2012

Real-Time Evolutionary Learning of Cooperative Predator-Prey Strategies

Mark Wittkamp

Luigi Barone

Philip Hingston
Edith Cowan University

Lyndon While

Real-time Evolutionary Learning of Cooperative Predator-Prey Strategies

Mark Wittkamp[1] **Luigi Barone**[1] **Phil Hingston**[2] **Lyndon While**[1]

[1] School of Computer Science and Software Engineering
University of Western Australia,
Crawley, Western Australia
Email: {wittkamp,luigi,lyndon}@csse.uwa.edu.au

[2] School of Computer and Security Science
Edith Cowan University,
Mount Lawley, Western Australia
Email: p.hingston@ecu.edu.au

Abstract

Despite games often being used as a testbed for new computational intelligence techniques, the majority of artificial intelligence in commercial games is scripted. This means that the computer agents are non-adaptive and often inherently exploitable because of it. In this paper, we describe a learning system designed for team strategy development in a real time multi-agent domain. We test our system in a prey and predators domain, evolving adaptive team strategies for the predators in real time against a single prey opponent.

Our learning system works by continually training and updating the predator strategies, one at a time for a designated length of time while the game us being played. We test the performance of the system for real-time learning of strategies in the prey and predators domain against a hand-coded prey opponent. We show that the resulting real-time team strategies are able to capture hand-coded prey of varying degrees of difficulty without any prior learning. The system is highly adaptive to change, capable of handling many different situations, and quickly learning to function in situations that it has never seen before.

Keywords: evolution, learning, multi-agent, predator-prey

1 Introduction

Games are often used as test-beds to further the development of computational intelligence techniques. They are suitable for this task because they involve similar problems to those encountered in real life, but are simpler and more clearly defined, generally with a well understood goal. Video games present a particularly interesting problem domain in that they typically have a far greater number of actions available for players to make and these actions have temporal significance. The development of adaptive behaviour using opponent modeling with evolutionary algorithms has been demonstrated before (Wittkamp 2006), (Wittkamp 2006), but the problem becomes much more difficult when we require the learning to occur in real-time, as the game itself is being played.

Artificial players that train offline (generally by playing the game) can have a near limitless amount of training time available to them. The learning and fine tuning of artificial players could run continuously for many days or weeks until desirable behaviours have been found. Contrast this with real-time learning, where there is very little time to run simulations and the processor must also be shared with the game engine itself. Computational intelligence techniques require many iterations and many more test cases for the evolution process to yield desirable results. In order for a real-time approach to be feasible, standard computational intelligence techniques will need to be sped up.

1.1 The Case For Real-Time Learning

Despite a large amount of research in the field of video game AI, the majority of AI strategy in commercial games is still in the form of scripted behaviour (Berger 2002). Developers turn to scripts for a number of reasons; they are understandable, predictable, easy to modify and extend, and are usable by non-programmers (Tozour 2002). Scripts often have parameters that may be optimised using computational intelligence techniques offline, but the learning aspect is rarely a component in the released product (Charles 2007).

While scripts can respond to the actions of human players, artificial agents (or "bots") are often inherently exploitable due to their inability to adapt. Once an agent's weakness has been discovered it can be exploited time and time again and soon the game fails to remain challenging or realistic and human players may lose interest. No matter how thorough the training process, in many modern games there are too many possible scenarios to expect that a hand-coded player will be able to handle them all equally well.

Scripted bots and their predetermined behaviour are susceptible to being overly repetitive or unrealistic, especially if the bots find themselves in a situation that the developers did not foresee. Stochastic systems can be used to introduce some variety into the behaviour of artificial players, but they may offer only slight variation to some predetermined strategy. Too much variation has the potential for creating seemingly random or irrational behaviour which adversely affects a human player's sense of immersion in the game environment.

Another common limitation of current game AI is

that teams of agents tend to be overly self-interested. While many good agents may be useful for a team, this is very different from team-interested agents who can understand and prioritise the good of the team over individual gain. Without team based learning, artificial players run the risk of being overly "greedy" to the detriment of the team. No matter how well the individual parts may be tuned, certain team strategies may never arise — a self-interested individual would not sacrifice itself to draw fire away from teammates or to lead opponents into an ambush, for example. Team based learning is useful where the goal to be accomplished is too complex to be achieved by individuals without team coordination, RoboCup soccer (Kitano 1997) is a good example.

The real-time learning and continuous adaptation of a team of artifical agents is desirable for a number of reasons. An agent capable of real-time learning would be inherently robust just as strategies learnt offline are inherently exploitable. Ideally, an adapting agent could be expected to perform in situations never considered by the game developers. Quinn et al demonstrated the use of a real-time evolutionary learning system for the task of cooperative and coordinated team behaviour for robots (Quinn 2002). The aim was for the team to move to a new location while remaining within sensor range of each other all times. Despite being a relatively simple task, it is an encouraging result.

Our previous work in the domain of Pacman (Wittkamp 2008) was a proof-of-concept study in "simulated" real-time — that is, the learning was continuous and took place in parallel with the agents acting in the environment, but the environmental simulation was paused to allow the learning system to explore strategies. This paper takes the next step and investigates to what extent sufficent learning is possible in real-time. We explore the use of computational intelligence techniques for real-time learning in a simple prey and predators domain. Focusing on team-work development, we examine how these techniques can be used to evolve strategies for a team of predators aiming to capture a single prey opponent.

