Partial Observability During Predictions of the Opponent's Movements in an RTS Game

Simon Butler, *Member, IEEE*, and Yiannis Demiris, *Senior Member, IEEE*

***Abstract*— In RTS-style games it is important to be able to predict the movements of the opponent's forces to have the best chance of performing appropriate counter-moves. Resorting to using perfect global state information is generally considered to be 'cheating' by the player, so to perform such predictions scouts (or observers) must be used to gather information. This means being in the right place at the right time to observe the opponent. In this paper we show the effect of imposing partial observability onto an RTS game with regard to making predictions, and we compare two different mechanisms that decide where best to direct the attention of the observers to maximise the benefit of predictions.**

I. Introduction

When working in an adversarial environment it is of considerable advantage to be able to recognise and predict the movements of the opponent's forces, so that the most beneficial counter-strategy can be generated. Therefore it is very important to know the detailed whereabouts of the opposing team's units, so that predictions about their future behaviour can be formed.

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 [1], and full-immersion is crucial in creating an entertaining game [2].

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 AI research. Even so, in this paper we assume that, though it greatly increases the difficulty of the problem, it is desirable to have the computer-controlled player compete within the same framework as the human player. This paper builds on our previous work on using *simulation-theory* for prediction in a multi-agent environment [3, 4]. Here, we try to give parity to both computer and human players by limiting their view of the environment to a composite of the areas that can be perceived by a scout (or, in general, an *observer*). Therefore the deployment and coordination of the scouts that perform the observations of the opposing units is an important issue.

The authors are with the Electrical and Electronic Engineering Department, Imperial College London, UK (email: {simon.butler, y.demiris}@imperial.ac.uk).

In this paper we investigate how making the system partially observable affects the number and timeliness of predictions that can be made. We compare two methods 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.

II. Background

Given that the computer-controlled player has a limited view of the environment, the problem is 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) [5]. Stimulus-driven means attending to the most *salient* regions of the environment and applying a winner-takes-all strategy [6]. 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 [7, 8]. 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 game so a method to dynamically infer the top-down information by predicting the player's actions is needed.

In previous work we have used an architecture (dubbed HAMMER) that uses a mechanism derived from *simulation theory* for predicting the movements of the opponent [4]. It uses the same models for both prediction and execution, so it can reuse behaviours to both act overtly and to simulate actions and their consequences. Other work by Southey et al. [9] uses hidden semi-Markov models (HSMMs) to infer motion and potential targets from partial trajectories. Alternative approaches to obtaining top-down information include opponent modelling [10] or planning [11, 12] but the type of information obtained from these approaches is of a higher level than can be directly used by an attention mechanism.

In this paper we incorporate the goal-directed information with other content features to make an threat-based attention mechanism to increase the effectiveness of the observations. The goal-directed information can be obtained by any of the approaches listed above, however, here we shall use

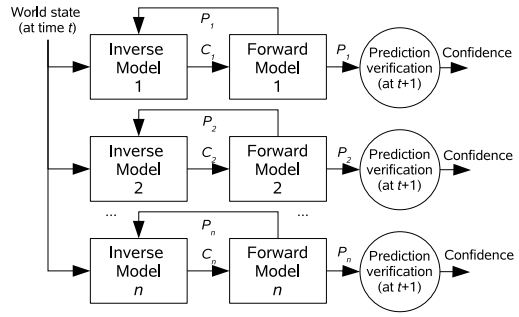

Fig. 1. The HAMMER architecture implements the principles of the simulationist approach. Multiple inverse models receive the world state and suggest possible commands (C_1-C_n), which are formed into predictions of the next world state by the corresponding forward model (P_1-P_n). These predictions are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

HAMMER, our simulation-based architecture, to generate predictions.

III. Architecture

A. HAMMER

The HAMMER (Hierarchical Attentive Multiple-Models for Execution and Recognition of actions) architecture provides a base for the predictive system (see Fig. 1). It is comprised of three main components: the **inverse models** (plan generators), the **forward models** (predictors) and the **evaluator** [5, 13].

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 generates 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 indicate 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.

The inverse and forward models used in this architecture are application-dependent and are described in Sec. IV.

B. 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. I, this complete state is not generally available to the player in RTS games, and obtaining it is costly in terms of time and resources. Therefore, we restrict ourselves to obtaining only the information that is required for particular inverse models. For example, a *surround* manoeuvre IM might require observations of many enemy units within a certain vicinity of a target.

