The University of York – Department of Computer Science

Submitted in part fulfilment for the degree of MEng Computer Systems and Software Engineering

Evolving Specialised Species in Diverse Simulated Ecologies using a Subsumption Architecture

Daniel Everard

8th May 2007

Supervisor: Dr Suresh Manandhar

Number of words = 15,774, as counted using Microsoft Word word-count facility.

This includes the body, bibliography, and appendix of this report.

Contents

1. Abstract

This project is targeted at reviewing the process of speciation and adaptation within variable environments. The project aims to simulate speciation within a population of learning agents operating in a multi-agent simulated ecology. Following on from existing work on the subsumption architecture [Bro85], agents are equipped with a proven learning control system. An environmental design tool is developed to introduce the geographical diversity necessary to facilitate species emergence, and tests are performed to confirm the correctness of the model. Inspiration and modelling assumptions are drawn from nature and biological research where possible in an effort to simulate real-world phenomenon as closely as possible.

A Note Regarding the Predecessors of this Work

The development of a simulation environment for the subsumption architecture required an in depth understanding of the concepts used to develop the solution presented in [Will06]. For this reason the review section of this document, particularly the chapters reviewing reinforcement learning and macro architectures, cover similar ground. Where alternative, more up-to-date or relevant material has become available, this has been covered to avoid excessive repetition of work.

Throughout this document, several references are made to the body of work developed by Simon Williamson and Sam Fairhurst, both of whom extended a Java implementation of the subsumption architecture and a partner simulation environment. This implementation forms the basis for the work developed in this paper, and will be referred to as the 'Java implementation' throughout.

2. Review

2.1 'The Origin of Species'

Evolution is the term used to describe the changes in a population's traits over time. Traits are encoded in a chromosome and passed on to new members of the population through reproduction. Over time, changes are facilitated by processes of crossover (reproductive combination) and mutation (random change), which lead to new traits. These new traits may yield an individual with a better or worse chance of survival in its habitat, leading to a better or worse chance of its particular genes being passed on to the next generation via reproduction. This process of *natural selection* was famously introduced by Charles Darwin in his book, 'The Origin of Species' [Dar98].

Over time, this selection mechanism has been shown to lead to a population developing to perform better in their (possibly changing) environment. This mechanism is known as *adaptation*. Another 'mechanism of change' is *genetic drift*, which is the change observed in allele frequencies over a number of generations due to the sampling error in the selection of alleles inherited by a new generation. [Amos98] For example, a chance event wiping out more green beetles than brown beetles will lead to a larger sample of brown beetles surviving to the next generation [Berk07(a)]. Finally, *genetic drift*, the physical migration of individuals leading to a wider distribution of their genes, is an important mechanism.

Two forms of evolution are widely recognised, both definitions relying upon the concept of *species* explored later. Evolutionary change below the level of species, i.e. changes to the allele frequencies within a species, is termed *microevolution*. *Macroevolution* is the term for evolutionary process operating at or above the species level. Gradual microevolution is thought to lead to macroevolution over time, but the distinction is clear and is useful when considering the process of *speciation*.

2.1.1 Observing Adaptation

Perhaps the most famous and accessible example of adaptation is that of the peppered moth. Studies throughout the 20th century addressed the observed colour variation in peppered moth populations. Before the industrial revolution in Britain, the population of peppered moths were predominantly light coloured, known as the 'typica' form. The industrial revolution led to a darkening of trees, and to the death of many light coloured lichens against which the moths were camouflaged. For this reason, the light coloured moth became easier to spot against the background of its habitat, and conversely moths with an allele dictating darker colouration became better camouflaged. The predatory birds which fed upon the moths therefore ate a higher proportion of *typica* moths, leading to a shift in allele frequency of the dark colouration allele.

Generally it is the work of entomologist Bernard Kettlewell cited as highlighting the evolutionary basis for this exhibition of melanism through experimentation, with his experiments aimed at testing hypothesises presented by James W. Tutt in the late 19th century. Tutt observed the phenomenon described above, and his hypothesises captured the evolutionary basis for the observed behaviour. [Gra99]

Kettlewell performed a series of experiments in which a population of peppered moths were placed inside a large aviary with a small number of predatory birds. His stated objective, as summarised in [Gra99] was to "determine whether or not birds ate cryptic moths at rest on their normal backgrounds, and whether, in the case of polymorphic moth species, the birds ate them selectively". His findings were that the most conspicuous of the moths were the first to be eaten.

Since Kettlewell's experiments, many evolutionary biologists and creationists have argued against his findings, but none have found a large enough hole in the methods he used to convince the scientific community to question the correctness of Tutt's hypothesises. A review of the development of this research is presented in [Gra99].

2.1.2 Species

The definition of a species is controversial and elusive. Biologists disagree on the definition, but it is agreed that species form the fundamental taxonomic units of biological classification. [SEP02] The encyclopaedia entry in [SEP02] finds that species are "first and foremost genealogical entities", but summarises several interpretations of the term, and from a typically philosophical perspective, argues that they are all correct.

The formation of species is called speciation – the "process by which new species arise from existing species"[UWB05]. The traditional model of speciation, attributed to Ernst Mayr for his biological species concept, suggests a "species is a group of interbreeding populations reproductively isolated from others" [Mayr62]. Despite excluding from consideration species which reproduce asexually (plants), this definition has held its place for decades. It has led to the concept of traits, or 'isolating mechanisms', which "prevent gene flow between diverging populations" [Bak05,p2].

[Bak05] is an interesting review of adaptive speciation of the kind this work hopes to simulate. The author notes that the theory of evolution accounts for adaptation within a species, specifically linear *phyletic speciation*, which occurs as changing environmental pressures cause an adaptive, evolving species to acquire different traits over time [Bak05]. Phyletic speciation is not considered as 'proper' speciation, characterised by the branching of species into two or more distinct and co-existing species. This 'speciation proper' is named *divergent speciation*, and results from "the reproductive isolation of two parts of a population"[UWB05], as detailed in Mayr's work.

Two mechanisms for the divergence of isolated populations are considered in [UWB05]:

- Independent genetic variation and adaptation within the two groups
- Geographic environmental differences leading to differing selection criteria

The divergence over time might cause the two groups to become unable to interbreed, even once the isolating barriers are lifted. This last point presents important questions for consideration in this project: How does one differentiate one species from another, and how does one simulate the barrier to reproduction? Looking once again to the biological world for inspiration we can see that by simulating 'failed' reproduction attempts where allele differences are significant enough, we can hope to observe an initial phase of speciation where only environmentally suited individuals will survive, but there may be more than one 'type' of suitability (e.g. carnivore/herbivore). By re-using the mechanism providing this barrier to reproduction in analysis, we can hope to distinguish each species existing within the ecology.

This capability was in fact built into the previous project [Will06]. A global parameter (sexual selectiveness) was used to define the maximum difference in the sum of allele values for two 'mating' agents in order for them to be successful. However, a gene-for-gene comparison and threshold might improve the differentiation ability of the system. For instance, the existing implementation might compare the following (simplified) chromosomes and consider them 'similar' enough to mate successfully.

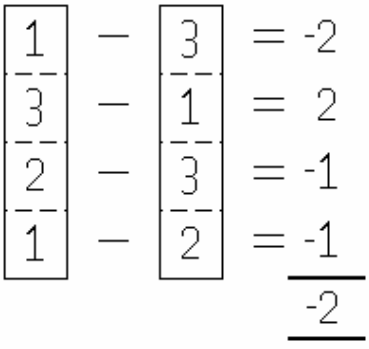

Figure 2.1.1: Implementing a barrier to cross-species reproduction – existing implementation

This is clearly not consistent with the biological inspiration the work was attempting to harness. This project will focus on improving the over simplified models of the previous work to evolve species more successfully.

Evaluating the range of species in the ecology is not a trivial task. A large range of feasible allele configurations may lead to one, two or many more 'species' emerging. The process of distinguishing species will be considered in section 4, 'Experimental Procedure'.

2.1.3 Isolating Mechanisms

The mechanism by which a population may split into isolated groups can be geographic or non-geographic. Geographic isolation leads to *allopatric speciation*. The most dramatic of examples is that of continental drift, where a body of land is split, isolating two halves of a population. Figure 2.1.2 illustrates the geographic isolation process in the case of a river loop closing off. A more subtle form of geographic isolation might be the case of a few members of a population migrating to a geographic area where other members of the population are unlikely to visit.

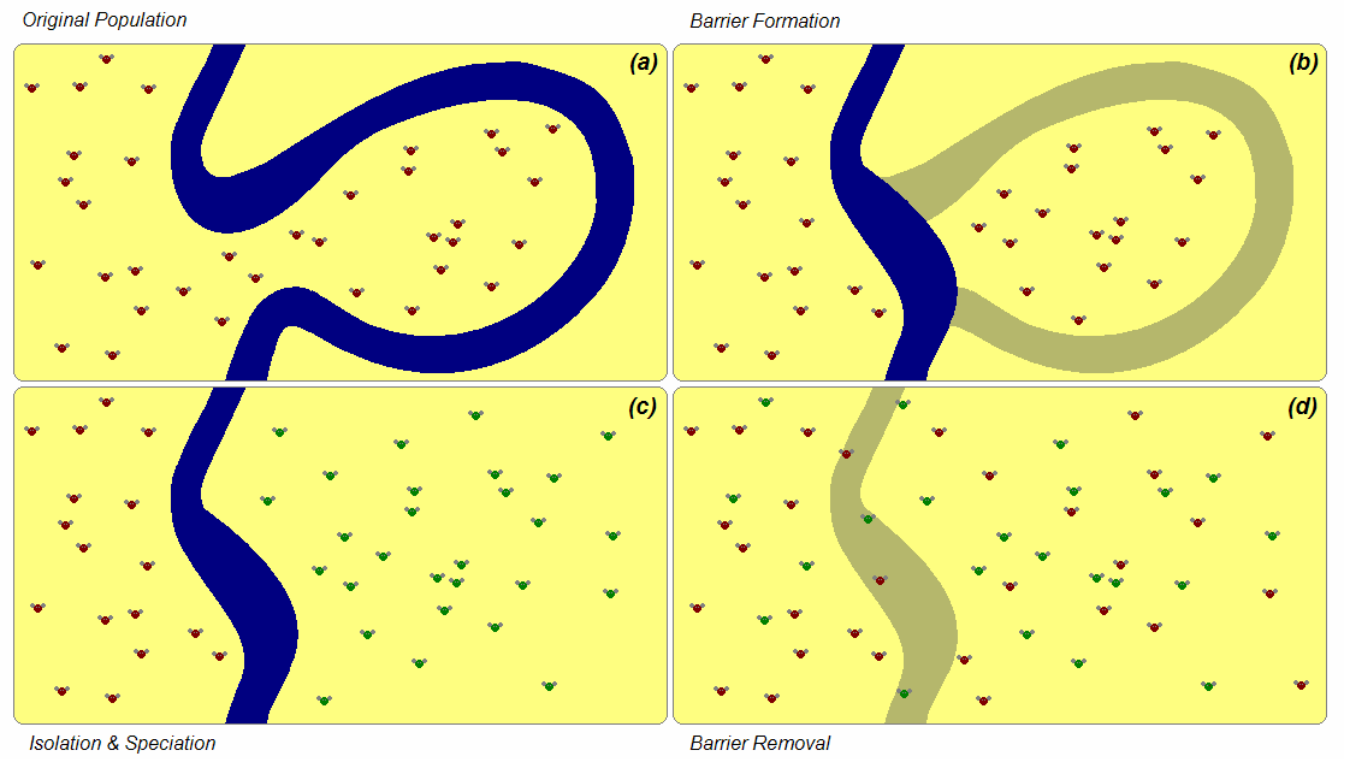

Figure 2.1.2: Vicariant Speciation, an example of geographic isolation
(a) – (b): Geographic change leads to isolation of sub-populations
(c): Evolutionary processes lead to speciation
(d): Geographic barrier disappears, but species are not capable of reproduction

Other isolating barriers do exist. Changes in a few individuals preventing them from breeding with others will lead to isolation and possible divergence. An example is that of mutations within a population leading to physical changes (colour, size) making a small group of individuals unattractive to the majority of the population. Interbreeding amongst this minority will lead to a new species in that a good deal of divergence has already occurred. This reproductive isolation is known as *sympatric speciation*, and is rarely observed in animals, but is often the trigger for the evolution of new species of plant. [UWB05]

This project aims to simulate divergent speciation amongst a population of simulated organisms. The simplified model of characteristic genes (perhaps overly so) leaves little hope of observing sympatric speciation. Therefore the design of the system must consider ways of simulating geographic isolation in order to observe allopatric speciation. The existing implementation does not allow for any environmental

variation beyond a basic and poorly considered attempt to develop distinct populations of land and water based agents, and two possible configurations for resource distribution.

2.2 Artificial Life & Machine Learning

The study of artificial learning is today a wide and well-researched area of computer science. This project joins many others in attempting to replicate the behaviours exhibited by natural organisms using a highly abstracted model of real-world intelligence facilitators. The 'agents' developed in this work live and interact with a model ecology, where the actions of many individuals lead to observable consequences for the society of agents as a whole. Species may emerge, and new, specialised behaviours may be learned and shared, enabling the agents to survive more efficiently. This form of simulation falls into the field of 'Artificial Life', or Alife, as summarised in [Pac00] where the authors cover the history and motivations of the discipline, focusing on the benefits of the field for biological researchers looking for answers where standard mathematical biological methods have hit a wall.

