PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science and Electronic Engineering
College of Environmental Sciences and Engineering

Agent-Based Co-evolutionary Algorithms for the Simulation of Dynamic Ecosystems

Gary Ferguson

Submitted in partial satisfaction of the requirements for the
Degree of Bachelor of Science
in Computer Science

Supervisor Dr. Franck Vidal

May 2020

Acknowledgements

> *The general who wins the battle makes many calculations in his temple before the battle is fought. The general who loses makes but few calculations beforehand.*
>
> — **Sun Tzu**

I'd like to acknowledge my cats for their continuing moral support.

Statement of Originality

The work presented in this thesis/dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

Gary Ferguson

Statement of Availability

I hereby acknowledge the availability of any part of this thesis/dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein. I further give permission for a copy of this work to be deposited with the Bangor University Institutional Digital Repository, and/or in any other repository authorised for use by Bangor University and where necessary have gained the required permissions for the use of third party material. I acknowledge that Bangor University may make the title and a summary of this thesis/dissertation freely available.

Student:

Gary Ferguson

Abstract

This project explores concepts such as artificial life (alife) and Evolutionary Algorithm (EA) for creating diverse and dynamic ecosystems, as well as briefly delving into some other topics. This implementation is aimed at video games that rely on large immersive environments, such as that of the role-playing game (RPG) genre or of recent popularity, walking simulators in which the focus is in most cases solely on environment. A lot of modern games lack compelling wildlife, often becoming predictable and repetitive very quickly. This approach hopes to solve these issues and provide an ever-changing ecosystem of interesting creatures exhibiting emergent behaviours.

In particular, this paper will be evaluating the application of co-evolutionary algorithm (CoEMAS) for the creating these dynamic ecosystems. This form of EA takes a decentralised approach to selection in breeding, allowing for more speciation and agents that may even become codependent on each other for survival. This will allow for a much wider array of emergent behaviour to take place. The aim of this research is to create a modular end product that would be usable by game developers as a framework for the creation of their own games ecosystems; in turn increasing the immersion of their final product.

Contents

1 Introduction **1**
1.1 Aim 1
1.2 Objectives 1
1.3 Hypothesis 2
1.4 Client 2

2 Background **4**
2.1 Evolutionary Algorithms 4
2.1.1 Genetic Algorithms 5
2.1.2 Cultural Algorithms 6
2.1.3 Co-evolutionary Algorithms 6
2.1.4 Evolutionary Multi-Agent Systems 7
2.1.5 Co-evolutionary Multi-agent Systems 7
2.2 Artificial Life 7
2.2.1 In Entertainment 8
A Brief History 8
The Creatures Series 9

3 Design **12**
3.1 The Environment 12
3.2 The User Interface 13
3.3 The Agents 13
3.3.1 The Genome 14
3.4 Evolution 15
3.4.1 Fitness Evaluation 16

4 Implementation **17**
4.1 The User Interface 18
4.2 The Environment 18
4.2.1 Map Tiles 20
4.2.2 Food Sources 21
4.3 The Agents 22

5 Discussion **26**
5.1 Results 26

5.2 Speciation 27

6 Conclusions **30**

6.1 Future Improvements 31

References **34**

List of Figures

2.1 this demonstrates the basic principles behind the two genetic operators used, crossover and mutation. Crossover being the choosing of a random point at which to combine the two genomes and mutation the chance of one of the values within the genome being changed to a random variable. 5

2.2 A screenshot from the game Simlife, giving an overview of a species current gene pool. 9

2.3 A screenshot from Creatures Deluxe edition. In it we can see three norns, one of which is attempting to communicate. 10

4.1 This diagram shows the core concepts of CoEMAS. With the map being split up into nodes an agent may choose to migrate to and from. As well as this, showing the internal processes of an agent, from how decisions are made, and the linkage to it's genetics and current resources. 23

6.1 A screenshot from the final application. In it you can see the three main species and the two plant food types. 30

List of Tables

4.1 keyboard controls 19
4.2 settings for noise generation 20
4.3 all the available tile types made for this project 21
4.4 food spawners 22
4.5 food types 22

5.1 This shows the results of one particular run in which only apple eating specimen survived initially. Although, initially there is almost no speciation, it does demonstrate the dominance of flocking behaviour and the general structure required for initial survival. 27
5.2 This shows the results of one particular run in which only apple eating specimen survived initially. Although, initially there is almost no speciation, it does demonstrate the dominance of flocking behaviour and the general structure required for initial survival. 28
5.3 A population showing much more speciation. 29

Chapter 1

Introduction

1.1 Aim

the main focus of this project is to create a simulated ecosystem filled with artificial life (alife) agents that across generations will be capable of evolving to adapt to their environment. This has been achieved through the use of co-evolutionary algorithm (CoEMAS), a form of Evolutionary Algorithm (EA) that takes a decentralised approach to selection in breeding and divides a problem across several sub-populations, wherein the sum of these parts equals the solution.

The agents within this ecosystem will be required to meet their needs, such as thirst and hunger. Each agent also has it's own genome, within which various information is stored on how that agent will function within the given environment, such as it's sight range and the thresholds at which an agent will seek a resource, if it exhibits flocking behaviour and the parameters for that, and more. A procedurally generated environment complete with plants that spawn food is provided for the agents to live in and explore. The end product will be executable within a Linux terminal, featuring a basic ASCII-based user interface (UI), made with the ncurses [7] programming library.

1.2 Objectives

- Create alife agents complete with a high-level implementation of a genome.

- Allow agents to show a range of behaviours, such as seeking resources, mating, and migrating.
- Agents should evolve over time through each generation to better adapt to their environment through the use of CoEMAS.
- The system should allow for varied speciation in the agents.
- Have procedurally generated 2D environments for agents to live in complete with plant-life and water sources.
- Display the environment within a Linux terminal using an ASCII-based UI.

1.3 Hypothesis

Using CoEMAS to create artificial life agents will allow for an ecosystem that displays speciation and a wide array of emergent behaviours. It will also give us an interesting look into the machinations involved in Darwin's theory of evolution as we see the rise and fall of species. The procedurally generated environment, along with the distribution of food spawning trees will encourage agents to migrate between areas and settle in ones with large enough resource density to support them.

1.4 Client

The main purpose of this project is to produce a modular system that could be implemented as a component within a video game. This system would allow players to be provided with a much more immersive game world, allowing them to observe a much more dynamic and rich wildlife. It would also be possible for the wildlife to change over time, reducing the monotony that a standard approach would have over long periods of gameplay. This would mainly come in use for games that rely heavily on immersive worlds, such as that of the role-playing game (RPG) genre, or the numerous walking simulators of recent popularity that focus solely on the experience the player has within the environment.

As well as it’s application within the video game development industry, this project could also be used in a research context or in parallel to this as an easy to understand learning tool. As the framework is loosely based off concepts displayed in real-world evolution, it may also be used to replicate that, allowing us to better understand how particular nuances of evolution might take place.

Chapter 2

Background

2.1 Evolutionary Algorithms

In nature biological evolution has many mechanisms to promote the continuous development or evolution of a species, such as reproduction, mutation, recombination, and selection, EAs take inspiration from these mechanisms to optimise solutions in a given problem space. To do this they implement what is known as the fitness function, which is how each specimen is evaluated against the given problem, or simply how optimally it solves that problem. Specimen with a higher fitness score are more likely to reproduce and therefore have their genetics carried onto the next generation [22].

EAs perform well with an array of problems as they do not make any assumptions as to what the most optimal solution might be, without human bias being involved EAs are capable of coming up with solutions to a problem that might be completely unexpected. They have been used in a wide array of fields, such as the design of new materials that are tougher and lighter, creating chassis designs for cars with improved aerodynamics, optimising software, and even in video games for generating bots that can adapt to a human players actions and adjusts it's own tactics to counteract them.

Classical EAs have many downfalls, most importantly for this project the specimen lack the ability to gather or utilise information on their environment so that they can achieve their goals. As well as this, they lack biological and social behaviours, such as rivalry and cooperation [6]. If a solution is found that would allow a EA to do this then we could get very interesting results that would show way more emergent behaviour. One possible area to work on around this is based on the implementation of an agent-based approach that would have senses as well as basic communication skills, such as basic speech and leaving scent markers. This could allow us to see the marking of territory, the tracking of prey, and the prey avoiding a predators scent.

