Predator/Prey NPC behaviour in video games

Edgar Park - 40310835

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc (Hons) Games Development

School of Computing

April 23, 2024

Authorship Declaration

I, Edgar Park, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed:

Date:

Matriculation no: 40310835

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

In video games, non-playable characters (NPC) play a major role when it comes to making the game fun and engaging to play. Up until now, NPCs used a pre-determined set of rules that allowed them to perform repetitive actions, making the game fun, but occasionally boring. While this is still being used, nowadays we see more and more games that use neural networks to perform tasks that depend on player actions and the conditions of the environment.

This dissertation paper provides insights into research, implementation and tests performed in NetLogo simulation environment. The aim of this research is to identify specific evolutionary algorithms that would improve the behaviour of an NPC by making it more dynamic and adaptive. Furthermore, we specifically focus on NPC behaviour in extreme conditions, such as environmental change (wildfire/flooding).

We investigate video games and their approach to AI, with the focus on neural network and evolutionary algorithms. Furthermore, we look at a number of attempts to implement an evolutionary algorithm in other video games and identify that NEAT or rtNEAT could be suitable for implementation. We chose NetLogo environment, where we can create a simulation that would use a similar approach to NEAT.

We discuss in detail agent based modelling, agent rules and biology inspired population models that is used in the implementation. In addition, we provide insights into neural network and genetic algorithm that are the main components required to create a simulation.

The implementation provides a detailed description in creating the simulation with agents that use neural networks and evolve them using genetic algorithm. Furthermore, we provide details about environmental change implementation and the necessary user interface setup.

By testing and analysing the results, we were able to draw a conclusion that using evolutionary algorithms and neural networks can be a valuable tool for future games, especially in creating an adaptive and dynamic NPC character.

Contents

List of Figures

Acknowledgements

I would like to thank my supervisor Dr Simon Powers for all the support, advice and guidance throughout the development of this honours project and my final year at Edinburgh Napier University.

In addition, I would like to thank my second marker, Dr Andreas Steyven, for the support and advice given in the first trimester.

Finally, I would like to thank School of Computing for providing me with the necessary knowledge required to complete this project and make the most from my education.

1 Introduction

The content in the following sections provide initial information about the project. Its aims with deliverables and constraints is discussed as well as motivation behind the idea. Finally, chapter outlines provide brief insight into what is further ahead in this dissertation.

1.1 Motivation

With artificial intelligence entering the spotlight and gaining popularity in recent years, it can be seen that the future world will be different from what is it now. Robots and AI will have a dominant presence in technology department, automating certain tasks that help humans in a daily life. Furthermore, by using AI, the workload can be significantly reduced and a lot of time saved when developing a new product such as, apps or video games.

In recent years, more game companies are using AI not only to automate games development, but also using it to create game worlds and characters. AI has been in development for a long time and nowadays the improvement can be seen in machine learning, which is often used in video games. Each machine learning technique is an improvement of the previous one; therefore, attracting an interest of engineers who want to see how far this technological evolution can go.

The motivation in this project comes from an interest in video game characters and how they can learn and behave in a realistic way by using evolutionary algorithms. Using NEAT and rtNEAT to train agents is still a fresh topic and a lot can be discovered and improved, Stanley (2002). NEAT creates offsprings that are developed from the fusion of genetic algorithms (GA) and artificial neural network, Mohabeer (2013). By using this algorithm, it is possible to find a solution to making a video game AI learn and behave in a more realistic way than it is now with a possibility for it to be less computationally expensive than machine learning. Currently, both EA and ML can be computationally expensive, especially in bigger games that require complex algorithms and a lot of data. Finally, with time, further research and experimenting, the solution will eventually be found to make it less computationally expensive. The goal of the project in this dissertation is to discover new ways of achieving dynamic agent behaviour using evolutionary algorithms.

1.2 Research Question

The following research question is to be considered in this project:

- How can evolutionary algorithm be used in creating a more dynamic and adaptive agent behaviour in a video game world where environmental change has impact on life?

1.3 Aims & Objectives

The aim of this project is to research, implement, test and evaluate an improved predator/prey AI behaviour system using evolutionary algorithms. In order to achieve this, the following is required:

- Identify evolutionary algorithms that can be used in a predator/prey behaviour system implementation.
- Implement population models that simulate a living environment.
- Implement a realistic environmental change scenarios that would possess threat to living organisms.
- Implement a neural network control for individual agents.
- Implement evolutionary algorithms that evolve agents neural network and improve their performance over generations.
- Run experiments on multiple simulations and gather data about differences/improvements/any unusual AI behaviour.

1.4 Deliverables

Deliverables of the project:

- A simulation that shows prey/predator agents evolving and surviving over time in different environments.
- Test results on how well agents evolved and analysis of any improvements/differences found during observation.
- Dissertation paper that provides insights into research information.

1.5 Constraints

A few factors that could affect the overall success of this project:

- Time - is required to learn evolutionary algorithms, such as rtNEAT and how it works. Furthermore, time will be required to learn NetLogo programming language as well as various population models (Lotka Volterra, Logistic Growth, Sugarscape) that will be used to solve the problem.

1.6 Chapter Outlines

- Chapter 1 - Introduction. Information about the project, aims & objectives, research question and motivation.
- Chapter 2 - Literature Review. A critical analysis of an existing research and existing work related to this project.
- Chapter 3 - Methodology. Approach and methods that were used during implementation. Also, detailed explanation of how the simulation was implemented.
- Chapter 4 - Testing. A discussion and a list of test results that were carried out.
- Chapter 5 - Conclusion. A summary of the dissertation and future work.

2 Literature Review

2.1 Introduction

Over the past few years, artificial intelligence started to rise and gain more recognition in a modern day society. From simple robots that help in a daily life to complex systems that learn from user input and produce new solutions to problems, such as ChatGPT. Artificial intelligence is primarily used in software and robotics; however, more now than before AI can be found in video games, which is a great achievement in gaming industry. Therefore, many companies nowadays use artificial intelligence not only for simple tasks in creating a video game, but it is also used to give life to NPCs (non-playable characters).

Artificial Life or A-Life is a form of artificial intelligence that attempts to re-create life-like organisms. It can be in any form, such as software, hardware or biological. Whilst there is a lot of research about artificial life, there is a gap when it comes to predator/prey behaviour in a virtual world. In fact, according to research, there is not much known about how animals behave in extreme conditions and how it could be re-created in a video game world (Yamada et al., 2020).

This literature review investigates and discusses artificial life used in video games and in particular, A-Life used for non-playable predator/prey characters. Furthermore, it looks into their behaviour in extreme conditions such as weather and environmental change. In the end of this literature review, it should be clear to the reader about what this dissertation is trying to achieve and provide enough knowledge to understand how A-Life works in video games.

2.2 Artificial Life

2.2.1 Overview

What exactly is Artificial Life (A-Life)? It is a field of study that is concerned about natural life, processes and evolution through the simulation of computer models, robotics or biochemistry. A-Life has three categories where it tries to solve various challenges. It includes the origin of life, evolutionary potential and life's connection to mind and culture (Bedau, 2003).

It is widely used in a variety of fields such as science, engineering, entertainment and other. The following discussion introduces brief introduction to various approaches when creating artificial life; however, the main focus is on evolutionary algorithms and the techniques used.

2.2.2 Soft Life

"Software Life" is an approach that involves computer simulations that can emulate life-like behaviour. For instance, a computer virus is a form of artificial software life. It has the ability to replicate and spread across other systems while dealing damage to software (Belk et al., 2020). This behaviour is negative; therefore, antivirus and firewalls use machine learning techniques to learn new threats and provide security by preventing virus from spreading.