A dictionary definition of Alife is hard to come by, and few agree on the boundaries of Alife systems, but the definition from [Will06] cites [Lan05], stating:

> *"[Alife is] the name given to a new discipline that studies "natural" life by attempting to recreate biological phenomena from scratch within computers and other "artificial" media. Alife complements the traditional analytic approach of traditional biology with a synthetic approach in which, rather than studying biological phenomena by taking apart living organisms to see how they work, one attempts to put together systems that behave like living organisms"*
>
> [Lan05]

Alife systems tend to share a bottom-up approach to building intelligent systems. Agents operating within a simulated ecology are often equipped with some actuator mechanisms (e.g. behaviours), and the means to make informed decisions as to which behaviour is most befitting a particular situation. By contrast, traditional artificial intelligence (AI) systems utilise a top-down approach.

The agents in an Alife system operate with a complex architecture of their own, but as with the predecessors to this project, the actions of individuals are rarely our primary interest. Rather it is the complexity that the system as a whole exhibits which interests us. For example, [Fai04] was interested in observing high-order interaction dynamics such as flocking and a predator-prey distinction. These are examples of complexity born from low-level interaction, and the emergence of such complexity is often unexpected and unpredictable. We run the same basic rules many times, and often behaviour more complex than the sum of the system components' individual capabilities is observed. The tendency to view agents in such a system as artificial 'life-forms' has led some researchers to consider more explicitly the interactions of these agents. [Mat97] is a study of explicit social interaction, and describes an experiment utilising social reinforcement to allow the population of agents to consider social feedback as well as individual feedback.

Underpinning the high-level field of Alife, are concepts borrowed from artificial intelligence, particularly from the field of machine learning. Machine learning is concerned with the task of making a single machine learn, not necessarily to live or survive in an ecology as with Alife, but to learn anything from parameters for equation curve fitting, to the best moves to make in a chess game, to voice patterns in speech recognition. Machine learning is an age-old discipline. Scientists have wondered if machines could be made to learn since their invention, and today many mechanisms have proven themselves as reliable learning tools.

[Wal97] is a comprehensive survey of the founding methods in Alife, which have remained dominant throughout the past decade. The review opens looking at some of the simplest Alife systems – cellular automata. These are systems comprising an "n-dimensional array of cells which use a deterministic set of rules to determine their future state." [Wal07,p5] A famous example of cellular automata systems is the two-state, two-dimensional Conway's Game of Life, which uses the following rules [Will06]:

- A dead cell is born if exactly three of its neighbours are alive;
- A cell dies unless two or three of its neighbours are alive.

Conway did very little published research into the Game of Life, but it will remain the most famous and accessible example of Alife for some time. Each individual has no complexity of which to speak, but emergent behaviour on the global game board is highly complex, and harnessing this complexity has been the focus of a good deal of research. [Ren01] is a fine example, where the author has implemented a stack-based Turing machine using the Game of Life. The size of the machine can be extended with the size of the game, making the game Turing complete.

2.2.1. Evolutionary Optimisation Techniques

[Wal97] continues to consider simulated annealing, an optimisation technique designed to avoid sub-optimal local optima in search. The technique is often not considered to be an Alife technique, but its nature leads the author into an exploration of evolutionary techniques, including the Genetic Algorithm.

The Genetic Algorithm

The genetic algorithm (GA) is a hugely important mechanism for evolutionary simulation, and is the most prominent algorithm in use for Alife simulations today. It is highly relevant to the system developed in this work, with its genetic operators forming the basis for the 'mating' action between agents in the simulated ecology of our system. The standard genetic algorithm operates upon a population of candidate solutions. Each solution is an abstract representation of the problem domain, the 'chromosome' (for example, the coefficients in a curve fitting task, or the characteristic gene-pool of a simulated agent). Each chromosome comprises a number of 'genes', each gene is an instance of a particular 'allele', e.g. a numeric value or binary bit [MF94].

The terminology above is borrowed from the world of biology, and so it is with the genetic operators, responsible for generating 'child' solutions from the initial population such that each iteration of the GA facilitates the evolution of solutions, hopefully towards a global optimum. The first operator is the crossover. This is responsible for 'mating' two parent chromosomes, applying some crossover mask such that the child chromosomes each share characteristics of both parents. In this respect, we expect that two 'fit' parent solutions will produce a good (perhaps better) child solution. Figure 2.2.1 shows the three most common crossover operators. The single- and two-point techniques choose one and two 'crossover points' respectively, with each point conceptually toggling which parent the child chromosomes will inherit the remaining genes from. Uniform crossover generates a randomised bit-string crossover mask, such that each offspring has genes sampled uniformly from the two parents [Mit97].

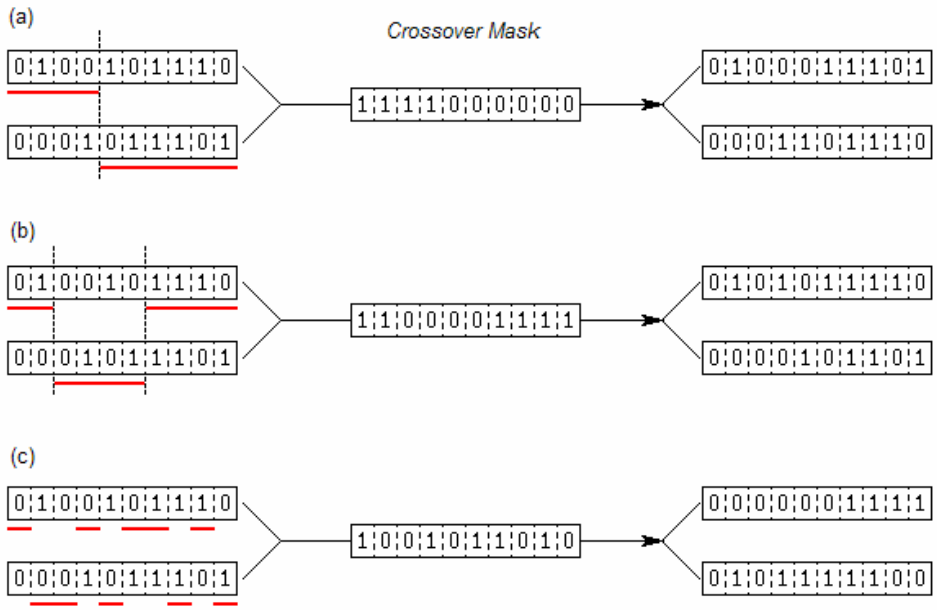

Figure 2.2.1: Crossover operators. (a) Single-point crossover, (b) Two-point crossover, (c) Uniform crossover

The second of the basic operators is mutation. Each gene in a chromosome has a small chance of being altered with each iteration of the simulation. The new value is not chosen in an informed manner, therefore the system is led into some unguided exploration of the potential search space. The mutation rate is an important parameter in any GA-inspired simulation, and a suitably high mutation rate is often required to prevent the algorithm converging upon a non-optimal local optima.

The fitness of each candidate solution is determined using a fitness function. Often this is a simple equation, where a simple statistical measure of accuracy will reflect the fitness of the solution. With evolving agents such as those in this project, the overall performance of the agent over its lifetime will accurately reflect the fitness of the chromosome. With each candidate's fitness evaluated, a process of selection returns the population to its size before the crossover. A common form of selection is tournament selection, where two random candidates are chosen to compete. With some probability p the fitter candidate is selected, and with probability $(1-p)$ the less fit candidate is selected [Mit97].

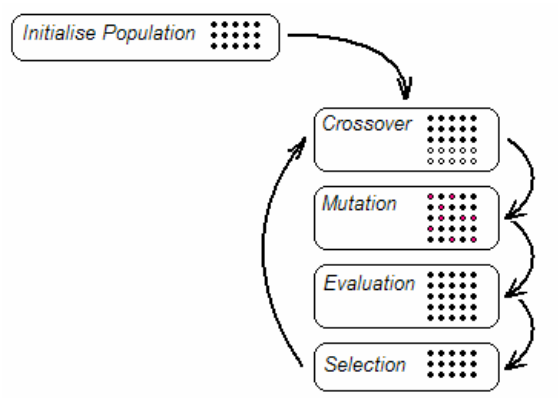

Figure 2.2.2: The standard Genetic Algorithm life-cycle

Other evolutionary algorithms exist, often related to the original GA in some respect. For each member of a population of size *NP*, the Differential Evolution [SP95] technique utilises several random members of the population of possible solutions to determine the solution vector values for an evolved child. For each member of the population, a temporary candidate is evolved based upon the selected member of the population (the parent), and three other randomly selected candidates. A recombination process is applied whereby at least one parameter in the child solution assumes the value of the 'evolved' candidate, and the other parameters have a probability of *CR* (the crossover rate) of taking on the evolved value, and a probability of (1-*CR*) of assuming the original parent value. Parent and child solutions then compete for inclusion in the next generation via basic tournament selection (the fittest candidate wins).

The simulation presented in this project relies upon a behavioural architecture to dictate the timing and circumstances under which crossover and mutation (reproduction) occur. Agents are driven to mate by their learning architecture when the circumstances are correct. The fitness of an agent is determined implicitly by its success in the simulation – agents better suited to survival in an environment will survive, those which are not will perish. In this way, the simulation implements a selection mechanism far closer to natural selection and evolution than a standard GA.

2.2.2. Is Alife a Useful Scientific Field?

The use of Alife is widespread. What was initially a series of experiments to discover whether naturally inspired techniques could accurately replicate natural evolution/learning, is now a large and growing field of research. Alife techniques are used as touched upon above; parameter optimisation through adaptive heuristic search is a recognised, efficient optimisation technique. An increasingly popular use of Alife however, is to try and replicate natural processes in order to find answers to the mysteries biologists are scratching their heads over.

[Will06] notes the application of Alife to two research projects. The first [PS02] sees an investigation into the leg placement of a fruit fly used to model the leg placement for a mobile robot. The second [ABB02] sees a simple computer simulation used to test hypothesises on the hawk moth *Deilephila elpenor*. This second example contrasts with the first in that the simulation yields a deeper understanding of the biological organism, rather than using the real organism as inspiration for improved simulations.

A further example of computer simulation used in this manner is the work presented in [TSC06], where the authors document a simulation of honeybee colonies. Honeybee colonies rely on a group of foragers to collect nectar from flowering plants, and receivers to store away the nectar in hives. "The size of the two workgroups (foragers and receivers) are precisely regulated by dances performed by forager bees, a process that represents adaptive behaviour of a super-organism" [TSC06,p1]. The authors simulated the interaction between these two types of bee in order to test hypothesises on the advantages of larger colonies in collecting nectar.

2.3. Agent Architectures & the Subsumption Architecture

The term 'agent' is often used to describe each of individual part or component of a complex system, defined as "group or organization which is made up of many interacting parts". [MM02] The term is used to describe the component parts of complex systems including economies, immune systems or the global climate [MM02], but in the case of this project an agent is an individual simulated organism living within our ecology with its own chromosome and unique, learned behavioural properties. In this respect, we are well aligned with the definition of [Rus02]:

> *"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators"*
>
> [Rus02,p32]

The aim of this project is to extend previous work by Simon Williamson and Sam Fairhurst, whereby the learning of interacting agents in a multi-agent environment was studied. The agents relied upon a sensory perception of their environment to determine the best course of action for the next and future points in time, and were equipped with an implementation of Brooks' subsumption architecture.

The subsumption architecture is an example of a behaviour-based control system. In the early 90s, Pattie Maes attempted to make explicit the distinction between traditional knowledge-based architectures and progressive behaviour-based approaches [Maes93]. A summary of these observations is provided in figure 2.3.1.

Knowledge-based AI (KBAI)	Behaviour-based AI (BBAI)
Model isolated competencies	Multiple integrated competencies (move, avoid collision etc.)
Closed - no direct environmental interaction	Open - situated in environment
Singular - one problem at a time	Autonomous, must deal with many (conflicting) problems at once
Built upon stataic declarative knowledge structures	Empasis on resultant behaviour, not internal 'knowledge'
Non-adaptive	Developmental - continual adaptation
Dependent upon symbolic internal representations of environmental perception	

Figure 2.3.1: Characteristics of KBAI & BBAI. [Maes93]

Further attempts to classify agent architectures have been made. [Mat01] defines four categories:

- **Reactive Control**: Inspired by biological 'stimulus-response' concept, tightly couples sensory inputs and effector outputs, ensuring fast response. Cannot learn from experience over time.
- **Deliberative Control**: Uses full array of sensory input and internal knowledge base to perform search over entire search space. Performs planning to determine best path to goal state.
- **Hybrid Control**: Combines reactive and deliberative components. Deliberative controller plans ahead while reactive module deals with immediate needs, perhaps due to unexpected environmental changes. Difficult to implement an effective 'power sharing' scheme between components.
- **Behaviour-based Control**: Bottom-up architecture comprises a series of task achieving behaviours. Behaviours are defined as "observable patterns of activity emerging from interactions between the robot and its environment". [Mat01,p6] Behaviours are defined and added until their interaction results in the correct operation of the agent as a whole.

