Proceedings of the Third International Symposium on AI & Games, Daniela M. Romano and David C. Moffat (Eds.), at the AISB 2010 convention, 29 March – 1 April 2010, De Montfort University, Leicester, UK

Using a Cognitive Architecture for Opponent Target Prediction

Simon Butler[1] and **Yiannis Demiris**[1]

Abstract. One of the most important aspects of a compelling game AI is that it anticipates the opponent's actions and responds to them in a convincing manner. The first step towards doing this is to understand what the opponent is doing and predict their possible future actions. In this paper we show an approach where the AI system focusses on testing hypotheses made about the opponent's actions using an implementation of a cognitive architecture inspired by the *simulation theory of mind*. The application used in this paper is to predict which target the opponent is heading towards, in an RTS-style game. We improve the prediction accuracy and reduce the number of hypotheses needed by using path planning and path clustering.

1 INTRODUCTION

The use of human-like artificial intelligence (AI) in complex computer and video games has the potential to provide gamers with not only the ideal opponent, but also helpful supporting non-player characters or even just useful predictions of the opposition's strategy in a multi-player game [16]. To achieve this, one avenue to explore is to enable the AI system to recognise and predict the opposing player's actions and movements using observations of their units, obtained from within the game. Once the opponent's objectives can be estimated, the system has a better chance of selecting appropriate moves for the computer-controlled units.

Traditionally, the AI systems in commercial video games were based on scripted events or unwieldy finite state machines, which were time-consuming to construct and error-prone [24, 21]. More recently, games have begun to make use of planners to decide on the computer-controlled player's actions [20], however, research into making predictions about the opposing player is still in its infancy. We propose that taking an approach inspired by neuroscientific and psychological data may provide a more principled way to design parts of the AI decision making system that can predict a player's behaviour. This could lead to an AI that produces more realistic results and, ultimately, a more enjoyable game.

We propose that an interesting way to create an AI that attempts to predict the opponent's actions is to base it on theories of how the human mind operates. The benefits of such cognitive architectures are that they can:

- infer possible mental states of the opponent;
- adapt the performed behaviour, based on such inferences;
- be scalable to the amount of computational resources available;
- have a well-defined architecture, which makes the complexities more manageable.

[1] Electrical and Electronic Engineering Dept., Imperial College London, UK. Email: {simon.butler, y.demiris}@imperial.ac.uk

In this paper we discuss how we adapted a cognitive architecture based on the *simulation theory of mind* to opponent prediction within a gaming scenario. The system is designed for any application that can make use of predictions of multiple units from observations. Here, we have limited the scope to just display the most likely target for each of the opponent's units in a multi-player RTS-style (Real-Time Strategy) game. We detail the techniques we used to improve the accuracy of predictions and reduce some of the computational cost of using this approach.

2 BACKGROUND

It can be seen that recognising and predicting the goals and intentions of the opponent is essentially a problem of matching the state (the situation of each of the players and the game environment at a single point in time) to that produced by a model, where the model represents a possible goal or action. It is also thought that something similar to model-matching happens in the human mind too, through an ability known as *theory of mind*, which attributes *mental states* (beliefs, desires, intentions, etc.) to others and uses them to understand how others will behave [19].

2.1 Theory of Mind

Psychology gives us some clues as to how we, as humans, perform theory of mind, i.e., how we create these models and use them to 'read the mind' of other people. There are two main competing categories of explanations for this cognitive function: *theory theory* and *simulation theory*. Theory theory suggests that the mind builds up a set or rules and laws that are used to give us 'common sense' theories on how others will behave. On the other hand, simulation theory suggests that we do not use explicit theories, rather, we perform a mental simulation (i.e. imagine) of how we would respond if we were put in the situation of the opponent and use that to come to a prediction or explanation [19, 10, 9, 18].

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 plan recognition and prediction. The first approach matches the game states to generalised templates or rules, which are collectively known as *descriptive* 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.

2.2 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. For example, in the multi-agent domain, work has been done to analyse spatio-temporal traces for coordinated motion, whether moving apart or together [8], and, using a similar technique, spatio-temporal models can be created to encode group behaviour, then the traces matched to these models [22]. These approaches have a lower computational cost and may be more robust than generative models, however, more complex behaviours may not be able to be described using these types of models alone.