The method for deciding which inverse models deserve being attended to becomes a question of resource scheduling. One of the most basic scheduling algorithms is *round-robin* scheduling [14], where the required units are observed in a fixed sequence. When the last unit has been observed the sequence starts again from the beginning. We can, however, utilise other information we have available in the system 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.

1) Utility: The utility of an observation can be formulated by estimating the threat of the unit requiring the prediction, e.g., how far it is away from the nearest target.

2) 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.

3) 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.

4) 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. We combine these measures into a threat-based attention mechanism (described in Sec. IV-G) that optimises the observations based on the objectives outlined below.

IV. Implementation

To illustrate the effects of attention, we use a simplified RTS-style game to make predictions about the movements of the opponent's units. We use this information to infer the target that they are heading towards. The main objective is to always strive to have predictions of the opponent's units

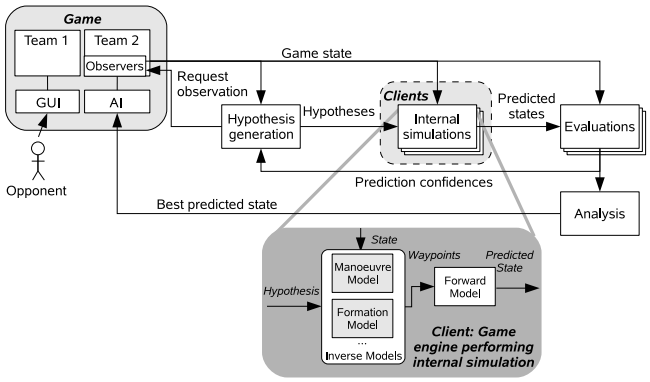

Fig. 2. System implementation—incorporating HAMMER into a partially-observable game.

before they reach their targets, and the earlier the predictions are made the better.

A. System overview

The implementation of the system is divided into three main sections (as shown in Fig. 2): the game instance 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). It is the 'blue' team that runs the predictive system against the opposing 'red' team. Hypothesis sets H_s^u are formed (in the *hypothesis generator*) for each unit $u = 1..n_o$ and hypothesis type $s = 1..S$—although there is just one hypothesis type (*go-to target*) used in the experiments in this paper. Each hypothesis instance within these sets $h_s^u(\mathbf{b})$ is given parameters (e.g. for the *go-to target* hypothesis type $\mathbf{b} = \{\beta, e, f\}$ where β is one of the targets, e is the execution (and hence observation) duration—in this case 5 seconds, and f is the speedup parameter of the FM—in this case 4 times normal speed). For each hypothesis instance the assigned unit u is then simulated by the FM (in faster than real-time, as controlled by the speedup parameter f) using the control signals generated by the IM (the *internal simulations* 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 used by the AI system.

B. Inverse Models

The inverse models depend on the type of plans and actions available to each of the units—fortunately, for many games, 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 outputs a series of ideal waypoints to route a unit to a specific goal position, or the *formation* IM simulates repulsive forces between units that updates the target positions of the units to keep them in a specific formation [3].

These IMs are parameterised to cover the range of behaviour that each unit can perform. 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 used by the *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.

In this paper we will just be using a simple straight-line *go-to* IM. In our previous work we discuss how inverse models can be optimised to reduce the number that need to be examined [4], however, for ease of analysis, we do not perform this optimisation here.

C. Forward Models

The forward models 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.

The engine of the game is based on Delta3D [15], which is an open-source project to integrate various software libraries into a coherent platform for simulation and games.

The 3D engine was used to model a large outdoor terrain (see the screenshot in Fig. 3), with the height and other features (texture, trees, buildings, etc) of the terrain being displayed based on 2D feature maps. The physics engine 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

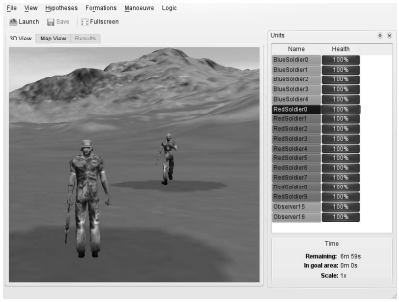

Fig. 3. Game screenshot showing the graphical interface for the human-controlled teams.

physics engine provides a stochastic element to the outcome of scenarios.

To support the use of the game engine as a forward model, it has to be able to transfer the current 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 is then 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.

D. Evaluation Process

After the inverse and forward model pair produce 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:

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

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 $d = 1$ metre 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 ($\max(|\vec{a}|, |\vec{p}|)$).

