inverse model will be discussed in the rest of this chapter.

5.2.1 Hypothesis sets

To support the concept of parameterised inverse models, the hypotheses have to be grouped into sets that share the same inverse model, but have different parameters. Therefore, when the hypotheses are generated (see Algorithm 2), and the inverse model is parameterisable, a set of hypotheses (h^s) is created with a different parameter applied to the hypotheses contained within. These hypotheses are mutually exclusive, so the unit is assumed to be performing the hypothesis that has the highest confidence score in the set. These sets can then be grouped according to Algorithm 3.

The parameters for each set are also updated after each time the confidences are calculated and the hypothesis generator is executed again.

Evaluation function

Given the new set of mutually exclusive hypotheses, the confidence function needs to be updated. Considering if h^g contains an hypothesis set, the best matching hypotheses for a particular group of hypotheses becomes the maximum of the parameterised hypotheses h^s in the set in h^g at time t:

$$c(h^g, t) = \max_{h^s \in h^g} (c(h^s, t)) \tag{5.1}$$

Therefore, the confidence $c(h^s, t)$ has to be modified from Equation 4.3.

$$c(h^s, t) = \sum_{h \in h^s} \frac{c(h)}{n} \tag{5.2}$$

where parameterised hypothesis h^s contains a number of hypotheses h for each unit, which are averaged, given n is the total number of hypotheses in h^s.

5.3 Iterative granular parameterisation

The most accurate way of choosing all the the position parameters for the *go-to* inverse model is to simply choose those positions that correspond to all the possible targets. However, if there are many targets, this would create a lot of hypotheses to evaluate.

One approach to reduce the target positions to a fixed number is to iteratively simulate with a set range of different headings. Subsequently the the angular resolution would be increased in those segments where the confidence is high, and decrease the resolution for those with low confidences. This technique will be termed *iterative granular parameterisation* (IGP).

The algorithm operates as follows: First, the target headings are equally spaced, according to the number of segments (n_s). When the same heading gets the highest confidence more than n_t number of times in a row, and the confidence is greater than a threshold (c_t), the depth is increased one level, and the headings are recalculated based on the starting heading of the winning segment. The angle between the headings is reduced so there is a higher resolution around the main heading, so targets can be resolved into different segments. The angle opposite the main heading is kept wide to cover the remaining area, in case the unit changes direction. This repeats until the depth reaches

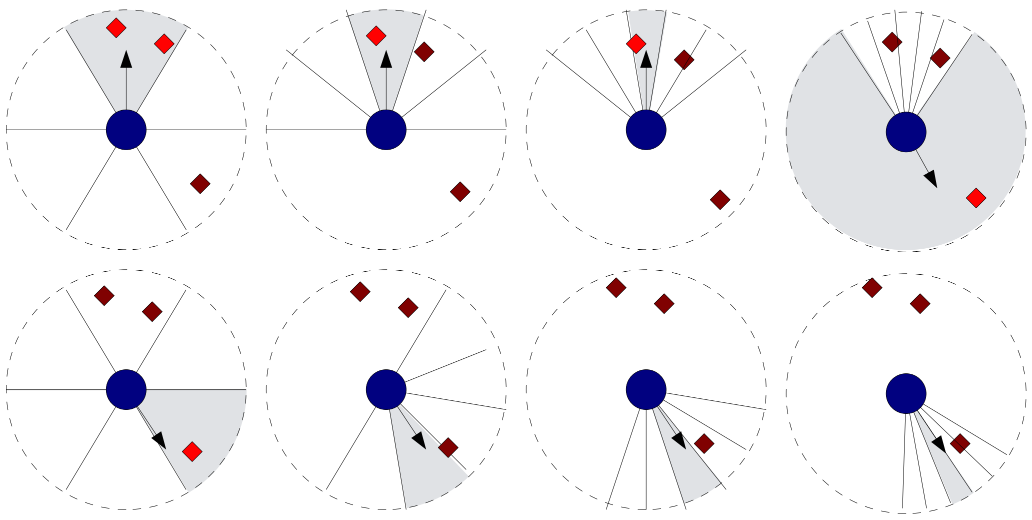


Figure 5.1: **Example of the iterative granular parameterisation applied to target direction.** The top row shows the agent heading towards the topmost agent, with the segments getting increasingly high resolution around the target. When the agent changes direction and starts moving to the bottom right, the segments decrease their resolution back to the initial state, then start increasing their resolution around the target direction again.

a maximum level (n_l). See Figure 5.1 for an example, and Algorithm 4 for the details of the algorithm. Ultimately, the target units that lie within the direction with the highest confidence are considered to be the units that the observed unit is heading towards.

Conceptually, this a mix between a tree search, where the number of segments are the number of children on each node (minus one—as the remaining segment is the parent node) and the number of levels are the levels of the tree, and a gradient-descent optimisation, where the differently sized segments allow for the correct segment to be 'homed in on'.

5.3.1 Experiments

There are a number of parameters to tune for this algorithm, however, the most significant parameters are the number of segments (n_s) as that affects the number of hypotheses that are generated, and the number of levels (n_l) as that affects how many times the resolution can be increased.

To test the effect of changing the algorithm parameters, an experiment was devised that pits an autonomously controlled opponent against a set of static targets. This is so that the human element of controlling the units throughout the scenario can be removed, therefore many trials can be performed and the results easily compared.

To support autonomous operation, the synthetic environment architecture was ex-

Algorithm 4 Iterative granular parameterisation of target direction

```
Require: $n_l$ the number of levels, $n_s$ the number of segments, $d$ the current depth, $n_t$ the
  number of times the same segment has to have the maximum confidence to increase the
  depth, $c_t$ the threshold, $\alpha$ the angle decrease parameter
  $c$ :=getMaxConfidence()
  if $c$ was in the same segment as the last $n_t$ predictions and $c > c_t$ then
    if $d < n_l$ then
      $d := d + 1$
    end if
  else
    if $c$ was in the segment opposite of the previous highest confidence, or $\pm 1$ segment
    then
      $d := 0$
    else if $d > 0$ then
      $d := d - 1$
    end if
  end if
  $\theta_d := \frac{2\pi \exp(d \ln(\alpha))}{n_s}$
  $\theta_{\text{opp}} := \frac{2\pi - \theta_d(n_s - 2)}{2}$
  $\theta$ := main forward segment heading
  for $i := 1$ to $n_s$ do
    if $i$ is opposite segment or opposite segment + 1 then
      head towards $\theta := \theta + \theta_{\text{opp}}$
    else
      head towards $\theta := \theta + i\theta_d$
    end if
  end for
```

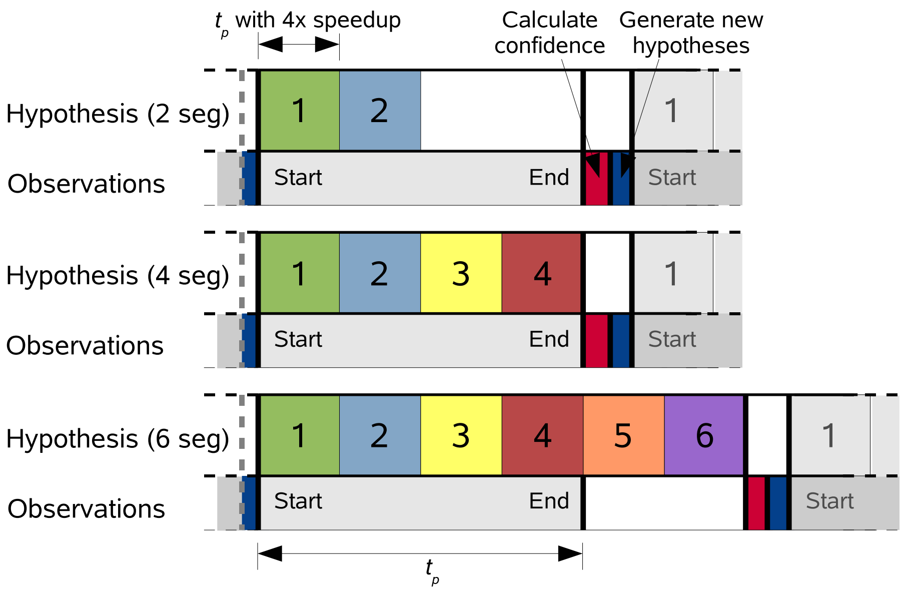
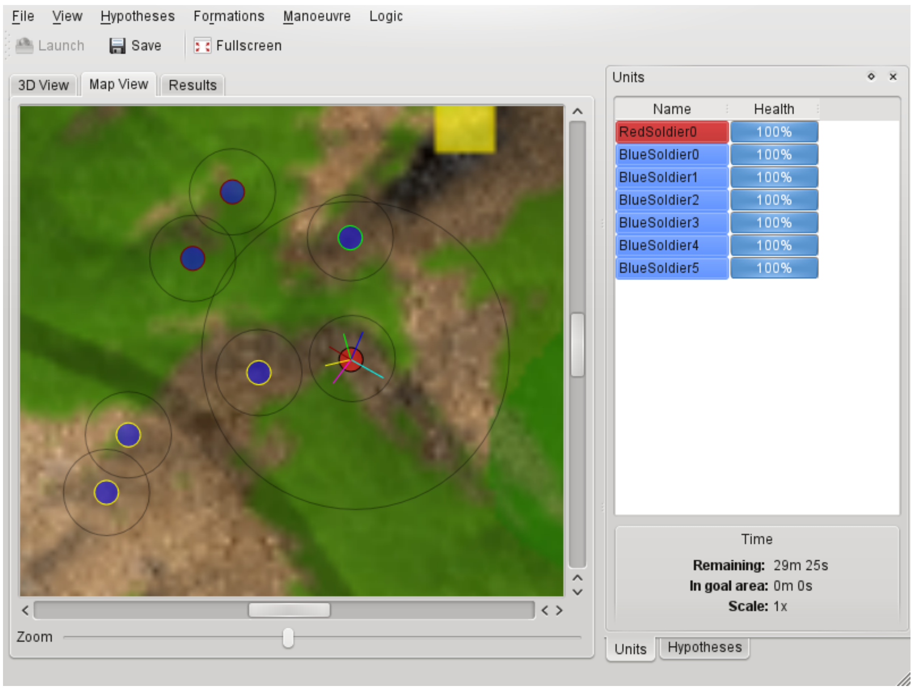


Figure 5.2: **An illustration of the timings for each of the three different numbers of segments used in the experiment.** Note that for two segments the client is 50% utilised, 100% for 4 and 150% for 6. For utilisations of over 100% it means that fewer prediction cycles (i.e. hypothesis sets) can take place within the same time period.

tended with the concept of different *games*. This means that games can be created so that a predefined set of relevant hypotheses are applied to the opponent, and specific inverse models can be applied to both teams according to the rules of the game. The game can be selected by changing the configuration XML file that the environment is launched with.

For this experiment the *direction game* was created that initialises the *direction hypothesis* for the attacker. This hypothesis is responsible for updating the heading parameters after each time the confidences are generated. Therefore there is one hypothesis group in H^g for the one single hypothesis type. The group contains the target units u and an hypothesis set H^s where $s = 1..n_s$ for the target parameter for each segment, making a total of n_s hypothesis instances to evaluate. The interval between observations is set to be 5 seconds, there is one client available for each of the hypothesis sets to execute the internal models, and the forward model contained within it is running at 4 times real-time (see Figure 5.2).

The game also assigns the attacker the *track* inverse model. This model takes a target parameter which is set from a random choice of the three nearest targets that the attacker should head towards. When the attacker reaches within ten metres of the target then the target is removed and the attacker's inverse model is updated with another random target by the game. The game ends when all of the targets have been visited by the attacker.



Name	Health
RedSoldier0	100%
BlueSoldier0	100%
BlueSoldier1	100%
BlueSoldier2	100%
BlueSoldier3	100%
BlueSoldier4	100%
BlueSoldier5	100%

Time

Remaining: 29m 25s
In goal area: 0m 0s
Scale: 1x

Figure 5.3: **A screenshot of the 2D map GUI for the iterative granular parameterisation experiment.** This shows the experimental setup of 6 randomly positioned targets and one attacker. The observing team is running the direction hypothesis on the attacker and the coloured lines show the headings of each of the directions being hypothesised, and correspond to the coloured outline of the targets that are in each of the segments.

There are six randomly positioned targets and one randomly positioned attacker, within a 1000x1000m area, and each unit is separated by at least 150m (see Figure 5.3 for an example trial). The number of segments was either 2, 4 or 6 and the number of levels was from 1–5. This was repeated for 20 trials, giving a total of 300 runs. The other parameters were fixed at $\alpha = 0.6$, $n_t = 1$, $c_t = 0.5$, and initially, $d = 0$.

It should be noted that running more than 4 hypotheses (i.e. more than 4 segments) when using only one hypothesis client will result in fewer hypothesis sets being run within the same time period as when running 4 or less hypotheses. This is because the simulation of the predictions takes longer than t_p, and a complete prediction cycle has to wait for either t_p or for all of the simulations to complete, whichever is longer. This could be mitigated by adding additional clients, however for the purposes of these experiments, the increase in the prediction cycle for 6 segments was not crucial as the targets are spaced far apart.

5.3.2 Results

The obvious way of showing whether the iterative granular parameterisation of the target parameter of a *go-to* inverse model is successful is whether the segment with the highest confidence contains the correct target when the attacker reached each target. The average percentage of times the target was correctly predicted is shown in Figure 5.4.

This shows that, in general, the target is predicted correctly. However, it does highlight a weakness in the algorithm when using just two segments and more than one level. In this case, because the segment headings are based on the result of the previous headings, there can only be two headings—one to the north, and one south. Even though the segments may narrow, this doesn't allow for the main heading to change, therefore, when in a level greater than one, it is very easy to get a higher confidence for the main segment when the target actually lies in the opposite segment. This is because the evaluation function (Equation 4.2) becomes negative when the difference between the predicted and actual positions is greater than 90°. Therefore, because of this flaw, the use of 2 segments is ignored for the rest of the results.

To see how well the parameters perform in distinguishing between the target the attacker is heading towards and those in the vicinity, it is useful to know how many targets are in the highest confidence segment. If there are many then the segments must be large

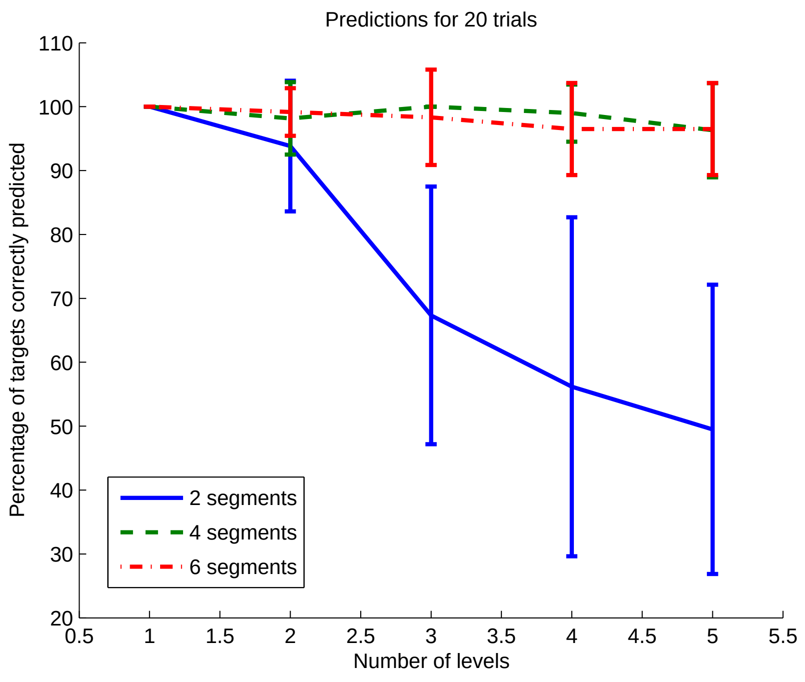


Figure 5.4: **The percentage of targets correctly predicted by the time the attacker reached each target.** A correct prediction is when the segment with the highest confidence contains the same target that the attacker was heading towards.

and the prediction is less useful. If there is on average just one target, then the predictions are perfect.

As can be seen from Figure 5.5, with just one level, there is always, on average, more than one target in the maximum confidence segment. The ideal of just one target in the segment is achieved when using 6 segments with 2 levels and 4 segments with 3 levels. At levels greater than these for the respective number of segments, the average number of targets decreases below 1, meaning that the maximum confidence segment is not always one containing a target. This is likely due to the heading of the attacker not aligning with the heading of a segment, hence the highest confidence oscillates between two segments.

Another measure of success is for what percentage of the predictions did the maximum confidence segment contains only the correct target, over the whole scenario. As shown in Figure 5.6, it is clear that the most consistent predictions were achieved when using only two levels, and that there is not much difference between 4 and 6 segments.

To understand why the percentages in Figure 5.6 are not higher, it is useful to see the percentage of times the target was reached and the maximum confidence segment at any point contains only the correct target. Figure 5.7 shows that it is not always possible to

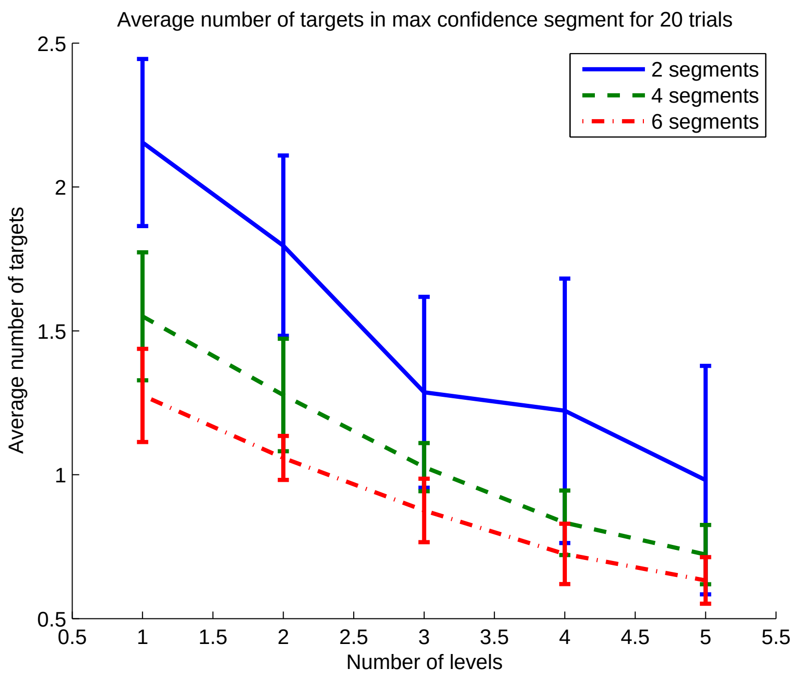


Figure 5.5: **The average number of targets in the maximum confidence segment.** An average on one target is the most useful.

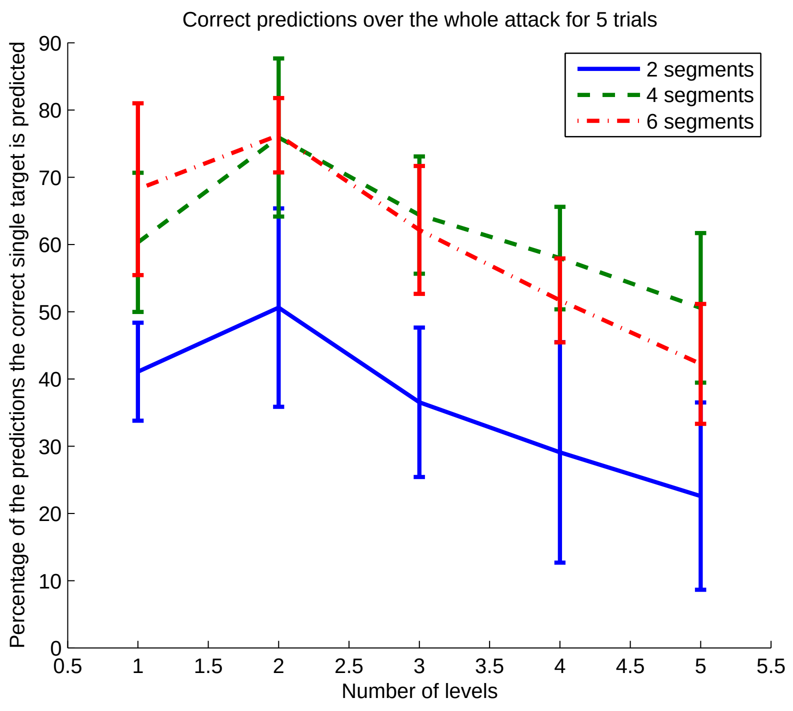


Figure 5.6: **The percentage of times the maximum confidence segment contains only the correct target, over the whole scenario.** It is clear that just 2 levels provides the best predictions.

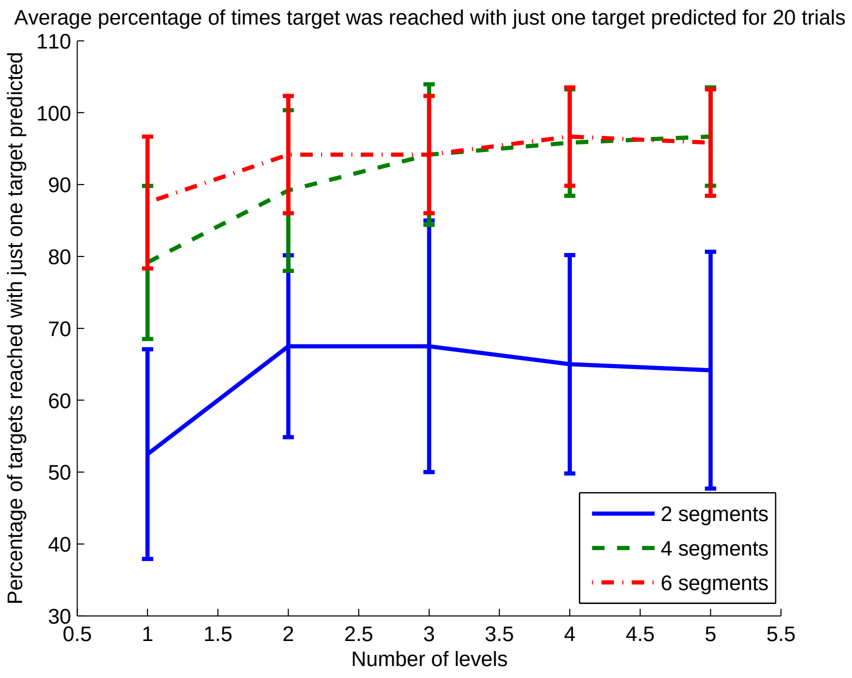


Figure 5.7: **The percentage of times the target was reached and the maximum confidence segment at any point contains only the correct target.** This illustrates that sometimes, no matter the level, getting only the correct target is impossible due to two targets lying on the same heading.

get just a single correct target in the maximum confidence segment because the average is below 100% for all parameters. This is likely because when there is another target behind the target the attacker is heading for, on the same heading, it is impossible to only get the correct target in the maximum confidence segment. It also shows that using 6 segments is slightly better than 4 at identifying the correct target, but the difference is not as clear as one might expect.

5.3.3 Discussion

Using these results to choose the best parameters for the algorithm, it is clear that using 2 segments is not worthwhile without modifying the algorithm. An interesting extension would be to move the centre of the winning segment to the current heading of the attacker, which would go a long way with fixing the performance of using just 2 segments. However, that assumes that the unit's heading can be obtained with good accuracy.

From the results it is clear that using 2 levels gives the best performance out of the 5 levels tested. Also, using 6 segments does not give a significant advantage over 4 segments. Therefore, given the saving in computational resources over running an extra 2 predictions, it is better to use only 4 segments. In this case, as only one client was used in the experiments, the reduction of the number of segments means that the client returns to 100% utilisation, rather than simulation of the predictions taking longer than t_p.

It is interesting that adding extra segments and levels does not give any significant increase in performance. As mentioned above, it could be that the heading of the attacker is not aligned with the heading of a segment, hence the highest confidence oscillates between two segments. It could also be that the density of targets is quite low in these experiments, and if there were more, closely spaced, targets then the extra levels and segments could help.