2.1.1 Genetic Algorithms

One of the most commonly used form of EAs are known as genetic algorithms (GAs): a form of search algorithm developed by John Holland in 1960, along with his students and colleagues at the University of Michigan. Taking inspiration from Darwin's theory of evolution it implements mechanics for natural selection and genetics. Their goal was to look at the adaptive processes found in natural systems and use this to design artificial systems software that retains the important mechanisms found in natural systems. It relies on the genetic operator of selection, mutation, and reproduction, although others may be used [23].

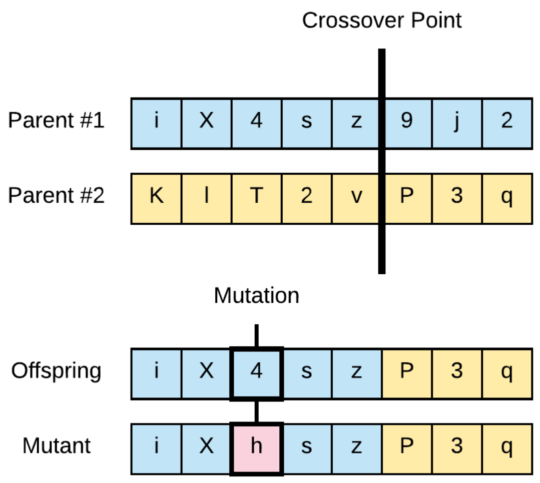


Figure 2.1: this demonstrates the basic principles behind the two genetic operators used, crossover and mutation. Crossover being the choosing of a random point at which to combine the two genomes and mutation the chance of one of the values within the genome being changed to a random variable.

GAs can be simplified to the process of copying strings and partially swapping them with each other, which is referred to as selection and crossover. Selection can be implemented in many ways, one common method to do this can be thought of as a biased roulette wheel, wherein specimen with a higher fitness score are more likely to be landed on, or one might simply take a sample from a percentage of specimen that rank among the fittest and breed new specimens with them. The mutation operator can then be used in the creation process; by adding a hint of randomness it allows a more thorough and optimal exploration of the problem space. Keep in mind that there are many approaches and variations of implementing any of these operators; a crossover operator for example might makes use of multiple splitting points in the strings, or changing the points at which these are split.

2.1.2 Cultural Algorithms

Created by Robert G. Reynolds in 1994 [19], cultural algorithms (CAs) incorporate the idea of using domain knowledge gathered throughout the evolutionary process to optimise the search process. This allows for the simulation of social and cultural changes within species. CAs evolution model is derived from proposed theories in sociology and archaeology. These theories indicate that cultural evolution's inheritance process can be seen at both a micro-evolutionary and a macro-evolutionary level [3].

Individuals at a micro-evolutionary level can be described in terms of behavioural traits, of which can be deemed socially acceptable or unacceptable. This represents the knowledge that is gained by an individual through each generation. On the macro-evolutionary level, individuals can generate what is known as a mappa [17], or a generalised description of their experiences. These mappa can then be combined to form a group mappa.

2.1.3 Co-evolutionary Algorithms

These are a form of EAs that implement the process of co-evolution, which is where a change in one species will result in a response from the others. In a co-evolutionary algorithm (CEA) each species may be trying to solve a separate problem from each other or taking a different approach to optimising a problem [11]. The benefit of this approach is best seen in problems that might not have a tangible way of formulating an explicit fitness function or require more population diversity.

2.1.4 Evolutionary Multi-Agent Systems

Evolutionary multi-agent systems (EMASs) are the result of combining two paradigms: multi-agent systems (MASs) and EAs. This approach provides mechanisms for decentralising the problem solving process across numerous intelligent evolving agents. These agents are not only able to communicate like in a MAS, they are also able to reproduce with each other, die, compete for resources. They able to manage all these behaviours and make decisions autonomously. Due to these features the selection mechanisms used in classical EAs cannot be applied, instead the evolutionary process becomes decentralised, reliant on each agents ability to thrive [5].

In a EMAS one of most important factors in determining an individuals fitness is the amount of a non-renewable resource called life energy possessed [20]. Those individuals with high levels of energy are much more likely to reproduce than those with low levels, as well as this those individuals with low energy levels are much more likely to die.

2.1.5 Co-evolutionary Multi-agent Systems

This system allows the problem to be divided across several sub-populations, with the population as a whole providing the solution. Agents within this system make use of three profiles; resource, reproduction, and migration. In the resource profile the agent can seek resources, get them, or die. The reproduction profile allows them to seek out a partner, clone themselves, perform recombination of the two individuals genetics, as well as mutations. The agent is also then able to give some of it’s resources to the new offspring.

2.2 Artificial Life

Alife is a wide field that covers a wide array of subject matters, from artificial ecosystems, evolution, to morphogenesis, and much more. In the words of Chris Langton, founder of the field, the literal and yet purposefully vague definition of alife is "life made by humans rather than by nature" [13]. To elaborate on this further, alife is the interdisciplinary study of living systems, taking the essential general properties found within them and adapting them for usage in man-made systems, as to synthesise alife-forms [1].

There are three main forms of alife, soft, hard, and wet. Soft alife is purely software-based simulations. Hard alife is implemented in hardware. Such as, automatically guided machines, capable of doing tasks unsupervised. Wet alife is biochemical-based. It involves the creation of living systems from synthetic DNA.

2.2.1 In Entertainment

A Brief History

There are many examples of alife being used within the entertainment industry. The early foundations of this being laid out by Reynolds [18] in 1987. Reynolds work went onto be used to model the behaviour of bats in the movie Batman II in 1992, one of the most notable early appearances in alife being used for animation in a mainstream movie. Reynolds work was further built upon in 1994 by Terzopoulos and his colleagues [21] who modelled in great detail the behaviour of fish. This included behaviours such as mating, feeding, learning, and predation. This work has went onto to be used in many short animated movies and video games.

Before Reynolds seminal work, there was a game called Little Computer People [4], written by David Crane for the Commodore64 in 1985. It focused around a man and his dog, requiring a player to engage with him to satiate his needs or he would become unhappy or sick. The man would have a unique set of behavioural traits dictating his actions that would be generated from a serial number unique to each copy of the game. The game did not feature any learning but was diverse and novel enough at the time to garner enough attention to earn it’s place as a cult classic.

In 1993 we saw the release of Simlife [14], released by Maxis. This title was promoted as being inspired from alife research, featuring a simulated ecosystem, with variable terrain, climate, and a plethora of both plant and animal life. The game was packaged along with a notebook and it was suggested that the player take notes on their experience. The landscape featured a variety of geographic features and resources, which were distributed using cellular automata techniques. It featured intrinsic adaption, as it made no use of an explicit fitness function, creatures known as orgots would breed based on their own internal states, seeking a mate who is also ready to mate and within proximity.

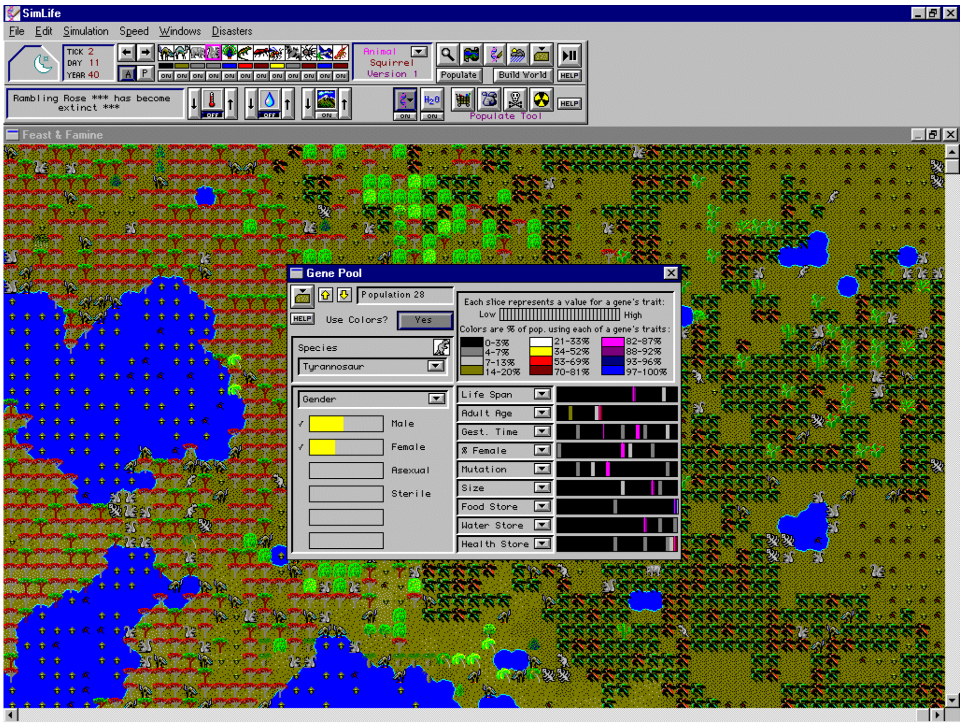


Figure 2.2: A screenshot from the game Simlife, giving an overview of a species current gene pool.