The real-time system we propose makes use of continuous short-term learning to regularly update predator strategies. Our approach aims to parallelise offline learning through lookaheads and simulations with actual game play. Constant adaptation over short time periods means the predators need not learn complex general strategies, but rather focus all attention on current the state of the game.

2 The Iterative Real-time Team Learning System

Our real-time learning system is a novel implementation of an Evolutionary Algorithm, designed to run in parallel with the game environment and to iteratively evolve a team of agents via an analogy of Darwinian selection. Learning takes place continuously within discretised time slices; during each time slice, a role is selected for training.

The system first looks ahead to the predicted state at the start of the next time-slice (ES_{t+1}). This state is used to determine which role to train and from which population (each role maintains its own population). Each time-slice, a single role is trained in a round-robin fashion. How these roles map to the agents is up to the implementation, but for this study we use a direct one-to-one mapping of each role to a unique predator. It may be advantageous to organise the mapping of roles to predators in a more meaningful way (such as by distance to the prey) and then automatically switch the strategies used by predators as their circumstances change, but we plan to address these considerations in future work.

The lookahead state (ES_{t+1}) contains the expected state of the environment and all agents one time-slice into the future. The learning system has access to the predator strategies, and also the prey strategy — that is, simulations run have accurate models of how the enivironment and all agents contained will behave. When training a particular role, the role is replaced in the lookahead state and then a simulation from this state (ES_{t+1}) is completed. Even though only a single predator is traiing during any given time slice, the fitness measure used evaluates the team as a whole rather than sanctioning the individual directly. The individuals in the population are each evaluated by their contribution to the predator team's predicted performance at the end of the next time slice. The evolutionary algorithm uses this performance data to create the next generation of strategies.

The evolutionary process takes place in real-time, in parallel with actual events in the game environment. As many generations as possible are completed during the time slice, with the fittest individual from the evolving population being used to replace the role for play in the next time slice. We use the same fitness function as that of (Yong 2001) as described below where d_0 is the sum of all predator's starting distances to the prey, and d_e is the sum of the ending distances. The system is depicted visually in Figure 1 and written up as pseudo-code in Algorithm 2.1.

$$f = \begin{cases} d_0 - d_e/10 & \text{if prey not caught} \\ 200 - d_e/10 & \text{if prey caught} \end{cases}$$

We use an elitist selection scheme where the top half of the population reproduces by one-point crossover and mutation to replace the bottom half of the population. Mutation is applied randomly to a single weight of the individual, with 0.4 strength. We cap our simulation time at 600 game ticks (roughly 40 seconds). Though we are interested in completing a capture far sooner than that, we allow the simulations to run up to 600 game ticks for data collection purposes.

We allow our learning system to have access to a perfect simulation model. While playing, the predators do not explicitly communicate. For the lookahead and training simulations the predator currently undergoing a learning cycle has access to every other predator's agent model. That is, the learning system will have a perfect understanding of what each of its team mates will do in any given scenario and these are used to train a predator. This is possible due to predators being completely deterministic given any scenario.

Given the learning system's intimate knowledge to all agents' strategies and that the game environmemt is completely deterministic, the prey opponent model is the only remaining uncertainty in the lookahead and simulation process. In this paper, in order to completely remove noise in our simulations, we assume access to a perfect model of the prey opponent. Having a perfect opponent model is no small assumption, but the aim of this paper is to demonstrate the effectiveness of our real-time learning system compared to an offline approach. If our predators were learning offline by training against a particular prey, then the offline learning system would also have access to a perfect prey model. In a real-time scenario this may be infeasible because the opponent may be

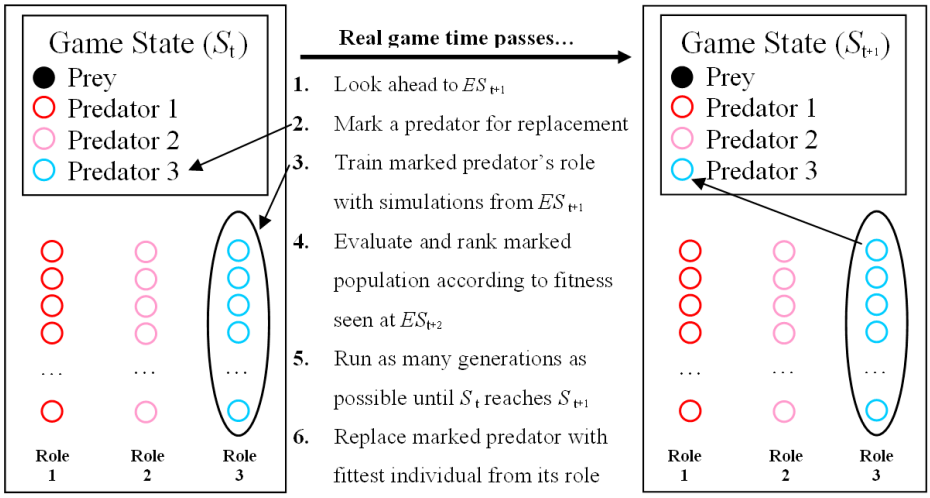

Figure 1: Pictorial representation of the real-time learning system

"black box" or simply not available for use in simulations — consider the case when playing against a human opponent in real-time. Section 6 discusses our intended future work with respect to inaccurate opponent models and other sources of simulation noise.