In the paper which introduced the subsumption architecture to the AI community [Bro85], Brooks addresses the differences in approach between traditional top-down AI and the bottom-up approach illustrated by behaviour-based approaches such as the subsumption architecture. [Wal97] describes the top-down approach as a pyramid, where top level tasks are the end-game, and tasks to achieve this goal are split into smaller sub-tasks until a system is capable of achieving the complete task. The resultant systems are

bulky and rigid, with less room for adaptation than a looser decomposition of the task. These systems rely upon an internal symbolic representation of the environment, with search methods employed to plan action sequences to traverse the state space from the initial state to the goal. [Rus02]

2.3.1. The Subsumption Architecture

In [Bro85], Brooks illustrates the decomposition of a behaviour-based control system. He outlines a synopsis of the driving requirements for a robot control architecture, including the need to deal with multiple conflicting goals effectively, accepting and utilising multiple sensory inputs, robustness and additivity (the ability to deal with additional behaviours/sensors). The design ethos is detailed in Brooks' belief that "complex behaviour need not be the product of an extremely complex control system" [Bro85,p3]. Additional design assumptions are that relational maps are desirable to prevent cumulative errors in positioning, and that sensor failure should have the effect of disabling the control system.

Brooks suggests a decomposition of the problem domain into a number of 'levels of competence' [Bro85,p6]:

- Level 0: Avoid contact with objects
- Level 1: Wander randomly around the environment
- Level 2: Explore by 'spotting' objects and heading for them
- Level 3: Map the environment and plan routes
- Level 4: Notice changes in the environment
- Level 5: Perform tasks related to certain identifiable objects
- Level 6: Plan changes to the world
- Level 7: Reason about behaviour of objects

Each level is designed to ensure the previous level is catered to with priority, as these lower levels represent the most basic of needs.

In designing the system, Brooks suggests a robust methodology whereby we begin by implementing a 'level 0' control layer and debug it thoroughly before proceeding to implement further levels. The higher control layers are able to 'inject' data into the normal data flow of the lower levels layers [Bro85]. In this manner a fully competent control system is built up, with each layer operating unaware of others, but reflecting the desired actions of higher levels when the higher levels wish to subsume the lower levels.

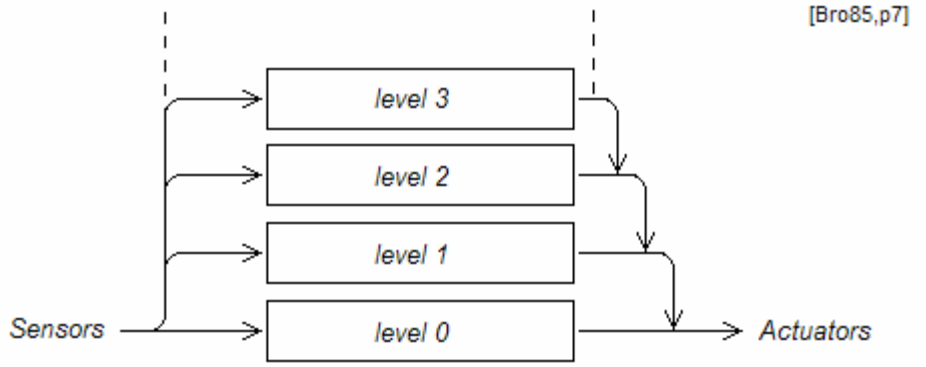

Figure 2.3.2: "Higher levels subsume the roles of lower level layers when they wish to take control", [Bro85,p7]

Brooks successfully implements a level-2 capable robot control system. The paper illustrates the actions taken by a robot equipped with this system with and without the level-2 layer activated. The robot

successfully navigates the environment and carries out commanded goals. [Bro85]

2.4. The Foundations of Reinforcement Learning

2.4.1. Markov Decision Processes

The subsumption architecture, along with the majority of learning architectures explored in this review, assume the standard Markov Decision Process (MDP) framework for reinforcement learning. First introduced by Ronald Howard [How60], the MDP framework considers a sequential decision problem, where the quality (utility) of a solution depends on a sequence of actions, rather than a single decision. The following summary uses the notation presented in [Rus02] as a foundation.

The standard Markov Decision Process operates on a set, S, of distinct states an agent may perceive. A transition model (e.g. $T(s,a,s')$) represents the probability of an agent in state s ending in state s' through action a. Due to the sequential nature of the decision problem the utility function for an agent will rely not on any single decision, but a sequence of transitions. A reward function $R(s)$ represents the reward for moving into state s, and a utility function will consider the reward earned in each state encountered in an expected run.

A fixed action sequence will not solve the majority of problems, as the probability of reaching the desired state with each move is not 100%. The MDP procedure instead yields a *policy*, specifying the best action to take in any reachable state. The policy, denoted π, will define $\pi(s)$: the recommended action for an agent in state s. The policy represents a complete mapping from the range of possible stimuli to recommended responses, and therefore represents a reactive control system, as defined in the previous section. The environment is stochastic in nature; therefore each run with any policy leads to a different environment history. However, we can find an optimal policy, π^*, by maximising the *expected utility* of the policy.

Two systems for assigning utilities to action sequences exist:

- **Additive Rewards**: simply add successive rewards.
  - $U_h([s_0, s_1, s_2, ...]) = R(s_0) + R(s_1) + R(s_2) + ...$
- **Discounted Rewards**: place higher value on rewards in the near future.
  - $U_h([s_0, s_1, s_2, ...]) = \gamma^0 R(s_0) + \gamma^1 R(s_1) + \gamma^2 R(s_2) + ...$

The use of discounted rewards offers prevents the generation of policies which never reach a terminal state. A *proper policy* will always reach a terminal state, and discounted reward systems give preference to policies which terminate in the quickest time.

Several algorithms exist to calculate an optimal policy. Value iteration and policy iteration are the two most widely used. Value iteration calculates the utility of each state individually and then uses the expected utility from each state to choose the optimal action to take in any situation. The utility of a state is effectively "the expected utility of the state sequences which might follow it" [Rus02,p619]. Formally:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| \pi, s_0 = s\right].$$

[Rus02,p619]

The true utility of a state is therefore simply the above value calculated using the optimal policy, $U(s) = U^{\pi^*}(s)$. Using this value, we can choose the action which maximises the expected utility of the next state. The *Bellman equation* encapsulates the relationship between the utility of a state and the utility of its successor:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s').$$

[Rus02,p619]

The value iteration takes the fact that for *n* states we have *n* Bellman equations to resolve, and *n* unknowns. The Bellman equation is non-linear, so algebraic techniques are ineffective. Instead an iterative value propagation procedure is employed. For each state, initialised with a random utility, the *Bellman update* recalculates the right-hand side of the Bellman equation for a state and plugs the result into the left-hand side, hence updating utilities for neighbouring states. This process continues until equilibrium is reached.

Policy iteration is a more efficient mechanism, involving a repeated cycle of evaluation and improvement upon the initial policy. The policy is evaluated, updating the state utilities. Policy improvement involves considering each state in turn and determining whether an alternative action could yield a greater utility, based upon the updated utilities of neighbouring states. Because the policy is fully specified at each stage, the 'max' operator is not necessary, and a simplified set of Bellman equations may be solved using linear algebra:

$$U(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U(s') .$$

[Rus02,p625]

2.4.2. Reinforcement Learning Techniques

> *"The task of reinforcement learning is to use observed rewards to learn an optimal (or nearly optimal) policy for the environment"*
>
> [Rus02,p763]

The standard Markov decision process considers an agent with complete knowledge of the environment. In reinforcement learning techniques, no knowledge of the environment is assumed.

In passive reinforcement learning, the policy of an agent is fixed. The task is to learn how good the policy is. This is done via a series of trials using the policy, beginning at the start state and halting when the terminal is reached. Each move will yield some reward, which contributes to the policy utility via discounted rewards, as illustrated in the previous section. In direct utility estimation, this mechanism is ignored. A running average utility is stored for each state and is updated at the end of each trial, however the utility of subsequent states is not considered and therefore this mechanism converges very slowly. [Rus02]

Adaptive dynamic programming makes use of the connections between states. A transition model is learned as the policy dictates actions from one state to another. A set of Bellman equations is generated, expressing the connections between states, and these are solved using a policy iteration mechanism. A precursor to Q-learning, temporal difference learning, reasons that we needn't solve all these Bellman equations each time evidence is collected. Simply put, the utilities of the two states involved in a transition are updated, with no need for a model of the environment:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)) . \qquad \alpha = \textit{learning rate}$$

[Rus02,p769]

2.4.3. Q-learning

Q-learning is a form of temporal difference learning which uses an alternative action-value representation to the methods presented above – the concept of a Q-value is introduced. $Q(a,s)$ represents the value of performing action a in state s. This representation is equivalent to storing an extra bit of information in the utility function, and effectively removes the dependence upon a model, hence Q-learning is considered a model-free method.

$$U(s) = \max_{a}(a, s) .$$

[Rus02,p775]

A similar equation (a) to the Bellman equation ties together consecutive states, allowing us to update the model with new feedback as with adaptive dynamic programming, however the use of $T(s,a,s')$ requires a model of the environment. Temporal difference concepts (b) can remove this dependency.

(a) $$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s') .$$

(b) $$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s))$$

[Rus02,p776]

2.5. The Subsumption Architecture and Reinforcement Learning

2.5.1. Review of [Maes90]

The concept of reinforcement learning has been applied to the subsumption architecture. In [Maes90] regular collaborators Pattie Maes and Rodney Brooks, present an algorithm employing the concepts of negative and positive feedback to facilitate advanced learning behaviour in a behaviour-based robot. The algorithm was presented in response to widespread criticism of the subsumption architecture, claiming the architecture would not afford learning about its surroundings without a degree of hard-coding by the designer.

The algorithm equips each behaviour with a learning module, achieving fully distributed learning without any central learning component.

The authors initially summarise the behavioural structure of the subsumption architecture, but note the difficulty encountered when deciding which layer of control will be active at a given time. In traditional behaviour-based systems this problem was solved by pre-compiling the control flow and priorities among behaviours by hand [Bro85] or automatically [Ros86], both cases involving some 'switching circuitry', completely fixed by the designer at compile time.

The driver for the Maes and Brooks algorithm is the need to learn which behaviour becomes active, and at which point. The result of their work is a completely distributed learning algorithm. There is "no central learning component", "each behaviour tries to learn when it should become active" [Maes90,p1].

> *It does so by (i) trying to find out what the conditions are under which it maximises positive feedback and minimises negative feedback, and (ii) measuring how relevant it is to the global task (whether it is correlated to positive feedback).*
>
> [Maes90, p1]

The learning task, as summarised in [Maes90] involves a robot with a vector of binary perceptual conditions, a set of behaviours (set of processes using sensing and actions), a positive feedback generator and a negative feedback generator. The learning task is to "incrementally change the precondition list of behaviours so that gradually only those behaviours become active that fulfil the following two constraints"

- **Relevant** behaviour is positively correlated to positive feedback and not positively correlated to negative feedback
- **Reliable** behaviour receives consistent feedback (probability of positive (negative) feedback close to 1 (0).

Requirements imposed on the task are that the algorithm should be distributed, and that the following constraints should be considered in order to ensure the system is feasible for real-world applications (e.g. robots). The algorithm should deal with noise, be computationally inexpensive, and support re-adaptation in the case where the robot itself changes, or the environment changes.

Assumptions are made. The authors assume that some conjunction of preconditions exist for each behaviour such that the probability of positive (negative) feedback lies within some parameterised boundary from 1 (0). The assumption is made that feedback is immediate, and does not involve action sequences. A final assumption is that only conjunctions of conditions can be learned. These assumptions reduce the search space for an agent with m binary perceptual conditions and n behaviours to $n * 3^m$.

The Maes & Brooks algorithm initialises the machine to a state where each behaviour has a minimal precondition list of those conditions required for the behaviour to execute. Each behaviour stores data (Figure 2.5.1) regarding the number of instances of positive/negative feedback both when the behaviour is active and inactive. These statistics are initialised at some value (e.g. N) and decayed over time (by

multiplication with $N/N+1$) in order to ensure that older experience has less impact on the learning process than newer data.

	active	not active
positive feedback	j	k
no positive feedback	l	m
	active	not active
negative feedback	j	k
no negative feedback	l	m

Figure 2.5.1: Behaviour relevance matrices for positive and negative feedback

Three measures are employed in order to make use of these statistics in the learning task. The correlation between positive feedback and the action of a behaviour is derived from the Pearson product-moment correlation coefficient, and is given by:

$$corr(P,A) = \frac{j_P * m_P - l_P * k_P}{\sqrt{(m_P + l_P) * (m_P + k_P) * (j_P + k_P) * (j_P + l_P)}}$$

[Maes90, p3]

It is a "statistical measure of the degree to which the status of the behaviour (active or not active) is correlated with positive feedback happening or not" [Maes90, p3]. The value ranges from (-1, 1), where negative values represent a negative correlation (feedback is less likely when the behaviour is active) and visa versa, with a correlation of zero representing no correlation. The value of *corr(N, A)* is calculated similarly.

Using these values of correlation the relevance of a particular behaviour can be defined, then used to determine the probability of the behaviour becoming active:

$$relevance = corr(P,A) - corr(N,A)$$

[Maes90, p3]

This relevance ranges from (-2, 2), with more relevant behaviours assigned a greater probability of becoming active at any point in time. From these statistics, a measure of the behaviour's reliability can also be calculated, ranging from 0 to 1, higher values imply that the behaviour receives consistent feedback. The authors' intention was for the reliability to be used to gauge whether the behaviour should be improved (e.g. learn further preconditions).

$$reliability = \min\left(\max\left(\frac{j_P}{j_P + l_P}, \frac{l_P}{j_P + l_P} \right), \max\left(\frac{j_N}{j_N + l_N}, \frac{l_N}{j_N + l_N} \right) \right)$$

[Maes90,p3]

An additional set of statistical matrices are maintained for a *monitored precondition*. The correlation between the state of a precondition and positive/negative feedback can be calculated in a similar manner to the above equations, such that when a particular precondition is sufficiently correlated to feedback it is adopted, with a desired value, on or off, depending on whether the correlation was with positive or negative feedback respectively.

