

The University of Birmingham  
School of Computer Science  
MSc in Advanced Computer Science

Mini-project

**Investigating the Drivers of  
Co-evolution in a Population of Agents**

John S. Montgomery  
msc37jxm@cs.bham.ac.uk  
Supervisor: Dr L. Jankovic

January 11, 2004

## **Abstract**

To investigate the co-evolution of agents in an artificial environment I have written a physically based simulation, that uses rigid body mechanics. In this environment agents receive input from sensors that provide mock sensory data and using a simple, evolved, neural network determine how to activate their effectors to propel themselves on a two dimensional plane. Evolution is controlled by a genetic algorithm that is periodically called as the simulation runs. This affords us a great deal of top-level control of and feedback from the simulation, but as the simulation is ongoing the agents still exist in an "ecology".

I have focused on the competitive co-evolution of two species in a predator-prey dynamic. Various simulations were performed with differing simulation and species parameters to see what affected the ability of each species to compete effectively.

It was found that the predators in general had a much harder task than the prey and so were often unable to compete effectively, sometimes even with major advantages. This has been attributed to the inability of the neural networks topologies used to foster the detection of motion. This in turn meant that the predators were unable to effectively intercept their prey. This paves the way for future extensions that utilise more sophisticated topologies and better methods for evolving neural networks.

### **Keywords**

Co-evolution, Agents, Physical Simulation, Predator Prey Dynamics, Braitenberg Vehicles, Genetic Algorithm.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Simulating Co-evolution . . . . .	4
1.1.1	Top Down Simulation . . . . .	4
1.1.2	Emergent Simulation . . . . .	4
1.2	Top Down and Emergent Simulation . . . . .	5
<b>2</b>	<b>Methods</b>	<b>6</b>
2.1	Physical Simulation . . . . .	6
2.2	Dealing with Collisions . . . . .	7
2.3	Simulation Performance . . . . .	7
2.4	System Ecology . . . . .	7
2.4.1	Agents . . . . .	7
2.4.2	Food and Rocks . . . . .	9
2.4.3	Genetic Algorithm and Fitness . . . . .	10
2.5	Experimental Tools . . . . .	10
2.5.1	Simulator . . . . .	10
2.5.2	Data Presentation . . . . .	11
<b>3</b>	<b>Results and Discussion</b>	<b>12</b>
3.1	Simple Agents . . . . .	12
3.1.1	Run 1 to 3 . . . . .	12
3.1.2	Run 4 . . . . .	13
3.1.3	Run 5 and 6 . . . . .	13
3.2	More Complex Agents . . . . .	18
3.2.1	Run 7 . . . . .	18
3.2.2	Run 8 . . . . .	21
3.2.3	Run 9 . . . . .	21
3.3	Interpretation of Results . . . . .	24
<b>4</b>	<b>Conclusion</b>	<b>26</b>
4.1	Possible Extensions and Improvements . . . . .	27
<b>A</b>	<b>Mini-project Declaration</b>	<b>30</b>

<b>B</b>	<b>Statement of Information Search Strategy</b>	<b>32</b>
B.1	Forms of Literature . . . . .	32
B.2	Appropriate Search Tools . . . . .	32
B.3	Search Statements . . . . .	32
B.4	Search Evaluation . . . . .	32
<b>C</b>	<b>Simulation Application and Source Code</b>	<b>33</b>

# List of Figures

2.1	Agent Sensors . . . . .	8
2.2	Agent Neural Network . . . . .	9
3.1	Fitness: Run 1 (Predator Solid Line, Prey Dotted Line) . . . . .	14
3.2	Fitness: Run 2 (Predator Solid Line, Prey Dotted Line) . . . . .	15
3.3	Fitness: Run 3 (Predator Solid Line, Prey Dotted Line) . . . . .	16
3.4	Fitness: Run 4 (Predator Solid Line, Prey Dotted Line) . . . . .	17
3.5	Fitness: Run 5 (Predator Solid Line, Prey Dotted Line) . . . . .	19
3.6	Fitness: Run 6 (Predator Solid Line, Prey Dotted Line) . . . . .	20
3.7	Fitness: Run 7 (Predator Solid Line, Prey Dotted Line) . . . . .	22
3.8	Fitness: Run 8 (Predator Solid Line, Prey Dotted Line) . . . . .	23
3.9	Fitness: Run 9 (Predator Solid Line, Prey Dotted Line) . . . . .	25

# Chapter 1

## Introduction

### 1.1 Simulating Co-evolution

There has been a lot of work on simulating agents co-evolving in artificial environments [6] [8] [9] [10]. These simulations attempt to avoid the "over-simplicity of game-theory models" [8], but still make themselves amenable to analysis. I shall summarise the two broad classes of such simulations and discuss how I intend to combine approaches from both.

#### 1.1.1 Top Down Simulation

Due to the finite nature of computing power it is often the case that these simulations are deliberately specific. They often involve evaluating individuals for specific behaviours in a manner similar to a genetic algorithm [8] [6] [9]. The simulations impose on the agents a set of desired behaviours (e.g. pursuit and evasion). They can thus be considered as "top down" simulations, where the expected "correct" behaviour is known (at least to some extent) ahead of time.

In Reynolds [6], Cliff & Miller [8] and Sims [9] agents are paired off and compete directly against each other. There is a definite "fitness function" used to rate how well each agent has performed. Multiple evaluations may occur, but there is still a definite idea of what the agents are meant to do. The fitness function can then be carefully crafted to apply as much evolutionary pressure as seems necessary. This approach has the advantage of more *efficient* use of the available resources and can still lead to interesting and unexpected results, but it does still involve "many trade-offs" when "fitness is determined by discrete competitions between individuals" [9].

#### 1.1.2 Emergent Simulation

The other approach is to simulate as much as possible and leave the simulation open ended, with the hope that we shall observe interesting emergent behaviour. This is best typified by PolyWorld [10] in which apart from an "ad hoc fitness

function” the agents themselves determine when they eat, breed and/or fight. As so little *correct* behaviour is specified this has the potential to produce very interesting agents, but as with the open ended nature of real evolution it works best with plenty of individuals and a lot of time. It can also be very difficult to figure out what has actually happened to cause what we have observed. Braitenberg calls this the ”law of uphill analysis and downhill invention” [2], because it is easy to take lots of simple pieces and put them together, but it can be very hard to work out how they interact exactly.

Tierra [13] is another example of an emergent simulation, but is concerned with evolving artificial organisms represented as programs on a virtual machine. As of such it is more akin to evolving virii than braitenberg style agents. However it does have the advantage of being truly open ended, as the organisms very structure is open to evolution, but still surprisingly open to analysis, again due to the organisms structure.

## 1.2 Top Down and Emergent Simulation

The intent of this work is to simulate agents somewhere on the middle ground between the two approaches (Top Down and Emergent). Unlike the top down simulations [8] [6] [10] [9] the evaluation of an individual agent will not be a one time thing, but on going. However unlike PolyWorld [10] agents only have to worry about one thing - staying alive (through securing energy). Also unlike PolyWorld [10] which sometimes had trouble producing ”Evolutionary Stable Strategies”, dead agents will not be removed from the simulation immediately, they will merely have a higher chance of being removed and a lower chance of being selected for generating offspring.

In effect I am using a genetic algorithm to control the evolution of populations of agents in an ongoing simulation. One could view it as a sort of programmed grim reaper and baby carrying stork, which is to say that life and death are controlled by entities external to the agents. This affords us a greater deal of ”top down” control over the populations, hopefully letting us rein in some of the complexity in the simulation.

I have also chosen to study a classic predator-prey dynamic. As of such there are two separate populations that cannot interbreed, with one population designated as herbivore and one as carnivore (which implicitly defines their roles). The predator-prey dynamic is ”important, interesting and useful” [8], but still fairly simple to comprehend and therefore analyse.