2.2.3 Hard Life

"Hardware Life" focuses on robots and hardware implementation that can produce a life-like behaviour. This approach is usually combined with "soft life" to make autonomous entities that can be used in everyday life. A study about hard life suggests that western countries are more focused on developing AI whereas Asian countries are developing more human, animal-like robots (Belk et al., 2020). If both continents would combine their efforts, it could lead to new discoveries and evolution.

2.2.4 Wet Life

This approach to A-Life focuses on biochemical elements; therefore, it is used to create artificial organs. Scientists, doctors and researchers are working closely together to find a solution to extending human life. Artificial organs are already available and being used to replace some human parts such as heart or missing limbs. Langton (2018) discusses a biology of such possible life and investigates how artificial life can be made by a human, rather than nature.

2.2.5 Transhuman Life

This type combines all three approaches together. It argues that human mind can eventually be uploaded into a robot; therefore, creating immortal humans. Scientists and researchers are constantly working towards discovering a way to achieve this goal however, this may not happen anytime in the future due to how complex human mind is and not enough information about transferring someones' mind into another entity (Mohanty, 2023).

2.3 Neural Network

2.3.1 Overview

Neural networks are used in machine learning and it is a technique that tries to simulate the behaviour of a human, animal or another organism brain activity. With such a powerful technique, humans were able to evolve in the world of computers and robotics. Because computer thinking is faster than a human brain, it does not only provide the speed advantage, but it can also learn and adapt to certain conditions and evolve on its own (Zou et al., 2008).

There are different techniques of machine learning that serve for different purposes. In this section, the focus is on certain techniques that make neural network learn and improve overtime. It is used more often now than before, especially that this world has entered an AI generation, where many things have some sort of machine learning implemented. This includes video games that often use neural networks to simulate human/animal behaviour in a realistic way (Chan et al., 2022).

2.3.2 Deep Learning

Deep learning is one of the machine learning techniques that is used in artificial neural networks to learn from data. This algorithm requires data sets to be inputted so that it could learn the expected outcome and then use it to recognise similar patterns. Deep learning is widely used in image recognition, natural language processing, speech recognition and many more. Deep learning is relevant to supervised, unsupervised and reinforcement learnings (Schmidhuber, 2015).

Lecun et al. (2015) argues that unsupervised learning is being overshadowed by the success of supervised learning. The use of labelled data and known outputs enables supervised learning to be more accurate and efficient; however, unsupervised learning is becoming more used in human and animal learning, with no known outputs, thus the algorithm requires observation of the surroundings to learn similar patterns (Moshayedi et al., 2022). Both learning techniques can be seen being used in video games. Often, supervised learning is combined with reinforcement learning to train the model; therefore, improving performance and increasing robustness (Justesen et al., 2020).

2.3.3 Reinforcement Learning

Reinforcement learning is a powerful technique that is used very widely in robotics and video games. It is concerned about interaction with the environment and making a series of decisions that lead to failure or success. The goal of RL is to enable one to be punished when making a mistake or to get a reward when making a correct decision that ultimately leads to success or a goal (Eldahshan et al., 2022).

In games development, it requires a significant amount of time and effort to research and understand how RL works and how to implement a wanted behaviour. With deep reinforcement learning, engineers can establish a large set of rules to attain a desired behaviour; however, Eldahshan et al. (2022) argues that rule-based system is obsolete due to high-density data in machine learning in recent years. This statement might be true in some cases; however, rule-based approach is popular and widely used in many applications in robotics, healthcare, video games and other areas.

In video games, engineers often create algorithms that can be used by AI to play against itself and improve overtime. This can vary on different games, but such an examples would be "Go" where AlphaGo algorithm played against itself until it became very powerful and started defeating the best players in the world (Matsuo et al., 2022). Another example is AlphaStar in StarCraft. It played itself, learned new strategies, techniques and then competed against the best StarCraft players and beat most of them. Both algorithms were developed by DeepMind. These few examples show how powerful reinforcement learning can be and that the use of A-Life and RL can be beneficial for both (Ha, 2019).

2.4 Evolutionary Algorithms

2.4.1 Neuroevolution

This machine learning technique is an effective approach to solving reinforcement learning problems. It is used in artificial life and evolutionary robotics by applying evolutionary algorithms in order to create an artificial neural network (Lehman & Miikkulainen, 2013). The inspiration for this technique is taken from natural selection, which focuses on evolving organisms and adaptation to environment.

"Genotype" is a set of parameters that define the structure and connectivity of the neural network. It has layers with neurons, weights and biases. Each genotype is encoded and tested inside a neural network. Then the performance values are recorded and determined for genotypes; therefore, a new population is generated by combining them or changing neural network encoding (Risi & Togelius, 2015). Neuroevolution has two types of encoding: direct and indirect. Direct encoding focuses on individual neural network that directly represents its phenotype. It allows genotype to specify the number of neurons, weights and connection between them in a network. Whereas indirect encoding focuses on the genotype that specifies a set of rules for generating the networks' phenotype.

Neuroevolution plays a major role in video games as it helps to create a living environment and more enjoyable/challenging experience for the player. According to Lehman & Miikkulainen (2013) survey, majority of cases where neuroevolution was used, was to learn to play a game or control a non-playable character. Finally, neuroevolution is a powerful tool to train ANN and it is becoming more popular in the field of AI.

2.4.2 NEAT

Following NE, there is "NEAT" - Neuroevolution of Augmenting Topologies which, is an extension of neuroevolution. It evolves both the topology and parameters of the neural network (Ibrahim et al., 2019). It starts simple and adds complexity to the hidden layers overtime; therefore, changing neural networks' topology and weights (Gluck & Wightman, 2015). The initial population grows by adding new connections and nodes. When thinking of links and neurons in neural network, mutation and crossover is used to evolve them. With crossover, a few combined artificial neural network genes (usually taken from two parents) create a new chromosome (that is used in offsprings), which helps to evolve and explore new topologies (Pénisson et al., 2020). Furthermore, by using mutation, it is possible to randomly change these genes and create new traits or abilities, whether it's good or bad.

NEAT is used to accomplish a variety of tasks, from controlling robots to performing complex tasks, such as playing video games at a superhuman level. The way it works, it starts with a population of small genomes and over each generation, new genes are added. Each gene expands their search space; therefore, it is added to a search space where it did not exist before. This approach can lead to creation of more complex phenotypes, thus it opens up possibilities to discover new solutions to world problems.

2.4.3 Mutations, Crossovers and Offsprings

Mutation and crossover are two genetic operators that are used to create an offspring. With mutation, neural network weights and structure are randomly changed as well as nodes can be added or removed; therefore, this can introduce a new genetic material and improve neural networks overall performance. Moreover, by using crossover, the two chromosomes (from both parents) can be combined and assigned to an offspring in a new generation. Once an offspring is created, it is added to the general population and new fitness is calculated until the end of current generation. Therefore, by using selection methods, the fittest offsprings will be selected and they will undergo the same procedure that will make the next generation even more evolved.

2.4.4 rtNEAT

A further extension of NEAT is real time NEAT (rtNEAT). With NEAT, all training is done offline and the best neural network is selected whereas rtNEAT training occurs in real time; therefore, enabling the ability to learn fast and adapt to changing environment. During the process, an agents fitness is evaluated over some period of time. Thereafter, a selection process occurs to find most suitable individuals to be parents whose chromosomes can be crossed-over and mutated to produce an offspring (Liang et al., 2023). A continuous evaluation and evolvement improves the overall performance; however, it depends on a video game and specific scenarios if performance will be better or worse.