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 $\mathbf{b}$:

$$c_s^u = \max_i c\left(h_s^u(\mathbf{b}_i)\right) \quad (2)$$

E. Observations

We restrict the observability of the opponent's units so high-resolution sensors (observers) are required to obtain the exact positional information of the units for the purposes of making a prediction. To free us from having to perform an initial blind search for the units we assume that we have 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 [16], e.g., using the characteristics of the sensor and the environment to ensure coverage. It should be noted, however, that our 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 our 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 accuracy can be covered by the viewable area of the high-resolution observer.

F. System operation

At the start of the game, the clients are launched that host the inverse and forward models, which perform the predictions. The number of clients is limited to the amount of resources that are available, i.e. 2–3 clients per CPU core over several machines in our current non-optimised implementation.

An outline of how the predictive system interacts with the observers is shown in Fig. 4. First the initial position information for the units is gathered. The positional information for the targets (the 'blue' team) is fully observable

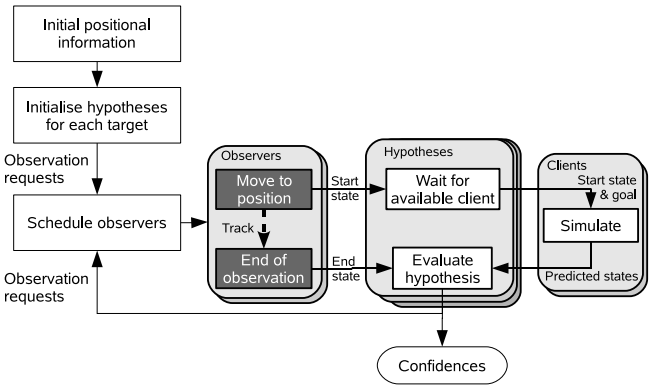

Fig. 4. Observation-based system operation

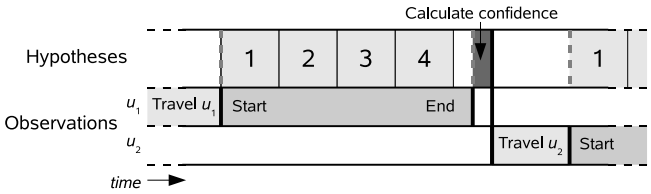

Fig. 5. Timing for observations using one observer, one client and four hypotheses

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 β for duration e. The scheduler takes these requests and decides which to attend to by assigning each observer a unit to visit (described in Sec. IV-G).

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 hypotheses. 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 d on the client (with a speedup of f) 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 Eq. 1 and the result is averaged over a time window to get the confidence for the hypothesis $h_s^u(\mathbf{b})$. 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 assigned a new observation target and the loop repeats. This loop continues until the end of the game, i.e., when all of the hypotheses become invalid because all of the targets have been destroyed. Fig. 5 shows the timing for one complete observation duration and the start of the next observation when using one observer, one client and four hypotheses.

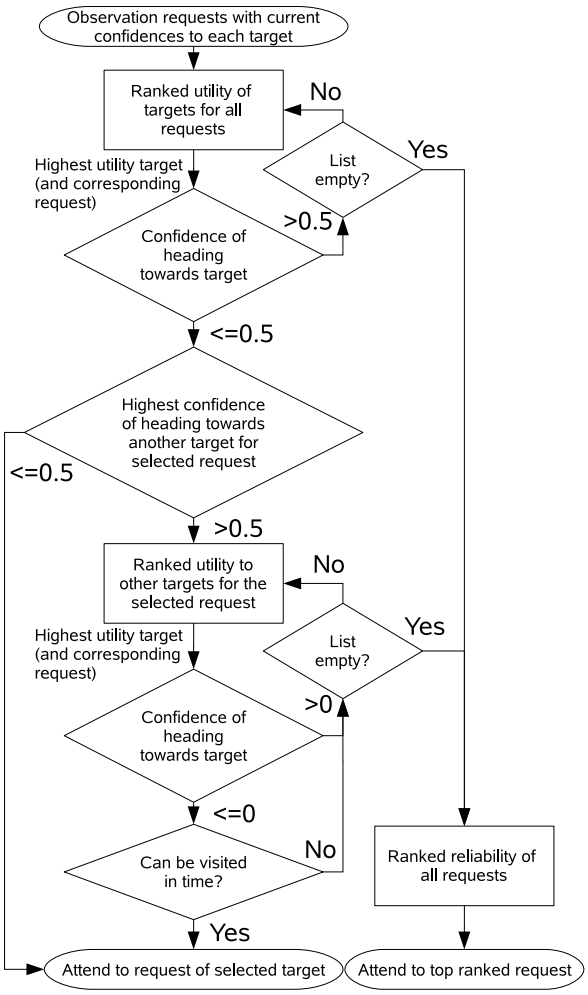

Fig. 6. Observation request scheduling flow diagram

G. Threat-Based Attention

The scheduler chooses the next group of agents to observe from the list of requested observations. Based on the objectives stated in Sec. IV we have devised a utility rating that caters for the criteria below. We shall refer to this implementation as a threat-based attention mechanism.