The other assumption this approach depends on is that the heading of an agent is a good indicator of the target it is heading for, which may not always be the case. This is because the attacker may not always want to move in a straight line, for instance there may be terrain features obstructing the most direct route, or, if the target is also moving, the attacker may be trying to intercept.

Overall, this algorithm shows how inverse model parameters can be chosen to iteratively search the target space, however, it can be difficult to tune the algorithm parameters to give a good trade-off between accuracy and computational cost.

5.4 Parameter clustering

Two of the main drawbacks of IGP are that if the attacker does not head straight towards the target due to untraversable terrain features then the predictions may be wrong, and if there are fewer targets than numbers of segments, then it would be better to use the target as the parameter directly.

A different approach to implementing a parameterised *go-to* inverse model that addresses these flaws with iterative granular parameterisation, is to cluster the targets so only the targets that are sufficiently different are used for hypotheses. The solves the problem of trying to tune to the ideal number of segments. However, the use of uneven terrain means that a simple clustering in euclidean space of the targets is not ideal. This is because, although the targets may be close together, they may be separated by a terrain feature (e.g. a wall) that means the route needed to be taken to each target is very different. This can be solved by taking a two–stage approach:

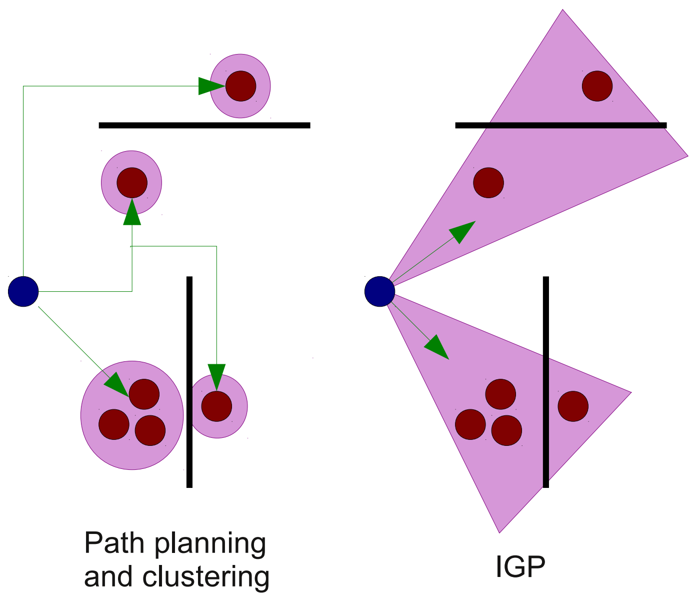


Figure 5.8: **An example of path clustering vs IGP.** The diagram shows that the path clustering generates more appropriate targets when on uneven terrain.

- Generate paths to each of the possible targets with a path planning algorithm (e.g., A*).
- Perform the clustering on the paths themselves, and simulate only the paths that are in the centre of the clusters.

An example of which can be seen in Figure 5.8, comparing it with the IGP approach described previously. The clustering approach is discussed in the remainder of this section.

To instantiate the *go-to* inverse model a unit u or group g is needed to apply it to, so its current (starting) position s is obtained within the environment. It can be assumed that there is a set of r possible targets $T = \{t_1, \ldots, t_r\}$, and a set of r paths $P_s = \{p_1, \ldots, p_r\}$ need to be calculated that go from s to each of the targets in T. Each path consists of m edges between nodes (path segments) in the graph M, where m depends on the route of the path, so $p_i = \{p_i^1, \ldots, p_i^m\}$.

5.4.1 Path Generation

A popular way to calculate an optimal path between two points is by using the A* search algorithm which uses heuristics to reduce the search space Hart et al. (1968). The heuristic is based on the minimal cost path to the target.

The A* algorithm can be summarised by first using the evaluation function $\hat{f}$ in (5.3) for each of the adjacent nodes of a starting node, then selecting the node with the lowest

$\hat{f}$ score. This is repeated for each subsequent selected node until the goal node is reached, whilst maintaining a list of visited nodes to avoid loops. Details of the algorithm can be found in Hart et al. (1968), but the main implementation detail is the evaluation function:

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \tag{5.3}$$

where n is any node in the subgraph M_s that is accessible from the start node s, $\hat{g}(n)$ is the cost of the path from the start node s to n with the minimum cost found so far, and $\hat{h}(n)$ is the least-cost estimate from n to the goal node t.

For the algorithm to always find the optimal path then $\hat{h}(n)$ must always be less than the actual cost of moving from n to t as calculated using $\hat{g}$. This is so that when a candidate path is found to the goal, the other pending paths with a lower $\hat{f}$ can safely be discarded because they cannot replace the estimated $\hat{h}(n)$ term with an actual path of lower cost.

In this implementation, the positions in the environment are represented by three-dimensional vector of single-precision floating point numbers, however the A* algorithm requires distinct nodes to generate paths. Helpfully, the terrain in the game environment is formed by scaling and interpolating a low resolution 512×512 pixel terrain heightmap, so this discretised heightmap grid is used as the graph M to search. This means that all nodes are connected, so $M_s = M$. Distances in the evaluation function are based on travelling on the heightmap rather than in the actual environment so any node $n \in M_s$ has an (x, y) position that indexes the heightmap, and a corresponding height z, so $n = (x, y, z)$. Also each node is assigned a terrain type (either road, forest or normal) which affects the speed of the unit u being routed.

In this scenario the least-cost estimate $\hat{h}(n)$ is an approximation of the minimum time it takes to get from position n to the goal position t. Time is distance divided by speed, and the maximum speed of the unit u is used to get the minimum time. Therefore:

$$\hat{h}(n) = \frac{\|t - n\|}{v_{\max}}$$

where $\|t - n\|$ is the euclidian distance from n to t, and $v_{\max}$ is the maximum speed of this unit type for any terrain type.

The cost $\hat{g}(n)$ is the accumulated minimum cost found so far, plus the cost to the next node n,

$$\hat{g}(n) = \hat{g}(n_p) + \hat{g}(n_p, n)$$

where n_p is the parent node of n, and $\hat{g}(n_p, n)$ is the cost to move from n_p to n which takes into account the distance and speed of the unit u.

This adjacent cost function to move from the parent to an adjacent node $\hat{g}(n_p, n)$ is defined as the distance divided by the speed over the current terrain type, with a limit on the maximum traversable gradient controlled by β:

$$\hat{g}(n_p, n) = \beta \cdot \frac{\|n - n_p\|}{v(n_p, n)} \tag{5.4}$$

where $\|n - n_p\|$ is the euclidian distance between adjacent nodes.

The limit on the maximum traversable gradient is:

$$\beta = \begin{cases} \infty & \text{if } |\Delta(n_p, n)| \geq \mu \\ 1 & \text{otherwise} \end{cases}$$

where $|\Delta(n_p, n)|$ is the absolute gradient between n_p and n and μ is the maximum gradient traversable by the unit u, in this case all units have $\mu = 1$,

The speed over the current terrain type is an average of the speed on the terrain at n and n_p:

$$v(n_p, n) = \frac{v(n_p) + v(n)}{2}$$

where $v(n)$ gets the maximum speed of the unit on the terrain at node n:

$$v(n) = \begin{cases} v_{\text{road}} & \text{if road}(n) \text{ is true} \\ v_{\text{forest}} & \text{if forest}(n) \text{ is true} \\ v_{\text{normal}} & \text{otherwise} \end{cases}$$

where v_τ is the speed of the unit u over terrain type τ, e.g. for a tank on normal terrain its speed $v_{\text{normal}} = 40$, on a road $v_{\text{road}} = 60$ or in a forest $v_{\text{forest}} = 4$, (the unit of speed is

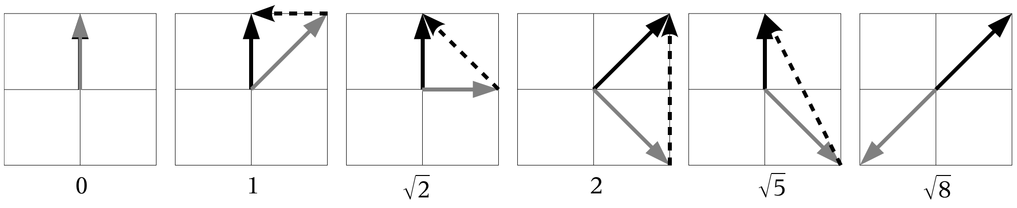
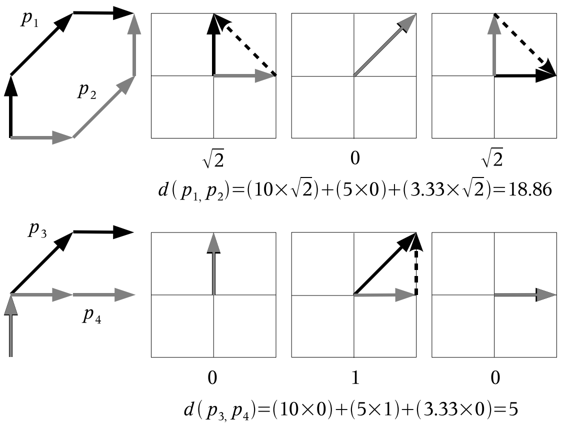


Figure 5.9: **Examples for each of the possible path deviation scores.** The vector for one segment of the first path (p_v^i) is shown in black and a segment from the other path (p_w^i) is in grey. The difference between the vectors is shown as a dashed arrow, and the path deviation score ($\|p_v^i - p_w^i\|$) is the magnitude of this vector, the value of which is shown below each example.



p_1
p_2
$\sqrt{2}$
0
$\sqrt{2}$
$d(p_1, p_2)=(10\times\sqrt{2})+(5\times0)+(3.33\times\sqrt{2})=18.86$
p_3
p_4
0
1
0
$d(p_3, p_4)=(10\times0)+(5\times1)+(3.33\times0)=5$

Figure 5.10: **Two examples of the path distance metric being used on pairs of paths.** For p_1 and p_2 the paths go between the same two points, but take different routes, which gives a high distance measure. Whereas for p_3 and p_4 the paths share a similar path before diverging and ending at different points, yet the distance score is low. Note that differences in the paths near the start of the path have a higher weighting, and low scores indicate good path similarity.

approximately 0.05 terrain divisions per second).

5.4.2 Path distance metric

After creating paths to each of the targets, they need to be clustered into similar groups. Usually clustering algorithms operate on points in a two-dimensional space, using euclidian distance between points as a measure of similarity. However, in this application the paths form a more abstract space, so a new metric is defined to decide how similar two paths are.

For our purposes the location of the destination is less important than how the paths diverge near the starting point. This is because the confidence values for an hypothesis are generated after a short amount of time, so only paths that have diverged in that time can be differentiated.

The paths are made up of vectors that go to adjacent nodes in the terrain map, so paths are compared using the magnitude of the difference between the two vectors for each segment along the path, i.e. a pointwise measure of path deviation (e.g. see Figure 5.9), up to the shortest of the two paths. A scale factor was added that decreases along the path, so deviations at the beginning have more influence over the path distance score, see Figure 5.10. Therefore the distance between two paths p_v and p_w is:

$$d(p_v, p_w) = \sum_{i=1 \ldots m_{\min}} \alpha(i) \cdot \|p_v^i - p_w^i\| \tag{5.5}$$

where $m_{\min} = \min(|p_v|, |p_w|)$ ($|p_v|$ is the number of path segments in path p_v), and $\alpha(i) = \frac{10}{i}$. The constant in the α term was set experimentally.

5.4.3 Path clustering

Once there is a measure of the distance between two paths, it can be used to find the similarity matrix that compares each of the paths with all of the others. After that, several different clustering algorithms could be used, however here a *spectral clustering* algorithm will be used that automatically chooses the number of clusters (Zelnik-Manor and Perona, 2004; Takács et al., 2007).

Spectral clustering is an emerging approach to clustering (Shi and Malik, 2000; Ng et al., 2002; Bach and Jordan, 2006), which is shown to be able to effectively produce groupings equivalent to those produced intuitively by humans (Shi and Malik, 2000; Zelnik-Manor and Perona, 2004). Spectral clustering is also appealing because it is related to manifold learning (Lafon et al., 2006; Ham et al., 2006), which is about finding low-dimensional representations of data along geometrical constraints. It is expected that formations and manoeuvres are subject to such constraints and therefore manifold learning may be able to reduce dimensions effectively in such problems.

From the similarity matrix an affinity matrix is formed using the equation $A_{ij} = \exp\left(\frac{-d^2(p_i,p_j)}{\sigma^2}\right)$ where $\sigma = 10$. The σ term controls the scaling, hence it affects how 'close' the items have to be to be considered to be in the same cluster. This term is set experimentally and only needs to be set once, depending on the order of magnitude of the similarity metric.

Details of the rest of the algorithm can be found in Zelnik-Manor and Perona (2004), but to summarise: First the eigenvectors of the normalised affinity matrix are rotated to get block diagonals using gradient descent. The number of eigenvectors that are used determines the number of clusters, and this is done by finding the number of eigenvectors that produces the best quality rotation. The paths are then assigned to clusters based on the maximum dimension of the rotated eigenvectors.

The MATLAB implementation of the spectral clustering algorithm by Zelnik-Manor and Perona (2004) was reimplemented in C++ using the Eigen[1] matrix library to get a significant speed increase in performing the clustering.

5.4.4 Hypothesis sets

The targets (hence paths) that are finally used by the inverse models are chosen by using a representative path from each cluster. This representative path is chosen by taking the one that is closest to the centre of the the cluster in the eigenspace.

Usually, the hypotheses sets containing these paths would be created on a per-unit basis during Algorithm 2. However, although the clustering algorithm is quite fast to execute, its use should still be minimised as it requires in the order of ten milliseconds to run. Therefore, the hypothesis generation is modified slightly to allow for this. Algorithms 2 and 3 are run first to get the hypothesis groups h^g and the centre of these are used as the starting point to create a path to each of the opponent's positions, and the paths clustered to get the targets. The hypothesis set is then formed by adding the target parameters to the hypotheses, for details see Algorithm 5.

5.4.5 Experiments

To test the effectiveness of the path-to-target hypothesis that uses the path clustering, it is compared against the straight-line-to-target hypothesis for a single trial. The human-controlled opponent ('red' team) has one unit, and has the objective to destroy one of the 3 targets (part of the 'blue' team) that are positioned throughout the environment.

The environment and location of the targets is shown in Figure 5.11. A barrier is shown going east-west across the environment with a small gap in the middle. Targets 1 and 2 lie beyond the barrier, one on either side of the gap. A third target lies behind a barrier that is only accessible by going around it to the north. The scenario is played out

[1]http://eigen.tuxfamily.org/

Algorithm 5 Creation of hypotheses sets

```
Require: $H^g$ the list of all hypothesis groups, $U^t$ the list of all the targets
  for $i := 1$ to $|H^g|$ do
    if $h_i^g$ contains parameterised target hypotheses then
      for $j := 1$ to $|U^t|$ do
        $p_j :=$ path to $u_j^t$
      end for
      for $j := 1$ to $|P|$ do
        for $k := 1$ to $|P|$ do
          if $j <> k$ then
            $a_{jk} := \exp(\frac{-\text{comparePaths}(p_j, p_k)}{\sigma^2})$
          end if
        end for
      end for
      $C := \text{cluster}(a)$
      $T :=$ first target in each cluster from $C$
      for $j := 1$ to $|T|$ do
        add $h^s$ with a set parameter $t_j$ to $h_i^g$
      end for
    end if
  end for
  return $H^g$ the list of hypothesis groups
```

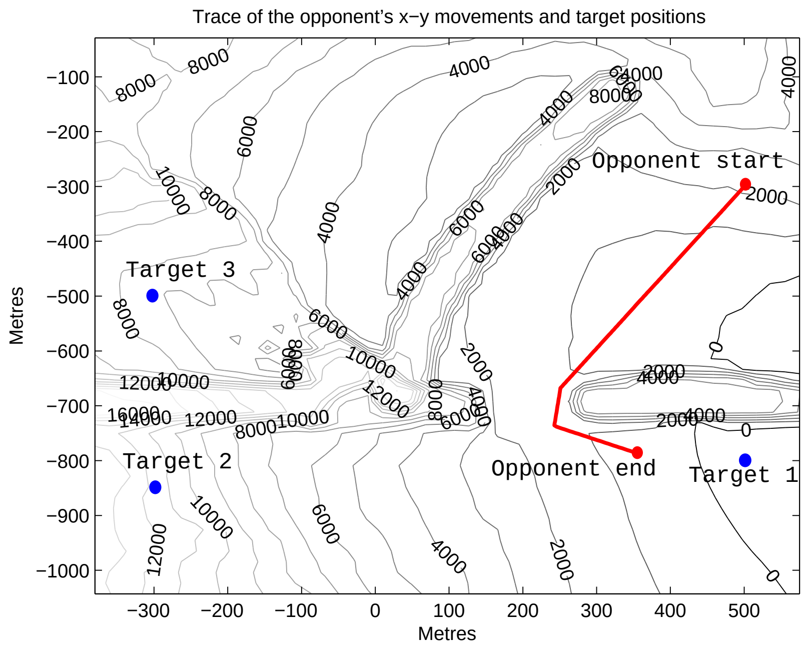


Figure 5.11: **Scenario overview.** This shows the movement of the opponent's unit, and the positions of the target units for the duration of the scenario.

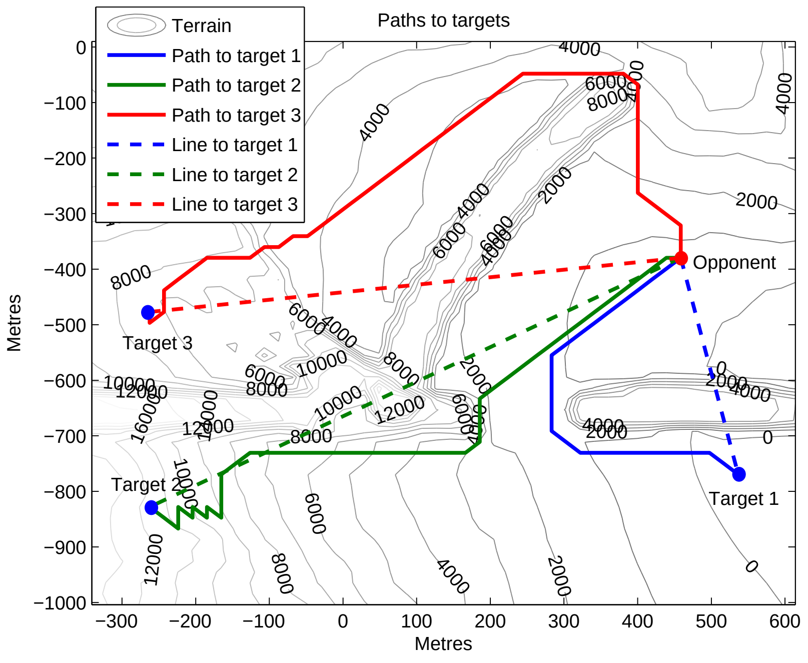


Figure 5.12: **Predicted paths.** This shows the paths for both hypotheses at timestep 41. It can be seen how the paths to targets 1 and 2 have a high path similarity, and that the straight-lines are quite dissimilar to the paths for targets 1 and 3.

so that the opponent heads towards the gap in the barrier which leads to Targets 1 and 2, and then, at around timestep 130, it goes towards Target 1. Target 3 remains behind the barrier so that it requires the opponent to move in the opposite direction in order to reach it. The targets are static throughout the scenario.

The two hypotheses (path-to-target and straight-line-to-target) were instantiated for the opponent's unit, with the target positions as parameters. Therefore there are two hypothesis groups in H^g for the two hypothesis types, each containing the same unit u and an hypothesis set H^s where $s = 1..3$ for the three target parameters, making a total of six hypothesis instances to evaluate, see Figure 5.13. As before, the interval between observations is set to be 5 seconds, there is one client available for each of the hypothesis sets to execute the internal models, and the forward model contained within it is running at 4 times real-time. This means each each internal simulation runs for approximately 1.25 seconds to generate a prediction, therefore all instances in each set can complete within the interval. Hence, each hypothesis set generates confidence values for each target approximately once every 5 seconds. To easily illustrate how the clustering approach helps in reducing the number of parameters that need to be simulated, for this experiment the clustering algorithm was restricted to always produce one less than the maximum number

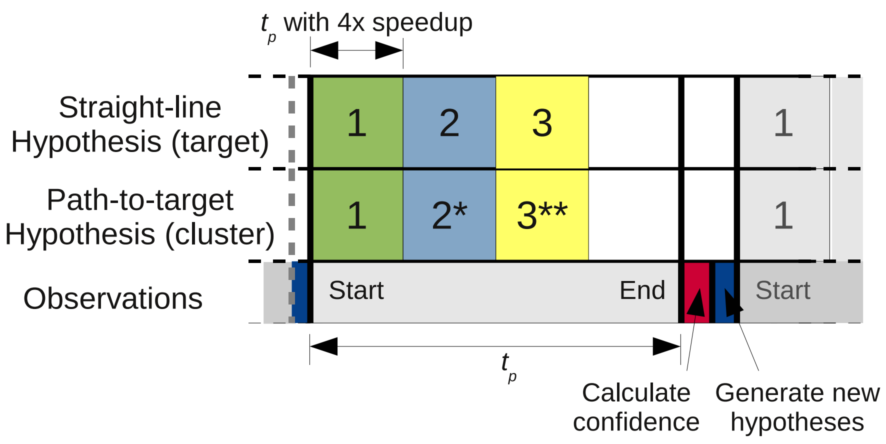


Figure 5.13: **Timing diagram for the two hypotheses.** The straight-line-to-target hypothesis always requires three parameters to be simulated, whereas the path-to-target hypothesis clusters the paths so can use from 1–3 parameters. *optional. **not used in this experiment.

of clusters where each cluster only contains one unit—so in this case 1 or 2 clusters. This restriction is lifted in the subsequent experiments in this chapter.

The path planning and clustering occurs each time the path-to-target hypothesis set starts a new prediction (in the *generate new hypotheses* section marked in Figure 5.13), and it is used to decide which hypothesis instances (i.e., targets) need to be executed. The time taken to compute this is negligible (approx. 10ms) relative to the savings from any reduction in the number of hypothesis instances executed.

An example of the outputs of the inverse models for both of the straight-line-to-target hypothesis and the path-to-target hypothesis for each target at timestep 41 is shown in Figure 5.12. From this it can be seen how the paths to targets 1 and 2 have a high path similarity, and that the straight-lines are quite dissimilar to the paths for targets 1 and 3.

5.4.6 Results

The confidence level to each of the targets using the straight-line-to-target hypothesis is shown in Figure 5.14. The highest confidence target is possibly correct throughout the scenario, however, as expected, the terrain features are not considered so it misses the fact that the opponent also has a high likelihood of additionally heading to Target 2 up until timestep 130, and that Target 3 is an unlikely target as the shortest accessible path is in the opposite direction to the observed movement.

As can be seen from Figure 5.15 the path-to-target hypothesis correctly identifies

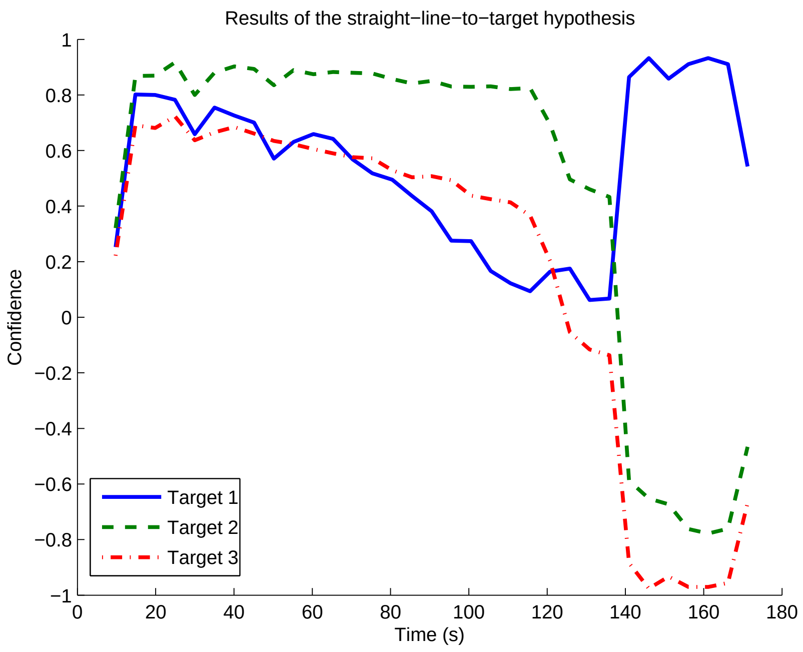


Figure 5.14: **Target confidence using the straight-line-to-target hypothesis.** Target 1 is only identified as the most confident target after timestep 140.