# Chapter 2

## Methods

### 2.1 Physical Simulation

The physical simulation consists of relatively simple rigid body mechanics [11] [3] [4] [5] of Braitenberg [2] style "Vehicles" (Agents) Each Agent (Vehicle) in the simulation has position, orientation, mass, moment of inertia, momentum and angular momentum.

At each time step the effects of applied forces and collisions (impulse forces) are calculated and the state of the system is updated. Each physical body (Agent) is also slightly damped and has friction applied to it, to help things come a halt when no forces (internal or external) are applied to it. This also helps with small inaccuracies in the simulation, as they will not have the chance to cause too much damage.

The Agents have a pair of "motors" that can be used for movement. The motors are placed around the edges of the Agents boundary and are always aligned with the Agents current direction. By using the motors either co-operatively or antagonistically the Agents can move around and change direction, in a manner similar to a tank. That is to say if the motors fire in opposing directions the Agents will turn on the spot, but if the motors fire in the same direction the Agent will move in a straight line. Currently the motors are the only way that an Agents direction can be changed. This is because when collisions occur the Agents circular shape and the fact that friction is not considered during a collision means that all other forces applied to an Agent, currently, act through the centre of gravity and so have no angular effects.

One of the virtues of using rigid body mechanics is that would be trivial to add more motors to an Agent, possibly positioned at different angles. In particular it would open up the possibility of evolving an Agents morphology. Evolving motor placements and body shapes would present an interesting proposition for studying co-evolution.

## 2.2 Dealing with Collisions

The Agents can collide with each other, as well as with static objects (food, rocks etc) and the boundaries of the environment. For a simulation with 200 Agents and roughly the same number of static objects this presents quite a few collisions to check. In general for  $N$  agents and  $M$  static objects we would require  $(N^2 - N)/2 + NM$  collision checks [7]. For  $N$  and  $M$  equal to 200 this would mean 59,900 collision checks. Although an individual collision detection is not particularly costly (being a simple distance computation) having a large number of them slow things down a lot.

In a typical, non-pathological, simulation we would expect the Agents and static objects to be more or less uniformly distributed. We would therefore expect only relatively small proportion of the objects in the simulation to be colliding (or near collision). We can exploit an objects spatial location to help us only consider checking collisions between objects that we already know to be fairly close.

The simulation space was into a grid [7]. Objects are then placed into whichever grid square they overlap with (possibly multiple squares). Then collisions are only considered between objects that are in the same grid square. A little extra work is also needed to avoid repeated collisions, but this is fairly trivial.

Using this technique we would expect and do achieve better scalability. For example with  $N$  and  $M$  equal to 200, as before, and with a 10 by 10 grid (forming 100 squares) we might expect 5 or so static objects and agents in each square. This would mean for each agent we would only be checking for collisions with 4 other objects (on average). This is a slightly optimistic case, but it would mean something closer to 1000 collision checks, which is much better than the 59,900 previously needed.

## 2.3 Simulation Performance

For most of the simulations I have been running performance has been "better than real-time". In the simulation 25 simulation steps equate to 1 second in simulated time. Even running on relatively out of date hardware (Pentium 2 450Mhz/G3 400Mhz) simulations with 200 agents and 200 fixed objects run nearly three times faster than real-time. That is to say 180 hours of simulated time takes just over 60 hours to complete. On the departments cluster running times are closer to 20 hours.

## 2.4 System Ecology

### 2.4.1 Agents

Each agent consists of a simple circular rigid body. In the simulations I have run each agent has two motors which it can use to move itself around its world

with. They also have had two "sensor groups", roughly analogous with eyes. Each sensor group comprises one light sensor and two colour sensors. Each light sensor gives a stronger output when something is nearer to it (based on the square of the distance of the object). The colour sensors work in a similar way, but their output is additionally modified by the colour of the object, based on how closely that objects colour matches the sensors colour. The two types of sensors can then be considered roughly equivalent to rod (light sensors) and cones (colour sensors) in an eye. Each sensor group is given an equal proportion of the agents *field of vision* (Figure 2.1). A sensor group would only "see" objects inside its part of the field of vision.

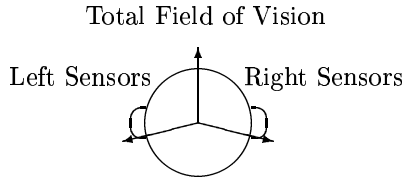


Figure 2.1: Agent Sensors

The six sensory inputs and two outputs are controlled by a fully connected feed-forward neural network. The network is also given an additional fixed input, to help it produce some output even when the sensors are not registering anything. The neurons are simple summation units (with a bias). This was done to help create networks that respond in a similar way to those in Braitenberg. In addition to a bias on each neuron there is also a weight on the connection between neurons. All biases and weights were only modified through evolution, no learning occurred during an agents lifetime.

Early simulations made use networks with only input and output neurons - no intermediate or hidden neurons. Later simulations added four hidden neurons (Figure 2.2). Due to the fully connected nature of the networks even this handful of hidden neurons greatly increased the size of the agents genome (from about 20 values for a network, to 44 values). This was one of the problems of such a simple architecture. Also the fact that non-recurrent networks were used may limit the potential behaviour of agents. For example they would be unable to detect differences between successive inputs - limiting their ability to notice movement.

As well as being able to evolve their neural networks agents also had several other parameters that could be evolved. Among these were sensor colours (which colours sensors were affected by), sensor sensitivities, sensor angle, and motor angle. As these parameters were relatively high level they were convenient for

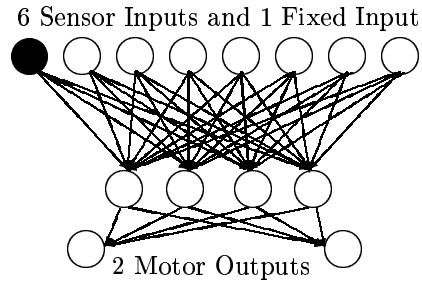


Figure 2.2: Agent Neural Network

recording throughout a simulation. For example by recording the average sensors colours and sensitivities of a population we could hopefully draw conclusions about what it was "interested" in. Sensor angle (the agents field of vision) was also of interest, because it might reveal parallels with real the real evolution of predators and prey, where prey tend to have wide fields of vision and prey tight, forward looking, fields of vision. As opposed to the "black-box" nature of a neural network, these parameters can prove useful in a simulation such as this.

After this there were several user definable parameters. These include motor power and sensor range. Typically predators were given slightly more motor power than prey, so that they have a chance of catching them, even if the prey moves directly away from them. An agents colour (e.g. red for predators and blue for prey) and eating habits (carnivore or herbivore) could also be specified.

### 2.4.2 Food and Rocks

The agents environment also contained a number of static objects. There were two types of object: "food" and "rocks".

Rocks are simple static objects that are grey in colour. They only serve to get in the way of agents, possibly trapping the unwary. When an agent collides with a rock its energy is reduced slightly. The intent being that this should be enough to dissuade agents from running into rocks.

For carnivores a piece of food is equivalent a rock, except green in colour. Only herbivores can "eat" food. When a herbivore collides with a piece of food the food is removed from the simulation and the herbivores energy is increased by a fixed amount. The amount of food in the simulation is kept constant, so whenever a piece is eaten a new piece will be placed randomly later.

Conversely when a carnivore collides with a herbivore energy is removed from the herbivore (at least half) and given to the carnivore. The herbivore is then randomly displaced to another point in the simulation, representing the fact that it has been "killed".

### 2.4.3 Genetic Algorithm and Fitness

While the simulation runs a pseudo steady-state genetic algorithm (ref Mel Mitchell) is applied to each population of agents (e.g. the predators and the prey). Periodically the populations are evaluated. Next either the worst or a random member of the population is removed, then two parents are selected. The parents are selected based on their fitness and are recombined (with a certain probability) and the single offspring is mutated (again with a certain probability). The offspring then joins the "juvenile" segment of the population, which means it does not get selected for removal or breeding initially. Then the juveniles from the previous selection are promoted to "adulthood", so that they can be selected next time round. In this way the agents gain an initial grace period in which they can "catch up" with the rest of the population and not be at too much of a loss just because they.