Other methods include using training data to construct a model. For example, in the games domain, the player's tactical behaviours are shown to be robustly recognised using a naïve Bayes classifier and are applied to directing a non-player character [25]. Also, in the domain of a football (soccer) match, explicit rules are extracted from training data [1] and applied to the current observations. These approaches make it easy to classify new situations, however they could be prone to over-fitting if only a limited amount of training data is available.

2.3 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), in the single-agent domain [3], and in the games domain [11]; by using Monte Carlo simulation [2] or by simply using a planner when the the number of choices can be limited [15]. A number of architectures have also been directly inspired by simulation theory by using internal models to perform prediction through simulation [7, 6].

One such approach (dubbed HAMMER—Hierarchical Attentive Multiple Models for Execution and Recognition of actions) uses a hierarchical system of inverse (plan) and forward (predictor) models, with each model pair (collectively known as internal models) being configured to a particular action [7]. The system finds the matching action by using the model that has the lowest error between the predicted state from each forward model and the actual observed state. This architecture has the advantage that it works well in on-line situations because it will always produce the most closely matching model at any point, rather than other approaches which make a selection then backtrack if it fails. However, it assumes a vast number of hypotheses can be evaluated in parallel, which may not always be practical. A review by [6] shows the applicability of the use of these internal models to multi-agent systems, and HAMMER has been adapted to be used in a multi-agent fully-observable predictive system, as described in [4]. Related work by [23] on segmenting spatio-temporal traces of multiple agents paves the way for applying inverse models to distinct groups of agents.

2.4 SYSTEM

Of the two theories proposed by psychology (see Section 2.1), it is simulation theory has a large body of support from neuroscience [13]. 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 [13] suggests that: when we think of an action we use the motor structures of the brain but actuation is deactivated; perceptual activity can be internally generated within the brain; and 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 [14]. Not only does this theory explain how we infer how others will respond to certain situations, but 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 conceptually maps quite well to a cognitive architecture for action recognition: 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 [6].

Given the successful implementation of the simulationist approach to action recognition in the robotics domain [7] we decided to apply these principles to the games domain.

2.5 Simulationist-architecture

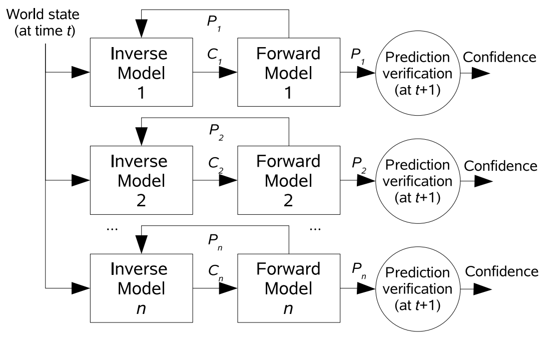


Figure 1. The HAMMER architecture implements the principles of the simulationist approach. Multiple inverse models receive the world state and suggest possible commands (C_1-C_n), which are formed into predictions of the next world state by the corresponding forward model (P_1-P_n). These predictions are verified on the next time step, resulting in a set of confidence values.

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 players can perform. The HAMMER architecture provides a starting point for the system (see Figure 1). It is comprised of three main components: the **inverse**

models (plan generators), the **forward models** (predictors) and the **evaluator** [7, 6].

An inverse model (IM) takes the current world state (which includes the location of each of the units the player controls), a set of units to control, 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 each unit to perform so that they collectively achieve the implicit or explicit target goal(s). 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, on each timestep 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. This model can then be simulated further into the future to provide a prediction of upcoming events.

2.6 Hypotheses and Internal Models in Games

The inverse and forward models used in this architecture are application-dependent. In the games domain, the inverse models will likely depend on the type of plans and actions available to each of the units—fortunately it is very likely that these models already exist as a library of actions a computer-controlled player can perform. For example, the *go-to* IM could output a series of ideal waypoints to route a unit to a specific goal position, or the *formation* IM could simulate repulsive forces between units that updates the target positions of the units to keep them in a specific formation.