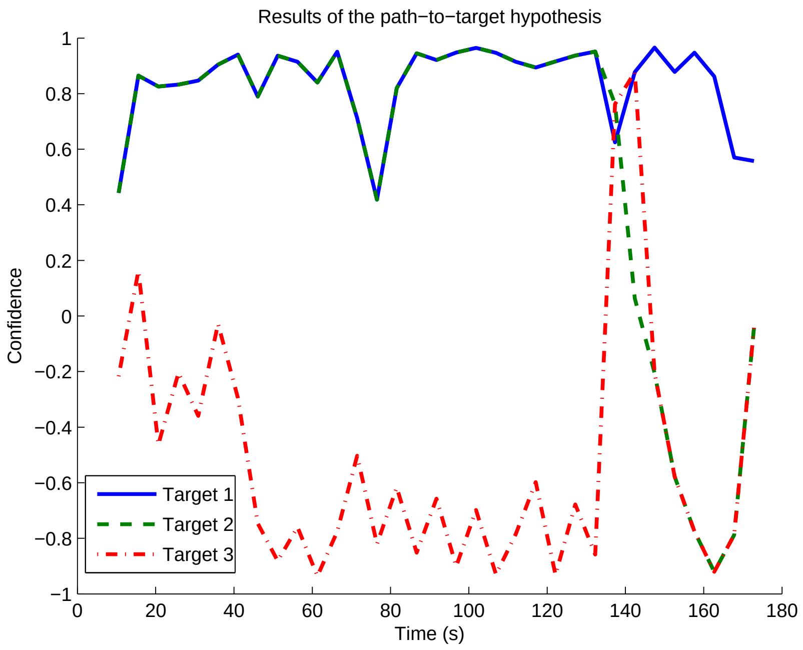


Figure 5.15: **Target confidence using the path-to-target hypothesis.** Target 1 has a consistently high confidence throughout the scenario.

and combines Targets 1 and 2 in the first part of the scenario, whilst giving Target 3 a low confidence. After the opponent traverses the gap it correctly splits off Target 1 and gives it a highest confidence, and combines the other two targets and assigns them a low confidence. There is a small artifact at around timestep 140 where the clustering algorithm combines targets 2 and 3, and then combines targets 1 and 3. This is because, as explained previously, the experiment is setup to only allow one or two clusters, so in these two cases the clustering algorithm determines that the cluster with two targets should include target 3, which is slightly counter–intuitive, however, at timestep 140, the three paths initially diverge in three equally spaced directions from the opponent, and, in this case, forcing the grouping of two together means that one of them is target 3, even though the clusters are far apart in eigenspace.

5.4.7 Discussion

It can also be seen that the path-to-target hypothesis manages to out-perform the straight-line-to-target hypothesis whilst also executing the hypothesis with the need for fewer computational resources. Obviously it is not a surprise that path-planning improves the accuracy on an uneven terrain, but it does show that it can be done even when executing fewer internal models. This is because throughout the path clustering algorithm combines two of the paths that are similar, hence only paths to two out of the three targets need to be executed and tested.

It also shows that the path planning inverse model produces similar routes than chosen by a human operator as the maximum confidence hypothesis averages approximately 0.8.

5.5 Comparison of Parameterisation Approaches

In this final section of the chapter, the performance of the iterative granular parameterisation will be compared with the path clustering approach to inverse model parameterisation in a number of automated trials. This is to show how well both methods compare when reducing the number of parameters that need to be simulated from the standard 6 parameters for the straight-line go-to inverse model.

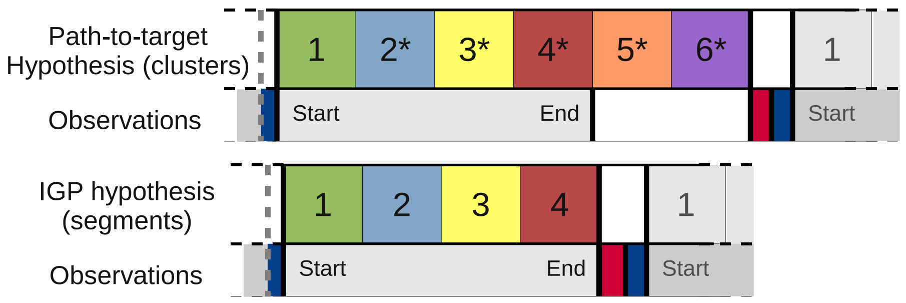


Figure 5.16: **Timing diagram showing the number of parameters for the path-to-target and the IGP hypotheses and how they compare to the observation time.** The IGP parameters are fixed to 4 segments, whereas the path-to-target clusters the targets, resulting in 1–6 parameters, depending on the number of clusters chosen (*optional). If there are 4 or fewer clusters then the observation timing becomes the same as IGP.

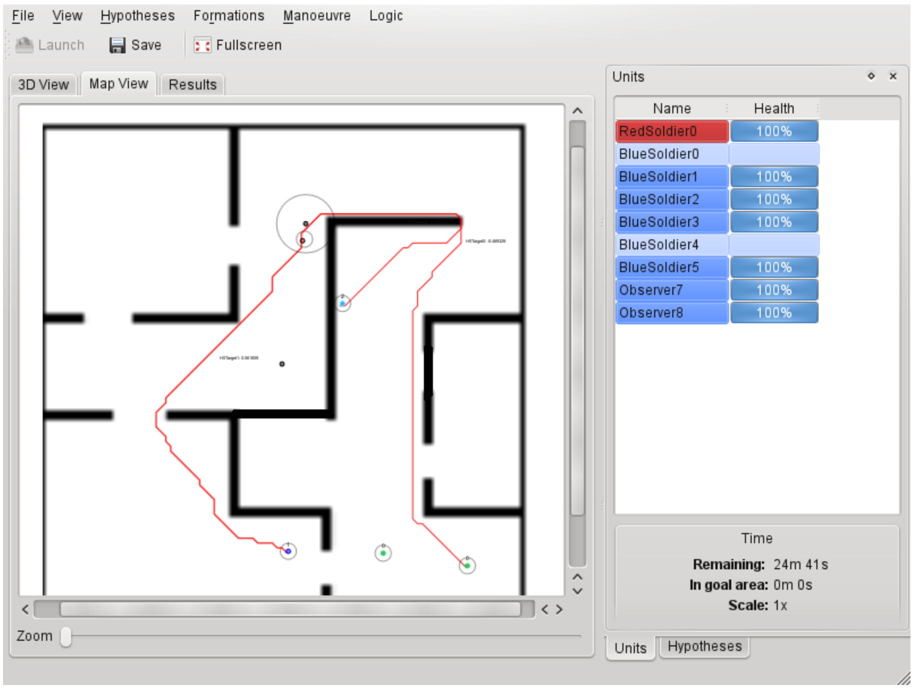
5.5.1 Experiments

To compare the the parameter clustering approach with IGP, the terrain was modified to have a maze-like heightmap, with multiple routes to a target, as shown in Figures 5.17 and 5.18. Whilst this does favour the path planning of the clustering approach, the routes are realistic and, given the size of the segments, IGP should be able to cope with some path deviations from the direct route. Twenty trials were performed using a similar experimental setup as for IGP with 6 static targets and one attacker. The difference is that the attacker uses an inverse model that creates a path to the target over the uneven terrain. Each trial consists of one run using the path clustering and another run using IGP. For the IGP runs, the number of segments (n_s) was 4 and the number of levels (n_l) was 2. The number of parameters this creates for both hypotheses can be seen in Figure 5.16.

5.5.2 Results

As can be seen from Figure 5.19, the path clustering performs much better than IGP. Over 95% of the time the path clustering approach predicts the correct target, whereas the with IGP, the correct target is only predicted 60–75% of the time, with a much larger standard deviation.

Figure 5.20 shows the breakdown of how many clusters are chosen throughout the scenario. The average number of clusters is 2.6. This compares very favourably to the 4 segments that IGP uses, which translates to simulating, on average, 1.4 fewer targets per



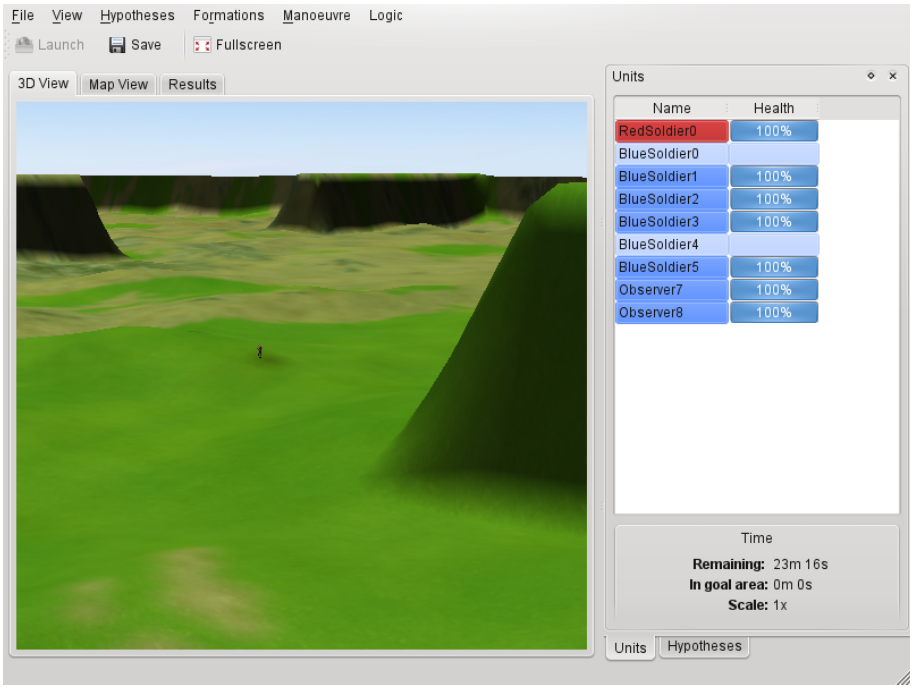
Units

Name	Health
RedSoldier0	100%
BlueSoldier0	
BlueSoldier1	100%
BlueSoldier2	100%
BlueSoldier3	100%
BlueSoldier4	
BlueSoldier5	100%
Observer7	100%
Observer8	100%

Time

Remaining: 24m 41s
In goal area: 0m 0s
Scale: 1x

Figure 5.17: **Screenshot of the map interface showing the maze scenario.** This screenshot was taken part way through a scenario, after two targets have been destroyed. The interface shows that three of the four targets have been chosen for predictions, and the predicted paths are shown, along with their current confidence.



Units

Name	Health
RedSoldier0	100%
BlueSoldier0	
BlueSoldier1	100%
BlueSoldier2	100%
BlueSoldier3	100%
BlueSoldier4	
BlueSoldier5	100%
Observer7	100%
Observer8	100%

Time

Remaining: 23m 16s
In goal area: 0m 0s
Scale: 1x

Figure 5.18: **Screenshot of the 3D interface showing the same scenario.** Here, the attacker is shown in the centre of the view, and the scale of the terrain can be seen.

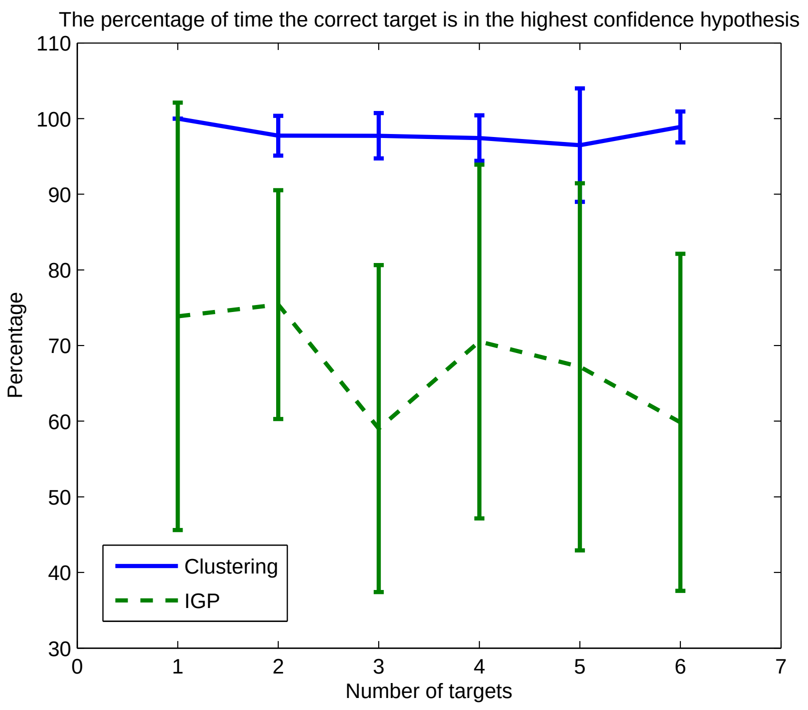


Figure 5.19: **How often the correct target predicted for both the clustering and IGP approaches.** Generating paths to targets clearly gives better predictions when using an uneven terrain.

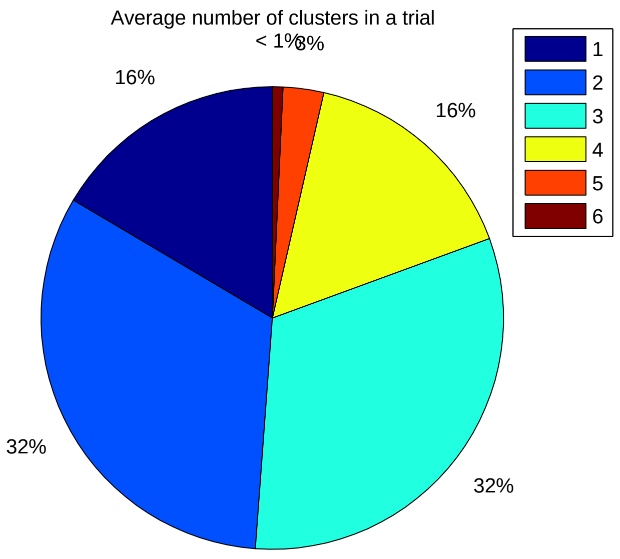


Figure 5.20: **The number of clusters chosen throughout a trial.** This shows that there are rarely 5 or 6 clusters, giving, on average, fewer number of parameters needing to be simulated than the IGP approach using 4 segments.

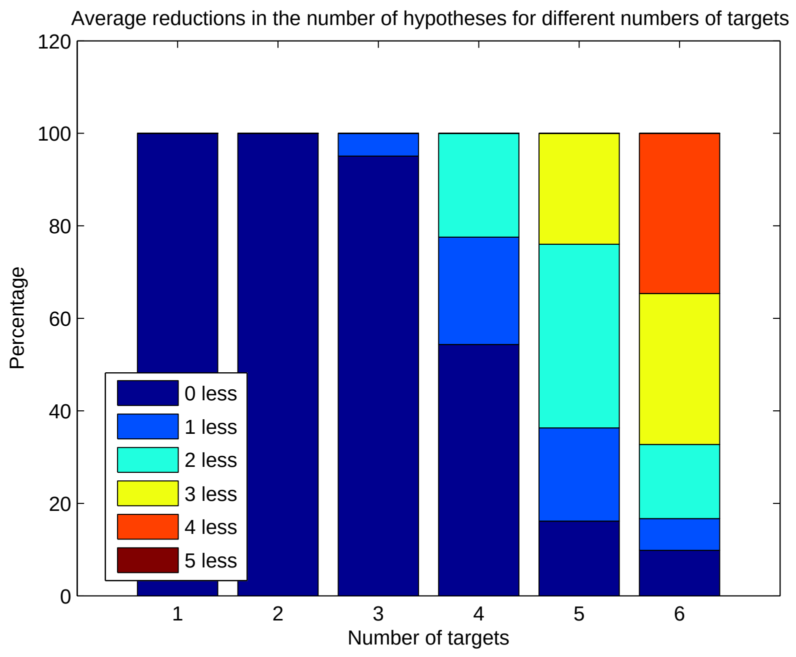


Figure 5.21: **The reduction in the number of parameters that are simulated.** This is compared with simulating a path to every target. It shows that as the number of available targets increases, the greater the potential for the clustering to reduce the number of targets needing to be simulated.

hypothesis set by using the clustering, giving good computational savings.

Figure 5.21 illustrates the advantages of clustering over simply generating paths to each of the targets. This is done by showing the reduction in the number of hypothesis parameters that are simulated, for different numbers of available targets. For example if there are paths to 5 targets and these are grouped into 3 clusters, then only 3 target parameters need to be simulated, giving a reduction of 2 target parameters. From the figure it can be seen that the clustering approach does not have any clusters containing more than one target when there are only one or two targets (the first two columns are dark blue, showing no reduction in the number of clusters, i.e. the number of clusters equals the total number of targets). However, when the number of targets increases, the benefits of the clustering become clear, so that, for example, about 40% of the time when there are 6 targets, only paths to 2 of those need to be simulated (shown in orange as a reduction of 4 targets over the maximum 6, leaving only paths to 2 targets actually needing to be simulated).

5.5.3 Discussion

Clearly, the path clustering approach performs better than IGP when using an uneven terrain. The benefits of clustering are more apparent when there are more targets, but it also out-performs IGP when there are fewer than 4 targets as it only generates paths to the available targets.

The drawback of performing the clustering is that it requires there to be a discrete set of possible parameters, with a similarity measure between them. This works well for targets and positions, but is less easy to define for other linear parameters (e.g., distances or speeds) that other inverse models may require. In these cases, IGP is easier to adapt, as it already searches a linear parameter space (the attacker's heading).

5.6 Summary

This chapter has shown how it is advantageous to parameterise the inverse models to reduce the over number of models and give a wider space of actions that can be simulated.

Two methods for searching the parameter space of the *go-to* inverse model were investigated. Iterative granular parameterisation works based on simulating possible headings, but is hard to tune to the specific number and density of possible targets and assumes the attacker only moves in a straight line. The clustering of paths works better in uneven terrain, because the the path is more realistic, there are less parameters to tune, and the number of simultaneous parameters to simulate gives good computational savings, at the slight extra cost of performing the clustering.

Chapter 6

Attention

6.1 Introduction

Within certain domains, such as in Real-Time Strategy (RTS) games, knowledge about the opponent is usually obtained by simply assuming that perfect global information is always available. In this domain the computer-controlled player usually has information about the human player's units, even though the converse is not true: the human player can only see their immediate surroundings. This has the advantage of making the AI system easier to design, but can lead the human player to feel that the game is not being played on a level playing field, as the computer-controlled player has gained an unfair advantage. This is potentially damaging to a game because a perception of cheating can result in the player feeling less immersed (Kücklich, 2008), and full-immersion is crucial in creating an entertaining game (Brown and Cairns, 2004).

The extent to which a computer-controlled player needs to be bound to the same rules that the human players abide by is largely unexplored in game AI research. Even so, full observability is an unrealistic assumption for the robotics and military domains (as stated in Section 3.2.3). Hence, here, both teams will have their view of the environment limited to a composite of the areas that can be perceived by a scout (or, in general, an *observer*). This then creates an important issue concerning the deployment and coordination of the scouts that perform the observations of the opposing units.

This chapter investigates modifying the predictive system to be partially observable, and how this affects the number and timeliness of predictions that can be made. Two methods are compared for deciding on where best to direct the attention of the observer, the first being a simple round-robin scheduler, and the second being a threat-based attention mechanism.

6.2 Attention mechanisms

Given that the teams have a limited view of the environment, the problem becomes where to allocate the limited resources to cover the most relevant areas. Therefore, a potential solution to this problem is the use of an *attention* mechanism.

The inputs to an attention mechanism can either be stimulus-driven (bottom-up) or goal-directed (top-down) (Demiris and Khadhouri, 2008). Stimulus-driven means attending to the most *salient* regions of the environment and applying a winner-takes-all strategy (Itti, 2000). In the context of an RTS game this means the areas of the map with, e.g., the highest concentration or fastest movement of units. However it is well known from human psychophysical experiments that top-down information can affect bottom-up processing (Wolfe, 1994; Treue and Trujillo, 1999).

Top-down information can be derived from other sources of knowledge about the observed actions, for example from analysing the terrain, or by knowing the player's goals. However, all of this information is not available prior to the start of the scenario so a method to dynamically infer the top-down information by predicting the player's actions is needed.

In this section goal-directed information obtained from the predictive system is incorporated with other content features to make an attention mechanism to increase the effectiveness of the observations.

6.3 Partial Observability

The architecture described so far assumes the complete state information will be available for all of the inverse models whenever it is needed. However, as mentioned in Sec. 6.1, now the access to the state is restricted, and obtaining it is costly in terms of time and resources. Therefore, it is important to only obtain the information that is required to evaluate particular inverse models. For example, a *surround* manoeuvre inverse model might require observations of enemy units only within a certain vicinity of a target.

The method for deciding which inverse models deserve being attended to can also be seen as a form of resource scheduling. One of the most basic scheduling algorithms is *round-robin* scheduling Stallings (2000), where the required units are observed in a fixed sequence. When the last unit has been observed the sequence starts again from the beginning. However, other information available in the system can be used to make a more intelligent goal-directed attention mechanism. Such information includes: utility of making the observation; cost of moving to the unit's position; reliability of existing observations; and confidence of the current prediction for the unit.

Utility The utility of an observation can be formulated by estimating the threat of the unit requiring the prediction, (which may be as simple as being based on the distance from the player's units to the requested opponent's unit, i.e., if an opponent's unit is close it poses a greater threat than one further away).

Cost It is very likely that the observer has to move to make the requested observation, so this can be taken into account with the cost being proportional to the distance from the current position of the observer to the last-known position of the unit.

Reliability The reliability of the position information relates to the variance from the last-known position of the unit being requested. The variance is initially set when the unit is first observed, and it is based on the resolution of the observer and the distance over which it performs the observation. The variance then increases linearly since the time from last observation, based on the worst-case speed of the observed unit.

Confidence The confidence of the position of a unit relates to how well an assigned model fits the previously observed trajectory. If the confidence is high then it is assumed that the predicted next position is accurate.

These factors can be combined in many ways—the desired purpose of the predictions and hence the attention mechanism affects whether, for example, attention should be focussed on units with a high-confidence prediction to gather more detailed predictions, or whether a more conservative approach should be taken to focus on units that do not have a good prediction but have a high utility.

6.3.1 Objectives

To illustrate the effects of attention, a similar scenario to that used in Section 5.3.1 where the synthetic environment is used make predictions about the movements of the opponent's units. This information is used to infer the target that they are heading towards, where the targets are static and randomly placed in the environment. The main objective is to always strive to have predictions of the opponent's units before they reach their targets, and the earlier the predictions are made the better.

A threat-based attention mechanism that optimises the observations based on these objectives is described in Section 6.4.3. The following sections describe how the system was modified to support partial observability.

6.4 Implementation

6.4.1 Observations

The observability of the opponent's units is restricted, so high-resolution sensors (observers) are required to obtain the exact positional information of the units for the purposes of making a prediction. To avoid having to perform an initial blind search for the units it is assumed that there is some belief of basic low-resolution positional information for the opposing units, but this information is insufficient to identify their type and exact trajectory. For example there could be aerial surveillance, or some other sensor network to detect the areas of the environment that units occupy. Such static sensor placement is a well-researched area (Qi, 2001), e.g., using the characteristics of the sensor and the environment to ensure coverage. It should be noted, however, that this approach is not dependent on this capability because, after the initial search, the predictive system can be used to keep track of the units.