The fitness value for an agent can be one of two values, either 1 or -1. If the agents *energy* is 1 unit or above then their fitness is 1. It is -1 otherwise. The two values correspond to whether an agent is "alive" or "dead". By not immediately removing agents from a simulation when their energy is low evolution is, hopefully, able to have a smoother ride. This is because the fitness only applies to that one point in time and so having a more continuous fitness function might prove misleading as some agents may simply be lucky (or unlucky). The best bet for an agent then is to remain "alive" for as long as possible, not necessarily to get lots and lots of energy. Also because of the agents metabolism uses up their energy, an agent must seek out more energy otherwise it will quickly wind up dead.

Mutation was fairly straightforward. The only interesting caveat was when mutating the neural network. As well as basic addition of small random numbers, weights in the network could be negated or duplicated. The negation was intended for very simple networks and allowed a connection to change from inhibitory to excitatory (or vice versa) in one step. Duplication of weights was intended to allow mutation to introduce some network symmetry. Duplicating the weights for one sensor group to another might prove useful to help an agent react in the same manner with objects on the left or the right.

Recombination used a standard global discrete recombination for recombining neural networks. That is each value in the network was probabilistically chosen from one of the parents. This type of recombination is known to be quite disruptive unless the networks are already very similar. Due to this recombination was not always used when generating offspring.

## 2.5 Experimental Tools

### 2.5.1 Simulator

Java was chosen for writing this simulator, because of the authors familiarity with Java development. The ability to use different operating systems (Mac OS X and Linux) for development and for running was also a benefit. As the core

of the simulator could run without a graphical user interface (GUI), platform differences in Java implementations could also be avoided.

The main class for running the simulation (without a GUI) was "proj1.Environment". This class ran the genetic algorithm (selecting agents to remove and for breeding) and the rigid body simulation. It was also responsible for outputting data to file and other house keeping tasks.

The rigid body simulation was controlled by the class "proj1.simulator.Simulation". This class knew nothing about "Agents" or the genetic algorithm. It was merely concerned with "Vehicles", other physical objects, the forces acting on them and collisions between them. To allow specialised collision handling a "collision handler" could be provided by implementing the interface "proj1.simulator.CollisionHandler". The Environment class implemented this interface, so that it would know when agents had collided with food or with each other, allowing it to make adjustments to their energy among other things.

Vehicles in the simulation could be given a "brain", by implementing the interface "proj1.simulator.Brain". As the rigid body simulation progressed the "brain" would be updated with new inputs (from the sensors) and be allowed to calculate the relevant motor outputs. Among other things this allowed me to try out hardwired brains, to check that the sensors and motors were working correctly.

Interfaces for genotypes (proj1.evol.Genotype) and phenotypes (proj1.evol.Phenotype) were also created. This helped make it straightforward to try out different genotype representations with minimal changes.

### 2.5.2 Data Presentation

Data from the running simulation was output as plain text comma separated values (csv). Initially standard spreadsheet software was used to view these files and generate graphs, but this quickly became cumbersome. As the number of values being stored increased it became necessary to write a script to process the data and generate the graphs automatically.

Python and the matplotlib [1] library were used to this end. "matplotlib is a pure python 2D plotting library with a Matlab syntax" [1]. It allows the generation of either PNG (Portable Network Graphic) or PS (Postscript) files. The former being useful for previewing the graphs after a run and the latter for inclusion in Latex files (i.e. this report).

## Chapter 3

# Results and Discussion

As the simulations progressed data about the current state of each population was outputted to a file. This file could then be run through a simple Python script to generate a series of graphs representing the change of various attributes over time. This made it quite easy to "eye ball" the data and get a good feel for what had happened during a run.

All simulations featured 150 pieces of "food" (constantly replenished) and 100 "rocks" uniformly distributed throughout the environment. The genetic algorithm was set to run once every 100 time-steps (4 simulated seconds), when an agent from each population would be removed and replaced with an offspring from the remaining population.

Unless otherwise mentioned graphs were generated using a "moving average" of 100 points, to help smooth out the otherwise quite noisy data.

On all graphs the predators fitness is shown as a solid line whilst the preys is shown as a dotted line.

### 3.1 Simple Agents

Early runs (Figures 3.1- 3.6) used very simple neural network controllers with no hidden neurons. The simple network architecture meant that analysing results was easier, there were only so many things that the agents could do with their inputs.

#### 3.1.1 Run 1 to 3

Runs 1 to 3 (Figures 3.1- 3.3 were "baseline" runs. Their objective was simply to see what was "normal" for a simulation. The only real difference between them was that in Runs 2 and 3 the size of the environment was increased. The increase in size made to see if this would affect the evolution of the field of vision of the agents.

There were 100 prey agents and 50 predator agents in each run. As can be seen there is a fair bit of variation in how both species average fitness changes over time. It is however fairly safe to say that the predators never completely dominate and at worst pretty much fail to compete. This is probably because the predators task is much harder, as they have to find moving targets. The predators usually seemed to turn this problem on its head: predators were observed circling on the spot (in reverse). They would then lunge at any prey that came nearby, using their greater motor power to try and close the gap quickly. This tactic was only really possible because the agents sensors had a very limited range. For the agents it was a bit like wandering in very thick fog, so actively pursuing prey was very difficult, as it did not take much for them to move out of sight. Due to the stateless nature of the agents networks, out of sight meant out of mind too.

Increasing the environment size appeared to have a slight effect on how the field of vision evolved. It seemed that increasing the environment size tended to increase the prey agents field of vision, because a side effect of increasing the environment size was to make the environment sparser. This would mean that having "precise" sensors for navigating in a dense space were not required, so the prey could afford to "keep their eyes out" for predators instead. However due to the small number of simulations actually run (7 total), this is not a statistically significant observation.

### **3.1.2 Run 4**

The intent of Run 4 (Figure 3.4) was to see what effect increasing the predators population size had on their ability to compete - would they be more competitive or less? The predators population size was increased to 100, to equal that of the prey.

The increased population size did make the predators, collectively, much more competitive. In all three simulations performed the predators and prey were closely matched. This may have been simply the case that the increased population size made it more likely that viable predators would be contained in the initial, random, population.

The increased population size also raised the average age of the predators. As the same number of agents were being removed at the same rate as previously, doubling the population tended to halve the chance that any particular agent would be removed. This would tend to make the genetic algorithm a bit more forgiving and so agents that were actually doing quite well, but were just unlucky in their current situation (e.g. being far away from prey) would tend to have a slightly longer time to move to a better location.