The criteria we are trying 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 we need to rank the targets 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 we also need to take into account the time it takes for the observer to reach the attacker to perform the observation. Therefore we rank the utility of the targets based on the *leeway*, which is the time difference between the observer reaching the attacker and the attacker reaching the target. In this paper we assume the attacker travels at their maximum speed and in a straight line (likewise for the observer). We could extend this to reuse the IM-FM architecture, but, when deciding where to send the observer,

we only have low-resolution position information available, so we would require a corresponding low-resolution forward model with simplified unit dynamics.

We also combine this leeway information 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 we are free to choose the attacker 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. We help mitigate this 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 we use the reliability of the observation 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.

We should note 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 Fig. 6.

V. Experiments

We set up a scenario to test the effects of partial-observability on a RTS-style game. 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 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 they 'kill' the target when they reach within 10m of it. The units have a maximum speed of 8m/s. The attackers also have a 1 in 10000 chance of re-evaluating their target choice on every frame, and the game operates at approximately 30 frames per second. Each trial runs until all of the targets are dead, or for 30 minutes, whichever occurs first.

The positions of the targets are randomly chosen for each run of the game, with the only constraint being that they lie within a 1000m^2 area and are at least 150m away from each other. The attackers are also randomly positioned within the same area but must be at least 300m away from each other and their targets. This makes the scenario more realistic as opposing units wouldn't start near each other in an RTS game—which 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 Fig. 7.

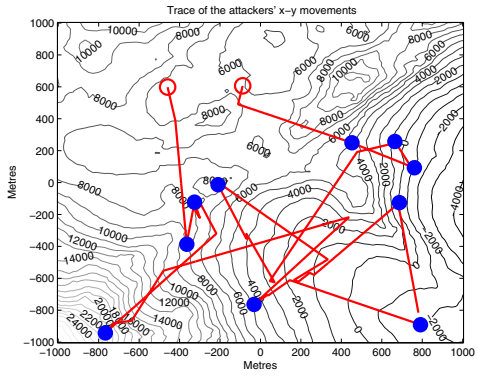

Fig. 7. Example scenario for two attackers (open circles) and ten targets (filled circles). The movements of the attackers are shown by the thick lines. The targets do not move throughout the scenario. A contour map of the terrain is shown in the background.

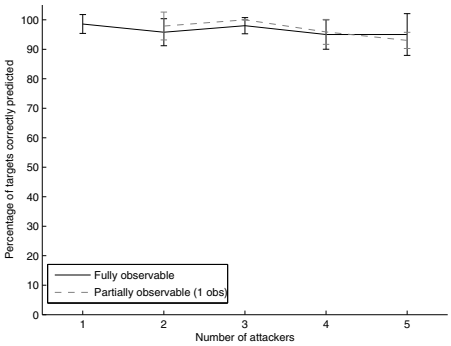

Fig. 8. No reduction on the average percentage of targets correctly predicted between full and partial observability for small numbers of attackers.

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 success of a game 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 Fig. 8, 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

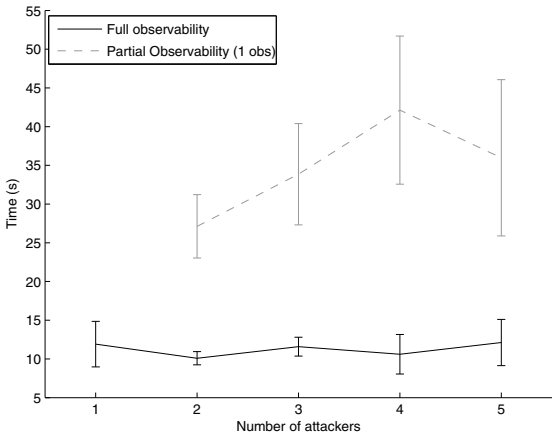

Fig. 9. Average time difference between an attacker changing target and getting a correct prediction is greatly increased for partial observability over full observability for different numbers of attackers.