In this implementation the observers are helicopters with a 'camera' attached to their underside, pointing straight down, therefore they have a viewing circle that depends on the altitude. The camera has a fixed resolution of 50 'pixels' along the radius, so the position of each unit is quantised to one of these 'pixels'. Therefore whilst a unit is being observed, the variance of its position depends on the altitude of the observer. If the unit is not being observed then the variance of its position grows with time, based on the maximum speed of the unit. The low-resolution observer provides positional information within an

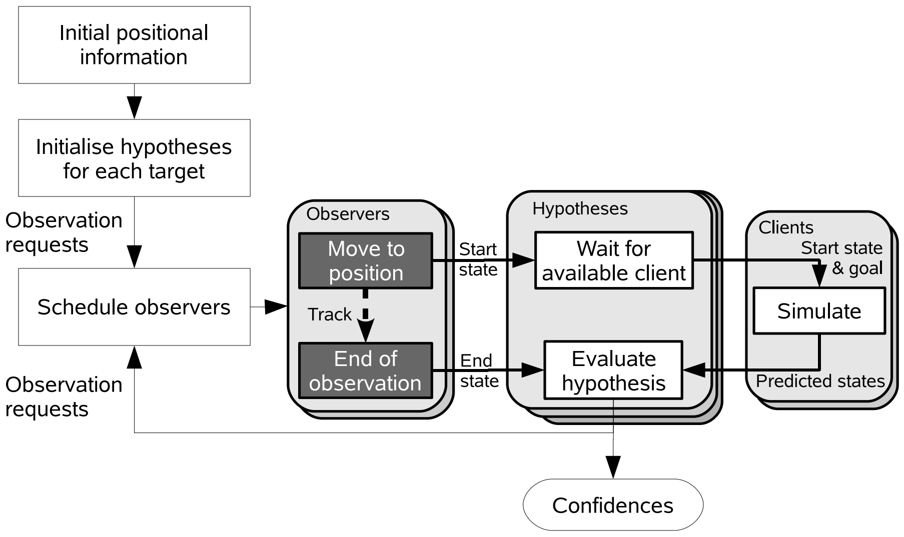


Figure 6.1: **Observation-based system operation.**

accuracy that can be covered by the viewable area of the high-resolution observer.

6.4.2 System operation

An outline of how the predictive system interacts with the observers is shown in Figure 6.1. First the initial position information for the units is gathered. The positional information for the targets (the 'blue' team) is fully observable as this is the team running the predictive system, however, the positions of the opposing units must be gathered from the available low-resolution sensors. This information is used to create the hypotheses.

The hypotheses initialise by requesting observations of their assigned target u for duration t_p. The scheduler takes these requests and decides which to attend to by assigning each observer a unit to visit (described in Section 6.4.3).

When an observer reaches the requested observation unit u it sends the state information (the position, orientation and current speed of the unit) it observes to the relevant hypothesis groups. The hypotheses each wait for a client to become available then forwards the state and the hypothesis' parameters to it. The inverse and forward models are executed for duration t_p on the client (with a speedup of s) and the predicted position of the unit u is returned. The hypothesis then waits for the observer to return a real observation of the unit u for the predictions to be compared against. The confidence for the unit is then calculated according to Equation 4.2. This confidence is then fed back to the observation scheduler, and the observer becomes idle.

Whenever an observer becomes idle, and there are pending observation requests, it is

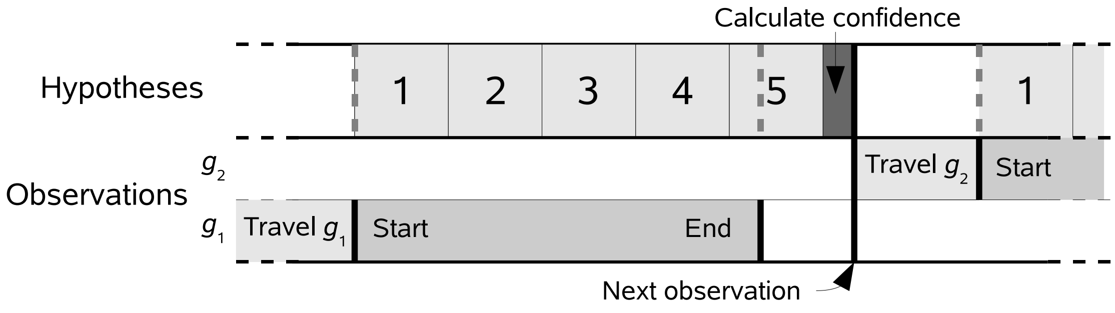


Figure 6.2: **Timing diagram of an observation**. The observer moves into position to observe g_1 and records the observed state. Then the five hypotheses in the set are simulated that require this initial state. After the specified duration the observer records the state observed state again. When the hypotheses have finished the confidence is calculated and the next position for the observer is calculated, then the loop restarts, this time with group g_2.

assigned a new observation target and the loop repeats. This loop continues until the end of the scenario. Figure 6.2 shows the timing for one complete observation duration and the start of the next observation when using one observer, one client and five hypotheses.

6.4.3 Threat-Based Attention

The scheduler chooses the next group of agents to observe from the list of requested observations. Based on the objectives listed in Section 6.3.1, a utility rating was devised that caters for the criteria below. This implementation shall be referred to as a threat-based attention mechanism.

The criteria to optimise for are:

- to have a prediction that the attacker is heading towards a target before it reaches it;
- to make predictions as early as possible.

This means the targets need to be ranked based on how close they are to their nearest target, so to have the best chance of making a prediction before the attacker reaches the target. However, the time it takes for the observer to reach the attacker to perform the observation also needs to be taken into account. Therefore the utility of the targets is ranked based on the *leeway*, which is the time difference between the observer reaching the attacker and the attacker reaching the target. It is assumed that the attacker travels at their maximum speed and in a straight line (likewise for the observer). This could be

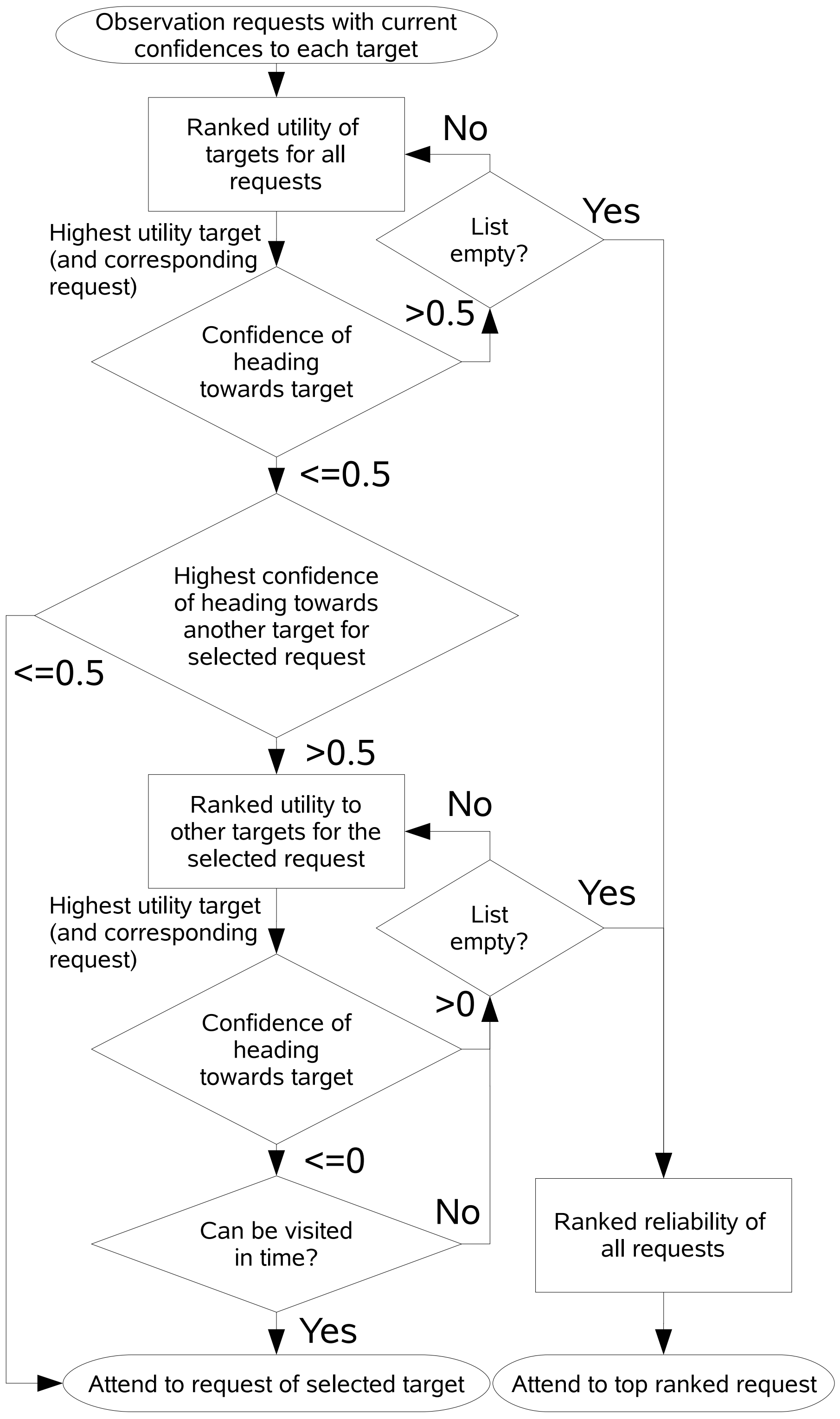

Observation requests with current confidences to each target
Ranked utility of targets for all requests
No
List empty?
Yes
Highest utility target (and corresponding request)
>0.5
Confidence of heading towards target
<=0.5
Highest confidence of heading towards another target for selected request
<=0.5
>0.5
Ranked utility to other targets for the selected request
No
List empty?
Yes
Highest utility target (and corresponding request)
>0
Confidence of heading towards target
<=0
Can be visited in time?
No
Ranked reliability of all requests
Yes
Attend to request of selected target
Attend to top ranked request

Figure 6.3: Observation request scheduling flow diagram

extended to reuse the inverse–forward model architecture, but, when deciding where to send the observer, there is only low-resolution position information available, so it would require a corresponding low-resolution forward model with simplified unit dynamics.

This leeway information is also combined with the confidence information to find out whether the attacker is already being predicted to head towards the nearest target. If it has a high confidence, until the attacker moves so that another target will have a lower leeway score, further observations (and predictions) of this attacker are not required. This means the attacker can be chosen with the next lowest leeway that doesn't have a high confidence.

With this approach the observers may end up concentrating on an attacker that is close to a target when the attacker has a high confidence of heading towards another target. This means it is harder to make predictions as early as possible for the more-likely case that the attacker doesn't change target. This can be mitigated by checking other attackers with low maximum confidences and if they have a minimum leeway target that can be observed, executed, and returned from, within the leeway time of the current attacker's target, then they should be observed.

If all of the closest leeway targets for all of the attackers have a high confidence then the reliability of the observation is used as a measure to decide which attacker to observe. The reliability is decided based on the time since the last observation. When this measure is used for all of the observations, it effectively becomes the same as a round-robin attention mechanism.

It should be noted that the cost of making an observation is only indirectly taken into consideration by the leeway calculation. If the observers had, e.g., limited fuel or needed to avoid certain regions of the map, then this approach would need to be extended to schedule more than one observation into the future.

The exact algorithm is shown in Figure 6.3.

6.5 Experiments

A scenario was set up to test the effects of partial-observability in the synthetic environment. The scenario takes place within a large open environment with arbitrary terrain (i.e., the terrain does not affect the ability for the units to reach their target). There are

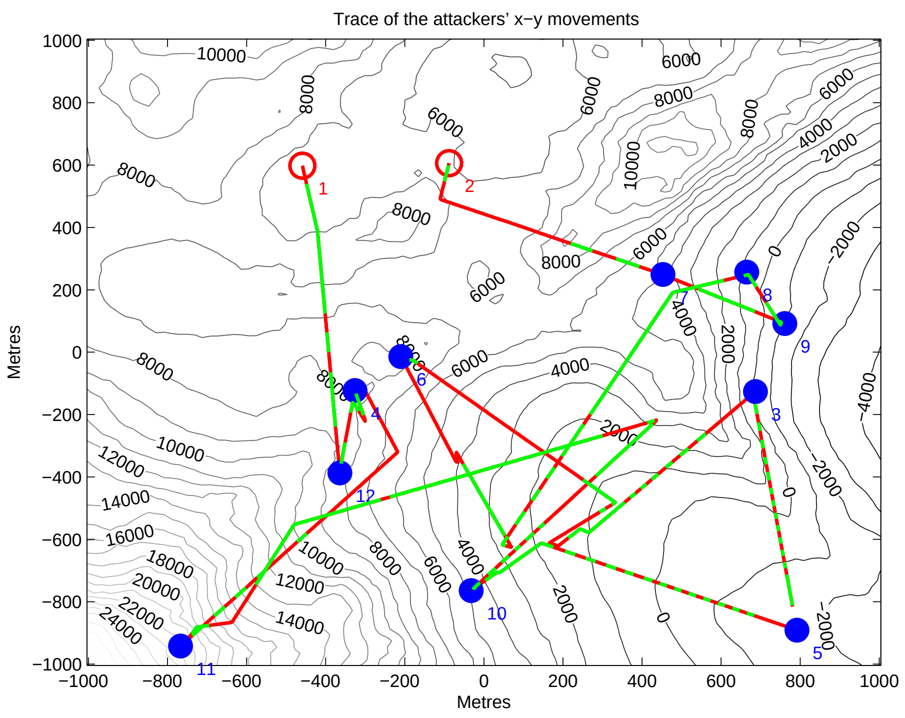
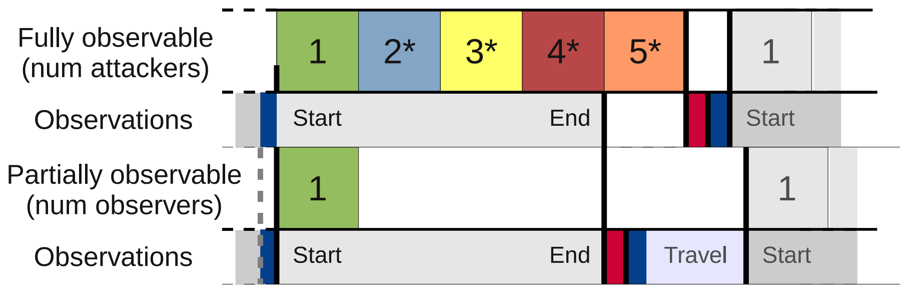


Figure 6.4: **Example scenario for two attackers (open circles) and ten targets (filled circles).** The movements of the attackers are shown by the thick lines. The targets do not move throughout the scenario. A contour map of the terrain is shown in the background. The highlighted green sections show when the helicopter is observing the attacking unit.

two teams—the 'red' and 'blue' teams. The blue team acts as a series of static targets for the attacking red team. The red player is simulated by a few simple rules: the attackers make a random choice between the nearest three blue targets and each will head straight towards their chosen target. The attackers have a maximum speed of 8m/s and they destroy their target when they reach within 10m of it, at which point they randomly select a new target among the nearest 3 targets. Each trial runs until all of the targets are dead, or for 30 minutes, whichever occurs first.

The difference in this experimental setup from Chapter 5 is that the attackers also have a 1 in 10000 chance of re-evaluating their target choice on every frame, and the engine operates at approximately 30 frames per second, giving an impromptu target change approximately every 5 minutes. This means that even if an attacker is predicted to go to a certain target, it may switch target before it reaches the original target. This is important as the attention mechanism cannot ignore threatening attackers, even if they have a strong confidence of heading elsewhere.

The other modification is that the attackers are randomly positioned at least 300m



Fully observable (num attackers)
1
2*
3*
4*
5*
1
Observations
Start
End
Start
Partially observable (num observers)
1
1
Observations
Start
End
Travel
Start

Figure 6.5: **Timing diagram showing full observability vs. partial observability with up to five attackers and one observer respectively.** The experiment is run starting with 5 attackers, until none are left, which means for full observability the number of parameters go from 5 to 1, whereas with partial observability the number of parameters stays constant at the number of observers (*optional). Note that the travel time is illustrative only, its duration changes depending on the target.

away from each other and their targets. This makes the scenario more realistic as opposing units are unlikely to start near each other, and this allows a reasonable amount of time to perform an observation before the first target can be acquired. An example trace of a scenario is shown in Figure G.11.

6.6 Results

6.6.1 Comparison between full and partial observability

The first set of runs shows the difference between full and partial observability. Full observability was approximated by using the same number of observers as attackers. Ten trials were averaged for full observability with 1–5 attackers, and another ten trials were performed for partial observability with 2–5 attackers and one observer, with half of the trials using the threat-based attention and the other half using the round-robin attention mechanism and the result being averaged. The timings for both full and partial observability are shown in Figure 6.5.

The success of a trial was measured by the proportion of times a correct prediction was made before an attacker reached a target. As can be seen from the results in Figure 6.6, partial observability doesn't reduce the success rate—both full and partial observability get around 95–100% of the target predictions correct before the target is reached. The main reason for a missed prediction is due to the attacker being able to change target at any time (albeit with a low probability), and this is applicable to both full and partial observability. This means the attacker could be heading to a target but then change and

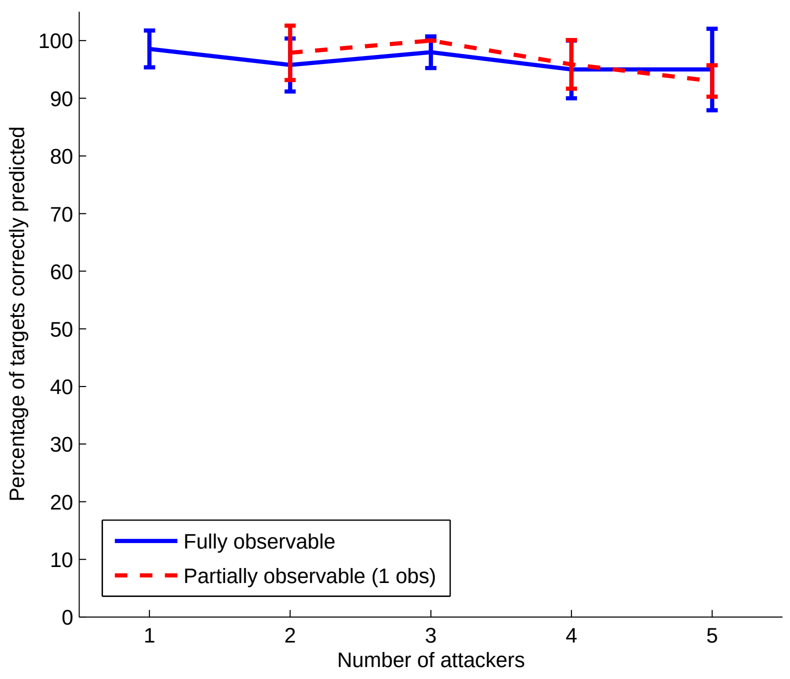


Figure 6.6: No reduction on the average percentage of targets correctly predicted between full and partial observability for small numbers of attackers.

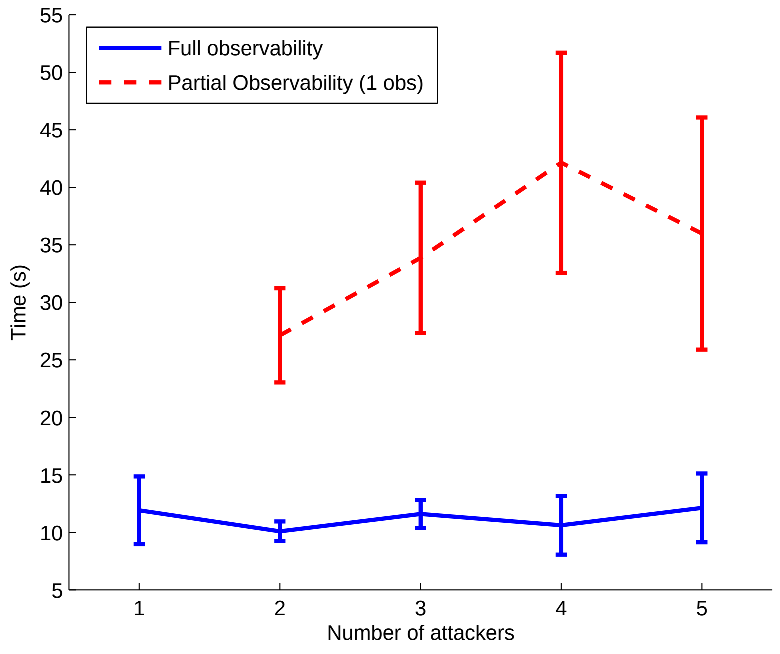


Figure 6.7: Average time difference between an attacker changing target and getting a correct prediction is greatly increased for partial observability over full observability for different numbers of attackers.

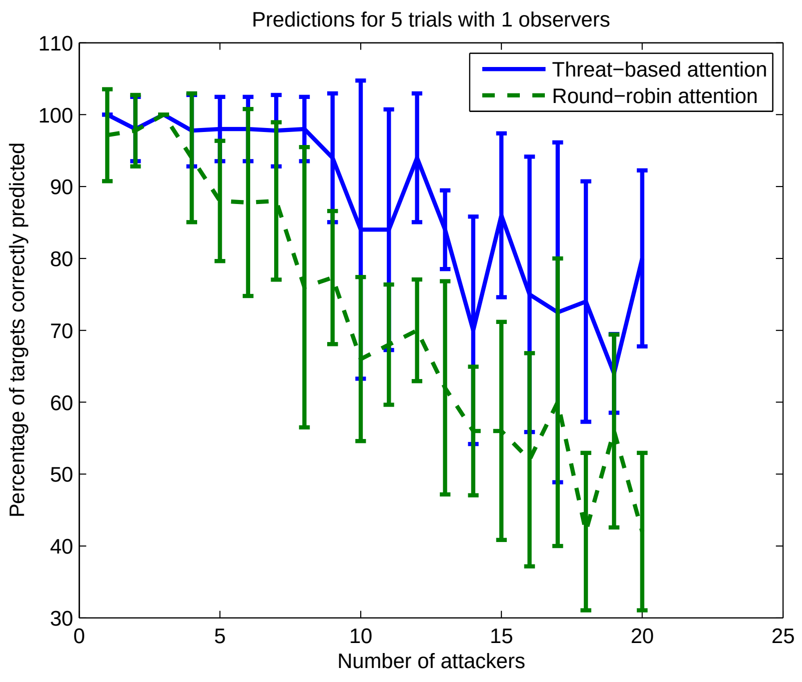


Figure 6.8: The average percentage of targets correctly predicted with one observer, ten targets and different numbers of attackers is significantly better for the threat-based attention over the round-robin scheduler.

turn to a very close-by target and there not be enough time to make a prediction before the target is reached.

As can be seen in Figure 6.7, partial observability does, however, have a large effect on the latency between an attacker changing target and receiving an observation and hence a prediction. As one would expect, full observability averages around the 10 second mark, because predictions take between approximately 5 and 12.5 seconds to complete, depending on the number of targets left. However, partial observability means that there can be a relatively long time before an observation is made if the attacker is non-threatening (using threat-based attention) or if it changes target just after being observed and has to wait for all the other attackers to be visited before getting another observation (using round-robin attention).

6.6.2 Comparison of attention mechanisms

The second set of runs show the effect of using different attention mechanisms on the percentage of targets that are correctly predicted for different numbers of attackers. Five trials were averaged for both the threat-based and the round-robin attention mechanisms,

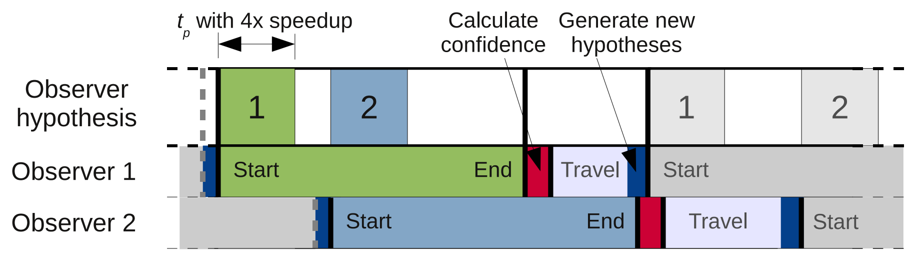


Figure 6.9: **Timing diagram showing two observers using one client.** The travel times are illustrative, so the gap between the two hypotheses being simulated will vary. If the start of one observation overlaps the prediction of another, then the next prediction is queued until the current prediction has finished being simulated.