### **3.1.3 Run 5 and 6**

Up until this point the predators had always had a speed advantage or more precisely a motor power advantage. This allowed them to accelerate faster and potentially reach higher speeds than the prey. I now wanted to see whether

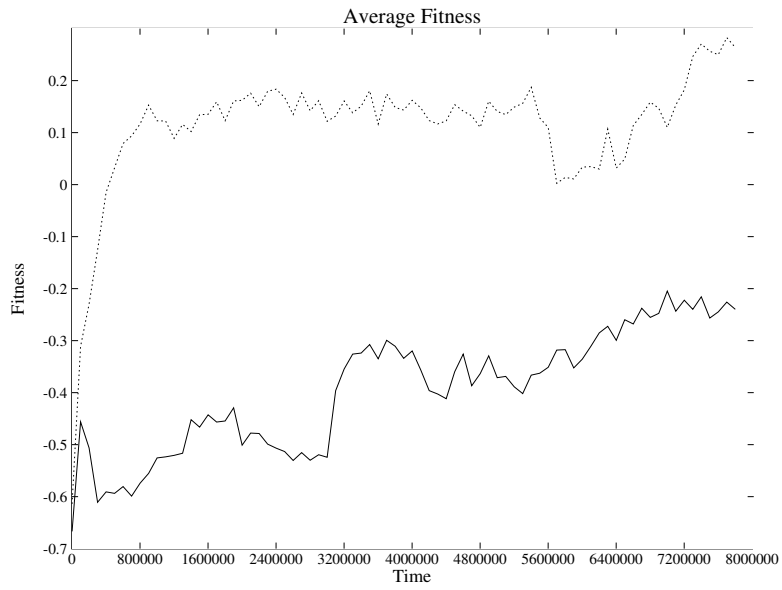
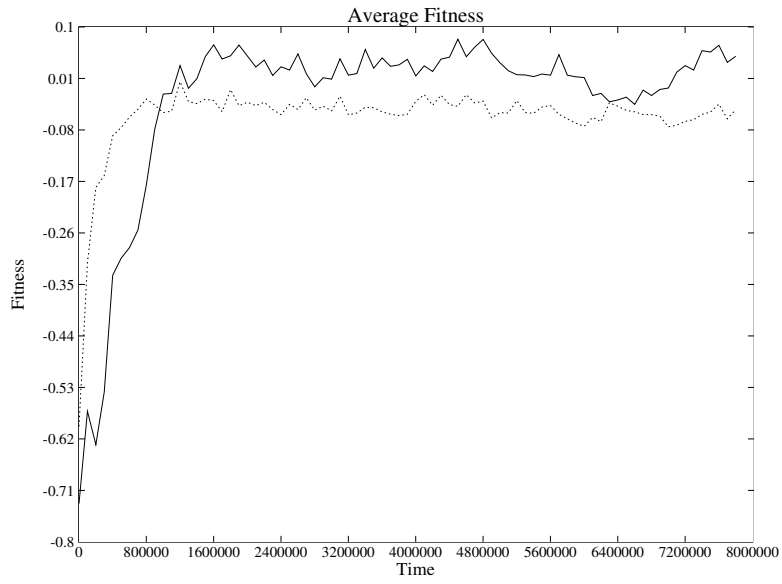


Figure 3.1: Fitness: Run 1 (Predator Solid Line, Prey Dotted Line)

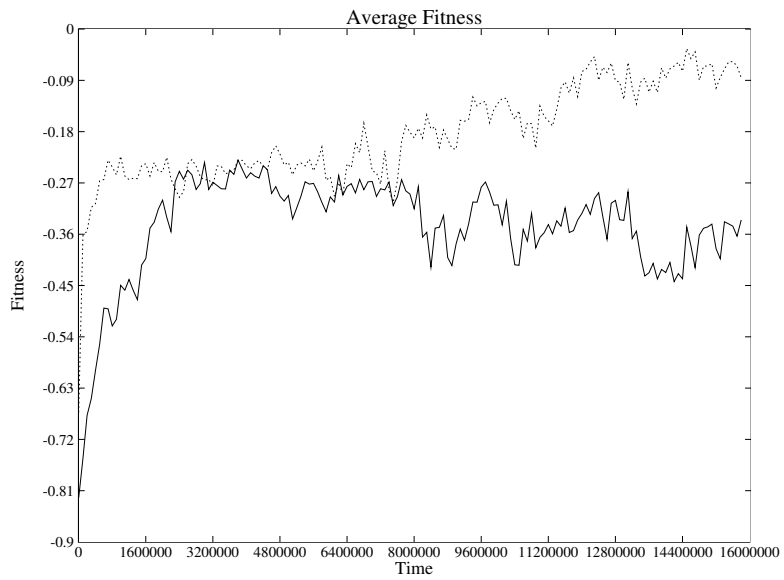
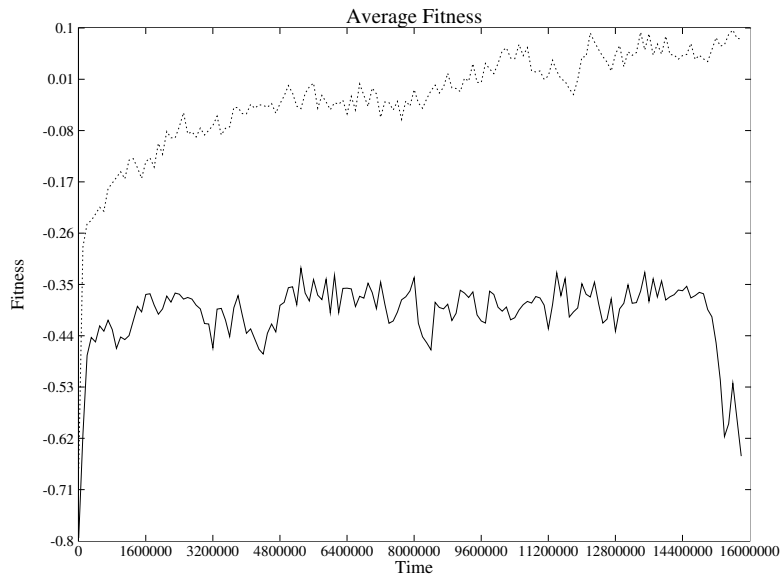


Figure 3.2: Fitness: Run 2 (Predator Solid Line, Prey Dotted Line)

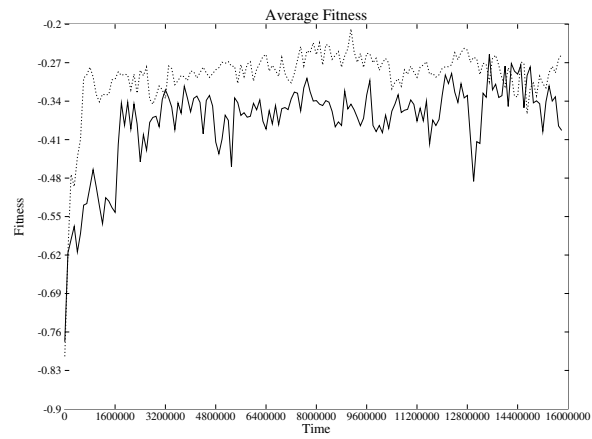
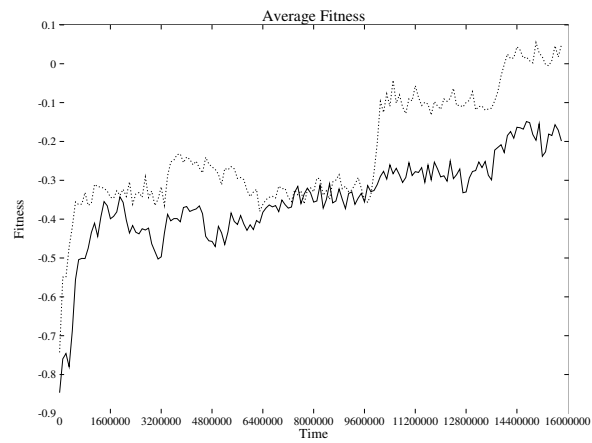
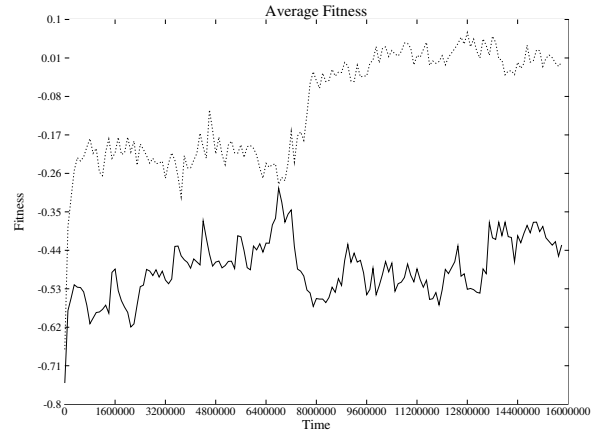