The Creatures Series

One of the most notable and prominent early commercial successes being the Creatures series [12], created by Steve Grand. The first Creatures game was released in 1996, with five more major releases from then leading up until 2001. The Creatures series pushed it's artificial life basis heavily in marketing, as not only an entertaining game that allowed you to play with cute creatures known as norns, but also as an opportunity to take part in alife research. For it's time period Creatures was well respected by many for the massive leaps in alife it introduced, and in such a way that made the field accessible to a wider audience.

The core gameplay relied on having players foster an emotional bond with their creatures as they raise and care for them as they progress through stages of maturity, from child to adult and eventually death. The artificial intelligence (AI) being somewhat flawed and slow to learn was in fact what gave these norns so much of their believability and charm as they explored their environment in a clumsy and goofy manner, eventually maturing into quite intelligent creatures.

Creatures made use of artificial neural networks, artificial biochemistry, and artificial evolution, allowing detailed simulations of both biological and neurological functions [2]. Norns could interact with their environment as well as each other with their primitive linguistic capabilities. The player is even able to talk to them through written word, allowing a user to teach their norns basic nouns and verbs through association. Their neural networks were capable of generalisation, taking information they learn from one scenario and applying it to a similar one, this was aided by the simplification of their environment, where each object belonged to a category; a necessity of the hardware limitations of the time period [9].

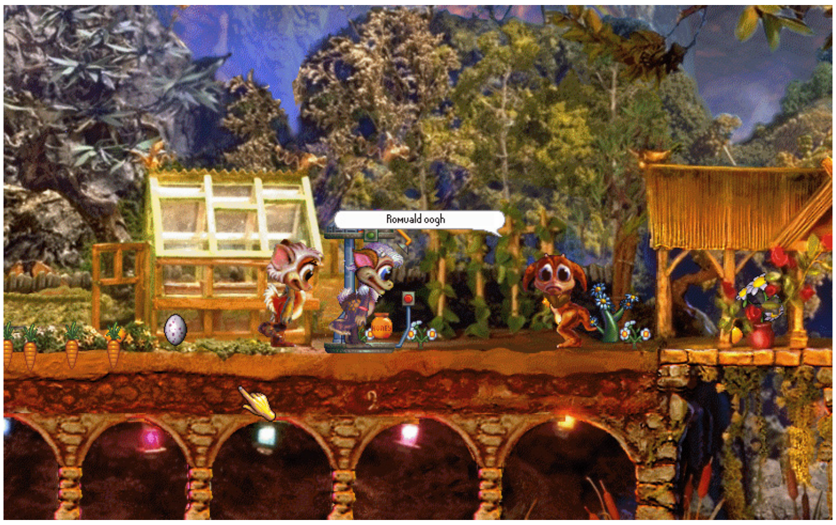


Figure 2.3: A screenshot from Creatures Deluxe edition. In it we can see three norns, one of which is attempting to communicate.

The norns could sexually reproduce, using crossover and mutation to create offspring. It was possible for offspring to even have more or less genes than the parents, meaning that the complexity of the norn’s biochemistry and neural network could increase. Along with this, as norns could learn from one another this allowed for cultural transmission of knowledge. Meaning norns were capable of both biological and cultural evolution across multiple generations.

Chapter 3

Design

3.1 The Environment

The first components design that must be considered is that of the environment's. It must somewhat resemble a realistic environment, but not necessarily be fully realised in 3D as this will be costly performance wise and add much unneeded complexity to the system at this phase of the framework's development. Thus, a 2D vector of tile objects was chosen. These tile objects can contain game objects, such as spawners and food, as well as this they have an elevation variable which is used to generate the tile type. There are a variety of tiles to represent water, grassy areas, mountains, and so on. These different terrain types, as well as being aesthetically pleasing and easy for a user to visually interpret, also dictate where water sources are and where food spawners can be potentially placed, and thus how food is distributed across the map.

Procedurally generated environment's were selected over that of manually designed environment's, as this allows a much wider variety of environment's to be created for the agents to explore. There are many options in regards to procedural generation, each with their own benefits and downfalls, a simplex noise approach was chosen for it's simplicity and performance. Manually designed environment's although much more time consuming to build could allow much more controlled and specific scenarios for agents to adapt to. This also has the added benefit that the environment can be recreated using the seed value along with the other inputs, meaning it is not necessary to store each individual tile, only what game objects are contained on it.

3.2 The User Interface

For this project the choice was made to use a very basic ASCII-based user interface; this decision was made as the focus is not on the graphics but the mechanics and therefore using a large complex engine would only be a distraction from the focus of the project. By simplifying the graphics to only what is necessary we can get the most out of the project performance wise, as well as allowing for modularity as it would be very simple to change the user interface to one of the end users choice. In order to create this engine ncurses [7] has been used; a programming library written in C under the X11 license for creating text-based user interfaces that can be ran within a terminal emulator for Linux.

The other path that might have been taken with this project is developing in Python, which offers curses, the python implementation on ncurses. The reason C++ was chosen over Python was mainly due to the nature of the system, as the scale and complexity of the system increases Python would struggle to keep up with modern C++. Both offer a variety of features that would prove useful for this project, but C++ has the advantage when it comes to control over how memory is managed.

Another option may have been to build this within an existing game engine, such as Unity or Unreal. I think there are merits to this approach as it would allow much more realistic rendering of environments, but it would also restrict this framework to that engine without more work. A ncurses implementation can be created to be fully modular so that all of the classes can be taken and used elsewhere, such as integrating them at a later point in Unity, which allows C++ to be integrated as a plugin or library.

3.3 The Agents

Agents within this framework should be able to at least exhibit basic behaviours, such as seeking food, water, and a mate, as well as migration, including wandering and flocking. These behaviours are important as these model the foundational physiological layer in Maslow's hierarchy of needs [15], excluding mating which belongs to the third tier.

Mating behaviour belongs to the social layer, which is the third layer. The second layer is safety, this is just not safety from predators, but also the represents stability in their life. To quantise how these basic needs are met, a variable we shall dub the comfort value is used. The comfort value will increase once the physiological needs have been met and will decrease when they are not, once this meets a threshold value dictated within their genetics they will then seek a mate. This value is basically a representation of an agents well-being and mood. This also serves to gate the mating behaviour for specimen that have successfully managed their needs over a period of time, which should filter out less competent agents from breeding.The comfort value is similar to the fitness value, but we will use a separate function for that, this simply controls the rate of reproduction. To better balance this out a flat cost should be applied to producing an offspring.

3.3.1 The Genome

As for an agents genome, there are two main approaches to how these are coded; high-level and low-level. High-level, wherein it is very clear what each genes purpose is and they may be represented as a variety of variable types. Whereas low-level represents the genome as a bit string, which is closer in line with how real world genetics work. The benefits of high-level are that it is easier to maintain and edit, each gene is modular and can easily be built on by adding more variables.

Low-level, although more realistic, isn't as flexible and easy to add onto as a high-level approach. As this system is intended to be used by an end user and edited to meet their needs, a high level approach should be used. A high-level approach would also in this case require less processing and memory requirements and as this is intended for implementation within a video game the ratio between efficiency and depth is important.