Olesen et al. (2008) explores dynamic game balancing between NEAT and rtNEAT in real time strategy (RTS) games. NEAT was used to optimise the performance of the agents and to investigate challenge factors; therefore, the results showed that challenge factors contribute towards the increasing challenge. Then rtNEAT was implemented in order to minimize the difference of challenge between the agent and the opponent in real time. The end result produced the evidence for the effectiveness of real time learning suggesting that rtNEAT can be successfully applied for dynamic game balancing in RTS games. On the other hand, this approach might not be suitable for games that have less interaction between the player and the opponent.

2.5 Video Games

2.5.1 Overview

Video games are on the rise in recent years, with new technologies and realistic graphics emerging, which brings new games into the spotlight. They are used not only for fun, but also to educate and learn too. Various simulation games help industry professionals to improve their knowledge and experience as well as to teach others. What makes the game interesting to a player? Each person would have a different answer; however, research suggests that a game world with non-playable characters (NPCs) is what makes most games alive and immersive.

This is where neural network, reinforcement learning and evolutionary algorithms come into play. With the use of these techniques and approaches, an NPC AI can become alive and behave in a realistic way while still evolving. The most common approach to NPCs is to use reinforcement learning so they could learn from its own experience. Another approach is NEAT; however, it is not used as much because it can be a slow process to evolve neural networks. In addition, it is likely that in the future, NEAT will be better researched and new ways will be discovered to make the agent training more efficient with the focus on multi-objective learning.

2.5.2 NPCs (Non-Playable Characters)

Non-playable characters have an important role in video games. They add realism and immersion whilst being a crucial part of the game, or simply just making the world less empty and boring. NPCs can be either very simple and walk from A to B and have a small dialogue, or they can have a complex AI system that allows them to live a real life and evolve. Most of the games use scripted NPC behaviour trees, which become repetitive and can make the game boring after some time; therefore, making the game less replayable (Reddemann, 2015). On the other hand, there has been a rise in newer games that use neural networks and evolutionary algorithms more often, which help NPC characters to evolve and have random behaviour that makes the gameplay more interesting to the player.

According to Schrum (2008) most of the learning methods involve a single objective when learning; however, NPCs in newer games have multiple objectives to accomplish, which can make the game very complex and difficult for learning methods like reinforcement learning to decide how it should be optimised. For this problem to be solved, there is a need of multi-objective optimization (this is where EAs come to the rescue).

2.5.3 Predator/Prey NPCs

When hearing words predator/prey, the first thought that comes to mind is "one animal hunting another one and vice versa"; however, this is not always the case. Predator and prey can be any kind of organism that tries to survive and it involves hunting and evasion.

When looking at video games, the same principal applies to NPCs too. For instance, Pac-Man is a game where ghosts are hunting a player until it collects a special pellet that reverses the mechanic and a player is able to hunt ghosts for a short duration. The original Pac-Man does not use neural networks or AI. Instead, it uses a pre-programmed rule based state machine that allow ghosts to behave in a certain way depending on the players location (Yannakakis & Hallam, 2005). Some of the modern day Pac-Man is using neural network and AI to make the game more random and fun (Sun, 2015). Ghost motions are managed by a fully connected multi-layered feedforward neural controller; therefore, it uses sensors to inspect the environment and decide the next move.

When creating neural networks for predator/prey NPCs, should simple evolution or coevolution strategy be used? Chen et al. (2019) talks about how they compare and which one is better. Multiple tests showed that when two or more populations are competing, coevolution outperformed simple evolution. However, this only works when populations are designed to either hunt or evade; therefore, they have to constantly improve and evolve (Nolfi, 1998).

Finally, using coevolution and NEAT can be a good combination in creating a competitive population that tries to affect each other's evolution. This is discussed further in the next section.

2.5.4 Coevolution

Firstly, evolution is a process when neural network weights and connections change over generations. Secondly, coevolution is similar to evolution; however, the difference is that coevolution requires two or more species that can interact with each other. This way all the species can affect each other's evolution. Chigot & Wilson (2022) discusses creating the environment and population at the same time, using coevolution. Instead of having two species, an environment is created to affect evolution of the population and vice versa. When the population learns the environment and it becomes easier, the environment evolves into a more challenging one and the process continues.

It is a difficult task to create agents that can behave in a realistic and complex way. However, coevolution is a field of study that is constantly improving and in the future, it will be possible to achieve complex scenarios much easier than it is now. Dziuk (2011) talks about "shaping" process which works well in both evolution and coevolution. It is a process where an agent can influence its own environment; therefore, it can be used alongside coevolution for improvement. For instance, if a predator organism is hunting for a prey, it will need to improve its own abilities before affecting preys evolution and the same process goes for prey, constant improvement; therefore, trying to affect another's evolution (Elfeky et al., 2021). Ultimately, this approach can make coevolution more powerful in competitive environments.

2.6 Environment

2.6.1 Overview

To further understand what will be explored in this project, a brief introduction about the changing environment and how NPCs behave under new conditions is required. The following will consist of some examples of environmental change followed by a further discussion of a change in behaviour in NPCs and how they adapt and survive, with the primary focus on predators/preys. Furthermore, a brief discussion about the population models will provide insights into how it can be used in artificial life to help organisms evolve and survive throughout their lifetime.

2.6.2 Environmental Change

Environment is something that every video game has. It can be anything that is in the surrounding area, things, organisms, weather and so on. The focus here is on the environmental change that can affect other living organisms in video games. In order to understand how this works, a research is needed into a real world environmental change. Over the years, a variety of factors caused not only the climate, but also the whole environment to change drastically (Hardy, 2003). Most of this is caused by humans; however, other factors, such as volcanic eruptions, earthquakes and tsunamis are making the planet to change as well. All this has contributed towards a different earth. Normally, the change in environment happens slowly; therefore, most of the living organisms have enough time to adapt and survive (Raihan, 2023).

In video games, environmental change is a growing topic; therefore, many games nowadays feature realistic environments that are affected by environmental/climate change (Dhiman, 2023). For instance, Far Cry 5 and 6 have some references to environmental change that not only affect players gameplay, but at the same time raise awareness about the causes and damage that can be done. Another game that deals with climate change is The Last of Us. It provides visual information about how the world would look and it reflects on challenges of living and surviving in a different looking world.

Overall, environmental change is a wide topic to discuss and beyond the scope of this project. Therefore, the focus will be on a few climate change factors that can help to understand and improve predator/prey behaviour in a video games.

2.6.3 NPC Behaviour

In the real world, every organism will react and behave differently. Some may adapt and survive easily whereas other organisms might not live long. It all comes down to knowledge and skills of survival. In a video game world, engineers try to recreate a realistic behaviour using neural networks and latest technologies; however, the downside is that it is hard to predict how organisms such as animals would behave in a different world. The research is still ongoing and there is a lot to be discovered, but for now, future forecasts and predictions can help to solve this problem (Bastille-Rousseau, 2018).

The effects of environmental change are likely to create new phenotypes and alter how predators/preys interact with each other. In a more extreme world, there will be more competition and hostility (Laws, 2017). It is likely that weaker organisms will try to improve their evasion abilities whereas predator organisms will hunt and they will only improve if there are no more weaker organisms left in their search area. Taking this into consideration, a shaping coevolution neural network (SCNN) of NPCs would develop faster.

2.6.4 Sugarscape model

Sugarscape is a population model for artificially intelligent agent-based social simulation (ABSS). It is used to study a wide range of areas such as wealth distribution, cooperation, conflict and inequality. The simulation works by having a two dimensional grid where agents can move around; therefore, harvest and gather sugar or spice, which represents wealth that agents use for reproduction, evolvement and engagement in other activities (Kehoe, 2016).