Figure 3.3: Fitness: Run 3 (Predator Solid Line, Prey Dotted Line)

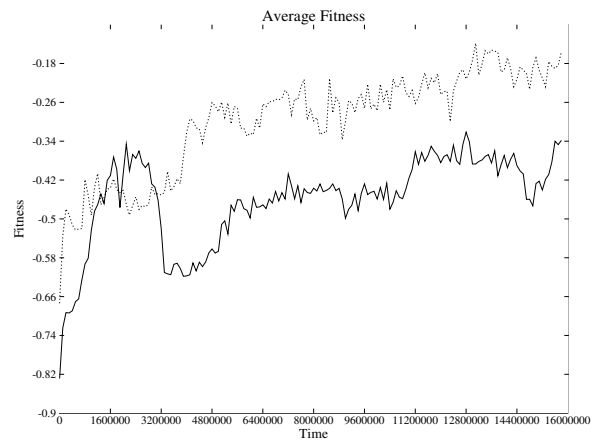
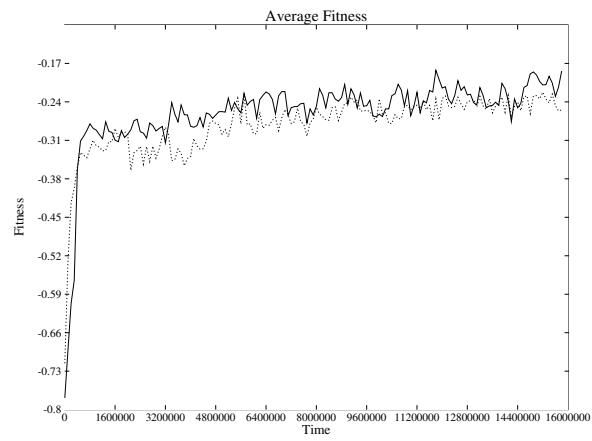
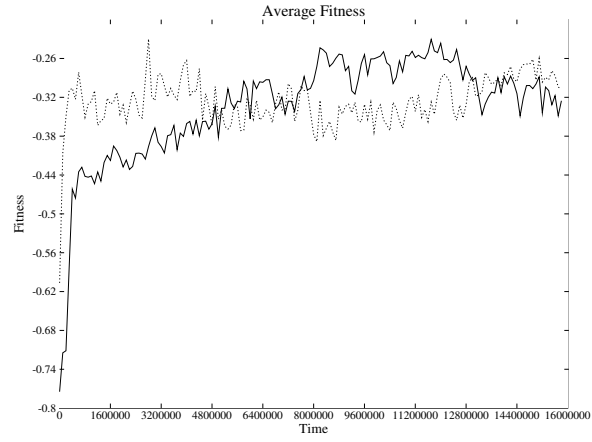


Figure 3.4: Fitness: Run 4 (Predator Solid Line, Prey Dotted Line)

the predators really needed this advantage or not and if they did could they be given a different advantage instead?

Run 5 (Figure 3.5) was a completely "fair" simulation. Both population sizes were the same (100) and the predators had no speed advantage.

Tellingly the predators did not do very well at all. In fact in one particular simulation the prey showed a near "perfect" fitness curve (inverse exponential in nature). In that case the prey must have hardly worried about the predators at all. The prey in that run had a very sensitive green sensor and were then probably relying on basic light sensor to avoid the predators. By comparison the predators colour sensors were both setup to react more to a combination of red *and* green, which seems a fairly bad combination considering that the prey were blue! Although the use of a realistic physical model meant that prey could not turn on the spot and run exactly away [6] and thus never be caught, it appeared that this was not enough for the predators. Perhaps the simplistic nature of the networks made it difficult to really take advantage of this.

Run 6 (Figure 3.6) was the same as run 5, except that the sensor range of the prey was halved, to give the predators a slightly different advantage.

The predators performed nearly as badly as they had done in Run 5. A larger sensor advantage may have been required. It may have been the case that the prey could cope with the smaller sensor range as their food was static and they only needed enough sensor range to avoid predators at the last minute, who could still be quite easily "lost".

## 3.2 More Complex Agents

Later runs (Figures 3.7- 3.9) used networks with four hidden neurons for at least one of the species. The hope being that this would lead to potentially more interesting behaviour.

### 3.2.1 Run 7

Run 7 (Figure 3.7) was another "fair" simulation. Both populations were equal in size and the only difference between them were their eating habits. However as opposed to run 5 (another fair simulation) the agents neural network controllers had four hidden neurons. The intention of this simulation was to observe the effects (if any) of increasing the complexity of the agents neural networks. Would there be any noticeable differences when compared with Run 5? Would the predators be able to exploit the increased network complexity to gain some sort of advantage?

Just like in run 5 the "fairness" of the simulation meant that the predators were unable to compete effectively. In three of the simulations the predators barely improved through out the run. The average network variation (the average difference between all networks in a species) stayed relatively high for the predators, where as the prey usually settled down a bit.

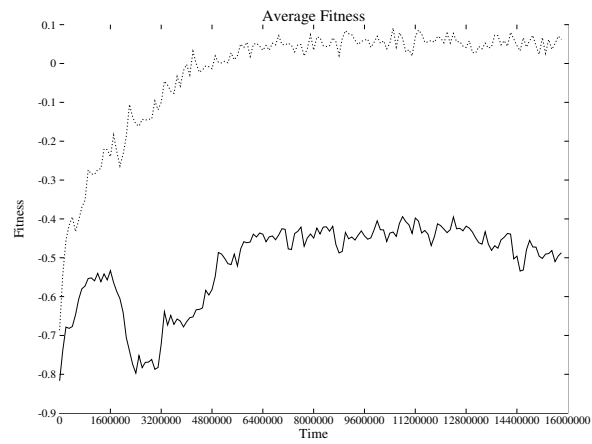
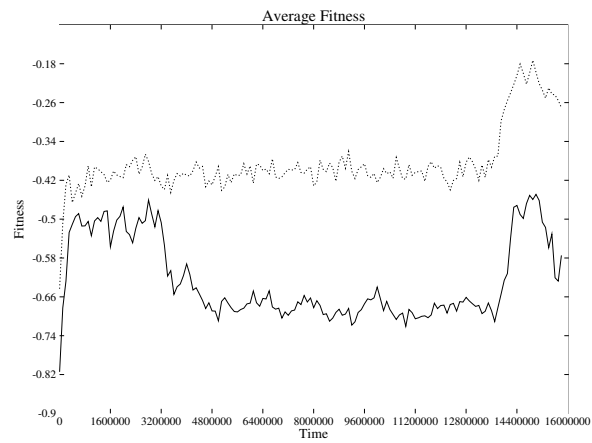
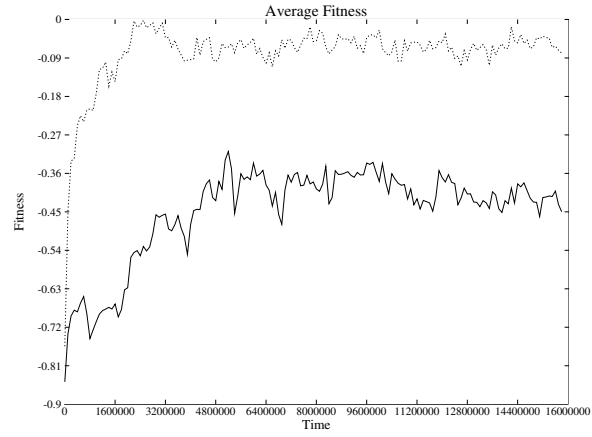


Figure 3.5: Fitness: Run 5 (Predator Solid Line, Prey Dotted Line)