The forward models will depend on the game dynamics, and could be statistical models or simplified physics models. However, these may have fairly low fidelity as, for instance, the actual trajectory that the unit will take will be dependent on the interpolation used between the waypoints generated by the IM, 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 effect of these factors cannot necessarily be easily encoded in a statistical model, therefore, the simplest way to get a high quality forward model is to use multiple instantiations of the game environment itself. This enables each instantiation to be fed different hypothesised actions of the opponent's units from the inverse model, and for them to return a predicted state to be compared with the actual state of the game. Hence, we chose to use the game engine to simulate the outcome of the inverse models for greater prediction accuracy.

Several other aspects of the HAMMER architecture have to be modified for application to the games domain. Firstly, the single timestep interval between predictions needs to be lengthened because the timestep of a typical game is less than 30ms. Secondly the objectives of the opponent may change at any time, so simply accumulating the confidence values of each model could mean a long latency before recognising a change in objective. Therefore, instead, the confidences are averaged over a time-window and the model with the resulting highest confidence is considered to be the best prediction.

In this paper we define a plausible action for a unit to be a *hypothesis*; a hypothesis that has been assigned a unit and parameters to be an *instantiated hypothesis*; and the execution of a hypothesis instance to generate a predicted state to be an *internal simulation*.

3 IMPLEMENTATION

The architecture is not limited to any particular game genre, however the predictive power of the system is best illustrated with a game that has an inherent strategic element. Hence we used our own RTS-style game as an implementation platform.

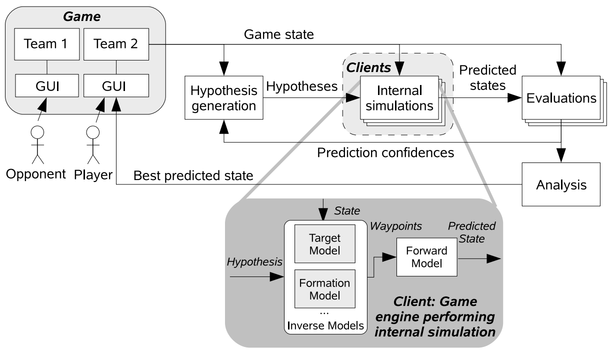


Figure 2. System implementation. The HAMMER architecture implemented into a game framework.

The implementation of the system is divided into three main sections (as shown in Figure 2): the player-controlled game instances that host each team of units (tanks or soldiers in this case); the clients running the game engine in 'hypothesis' mode, executing the requested internal models; and the hypothesis manager that is responsible for generating hypotheses based on the game state, sending the state to the client at the required time and evaluating and analysing the predictions to present the results to the player.

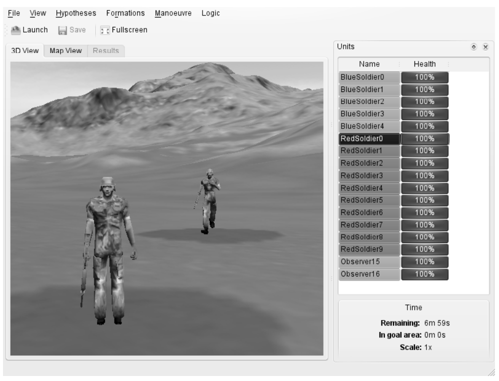
The game hosts two teams of units, n_o opposing units (the 'red' team) and n_t targets for the opposing units (the 'blue' team). Each team is controlled by a single player. Hypothesis instances are created by the *hypothesis generator* and are assigned a specific IM with any optional parameters to achieve the goal, a set of units to apply it to, a duration, and a speedup parameter. These units are then simulated by the FM (in faster than real-time, as controlled by the speedup parameter) using the control signals generated by the IM (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 the confidence that those units are achieving the goal. The best performing hypotheses are analysed (the *analysis* block) and a future predicted state is sent back to the game to be presented to the player.

The hypotheses instantiated by the hypothesis generator form sets H_s^u where $u = 1..n_o$ and $s = 1..S$ where S is the number of different hypothesis types (i.e., the number of available IMs). H_s^u consists of $h_s^u(\mathbf{b})$ which are instantiated hypotheses of the same type and applied to the same unit, but differ in their parameters $\mathbf{b}$.

3.1 Game Engine (and Forward Model)

The engine of the game is based on Delta3D [5], 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.