According to An (2017), using agent-based models such as sugarscape is becoming increasingly important in understanding life sciences. It is particularly valuable in the areas that are not yet fully understood. One such an area is “inequality”, which is a major factor when it comes to living and surviving in a group (equality). A study from Rahman (2007) reveals abnormal situations that occur when inequality increases, such as poverty and higher mortality. This is particularly relevant when studying predator/prey environment in artificial life and video games. In order to create a more realistic and dynamic behaviour, sugarscape can be combined with other population models such as Lotka Volterra and Logistic Growth. Therefore, experiment results could be surprising and it has the potential for new discoveries.

2.6.5 Logistic Growth

Logistic Growth (LG) is a mathematical model that describes how populations grow over time. It uses carrying capacity to understand when the limit of growth is reached; however, this depends on the resources that are available in the environment. Tsoularis (2002) talks about a variety of growth curve models, with each designed to address a specific research question whether it is about a small or large populations, or physical growth of a property for an organism. For instance, Triambak (2021) applied a power law growth model to COVID-19 fatality data. Multiple countries were used to collect data from the first wave and it accurately predicted epidemic growth rates in the following waves. A latent growth model (LGM) was successfully used before in predicting growth curves; however, it needed to be improved and a power law LGM was developed. With this variant, it is possible to get more accurate predictions; therefore, it can be used in variety of areas such as natural science and to measure predator/prey growth rate using s-curve (Kucharavy, 2015).

2.6.6 (Competitive) Lotka Volterra

Lotka Volterra (LV) is a population model that studies the dynamics of two interacting species. This model has been used for over a hundred years and its purpose is to apply mathematical equations in biology to study predator/prey behaviour (Cherniha, 2022). The classic LV model assumes that one species are feeding on the other species and that prey has unlimited amount of food supply (Anisiu, 2014). While using this model nowadays is still useful, in order to achieve a more complex behaviour, improvements are needed.

Over the years, there have been many LV variants developed in order to incorporate more complex dynamics. One such a variant is “Competitive” LV model. It extends the classic model by introducing competitiveness within the two living species (Zhu, 2009). Furthermore, an environmental change could have a big impact on a competitive level. Yin (2009) discusses having random environments and how the population would summer from sudden environmental shocks such as earthquakes, hurricanes etc. Moreover, equilibrium points (stable/unstable) are introduced, which is a state where the system does not change over time. The carrying capacity of an environment represents a stable equilibrium point for a population where population size will remain the same unless there is a change in the environment or population dynamics’.

Whilst CLV in an interesting variant worth exploring further, there are many other variants that could be useful such as omnivory LV where predator can eat other species too, or spatially explicit LV, which takes into account spatial distribution of two species that can lead to more complex dynamics.

2.7 Existing Work

2.7.1 Rain World

Rain World has one of the most complex ecosystems in a video game world, where all creatures and plants interact with each other in a realistic way. The game has multiple ecosystems, with each having different resources and predator/prey agents. The player controls an organism that has to feed and survive. Moreover, predators use a similar behaviour as they have to hunt the player in order to survive.

The interesting part of the game is how predators behave in certain situations. Their neural network evolves as the game goes; therefore, making them smarter and more hostile. A random gene mutation can lead to unexpected behaviours such as, if a player kills a predator, more predators from the same species will patrol the area. Another interesting behaviour is when a predator catches the player, it will try to take it back to its lair, but if it encounters another predator in its path, they will fight with each other, thus giving the player an opportunity to escape.

An important aspect in this game is the environment that changes depending on a day/night cycle. It is safer during the day and more dangerous at night. Consuming food and water is a crucial part of survival in this game as well as hiding from dangers. The player organism can befriend other creatures; therefore, there is chance that they would end up helping the player in some way. Every organism in the game has a simple neural network that evolves when interacting with other organisms.

Overall, Rain World is a great representation of various ecosystems and realistic behaviour/interaction that makes this game a great example for this project (Jakobsson, n.d.).

2.7.2 NERO

Neuro-Evolving Robotic Operatives (NERO) is a video game that uses rtNEAT to evolve the topology of a neural network. With rtNEAT, neural networks evolve in real time; therefore, allowing organisms to adapt to unpredictable environments and behave in a way that would make sense in the present situation (Stanley et al., 2005). The NERO game allows users to adjust various settings, such as how good or bad agent group should behave in certain situations, hide or shoot/stick together or go alone and more. Mohammad (2022) discusses how agents learn to navigate the maze in real time while a user is placing walls and creating different shapes of a maze. Each agent would evolve incrementally and further generations showed that each agent developed its own topology and completed the maze in different ways.

It can be seen that while agents learn in real time, each generation has increased its neural network complexity with it leading to a successfully accomplished task or problem solved. This is achieved by keeping the track of genes and later combining them with compatible ones (Miikkulainen, 2006). NERO is a great game to experiment with, also it can teach about how rtNEAT works while observing results in real time.

2.7.3 TORCS

The Open Racing Car Simulator (TORCS) is an AI racing game and a research platform. The game uses NEAT for training race cars and learning the race track. It is open source; therefore, engineers and developers can use the source code for editing and implementing new solutions to evolutionary algorithms and improving machine learning. There are many different versions of the game and for this instance, the focus is on a version developed by Cardamon (2010). The approach used in this project allows a finer online learning with performance improvements with each new lap. Two methods (NEAT and rtNEAT) were used on three race tracks and four evaluation strategies. After each lap, the AI agent was able to learn the path they just used and evolve its knowledge by adding new information as they go through further laps. Performance wise, rtNEAT was faster and better than NEAT and it allowed for an agent to evolve faster.

After further experiments, it can be seen that NEAT slowly catches up with rtNEAT and in the end, there is no significant difference between them. NEAT is able to produce more drivers capable of driving on unknown race tracks whereas rtNEAT focuses on a smaller population size to train. The final test results show that NEAT outperformed rtNEAT on more difficult race tracks. Finally, whilst NEAT trains more agents with better performance on more complex tracks, rtNEAT is a great method to use when training smaller populations with simplified fitness evaluation. Finally, rtNEAT can be further improved and eventually do better on complex race tracks.

2.8 Conclusion

To conclude the chapter, this literature review provides insights into artificial intelligence with specific focus on evolutionary algorithms and behaviour of non-playable characters (NPCs) as well as its adaptation and survival in changing environments. The exploration and research provides valuable information about how it works and how it can be achieved.

The first half of this literature review broadly discusses various approaches to creating Artificial Life. Thereafter, the topic was narrowed down to neural networks, where the research suggests different approaches to teaching an AI agent and therefore, evolutionary algorithms were introduced.
Neuroevolution is a method of evolving and improving neural network. This method plays a crucial role in games development. It allows creating a more dynamic and challenging NPC characters as well as environments. Furthermore, it helps in teaching NPCs how to play the game better or/and manipulate it to gain or lose advantage, thus enhancing player's experience.

We have also discussed an extension of NE, NEAT (Neuroevolution of Augmenting Topologies), an algorithm that goes beyond traditional NN training by evolving the parameters and structure of it. The way it works, it starts with simplest form of NN; therefore, gradually adding the complexity over generations, thus creating more sophisticated neural networks. Another way of training agents can be achieved with rtNEAT, which is an extension of NEAT with the capability to train in real time. This can be described as offline (NEAT) and online (rtNEAT) learning. Agents live in the world while learning at the same time and therefore, produce smarter offsprings. The only downside is that rtNEAT should be used in video games with more interaction between the player and other agents. It was suggested that NEAT should be used in games where AI agents spend more time idling than interacting.

Crossovers and mutations are crucial genetic operators that creates new chromosomes with mixed genes from parents and this is used for new generation offsprings. With crossover, genes of two NN can be combined and with mutation, it is possible to randomise the parameters or structure of NN. Both can be used together or individually.

Coevolution is a method that involves two or more species that compete and can influence each others evolution. This is an interesting method that can produce a more competitive and challenging environments in video games by increasing its complexity. As coevolution continues to evolve, by adding NEAT, there is a potential of creating a more realistic and challenging video games in the future.