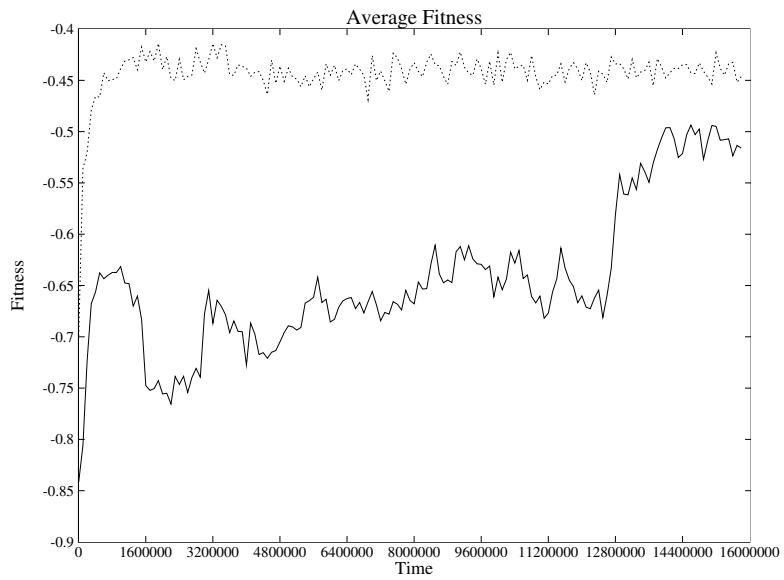
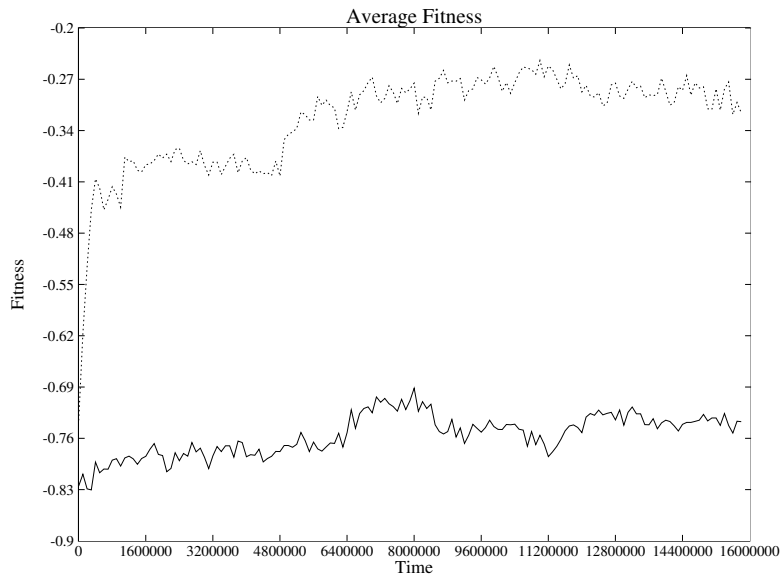


Figure 3.6: Fitness: Run 6 (Predator Solid Line, Prey Dotted Line)

Perhaps the predators would eventually find a good network setup to allow them to capture prey effectively, but it looked like it might take a while. It may have been interesting to have run this simulation with larger populations. It may simply have been that the more complex networks meant that the initial random population was unlikely to contain good predators.

An interesting side effect of increasing the network complexity is that for the first time the motor angle parameter actually evolved in a meaningful way. Usually this parameter, which controls the position of the motors on the agent body, stayed largely fixed at 90 degrees. Occasional agents may have different motor angles through mutation, but these rarely carried through into the rest of the population. In a few of these simulations the motor angle of the herbivores changed for extended periods of time. The increase was only by about 5 degrees, but it was quite pronounced. I presume that the non-linear nature of the network now allowed the agent to take advantage of a slightly less linear control mechanism. It may be possible that this enabled the prey to more accurately control their movements.

### 3.2.2 Run 8

Run 8 (Figure 3.8) re-introduced the predators speed (motor power) advantage. This was the same as setup as run 4, but with more complex networks. The goal of this run was to gauge the effects of the more complex networks under the conditions in which previously the predators had performed well. One would expect the predators to have compete as well as in run 4.

In fact this was *not* the case. Compared to Run 7, where the predators did not have any advantage, there was an improvement. The increase in network complexity must have been to blame, as everything else was the same as in Run 4. It appears that finding a usable network for the prey was easier and quicker than finding an equivalent one for the predators.

As with Run 7 the prey evolved their motor angle. This happened in all but one of the simulations in Run 8. It seemed that the (slight) increased threat of the speed advantaged predators may have necessitated this change. Moving the motors off from 90 degrees to either side would have the effect of changing an agents turning behaviour. The motors would now a slightly smaller turning effect, perhaps allowing prey to be more precise about how they turn. This could be a great benefit when trying to avoid predators, as over-steering could prove fatal.

### 3.2.3 Run 9

In Run 9 (Figure 3.9) the predators retained their speed advantage and the prey lost the more complex network. The aim of this experiment was to see if the predators could take advantage of the complex network, if the prey did not also have one.

Overall the predators did better than in the previous run (Run 8), but they were still not really threatening the prey very much. By comparison with Run

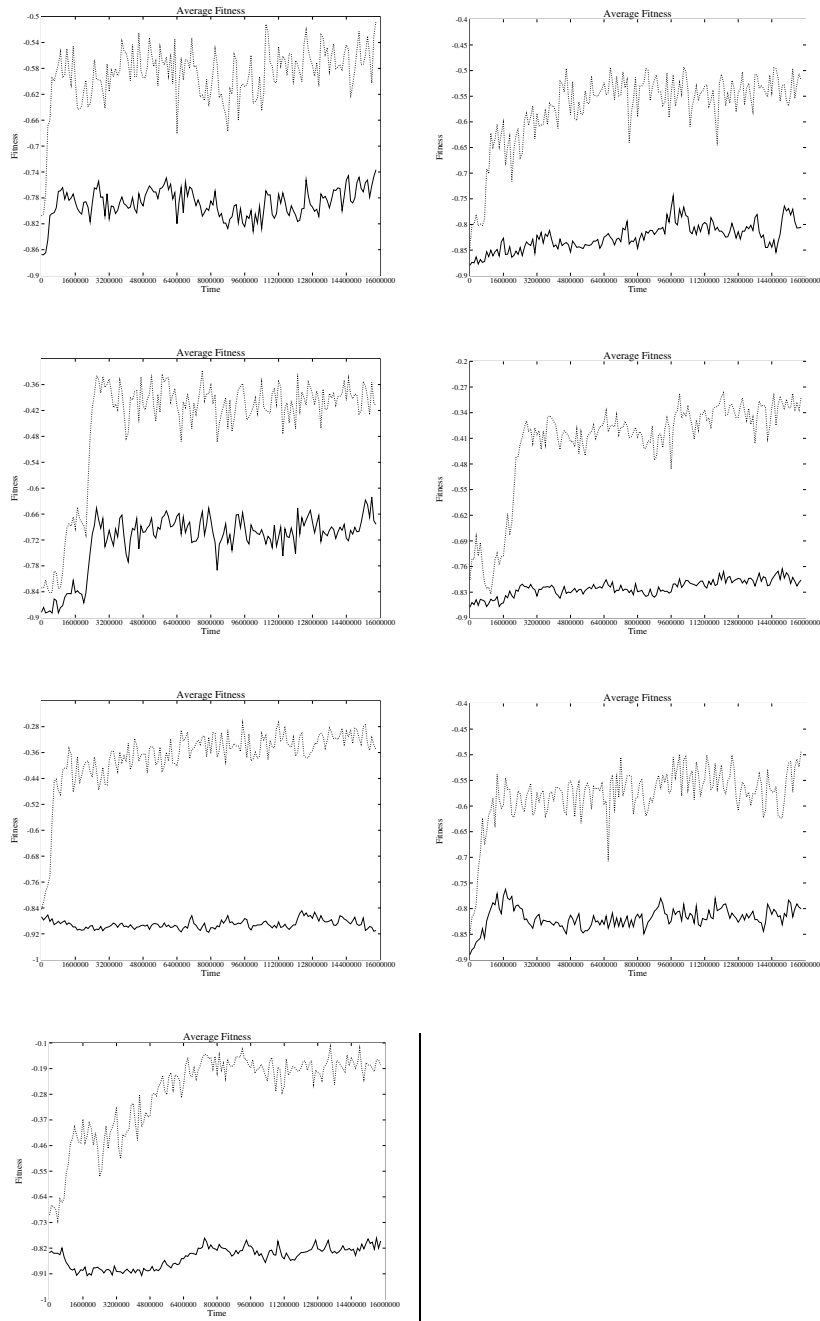


Figure 3.7: Fitness: Run 7 (Predator Solid Line, Prey Dotted Line)