Name	Health
BlueSoldier0	100%
BlueSoldier1	100%
BlueSoldier2	100%
BlueSoldier3	100%
BlueSoldier4	100%
RedSoldier0	100%
RedSoldier1	100%
RedSoldier2	100%
RedSoldier3	100%
RedSoldier4	100%
RedSoldier5	100%
RedSoldier6	100%
RedSoldier7	100%
RedSoldier8	100%
RedSoldier9	100%
Observer15	100%
Observer16	100%

Time

Remaining: 6m 59s
In goal area: 0m 0s
Scale: 1x

Figure 3. Game screenshot. This shows the graphical interface for the human-controlled teams.

The 3D engine (OSG) was used to model a large outdoor terrain (see the screenshot in Figure 3), 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 characters and vehicles, with additional control of their aiming and firing mechanisms. The noise introduced by the physics engine provides a stochastic element to the outcome of scenarios.

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

To support the use of the game engine as a forward model, the engine has to support a few features not usually exposed with an external API. It needs to be able to read the complete current state of the environment and transfer that state to 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 the main game engine.

3.2 Inverse Models and Parameterisation

Any reasonably complex game will have many different inverse models, and these need to be parameterised to cover the range of behaviour that each unit can perform, whilst reducing the number of models needed to a manageable level.

Most parameters 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 IMs to execute. For example, a position parameter that could be used by a *go-to* inverse model could hypothetically 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. For example, units are likely to be either heading to: attack a group of opposing units, a good defensive position, or rendez-vous with another group of units. 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 we shall discuss optimising the target position parameter for the *go-to* IM in our RTS game. The simple approach is to instantiate an IM that sends the unit in a straight line towards the designated target, and to instantiate that IM for each of the targets in the game (we shall refer to this as the straight-line-to-target hypothesis). However, there is no guarantee that by following the straight line the unit is able to reach the target, and additionally, there is likely to be considerable duplication of effort if multiple targets lie on the same heading from the unit.

One way to address these flaws is to generate paths to each of the possible targets and then cluster them to find similar paths (we shall refer to this as the path-to-target hypothesis). Therefore only paths that are in the centre of the clusters need to be simulated. This approach is discussed in detail below.

3.2.1 Path Generation

In games, a popular way to calculate an optimal path between two points is by using the A* search algorithm, which uses heuristics to reduce the graph-based search space [12]. 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 (1) 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 [12], but the main implementation detail is the evaluation function:

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \tag{1}$$

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.

In our implementation, we used the low resolution 512×512 pixel terrain heightmap grid as the graph M to search. For the least-cost estimate $\hat{h}(n)$ we approximate the minimum time it would take for the unit to travel to the goal position. We set the path cost function $\hat{g}(n)$ to depend on the speed of the unit and the type of terrain to traverse. It also depends on the gradient between each node, as each unit type has a limit on its maximum traversable gradient, hence some nodes may be unreachable to certain unit types.

3.2.2 Path distance metric

The A* algorithm ensures that relevant terrain features are considered when appropriate paths to targets are generated, however it still means that there could potentially be very similar paths to targets that are near to each other. These paths could easily be combined into a single hypothesis instance using a clustering algorithm to save simulating all of the possible paths.

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 we have to define our own metric 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 a hypothesis are generated after a short

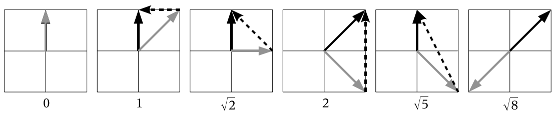


Figure 4. Examples for each of the possible path deviation scores. The vector for one segment of the first path (p_v^i) is shown in black and a segment from the other path (p_w^i) is in grey. The difference between the vectors is shown as a dashed arrow, and the path deviation score ($\|p_v^i - p_w^i\|$) is the magnitude of this vector, the value of which is shown below each example.

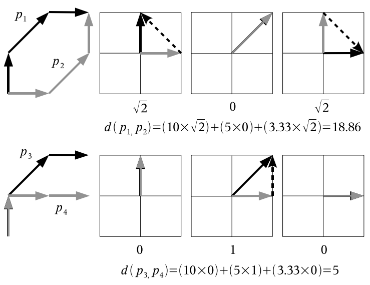

p_1
p_2
√2
0
√2
$d(p_1, p_2)=(10\times\sqrt{2})+(5\times 0)+(3.33\times\sqrt{2})=18.86$
p_3
p_4
0
1
0
$d(p_3, p_4)=(10\times 0)+(5\times 1)+(3.33\times 0)=5$

Figure 5. 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.