3 Experimental Domain

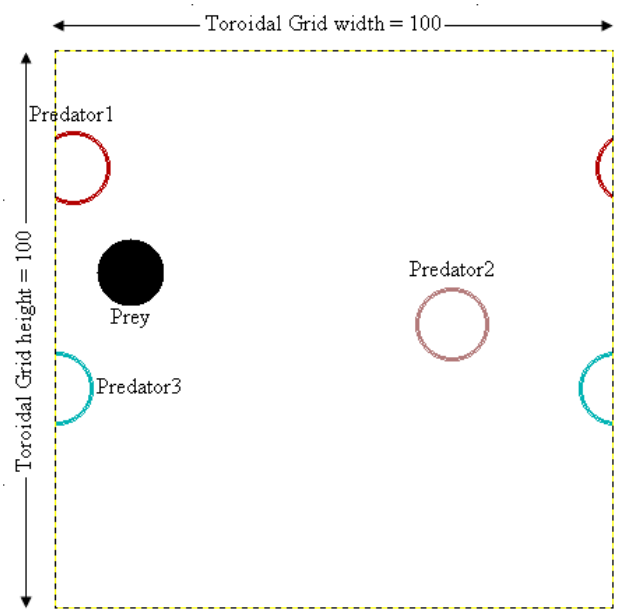

Figure 2: The prey and predators environment

We have developed a system for learning effective team strategies in real-time as a game is being played. We allow for no prior offline learning; all learning takes place while the game is being played. To test our system, we use the prey and predators domain studied in (Rawal 2010, Yong 2001). We are interested in evolving a team of predator strategies to coordinate their movements to trap and capture the prey in real-time.

3.1 Prey and Predators Environment

The game environment we use is closely modelled from that of (Yong 2001). In this predators-prey environment, we have a single prey and a team of 3 predators. The goal of the predators is to catch (making contact with) the prey. The prey's aim is simple; avoid being caught by the predators.

We are interested in training the team of predators in real-time to cooperate with each other towards the goal of catching the prey. In all but one experiment the predators and prey move at the same speed, thus making the task of capturing any competent prey impossible without some degree of cooperation — in the remaining experiment, the prey is given a powerful advantage by being able to move at 3 times the speed of the predators.

The environment for all experiments is a $100 * 100$ toroidal grid without obstacles where agents (prey and predators) are represented by circles of radius 6. In this an environment a simple hand-coded prey could quite easily evade 2 predators indefinitely, thus the task of capturing the prey will need the cooperative actions of all 3 predators working together. The initial setup places the 3 predators in a corner of the toroid grid (being a toroid, they are all one and the same) and the prey is randomly positioned. The speed of all agents is fixed — each is either moving at this speed or stationary; there is no in between.

A predator travelling across the toroid diagonally from corner to corner (the longest straight-line path across the toroid) takes 150 *game ticks*, which takes 10 seconds in real-time. This time was chosen as this seemed a realistic speed for the game if it where made to be playable by a human. What this means is that in the time taken for a predator to cover this distance, there are 150 decision points for every agent. The number 150 was chosen to match that of (Yong 2001) for which we aim to compare results.

Algorithm
2.1: Real-Time Evolutionary Team Learning System( )

comment: Initialise a population (P_r) of individuals for each identified role (r)
for each $r \in Environment.Roles$
do $\{P_r \leftarrow$ CreatePopulationOfIndividuals( )

for each $t \in Time - slices$
do
- **comment:** Capture the current state of $Environment$ to S_t
- $S_t \leftarrow Environment$.GetState( )
- **comment:** Look ahead from the captured state to the next expected state ES_{t+1}
- $ES_{t+1} \leftarrow$ LookAhead($S_t, OpponentModel$)
- $MarkedRole \leftarrow$ ChooseRole(ES_{t+1})
- **for** $g \leftarrow 1$ **to** $NumGenerations$
  - **in parallel for each** individual $i \in P_{MarkedRole}$
    - **do**
      - $StartStates[i] \leftarrow ES_{t+1}$
      - $StartStates[i]$.ReplaceRole($Environment.Roles[MarkedRole]$, $P_{MarkedRole}[i]$)
      - $ES_{t+2}[i] \leftarrow$ RunSimulation($StartStates[i]$)
  - **do**
    - **comment:** Evaluate $P_{MarkedRole}$ by inspecting expected end states (ES_{t+2})
    - Fittest $\leftarrow$ Evaluate($P_{MarkedRole}, ES_{t+2}$)
    - **comment:** Evolve the next generation of individuals for P_r
    - Epoch(P_r)
- $Environment$.ReplaceRole($Environment.Roles[MarkedRole], Fittest$)

3.2 Hand-coded Prey Controllers

In order to test our domain we have created some hand-coded opponents capable of evading the predators to varying degrees. The 3 different prey strategies we have created are *Simple*, *Repelled*, and *Fast*. These are listed in increasing order of how difficult they are to capture, as comfirmed by the results in Section 4.1.

1. **Simple:** our most basic of preys; its strategy is to always head directly away from the predator closest to it. This prey always travels at the same speed as the predators. The Simple opponent is based on the description of the prey opponent used in Yong and Miikkulainen's work (Yong 2001); we use this prey as a simple starting point and to allow more meaningful comparisons between our approaches.
2. **Repelled:** a more complex prey that aims to avoid predators proportionate to their proximity. For all predators, the prey applies a force of repulsion equal to $1/d^2$ in the direction of the predator, where d is the minimum toroidal distance from the prey to that predator. This prey moves at the same speed as the predators, heading in a direction determined by the sum of the repulsive forces. Our aim in creating the Repelled prey was to create a strong training partner for the bulk of our experiments, after initial experiments seemed to indicate that capturing the *Simple* prey did not sufficiently challenge our system.
3. **Fast:** a prey that employs the same strategy as the Repelled prey, but one that travels at 3 times the speed of the predators rather than at the same speed. This prey provides a very difficult capture task intended to push our learning system beyond its limits.