Moreover, we discussed about various population models that are used in biology with important mathematical equations. This takes a big part in helping to understand how predator/prey interaction works; therefore, creating a path to more experimenting and new discoveries.

Finally, this literature review ends with some of the existing work such as Rain World or NERO with valuable insights into the practical applications of neuroevolution in video games. From complex ecosystems to AI controlled robots that use evolving neural networks in real time. Lastly, neural networks and evolutionary algorithms is a powerful tool for enhancing video game experience. The methods discussed are already being implemented in games; however, it is still a fresh field of study and a lot can be researched and improved in the future. As technology advances, more sophisticated and immersive worlds will be created for the players to explore.

3 Methodology

3.1 Introduction

There has been a significant advancement in the field of artificial life used in video games in recent years, with growing interest in non-playable character behaviour and adaptation to real world scenarios. This study aims to address this gap by investigating evolutionary algorithms, specifically NEAT/rtNEAT that would enable real time training of an agent. Furthermore, population based models from biology will be used to create agents that can live individual lives as well interact with other agents and learn as they try to survive. Moreover, by creating such a simulation, we can investigate the behaviour of the organisms and how they change and adapt to different environments over time.

This chapter contains detailed description of the implementation of predator/prey model used in NetLogo programming environment for creating this simulation. We will present two types of species that will be further extended with various population models and placed in challenging environments. To ensure the success of the simulation, multiple experiments/tests will be done to determine that both species can change their behaviour and adapt to changing environments whilst still competing for resources.

The environment in this project vary from climate change to natural disasters. Both environmental approaches offer an opportunity to study how agents adapt and survive together as well as individuals. Moreover, the experiment results and data gathered should provide a better understanding in changing agent behaviour; therefore, providing enough information for researchers and developers to develop a better predator/prey NPC system in video games.

Sections covered in this chapter:

- Programming Environment
- NetLogo
- Agent Based Modelling (ABM)
- Conceptualization
- Design

- Implementation
- Validation
- Conclusion

By the end of this chapter, the reader should have a clear understanding about the approach taken to solve predator/prey behaviour problem as well as enough information about the software used and how to recreate the simulation.

3.2 Programming Environment

The simulation was developed using NetLogo (Tisue (2004)) programming language and its environment. Furthermore, to achieve more realistic results, there is a need of a powerful computer; however, a laptop was used for development with the following specifications:

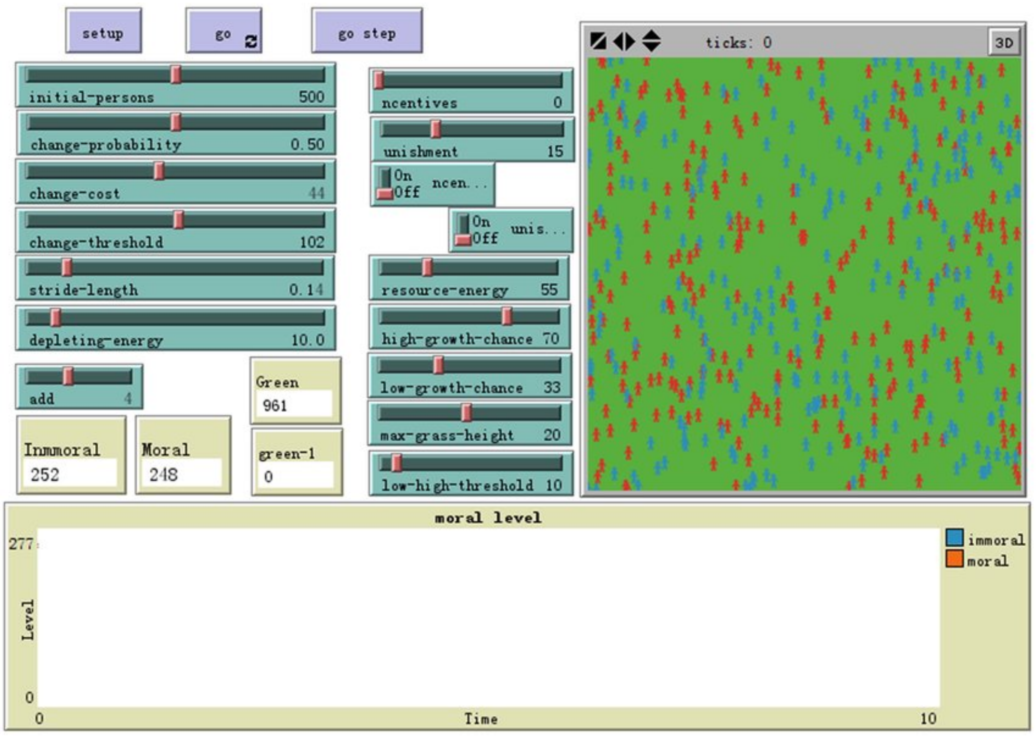
- OS: Windows 11
- RAM: 16GB
- CPU: Intel i5
- Graphics: Intel Iris Xe

3.3 NetLogo

NetLogo is a powerful cross platform programming language and modelling environment for simulating complex systems that evolve over time. By giving instructions to individual agents, a developer/researcher can observe their interactions and evolving behaviours; therefore, these kind of simulations are used in a wide range of disciplines, such as natural/social sciences and engineering. Moreover, NetLogo has a user friendly graphical interface which makes the development process much easier and faster. In NetLogo, individual agents are called "turtles" that navigate in the grid environment called "patches", that are programmable agents too. Agents can interact and do multiple tasks concurrently (at the same time). Furthermore, "observer" is another type of agent that gives instructions to turtles and patches, thus there can only be one observer per model.

NetLogo is specifically developed for the use of agent based modelling, which has the necessary components that are required for developing the simulation. It has some built-in tools that made this project slightly easier to implement such as, special primitives (breed, turtles-own, globals). For instance, patches-own is a shared property and every variable declared inside is associated with every patch in the world (which is convenient if you want to perform an action on all of the patches). In addition, libraries provide necessary tools to improve the simulation. For instance, in this project, we used "profiler" to test speed of the simulation and find out which function is taking the most time; therefore, making adjustments to make it run faster. In addition, NetLogo provides a bunch of other models that can be used to learn about a specific function or just understand how it works. Lastly, other users have created some experimental NetLogo libraries/extensions that can be downloaded and used in the simulation (no external assets were used in this project).

By using this software, it is possible to create a simulation with a changing environment whilst observing agent behaviour and interaction with each other. Furthermore, we are interested in experiment results; therefore, filling the gap in our research discussed in previous sections.



```
initial-persons        500
change-probability     0.50
change-cost            44
change-threshold       102
stride-length          0.14
depleting-energy       10.0
add                    4
ncentives              0
unishment              15
On/Off ncen...         Off
On/Off unis...         Off
resource-energy        55
high-growth-chance     70
low-growth-chance      33
max-grass-height       20
low-high-threshold     10
Green                  961
green-1                0
Inmoral                252
Moral                  248
```

Figure 1: An example of NetLogo Interface

3.4 Agent Based Modelling (ABM)

Macal (n.d.) explains, ABM is a computational method for studying complex systems that can be difficult to model with traditional approaches, such as mathematical or statistical models. The interaction with other agents and environment as well as the behaviour of individual agents can be captured using agent based modelling. Data gathered from tests can later be used to compare agent behaviour and to understand how it changes over time. ABM typically uses three components: agents, environments, rules.

Agents are autonomous organisms that function independently and interact with other agents as well as the environment that is within its range. It is an organism that can have attributes or characteristics, decision making capabilities, behaviours and more.