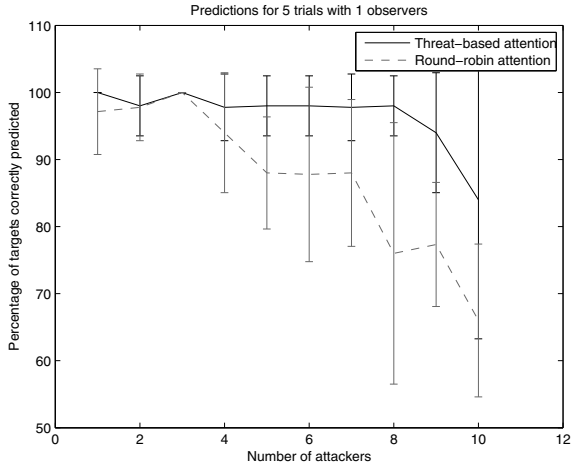

Fig. 10. The average percentage of targets correctly predicted with one observer, ten targets and different numbers of attackers is significantly better for the threat-based attention over the round-robin scheduler.

observability. This means the attacker could be heading to a target but then change and 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 Fig. 9, 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).

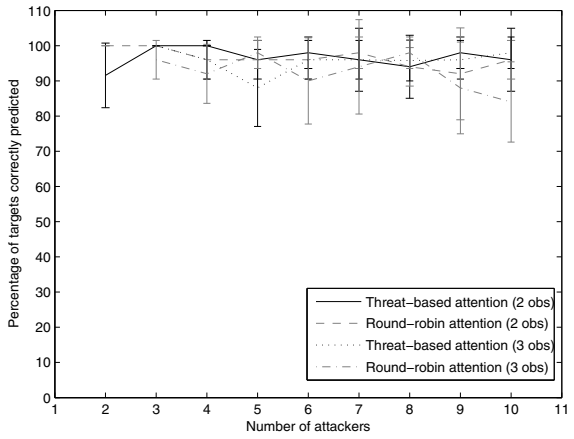

Fig. 11. The average percentage of targets correctly predicted with 2 or 3 observers, ten targets and different numbers of attackers is not affected by changing the attention mechanism.

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, for 1–10 attackers. As can be seen from Fig. 10, 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. With two or more observers the results shown in Fig. 11 hover around 90–100% for the ten attackers, and there is no noticeable difference between round-robin or threat-based attention.

VI. 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 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.

VII. Conclusion

In conclusion, we have extended our predictive architecture to be partially observable, hence the need to use observers to gather positional information about the opponent. We have shown how, in the scenario used in this paper, 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 we restrict ourselves to use just one observer, however, this is significantly helped by using our 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.

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 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 only a simple inverse model, the same architecture can be applied to any action the units can perform.

VIII. Future Work

Whilst the partial observability reduces the computational cost, further optimisations are needed within the forward model (such as optimising the tradoff between accuracy and speed in the physics engine) for this approach to become viable in a commercial game, using current hardware. 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.

In these experiments we simplified the system by using fewer attackers and targets than might be expected in a typical RTS game. The system could, however, easily be scaled up by assuming that opposing units travelled in groups, and the predictive system extended to take an average confidence over all of the units within a group.

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.

References

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

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

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

[4] ——, "Using a cognitive architecture for opponent target selection," in *Proceedings of the Third International Symposium on AI and Games*. SSAISB, 2010, pp. 55–61.

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

[6] L. Itti, "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.

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

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

[9] F. Southey, W. Loh, and D. Wilkinson, "Inferring complex agent motions from partial trajectory observations," in *IJCAI'07: Proc. of the int. joint conf. on Artifical intelligence*, 2007, pp. 2631–2637.

[10] S. C. J. Bakkes, P. H. M. Spronck, and H. Jaap van den Herik, "Opponent modelling for case-based adaptive game AI," *Entertainment Computing*, vol. 1, no. 1, pp. 27–37, January 2009.

[11] F. Sailer, M. Buro, and M. Lanctot, "Adversarial planning through strategy simulation," in *2007 IEEE Symposium on Computational Intelligence and Games*. IEEE, April 2007, pp. 80–87.

[12] J. Laird, "It knows what you're going to do: Adding anticipation to a Quakebot," in *AGENTS '01: Proc. of the int. conf. on Autonomous agents*, 2001, pp. 385–392.

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

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

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

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