[Maes90] continues to summarise the global parameters of the method:

- **Correlation** level required for precondition adoption
- **Time** a precondition should be monitored for before attempting adoption
- Desired **reliability** of a behaviour

The final section of the paper introduces an experimental robot using the new algorithm. A six-legged robot was built with the goal of making it learn to walk forward. The task is very complex, and there was plenty of research into efficient walking gaits to compare the learned behaviour to. The amount of forward

motion was input as positive feedback, and negative feedback provided by backwards motion. The experiment was successful. The robot learned to walk efficiently, adopting a 'tripod gait'. [Maes90,p5]

2.6. Learning New Behaviours – Macro Learning

Simon Williamson's work in [Will06] focused a good deal of attention on the process of learning new behaviours. Simon's project implemented a standard and reinforced macro learning procedure, utilising feedback to maintain a 'score table' for pairs of behaviours in a particular architecture. A particularly strong pair of behaviours, i.e. two consecutively executed behaviours which generated a good level of positive feedback, would be used to construct a macro behaviour encapsulating both behaviours in a single behaviour, with a single learning component. Nested definitions could lead to infinitely large macros.

Simon's work included a detailed review of one particular system of macro learning – the options framework. An option, as presented in [Sutt09], comprise a state-action policy, π, a termination condition, and a set of initial states, *I*. A policy may become active when the agent is in a state $S \in I$. The option policy is followed until the termination condition evaluates to true. [Will06] presents a summary of a modified Q-learning framework implementing reinforcement learning for the options framework.

2.6.1. Hierarchical Macro Learning Framework – [And98]

In [And98] the author, David Andre, presents an algorithm utilising hierarchical macro learning and asynchronous value iteration methods such as prioritised sweeping, which he then implements and tests within a simple environment. Other similar research was being conducted by Sutton, Precup and Singh in the late 1990s, with this work closely related to that presented in [Sutt99], where the options framework allows learned macros to be modified in part. In criticism of the options work, the author notes that "their method of learning options is somewhat undesirable because it still requires the user to specify explicit subgoals". [And98, p2] The method presented by Andre automatically learns hierarchical behaviours using only a list of possible subgoals, removing the dependence on explicit user input.

Andre's work is based upon the standard Markov Decision Process (MDP) framework for reinforcement learning [Kael96], see previous section introducing this framework.

Model-free methods such as Q-learning are not considered, instead model based methods are used to "utilize the model of the environment to perform value propagation steps (planning) during learning" [And98, p3]. An iterative process of action, reward value updates, and value propagation takes place, with methods differing in which states undergo the value update after each iteration. For example, all values could undergo the update procedure, or as with prioritised sweeping, just those states most likely to undergo large changes.

The format of macro learned by the system is constrained to simple do-until loops, with a preset number of actions (the behaviour specification). The actions can be built in primitive actions, or other macros. Additionally, a termination function is defined as a conjunction of literals from the same set as those used in the behaviour specification, derived from the state, *s*, as a set of features, *F*. This feature set may include perceptual information, allowing a macro to terminate when a wall is sensed in front of the agent, etc.

An initial attempt by Andre's team used another conjunction of literals as a precondition list (similar to that used by the subsumption architecture for behaviours), whereby iterative improvement of the conditions and the behaviours was supposed to lead to increasing performance. This method failed, primarily due to a lack of coherence between the parts of the system for learning the preconditions and those for developing the behaviour. A second system addressed these problems.

The algorithm Andre presents treats each percept assignment, $p = v$ (e.g. $row = 1$ or $column = 2$ in a 2-D grid world) as a potential sub-goal in the world, and specifies for each, a set of states consistent with the sub-goal. A path to the sub-goal states is calculated from other states using dynamic programming to calculate the discounted likelihood of reaching the sub-goal states for each state, $L_{p=v}(s)$. The algorithm also tracks the number of steps from each state to the sub-goal, $d_{p=v}(s)$. Using these statistics a score,

$y_{p=v}$ is calculated for each percept assignment, the score representing the "likelihood-weighted number of states at which a decision about what action to take must no longer be taken":

$$y_{p=v} = \sum_{s:d_{p=v}(s)>1} L_{p=v}(s)$$

[And98, p5]

A subgoal is chosen given the scores, and a constrained behaviour is developed to achieve the subgoal.

- Build a set, *R*, of states sufficiently likely to reach the goal;
- Build a set number of decision rules of the form $[\mathbf{if}(percept = value)\mathbf{then}(action)]$, such that the rule/action combination minimise the *policy loss* at each stage;
- After each decision rule is added, remove the states whose actions are specified by the rule from *R*;
- When the set number of rules have been added, a default action is defined for the remaining states in *R*.

The policy loss is defined as:

$$Ploss(f = v) = \min_{a \in A} \sum_{s \in R: s_f = v} \left\{ bestval(s) - \sum_{s' \in S: p(s,a,s')>0} p(s,a,s')L(s') \right\}$$

[And98, p6]

$$bestval(s) = \max_{a \in A} \sum_{s' \in S: p(s,a,s')>0} p(s,a,s')L(s')$$

[And98, p6]

In testing, Andre and his team tested the macro learner offline in a simple 4x8 world. Given a start state and single goal state the algorithm successfully identified a pair of useful subgoals (atTop=1 and atBottom=1) and developed a pair of macros to achieve these goals.

Finally, the author addresses the issue of integrating his method with standard reinforcement learning methods. His proposal follows work by Sutton et al. [Sutt99], and the essence of his approach is that the macro learning algorithm should be run at predefined intervals during the reinforcement learning run. Each time the algorithm is executed, a larger area of the state space will have been explored, resulting in increasingly complex macros. As each macro is defined, it can be added to the action set utilised by the reinforcement learning algorithm.

Andre's hierarchical learning framework appears to reflect the system implemented in the Java implementation very closely (or visa versa!). [Will06] presents very good results with both an immediate reward and delayed reward macro learning architecture.

3. Software Design

3.1. Re-implementing the Simulation

The simulation of species emergence requires that the implementation allows for the correct degree of evolution and adaptation for divergence to be observed given the right environmental factors. As summarised earlier the previous projects leading up to this work have worked upon developing and extending an implementation of Brooks' subsumption architecture in order to observe adaptive emergence and learning. Extensions to the subsumption architecture such as disjunction precondition adoption, macro learning and affective states were included with mixed success.

In deciding how to implement a simulation based upon this work in order to facilitate speciation, the suitability of the existing Java implementation was questioned. The system is limited in its scope for environmental variation, and the several 'bolt-on' extensions made up to this stage have resulted in a somewhat unfocused code base. The decision to re-implement was justified as follows:

- System already **strained** – experiments inefficient
- **Extensibility** – extending environmental simulator difficult
- **Efficiency** – a logical language would be better suited to the implementation of finite state machine implementations of individual behaviour layers
- **GUI** programming – Java is slower than Visual Studio .NET languages for GUI development
- Code base – **messy** at present, there is a need to bring code together
- **New functionality** – changes at a low level require modifications throughout the entire system

Several languages were considered, each with advantages for the simulation. Particular emphasis was placed upon the need for a flexible graphical user interface. The concepts explored in chapter 1 of this paper all dictated that environmental diversity was a requirement for speciation, with geographic isolation responsible for a good deal of speciation observed in nature. This need led to the conception of a 'world designer', allowing environmental features to be drawn directly onto a map for use in simulations. The following languages were considered:

- **Prolog** – logical language lends itself to efficient behavioural implementations
    - **P#** [PS07] - an open source cross-language compiler for Prolog generates highly efficient pre-compiled C# libraries from Prolog source code. Prolog could be used to code behaviours, and libraries could be compiled and called from within a C# simulation environment.
- **C#** - IDE facilitates fast GUI development. Object oriented language makes code easy to modularise and understand, with the concept of an agent easy to implement as an object. Simple threading framework for concurrent implementation.
- **LISP** – as chosen by Brooks for his code in [Bro85]. This code could be adapted and used as a base.
- **Java** – Current code base could be re-used for speed of development. Object oriented, as with C#.

C# was chosen as the implementation language. The similarities of C# to Java, at least regarding syntax, were a driver for this choice, as elements of the Java code base could easily be translated into C#. The possibility of improving performance using Prolog libraries compiled using P# was also attractive.

This section first presents a summary of the design cycle and basic architecture of the system, and later addresses the modifications deemed necessary to the existing architecture in order to facilitate speciation and faster learning. Several bugs in the existing implementation are discussed, with an evaluation of the

effect these bugs had on the results observed.

3.1.1. The Development Cycle

The process of re-implementing the simulation began with a thorough review of the existing code. The UML diagrams presented in [Will06] were used for this, and necessary or desirable changes were noted as documented in the following section. New development was targeted at: a) Debugging the existing solution to yield stronger results, b) Modelling perception and targeting mechanisms more effectively, c) Modelling a more varied environment to facilitate speciation.

After an initial period of development implementing the simulator and subsumption architecture model, development was carried out in an iterative implementation-evaluation cycle. This system could be criticised for resulting in a similarly fragmented and messy code-base to that being replaced, but a design considering all future additions was used from the start to prevent such an outcome.

3.1.2. Architectural Design

With a proven foundation in place, such as that developed in [Will06] and [Fai04], there was no reason for modifying the basic architecture of the system. The code implements the subsumption architecture as presented in chapter 9 of [Will06]. The modifications made to the architecture over the duration of the work presented in [Will06] were not implemented, as these capabilities are not necessary for the evolutionary process to execute effectively enough to simulate specialisation.

The simulation is built around a class, World, (see Figure 3.1.1) which represents a map of size 100x100 units. Each point on the map is represented in the system as a MapPoint object. This class encapsulates all operations on an individual point on the map, including the enforcement of physical rules such as non-negotiability. Each MapPoint has a surface type, a class encapsulating all properties shared by cells of the same type, for instance colour, negotiability and, if applicable, speed of negotiation. Surface types can also have some grazing value – a predefined level of energy and agent may harvest by grazing on the surface.

Each MapPoint object, stored in a two-dimensional array representing the world, has a resident and visitor placeholder for items such as agents, shrubs, and rocks. Resources and obstacles are defined as residents – they do not move – and agents move around the map as visitors. All items, including agents, extend the WorldObject class, which provides methods for maintenance of energy levels used by all sub-classes.

The basic agent architecture is presented in Figure 3.1.2. Agents are equipped with a perception of their environment at the beginning of each 'tick'. This perception drives the decision making process for the subsumption architecture. The agent has a 'Brain' which is simply the class responsible for drawing together the behavioural hierarchy and driving the subsumption process. The brain stores an ordered list of behaviours. The behaviours are each driven by a 'BehaviourLearningModule', named following the previous implementation convention. This class is responsible for monitoring a list of preconditions which must all be true for the behaviour to become active. A single new condition is monitored, as detailed in [Maes90], with a view to becoming a member of the adopted preconditions vector should the correlation between the condition and positive feedback prove high enough.

The behaviours implemented in the system once again are the same as those implemented in [Will06] and [Fai04]. Some modifications have been made however. The 'gap maintenance' behaviour appeared to have little function in the system, and therefore it was removed. Also, in order to encourage usage of the 'move towards' behaviour, a distinction between how far an agent can see and how far it can move was added, utilising an eighth gene, "ACTION_QUOTA". This gene defines how many actions an agent can take in any particular move. For behaviours such as 'move towards', the agent is permitted to search its full range of sight for a target, but there is no guarantee that the action allowance he has will get him to the target. In

order to prevent destructive side effects for the system, this system is bypassed for the mate and eat actions. Figure 3.1.3 summarises the behaviours and their purposes.