3.3 Predator Controller

A predator takes the form of a randomly initialised feed-forward neural network with 2 inputs, 5 outputs, and a hidden layer of size 10. The only inputs to the predators are their x and y toroidal distances to the prey. The predator's x and y coordinates on the toroidal grid do not factor into its decision making process. The outputs are *North*, *South*, *East*, *West* and *Stationary*.

The predator will remain still if the *Stationary* output exceeds that of all other outputs. Otherwise, the difference between the East and West outputs determines the x component of the predator's direction vector and the dfference between North and South determines y. The predator travels at a fixed speed, equal to that of the Simple and Repelled prey types (and one third the speed of the Fast prey). While this network representation could be used to define an agent that is capable of varying its speed, here we are only using it to describe the predator's direction, not magnitude. Like the prey, predators will always be either motionless or travelling at their predefined maximum speed.

4 Experiments and Results

4.1 Prey Strategy Evaluation

We have designed the Simple, Repelled, and Fast prey to be used as training partners to our real-time system. These strategies are described in Section 3.2. In this experiment we aim to confirm that the 3 prey strategies have a range of skill levels that make the problem increasingly difficult. We trial the prey against our real-time system with a fixed configuration. The experimental setup uses a population of 200, running for as many generations as real-time will allow — on average, the system made it through roughly 33 generations per time-slice.

Elapsed time n (game ticks)	Simple prey	Repelled prey	Fast prey
100	14	1	0
200	88	24	3
300	100	58	6
400	100	76	11
500	100	81	16
600	100	86	22

Table 1: Percentage of runs resulting in capture against various prey strategies by n game ticks.

Table 1 shows the results of each prey performing against our real-time adaptive predator team in an identical experimental setup averaged over 100 runs. The rates of capture are reported for various points of elapsed time and are therefore cumulative.

As expected, the Simple prey is the easiest strategy to capture. By 260 game ticks the real-time system managed captured the Simple prey in all 100 runs. Even at the time the simulations were capped at 600 game ticks, 100% was not achievable for this experiment against either the Repelled or the Fast prey, indicating that the Simple prey is clearly the least formidable opponent.

The real-time system took much longer to form an effective counter strategy to the Repelled prey than it took against the Simple prey. In the time that the Simple prey was completely dominated, the Repelled prey was only being captured 46% of the time, and reached an ultimate capture rate of 86% after 600 game ticks.

The Fast prey, employing the same strategy as the Repelled prey but at triple the speed, is clearly the most difficult prey to capture. This prey has the unfair advantage of being able to travel at 3 times the speed of the predators. The real-time system only manages to achieve capture in 22% of games after 600 game ticks — far lower than that achieved against the other hand-coded prey opponents. The aim in designing this opponent was to purposely create an extremely difficult task for our system; the results suggest that we have succeeded; this is indeed a very difficult prey to capture.

The results show that the real-time system is certainly very capable of producing effective predator team strategies in order to catch the prey without any prior learning. Within the time taken to move from one corner to the other (150 game ticks), the real-time controlled predator team manages to capture the Simple prey 60% of the time; this is a good result. Recall that 2 predators are not capable of capturing even the Simple prey and, due to the iterative learning construct of our system, it is not until after 3 time-slices (120 game ticks) that the real-time system has been given an opportunity to learn a strategy for each of the 3 predators.

This experiment's configuration was arbitrarily selected as a means of comparing the hand-coded prey strategies and to confirm that they are increasingly difficult prey to capture as intended. To observe a 100% capture rate being achieved after 260 ticks against the Simple prey is a most encouraging result.

The real-time evolved predator team manages to capture the more advanced Repelled prey strategy in 86% of cases and even manages to capture the Fast prey (a prey moving at 3 times the speed of the predators) 22% of the time. As previously mentioned, this experiment was not geared towards testing our hand-coded prey strategies and establishing a baseline, but rather towards achieving the most optimal configuration for learning. We expect our system's performance to improve in Section 4.2, when we aim to determined how the length of our time-slices affects learning performance.

4.2 Time-slice Experient

In this experiment, we investigate the effect of varying the length of the time-slice. The time-slice length affects both the rate at which new strategies are "plugged in" to the game as well how long the fitness evaluations are run. As the game progresses, learning takes place continuously across time slices, with each slice marking an insertion point for learnt strategies into the game.

Which length we use for the time slice has the potential to substantially impact the learning system. The length of the lookahead and the time taken until all predators have been given an opportunity to train are very significant factors both implied by the selection of a time-slice length.

Consider the case where we train using a time-slice of length 20. Random strategies are plugged in for all 3 predators and the training begins at 0 game ticks. The learning system looks ahead to the expected state of the game at 20 ticks, from here simulations begin in parallel for as many generations as time permits until the game reaches 20. At this point, the first predator strategy is inserted into the game and learning continues for the next predator which will be inserted into play at 40 game ticks, and then the final predator at 60. If our time-slice length was 40, then our learnt strategies would be more likely to see past myopic optima and be able to develop more effective long term strategies. However, we would be forced to wait until 120 game ticks until all predators had been given an opportunity to learn; an inherent tradeoff is seen.