Threat-based	Round robin
0.9897	0.9841

Table 6.1: Correlation coefficient

for 1–20 attackers. As can be seen from Figure 6.8, there is no difference between the attention mechanisms from 1–3 attackers, however, subsequently, the round-robin begins to perform worse and drops down to only 60% of correct predictions when there are 10 attackers. The threat-based attention performs significantly better up until 8 attackers, after which it begins to decline at roughly the same rate as the round-robin attention. This shows that the threat-based attention can help reduce the negative effects of only using one observer, but it has its limits and when the number of attackers becomes too great then another observer is required to keep the performance up.

Further trials were performed with two and three observers, with up to 40 and 60 attackers respectively, to match the same maximum of 20 attackers per observer as the first trial. An illustration of the possible timings with two observers is shown in Figure 6.9. If two observers start an observation at the same time, then, because there is only one client, one of the simulations will be queued until the client has finished the other prediction. Therefore, whilst there are four or fewer observers, the predictions will always be ready when the observation time period t_p ends.

The graphs (Figures 6.10 and 6.11) show similar trends for each of the trials, with round-robin attention decreasing linearly with increasing numbers of attackers, whilst the threat-based attention sees less of a decrease—the percentage of correct predictions is still around 90% with the most numbers of attackers. The similar trends between different numbers of observers can be seen visually from Figures 6.12 and 6.13, where the x-axis has been normalised to the same number of attackers per observer.

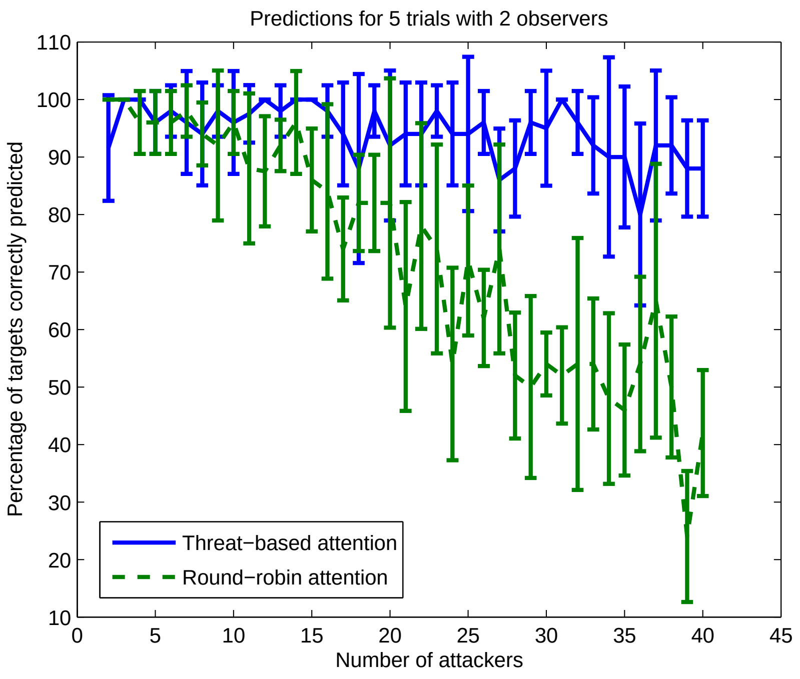


Figure 6.10: The average percentage of targets correctly predicted with 2 observers, ten targets and different numbers of attackers.

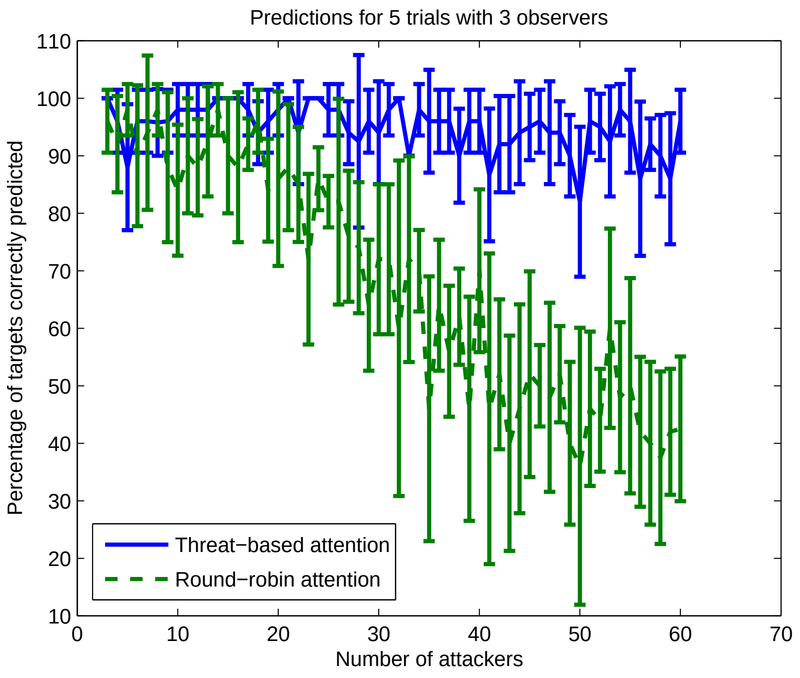


Figure 6.11: The average percentage of targets correctly predicted with 3 observers, ten targets and different numbers of attackers.

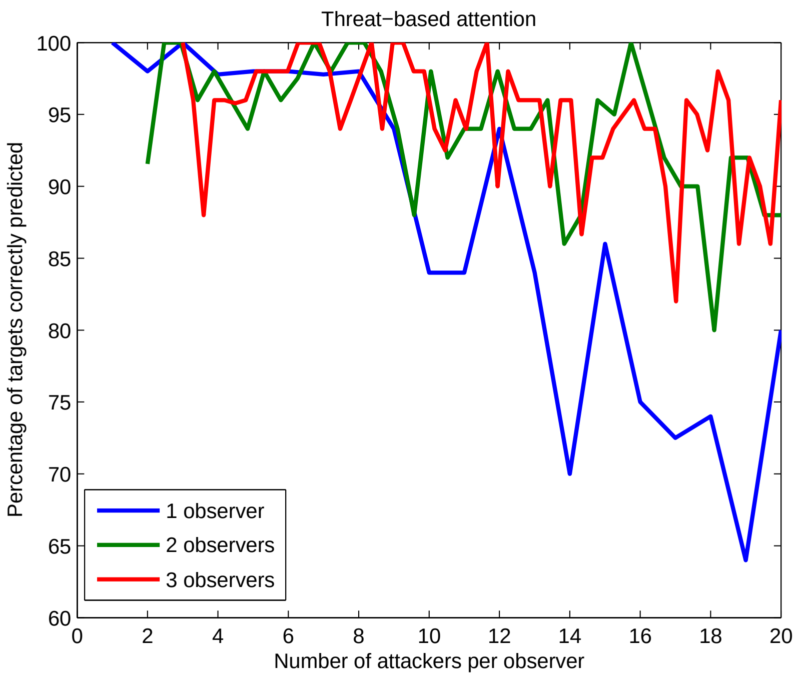


Figure 6.12: The average percentage of targets correctly predicted with 1–3 observers, ten targets and 1–20 attackers per observer using threat-based attention.

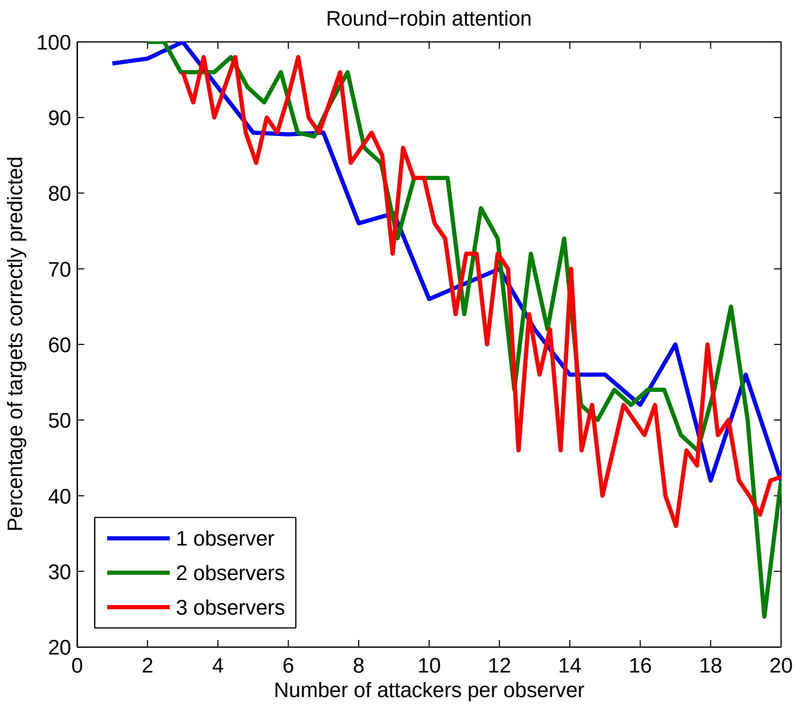


Figure 6.13: The average percentage of targets correctly predicted with 1–3 observers, ten targets and 1–20 attackers per observer using round-robin attention.

To confirm the trend, the correlation coefficient R^2 (see Equation 6.1) was calculated for each pair of trials and the mean of for each type of attention mechanism shown in Table 6.1. The three different numbers of observers a high likelihood of being correlated in both cases.

$$R^2 = \frac{\sum_{i=1}^{N}(\mathbf{m}_i - \bar{\mathbf{m}}) \cdot (\mathbf{m}'_i - \bar{\mathbf{m}}')}{\sqrt{(\sum_{i=1}^{N}(\mathbf{m}_i - \bar{\mathbf{m}})^2)(\sum_{i=1}^{N}(\mathbf{m}'_i - \bar{\mathbf{m}}')^2)}} \tag{6.1}$$

where $\bar{\mathbf{m}}$ denotes the arithmetic mean of $\mathbf{m}_i$. N is the maximum of the number of elements in $\mathbf{m}$ or $\mathbf{m}'$. As N is not equal for each of the trials, the smaller trial is linearly interpolated to match the number of elements in the longer trial.

6.7 Discussion

The results show that the quality of the predictions is maintained even when there is only one observer, and it is only the latency that is negatively affected. This means that the predictive system becomes much more efficient because it effectively enforces a priority mechanism so that only the most threatening attackers receive predictions. Therefore far fewer inverse and forward models need to be executed to get similar performance to the fully-observable case when there are 8 or fewer attackers.

Furthermore, the results show that, when using one observer, the threat-based attention mechanism effectively utilises the top-down information supplied by the predictive system to achieve significant performance gains over the round-robin attention mechanism.

When more than one observer is available the observations become more frequent, therefore the difference between the attention mechanisms became less appreciable, when using up to 10 attackers. This means that adding an observer does not significantly affect the performance of the other observers.

6.8 Summary

In conclusion, the predictive architecture has been extended to be partially observable, hence observers are used to gather positional information about the opponent. It has been shown how, in this scenario, partial observability only marginally affects the predictions made, when there are only a small number of units needing observations, whilst greatly reducing the computational cost of concurrently executing many internal models. When the number of attackers increases the effectiveness of the predictions decreases when restricted to use just one observer, however, this is significantly helped by using the threat-based attention mechanism.

The two main advantages of enforcing partial observability when making predictions for a computer-controlled opponent are:

- it results in a system that is more equal with the human-player's capabilities;
- it reduces the need to execute many internal models in parallel, greatly reducing the computational cost of the predictive system.

Chapter 7

Conclusions

7.1 Overview of the Thesis and its Contribution

This thesis set out to research the *simulation theory of intent prediction* in an adversarial multi-agent system, through operationalising and improving HAMMER—a single-agent simulation theoretical architecture. The aim of the research was to provide a principled architecture to understand the actions of a team of agents, so that it can be applied to making more challenging game AI, better performing multi-robot teams, and faster decision support for military scenarios.

This aim was achieved by first identifying the challenges of operationalising the HAMMER architecture in a complex multi-agent synthetic environment, and then by creating an architecture that allowed the manoeuvres and formations of an opposing team to be recognised by generating hypotheses and simulating them. This approach has never before been applied to multi-agent scenarios and the work was published in Butler and Demiris (2009). Significant effort was undertaken to produce a realistic synthetic environment for use as a test-bed for the system, and to provide a distributed system for simulating hypotheses across multiple hosts. The results show the merit of the system, however they also illustrate that the approach is quite computationally complex. Furthermore, the inverse models that were used were fairly limited, which reduces the scope for recognising a wide space of actions. Therefore it was important to increase the range of actions that could be performed whilst keeping the computational load to a minimum.

To address these issues, two directions were explored. First, methods to expand the action-space were investigated. This involved creating parameterised inverse models and finding ways to efficiently search the parameter-space. Two novel methods were implemented: a way to iteratively target the most relevant parameters, and a clustering method to find the least parameters that provide a good coverage of the space. Experiments show that both methods have merits, but the clustering method provided the best reduction in computational cost for a given prediction accuracy. The clustering approach was published in Butler and Demiris (2010a).

The second research focus was to introduce partial observability. This requires the use of an attention mechanism to force resources to be focussed on the most pertinent areas. A novel threat-based attention mechanism was developed for this purpose, and shows a good improvement over a static round-robin scheduler. Additionally, and more importantly, it shows that, when using the attention mechanism, the number of hypothesis groups that can be evaluated are limited to the number of observers, yet this does not greatly adversely affect the quality of the results. It compromises a slight increase in latency with a good saving in computational cost. This work was published in Butler and Demiris (2010b).

Additionally, the simulator and the framework for simulating hypotheses in a distributed manner, plus a toolkit containing the approaches discussed in this thesis for reducing the number of required hypotheses, is released as open-source software, and is available from `http://simon-butler.com/Thesis`.

The predictive platform as a whole is attractive because it uses the same models for both perceiving and acting, therefore it has the scope to be useful as a basis for game or robotic AI systems that need to anticipate the opponent's behaviours and to use this information to execute its own behaviours. It also has the added benefit that these models are likely to be already available as a library of actions the agent can perform, and, although the work shown here uses only a few simple inverse models, the same architecture can be applied to any action the agents can perform. However, the system has its limitations: accuracy or computational load can be adversely affected if behavioural models of the opponent are significantly different from those the predicting team use or if there is a vast array of possible actions an agent can perform at any one time.

7.2 Future Directions

There are several directions in which this research can take in the future. The work in this thesis illustrates that this approach is capable of predicting intent, however, there are further ways to reduce the complexity of simulating possible hypotheses. Another area that provides scope for interesting research is the question of deception and the additional levels of reasoning required when the opponent is trying to conceal their behaviour. Finally, the main scope for using this approach in real-world systems is to utilise the predictions to autonomously adapt behaviour. These areas of further research are discussed in the sub-sections below.

7.2.1 Optimisations

Whilst the partial observability reduces the computational load, further optimisations are needed within the forward model (such as optimising the trade-off between accuracy and speed in the physics engine) for this approach to become viable using current hardware, for instance, in a commercial game. However with the increasing number of cores available on upcoming hardware and the potential for GPGPU acceleration, this may be less of an issue in the future.

The threat-based attention mechanism could also be extended to make a more efficient assignment of observations by scheduling more than one observation into the future. This could be done by using a scaled-down version of the forward model to quickly generate predictions that could be fed into a travelling-salesman-style algorithm to compute the most efficient route.

7.2.2 Deception

This thesis makes no attempt to detect deception. However, if the opponent is aware that observations of the movements of their agents are being made, they may make an attempt to disguise their true intentions. An interesting extension to this work is to add a level of adversarial reasoning to the system, so that attempts to mislead an observer could be recognised.

7.2.3 Autonomous Decision Making

A particularly interesting use of this research is to autonomously make decisions based on the predicted intent of the opponent. Running the highest confidence hypothesis into the future to get a estimate of the movements of the opponent can be useful to make decisions about the actions the predicting team should perform. This would be particularly useful in the games or robotics domains where the same models can be used for both controlling their own team and predicting the opponents' actions.

7.3 Epilogue

The work presented here has shown how a specific theory about how the human mind operates can be utilised to develop a complex computational architecture in a challenging domain. This was done without losing the many advantages of the principled approach, such as scalability and robustness. Attempting to understand the intentions of a team of agents purely from observations is a difficult task, but worthwhile, since it underpins an ability to anticipate and perform intelligent responses to the opponents actions.

References

Avrahami-Zilberbrand, D. and Kaminka, G. A., "Hybrid symbolic-probabilistic plan recognizer: Initial steps," in *Proceedings of the AAAI Workshop on Modeling Others from Observations (MOO 2006)*, 2006.

Bach, F. R. and Jordan, M. I., "Learning spectral clustering, with application to speech separation," *Journal of Machine Learning Research*, vol. 7, pp. 1963–2001, 2006.

Baron-Cohen, S., *Mindblindness: An essay on autism and theory of mind.* The MIT Press, 1997.

Beetz, M. and Kirchlechner, B., "Computerized real-time analysis of football games," *IEEE pervasive computing*, vol. 4, no. 3, 2005.

Blakemore, S.-J. and Decety, J., "From the perception of action to the understanding of intention," *Nature Reviews Neuroscience*, vol. 2, no. 8, pp. 561–567, August 2001.

Blaylock, N. and Allen, J., "Fast hierarchical goal schema recognition," *Proceedings of AAAI*, vol. 6, 2006.

Bratman, M. E., "What is intention?" *Intentions in communication*, pp. 15–31, 1990.

Brown, E. and Cairns, P., "A grounded investigation of game immersion," in *Extended abstracts on Human factors in computing systems*, 2004, pp. 1297–1300.

Bui, H., Venkatesh, S., and West, G., "Policy recognition in the abstract hidden markov models," *Journal of Artificial Intelligence Research*, vol. 17, pp. 451–499, 2002.

Butler, S. and Demiris, Y., "Predicting the movements of robot teams using generative models," in *Distributed Autonomous Robotic Systems 8.* Springer, 2009, pp. 533–542.

Butler, S. and Demiris, Y., “Using a cognitive architecture for opponent target selection,” in *Proceedings of the Third International Symposium on AI and Games.* SSAISB, 2010, pp. 55–61.

Butler, S. and Demiris, Y., “Partial observability during predictions of the opponents movements in an RTS game,” in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games.* IEEE, August 2010, pp. 46–53.

Cohen, P. R. and Levesque, H. J., “Intention is choice with commitment,” *Artificial intelligence*, vol. 42, no. 2-3, pp. 213–261, 1990.

Csibra, G. and Gergely, G., “‘Obsessed with goals’: Functions and mechanisms of teleological interpretation of actions in humans,” *Acta Psychologica*, vol. 124, no. 1, pp. 60–78, January 2007.

Darken, R., Mcdowell, P., and Johnson, E., “The Delta3D open source game engine,” *IEEE computer graphics and applications*, vol. 25, no. 3, 2005.

Dearden, A. and Demiris, Y., “Learning forward models for robots,” in *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 2005, pp. 1440–1445.

Demiris, Y., “Prediction of intent in robotics and multi-agent systems,” *Cognitive Processing*, vol. 8, no. 3, pp. 151–158, September 2007.

Demiris, Y. and Khadhouri, B., “Hierarchical attentive multiple models for execution and recognition of actions,” *Robotics and autonomous systems*, vol. 54, no. 5, 2006.

Demiris, Y. and Khadhouri, B., “Content-based control of goal-directed attention during human action perception,” *Interaction Studies*, vol. 9, no. 2, pp. 353–376, 2008.

Devaney, M. and Ram, A., “Needles in a haystack: Plan recognition in large spatial domains involving multiple agents,” in *National Conference on Artificial Intelligence*, 1998.

Franklin, S. and Graesser, A., “Is it an agent, or just a program?: A taxonomy for autonomous agents,” in *Intelligent Agents III Agent Theories, Architectures, and Languages*, ser. Lecture Notes in Computer Science, Mller, J., Wooldridge, M., and Jennings, N., Eds. Springer Berlin / Heidelberg, 1997, vol. 1193, pp. 21–35.

Gallese, V. and Goldman, A., "Mirror neurons and the simulation theory of mind-reading," *Trends in cognitive sciences*, vol. 2, no. 12, pp. 493–501, 1998.

Gopnik, A. and Blackwell, B., "The theory theory as an alternative to the innateness hypothesis," *Chomsky and his critics*, pp. 238–254, 2003.

Gopnik, A., Glymour, C., Sobel, D., Schulz, L., Kushnir, T., and Danks, D., "A theory of causal learning in children: Causal maps and Bayes nets." *Psychological Review*, vol. 111, no. 1, pp. 3–32, 2004.

Gordon, R., "Simulation vs theory-theory," *The MIT Encyclopædia of the Cognitive Sciences*, pp. 765–766, 1999.

Ham, J., Ahn, I., and Lee, D., "Learning a manifold-constrained map between image sets: applications to matching and pose estimation," in *CVPR06*, 2006.

Hart, P. E., Nilsson, N. J., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, July 1968.

Heinze, C., Goss, S., and Pearce, A., "Plan recognition in military simulation: Incorporating machine learning with intelligent agents," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI99) Agent Workshop on Team Behaviour and Plan Recognition*, 1999, pp. 53–63.

Hesslow, G., "Conscious thought as simulation of behaviour and perception," *Trends in Cognitive Sciences*, vol. 6, no. 6, pp. 242–247, June 2002.

Hommel, B., Müsseler, J., Aschersleben, G., and Prinz, W., "The theory of event coding (TEC): A framework for perception and action planning," *Behavioral and Brain Sciences*, vol. 24, no. 05, pp. 849–878, 2001.

Intille, S. S. and Bobick, A. F., "A framework for recognizing multi-agent action from visual evidence," in *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence.* Menlo Park, CA, USA: American Association for Artificial Intelligence, 1999, pp. 518–525.

Itti, L., "A saliency-based search mechanism for overt and covert shifts of visual attention," *Vision Research*, vol. 40, no. 10-12, pp. 1489–1506, June 2000.

Ji and Egerstedt, M., "Distributed coordination control of multiagent systems while preserving connectedness," *IEEE Transactions on Robotics*, vol. 23, no. 4, pp. 693–703, 2007.

Johansson, S., "The spring project," 2010. [Online]. Available: http://spring.clan-sy.com

Johnson, M. and Demiris, Y., "Perceptual perspective taking and action recognition," *International Journal of Advanced Robotic Systems*, vol. 2, no. 4, pp. 301–308, 2005.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawai, E., and Matsubara, H., "Robocup: A challenge problem for AI and robotics," *RoboCup-97: Robot Soccer World Cup I*, pp. 1–19, 1998.

Kücklich, J., "Forbidden pleasures: Cheating in computer games," in *The pleasures of computer gaming: Essays on cultural history, theory and aesthetics*, Swalwell, M. and Wilson, J., Eds. McFarland, 2008, pp. 52–71.

Lafon, S., Keller, Y., and Coifman, R. R., "Data fusion and multicue data matching by diffusion maps," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 28, no. 11, pp. 1784–1797, 2006.

Leslie, A., Friedman, O., and German, T., "Core mechanisms in "theory of mind"," *Trends in Cognitive Sciences*, vol. 8, no. 12, pp. 528–533, December 2004.

Leslie, A., German, T., and Polizzi, P., "Belief-desire reasoning as a process of selection," *Cognitive Psychology*, vol. 50, no. 1, pp. 45–85, February 2005.

Leslie, A., "Pretense and representation: The origins of "theory of mind."," *Psychological review*, vol. 94, no. 4, pp. 412–426, 1987.

Liu, X. and Chua, C., "Multi-agent activity recognition using observation decomposed-hidden markov models," *Image and Vision Computing*, vol. 24, no. 2, pp. 166–175, February 2006.