The genome contains a variety of variables, dictating when needs should be met, if they flock and ranges for flee and pursuing, their dietary requirements, as well as lifespan, and sight range. The more the genome can influence, the more complex the ecosystem can be. Each of these genes has a range that is allowed, along with the maximum value amount a gene can change due to the creep operator.

3.4 Evolution

This is the main focus of this project and therefore the most important component. There are many approaches to be taken with this, from classical EAs to EMAS, there is also merit to CA. The benefits of classical EAs are that they are very straightforward to implement and for scenarios where a fitness function is easy to quantise it is very fast and efficient at finding the optimal solution. Although, the problems with this approach are numerous. The centralisation of the fitness evaluation and evolution process restricts the speciation available and wouldn’t allow for codependent species to evolve, thus limiting the possibilities of food chains and the variety of emergent behaviours that could be displayed.

The next possible solution is EMAS; a combination of classical EA techniques and those of a MAS. This approach decentralises the problem solving process across multiple agents and does not necessitate the need for an explicit fitness function. Each of the agents is capable of autonomous behaviour, such as reproducing, dying, and gathering resources. This is a lot closer to what is needed for this project but still falls short as there is still a tendency towards a lack of speciation.

The answer to these issues is known as CoEMAS. It takes the concepts used in EMAS and combines them with those of CEAs. This means not only is the problem solving process decentralised, but the solution to that problem is spread across several sub-populations. The solution in this case being a diverse yet stable ecosystem.

3.4.1 Fitness Evaluation

The fitness in this case is how well an agent has adapted to live in it's environment. If an agent survives and meets it's base physiological needs it may be allowed to mate. When a creature is ready to mate though is will go through a process of evaluating potential mates in it's vicinity. To allow for a greater degree of speciation a similarity bias has been used, agents will select other agents that are similar to it, but if the similarity exceeds 80%, they will be penalised in accordance to the degree of which it exceeds this threshold.

As well as similarity, the proximity of potential mates is also accounted for. The distance taking the available sight range into account is adjusted to a float value between zero and one, zero being a point at the maximum of their sight range and one being directly adjacent. This weighting of this half that of the evaluated similarity. This decision was made as agents would end up travelling large distances to find a mate that may only be a fraction of a percent better than one standing right next to it, which not only makes very little sense in the context of a natural system, but is also inefficient. As well as this, it allows separate flocks to remain separated from one another, while not ruling out breeding between them it will softly discourage it, this also encourages allopatric speciation to a degree.

Chapter 4

Implementation

Algorithm 1: Overview of the simulation

```
set up terminal window object;
initialise ncurses colours and colour pairs;
initialise terrain variables generate the environment;
fill creature vector with initial population;
add trees to the environment;
while the simulation is active do
    display the environment and the creatures;
    add the additional UI elements to the display allow each creature to
      take their turn of action;
    forall spawner objects in spawner vector do
        if spawners timer exceeds the spawn rate variable then
            spawn a food object on a viable tile within the spawn radius;
            reset the spawn timer;
        else
            increment the spawn timer;
        end
    end
    increment the clock;
    check for user input;
    refresh the display output;
end
destroy the window object and return to terminal;
```

4.1 The User Interface

With our implementation of ncurses [7] all the colours have been manually set, it is recommended to run your terminal emulator with the ’tango’ colour scheme as it was designed with this in mind. Once all colours have been initialised you can then set up the colour pairs to be used; each colour pair having a colour for the displayed character and it’s background.

The UI is kept to a minimal design as to not interfere with the central focal point, the map . There is a heads-up display (HUD) style output of basic statistics, such as the simulation time, births, and deaths and their causes. Also displayed is the age, needs, and current profile of the first two creatures in the vector, so they can be monitored to make sure everything is running as expected. The map is updated in real-time as the simulation advances, displaying the environment and the agents moving across it. The map is adjusted to the size of the terminal and if the terminal size is changed so will the map size as to make the best use of space.

There are some features for ease of use added to the UI accessible through key presses, you can see all of them in table 4.1. The most important of these is the ability to move the view port across the map, allowing the user to explore the environment, this feature is accessible with the arrow keys, or a VIM-like ’hjkl’ controls. You are able to pause the simulation at any time with the spacebar and still move through the environment. The HUD can be switched on and off at the users discretion. The world and creatures current states can be saved also, so that they can be loaded again at a later time. A number of randomly generated agents can be added at any time also, this is added to allow the user at their discretion to add more variety to the gene pool or to give a boost to the current population and prevent extinction.

4.2 The Environment

The environment’s terrain is procedurally generated using simplex noise [16]. The elevation values outputted by the simplex noise dictate which tile

Table 4.1: keyboard controls

Key Press	**Action**
Up key or ’k’	Move view up
Down key or ’h’	Move view down
Left key or ’j’	Move view left
Right key or ’l’	Move view right
f	Switch HUD off/on
a	Add randomly generated agents
s	Save world state
Spacebar	Pause
Escape and Enter	Quit

objects will be used at each grid coordinate. The lowest values will create water and the highest values will create mountains, in between this we have a range of plains, forestry, and sandy areas. In total there are seventeen varieties of tiles. As well as providing aesthetics to the display output, it also provides agents with obstacles to avoid, in the form of the deep water and the peaks of mountains.

For this implementation of simplex noise an array of techniques has been used to improve upon it to better create a more interesting and realistic environment. Firstly, the base height map is generated from the seed using the given frequency and scale. This process is repeated twice more at slightly differing frequencies and weights to create octaves, which will add more character to the height map. The height map must then be brought back to within the original range by dividing it by the total weight of all octaves, then add one to this value and divide by the number of additional octaves used.

The next step in terrain generation is called redistribution, wherein we shape elevation values to better fit our purpose. The first technique used is getting the noise value to the power of an exponent. Exponent values above one will accentuate the mountains, whereas lower values will smooth them out. Next, the noise is redistributed into terraces by first getting a rounded value from multiplying the noise by the quantity of terraces wanted. This value is then rounded and divided by the terrace value. Finally, the noise value is multiplied by 255, this value just sets the range of the noise making

Table 4.2: settings for noise generation

Variable	Type	Value
seed	double	randomly generated
scale	double	0.0039f
frequency	double	4.4f
terraces	unsigned int	64

it easier to work with. We now have the finalised height map ready for use. The values used for this project can be seen in 4.2.

Algorithm 2: The process of generating terrain from from noise.

```
Result: A 2D vector of tile objects
set the scale of the terrain;
get the seed value;
forall height map points in the 2D grid do
    generate noise at the given frequency and scale;
    generate two additional octaves of noise at differing frequencies
      and weights;
    add the octaves of noise to the original;
    divide the noise value by the total weight of all octaves;
    add 1 to the noise value then divide by the number of additional
      octaves used;
    set noise value to the current value to the power of an exponent;
    redistribute the noise into terraces;
    multiply the noise by 255;
    store this noise value as the points elevation;
    assign the point the relevant tile based on elevation;
end
```

4.2.1 Map Tiles

Each tile is it’s own object, having an integer variable attached to it that represents the colour pair used, as well as a vector to hold objects that are on that tile, an object limit, a Boolean variable in charge of whether this tile can be traversed by agents, and the character that is displayed on the tile. Within the world class all the standard tile objects that will be used are setup to be

Table 4.3: all the available tile types made for this project

Max Elevation	Tile Type	Is Passable?	Is Water Source?	Character	Character Colour	Background Colour
90	Deep Water	No	Yes	~	Light Blue	Dark Blue
110	Water	No	Yes	~	Light Blue	Blue
120	Shallows	Yes	Yes	~	Pale Blue	Light Blue
130	Shallows 2	Yes	Yes	N/A	N/A	Pale Blue
135	Sand	Yes	No	N/A	N/A	Yellow
138	Desert Sand	Yes	No	N/A	N/A	Pale Yellow
155	Plains	Yes	No	N/A	N/A	Dark Green
160	Savanna	Yes	No	N/A	N/A	Green
170	Grass	Yes	No	,	Yellow	Green
180	Long Grass	Yes	No	\	Dark Green	Green
190	Forestry	Yes	No	"	Dark Green	Green
200	Mountains	Yes	No	N/A	N/A	Dark Grey
210	Mountains 2	Yes	No	N/A	N/A	Grey
220	Mountains 3	Yes	No	N/A	N/A	Light Grey
240	Snow	Yes	No	N/A	N/A	White
255	Peaks	No	No	^	Light Grey	White

used as prefabs for building the environment. Using elevation data gathered from the height map we can choose which tile prefab should be used for each point on the 2D grid, you can see the tile types listed in 4.3, listed with the max elevation each tile can appear at, as well as other important information on their composition.