In Figure 3 we see that all time-slice lengths except for the extremes of 20 and 200 managed to achieve a 100% capture rate after 600 game ticks against the Simple prey. When running our real-time system using a time-slice length of 20, 95% capture is achieved, and 96% for a time-slice length of 200. The fact that the system does not reach 100% capture after 600 game ticks under a time-slice length of 200 is not surprising at all. The 3rd predator strategy is only plugged in at 600 game ticks, meaning that only 2 of the 3 predators have had an opportunity to learn at the game's end. From 400 game ticks onwards the system is running with 2 learned predators and 1 predator still unchanged from its original random initialisation. Impressively, the high performance result shows that 2 trained predator strategies are able to make use of their randomly intialised team-mate.

The rate of learning for the real-time system appears to be the same against the Simple prey across all time-slice lengths. The general pattern we see is a

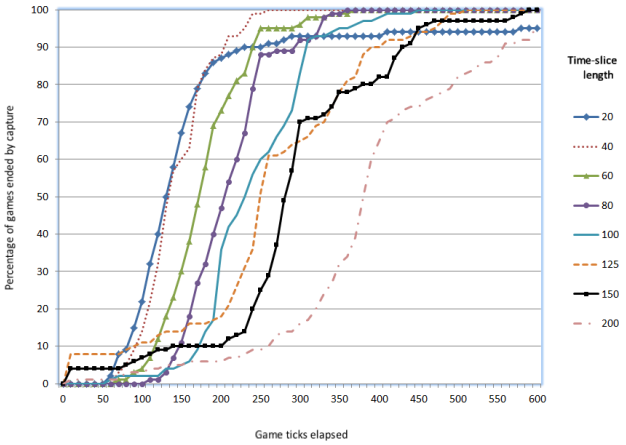

Figure 3: Effect of time-slice length on performance against Simple prey

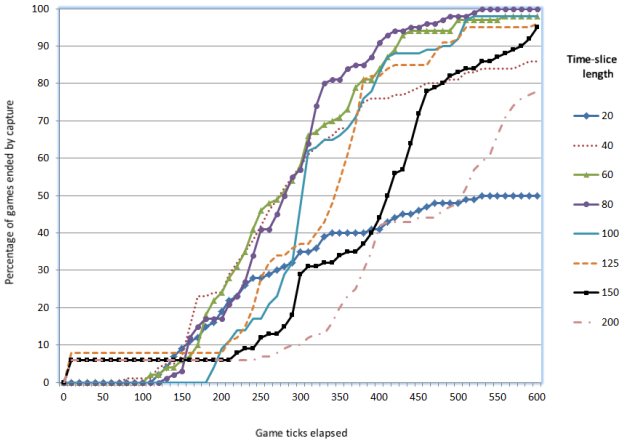

Figure 4: Effect of time-slice length on performance against Repelled prey

steep rise at about 3 times the time-slice length (once all predators have been given an opportunity to learn a strategy). The longer time-slice length runs suffer inherently because the time between insertions is longer, and the time until all 3 predators have learnt a strategy is also longer.

Against the Repelled prey, the most effective capture strategy at 150 game ticks was trained using a time-slice of 20, despite it not performing as well as the others, ultimately. A shorter time-slice length allows all predators to get through a learning cycle much sooner, but they will be limited in their understanding of the environment due to the short time-slice length. This almost always manifests itself in all 3 predators being bunched together and chasing the prey around the toroid over and over again as seen against the Repelled prey using a time-slice length of 20 in Figure 6. With an identical starting point (all predators beginning at the same location) but instead using a time-slice length of 80, after a few time-slices the predators learn to surround the prey and eventually capture as seen in Figure 7. While not as stark a relationship as we had expected, the tradeoff between forming effective strategies and the delay of actually being able to utilise them in the actual game is apparent.

If the time-slice is too short then the rewards of certain strategies are too distant to be recognised by the fitness function and thus will never be used as

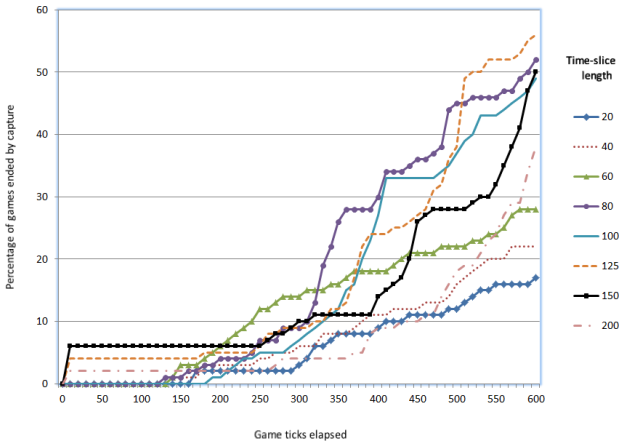

Figure 5: Effect of time-slice length on performance against Fast prey

predator strategies in actual play — i.e. the learning becomes trapped within local optima. For example, consider the case where all predators have converged upon chasing the prey and effectively acting as a single predator. If we now wish to evaluate a candidate predator strategy that breaks away from the other predators and head in the opposite direction in order to cut off the prey from the other side of the toroid. If the time-slice is too short then the fitness would be evaluated at a time where this predator had broken away from the others but had not yet caught up with the prey from the other side. The fitness function would then find this strategy to be ineffective because it is unable to see past the temporary hit to the fitness function required to make an improvement.