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 we chose to compare paths 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 4), 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. Therefore the distance between two paths p_v and p_w is:

$$d(p_v, p_w) = \sum_{i=1\ldots m_{\min}} \alpha(i) \cdot \|p_v^i - p_w^i\| \tag{2}$$

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.

3.2.3 *Path clustering*

Once we have a measure of the distance between two paths, we can use it to find the similarity matrix that compares each of the paths with all of the others. After that we can use several different clustering algorithms, however here we shall be using a spectral clustering algorithm that automatically chooses the number of clusters [17, 23].

From the similarity matrix we form an affinity matrix using the equation

$$A_{ij} = \exp\left(\frac{-d^2(p_i, p_j)}{\sigma^2}\right) \tag{3}$$

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 [17], but to summarise: 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.

3.3 Evaluation Process

Now that we have an inverse and forward model pair producing predicted unit positions, we need to evaluate which of these predictions match the observed behaviour.

After a hypothesis instance $h_s^u(\mathbf{b}_i)$ has finished executing it returns the starting position $\mathbf{x}_{\text{start}}$ (recorded at the beginning of the execution of a hypothesis), the actual position $\mathbf{x}_{\text{actual}}$ (recorded after the chosen hypothesis time), and the corresponding predicted position $\mathbf{x}_{\text{predicted}}$ (as calculated by the forward model).

From this information we can calculate the confidence function:

$$c\left(h_s^u(\mathbf{b}_i)\right) = \begin{cases} 1 & \text{if } |\vec{a}| < d \text{ and } |\vec{p}| < d, \\ \hat{\vec{a}} \cdot \hat{\vec{p}} \frac{\min(|\vec{a}|, |\vec{p}|)}{\max(|\vec{a}|, |\vec{p}|)} & \text{otherwise.} \end{cases} \tag{4}$$

where $\vec{a}$ is the vector from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{actual}}$ for unit u and $\vec{p}$ is the vector from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{predicted}}$ for unit u in hypothesis h, $\hat{\vec{a}}$ and $\hat{\vec{p}}$ are the normalised vectors, $|\vec{a}|$ and $|\vec{p}|$ are the magnitude of the vectors, and d is the deadzone that below which the unit is considered to be stationary (we used a value of $d = 1$ for our experiments).

This confidence function is designed to be 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, where 1 means high confidence, 0 means unknown confidence, and -1 means no confidence.

The dot product term tells us what direction the unit travelled, but we would also like 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 we scale the result of the dot product by the magnitude of the shortest vector ($\min(|\vec{a}|, |\vec{p}|)$), as a proportion of the magnitude of the longest vector ($\max(|\vec{a}|, |\vec{p}|)$).

An additional condition was added so that if the magnitudes of both vectors are less than d (in our case one metre) 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 this term adds a dead-zone with a radius of 1m, within which the unit is counted as stationary, i.e. if we predict no movement and there is only a small movement, then we are confident of our prediction, regardless of heading.

It follows that the best matching hypothesis for a particular unit u and hypothesis type s is that with the highest confidence over all the tested parameters $\mathbf{b}$:

$$c_s^u = \max_i c\left(h_s^u(\mathbf{b}_i)\right) \tag{5}$$

4 EXPERIMENTS & RESULTS

To test the effectiveness of the path-to-target hypothesis that uses the path clustering (as described in Section 3.2), we compared it with the straight-line-to-target hypothesis.

The 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.

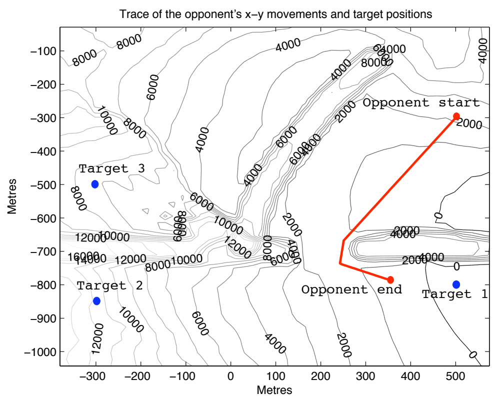


Figure 6. Scenario overview. This shows the movement of the opponent's unit, and the positions of the target units for the duration of the scenario.