4.2.2 Food Sources

There are currently two available types of spawner objects, apple trees and banana plants. They each have a 2% chance of being spawned within their set elevation ranges. Apple trees spawning at a higher altitude within the forestry tiles, which are the most common tiles place and therefore apples will be the most readily available food. Banana plants spawn at a slightly lower elevation, and are therefore closer to water meaning agents who eat them will have to travel shorter distances overall. You can see this information in table 4.4. A timer variable attached to each spawner dictates when a food object will be spawned from it, being placed within a set minimum and maximum radius. The time is initialised to a random value on creation so they will not all spawn food at once. This will allow for agents to form a more natural distribution across the map and encourage migration once an area has been depleted of it's currently available resources.

As for the food objects themselves, they each have calories and a lifespan. The calories is essentially how much nutrition that food object provides to an agent. The lifespan dictates how long it takes for that food to perish, they are in this implementation all lower than the spawn rate of the food to

Table 4.4: food spawners

Variable	Banana plant	Apple tree
Character	T	/\
Minimum altitude	160	180
Maximum altitude	170	195
Spawn rate	200	150

Table 4.5: food types

Variable	Banana	Apple	Corpse
Character	)	*	c
Calories	1	1.2	2 + remaining food
Lifespan	80	100	200

prevent buildup in neglected areas, and making plant-based food a limited but continuous resource. You can see the differing variables of food types in table /reftab:FoodTypes.

The banana has a slower spawn rate but a higher calorie count, meaning although this resource is less bountiful they can sustain an agent for longer. Apples are the basic and most readily available food type. There is a third food type, which is the corpse, generated when an agent expires. It's calorie count is dictated by the flat rate of 2 with the addition of the food resources the agent was holding when it passed away. It is possible for no corpse to be generated if the creature died of starvation. The corpse provides the most nutrition on average out of all food types, but is the most rare as it requires the death of an agent to be created. Corpses are ate by carnivorous scavenging species, which will co-evolve aside the two types of herbivores.

4.3 The Agents

Each individual is composed of three main components, these are their physical laws, biological laws, and the genome. The physical laws are what restrictions are applied to an agents' movements and energy usage, as well as encompassing what interactions the agent can make with the environment and each other. The biological laws are what dictates how an agents' fitness is evaluated.

Finally within the genome is where all the information on the individual is stored, this has a direct affect on it's behaviour, from when it eats and drinks, to whether it will flee, pursue, or completely ignore other creatures. Information from the genome is passed on in the breeding process.

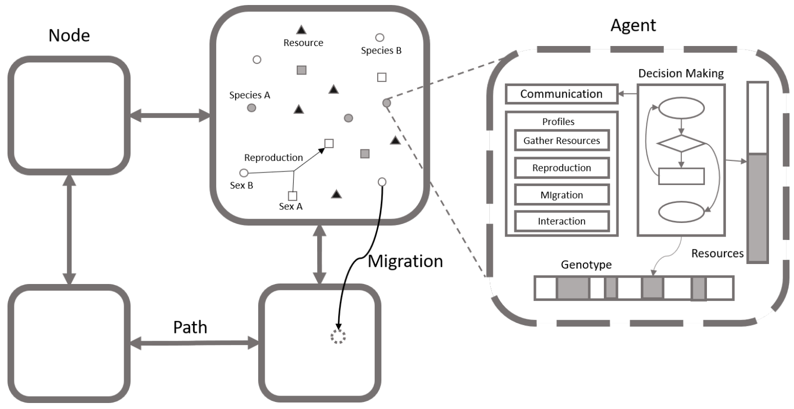


Figure 4.1: This diagram shows the core concepts of CoEMAS. With the map being split up into nodes an agent may choose to migrate to and from. As well as this, showing the internal processes of an agent, from how decisions are made, and the linkage to it's genetics and current resources.

Each agent has a set of variables attached to them, these for the most part can be separated into two categories; state variables and will variables. State variables represent the current status of the agent, this includes it's age, location, comfort level, and how much water and food it has reserved. Will variables represent the agents current wants and needs, in this system these are threshold values that must be met before an agent can enter a behaviour profile to seek food, water, or a mate.

To work out which behaviour the agent should currently be exhibiting you must compare the threshold values to the relevant state value, the value given represents the priority of that profile. The highest priority value calculated will determine the current profile. If all needs are currently met the agent may enter into a migration state, wherein agents will wander their environment; if other agents are within their sight range they may flock with them forming groups and moving in unison depending on the behaviours dictated in the genome.

Algorithm 3: The basis of an agents decision making

```
Result: The behaviour profile the agent is to be set to
initialise the priority values;
calculate the priority of each behaviour profile;
if there is a positive priority value then
    comfort level decreases according to genetics;
    set the profile to the one with the highest priority;
else
    comfort level increases according to genetics;
    enter the migration profile;
end
```

An agent can die if it fails to meet it’s needs, thinning the environment of those not well adapted. As well as the standard will variable there is also a comfort level which if not maintained can also be fatal. The comfort level is one of the most important state variables. This is because the comfort value dictates when the creature can and cannot breed. It is a measurement of how well the agent has been able to maintain it’s food and water levels over time. When not hungry or thirsty this value will increase and will decrease when they are not. This feature puts an emphasis on the survival of species that have longevity as it reduces the chance that agents will immediately start breeding once created, passing on weaker genes in the pool. The threshold for when breeding behaviour is decided on, like the other behaviours is genetically passed on so it is possible to see a variety of life spans and how they differ from each other.

An agent can move in one of eight directions within the 2D environment. For their path-finding capabilities an implementation of A* search algorithm [10] has been implemented with diagonal movements having a higher cost to better model realistic movement. For mating behaviour wherein an agent is seeking another suitable mate they will make use of the direction variable, which helps them predict the movements of another as to optimally move towards them, which can save a lot of movement at greater distances.

Chapter 5

Discussion

Theare a lot of ways to make the environment harsher for the agents to live in, encouraging agents to evolve to be more efficient. Food objects have a calorie variable that could be reduced, or the spawn rates of the food itself could be changed to make food even sparser. Changing the water levels is also a possibility, making traversal times between food and water higher. The metabolic rate of an agent can also be modified, which will change the rate at which food and water is used up.

5.1 Results

The results from this project proved to be very interesting. As more layers were added to the system the agents showed increasingly complex emergent behaviour. One of behaviours, known as a reproduction niche, is when a large group of creatures is concentrated around a small patch of food, showing movement similar to that of a spiral galaxy. Agents showing this behaviour will show high reproduction rates and a small variance in their distribution [8]. You could compare this behaviour to a swarm of flies surrounding a clump of food.

More basic behaviour such as flocking becomes common in later stages of the simulation, as this gene becomes more dominant the population will splinter off into flocks that will be constantly migrating to find fresh pastures. These individuals will typically have a longer lifespan than those showing reproduction niche behaviour. A more common variation of this was the flee behaviour not accompanied by pursue, particularly in the intermediary phase

Table 5.1: This shows the results of one particular run in which only apple eating specimen survived initially. Although, initially there is almost no speciation, it does demonstrate the dominance of flocking behaviour and the general structure required for initial survival.

Lifespan	Hunger	Thirst	Mate	Comfort Inc	Comf Dec	Sight Range	Diet	If Flocks	Flee	Pursue
8114	0.701071	1.4402	1.44827	0.0185749	0.014509	140	apple	1	14	9
8107	0.719932	1.4402	1.44827	0.0185749	0.0164766	95	apple	1	5	9
8107	0.701071	1.83447	1.44827	0.0185749	0.0164766	137	apple	1	5	9
8118	1.90074	1.4402	1.44827	0.0185749	0.014509	69	apple	1	5	9
8107	0.701071	1.4402	1.45567	0.0185749	0.0164766	95	apple	1	5	9
8107	0.701071	1.4402	1.44827	0.0185749	0.018553	137	apple	1	15	8
8107	0.701071	1.4402	1.45567	0.0185749	0.0164766	95	apple	1	5	9
8107	1.88566	1.4402	1.45567	0.0151067	0.0171495	83	apple	1	3	9
8107	0.701071	1.4402	1.44827	0.0185749	0.018553	137	apple	1	15	9
8114	1.90074	1.4402	1.44827	0.0185749	0.014509	140	apple	1	5	9

of the simulation. Fleeing behaviour may have had an advantage over those that flocked as this allowed them to separate from the herd and spread out, claiming their own territory, meaning they did not have to compete within a flock for resources.