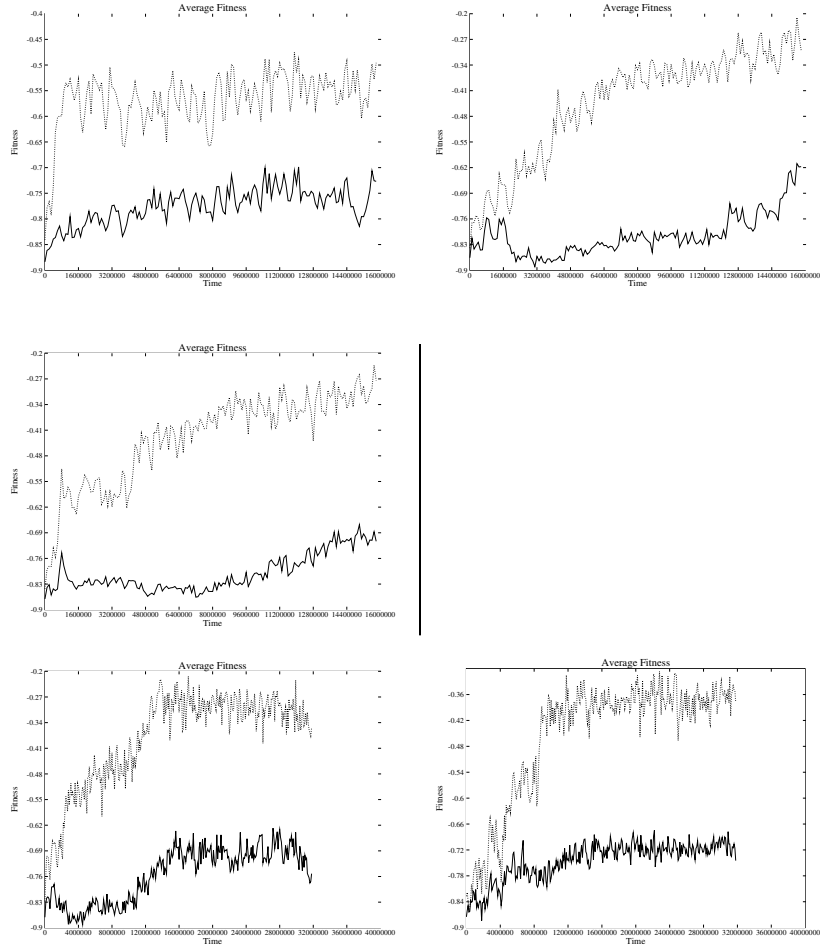


Figure 3.8: Fitness: Run 8 (Predator Solid Line, Prey Dotted Line)

4, when the predators also had the simple networks, the performance was much worse. So it seemed that the complex network was actually a hindrance!

The added complexity of the predators network was probably simply too much for them. It may be that the mutation and recombinations operators used were inappropriate for the more complex network and so evolution was hampered. The only thing worth noting about using the more complex was that the predators at least seemed more mobile and less likely to "wait and pounce". However this might have been exactly their problem - this behaviour may be an effective strategy that proved harder to evolve with the more complex network topology.

### 3.3 Interpretation of Results

It appears from most of the simulations performed that the predators generally had a harder time of it than the prey. Run 4 (Figure 3.4) was the only Run in which the predators achieved similar fitness values to their prey and even then that was with a big speed advantage and a large population. This seemed to be mainly because the sensors were quite ineffective.

More complex networks did not guarantee better fitness and/or sophisticated behaviour. I suspect that this is because the increased network size tended to slow down evolution. It also may well be the case that the wrong type of complexity was used. For example networks that contained recursive connections might have been a better choice.

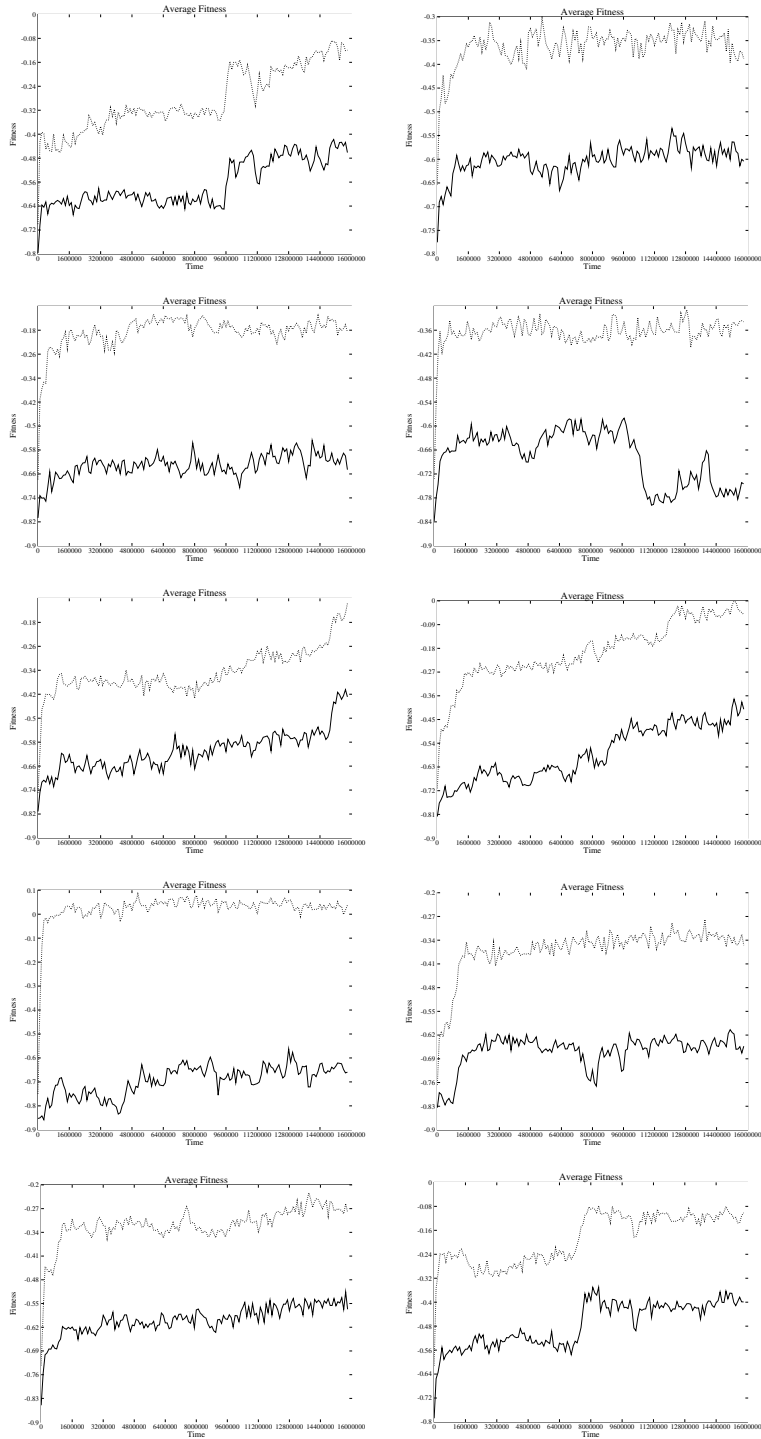


Figure 3.9: Fitness: Run 9 (Predator Solid Line, Prey Dotted Line)

## Chapter 4

# Conclusion

I have created an ecological simulator for studying the co-evolution of populations of agents in an artificial environment. Each agent has sets of sensors that are rooted in their environment and uses a simple neural network for controlling how it moves around in the environment.