The environment and location of the targets is shown in Figure 6. 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 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.

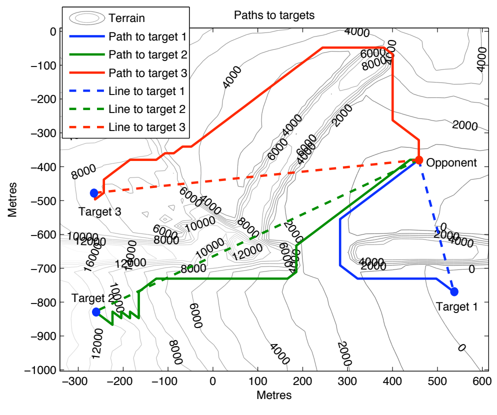


Figure 7. Predicted paths. This shows the paths for both hypotheses at timestep 41. It can be seen how the paths to targets 1 and 2 have a high path similarity, and that the straight-lines are quite dissimilar to the paths for targets 1 and 3.

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 we have two hypotheses sets H_s^u where $u = 1$ and $s = 1, 2$, and each contains three hypothesis instantiations, one for each of the three targets, $h_s^u(\mathbf{b}_i)$ where $i = 1..3$, making a total of six 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. 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.

The path planning and clustering occurs each time the path-to-target hypothesis set starts a new prediction, 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. 1ms) 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 7. 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.

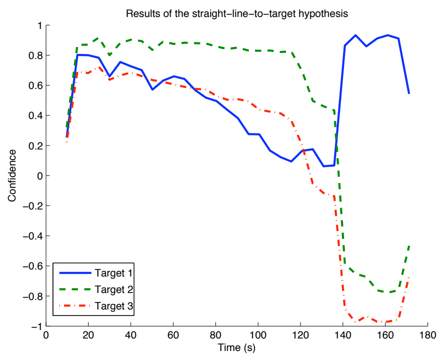


Figure 8. Target confidence using the straight-line-to-target hypothesis. Target 1 is only identified as the most confident target after timestep 140.

The confidence level to each of the targets using the straight-line-to-target hypothesis is shown in Figure 8. The highest confidence target is 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 9 the path-to-target hypothesis correctly identifies 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.

It can also be seen that the path-to-target hypothesis manages to out-perform the straight-line-to-target hypothesis whilst also execut-

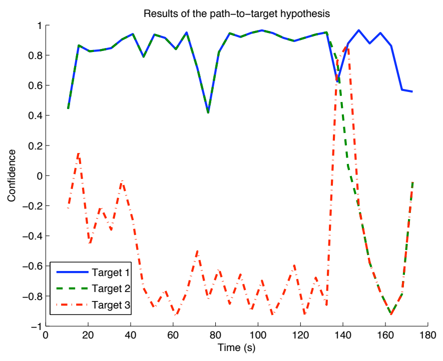


Figure 9. Target confidence using the path-to-target hypothesis. Target 1 has a consistently high confidence throughout the scenario.

ing 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.

5 CONCLUSION

In this paper we have discussed a cognitive architecture based on simulation-theory that is capable of providing predictions of an opponent's actions and intentions. We have shown some initial results on how this can be applied to the prediction of a single action (the selected target) of an opposing player in an RTS-style game. Through the use of A* path finding and spectral clustering of the resultant paths, the target predictions out-perform a naïve straight-line target predictor, whilst also successfully combining similar paths to avoid redundant executions.

The platform 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 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 in a typical game, and, although the work shown here uses two simple inverse models, the same architecture can be applied to any action the units can perform.

However, even though we have shown a way to reduce the number of hypothesis instantiations, problems are likely to occur when many different hypothesis sets (i.e., different types of inverse model) need to be executed concurrently, so future work involves investigating other approaches to reducing this computational complexity, such as using hierarchical inverse models. Also, adding the constraint of partial observability to the system would reduce the number of hypotheses as they can only be executed when there are reliable observations. This would also making the game more realistic, as the opponent would avoid getting the impression that the AI is "cheating" by having a global view of the environment. Additionally, there is an assumption that the opponent will behave in a manner that is similar the output of the models—future work includes looking at methods to tune the inverse models to match the observed behaviour of the opponent.