The environment, although rudimentary proved that it has many tools on hand to challenge the agents. Lakes and the peaks of mountains aided in the segregating the population, a map seed that provided more obstacles would see a higher degree of speciation typically. The distribution of tile types that can sustain food spawners would dictate whether a migratory pack creature or the solo acting timid creature would be more sustainable.

5.2 Speciation

There are a variety of species that this framework is capable of generating, as well as codependent relationships between them.

Comfort gene allows for a wide variety of species, from those that breed rapidly with each other, to those that might only breed once in their lifetime, if at all. The former might produce offspring of lesser quality but succeeds in carrying on its genes with a brute force strategy, whereas the latter has a higher chance of going extinct, but can create much higher quality offspring in a smaller number of generations. It is not unusual to see quick breeding species initially be dominant but through mutations their breeding will slow down to focus on creating higher quality specimen.

Table 5.2: This shows the results of one particular run in which only apple eating specimen survived initially. Although, initially there is almost no speciation, it does demonstrate the dominance of flocking behaviour and the general structure required for initial survival.

Lifespan	Hunger	Thirst	Mate	Comfort Inc	Comf Dec	Sight Range	Diet	If Flocks	Flee	Pursue
8114	0.701071	1.4402	1.44827	0.0185749	0.014509	140	apple	1	14	9
8107	0.719932	1.4402	1.44827	0.0185749	0.0164766	95	apple	1	5	9
8107	0.701071	1.83447	1.44827	0.0185749	0.0164766	137	apple	1	5	9
8118	1.90074	1.4402	1.44827	0.0185749	0.014509	69	apple	1	5	9
8107	0.701071	1.4402	1.45567	0.0185749	0.0164766	95	apple	1	5	9
8107	0.701071	1.4402	1.44827	0.0185749	0.018553	137	apple	1	15	8
8107	0.701071	1.4402	1.45567	0.0185749	0.0164766	95	apple	1	5	9
8107	1.88566	1.4402	1.45567	0.0151067	0.0171495	83	apple	1	3	9
8107	0.701071	1.4402	1.44827	0.0185749	0.018553	137	apple	1	15	9
8114	1.90074	1.4402	1.44827	0.0185749	0.014509	140	apple	1	5	9

An example of codependent relationships between species is that of a short sighted species, that has more focus on it’s immediate vicinity thus relies more on flocking behaviour, to that of the longer sighted species. The long sighted specimens act like leaders to the short sighted ones leading them to fresh patches of food.

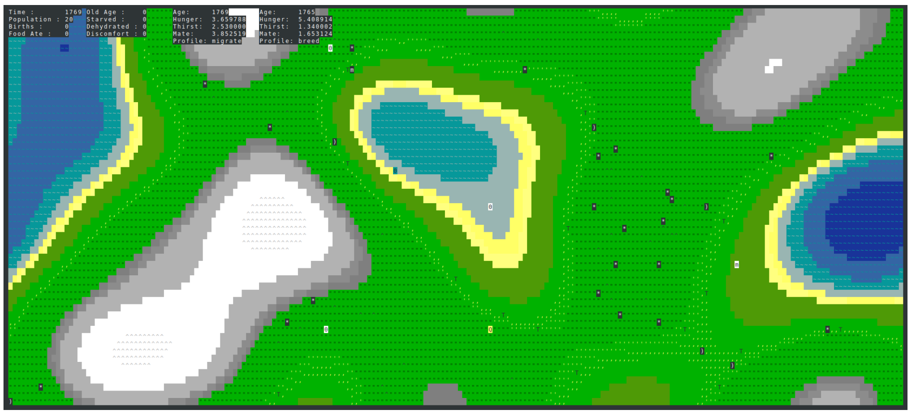
The carnivorous scavenging species is one that can only exist when there is a sufficient population of herbivorous creatures. They often will go extinct early in the simulation, then reemerge through the mutation of a herbivore back into the population and grow from there. As breeding is limited to creatures of the same diet this mutation must happen twice before there is a chance of this happening. Scavengers will more heavily rely on flocking behaviour as they need to stay with other creatures to get food, sight range may also become important depending on the scarcity of corpses.

Table 5.3: A population showing much more speciation.

Lifespan	Hunger	Thirst	Mate	Comfort Inc	Comf Dec	Sight Range	Diet	If Flocks	Flee	Pursue
4501	3.38666	2.7528	4.18413	0.0267988	0.0119032	155	carnivore	0	3	17
8576	2.76126	0.00344869	4.1689	0.0462415	0.0304405	208	banana	1	10	16
6335	0.528295	1.49664	2.60715	0.0292402	0.0178139	208	apple	0	2	2
4099	0.345178	2.24628	4.0341	0.0386703	0.019994	220	apple	0	2	8
7451	4.93931	2.75916	2.04205	0.0168426	0.0210264	76	carnivore	1	10	18
7264	0.809747	1.04002	4.82099	0.0238187	0.0486678	142	banana	0	20	4
3096	3.68288	2.19443	3.62866	0.0391021	0.0202034	91	apple	1	4	8
1595	0.42478	0.714846	2.84145	0.0114963	0.0267495	215	banana	1	2	19
4225	4.24902	2.05397	0.271823	0.030331	0.0133839	229	carnivore	0	20	4
1248	1.99106	1.40076	4.44536	0.0312937	0.0341542	163	banana	1	9	13
8654	3.26038	1.30984	3.99038	0.0163439	0.0179829	74	apple	1	7	5
4578	4.51231	0.811806	2.18901	0.0423874	0.0189222	180	apple	1	12	12
6164	0.726189	2.9101	2.2259	0.0279809	0.013809	73	banana	0	10	9
849	0.811315	0.168586	3.04104	0.0331175	0.0212304	159	banana	0	14	8
5215	0.474414	0.678679	4.06925	0.042861	0.0321463	168	apple	0	11	11
6902	0.189746	0.658838	0.869813	0.0454344	0.0110258	166	carnivore	1	15	3
2552	1.78824	0.608737	4.15642	0.0435392	0.0402479	127	banana	1	18	8
6556	0.564197	1.25822	1.6964	0.0249812	0.0144558	72	banana	1	15	18
1413	3.35163	2.38077	3.2189	0.0192479	0.0103233	128	banana	1	20	16
6731	3.95432	0.783125	2.03525	0.0311958	0.0253002	23	carnivore	1	2	20
7784	2.51151	1.33447	3.43642	0.0188749	0.0250881	151	apple	1	2	4
8919	3.89549	0.100538	0.464951	0.049885	0.0134823	244	apple	0	11	13
5258	1.89865	2.65241	3.70575	0.0383932	0.0312405	22	apple	0	19	5
8908	0.682649	1.82304	0.623622	0.024507	0.0290057	210	banana	0	12	19
9019	1.58469	0.820245	0.897169	0.0153324	0.0437834	221	banana	1	14	4
8258	3.15966	0.232189	3.28316	0.0122819	0.0280878	51	carnivore	0	14	4
6426	1.50994	1.00418	2.677	0.0337806	0.0446712	154	apple	0	18	13
4439	1.90274	1.18281	3.28224	0.0315826	0.0160184	28	apple	1	2	8
1132	1.09662	2.49634	2.93891	0.0320989	0.0340585	59	banana	1	13	7
7892	0.350897	2.68823	1.62338	0.0451802	0.0228783	243	apple	1	6	3
4232	3.63635	1.02008	1.02623	0.0327591	0.0132729	190	carnivore	0	11	6
7038	0.967826	2.03455	1.0893	0.030954	0.0230545	160	banana	1	9	10
8310	2.06105	1.05021	4.5944	0.0351634	0.0410221	180	apple	0	5	17
2595	4.42728	2.17074	0.689627	0.0268433	0.0145795	118	carnivore	1	19	19
9354	3.36057	2.19741	3.07734	0.0323371	0.019883	71	banana	1	12	16
1282	2.34832	0.795613	1.34679	0.0108489	0.0294831	32	carnivore	0	7	12
569	0.744609	1.9353	1.08734	0.0159557	0.0109146	249	carnivore	0	12	3
5229	2.49068	0.293011	4.14957	0.0322698	0.029724	260	carnivore	1	4	20
652	1.89811	2.31884	1.45056	0.0450245	0.025824	289	apple	0	15	9
8522	1.23872	1.02623	0.720702	0.0448152	0.0110258	166	carnivore	1	9	4
1400	0.0603676	0.557482	0.844848	0.049885	0.042776	244	apple	0	11	13
8521	1.23872	1.02623	0.720702	0.0448152	0.0110258	166	carnivore	0	9	4
8522	1.23872	1.02623	0.720702	0.0448152	0.0103031	166	apple	1	12	3
4578	0.474414	0.811806	2.18901	0.0423874	0.0189222	180	apple	0	11	12
9089	0.405904	1.25822	1.6964	0.0304203	0.0196124	72	banana	1	6	18
3177	0.690854	2.74395	2.32979	0.0340794	0.0275027	103	carnivore	0	17	9
8513	1.23568	1.02623	0.73667	0.0448152	0.0110258	166	carnivore	0	9	4
4504	1.23872	2.19155	4.70109	0.0448152	0.0103031	42	apple	1	12	3
7392	1.75564	0.0150187	1.12461	0.0448152	0.0110258	166	carnivore	1	15	8
8521	1.23872	1.02623	0.720702	0.0448152	0.0110258	166	carnivore	0	12	4
5588	1.55485	0.448703	0.869813	0.0436728	0.0110258	150	carnivore	1	15	15
297	0.383544	0.689681	0.869813	0.0438005	0.0110258	166	carnivore	1	3	3
8522	1.23872	1.02623	0.720702	0.0448152	0.0110258	161	banana	1	9	4
2684	3.64375	1.02008	1.02352	0.0446732	0.0132729	190	carnivore	0	11	9
4578	0.474414	0.232464	0.510924	0.0423874	0.0189222	180	apple	0	11	14
6902	4.24902	0.658838	0.271823	0.030331	0.0133839	166	carnivore	0	13	3
4578	0.474414	0.678679	2.18901	0.042861	0.0189222	180	apple	0	12	11
7038	2.76126	0.00344869	4.1689	0.030954	0.0304405	160	banana	1	9	16
303	3.38666	2.7528	0.852809	0.0267988	0.0114105	155	carnivore	0	3	17
1248	0.692509	1.40076	4.44536	0.024507	0.0341542	168	banana	1	9	13