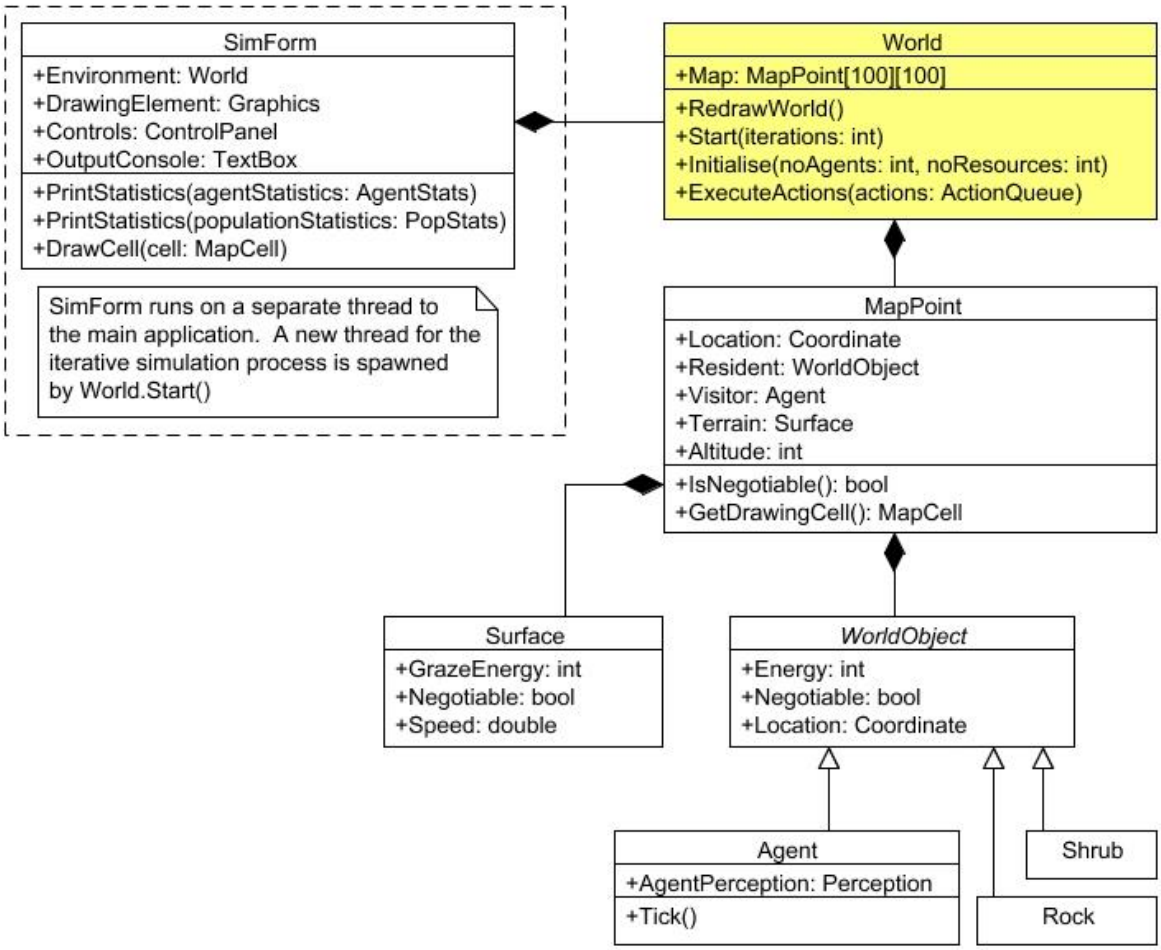

Figure 3.1.1: UML model of simulation implementation

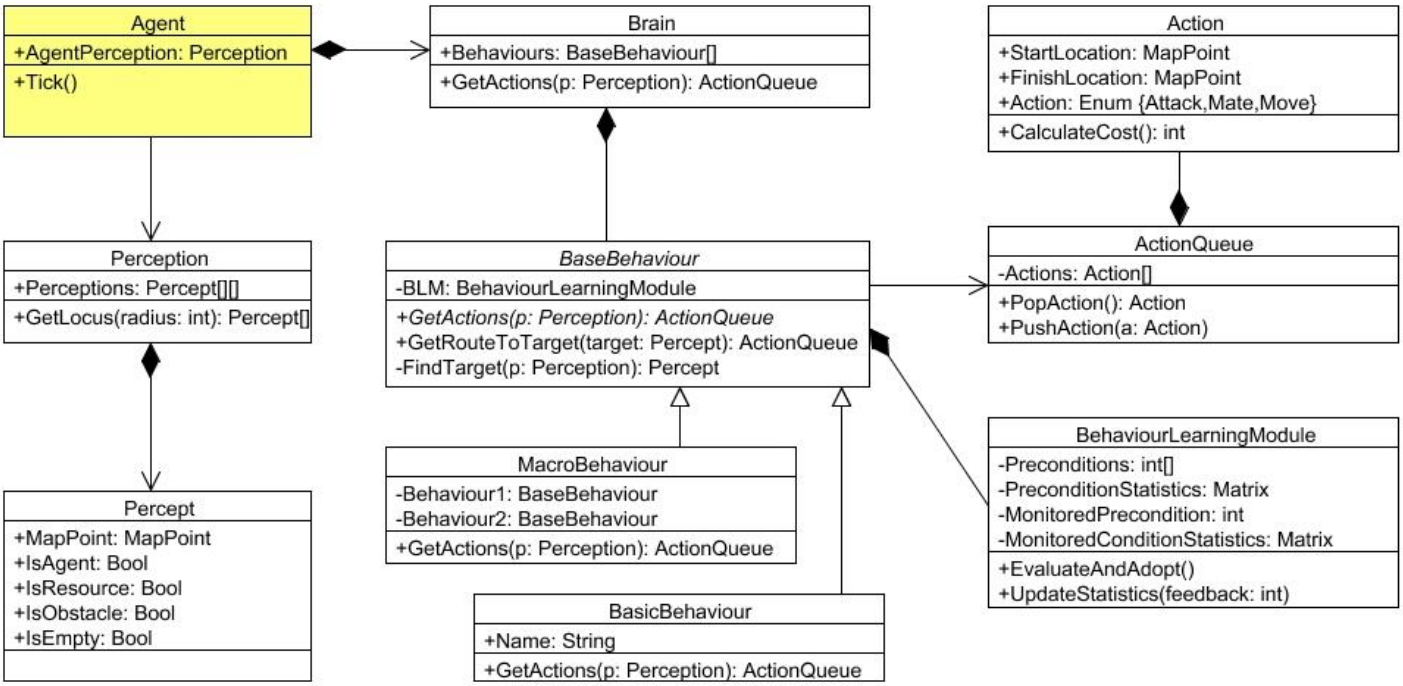

Figure 3.1.2: UML model of agent architecture

Behaviour	*Action*
RandomMovement	chooses a random target and moves as close to it as possible
MoveTowards	considers and chooses a target and moves towards it
MoveAway	moves away from a carefully considered target
Eat	attempts to eat the selected target object (plant, resource or terrain)
Mate	attempts to mate with a selected object

Figure 3.1.3: The five behaviours implemented in the simulation

Section 10 of [Will06] presents some worrying results, concluding that the basic Maes and Brooks algorithm is not capable of sustaining a stable population over a period of 20,000 iterations. It was hoped that by manipulating the simulation parameters this could be resolved.

In actual fact, it was found that the system was very sensitive to the choice of parameters, particularly parameters which had not been questioned before. Appendix A presents a list of the system parameters. Those which had a particularly high impact on the sustainability of the population were the regeneration rate, allele energy and the living cost base. This last parameter represents the variable in the calculation of an agent's living cost as follows:

```
foreach gene in agent's chromosome

      b = livingCostBase
      a = alleleValue
      livingCost = living cost + b^a
```

This had not been considered a parameter of the system before, but is very important. Setting this value too low inevitably leads to a population explosion, and too high leads to an extinction. By creating a parameter for this value, the system can be stabilised, providing a proven test bed for other experiments.

The addition of disjunction and macro learning in the previous project led to significant improvements in population sustainability and average population levels. The reliability of results was also significantly higher with disjunction learning. For the purposes of this project however, the additional computation overheads and implementation time required to implement macro learning motivated a decision to implement only disjunction enabled learning agents.

3.1.3. Teeth Effectiveness

In chapter 16 of [Will06] the effect of modifying the simulation's interpretation of the phenotype 'teeth' is evaluated. Two response curves are considered (from [Will06,p70]):

- **Linear**: "Increasing linear connection between tooth size and the ability to successfully eat other agents. Decreasing linear connection between tooth size and the ability to successfully eat resources."
- **Exponential**: "Increasing exponential connection between tooth size and the ability to successfully eat other agents. Decreasing exponential connection between tooth size and the ability to successfully eat resources."

[Will06,p70]

The experiments carried out to test these systems found that the exponential response curve caused higher levels of foraging and lower levels of predation. In the new implementation, it was desired that the agents should exhibit highly polarised behavioural characteristics, allowing for easier classification (e.g. agents with sharp teeth should be very unsuccessful foraging, and visa versa). Therefore the exponential interpretation was implemented.

3.2. Technical Modifications

3.2.1. Concurrent implementation of the turn-based agent architecture

A criticism of the Java implementation is that the efficiency of the simulation is poor. 100,000 generation runs take a long time to execute, in part because opportunities for concurrent execution are not taken. It would in theory be possible to implement each agent within its own thread, making decisions based only on access to an exposed portion of the map – the perception. The problem with this is potentially that the agents would then be making decisions based upon information likely to change before their actions are taken, meaning that their actions would often fail.

This could be addressed by choosing agents to 'have their turn', and ensuring that at each iteration, the active agents' perception maps do not overlap. Once they have had their turn for the iteration, they are removed from the pool of candidates and the process restarts, until all agents have had their turn. This is in effect a modification for efficiency and not a change to the fundamental simulation architecture. It would be the case that the outcome of the simulation would be partially affected due to this modification, but the system is a step closer to a real-time simulation than the previous iterative turn-based system, whilst avoiding synchronisation issues associated with complete parallelisation.

3.2.2. Sexual Selectivity

As addressed in the review of section 2.1.2, the Java implementation of the simulation uses an oversimplified barrier mechanism in order to prevent inter-species reproduction occurring. This system simply added all allele values for two potential mates and allowed reproduction if the absolute difference between these two values was below a globally parameterised threshold. The implementation presented in this project compares each gene separately, allowing reproduction only if each pair of genes' allele values are close enough.

3.3. Subsumption Architecture Considerations

3.3.1. Flocking infrastructure in use in Java implementation

In an attempt to develop high order behavioural characteristics, particularly flocking, the Java implementation made use of a 'flock map' in addition to the standard two dimensional perception map used by the agents in the decision making process. This map was a two dimensional array of binary values, the same size as the perception map. A '0' represented an empty space, or an agent not operating as part of a flock. Conversely, a '1' represented an flocked agent. Potential attackers could used this map to determine the size of the group it might be facing, and prey could learn to stick close to a flock for safety as a result.

Whilst this method resulted in the exhibition of flocking behaviour, this behaviour was somewhat of a foregone conclusion as the flocking map was conceived and designed carefully with this result in mind. A consideration for this project was to avoid implementing the flock map, and rather to incorporate the scope for multiple target assessment in behaviours. By assessing the location of multiple agents in the perception array, an agent would be able to utilise the reinforcement mechanism provided by the subsumption architecture rather than relying on hard-coded considerations.

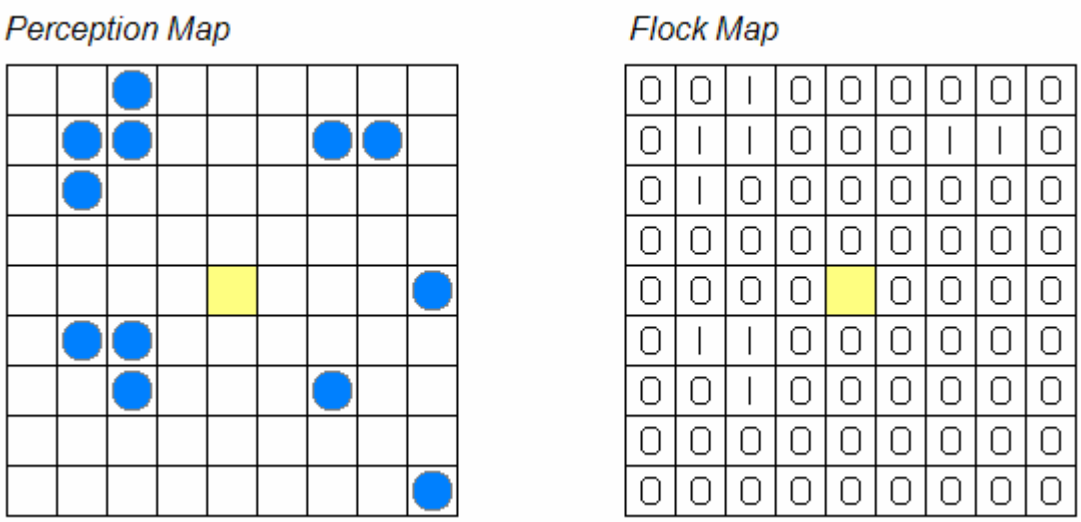

Figure 3.3.1: The flock map concept from [Will06]. The agent (central cell) will learn to choose a target from the lower left-hand corner as these are not members of a flock

3.3.2. Path-finding

All movement is very basic in the Java implementation. The previous work on this project has focused on proving the subsumption architecture can work in a basic simulation, but has generally lacked development of the simulation itself. For example, agents do not move along a path to their target, they simply jump a set distance. This makes feedback from failed movement difficult to account for, and probably affects the efficiency of the behaviour learning mechanism. Maybe integrating a basic path-finding algorithm into the movement action handler would result in better movement. Agents could have their 'intelligence' calculated from a feedback counter for all behaviours, and this would be used to determine how deep a path-finding search could go. At the very least, a path-*checking* mechanism would allow feedback to reflect how suitable the route to the target was, as well as how suitable the target itself was.

Path-checking Mechanism

The addition of more environmental variety makes very little difference without allowing the agents some mechanism with which to learn how the terrain has affected the result of a chosen action. An agent must therefore select a path to the target and monitor preconditions regarding the suitability of the path before an

action is taken in order to learn how the path characteristics have affected the outcome of the action.

To facilitate this, reuse of the BehaviourLearningModule was proposed. The module can be extended to contain more preconditions relevant to choice of path, and a BehaviourLearningModule can be instantiated once per agent to monitor preconditions which must be true for a path to be chosen. E.g. the path must be fully negotiable, and have no gradient greater than 45 degrees etc.