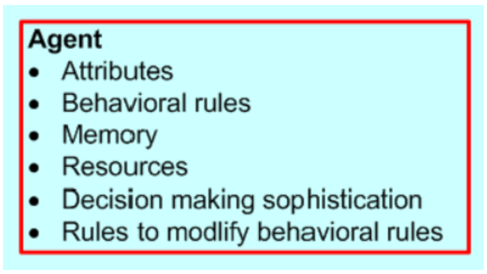

Agent
- Attributes
- Behavioral rules
- Memory
- Resources
- Decision making sophistication
- Rules to modlify behavioral rules

Figure 2: A Typical Agent

Agents use mechanisms that describe how they can interact with other agents, such as recognition of other organisms, communication, avoiding collisions, influence and other. Furthermore, agents can interact with the environment too, which makes them become situationally dependant; therefore, the behaviour will depend and change based on agent/environment interaction. An agent will often have a certain goal or condition that will drive its behaviour. Based on this information, they will analyse their own progress and make changes to its behaviour when required. In video games, agents are usually non-playable characters that use machine learning to analyse players interactions, behaviours, preferences and learn from it. An NPC is a crucial part of a video game world, because it makes the game world more interesting and engaging for the player. NetLogo can be used to do various experiments with NPCs and it does not require a very powerful computer. Moreover, the software provides a good testing environment and various algorithms can be implemented as well as a possibility to create a bridge between NetLogo and a 3rd party software for experimenting.

In NetLogo environment, agents have two types of interaction, direct and indirect. Direct interaction can have movement, resource consumption, state change, waste production and more. Whereas indirect interaction is mainly achieved through other agents, meaning that one agent could tell something another agent and if the first agent accomplishes the task it was given, the environment or patch could be affected in some way as a result.

Rules are established so that the agent would know what to do and how to change its behaviour accordingly. Some of the basic rules include movement, which specifies how an agent should navigate through the environment. Also communication, which tells the agent how to communicate with others, whether it is using signals or another way. Resource consumption establishes rules on how agents eat, drink or consume another resource. Lastly, the state change rules instructs the agent whether it is supposed to build or destroy something, thus affecting the environment.

In conclusion, using agent based modelling can simplify the complexity of the simulation that will be developed in this project.

3.4.1 Conceptualization

This section focuses on the concept of creating a simulation with realistic predator/prey behaviour and adaptation using various population models and evolutionary algorithms. Furthermore, the goal is to create a simulation, based on research conducted previously, which would allow us to answer the research question stated in this paper.

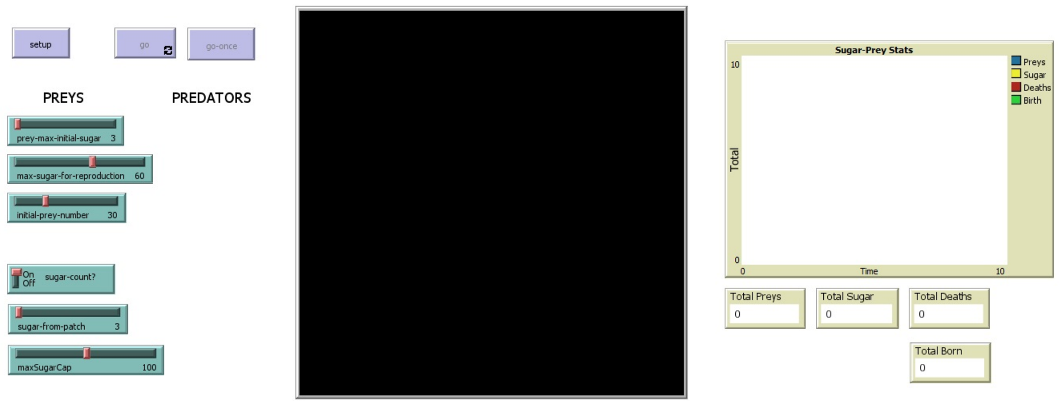
NetLogo provides the necessary user interface as well as a programming language that is used to build the simulation. The process is split into two parts. The first part is to build the foundation for the simulation, with the active agents. Moreover, a population model, sugarscape is used to keep the agents alive and interacting with the environment. This gives a starting point for further development. Furthermore, implementing population models such as Lotka Volterra and Logistic Growth increases agent complexity and their interaction by adding a more competitive and hostile environment. This requires the implementation of predator species that would be programmed to gain energy and feed on prey agents.

The second part of implementation focuses on extending the current simulation and adding hazardous environments, such as wildfire and flooding events. NetLogo makes it possible to implement various “patches” in the simulation with different settings; therefore, enabling agents to experience fast changing situations. The goal in this part is to achieve fast agent adaptation to changing environment with the use of neural network and evolutionary algorithms.

Structure

The structure of the UI in NetLogo is simple and straightforward. As seen in the figure below, we can have multiple monitors to show an increase/decrease in prey population as well as total amount of sugar available. The left hand side is mainly used for set up, such as how much sugar preys will have at the start of the simulation. By adjusting sliders, we can create a different environment and observe agent behaviour. The sugarscape model simulation act as an entry point towards a more complex implementation that will take place at a later stage. Details of full implementation can be found in the "Implementation" section.

3.4.2 Design



prey-max-initial-sugar 3
max-sugar-for-reproduction 60
initial-prey-number 30
On Off sugar-count?
sugar-from-patch 3
maxSugarCap 100

Total Preys 0
Total Sugar 0
Total Deaths 0
Total Born 0

Figure 3: Sugarscape - NetLogo UI

Agents

There are two types of agents that will be used in a simulation: Predator and Prey. Both can interact with each other as well as with the environment. Predator species follow a slightly different ruleset from prey species. They are more competitive by nature and more hostile; therefore, predators are implemented in a way that their survival and reproduction depend on prey hunting.

On the other hand, preys are more evasive and they survive by consuming sugar (or another resource that can produce energy) and reproducing themselves. We use Lotka Volterra population model to create a more competitive environment. Standard Lotka Volterra equations are as follows:

$$dx/dt = x - \alpha xy \tag{1}$$

$$dy/dt = -y + \beta xy \tag{2}$$

Where "x" is the size of prey and "y" is the size of predator populations. Both species are interacting and their population size depends on the population size of the other species. For instance, prey population size will increase when there are no predators present, and predator population will decrease when there are no prey in a vicinity. Furthermore, we extend the latter population model to include Logistic Growth, which essentially adds a carrying capacity to equations. The model is based on the idea that the population of a species will grow exponentially if there are no limitations to its growth, but eventually it will reach a carrying capacity, at which point the growth of the population will slow down and it will stabilize. The Logistic Growth equation for the prey is as follows:

$$x(1 - x/K) \tag{3}$$

With "x" being prey population size and "K" representing carrying capacity. In addition, predator equation is as follows:

$$y(1 - y/M) \tag{4}$$

Where "y" is the predator population size and "M" is the carrying capacity. The combined and updated equations for prey and predator are as follows, respectively:

$$dx/dt = \alpha x(1 - x/K) - \beta xy \tag{5}$$

$$dy/dt = \gamma xy(1 - y/M) - \delta y \tag{6}$$

With "α and δ" being growth rates for prey and predator populations, "β" - the predation rate, and "γ" - the consumption rate.

By incorporating Logistic Growth, we extended Lotka Volterra model and made the simulation look slightly more realistic; therefore, enabling the ability for predator and prey agents to be affected by available resource limitations as well as competition. While the equations provide a fundamental understanding of how these population models work, we do not have to literally write them in the code. That is because NetLogo uses pre-built models with these equations and therefore, they are already implemented at an individual level.