Chapter 6

Conclusions



```
Time :         1769 Old Age :     0 Age:      1769  Age:      1765
Population : 20     Starved :     0 Hunger:   3.659788 Hunger:   5.408914
Births :       0    Dehydrated : 0  Thirst:   2.530000 Thirst:   1.340002
Food Ate :     0    Discomfort : 0  Mate:     3.852519 Mate:     1.653124
                                    Profile: migrate   Profile: breed
```

Figure 6.1: A screenshot from the final application. In it you can see the three main species and the two plant food types.

Although this project may only represent a stepping stone towards the possibilities of of this system that are yet to be stored, it shows a very promising proof of concept. With only eleven variables making up the genotype of a creature it already shows promising speciation and interesting emergent behaviours, allowed through the implementation of CoEMAS. The high-level approach to genes proved it's merit, as the system went through stages of updates it was easy for new genes to be added on and it never became obtuse. By simplifying genetics down to only what's important it allowed for more focus and quicker development cycles than that of a low-level approach.

The agents were required to manage their needs, such as thirst and hunger, if managed correctly over an extended period of time their comfort value would increase. This value proved to be particularly useful, taking inspiration from the second and third layer of Maslow's hierarchy of needs, it attempts to quantify the general stability of a creatures living conditions as well as it's mood. Once this value has met a threshold dictated within it's genes the creature would seek a mate. taking into account both proximity and the genetic similarity of a mate.

Along with this, enforcing a flat breeding cost along with the sharing of a quarter of the remaining resources from the parents to the offspring, necessitated the creatures to adapt their needs into a hierarchy to not only ensure they survive the breeding process, but their offspring also had a sufficient amount to begin life. The combination of these physical and biological laws allowed the creatures to model an estimation of lifelike behaviour, without massive computational requirements.

6.1 Future Improvements

There are many areas within this project I would like to improve upon or add further functionality At this point layering more complex behaviours on top would not cost too much in terms of time as the framework has already been built. A wider variety of plant food and the introduction of non-fixed food sources, such as mushrooms. These more complex food types could even be given their own genetic structure, adding to the dynamic nature of the ecosystem.

Another area to work on in the future is the expansion of the senses, allowing for scent marking and more advanced speech functionality. Using values passed on through genetics or possibly even learned behaviour through artificial neural networks and hebbian learning would allow for each species to recognise different scents in their own way. A scent to one creature might signal a mate is nearby prompting it to follow the trail, but to another it may signal to avoid the area as it belongs to a predator. Species may also learn to in later generations mimic the scent of other creatures, this could lead to predators using scents to attract unwitting prey.

Arguably the most important improvement that must be made is the optimising of performance. Although there aren't any issues with it at this stage, it runs well even on low-end devices, making sure the foundations of this system are as close to optimal as feasible is important to ensure stability and scalability as new features are added. For example, currently during mating as creatures are not contained within the tile object, but all within a separate vector, the whole creature container must be scanned for a mate. This on a much larger scale would run into performance issues.

The movement system could be reworked to allow for full 360°movement, as well as allowing agents to be placed on more precise coordinates, this would not necessarily require more front-end work as all of this doesn't need to be shown. By moving onto a degree system a more accurate representation of physics could be implemented, such as acceleration and inertia, and how these concepts tie into energy usage.

One of the biggest undertakings in the continuation of this project will be implementing learning capabilities. This will involve the usage of artificial neural networks and hebbian learning. This might for example allow a creature to adjust their profile threshold values based on environmental stimuli, such as ignoring food if a predator is nearby. It may also allow them to learn in their lifetime which food sources to prioritise. They may even remember which areas of the map contain large concentrations of sources so that they could migrate back there later, possibly even forming a migratory path around the map.

Glossary

AI artificial intelligence. 10

alife artificial life. iii, 1, 7, 8, 9, 10

CA cultural algorithm. 6, 15

CEA co-evolutionary algorithm. 6, 15

CoEMAS co-evolutionary algorithm. iii, vi, 1, 2, 15, 23, 30

EA Evolutionary Algorithm. iii, 1, 4, 5, 6, 7, 15

EMAS evolutionary multi-agent system. 7, 15

GA genetic algorithm. 5

HUD heads-up display. 18

MAS multi-agent system. 7, 15

RPG role-playing game. iii, 2

UI user interface. 1, 2, 17, 18

References

[1] M. A. Bedau, 'Artificial life: Organization, adaptation and complexity from the bottom up,' *Trends in cognitive sciences*, vol. 7, no. 11, pp. 505–512, 2003 (p. 8).

[2] D. Cliff and S. Grand, 'The creatures global digital ecosystem,' *Artificial Life*, vol. 5, no. 1, pp. 77–93, 1999 (p. 10).

[3] C. A. C. Coello and R. L. Becerra, 'Evolutionary multiobjective optimization using a cultural algorithm,' in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*, IEEE, 2003, pp. 6–13 (p. 6).

[4] D. Crane, *Little computer people*, [CD-ROM], 1985 (p. 8).

[5] R. Dreżewski and L. Siwik, 'Agent-based co-operative co-evolutionary algorithm for multi-objective optimization,' in *International Conference on Artificial Intelligence and Soft Computing*, Springer, 2008, pp. 388–397 (p. 7).

[6] ——, 'A review of agent-based co-evolutionary algorithms for multi-objective optimization,' in *Computational Intelligence in Optimization*, Springer, 2010, pp. 177–209 (p. 4).