At this stage it is worth noting that it might be necessary to introduce some hard-coded path checking if the reinforcement mechanism implemented in the BLM is not robust enough to adopt sufficient preconditions to prevent 'stupid' action sequences being attempted. This should not occur however, as a failed action sequence should provide direct negative feedback to the monitored precondition in the agent BLM, ensuring the correct precondition negations are adopted. However, should the monitored condition not be responsible for the failure, there is a high chance the total negative feedback will result in uncorrelated preconditions being adopted regardless – perhaps all of them! This is the eventuality which would necessitate hard-coded intervention, or improvement/specialisation of the available preconditions.

3.3.3. The Precondition Monitoring System

The system previously implemented was effective at evolving architectures with fully reinforced behaviours. The behaviours each monitor a series of preconditions, with correlated preconditions becoming adopted, requiring that they evaluate to true for a given target before the behaviour become active. This mechanism relies on several assumptions:

- Our list of preconditions provide an exhaustive array of boolean evaluators by which the properties of a target can be determined and evaluated;
- The list of preconditions is monitored effectively, ensuring relevant preconditions are adopted in time to effect the agent's quality of life;
- Preconditions are re-evaluated over time providing a comfortable insurance mechanism where coincidence may have led to anomalous feedback;
- Parameters determining the level of correlation required for adoption are selected correctly.

These assumptions are generally not fully realised in the Java implementation. Previous developers have neglected to implement a mechanism to ensure uniform coverage of the preconditions, instead relying upon a sequential walk-through of the conditions in order. This mechanism might be modified to improve the speed of reinforcement, although a possible explanation for this choice might be that the agents are all operating on equal terms with the sequential mechanism, or that the preconditions are ordered optimally at design-time, ensuring that the preconditions are monitored as efficiently as possible.

An alternative, then, is to modify the monitoring mechanism to monitor more than one of the conditions at a time. Experimenting with monitoring of *all* conditions at once might yield results.

A bug related to this issue was found in the existing Java implementation during the development stages of this project, a bug which when fixed showed massive increases in speed and correctness of adoption for preconditions. Feedback to the monitoring tables for the monitored candidate precondition were being updated incorrectly (a possible cut-and-paste error), making feedback very balanced, and resulting in low correlation.

3.3.4. Target selection mechanism - Senses

Prior to selecting an action to perform, the agent must decide what object, be it an agent, resource or empty map location, it will target. The Java implementation of the subsumption architecture for this simulation utilises a uniform randomised selection mechanism to iterate over the candidates in the agent's perception of the environment. It was proposed to modify this mechanism to allow the agent to learn which of its

'senses' to give priority. The existing precondition infrastructure should allow for this mechanism, by introducing preconditions for 'eyes-first', 'ears-first', or 'nose-first'. The agent will be able to adopt the instinct to investigate interesting 'sounds' to its peripheries before selecting a target directly in front of it.

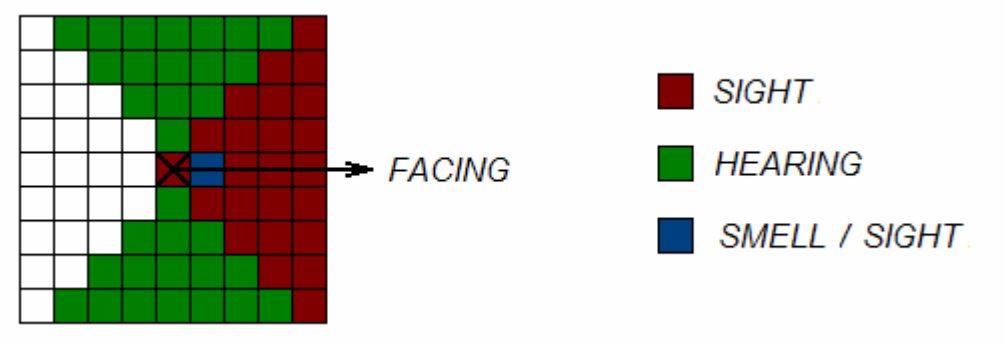

Figure 3.3.2: Potential model for modified perception sub-system

This, however, involves a deviation from the original usage of the preconditions monitoring system. Until now, these preconditions have been utilised only to gauge how suitable a given behaviour/target pair will be. My proposal is to utilise the preconditions to allow the behaviours to prioritise on perception input to the system, and therefore make a target selection more effectively.

It was hoped that this modification would lead to new behaviour – 'TurnToFace' – with the expectation that it would place priority upon hearing as sense of choice, and will target 'Something' most of the time.

3.3.5. TurnToFace Behaviour

The motivation for this behaviour is observation of natural learning and evolution. Animals observed in research for this project were noticed to place a high priority on aural stimuli, often stopping in their tracks during an attack or whilst sniffing out a trail to investigate noise from behind or to the side. An idea for a future project would be to compare the behaviours learned by a system implementing the above system of sensory perception to observed behaviours in animals. This was outside the scope of the present project, but a modified perception system could now support the implementation, and was used to add complexity to the simulation by

3.3.6. Action selection mechanism – Artificially Promoting Exploration/Reproduction

This mechanism previously included a uniformly enforced percentage of forced mating and random actions, leading to guaranteed population maintenance and a good degree of behavioural exploration. This is not desirable as it undercuts the subsumption architecture in what is effectively a hack, and not a valuable modification to the architectures mechanism. These hacks do address important issues regarding the subsumption architecture, but I believe better mechanisms can be put in place to overcome any problems. (e.g. the concept of group feedback or society feedback to overcome the mating issue)

This is difficult to implement without disrupting the feedback balance between behaviours which leads to well distributed learning and behavioural architectures which reflect the habitat of the agent.

3.3.7. Environment Complexity & Design Mode

The environment at the moment allows only for very simple characteristics of land units to be modified, therefore restricting our ability to test the architecture implementation and extensions within more complex environments. For example, there is no concept of gradient, which if implemented might allow us to observe specialised 'mountain breeds' of agent, with characteristics tailored to the conditions at higher altitudes. Perhaps agents of a type particularly likely to be preyed upon might choose higher altitudes as

attackers take longer to reach them travelling up hill.

During the design phase of this project, it became clear that the key to success was to be the potential for complexity within the environment, leading to solid simulation of the real-world conditions leading to speciation. To this end a 'design mode' was conceived, allowing complex environments to be drawn for the simulation. The key was flexibility, with the potential to develop scenarios where the environment changes during execution of experiments. Section 5.1 presents a summary of the development process for the environment designer.

3.3.8. Circular Map – Implementing a Globe

The present Java implementation was based upon a simple square world with concrete boundaries. It is interesting to consider the effect a 'wrap-around' world would have upon the actions of agents at the world perimeter. In fact, the lack of a perimeter should prevent failed actions occurring due to the boundary limitations.

4. Experimental Procedure

In evaluating the experimental results gathered during this project, there was a good deal of focus required. Each agent reports a full range of activity statistics with each iteration of the environment, and a class, PopulationStats, was defined in order to draw these statistics together and store them in an efficient, non-destructive manner allowing for processing. A review of the methods used in [Will06] shows that two primary statistic measures were used throughout the evaluation – population levels and predation levels.

4.1. Significance of Statistics

Throughout experimentation it is important to have some method by which to determine whether statistical evidence is significant. Whilst the findings of tests in such a randomised simulation as this are likely to sometimes support expectations, it is important to repeat the experiment and compute the significance of any results before null hypothesises can be rejected.

The standard test used throughout this project will be the popular t-statistic test. For two groups of results, the t-statistic represents a ratio of the difference between group means to variability between the two groups. In most cases this ratio will be used to determine whether difference in population levels in two different simulations is statistically significant over the number of tests run, or whether the difference can be accounted for by compounded error.

The t-statistic is defined as:

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{\text{var}_1}{n_1} + \frac{\text{var}_2}{n_2}}}$$

Where $\overline{X}$ = the sample mean, 'var' is the variance of the group, and n is the sample size.

4.2. Statistical Measures

4.2.1. Population Levels

The Williamson project makes extensive use of the population level with respect to time. In section 8 of [Will06], the evaluation mechanisms are summarised, and the author refers to the balance between energy input to the system and the number of agents which can be sustained by this energy. By adapting to the environment, be it resource rich or relatively barren, a population of agents can maximise the population size. In an extreme case, the inability to adapt might lead to extinction, as was the case in many early experiments.

Formally, as defined in [Will06], the population level $population(a)_i$ is the sample mean of the population level at time i for architecture a.

An extension of this concept, allowing for basic categorisation of agents based upon a gene value and environmental feature, defines $population(E = e, G = g, a)_i$: the sample mean of the population level at time i for architecture a, where E represents an environmental feature with the value e (e.g. Altitude=*7*, Terrain=*grass*), and G represents a gene with the value g (e.g. Colour=*yellow*).

4.2.2. Species Determination

Determining a species amongst a population of agents is a tricky task. As defined in the review, organisms of one species are not capable of reproducing with individuals from other species. This has been modelled

through chromosome comparison in the simulation, and this concept can be used to statistically analyse the population at a given time. By finding a group of agents sharing the capability to mate with each other, but being unable to mate with members of a similar group, we are able to say that each of these groups represents a species. Mathematically, we can determine species by identifying clusters of individuals in the multi-dimensional phenotype space. The maximum distance between any two individuals in each cluster is determined by the parameters used in the simulation to impose sexual selectivity. In the case of this simulation alleles whose values differ by more than 2 prohibit the two agents from mating successfully.

When evaluating the spectrum of alleles present in a particular population the complexity of the cluster analysis process can be reduced by paying attention to what it is one expects to observe. In the case of the experiments in this project the number of genes which will have a direct effect upon the success of an agent in a given environment is less than the full eight in each chromosome. In fact, in most cases only two or three of the genes have a direct impact upon the success of an individual. Therefore visual analysis of a scatter graph showing the gene values for only those relevant genes might allow much more efficient and flexible analysis of species emergence than over complex mathematical techniques. If visual analysis proves ineffective however, cluster analysis over the entire phenotype space can be used.

Cluster analysis is a statistical method which lends itself to the task of finding species within a population of chromosomes. The process can be implemented in several ways, some methods however are more complex than others. Given the fairly low complexity of the problem domain with the data in this project, a simpler implementation is suitable.

K-means Cluster Analysis

K-means clustering is used where one already has some idea how many clusters to look for in a particular data set. In an experiment where one is directly aiming to evolve two species, it is therefore the most efficient and applicable of the available cluster analysis algorithms. From [Stat04], k-means cluster analysis will find "exactly k different clusters of greatest possible distinction". For example, in the case of a basic speciation experiment attempting to evolve two species, the k-means method could be used with a value k=2.

The algorithm begins by generating k random clusters and moving individuals between the two groups in order to maximise the variability between clusters, whilst minimising the variance between individuals in the same cluster. Applying the k-means method to the speciation experiment, the variance calculated between the two clusters can be used to determine whether the clusters represent species proper, or simply the two extremes of a micro-evolved single species.

4.3. Procedure

Each test carried out during experimentation followed a similar procedure. Where necessary the statistical classes implemented within the system were modified to facilitate the required output. The experiments were generally executed less times than was desirable due to time constraints (a 50,000 iteration run will take 90 minutes to execute), but in general at least 10 runs were used to generate a reasonable sample. In general the following procedure was used.

- The environment designer was used to create the desired experiment set-up
- A predefined number (~200) of randomised agents were placed in random positions in the environment. Where necessary, agents with a predefined chromosome were placed manually in a particular area of the map in a fashion as close to random as possible
- A test run of between 10,000 and 50,000 iterations was executed
- Statistical output was tabulated and processed. The statistical significance of strong results was calculated and null hypothesises were rejected where the significance was lower than an alpha value

of 0.1. Graphs were used to present finding in this report.

5. Experiments

5.1. Implementing a World Designer Interface

5.1.1. Design & Implementation

If anything was certain after the analysis of requirements for this project, it was that the ability to design relatively complex simulation environments was a must. With every 'paint' program available as inspiration, the world designer was conceived. The requirements were as follows:

- Ability to quickly create diverse simulation environments with non-uniformly distributed resources
- Ability to make environmental changes during experiment execution in order to facilitate speciation
- Support for a range of environmental variables:
    - Terrain types (surfaces)
    - Resource/obstacle distribution
    - Resource regeneration (uniform and local)
    - Agent placement (random/specific placement & random/predefined phenotypes)
    - Suitable array of geographic barriers facilitating isolation

The environmental designer was therefore designed to allow the user to select one of a series of paintbrush style tools. The user can choose to place a single item, or to fill a rectangular area with a specific density (0-100%) of the object. Resources can be defined as 'seeds' (regenerating at a rate specified by a global parameter) or as one-off resources. The map can be fully compartmentalised using water or rows of obstacles, preventing agents passing a barrier. As for the agents themselves, they can be defined with a random chromosome, a predefined predator/forager chromosome, or a manually specified chromosome. In each of these cases, some or all of the individual genes can be specified as a random choice by using an 'R' rather than a particular value when specifying the gene value.

Figure 5.1.1: Design mode in the subsumption simulation application

5.1.2. Testing Process

The process of testing the environment designer was not so much a statistical process as a standard run through of test cases. Test cases for each map feature were executed. For example, agents were placed on a very small 'island' surrounded by water. The system was expected to prevent the agents from leaving this enclosure.

Each of the tools was used in order to test that the drawing facility worked, and short test runs were used to ensure regeneration of resources operated as expected.