Luotsinen, L. J. and Bölöni, L., "Role-based teamwork activity recognition in observations of embodied agent actions," in *AAMAS '08: Proceedings of the 7th international joint* *conference on Autonomous agents and multiagent systems.* Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 567–574.

Mathews, J. H. and Fink, K. K., *Numerical Methods Using Matlab (4th Edition)*, 4th ed. Prentice Hall, January 2004.

Michlmayr, M., "Simulation theory versus theory theory: Theories concerning the ability to read minds," Master's thesis, University of Innsbruck, Mar 2002.

Murphy, K. and Paskin, M., "Linear time inference in hierarchical HMMs," in *Advances in neural information processing systems 14: proceedings of the 2001 conference.* MIT Press, 2002, p. 833.

Ng, A., Jordan, M., and Weiss, Y., "On spectral clustering: Analysis and an algorithm," *Advances in Neural Information Processing Systems*, vol. 14, 2002.

Nichols, S. and Stich, S., *Mindreading: An integrated account of pretence, self-awareness, and understanding other minds.* Oxford University Press, USA, 2003.

Pearce, A. R., Caelli, T., and Goss, S., "On learning spatio-temporal relational structures in two different domains," in *Lecture Notes in Artificial Intelligence.* Springer, 1998, vol. 1352, no. 2, pp. 551–558.

Premack, D. and Woodruff, G., "Does the chimpanzee have a theory of mind?" *Behavioral and Brain Sciences*, vol. 1, no. 04, pp. 515–526, 1978.

Qi, H., "Distributed sensor networks—a review of recent research," *Journal of the Franklin Institute*, vol. 338, no. 6, pp. 655–668, September 2001.

Saria, S. and Mahadevan, S., "Probabilistic plan recognition in multiagent systems," in *Proceedings of International Conference on AI and Planning Systems*, 2004.

Shi, J. and Malik, J., "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

Soon, S., Pearce, A., and Noble, M., "Adaptive teamwork coordination using graph matching over hierarchical intentional structures," in *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004, pp. 294–301.

Stallings, W., *Operating Systems: Internals and Design Principles.* Prentice-Hall, 2000.

Sukthankar, G. and Sycara, K., "Robust recognition of physical team behaviors using spatio-temporal models," in *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems.* New York, NY, USA: ACM, 2006, pp. 638–645.

Takács, B., Butler, S., and Demiris, Y., "Multi-agent behaviour segmentation via spectral clustering," in *Proceedings of the AAAI-2007, PAIR Workshop.* AAAI Press, July 2007, pp. 74–81.

Tambe, M., "Tracking dynamic team activity," in *National Conference on Artificial Intelligence(AAAI96)*, 1996.

Tomasello, M., Carpenter, M., Call, J., Behne, T., and Moll, H., "Understanding and sharing intentions: The origins of cultural cognition," *Behavioral and Brain Sciences*, vol. 28, no. 05, pp. 675–691, October 2005.

Treue, S. and Trujillo, J. C., "Feature-based attention influences motion processing gain in macaque visual cortex," *Nature*, vol. 399, no. 6736, pp. 575–579, June 1999.

Vail, D. L., Veloso, M. M., and Lafferty, J. D., "Conditional random fields for activity recognition," in *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems.* New York, NY, USA: ACM, 2007, pp. 1–8.

Wellman, H., "Understanding the psychological world: Developing a theory of mind," *Blackwell handbook of childhood cognitive development*, pp. 167–187, 2004.

White, B., Blaylock, N., and Bölöni, L., "Analyzing Team Actions with Cascading HMM," in *Proceedings of the Twenty-Second International FLAIRS Conference*, 2009, pp. 129–134.

Wolfe, J., "Visual search in continuous, naturalistic stimuli," *Vision Research*, vol. 34, no. 9, pp. 1187–1195, May 1994.

Wolpert, D. M., Doya, K., and Kawato, M., "A unifying computational framework for motor control and social interaction," *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 358, no. 1431, pp. 593–602, March 2003.

Zelnik-Manor, L. and Perona, P., "Self-tuning spectral clustering," in *Eighteenth Annual Conference on Neural Information Processing Systems, (NIPS)*, 2004.

Appendix A

Publications

Work presented in this thesis has been published in the following papers:

- Partial Observability During Predictions of the Opponent's Movements in an RTS Game, S.Butler and Y.Demiris, in Proceedings of IEEE Conference on Computational Intelligence and Games, August 2010.
- Using a Cognitive Architecture for Opponent Target Prediction, S. Butler and Y. Demiris, in Proceedings of the Third International Symposium on AI & Games, pp. 55-61, SSAISB, April 2010.
- Predicting the movements of robot teams using generative models, S. Butler and Y. Demiris, in Distributed Autonomous Robotic Systems 8, pp. 533-542, Springer, May 2009.
- Multi-Agent Behaviour Segmentation via Spectral Clustering, B. Takacs, S. Butler and Y. Demiris, in Proceedings of the AAAI-2007 Workshop on Plan, Activity and Intention Recognition (PAIR), pp. 74-81, AAAI Press, July 2007.

Appendix B

Synthetic Environment

The synthetic environment developed for this thesis is a large piece of software containing approximately 30,000 lines of C++ code. It is dependent on the following open-source libraries: Delta3D, Qt, Boost, OpenSceneGraph, GNE/HawkNL, CEGUI, GSL, Eigen2. The code can be downloaded from `http://simon-butler.com/Thesis`.

To get an idea of the overall structure of the program, the collaboration diagram from the main Application class is shown in Figure B.1. Within this Application class there are two functions that are the most complex, the initialisation method and the preframe method. The functions that are called from these methods are shown in Figures B.2 and B.3 respectively.

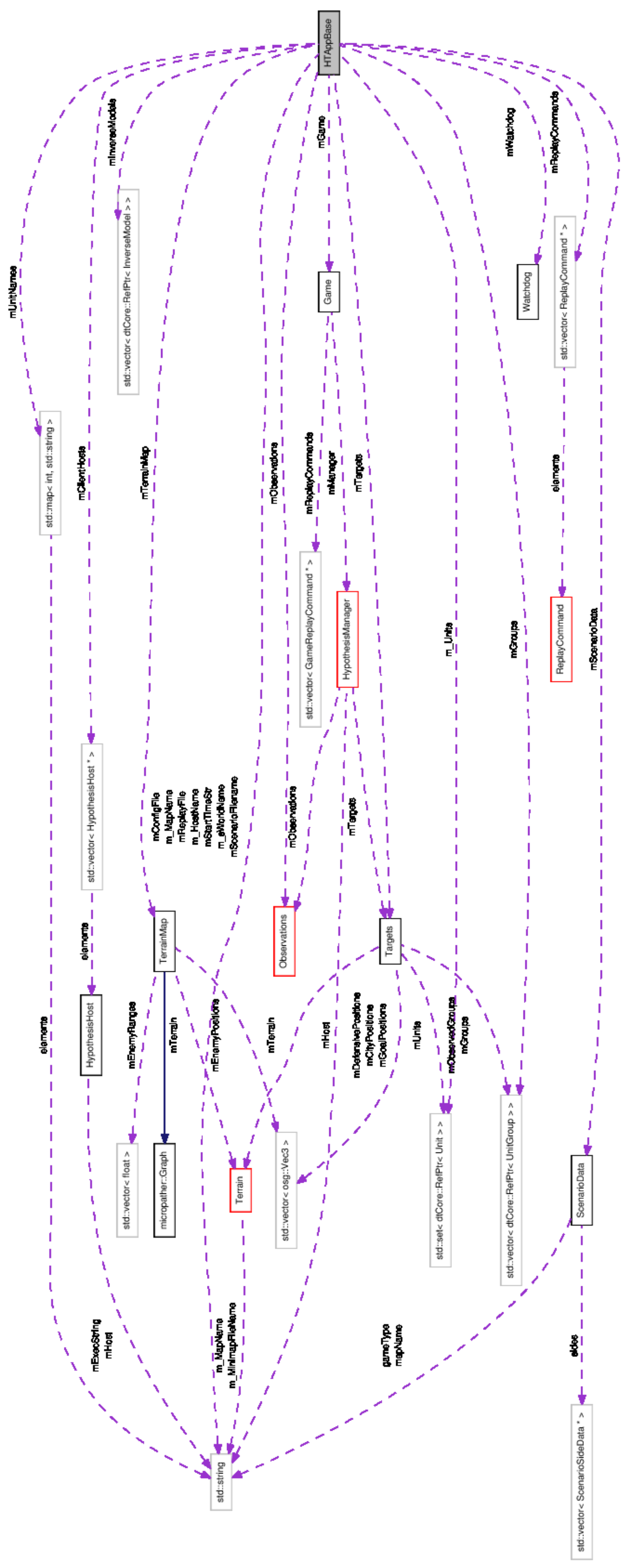


Figure B.1: A collaboration diagram showing the relation of the classes to the base Application class.

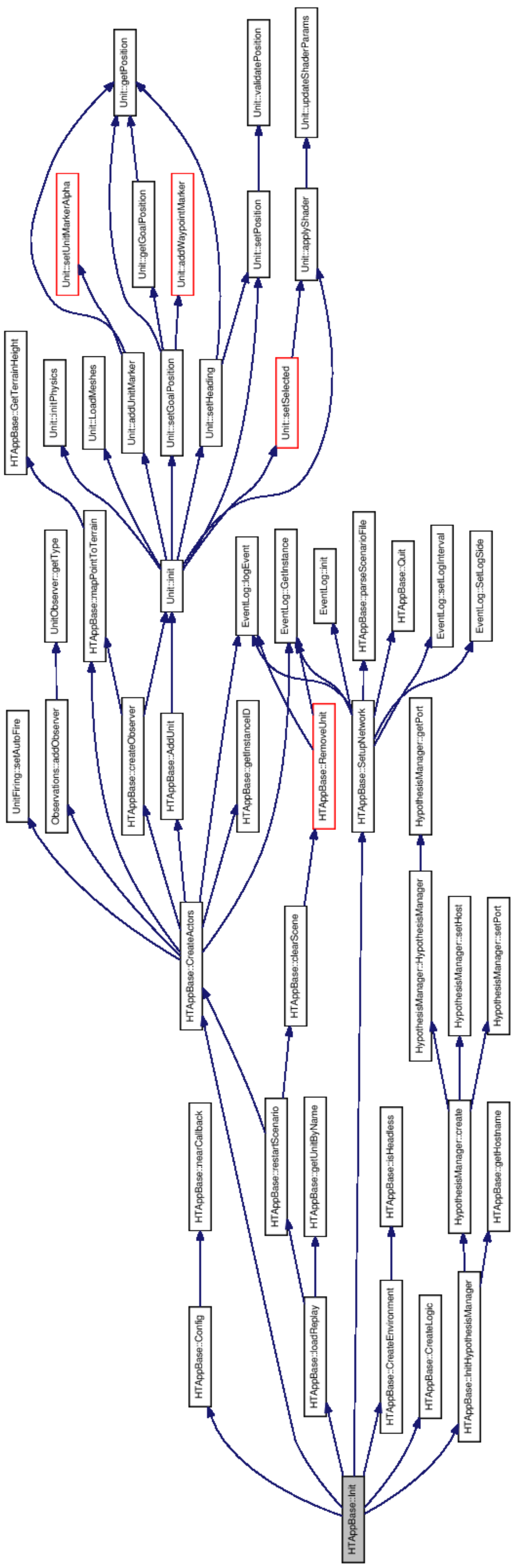


Figure B.2: The functions that are called from the initialisation method of the Application class.

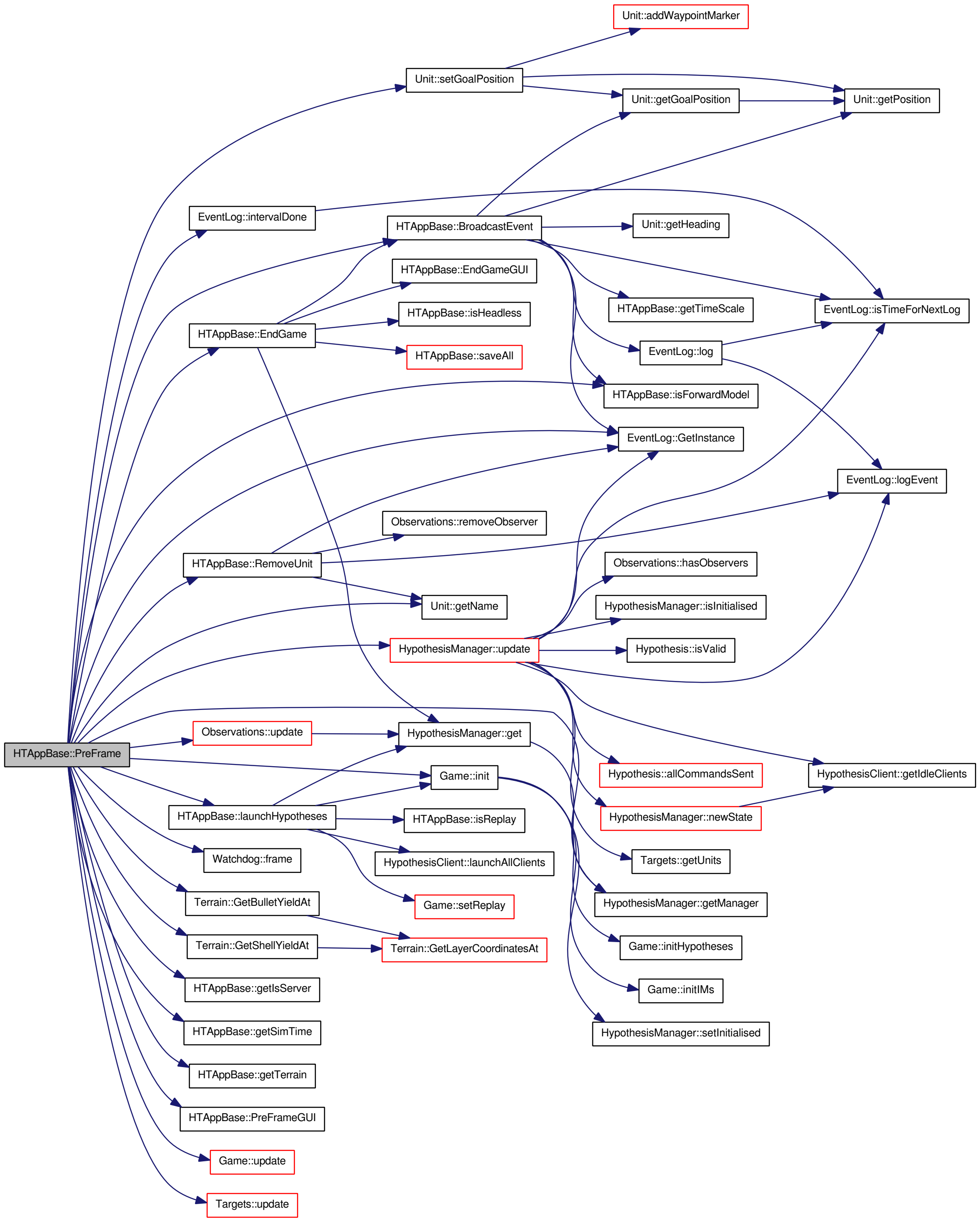


Figure B.3: The functions that are called from the preframe method of the Application class.

Appendix C

Ontology

This chapter defines the ontology required to communicate with the forward model. All of these properties are serialised and sent over the network to the clients for each of the relevant units when starting an hypothesis, and the dynamic properties are received from the clients when an hypothesis finishes. The static properties are read in from the configuration files at the start of the scenario, but still need serialising and sending to the client, as the configuration files are only available on the server-side.

C.1 Unit properties

C.1.1 Dynamic properties

Position The current (x,y,z) position of the unit.

Heading $\in \mathbb{R} : 0 \leq x < 360$. The current heading (in degrees) of the unit.

Velocity The current velocity vector (x,y,z) of the unit.

Health $\in \mathbb{R} : 0 \leq x \leq 100$. When the health property reaches zero, the unit is 'dead' and is removed from the scene.

C.1.2 Static properties

ID Unique identifier for the unit.

Side $\in \{\text{red}, \text{blue}, \text{none}\}$. *None* means the unit is a non–combatant. Additional meta unit sides were introduced (*all* and *active* to correspond to any, or just combative, units respectively), but these cannot be assigned directly to a unit.

Domain $\in \{\text{land}, \text{air}\}$. Waypoints for land units are constrained to always lie on the terrain.

Armour When a projectile detonates the decrease in unit health is calculated as $h = \frac{p}{x}$ where p is the damage from the projectile (depending on the distance from detonation and the terrain type), and x is the armour property.

Max Speed Speed on roads.

Average Speed Normal speed.

Slow Speed Speed in forests.

Slowing distance Distance from a waypoint to start slowing.

Stopping distance Distance to stop from a waypoint.

Turning rate This controls the turning circle of the unit.

Sensor range $r(g) = \{x \in \mathbb{R}\}$ where g is the terrain type. The maximum distance that the unit can observe other units, from within each terrain type.

Visibility $v(g) = \{x \in \mathbb{R}\}$ where g is the terrain type. The degree of visibility of the unit from within each terrain type. Whether a unit can see another is calculated as whether $|u_1 - u_2| < r\left(g(u_1)\right) v\left(g(u_2)\right)$.

C.2 UnitFiring properties

C.2.1 Dynamic properties

Reload time Once the unit has fired a projectile, the reload time is the time it takes before the unit can fire again (seconds).

C.2.2 Static properties

Projectile Type Shell or Bullet.

Projectile Mass Affects the trajectory of the projectile.

Projectile Radius Affects the collision geometry.

Projectile Velocity Launch velocity of the projectile.

Projectile Yield The yield of the projectile when it collides with the terrain or a unit. For shells, the damage p received by a unit is calculated using the inverse square law $p = \frac{gy}{d^2}$ where y is the yield of the shell, which is scaled by the yield on the terrain type g of the unit, and d is the distance between the unit receiving damage and the point of detonation. For bullets the damage is just $p = gy$ as only bullets that intersect with the bounding box of the unit inflict damage.

Projectile Range Maximum range of projectile.

C.3 UnitTank properties

C.3.1 Dynamic properties

All of these properties are required for each of the ODE bodies that make up the tank: the hull, turret, barrel, and 4 wheels.

Position The current (x,y,z) position of the ODE body.

Rotation The current (w,x,y,z) quaternion of the ODE body.

Velocity The current (x,y,z) velocity of the ODE body.

Angular velocity The current (x,y,z) angular velocity of the ODE body.

C.3.2 Static properties

Turret Threshold The acceptable angular error between actual and ideal turret position for firing.

Barrel Threshold The acceptable angular error between actual and ideal barrel position for firing.

Max Force The maximum force to apply to the joints.

Wheel X The x-position of the wheels right wheels, mirrored in the y-axis for the left wheels.

Wheel Y Front The y-position of the front wheels.

Wheel Y Rear The y-position of the rear wheels.

Wheel Z The z-position of all the wheels.

Wheel mass The mass of the wheels.

Vehicle mass The mass of the vehicle chassis. This affects the handling of the unit.

Suspension ERP The Error Reduction Parameter for the suspension. This specifies the proportion of the joint error that is fixed on each frame by applying corrective forces to bring the bodies back into alignment.

Suspension CFM The Constraint Force Mixing parameter for the suspension. This controls the proportion of the restoring force that is needed to enforce the constraint that is applied on each frame. When combined with the suspension ERP this effectively controls the spring and damper constants of the suspension.

C.4 UnitSoldier properties

C.4.1 Static properties

Turret Threshold The acceptable angular error between actual and ideal unit heading for firing.

Appendix D

Cubic Spline Interpolation

The cubic spline is a piecewise cubic polynomial (Mathews and Fink, 2004). The intervals are determined by the "knots" or abscissas of the data to be interpolated. The cubic spline has continuous first and second derivatives over the entire interval of interpolation.[1] For any point t in the interval $[t_i, t_{i+1}]$, the form of the spline is

$$S(t) = a_i + b_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3 \tag{D.1}$$

If we assume that we know the values y and y'', which represent the values and second derivatives of the spline at each knot, then the coefficients can be computed as:

$$a_i = y_i \tag{D.2}$$

$$b_i = \frac{y_{i+i} - y_i}{t_{i+1} - t_i} - \frac{(y''_{i+1} + 2y''_i)(t_{i+1} - t_i)}{6} \tag{D.3}$$

$$c_i = \frac{y''_i}{2} \tag{D.4}$$

$$d_i = \frac{y''_{i+1} - y''_i}{6(t_{i-1} - t_i)} \tag{D.5}$$

[1] `http://people.sc.fsu.edu/~burkardt/cpp_src/spline/spline.html`

Since the first derivative of the spline is

$$S'(t) = b_i + 2c_i(t - t_i) + 3d_i(t - t_i)^2 \tag{D.6}$$

the requirement that the first derivative be continuous at interior knot i results in a total of $N - 2$ equations, of the form:

$$b_{i-1} + 2c_{i-1}(t_i - t_{i-1}) + 3d_{i-1}(t_i - t_{i-1})^2 = b_i \tag{D.7}$$

or, setting $h_i = t_{i+1} - t_i$

$$\frac{y_i - y_{i-1}}{h_{i-1}} - \frac{h_{i-1}(y_i'' + 2y_{i-1}'')}{6} + y_{i-1}''h_{i-1} + \frac{h_{i-1}(y_i'' - y_{i-1}'')}{2} = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i(y_{i+1}'' + 2y_i'')}{6} \tag{D.8}$$

or

$$y_{i-1} + hi - 1 + 2y_i''(h_{i-1} + h_i) + y_i h_i = \frac{6(y_{i+1} - y_i)}{h_i} - \frac{6(y_i - y_{i-1})}{h_{i-1}} \tag{D.9}$$

Boundary conditions must be applied at the first and last knots. The resulting tridiagonal system can be solved for the y'' values.