Environment

The are two scenarios where environmental change affects predator/prey behaviour. First, extreme heat/wildfire simulation creates unliveable conditions for the agents and reduces the amount of resources available. The fire or heatwave will start on the left side of the monitor and will spread across the neighbouring patches accordingly. By adjusting wind settings, we create a random direction of where the fire/heat will spread next. Furthermore, every burnt patch will become inedible and have zero sugar available. Finally, agents can walk on burnt patches; however, they will not find any food there and are likely to die.

The second environmental change is a flooding event (natural disaster). The behaviour in this simulation is very similar to heat/wildfire scenario; however, instead of a wind, water pressure is used to adjust how much pressure will neighbouring flooded patches add to a new not yet flooded patch. This way the flooding will occur randomly depending on where the sugar is. Soaked/Underwater patches will become inedible and have zero sugar; moreover, the agents cannot walk on water and therefore, will die instantly if they walk into it or if the flooding catches them first.

By implementing these two scenarios, we can recreate a realistic environmental change that is often used in video games (as well as happening in real world too).

Rules

This section introduces rules that agents will follow in order to achieve expected behaviours. It is essentially a set of instructions that tell predators, preys and patches what to do and how to interact with the environment. Each rule describe a single action or a behaviour.

Prey: It should maintain their population size and avoid being overexploited by predators. Prey should actively explore their environment and look for patches with enough sugar, which is crucial for their survival and reproduction. Furthermore, prey should be able to adapt to changing conditions in their environment by optimizing chances of survival, find more efficient strategies for reproduction and change their behaviour. This not only takes into consideration the environmental changes, but predator presence too. In addition, prey may engage in cooperative behaviours to protect themselves from predators. By adhering to these rules, prey species should effectively manage their population size, limit vulnerability to predators and maintain a stable coexistence in the ecosystem.

Predator: It should actively search for weaker preys by patrolling their surroundings. Predators should improve hunting strategies and adapt new behaviours based on prey availability and environmental change. In addition, they should learn from each other's hunting successes and failures; therefore, improve hunting efficiency over time. Predators should allocate energy efficiently between hunting and reproduction; also, they should regulate their population, keeping the numbers within the carrying capacity.

Patches: It should have a unique location within the simulated environment, which can be represented using coordinates. Each patch should have various properties that define its characteristics depending on the environment type, such as "flooding" or "wildfire". Patches should be connected to form a network allowing agents to move between them and interact with different environments. Patches should be extractable (prey feeding), also it should regenerate itself, such as growth of sugar. Patches should influence agent behaviour by providing various challenges in the environment. Finally, patches should use different colours for different environments, such as flood is blue, sugar is yellow, wildfire is red and so on.

3.5 Implementation

In NetLogo, we use the following world settings - 70x70 grid with patch size of 3 pixels (one pixel would produce the best results, but it requires a powerful computer) and 30 frames per second. In addition, four different simulations were created that were used for testing purposes. Other simulation are as follows: no neural network, no environmental change, no predator activity, full simulation. In this chapter, we provide details of the full simulation with all necessary elements and extensions.

3.5.1 Overview

The UI consists of monitors, plots, sliders, buttons and the main world screen. Monitors are used for showing the statistics of various elements such as predator/prey birth and death rate, whereas plots are used for showing the actual numbers, such as sugar level in the world. Furthermore, sliders are used for adjusting carrying capacity and controlling the maximum sugar amount. Buttons are there for setting up and running the simulation. Finally, the main world UI screen provides visual information of a running simulation. The UI layout can have more or less options and settings depending on researcher needs and requirements. For this simulation, we are interested in adjusting the initial amount of preys/predators as well as their carrying capacity. This allows us to test the capabilities of prey/predator behaviour and how they adapt to certain conditions when there is less or more species in the simulation.

Sugarscape model is implemented first and foremost, which is connected to the rest of the UI. By using NetLogo programming language, we create "breeds" that represent preys/predators, and patches that hold various sugar amounts (including 0 - no sugar). Next, we initialise prey breeds with various settings such as shape, size, vision and energy, which gives life to the agents inside the simulation world. Furthermore, functions such as "move, eat, die, reproduce" instructs prey agents to move around while looking for sugar; therefore, creating offsprings once enough sugar is collected. We further extend current model to include predators that hunt preys. They behave in a similar way, except it hunts prey and consumes its sugar that is converted into energy, which predators can use for moving and reproducing offsprings.

The simulation environment is created using patches that hold information about sugar levels or another resource (in this case burnt or flooded patches). The first implementation of environmental change is heat/wildfire that simulates extreme weather and/or burning patches that reduces rate of survival and the availability of resources. Next, we simulate a natural disaster (tsunami), where water (or the ocean) would cover sugar patches gradually; therefore, making them inedible. Finally, encountering the ocean would be fatal for any living species and result in instant death.

Both breeds (predator and prey) use simple neural network that allows agents to explore, learn and create new solutions. These solutions are then used in a new generation of "smarter" agents. To make agents learn in real time and behave in a more realistic way, we have implemented a genetic algorithm that evolves neural network weights. Essentially, genes are assigned to weights and once fitness is calculated, these weights can crossover and mutate. The procedure is the same; however, the calculation is slightly more complex. With the neural network and genetic algorithm hybrid, agents can evolve weights during runtime and then pass the information to new generations. By achieving this, we simulate a similar structure to rtNEAT of continuous evolvement. The following few sections provide detailed explanation of how certain parts of the simulation were implemented.

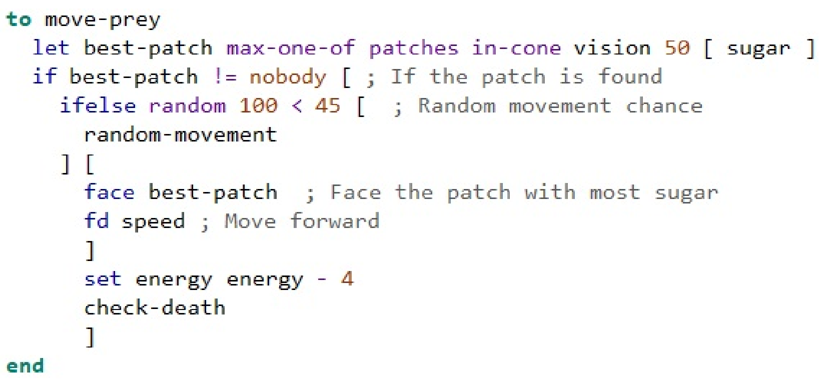
3.5.2 Sugarscape & Logistic Growth

This is the starting point of the simulation. A successfully implemented sugarscape model allows agents to explore the world, collect sugar that becomes their energy and survive. Firstly, we declare variables for sugar, re-growth, and when it was last consumed. Next, create a function that initialises sugar, such as colour, starting amount, re-growth amount and default time for when it was last consumed (default-0). This (and other setup) function is called from the main "setup" function, or alternatively it can be written inside of it (although performance might vary). Furthermore, the use of a slider with a global variable controls sugar density and distribution around the world. Finally, any black patches in the world are set to have 0 sugar.

Next, we implement prey agents that will feed on the created sugar patches. Define a new breed and call it prey. We repeat the process of declaring shared properties for all turtles and create "vision, energy and speed" variables. Thereafter, inside the main function we initialise the values for shared properties as well as specify the shape of an agent, also a slider is added for adjusting the initial prey population (these sliders can have constant values if the amount is fixed). Lastly, agents should spawn at random x and y positions each time the simulation is reset.

After initialisation, three new functions are created: prey-movement, prey-eat-sugar and prey-reproduction. The first function enables movement that prey will use for exploring the world. To make it more realistic, a vision variable is assigned with the value of 50 degrees that restricts prey vision and creates a vision-cone. By using this approach, the agent can only see patches in front and therefore, select the patch with maximum sugar. Once the agent knows where to go, movement function should roll the dice (a random number), which determines if the agent will go straight towards the sugar patch or will do a random move (left/right 45 degrees) instead. After each move, preys energy is subtracted by 5 and it is checked in case no energy left (none = death).