The evolution of each population of agents is controlled by a genetic algorithm, that works in a steady state fashion, to allow a "continuous" simulation, without generational gaps. The fitness of each agent is simply a measure of whether it is "alive" or "dead" and effects whether an agent should be removed from the simulation and whether it can breed. As this form of evolution does not use "discrete competitions between individuals" [9] agents therefore exist in a more complex environment, which better represents the interaction between agents in an ecology.

By using a genetic algorithm to control breeding I have removed *some* of the problems associated with producing "evolutionary stable strategies" [10] that can occur with complex ecological simulators. This also provides much simpler feedback to how a particular simulation ran. At the simplest level having an average fitness value, for a population, helps greatly with judging the relative performance of individual species. The genetic algorithm also gives me a great deal of control over the evolution of the agents, but by having a binary fitness ("alive" or "dead") I am able to preserve some emergent properties of more open ended simulations.

I have performed a series of simulations modelling a predator-prey dynamic. In these simulations it has been observed that the predators task, at least currently, is much "harder" then the preys task. This is due in part to the fact the the predator must pursue moving targets. The current sensors and network topologies used do not lend themselves well to this task, as there is currently no way for an agent to determine "2nd order" information (i.e. velocity). Such information might allow a predator to not only know where its prey currently is, but also to be able to predict where it is likely to be. This is also shown by the fact that simply introducing a more complex network, which could still not determine velocities, did not help matters and if anything made matters worse

for the predators.

This approach to simulating agents in an artificial environment could prove to be powerful, but is currently not fully exploited due to the simplistic nature of the agents neural networks and the problems with the sensors.

## 4.1 Possible Extensions and Improvements

There are several obvious improvements that could be made to this work. Improving the sensors is definitely important, particularly for the predators. It is hard enough to catch prey without worrying that you might lose sight of them at any moment. After that tweaks to how the genetic algorithm works may be in order. In particular mutation and recombination need looking at, as currently they are rather crude. How agents gain energy, how their metabolism and how both relate to fitness also needs examining. Otherwise the physical simulation itself seems to work quite well.

Implementing NEAT [12] would be very interesting, as it would allow networks to gain in complexity over time, but initially start out very simple. Given the predators performance with more complex networks this would be very useful. It would then be very interesting to see how each species responded in response to changes in the network topology of the other. The fact that NEAT also uses a meaningful recombination operator would hopefully also help things out a lot.

NEAT would also open up the way for the predators to gain much need velocity information, as it allows for recurrent connections (thus allowing simple time delays). Alternatively it would simply be interesting to provide velocity sensors, which would simply indicate whether another object is moving toward or away from the agent. Velocity information would mean that predators could "intercept" prey and hopefully become much more competitive.

It would also be interesting to introduce more morphological effects. For example allowing agents to have an arbitrary number of motors, with each exacting a metabolic cost when in use. We might then see agents specialised for speed and others specialised for efficiency or manoeuvrability.

# References

- [1] Matplotlib. <http://matplotlib.sourceforge.net/>.
- [2] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 1984.
- [3] C.Hecker. Physics, part 1: The next frontier. *Game Developer Magazine*, October/November 1996.  
<http://www.d6.com/users/checker/pdfs/gdmphys1.pdf>.
- [4] C.Hecker. Physics, part 2: Angular effects. *Game Developer Magazine*, December/January 1996.  
<http://www.d6.com/users/checker/pdfs/gdmphys2.pdf>.
- [5] C.Hecker. Physics, part 3: Collision response. *Game Developer Magazine*, February/March 1997.  
<http://www.d6.com/users/checker/pdfs/gdmphys3.pdf>.
- [6] C.W.Reynolds. Competition, coevolution and the game of tag. In R.Brooks P.Maes, editor, *Artificial Life 4*, Santa Fe Institute Studies in the Sciences of Complexity, pages 59–69. MIT Press, 1994.
- [7] D.Roberts. Collision detection: Getting the most out of your collision tests. Technical report, Doctor Dobbs Journal, 1995.  
<http://www.ddj.com/documents/s=983/ddj9513a/>.
- [8] D.Cliff G.F.Miller. Co-evolution of pursuit and evasion ii: Simulation methods and results. In P.Maes M.Mataric J.A.Meyer J.Pollack S.W.Wilson, editor, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB96)*, Complex Adaptive Systems, pages 506–515. MIT Press, 1996.
- [9] K.Sims. Evolving 3d morphology and behavior by competition. In R.Brooks P.Maes, editor, *Artificial Life 4*, Santa Fe Institute Studies in the Sciences of Complexity, pages 28–39. MIT Press, 1994.
- [10] L.Yaeger. Computational genetics, physiology, metabolism, neural systems, learning, vision and behavior or polyworld: Life in a new context. In C.G.Langton, editor, *Artificial Life 3*, volume 17 of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 263–298. Addison-Wesley, 1994.


- [11] M.Lunn. *A First Course in Mechanics*. Oxford University Press, 1991.
- [12] K.O. Stanley R.Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2), 2002.
- [13] T.S.Ray. An approach to the synthesis of life. In C.Langton C.Taylor J.Farmer S.Rasmussen, editor, *Artificial Life 2*, Santa Fe Institute Studies in the Sciences of Complexity, pages 371–408. Addison-Wesley, 1994.



<p><b>Project management skills</b></p> <p>Briefly explain how you will devise a management plan to allow your supervisor to evaluate your progress</p>	<p>I will create a timetable &amp; specifying what I hope to achieve and when.</p> <p>I will meet regularly with Dr. Jankovic to discuss my progress and the direction of the mini-project.</p> <p>I will also determine a "contingency" plan to be used if things do not go quite as planned.</p>
---	--

<p><b>Systematic literature skills</b></p> <p>Briefly explain how you will find previous relevant work</p>	<p>I will search for literature using keywords such as "co-evolution", "complexity", "autonomous agents". I shall change which keywords I search <del>for</del> using depending on the results I receive.</p> <p>I shall also look at research papers in the "Alife journal" as well from the Alife conferences amongst others.</p>
--	---

<p><b>Communication skills</b></p> <p>What communication skills will you practise during this mini-project?</p>	<p>I will use oral communication skills when discussing the progress and direction of my project with Dr. Jankovic.</p> <p>I will use written communication skills when recording results of experiments and also when writing my project report.</p>
---	---

Signed (student)   
Date: 16/10/03

Signed (supervisor):   
Date: 16/10/2003

Office use:
Copy for: student / supervisor / file

## Appendix B

# Statement of Information Search Strategy

### B.1 Forms of Literature

Conference papers are likely to be very important as are journal articles. Books and technical articles will also be relevant, but mainly for specific information relevant to the implementation of the projects software.

### B.2 Appropriate Search Tools

The Engineering Index will be used for locating conference papers and journal articles, as it is generally up to date and allows the easy viewing of abstracts. This is essential for deciding whether a paper may be relevant.

### B.3 Search Statements

The search terms to be tried are:

red AND queen  
simulation AND evolution AND agents

### B.4 Search Evaluation

The engineering index search for "red AND queen" returned 5 results that seemed potentially promising and the search "simulation AND evolution AND agents" 11 promising results.

Knowledge of existing work also played a big part in collecting references. One paper was retrieved after a conversation with a peer.

## Appendix C

# Simulation Application and Source Code

The source code and compiled executable versions of the simulator will be made available via the authors School of Computer Science personal website at:

<http://studentweb.cs.bham.ac.uk/msc37jxm/>