Appendix E

Configuration Files

Listing E.1: Sample XML configuration file containing initial unit positions

```
<?xml version="1.0" encoding="UTF-8" ?>
<scenario game="capture" map="test_map" time_limit="600" goal_time="60">
  <side name="red">
    <unit type="tank" number="3" instance="0">
      <startpos x="1600" y="300" />
      <endpos x="1700" y="300" />
    </unit>
    <unit type="tank" number="1" instance="0">
      <pos x="1900" y="300" />
    </unit>
    <unit type="soldier" number="5" instance="0">
      <startpos x="2200" y="300" />
      <endpos x="2300" y="300" />
    </unit>
  </side>
  <side name="blue">
    <unit type="tank" number="6" instance="1">
      <startpos x="1500" y="-1500" />
      <endpos x="1700" y="-1500" />
    </unit>
    <unit type="soldier" number="8" instance="1">
      <startpos x="2100" y="-1400" />
      <endpos x="2300" y="-1400" />
    </unit>
  </side>
</scenario>
```

Listing E.2: Unit XML configuration file containing unit parameters for a tank

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<unit name="Tank" flying="0">
  <movement max_speed="80" average_speed="40" slow_speed="30"
      turning_rate="8" slowing_dist="20" stopping_dist="5" />
  <physics mass="100" max_force="500" suspension_erp="0.2"
      suspension_cfm="0.0001" turn_erp="0.95" turn_cfm="0.01" />
  <wheels x="1.6" y_front="2.0" y_rear="1.8" z="0.3" mass="10" />
  <firing reload="10" turret="0.05" barrel="0.01" />
  <sensors layer_id="none" range="3000" visibility="1.0" />
  <sensors layer_id="forest" range="3000" visibility="0.6" />
  <sensors layer_id="city" range="3000" visibility="0.8" />
  <projectile mass="1" radius="0.5" velocity="500" yield="1000"
      range="3000" type="shell" />
  <meshes>
    <object name="Hull" mesh="StaticMeshes/T72hull.ive" />
    <object name="Turret" mesh="StaticMeshes/T72turret.ive" />
    <object name="Barrel" mesh="StaticMeshes/T72gun.ive" />
    <object name="Wheel" mesh="StaticMeshes/wheel.flt" number="4" />
  </meshes>
</unit>
```

Listing E.3: Sample terrain XML configuration file