We observe that a steep improvement, relative to time-slice length, occurs earlier when the time-slice length is longer. One would expect to see a strong performance increase after the third time-slice because this is when all predators have had a chance to learn. With longer time-slices, we see that this increase occurs *before* the final predator strategy is plugged into the game. This suggests that with a longer time-slice to train under, the 2 predators are able to formulate strategies that are highly effective and able to make use of the randomly initialised strategy still being used by the third predator.

The time-slice is so long that at most only 2 predators have had a chance to learn By the time the final predator strategy has been trained and is ready to be plugged into the game at 600, the simulation is over. The trend seems to be that longer the time-slice, the better the resulting team-strategy. Also, the more difficult an opponent is to capture, the more benefit can be expected from increasing time-slice length. From Figure 5, against the Fast prey, we see a far more varied performance result at the end of 600 game ticks.

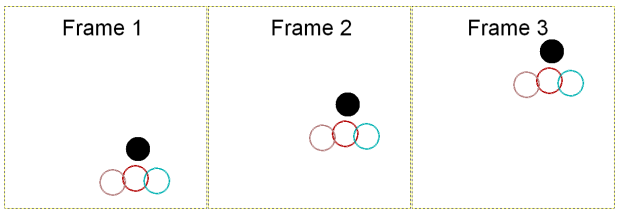

Figure 6: Short-sighted predator strategies fail to capture the Repelled prey using a time-slice length of 20 game ticks.

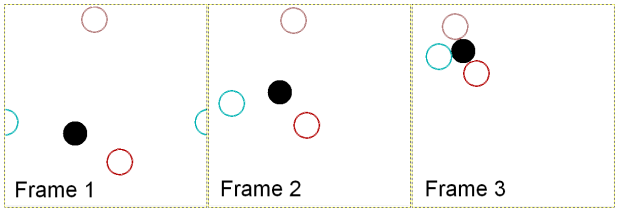

Figure 7: Predator strategies converge and capture the Repelled prey using a time-slice length of 80 game ticks.

For a counter strategy to the Simple prey, Figure 3, we observe that 20 ticks is not enough to produce an effective counter strategy. At 40 ticks, the predators are doing a lot better — reaching 100% capture rate after 260 game-ticks when training in real-time. Against the more difficult Repelled prey, 40 ticks was no longer enough to achieve a high capture percentage.

These results are in tune with what we would expect; with a more competent prey there would be more benefit to having a longer time-slice length. This is because a longer time-slice length means that predators are evaluated based on relatively long-term performance, encouraging and avoiding short term strategies that may be trapped within local optima. Indeed, this is exactly what we observe in the graph against the Simple opponent — we saw that a time-slice length of 20 game ticks was insufficiently short, and that a length of 40 seemed about right. Against the Repelled prey, we now observe that 20 ticks is far too short, achieving only a capture rate of 50% by the end of the game.

4.3 Real-time vs Paused

In this experiment, we compare our real-time system to a "paused" version of the same system. Our real-time system follows Algorithm 2.1, completing as many generations as it as able to within the time available.

The paused version works almost exactly the same, except it is guaranteed to complete a predetermined number of generations. This is possible because the game is paused in its execution at the end of each time-slice and waits for the learning to complete its desired number of generations. The paused version is able to tell us is how well we can expect our real-time version to perform given more powerful hardware that is capable of running more simulations in a given time.

Elapsed time (ticks)	Percentage captured
40	0.44
80	4.11
120	26.67
160	62.44
200	82.33
240	91.22
280	95.89
320	98.44
360	99.22
400	99.78
440	99.89

Table 2: Real-time adaptive prey's performance in the benchmark cases from (Yong 2001).

Tables 4.3 and 4.3 show the result of both the paused and real-time versions of our system playing against the Repelled and Fast prey strategy, respectively. What we found was that, there was no statistically significant difference between the paused and real-time systems in performance. The only statistically significant difference is in Table 4.3 against the Fast prey and for a population size of 10. In this instance, the paused version performs better.

These results are extremely encouraging; the real-time system manages to do just as well as the paused system against our best prey agents. What we found with the real-time version was that simulations were not very computationally expensive at all for this game, but the operations of creating a new generation from the previous one is. This explains why the real-time system manages to get through so many total simulations when the population size is high.

4.4 Comparison with Yong and Miikkulainen

Here we compare our real-time learning system with the offline approach of (Yong 2001). We train in real-time but then freeze our learnt strategies and pit them against 9 fixed starting states to assess how applicable the team strategies are to general situations. We model our game environment as close as possible to that of this study in order to make comparisons as meaningful as possible.

Yong and Miikkulainen experimented with a distributed and central control system for the prey. One conclusion made was that a distributed system was more effective than a centralised approach. Also, communication between predators was deemed unnecessary and that it overburdened the learning system. Without communication between the evolving predators, they learned faster and performed better with the emergence of more distinctive roles. The best predator strategies from (Yong 2001) were trained in the non-communicating, distributed system which we will refer to as the Yong-Miikkulainen system, from this point on.