5.2. Colour – Adapting to the Local Environment & Encouraging Speciation

After many test runs with the new implementation, several problems emerged. At the top of the list was the lack of evidence for adaptation occurring. It became clear that the Java implementation was aimed squarely at proving the subsumption architecture was capable of driving a multi-agent simulation, and that the concept of genetic adaptation was not properly evaluated. This distinction between individual *learning* and population *genetic adaptation* is very important for the concept of speciation. Research has been conducted to show that learned behaviours might be passed on through reproduction; however, the ability for an agent to reinforce its own behaviour selection mechanism does not necessarily influence the ability for a population to move towards optimality for a particular habitat.

Of course, one must question *why* the Java simulation does not demonstrate explicit adaptive behaviour. There are two key reasons and the first (lack of environmental variation) has been addressed in detail previously. The second lies in the modelling of agent chromosomes.

The Java implementation utilises seven genes to define each chromosome. Seven genes should provide enough scope for variation to facilitate specialisation, however the effect of each gene upon agent performance appears to be too low, therefore the natural selection process is somewhat hindered. The function and influence of each gene is summarised below:

Gene	***Summary of Purpose***
PERCEPTION_RANGE	Defines the size of the two-dimensional map representing an agent's perception
SIZE	The size of the agent. Simply used with other genes to determine winner of a fight
TEETH	Higher values represent sharper teeth. Determines success of eating agents/vegetation
LEGS	Determines agents agility on land. Similar effect to 'SIZE'
FINS	Used to determine agent agility in water. Not used as no water defined.
SDM	Special Defence Mechanism'. Has little effect.
GENDER	Dictates gender of the agent.

Figure 5.2.1: summary of genes used in Java implementation

By introducing a greater degree of variation into the gene pool by increasing the influence of each gene upon agent behaviour and success, it was hoped that the adaptive evolutionary process could be more conclusively observed.

To this end, a new pair of phenotypes were implemented. Drawing from the research of Kettlewell ([Kett55], see review section 2.1.1) as inspiration, the concepts of colour and camouflage were considered. As summarised in the review, Kettlewell's observations of peppered moth populations showed that the colour of an organism can be a key factor in determining the probability of attack during any given time frame. The hope was to show that the simulated effect of camouflage could lead to eventual adaptation of species to their environment. The implementation design is outlined below.

A phenotype determining the colour of an agent was added to the chromosome model. The 'SDM' and 'fins' genes were removed due to their low level of influence in the new implementation. The colour was defined as an integer allele value in the range [0..3]. The contrast between the colour of an agent and the colour of the local environment was then used to determine a probability of the agent being seen by other agents, in the range 30-100%.

The implementation of this mechanism was carried out within the 'Percept' construct introduced as part of the updated implementation (see figure 3.1.2: UML for Percept class). The constructor for the percept object calculates the probability of an agent being seen and modifies its properties to reflect this probability as follows:

- Percentage chance, *P*, of agent being seen
- Integer, *R* = *random(0..100)*

- $R < P$: Percept type set to **Agent**
- $R >= P$: Percept type set to **Unknown**

This has the effect of ensuring that the active agent will only know a target agent is residing in the location represented by the percept P % of the time.

Table shows the parameters decided upon relating to camouflage levels. Experimentation was required to ensure the desired level of adaptation.

Colour	Tarmac	Grass
0 - Green	100%	30%
1 - Yellow	90%	50%
2 - Brown	50%	90%
3 - Grey	30%	100%

Figure 5.2.2: Table showing probability of agent of specified colour being spotted on each surface

5.2.1. Hypothesis

Null Hypothesis H_{0a} : The addition of colour differentiation to the environmental simulation will have no effect upon the population levels of agents with respect to their colour. Formally:

$$population(AgentColour = X, Camouflage)_i = population(AgentColour = X, NoCamouflage)_i$$

and

$$population(AgentColour = X, Camouflage)_i = population(AgentColour = Y, Camouflage)_i : X \neq Y$$

for a statistically significant proportion of i.

5.2.2. Results

As a baseline for evaluation, this test was initially run with disjunction learning and environmental camouflage switched off. The first run showed results within 10,000 iterations, and therefore this was chosen as the run length for the experiments.

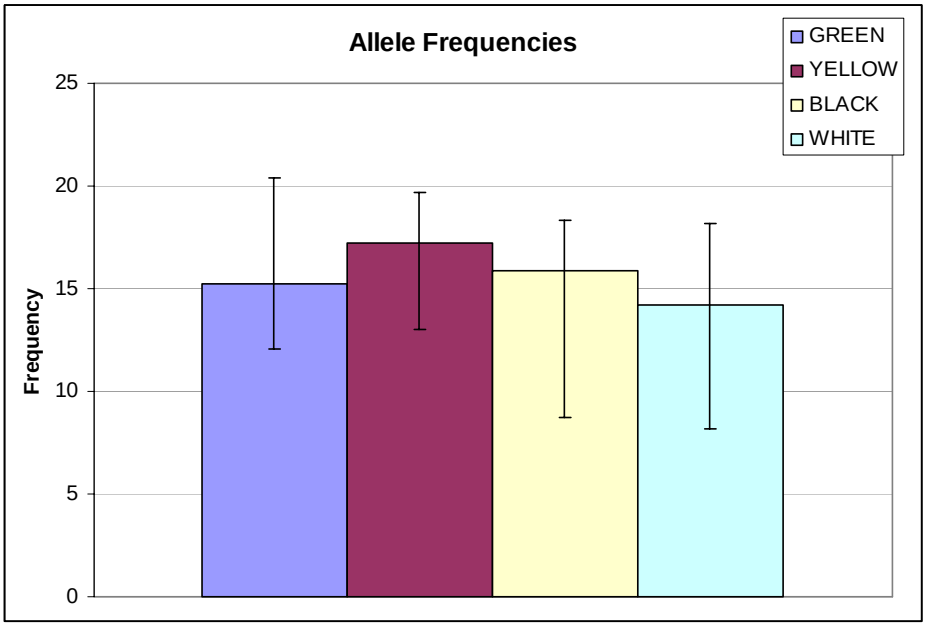

Figure 5.2.3: Allele frequencies of each colour after 10,000 iterations

Environment surface = grass

The expected outcome in this baseline test was for an even distribution of allele frequencies over the four possible values. The results support this expectancy, providing a solid base for further testing.

With camouflage enabled, initial evaluations using a random initial population of agents in a fairly resource rich environment showed higher visibility actually had a positive influence upon sub-population prevalence. This is be explained by the fact that mating occurred more frequently when the individuals were easier to spot. The agents with higher contrast colouration were more likely to mate successfully and pass on their genes. The balance between mating and attack rates were, by chance, well balanced in this case and therefore the modification to the simulation made very little difference to baseline population levels.

However, by introducing a manufactured population of predators into the environment, equivalent to Kettlewell's introduction of insectivore birds into an aviary full of moths, more promising results were observed. The population of agents was shown to adapt to the environment and over a period of 10000 iterations the frequency of camouflaged colour alleles were shown to increase to levels approximately twice that of the non-camouflaged colour alleles.

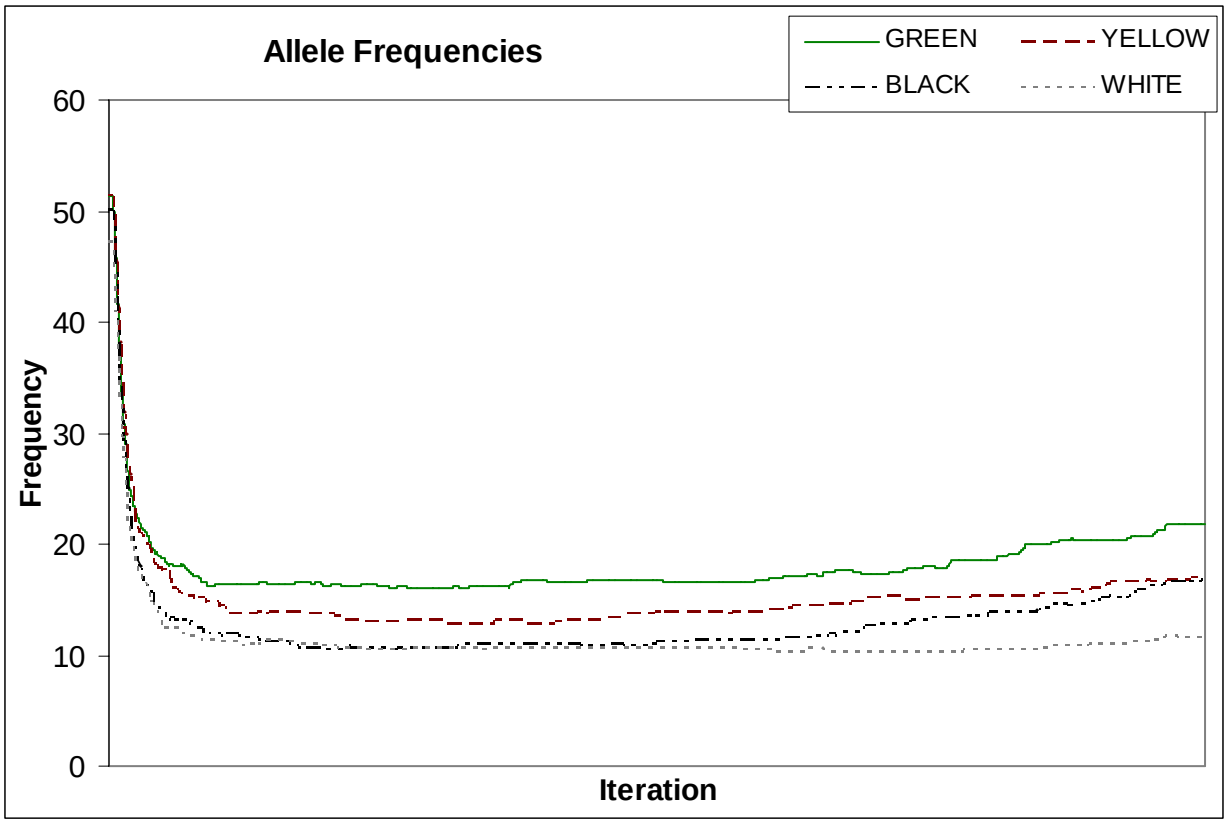

Figure 5.2.4: Allele frequencies of each colour over 10,000 iterations
Environment surface = grass

This result proved statistically significant. Over 20 sample sets of data were generated, and all of these sets showed similar quality results. Null hypothesis H_{0a} can be rejected based upon this evidence. It is the case that enabling an environmental camouflage mechanism leads to adaptive behaviour in a population of agents based upon the selection bias towards those who are less easily distinguished from their habitat.

5.3. Sustaining Multiple Species

A first hurdle in developing a simulation of speciation is ensuring that the simulation environment can sustain more than one 'species' in isolation without the population becoming extinct. In the Java implementation two basic sub-types of population (not species, as they could still interbreed) were able to co-exist and sustain each other. However, defining a single individual as one or the other was very difficult as interbreeding led to many intermediate individuals with characteristics from a parent of each type. The sub-populations could loosely be tagged predator and prey, simply because the strength and 'sharp' teeth of one type naturally led to greater success in attacks, and the blunt teeth of the other facilitated success in foraging.

The implementation of a stronger sexual selectivity mechanism in the .NET simulation prohibited interbreeding, allowing the sub-types to be classed as species, but in isolation these species were not capable of survival beyond approximately 10 generations.

In order to ensure that a species was able to self sustain, several possible measures could be taken:

- Ensuring all species were able to draw suitable sustenance from their habitat. This means carnivores must be able to eat shrubs, and herbivores must have plentiful resources to eat
- Introduce additional energy into the simulation through reproduction instead of relying upon parental degradation to 'fuel' offspring. This actually already happens
- Assume that a carnivorous species could never survive without a sustaining population of prey species. This is not exactly a measure to solve the problem, but it is the most accurate model of nature

Without resorting fully to the suggested assumption in the third point above, the simulation could be made more resilient by using offspring to introduce additional energy into the simulation (point 2 above). During an attack, the genes of an agent are degraded. Each time an allele value is reduced to support an agent during a fight, their energy is increased by a factor specified by the global parameter 'Allele Energy'. Tinkering with this value over a series of test experiments led to better results in these tests. Due to time constraints, there is no documented data for these tests. Please see the evaluation, section 6, for a critical examination of time management in this project.

5.4. Simulating Speciation

The process of stabilising the new implementation of the simulation led to severe time constraints during the experimental phase of this research. As shown in section 5.1 and 5.2, the foundations for speciation have been laid, with an application capable of simulating geographical barriers for isolation of sub-populations, and a proven example of adaptation with the learning architecture in place. The following is a summary of the experimental procedure for simulating allopatric speciation, based upon the example illustrated in figure 2.1.2.