```
<?xml version="1.0" encoding="UTF-8" ?>
<map name="test_map"
     horizontal_size="10000"
     min_alt="0" max_alt="1000"
     segment_size="500"
     segment_divisions="32"
     minimap="test_map.tga"
     height_map="HF.png"
     attribute_map="ATTR.png"
     texture_map="TX.png">
  <texture file="Textures/riverdalegrass.jpg" id="256"/>
  <texture file="Textures/redsand1a.jpg" id="512"/>
  <texture file="Textures/rock3.jpg" id="768"/>
  <layer type="mask">
    <params name="forest" file="forest_mask.png" color="0 0 0 0.2"
      recolor="0" shell_yield="0.5" bullet_yield="1.0" />
    <obj_params mean="1" sigma="0.5" group="100"/>
    <object side_image="Textures/Trees/Bleech3.png"
      top_image="Textures/Trees/Toptree-beech01.png" scale="50"/>
  </layer>
  <layer type="mask">
    <params name="city" file="city_mask.png" color="0.8 0.8 0.8 0"
      recolor="1" shell_yield="0.5" bullet_yield="1.0" />
    <obj_params mean="0.2" sigma="0.5" group="30"/>
    <object file="StaticMeshes/house1.ive" scale="1.33"/>
    <object file="StaticMeshes/house2.ive" scale="0.3"/>
    <object file="StaticMeshes/house3.ive" scale="0.53"/>
  </layer>
  <layer type="mask">
    <params name="defensive" color="1 1 0 1" recolor="0"
      file="defensive_mask.png" feature_type="box" new_feature_size="2"
      shell_yield="0.1" bullet_yield="0.1"/>
    <obj_params mean="1" sigma="0" group="100"/>
  </layer>
  <layer type="mask">
    <params name="goal" color="1 0.5 0.5 0" recolor="1"
      file="goal_mask.png"/>
  </layer>
  <layer type="mask">
    <params name="road" color="0.2 0.2 0.2 1" recolor="0"
      file="road_mask.png"/>
  </layer>
</map>
```

Appendix F

Intent Experiment Results

F.1 Introduction

The detailed results of the experiment described in Section 4.4.3 are included in this appendix. To recap, this experiment was designed to show the different inverse models being executed and the accuracy of the predictive system.

In the experiment there are five units in the centre of the environment, and a choice of 3 opposing units as targets. These targets are all the same distance away from the centre, but in different directions. The instructions given to the operator are to head towards (attack) one of the targets with a randomly chosen formation, then, after 2 minutes, turn around and retreat back to the starting position with another randomly chosen formation.

F.2 Results

The results of each of the five trials are shown below, with both the trace of the units movements and the confidence levels of each of the hypotheses throughout the scenario.

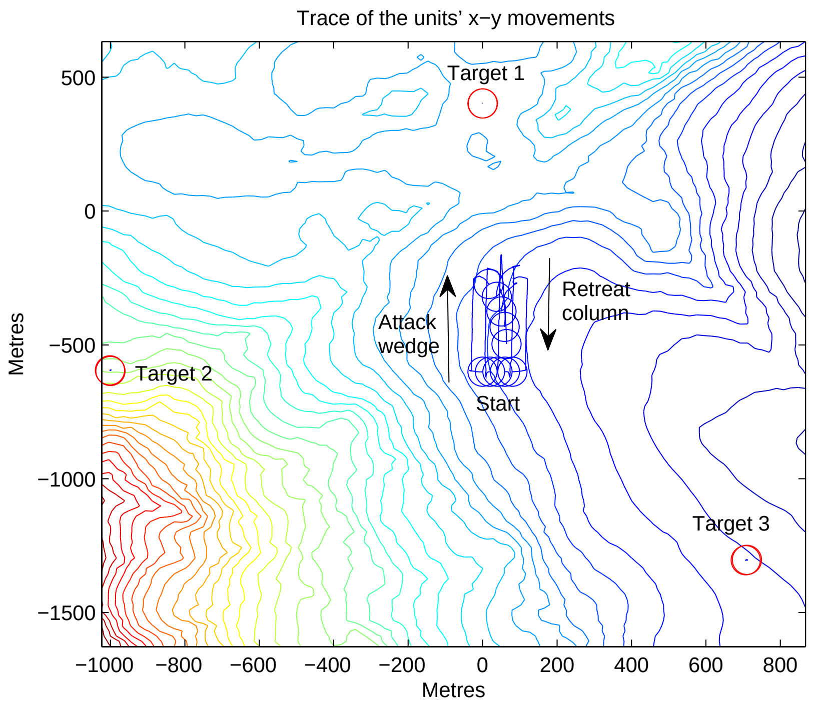


Figure F.1: **Trial 1 scenario.** First the units attack in a wedge formation, then retreat in a column formation.

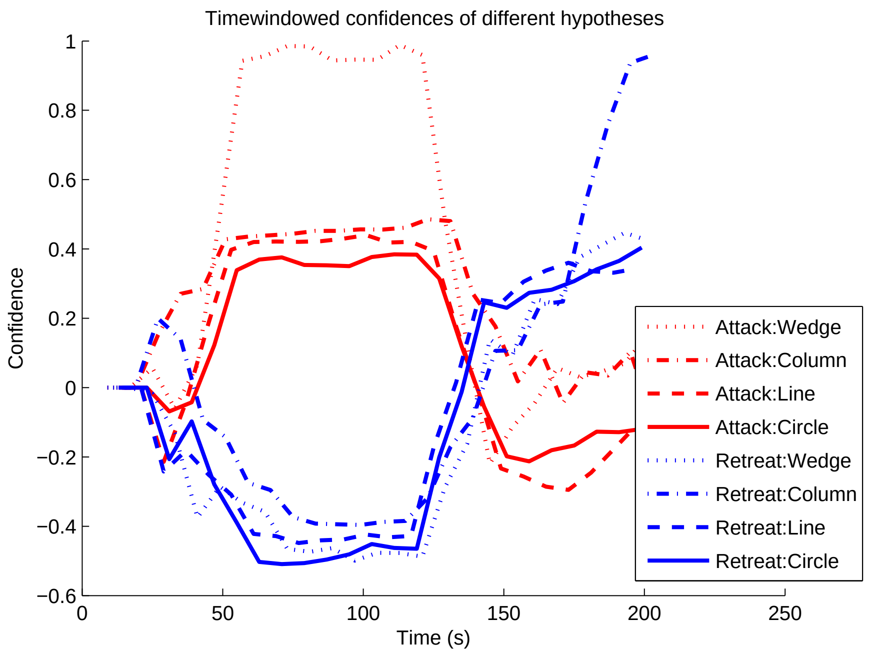


Figure F.2: **Trial 1 results.** The attacking wedge is recognised, followed by the retreating column, after a delay whilst the units reorganise.

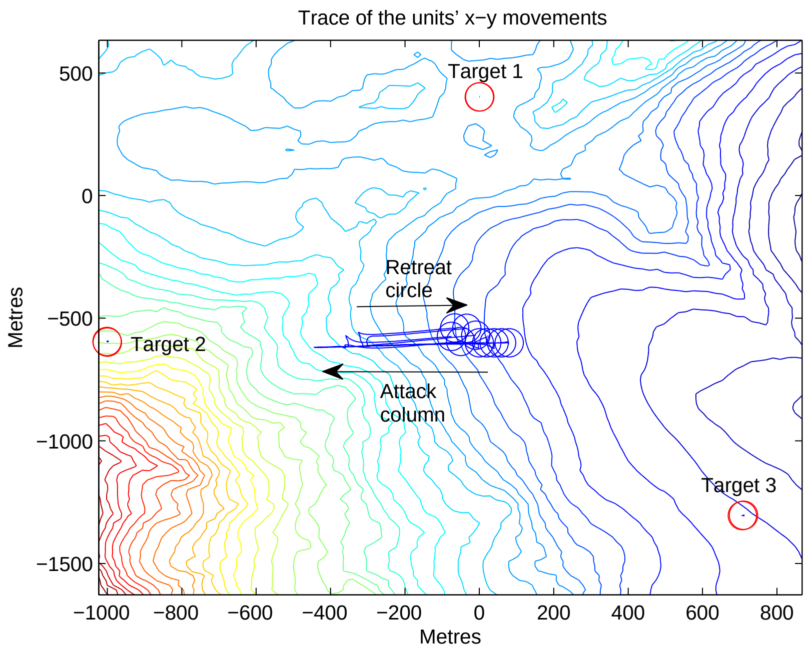


Figure F.3: **Trial 2 scenario.** First the units attack in a column formation, then retreat in a circle formation.

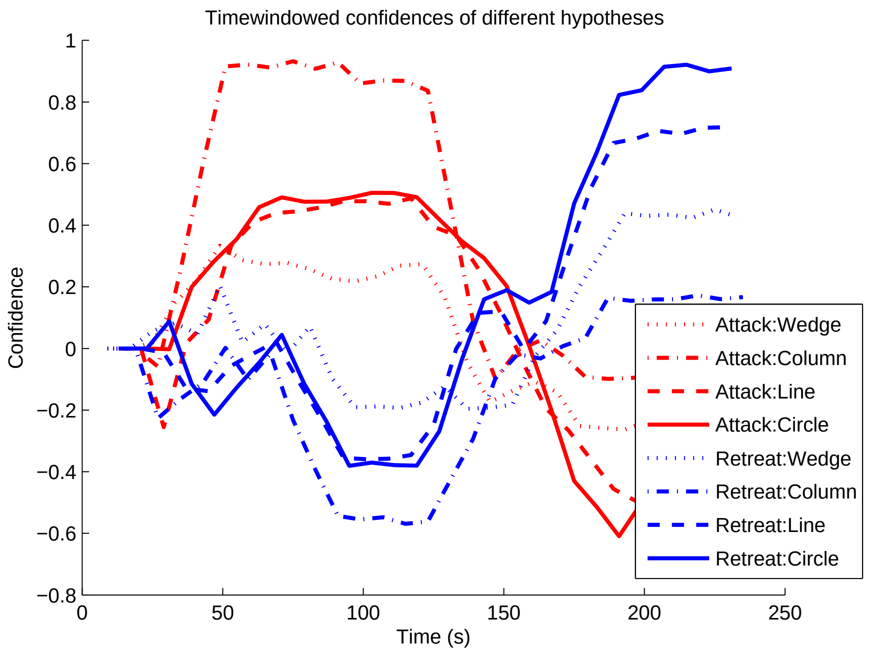


Figure F.4: **Trial 2 results.** The attacking column is recognised, followed by the retreating circle, but retreat line also has a quite high confidence due to the implementation of the formation following code.

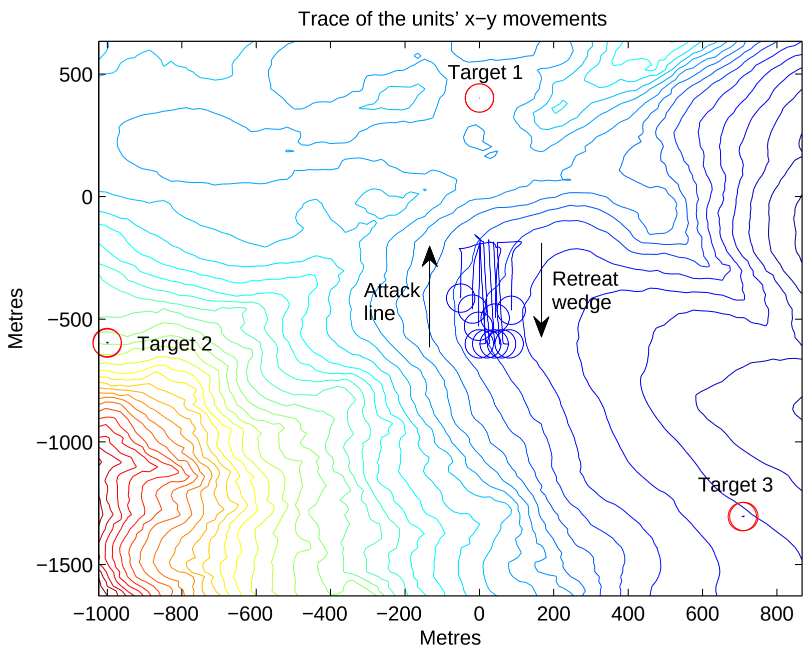


Figure F.5: **Trial 3 scenario.** First the units attack in a line formation, then retreat in a wedge formation.

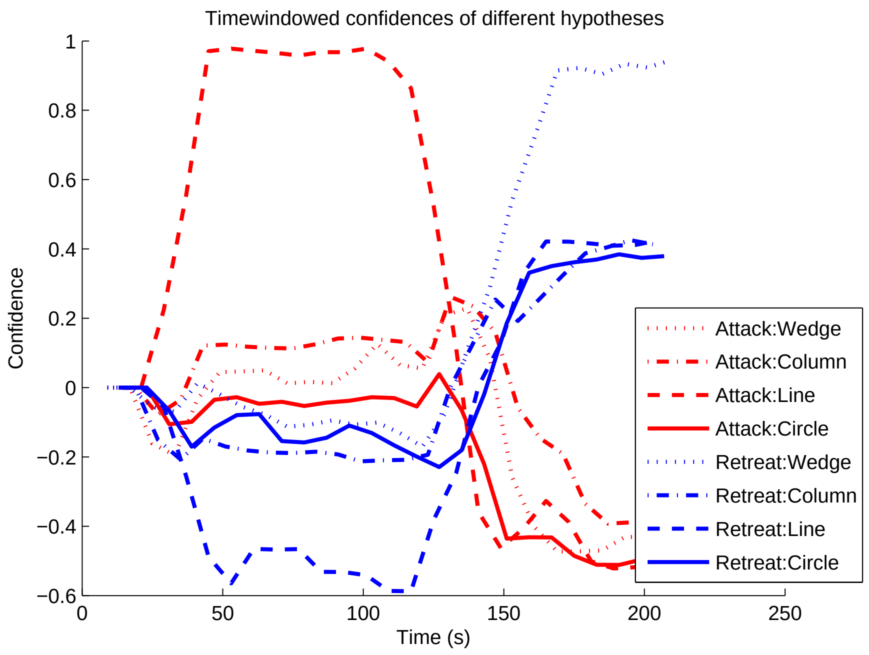


Figure F.6: **Trial 3 results.** The attacking line is recognised, followed by the retreating wedge.

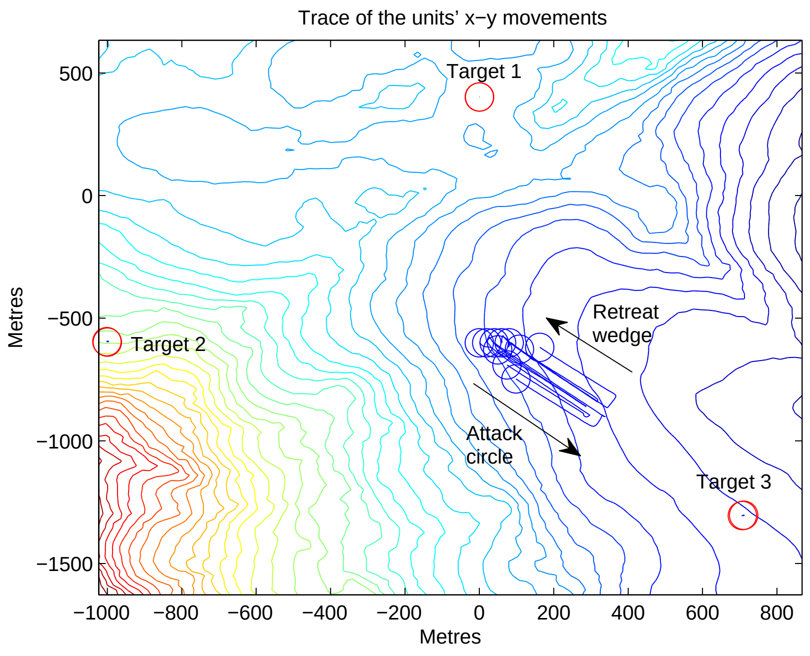


Figure F.7: **Trial 4 scenario.** First the units attack in a circle formation, then retreat in a wedge formation.

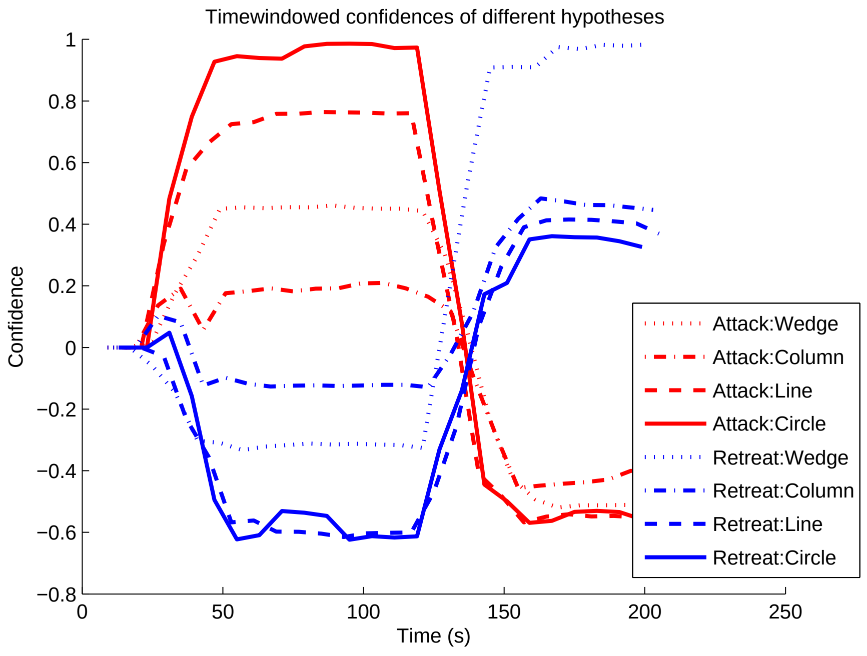


Figure F.8: **Trial 4 results.** The attacking circle is recognised (again with the same relatively high confidence of a line formation as noted in trial 2), followed by the retreating wedge.

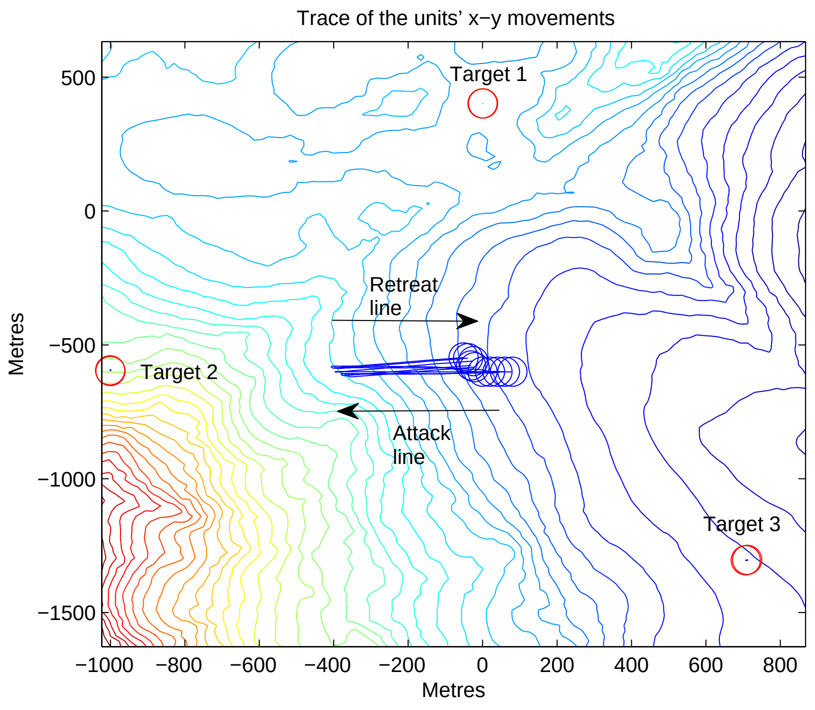


Figure F.9: **Trial 5 scenario.** First the units attack in a line formation, then retreat in the same formation.

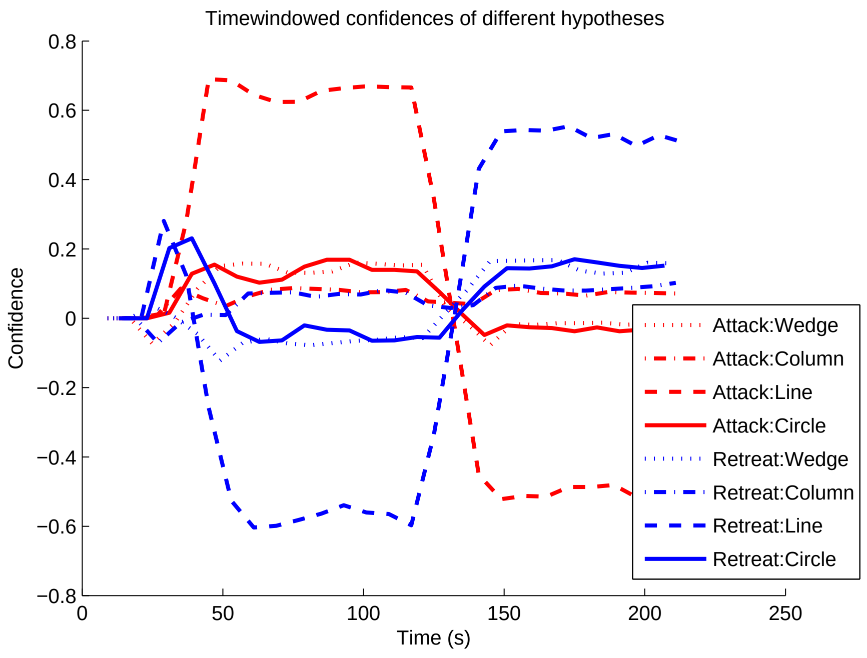


Figure F.10: **Trial 5 results.** The attacking line is recognised, followed by the same formation in retreat.

Appendix G

Example Observer Scenario

G.1 Introduction

This appendix details the exact movements of an observer through an example scenario, in the setting described in Section 6.5. The observer is controlled using the algorithm described in Figure 6.3.

G.2 Example Walkthrough

The complete scenario is shown in Figure G.1. The targets are shown as filled blue circles, and the attackers as empty red circles. The movements of the attackers are shown by the red lines, and when they are being observed, the lines are shown green. For the purposes of this illustration these traces assume the observer travels instantaneously between observations. For the subsequent figures, the exact movements of the observer are shown by the magenta traces. Further details of the observers movements are described in the rest of this appendix.

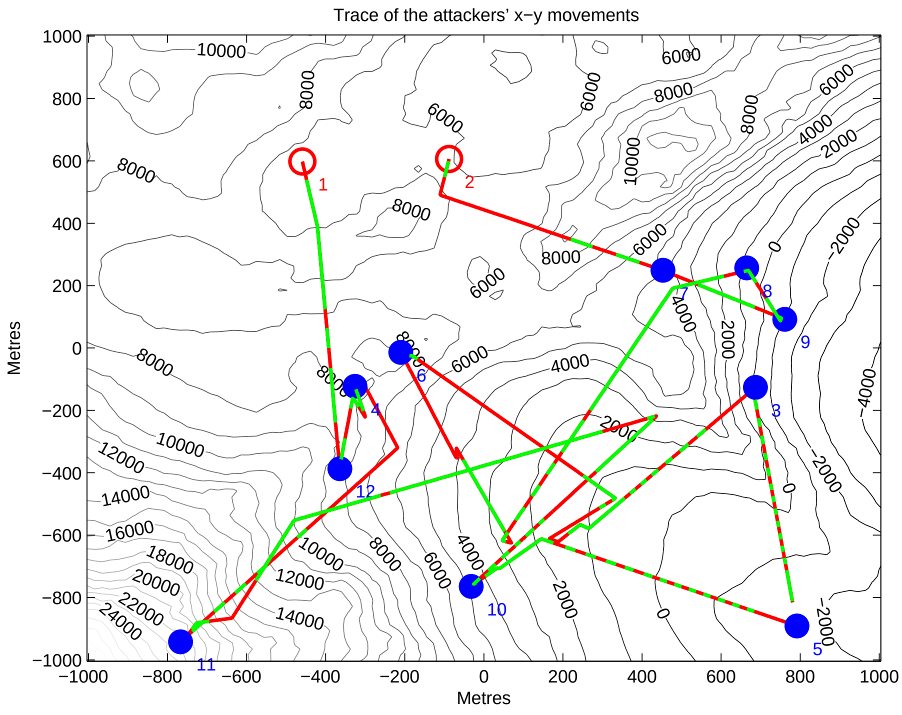


Figure G.1: **The complete example observer scenario.** The red circles are the attackers and the blue circles are the static targets. The green highlights show the times the attacker is being observed with the single observer.

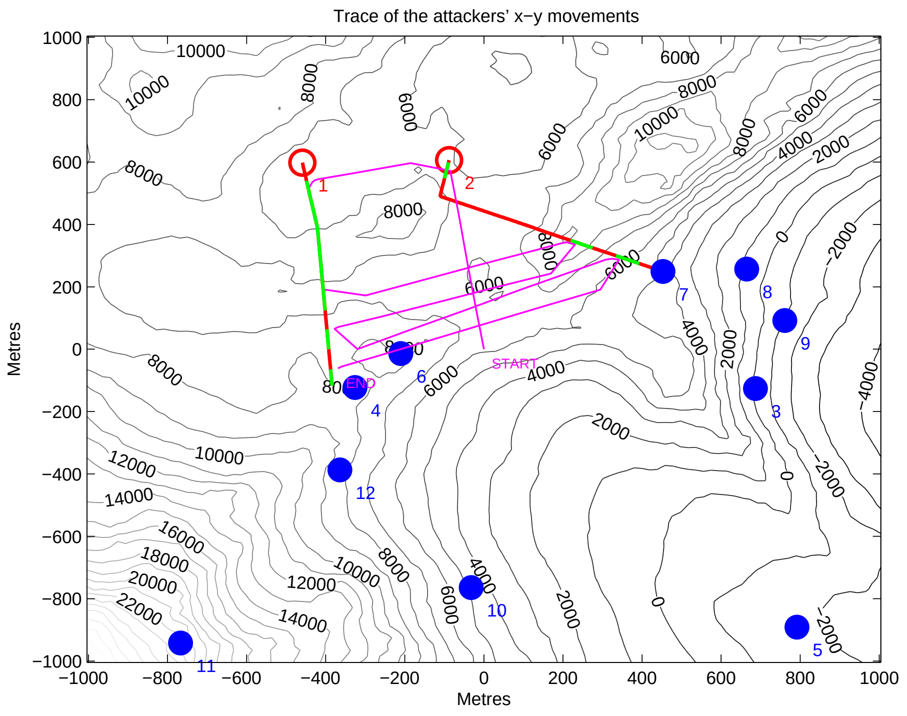


Figure G.2: **Observer movements until unit 7 is reached.** The observer first goes to attacker 2 as it is closest to its nearest target (the lowest leeway). After the first prediction, attacker 2 has a high confidence is is going towards its closest target (target 6), so the observer visits the attacker with the second lowest leeway, attacker 1. The observer continues the observations of attacker 1 because attacker 2 continues to have a high confidence of heading to its closest target (even though it is actually now heading towards target 7), whilst attacker 1's predicted target (target 4, then target 12) is not the closest one (target 6). When attacker 2 gets closer to target 7 it is no longer predicted to go towards its nearest target and has the lowest leeway, so the observer visits attacker 2. Attacker 1 now has the lowest leeway and is not predicted to go towards its nearest target, so it becomes the observee. Now attacker 2 has the lowest leeway, and because the previous prediction actually gave target 9 a slightly higher confidence than the closer target 7, the observer moves to attacker 2. Finally target 1 has the lowest leeway and attacker 2 is predicted with a high confidence to go to its nearest target (target 7) so the observer stays with attacker 1.

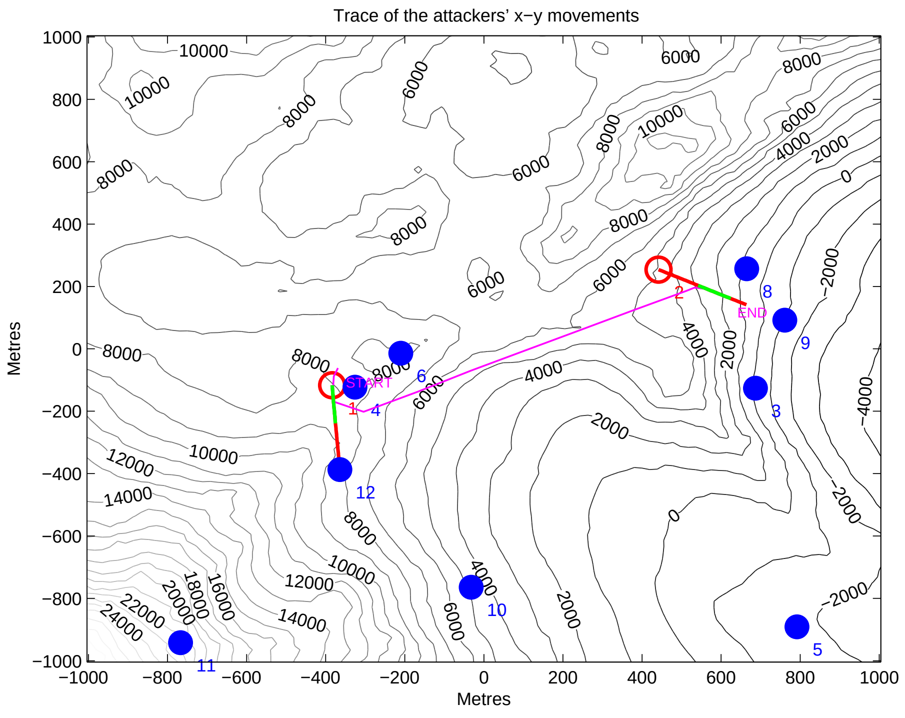


Figure G.3: **Observer movements until unit 12 is reached.** Attacker 1 is observed as it has the lowest leeway regarding target 4, then attacker 2 is observed as it has the lowest leeway regarding unit 8.

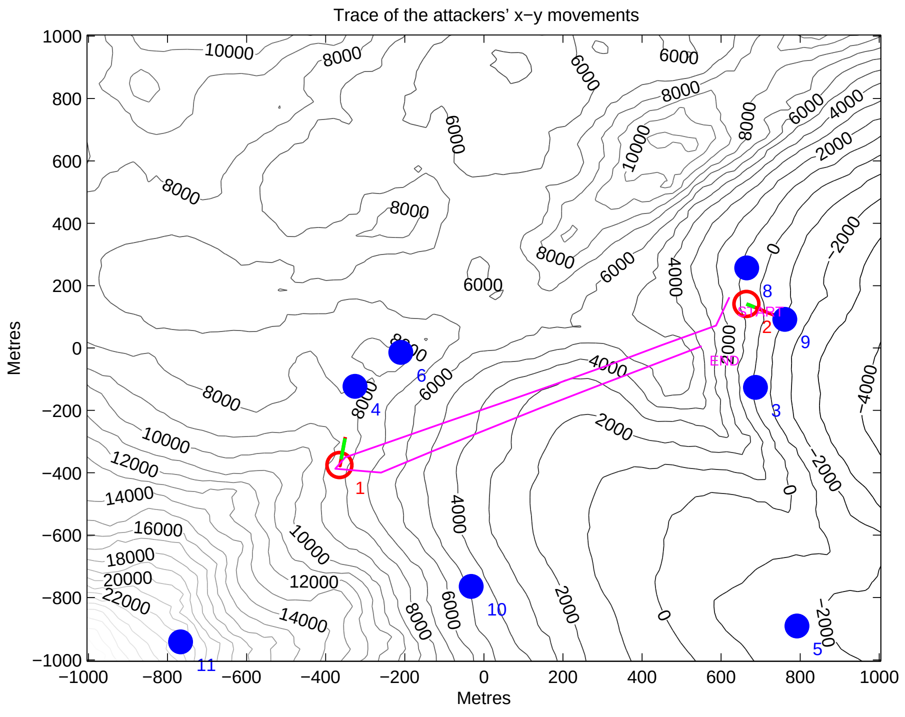


Figure G.4: **Observer movements until unit 9 is reached.** Attacker 2 finishes being observed, and has a high confidence of going towards its nearest target, so the observer moves to attacker 1. Then both have high confidence, so the observer enters round robin mode and goes back to attacker 2.

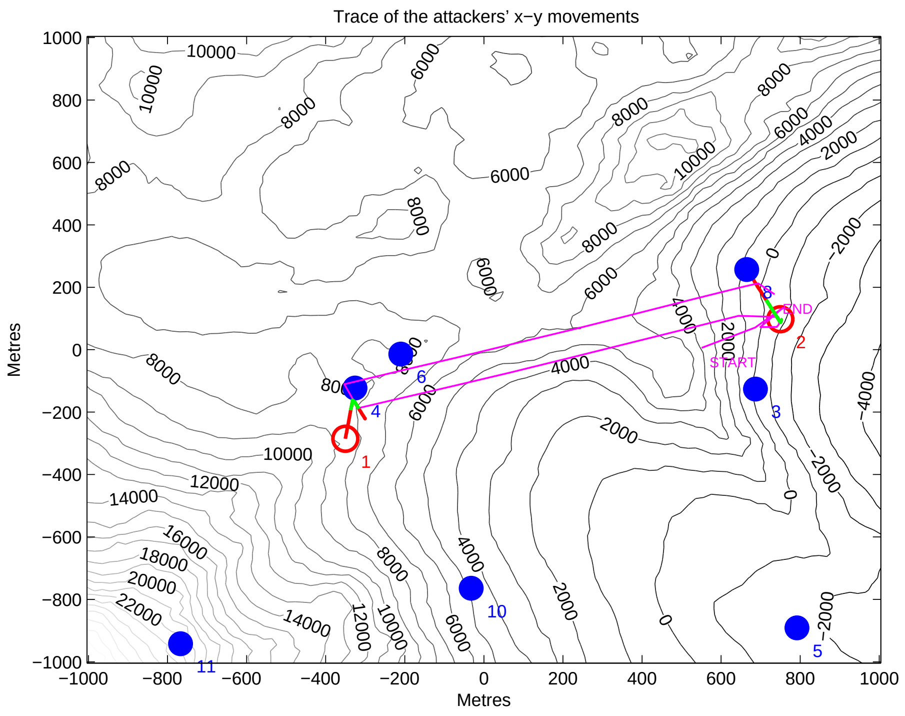


Figure G.5: **Observer movements until unit 8 is reached.** Continuation of the round robin mode, after attacker 2 switches target.

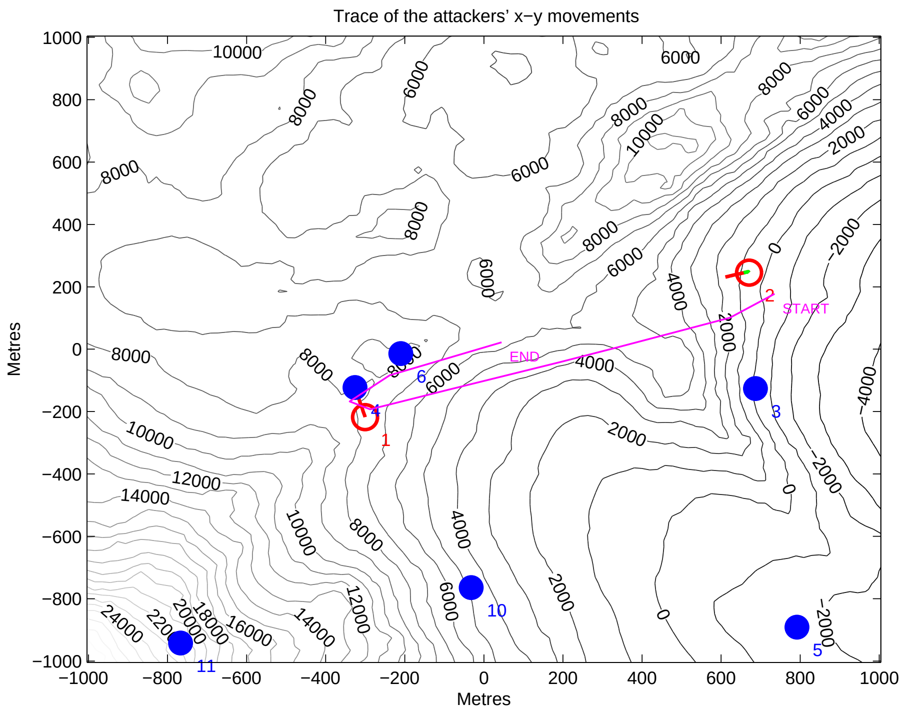


Figure G.6: **Observer movements until unit 4 is reached.** Continuation of the round robin mode.

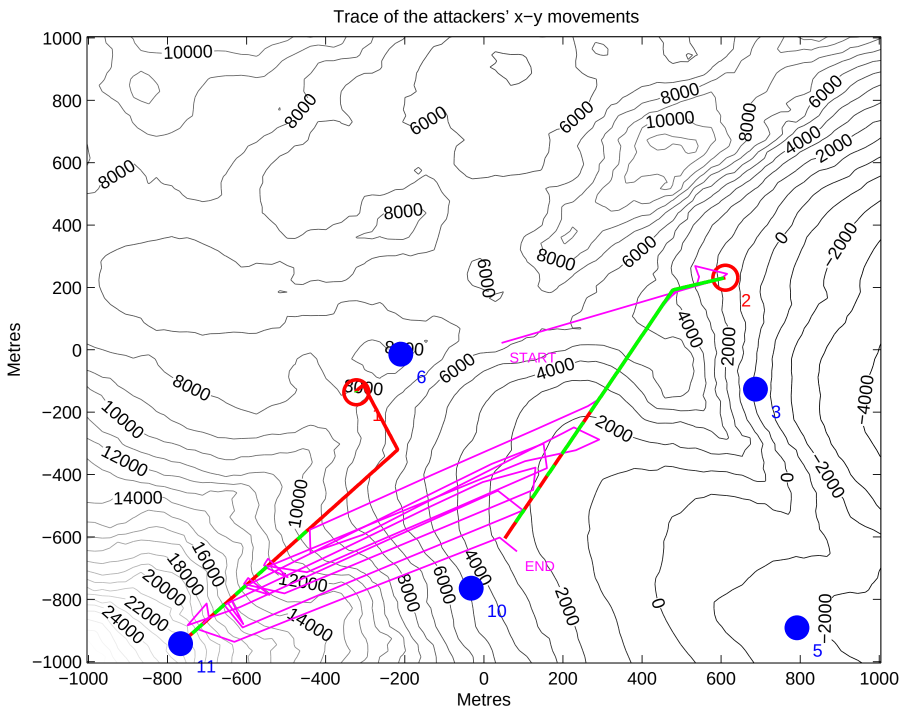


Figure G.7: **Observer movements until unit 11 is reached.** Attacker 2 is observed for a long time as attacker 1 is still predicted to go towards its nearest target (target 6), even though it has changed target to target 10, whereas attacker 2 is not predicted to go towards its nearest target (target 3). When attacker 2 is predicted to go towards its nearest target (target 10), the observer goes into round robin mode.

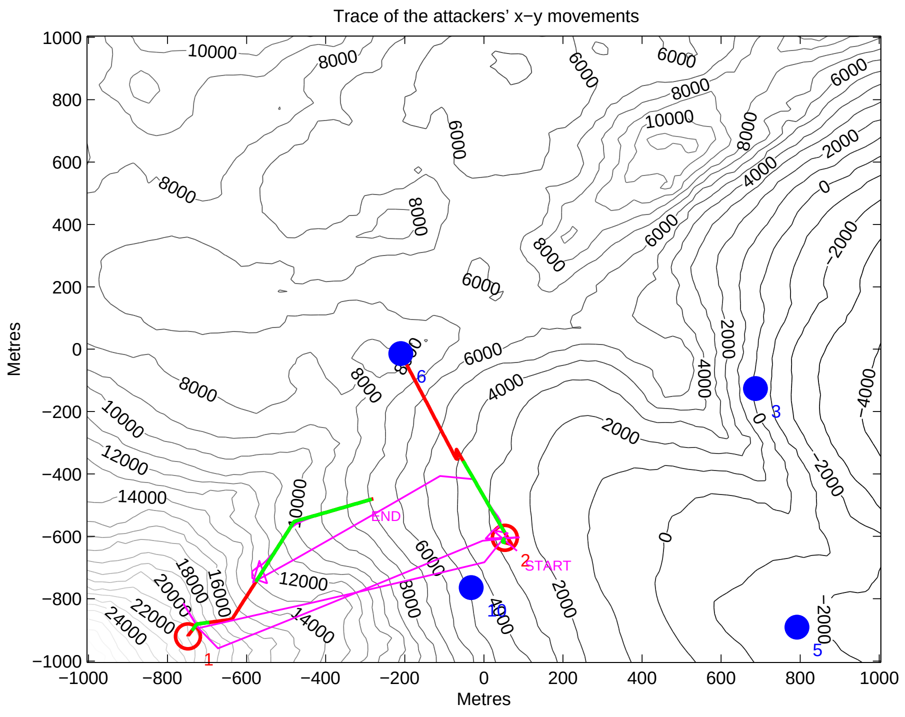


Figure G.8: **Observer movements until unit 6 is reached.** Attacker 2 has the lowest leeway regarding unit 10, until it is closer to unit 6, when the observer switches to attacker 1 which is not predicted to go towards its nearest target.

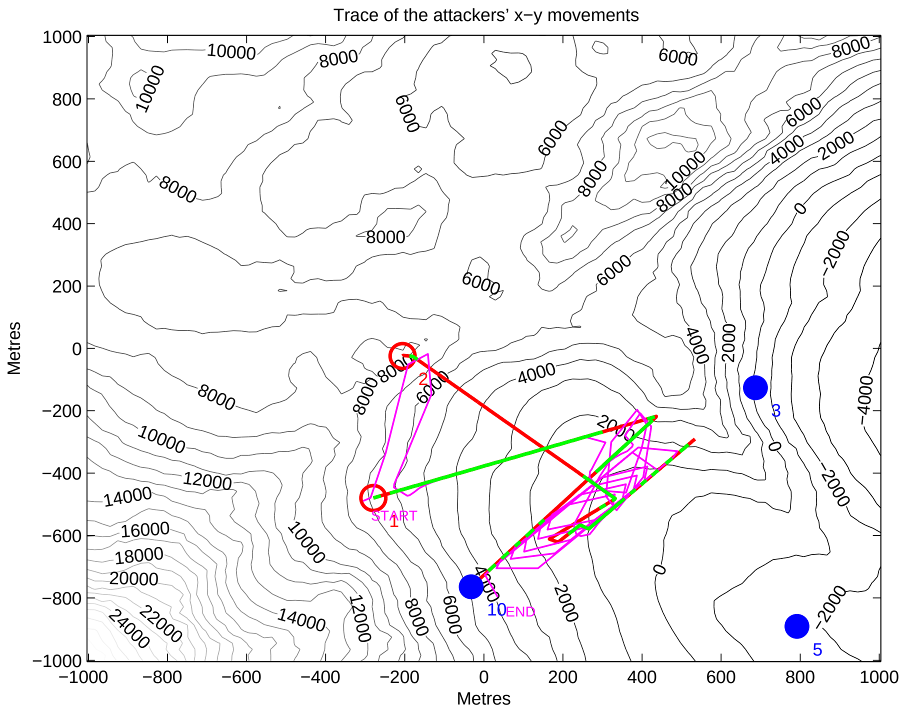


Figure G.9: **Observer movements until unit 10 is reached.** Attacker 2 is observed as it is no longer predicted to go towards its nearest attacker, but attacker 1 has a lower leeway, so it is observed until attacker 2 has a lower leeway. Then the observer enters round robin mode as both targets are predicted to go towards their nearest target.

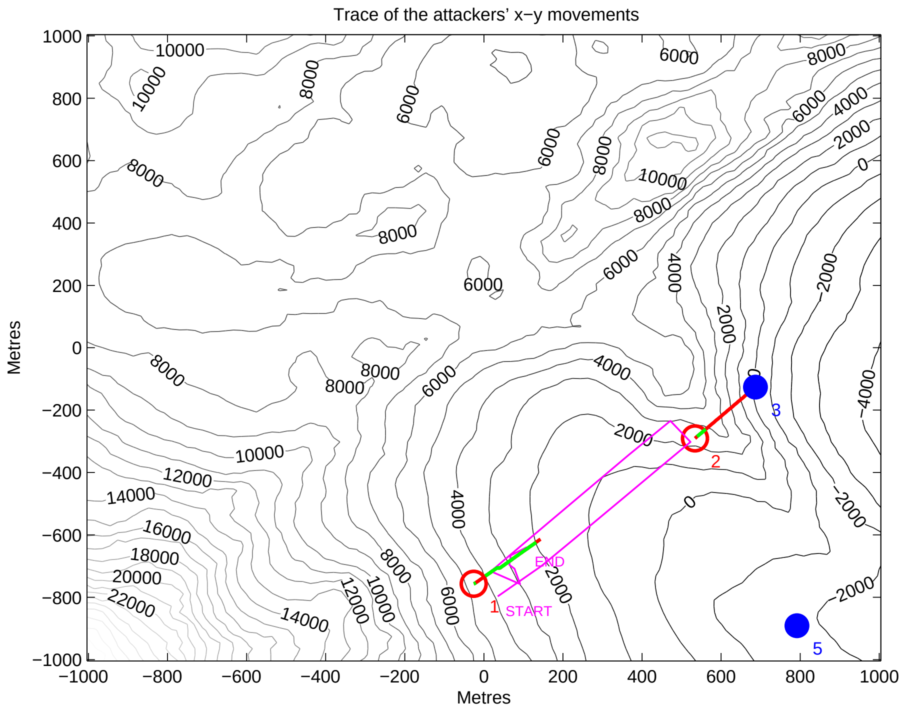


Figure G.10: **Observer movements until unit 3 is reached.** Observe target 1 as it is not predicted to go towards its nearest target.

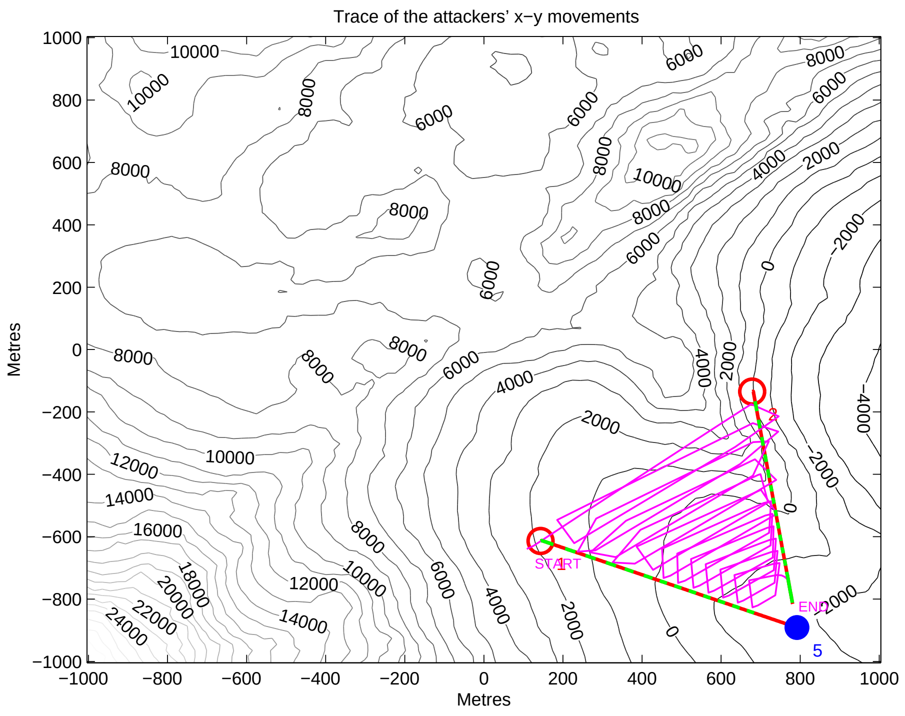


Figure G.11: **Observer movements until unit 5 is reached.** Round robin as both attackers are predicted to go towards their nearest target.