Yong and Miikkulainen trained their predator teams using 1000 trials per generation, with each trial being 6 simulations against prey beginning in a randomly determined position. A layered learning approach was used by Yong and Miikkulainen, which saw predator strategies being evolved in 6 stages: from a stationary prey, to incrementally faster prey until reaching the same speed as the predators. Our real-time system does not have the time to implement a layered learning approach such as this, so our system must tackle the full speed prey immediately. Once learnt, the strategies were tested in a set of 9 benchmark cases to determine how effective the predator team is at capturing the prey. These benchmark cases involve the predator teams all beginning in the corner of the toroid, with the prey beginning at 9 evenly spaced starting positions on the toroid. The 9 positions the centre-positions of each sub-square when the toroid is split up into a $3 * 3$ grid.

Table 4.3 shows the result of our real-time adaptive predator team when thrown into the same benchmarking as used in (Yong 2001). In these benchmark cases, the Yong-Miikkulainen system took an average of 87 generations to be able to solve 7 out of 9 benchmark cases, and this was done within 150 game ticks. At 150 ticks our system averaged 54% capture. Our real-time system falls slightly behind in this result, managing to achieve 7 out of 9 a bit later at 190 game ticks.

When compared in this way the real-time system falls behind. At each time-slice of 40 ticks one predator is given the opportunity to train so it is not until 120 game ticks that the real-time system is operating with a full set of learnt predator strategies. The team trained under the Yong-Miikulainen approach

	Population size	Generations	Total simulations	Average capture time
Paused	10	500	5000	348.63
Real-time	10	48.81	488.61	336.21
Paused	50	100	5000	275.61
Real-time	50	45.64	2281.88	309.09
Paused	100	50	5000	315.72
Real-time	100	42.31	3287.38	311.84
Paused	200	25	5000	293.53
Real-time	200	32.87	8461.38	307.91
Paused	500	10	5000	296.41
Real-time	500	25.23	12612.86	307.88
Paused	1000	5	5000	274.03
Real-time	1000	11.36	11358.83	272.84
Paused	2500	2	5000	271.29
Real-time	2500	5.91	14781.84	281.39
Paused	5000	1	5000	269.76
Real-time	5000	2.24	11204.0	274.86

Table 3: Real-time vs Paused, against Repelled prey

	Population size	Generations	Total simulations	Average capture time
Paused	10	500	5000	510.97
Real-time	10	49.94	499.41	557.95
Paused	50	100	5000	536.61
Real-time	50	46.09	2302.43	518.63
Paused	100	50	5000	503.53
Real-time	100	43.13	4312.98	515.86
Paused	200	25	5000	503.23
Real-time	200	36.97	7393.77	521.36
Paused	500	10	5000	525.86
Real-time	500	23.68	11840.86	498.61
Paused	1000	5	5000	467.13
Real-time	1000	14.33	14326.31	500.19
Paused	2500	2	5000	498.14
Real-time	2500	5.62	14043.09	470.78
Paused	5000	1	5000	499.63
Real-time	5000	2.11	10557.44	532.60

Table 4: Real-time vs Paused, against Fast prey

has all 3 predators plugged in from the start because of the offline learning it has already undertaken. If we consider the elapsed time of 270 ticks $(120 + 150)$ so that our real-time system has had the opportunity to play the same number of game ticks with a full team of predator strategies, then it achieves a 95.67% capture rate.

The Yong-Miikulainen predator team had its population of 1000 individuals run through an average of 87 generations 6 times (each evaluation consists of play in 6 random games). The total number of simulations to achieve 7 out of 9 capture in the benchmarks is 522000. The moment when our real-time system achieves 7 out of 9 in the bench mark cases is at 190 game ticks. At this time, the system has run our population of 2500 individuals through an average of 6.1 generations 4 times (once for each time-slice that has completed). This is a total of about 61100 total simulations, fewer than one eighth that required by the best predator team of (Yong 2001).

Our system uses the same neural network inputs and outputs as (Yong 2001) and the same training partner (the Simple prey), and achieves the same performance in less than one eighth of the simulations, and in real-time. The one weakness of our system is that the initial learning takes time to slot predator strategies into the game. While (Yong 2001) finished their simulation at 150 game ticks, their predators were all employing their strategies from the onset. At 150 ticks, our system averaged 54% captures but, of course, has spent a considerable portion of these ticks learning strategies for its predators.

4.5 Generalisation

In Section 4.3 we demonstrated that our real-time system was able to evolve team strategies to play with no prior learninng.

In this experiment, we allow the predators to evolve in real time (one after another in their allocated time-slices) and then "freeze" the predators' learning once the game is over. The "frozen" (no longer learning) predator strategies are then tested in a number of different starting configurations. This experiment is to determine how effective the frozen predator strategies are for the game in general.

We take the strategies learnt in real-time from Section 4.2 and freeze their learning after the game has come to an end. The predators are placed in 9 new game environments, with only the starting position of the prey being varied in each in order to assess the team's ability for general play in the prey and predators domain against an identical prey. The 9 prey starting positions are such that if the toroid were to be divided into a $3*3$ grid as described in Section 4.3, the prey begins in the center of each cell. We run trials against all 3 prey types, each time training against that particular prey in real-time and then testing for

general ability against that same prey in each of the starting positions.

Averaged over 100 runs of the real-time training, we recorded the total capture rate at various points in the game's play time. The results across all 3 prey types are similar — the strategies learnt in real-time completely fall apart when frozen and placed in the 9 new game environments. Not a single capture was achieved in any experiment, even in those where real-time learning routinely achieved perfect or near-perfect capture. While this may at first seem like a negative result, it is exactly what we expected to see.