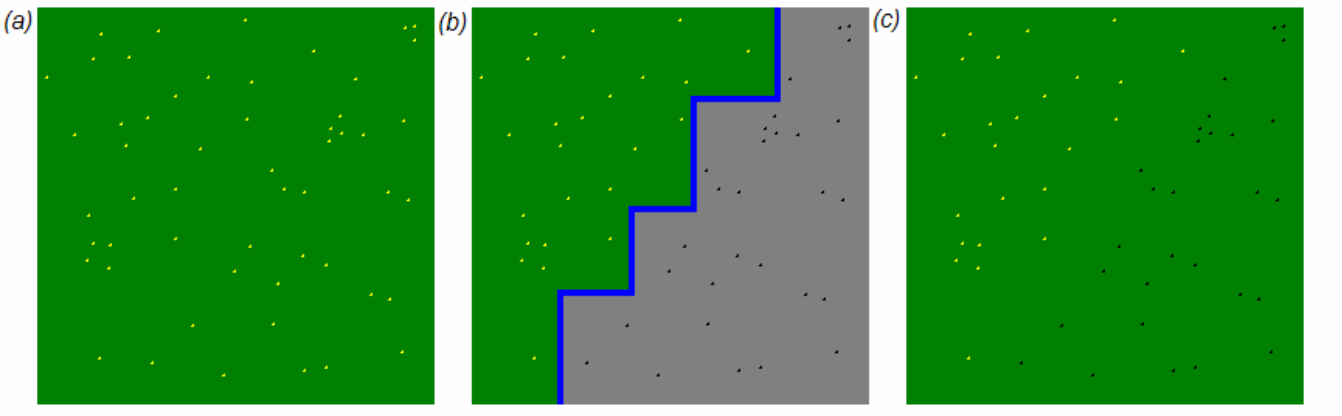

Figure 5.4.1: Speciation experiment

As shown in figure 5.4.1(a) above, a single species population will be evolved or placed manually in a well suited environment. In this case green, foraging agents are placed in a green environment with plenty of vegetation. Figure 5.4.1(b) illustrates the formation of a geographic barrier in the form of a river. This will isolate two sub-populations. The right hand sub-group are out of luck, and find themselves in a sparse, grey environment where they are not camouflaged at all, and will be forced to adapt to this environment by attacking each other and evolving to become black or white for camouflage. These two sub-groups will now be different enough to be unable to interbreed, and removing the geographic barrier, figure 5.4.1(c), will demonstrate this.

Cluster analysis of the population before and after the experiment should show a single cluster at stage one, eventually splitting into two distinct species.

6. Evaluation

6.1. Evaluation of Experimental Results

The limited selection of results gathered during this research seemed to be leading towards a successful speciation experiment. The process was dramatically hindered however, by a good deal of experimentation, debugging and tweaking required to stabilise the simulation.

In the process of designing the new simulator, several modifications were made which upset the balance between input energy and energy usage by agents. Throughout the design phase these modifications were justified, and the system as it stands is now capable of simulating adaptive processes acute enough to demonstrate speciation.

It is indeed the case that time management during the project could have been better, particularly after it became clear that the simulation was not as stable as was desired. By focusing more on cutting back the modifications such that a compromise solution between new functionality and stability could be reached, the result set could have been increased.

There remain several key issues with the implementation which need to be addressed in any future work. These issues are summarised in the next section.

As for the results which were gathered, they were very strong. The level of adaptation demonstrated explicitly in the experiment presented in section 5.2 shows that the agent architecture and simulation is capable of simulating environmental adaptation. The statistical significance of the results found was high, which gives hope for future experiments, as these can rely upon the adaptation mechanism to respond to changes in environment.

6.2. Known Issues

The biggest issue facing this research has been the sensitivity of the simulation to the array of global parameters available. The time spent finding a combination of parameters which facilitated a stable population of agents could have been spent performing useful experiments. However, this issue was compounded by the inefficiency of the simulation as a whole. Despite taking measures to improve efficiency of the subsumption architecture, the process of consulting six behaviours per agent each iteration is time consuming, and making good use of concurrency alone does not seem to have made a significant difference to run-time efficiency.

6.3. Extensions

Of course, the priority in any future work must be to rectify the issues above. By factoring out some of the global parameters without loosing the flexibility each of these provides, a future simulation might be slightly more reliable without the need for tweaking. In this vein is the possibility of using an evolutionary algorithm to determine the correct balance of global parameters by running a series of baseline simulations to determine the fitness of a group of parameters. Of course, the time this process would take would be considerable, but the manual process conducted in this project does not instil confidence in the alternative!

A feature of the Java implementation which was not implemented this time round was the XML settings file. Although manually manipulating the simulation settings was not a particular hindrance, the ability to schedule a series of tests overnight was missed.

Another area for investigation might be the statistics engine. A separate group of statistics reporting classes were implemented in the course of this project, and extending this concept would definitely add value to the system. In particular, graphical output direct from the application could save hours of statistical manipulation. Abstracting the implementation specific terminology from the system might even lead to a

generic multi-agent systems statistical reporting library, implementing an array of statistical measures, perhaps integrating with the XML test scheduling tool to provide a complete test-bed for systems such as this.

7. Bibliography of References

[ABB02] Almut Kelber, Anna Balkenius and Christian Balkenius. *Simulations of Learning and Behaviour in the Hawkmoth Deilephila elpenor.* In From Animals to Animats 7: Seventh International Conference on Simulation of Adaptive Behaviour. The MIT Press, 2002.

[Amos98] William Amos & John Harwood. *Factors affecting levels of genetic diversity in natural populations.* Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences. 353(1366): 177-186. 1998.

[And98] David Andre. *Learning Hierarchical Behaviours.* In NIPS '98 Workshop on Abstraction and Hierarchy in Reinforcement Learning, 1998.

[Bak05] Jason M. Baker. *Adaptive Speciation: the Role of Natural Selection in Mechanisms of Geographic Speciation.* Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences, Volume 36, Issue 2, June 2005, pages 303-326. 2005.

[Bar03] Andrew Barto & Sridhar Mahadevan. *Recent advances in hierarchical reinforcement learning.* In Discrete Event Dynamic Systems: Theory and Applications, 13, 41-77, 2003.

[BBC07] BBC News. *Mouse Brain Simulated on Computer.* http://news.bbc.co.uk/2/hi/technology/6600965.stm. Last Accessed 28th April 2007.

[Berk07] University of California, Berkley. *Understanding Evolution.* http://www.evolution.berkley.edu/evolibrary/home.php. Last Accessed 26th March 2007.

(a): Mechanisms of Change. http://www.evolution.berkley.edu/evolibrary/article/0_0_0/evo_16.

(b): Mechanisms: The Process of Evolution. http://www.evolution.berkley.edu/evolibrary/article/0_0_0/evo_14.

[Bro85] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. Technical Report, Cambridge, MA, USA. 1985.

[Cla06] John Clark. *Evolutionary Computation and Optimisation*, Lecture Slides. http://www-course.cs.york.ac.uk/evo/. Last Accessed 3rd March 2007.

[Dar98] Charles Robert Darwin. *The Origin of Species.* Gramercy Books. 1998.

[Fai04] Sam Fairhurst. *Complex behaviour learning for subsumption architecture based agents.* Third year project. University of York. 2004.

[Fir97] Richard Firenze. *Lamrck vs. Darwin: Duelling Theories.* Reports of the National Center for Science Education, 1997.

[Flan95] Flanagan C, Toal D, Strunz B. Subsumption Control of a Mobile Robot. Polymodel 16, Sunderland, 1995.

[Gra99] Bruce S. Grant. *Fine Tuning the Peppered Moth Paradigm.* Evolution, Vol. 53, No. 3, pp. 980-984. June 1999.

[How60] Ronald A. Howard. *Dynamic Programming and Markov Processes*, The M.I.T. Press, 1960.

[Kett55] Barnard D. Kettlewell. *Selection experiments on industrial melanism in the Lepidoptera.* Hered. 9:323-342. 1955.

[Lan05] Chris Langton. *What is Artificial Life?* Available from: http://www.biota.org/papers/cglalife.html. Last Accessed 30th April 2007.

[Maes90] Pattie Maes and Rodney A. Brooks. *Learning to Coordinate Behaviours*. In Eigth National Conference on Artificial Intelligence, Boston, Massachusetts, 1990. Morgan Kaufman.

[Mat97] Mataric, M. J. *Learning Social Behaviors*. Robotics & Autonomous Systems, 20, pp. 191-204, 1997.

[Mayr62] Ernst Mayr. *Accident or design: The paradox of evolution*. p.1-14 in The Evolution of Living Organisms (G W Leeper, Ed) Melbourne University Press. 1962.

[MF94] Melanie Mitchell & Stephanie Forrest. *Genetic Algorithms and Artificial Life*. In Artificial Life, 1 (3), 267-289. Available from: http://web.cecs.pdx.edu/~mm/GA.Alife.pdf. Last Accessed 6th January 2007.

[Mit97] Tom M. Mitchell. *Machine Learning*. Computer Science Series. McGraw-Hill, 1997.

[MM02] Melanie Mitchell & Mark Newman. *Complex Systems Theory and Evolution*. In Encyclopaedia of Evolution, (M. Pagel, editor), New York: Oxford University Press, 2002.

[PS02] Simon Pick & Roland Strauss. *Adaptive Leg Placement Strategies in the Fruit Fly Set an Example for Six-legged Walking Systems*. In From Animals to Animats 7: Seventh International Conference on Simulation of Adaptive Behaviour. The MIT Press, 2002.

[PS07] P# Home. http://www.dcs.ed.ac.uk/home/stg/Psharp/. Last Accessed 6th January 2007.

[Ren01] Paul Rendell. *Turing Universality of the Game of Life*. Available from: http://www.cs.ualberta.ca/~bulitko/F02/papers/rendell.d3.pdf. Last Accessed 2nd April 2007.

[Ros86] Rosenschein S.J. & Kaelbling L. *The Synthesis of Digital Machines with Provable Epistemic Properties*. Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge, Monterey, CA.

[Rus02] Stuart Russell & Peter Norvig, *Artificial Intelligence – A Modern Approach*. Prentice Hall, 2002.

[SEP02] Stanford Encyclopaedia of Philosophy – *Species*. http://plato.stanford.edu/entries/species/. Last Accessed 27th April 2007.

[SP95] R. Storn and K. Price. *Differential evolution -- a simple and efficient adaptive scheme for global optimization over continuous spaces*. Technical Report TR-95-912, ICSI, 1995.

[Stat04] Statsoft Inc. *Cluster Analysis*. http://www.statsoft.com/textbook/stcluan.html. Last Accessed 1st May 2007.

[Sutt98] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semimdps: A framework for temporal abstraction in reinforcement learning. Articial Intelligence, 112:181-211, 1999.

[TSC06] Ronald Thenius, Thomas Schmickl and Karl Crailsheim. *Economic Optimisation in Honeybees: Adaptive Behaviour of a Superorganism*. In Proceedings, From Animals to Animats 9, 9th International Conference on Simulation of Adaptive Behavior, SAB 2006, Rome, Italy, 2006.

[UWB05] University of Wyoming, *Principles of Biology Lecture Notes VII – Biological Evolution*. http://www.uwyo.edu/bio1000skh/lecture36.htm. Last Accessed 20th February 2007.

[Wal97] John F. Walker & James H. Oliver. *A Survey of Artificial Life and Evolutionary Robotics*. http://www.citeseer.ist.psu.edu/walker97survey.html. Last Accessed 2nd April 2007.

[Will06] Simon Williamson. *Macro Learning with a Subsumption Architecture*. Fourth Year Project, University of York, 2006.

Appendix: Global Simulation Parameters

```
public static class Settings
{
	// SIMULATION
	public const bool SHOW_ALL_ACTIONS = false;
	public const int D_REDRAW_FREQUENCY = 10;

	// SETTINGS

	// Do agents utilise the reinforcement mechanism to learn?
	public const bool D_LEARNING_CHILDREN = true;

	// Is disjunction learning enabled by default?
	public const bool D_DISJUNCTION_LEARNING = true;

	// * Frequency of new condition adoption and architecture reordering
	public const int D_SA_LEARNING_FREQ = 80;

	// * Defines the level of correlation required for adoption decisions
	public const double D_PRECOND_ACCEPTANCE_MARGIN = 0.6;

	// * encourages exploration of full behavioural scope
	public const int D_RANDOM_BEHAVIOUR_RATE = 5;

	// * encourages mating
	public const int D_MATING_BEHAVIOUR_RATE = 25;

	// * each gene costs X^alleleValue
	public const double D_LIVINGCOST_BASE = 2.72;

	// * number of resources placed randomly each iteration
	public const int D_REGENERATION_RATE = 0;

	// * number of resources placed in seed locations each iteration
	public const int D_SEED_REGENERATION_RATE = 5;

	// * are agents able to adapt to become camouflaged
	public const bool D_CAMOUFLAGE = true;

	// * Is macro learning enabled by default?
	public const bool D_MACRO_LEARNING = false;
	public const int D_MACRO_EVAL_FREQ = 2000;

	// DEFAULT PROPERTY VALUES

	// * Mutation rate for new chromosome generation (0..100)
	public const int D_MUTATION_RATE = 5;

	// * Default energy a resource is created with
	public const int D_RESOURCE_ENERGY = 700;

	// * Energy contribution of each allele
	public const int D_ALLELE_ENERGY = 150;

	// * Energy of agents created at initialisation
	public const int D_INITIAL_ENERGY = 60000;

	// * Age of consent!
	public const int D_MATING_AGE = 50;

	// * Minimum time between mates
	public const int D_MATING_INTERVAL = 50;

	// * Multiplier for living cost deductions - for tweaking
	public const double D_LIVING_COST_COEFFICIENT = 1.0;

	// ACTION COST COEFFICIENTS
	public const int D_MOVEMENT_ACTION_COST = 100;
	public const int D_REPRODUCTION_COST = 5000;
}
```