[7] GNU Project, *Ncurses*, version 1.9, 20th Jun. 2005. [Online]. Available: `ftp://ftp.gnu.org/gnu/ncurses/` (pp. 1, 13, 18).

[8] N. Gracias, H. Pereira, J. A. Lima and A. Rosa, 'Gaia: An artificial life environment for ecological systems simulation,' in *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems Eds C Langton, T Shimohara (MIT Press, Cambridge, MA) pp*, 1997, pp. 124–134 (p. 26).

[9] S. Grand and D. Cliff, 'Creatures: Entertainment software agents with artificial life,' *Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 1, pp. 39–57, 1998 (p. 10).

[10] P. E. Hart, N. J. Nilsson and B. Raphael, ‘A formal basis for the heuristic determination of minimum cost paths,’ *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968 (p. 24).

[11] L. Jiao, H. Wang, R. Shang and F. Liu, ‘A co-evolutionary multi-objective optimization algorithm based on direction vectors,’ *Information Sciences*, vol. 228, pp. 90–112, 2013 (p. 6).

[12] C. Lab, *Creatures*, [CD-ROM], 1996 (p. 9).

[13] C. G. Langton, *Artificial life: An overview*. Mit Press, 1997 (p. 8).

[14] Maxis, *Simlife*, [CD-ROM], 1993 (p. 9).

[15] S. McLeod, ‘Maslow’s hierarchy of needs,’ *Simply psychology*, vol. 1, pp. 1–8, 2007 (p. 14).

[16] K. Perlin, ‘Noise hardware,’ *Real-Time Shading SIGGRAPH Course Notes*, 2001 (p. 18).

[17] A. Renfrew, ‘Dynamic modeling in archaeology: What, when, and where,’ *Dynamical Modeling and the Study of Change in Archaelogy. Edinburgh University Press, Edinburgh, Scotland*, 1994 (p. 6).

[18] C. W. Reynolds, ‘Flocks, herds and schools: A distributed behavioral model,’ in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 25–34 (p. 8).

[19] R. G. Reynolds, ‘An introduction to cultural algorithms,’ in *Proceedings of the third annual conference on evolutionary programming*, World Scientific, 1994, pp. 131–139 (p. 6).

[20] K. Socha and M. Kisiel-Dorohinicki, ‘Agent-based evolutionary multiobjective optimisation,’ in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No. 02TH8600)*, IEEE, vol. 1, 2002, pp. 109–114 (p. 7).

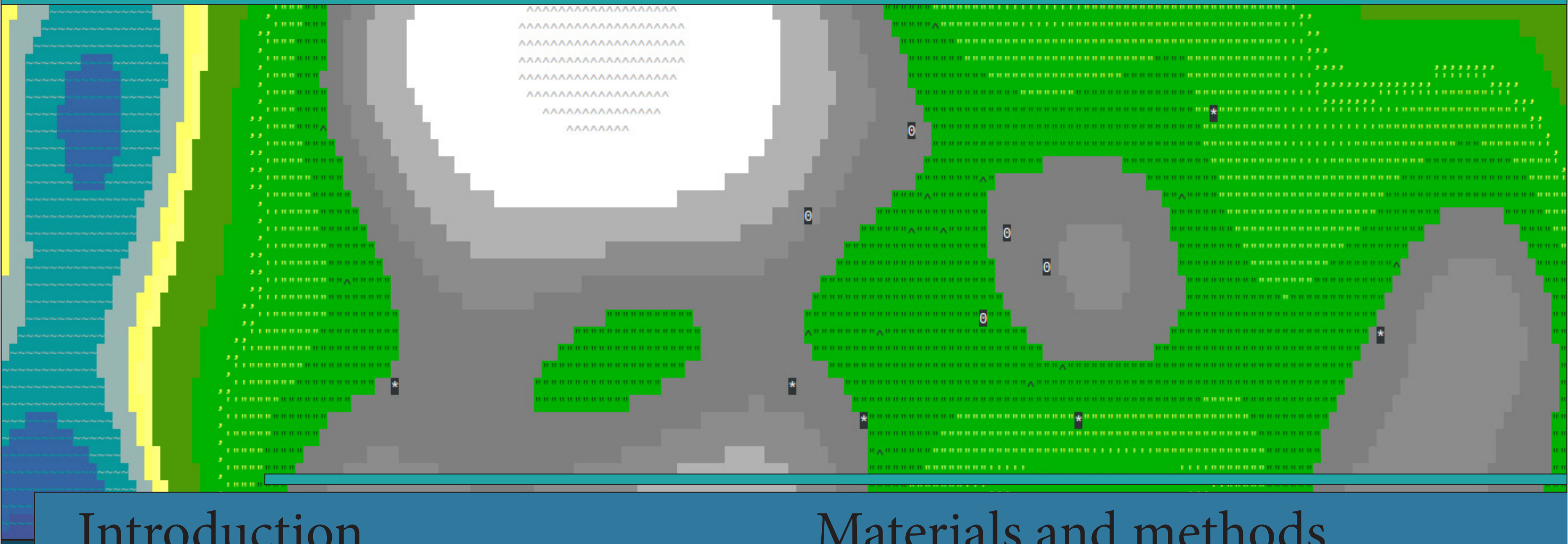
[21] X. Tu and D. Terzopoulos, ‘Artificial fishes: Physics, locomotion, perception, behavior,’ in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, pp. 43–50 (p. 8).

[22] P. A. Vikhar, ‘Evolutionary algorithms: A critical review and its future prospects,’ in *2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC)*, IEEE, 2016, pp. 261–265 (p. 4).

[23] G. Zames, N. Ajlouni, N. Ajlouni, N. Ajlouni, J. Holland, W. Hills and D. Goldberg, ‘Genetic algorithms in search, optimization and machine learning.,’ *Information Technology Journal*, vol. 3, no. 1, pp. 301–302, 1981 (p. 5).

Simulating artificial life with agent-based CCoEMAS

PRIFYSGOL BANGOR UNIVERSITY



Introduction

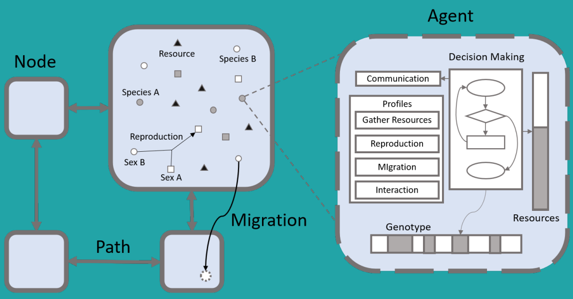
In this project we delve into the possibilites of creating a diverse and immersive environments for video games and research purposes by evolving varied critters within a simulated environment. A system like this one could be used as a tool for game developers to create interesting ecological life within their game worlds.

Materials and methods

Co-operative co-evolutionary algorithms for multi-agent systems (CCoEMAS)[1] is a special form of EA that allows for more diverse speciation and more interesting interactions. The agents are capable of both competing and co-operating with each other for resources used to sustain themselves and reproduce.

Results

The results from this project proved to be very interesting, as more layers were added to the system agents would show increasingly complex emergent behaviour. Examples of this included but are not restricted to reproduction niche (large group of critters concentrated around a patch of food)[2] and flocking.



Conclusions

Although the current behaviour allowed within the system is very basic it shows a lot of promise, by building upon the rules already in place more interesting behaviour can emerge. As agents learn how to cooperate and compete within their environment more complex hierarchies will emerge creating a rich and diverse ecosystem.

Future improvements

I would like to expand the system to include evolving advanced plant life which could co-operate with the wildlife in the environment to further propogate. An example of this would be barbed seeds that would attach itself to wildlife or fruit containing seeds that would be ate and later excreted by the creatures into the environment.

Literature cited

[1] Tan, K.C., Lee, T.H., Yang, Y.J. and Liu, D.S., 2004, October. A cooperative coevolutionary algorithm for multiobjective optimization. In 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583) (Vol. 2, pp. 1926-1931). IEEE.
[2] Gracias, N., Pereira, H., Lima, J.A. and Rosa, A., 1997. Gaia: An artificial life environment for ecological systems simulation. In Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems Eds C Langton, T Shimohara (MIT Press, Cambridge, MA) pp (pp. 124-134).

Author - Gary Ferguson
Supervisor - Dr. Franck P. Vidal