When training in real-time, the predators vary their strategies to restrict the prey, and slowly surround and close in on the prey. At the time of capture, at least 2 of the prey (and most often, all 3) converge on the strategy of heading directly towards the prey. The reason this strategy is effective at the end is due to the higher level strategies of surrounding and restricting the prey's movement that were learnt earlier and have since been discarded. When these predators' highly specialised end-game strategies are then frozen and placed in new game environments, what typically results is the same endless traversal of the toroid that we observed in Section 4.2.

The benefit of our real-time system is that the strategies formed are not necessarily robust or sound strategies for the game in general and can thus be simple to learn. A general strategy must be complex enough to deal with every game situation it may encounter; learning in real-time through continuous adaptation allows the system to learn highly situation-specific strategies without being overburdened by being required to learn how to play in all other situations. The predator strategies that are evolved in real-time are constantly changing to match the current state of the game environment. To develop this level of specialisation offline for all possible game scenarios would require far more learning and a more complex predator representation. The team strategies learnt in real-time always perform badly in the 9 scenarios due to the final strategy that the predators had at the time they were frozen.

The resultant strategies are bound to the specific game scenario at the time it is encountered. The team strategies become so specifically tailored to the task at hand that any hope for generality is lost. Depending on the domain, a system that behaves well in general may be very difficult to create and may not even be possible. A real-time learning system that can change itself to new game conditions alleviates the need to solve such a difficult task, when all that is required is for the predators to be able to focus on what it needs to do, when it needs to do it.

5 Conclusion

The results of this study are extremely encouraging. We have shown that the real-time team strategy is able to learn, in a reasonable amount of time to capture hand-coded prey of varying degrees of difficulty. It is capable of achieving competitive results with the paused version.

Not surprisingly, the strategies formed to not make for very robust, general strategies. The strategies formed by the real-time system are extremely specialised to whichever situation currently presented to it. The strength of this system is in its ability to adapt. This real-time learning system simulates a higher level strategy capable of handling many different situations, and indeed, situations that it has never seen before.

6 Future Work

For this paper, we have shown that our system is capable of discovering real-time cooperative strategies for the task of controlling predators in the Prey and Predators domain. The results of these experiments depend on the learning system's access to a perfect opponent model. This is ultimately an unrealistic assumption for the problem that we wish to extend our approach into — play against a human-controlled prey opponent. How accurate must our opponent model be before our learning system is able to form effective team counter-strategies? How (and when?) do we try to improve our opponent model in real-time? In future work, we intend to investigate the answers to these questions among others.

For ease of implementation, one thing we have done is to run simulations for the length of a time-slice. This means that the predator strategies are trained to play for the length of one time-slice but because there are 3 predators, these strategies are actually in play for 3 time-slices. Since a long time-slice length hinders the initial learning curve so much a possible solution would be to have initially short slices which could grow one all predators have had the opportunity to learn at least some sort of strategy. Or, rather than being discretised to these time-slices, it may be beneficial to have an "any time" approach to when new strategies could be plugged into the game.

Intuitively, it seems to make sense to train in the environment that one wishes to play in, however there are several reasons why this may not be the best or only solution. For real-time learning we are greatly restricted in how much training we can get through; increasing the simulation length will put even more strain on our system and limit how many simulations we can get through. When combined with our goal to account for noise in the opponent model, a longer simulation length may be far too noisy to provide any benefit at all. A successful approach may be one that has a variable time-slice length depending on the perceived accuracy of the opponent model.

References

M. Wittkamp and L. Barone: Evolving adaptive play for the game of spoof using genetic programming, in Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games, IEEE Publications.

M. Wittkamp, L. Barone, and L. While: A comparison of genetic programming and look-up table learning for the game of spoof, in Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games, IEEE Publications.

Wittkamp, M. and Barone, L. and Hingston, P.: Using NEAT for continuous adaptation and teamwork formation in Pacman. In: Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On, pp. 234–242, Perth.

Quinn, L. Smith, G. Mayley, and P. Husband: Evolving teamwork and role allocation with real robots, In Proceedings of the 8th International Conference on The Simulation and Synthesis of Living Systems (Artificial Life VIII), 2002.

ML. Berger: Scripting: overview and code generation, in AI Game Programming Wisdom. MIT Press, 2002, vol. 1, pp. 505510.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa: RoboCup: the robot world cup initiative, in Proceedings of the First InternationalConference on Autonomous Agents (Agents'97). ACM Press, 58, 1997, pp. 340347.

P. Tozour: The Perils of AI Scripting, Charles River Media, Inc.

Yong C. and Miikkulainen, R.: Cooperative Coevolution of Multi-Agent systems, University of Texas, Technical Report, 2001.

Yong C. and Miikkulainen, R.: Coevolution of role-based cooperation in Multi-Agent systems, IEEE Transactions on Autonomous Mental Development, 2010.

Rawal, A. and Rajagopalan, P. and Miikkulainen, R.: Constructing Competitive and Cooperative Agent Behavior Using Coevolution, 2010.

Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences, 1981. J. Mol. Biol. 147, 195–197.

May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg.

Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure, 1999. Morgan Kaufmann, San Francisco.

Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, 2001, pp. 181–184. IEEE Press, New York.

Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration, 2002. Technical report, Global Grid Forum.

D. Charles, C. Fyfe, D. Livingstone, and S. McGlinchey: Biologically Inspired Artificial Intelligence for Computer Games, Medical Information Science Reference, 2007.

National Center for Biotechnology Information: `http://www.ncbi.nlm.nih.gov`