REFERENCES

[1] M. Beetz and B. Kirchlechner, 'Computerized real-time analysis of football games', *IEEE pervasive computing*, **4**(3), (2005).

[2] Y. Bjornsson and H. Finnsson, 'Cadiaplayer: A simulation-based general game player', *Computational Intelligence and AI in Games, IEEE Transactions on*, **1**(1), 4–15, (March 2009).

[3] H. Bui, S. Venkatesh, and G. West, 'Policy recognition in the abstract hidden markov models', *Journal of Artificial Intelligence Research*, **17**, 451–499, (2002).

[4] S. Butler and Y. Demiris, 'Predicting the movements of robot teams using generative models', in *Distributed Autonomous Robotic Systems 8*, 533–542, Springer, (2009).

[5] R. Darken, P. Mcdowell, and E. Johnson, 'The Delta3D open source game engine', *IEEE computer graphics and applications*, **25**(3), (2005).

[6] Y. Demiris, 'Prediction of intent in robotics and multi-agent systems', *Cognitive Processing*, **8**(3), 151–158, (September 2007).

[7] Y. Demiris and B. Khadhouri, 'Hierarchical attentive multiple models for execution and recognition of actions', *Robotics and autonomous systems*, **54**(5), (2006).

[8] M. Devaney and A. Ram, 'Needles in a haystack: Plan recognition in large spatial domains involving multiple agents', in *National Conference on Artificial Intelligence*, (1998).

[9] V. Gallese and A. Goldman, 'Mirror neurons and the simulation theory of mind-reading', *Trends in cognitive sciences*, **2**(12), 493–501, (1998).

[10] R.M. Gordon, 'Simulation vs theory-theory', *The MIT Encyclopædia of the Cognitive Sciences*, 765–766, (1999).

[11] P. Gorniak and D. Roy, 'Probabilistic grounding of situated speech using plan recognition and reference resolution', in *Proceedings of the 7th international conference on Multimodal interfaces*, p. 143. ACM, (2005).

[12] P. E. Hart, N. J. Nilsson, and B. Raphael, 'A formal basis for the heuristic determination of minimum cost paths', *Systems Science and Cybernetics, IEEE Transactions on*, **4**(2), 100–107, (July 1968).

[13] G. Hesslow, 'Conscious thought as simulation of behaviour and perception', *Trends in Cognitive Sciences*, **6**(6), 242–247, (June 2002).

[14] M. Johnson and Y. Demiris, 'Perceptual perspective taking and action recognition', *International Journal of Advanced Robotic Systems*, **2**(4), 301–308, (2005).

[15] J. Laird, 'It knows what you're going to do: adding anticipation to a quakebot', in *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pp. 385–392, New York, NY, USA, (2001). ACM Press.

[16] J. Laird and M. van Lent, 'Human-level AI's killer application: Interactive computer games', *AI magazine*, **22**(2), 15, (2001).

[17] Lihi Z. Manor and P. Perona, 'Self-tuning spectral clustering', *Eighteenth Annual Conference on Neural Information Processing Systems, (NIPS)*, (2004).

[18] M. Michlmayr, *Simulation Theory versus Theory Theory: Theories concerning the Ability to Read Minds*, Master's thesis, University of Innsbruck, Mar 2002.

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

[20] J. Orkin, 'Three states and a plan: The AI of F.E.A.R.', in *Proceedings of the Game Developer's Conference (GDC)*, (2006).

[21] B. Scott, 'The illusion of intelligence', in *AI Game Programming Wisdom*, ed., Steve Rabin, 19–20, Charles River Media, Inc., (2002).

[22] G. Sukthankar and K. Sycara, '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*, pp. 638–645, New York, NY, USA, (2006). ACM.

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

[24] P. Tozour, 'The perils of AI scripting', in *AI Game Programming Wisdom*, ed., Steve Rabin, 541–547, Charles River Media, Inc., (2002).

[25] W. van der Sterren, 'Being a Better Buddy: Interpreting the Players Behavior', in *AI Game Programming Wisdom 3*, ed., Steve Rabin, Charles River Media, Inc., (2006).