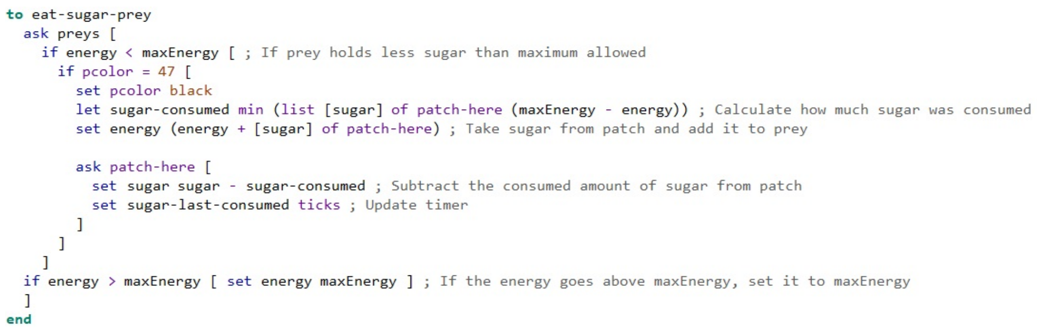
The second function allows sugar consumption and conversion into energy that prey uses to survive and reproduce. Create a new function that performs a check to see if preys current energy level is above the maximum allowed. If it is, set the current energy level down to maximum allowed, otherwise continue. Another check is performed to see if the patch that prey is standing on has a yellow colour (yellow - sugar). If so, that patch should change the colour to black, indicating that there is no sugar left. Then we find out how



```
to move-prey
  let best-patch max-one-of patches in-cone vision 50 [ sugar ]
  if best-patch != nobody [ ; If the patch is found
    ifelse random 100 < 45 [  ; Random movement chance
      random-movement
    ] [
      face best-patch  ; Face the patch with most sugar
      fd speed ; Move forward
      ]
      set energy energy - 4
      check-death
      ]
end
```

Figure 4: Prey Move code snippet

much sugar was consumed by calculating how much sugar was taken and how much a prey still needs to reach the maximum allowed; therefore, update prey energy. Finally, sugar is subtracted from the patch and assigned 0 value (no sugar). Update the timer that counts when this patch was consumed (for re-growth).



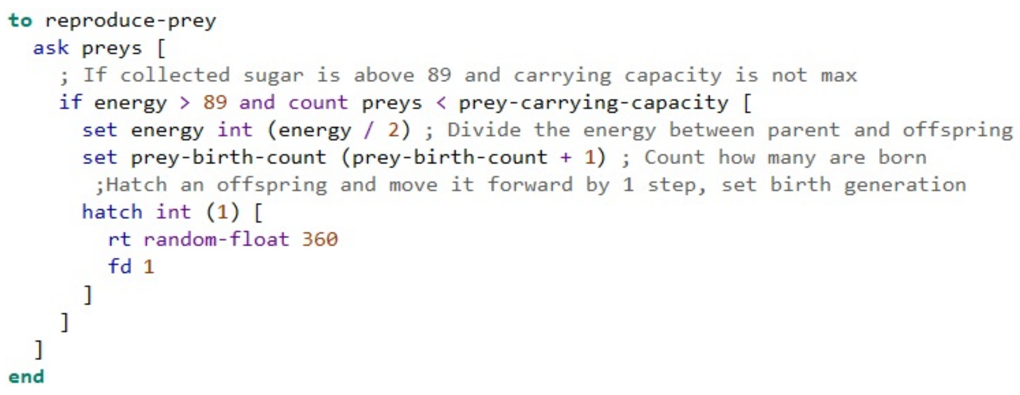
```
to eat-sugar-prey
  ask preys [
    if energy < maxEnergy [ ; If prey holds less sugar than maximum allowed
      if pcolor = 47 [
        set pcolor black
        let sugar-consumed min (list [sugar] of patch-here (maxEnergy - energy)) ; Calculate how much sugar was consumed
        set energy (energy + [sugar] of patch-here) ; Take sugar from patch and add it to prey

        ask patch-here [
          set sugar sugar - sugar-consumed ; Subtract the consumed amount of sugar from patch
          set sugar-last-consumed ticks ; Update timer
        ]
      ]
    ]
  if energy > maxEnergy [ set energy maxEnergy ] ; If the energy goes above maxEnergy, set it to maxEnergy
  ]
end
```

Figure 5: Sugar Consumption code snippet

The third function allows reproducing preys. Firstly, a check is performed to see if preys have enough energy to reproduce (89 in this case). We also check if preys have not reached their maximum population (carrying capacity). If both conditions are satisfied, an offspring can be hatched with half of the energy that a parent has. Lastly, birth count and parents energy is updated accordingly.

With this implementation, a sugarscape model is achieved where preys can freely move around and collect sugar while creating offsprings. In addition, we extended this model to include logistic growth that controls carrying



```
to reproduce-prey
  ask preys [
    ; If collected sugar is above 89 and carrying capacity is not max
    if energy > 89 and count preys < prey-carrying-capacity [
      set energy int (energy / 2) ; Divide the energy between parent and offspring
      set prey-birth-count (prey-birth-count + 1) ; Count how many are born
       ;Hatch an offspring and move it forward by 1 step, set birth generation
      hatch int (1) [
        rt random-float 360
        fd 1
      ]
    ]
  ]
end
```

Figure 6: Reproduction code snippet

capacity of the population.

3.5.3 Lotka Volterra

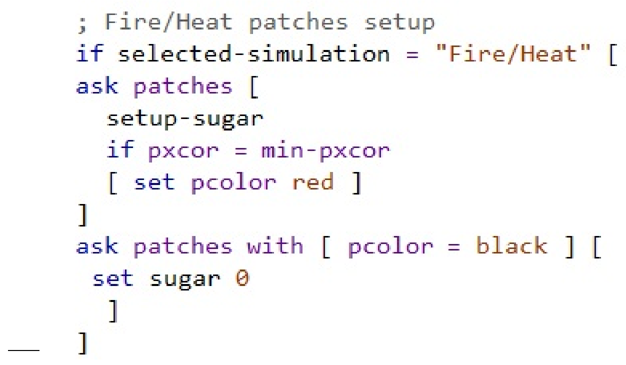
This part of implementation focuses on adding predator agents to the simulation and creating a more competitive environment. The setup and initialisation is the same as with prey agents. The only adjustments to make is to give it a different shape, colour and slightly increased speed variable (0.2). Predators and preys travel at the same speed and therefore, predators would never catch up with preys in a straight line, this is why we need a slightly different speed variable. This will be later enhanced with neural network.

Predator movement is the same, with a small adjustment to search behaviour, it should look for "one-of" preys and therefore, move towards it. The energy is reduced with each move.
Predator survival depends on prey population; they gather energy by killing prey and consuming its energy. A new function is created for hunting and consuming energy. A check is performed to see if any prey is on the same patch as predator. If so, the prey can be killed and its energy stored in a new variable. Finally, we check if the predator has less than maximum allowed energy and add the energy stored in a new variable to predators energy.

With LV model added to the simulation, the environment becomes more complex, competitive and hostile. These two models combined provide the base environment with agents; therefore, this will be further extended with environmental change and neural networks that will make agents learn how to adapt and survive.

3.5.4 Environmental Change

Fire/Extreme Heat. A few UI elements are required: sliders - for adjusting and randomising wind speed/direction and fire/heat probability, also a “chooser” that is used for selecting different simulations. Next, initialise sugar and create a vertical line of red patches that will be the starting point of fire movement. Fire will only spread on sugar patches, for instance, if sugar distribution were less than 15%, fire would not spread far.

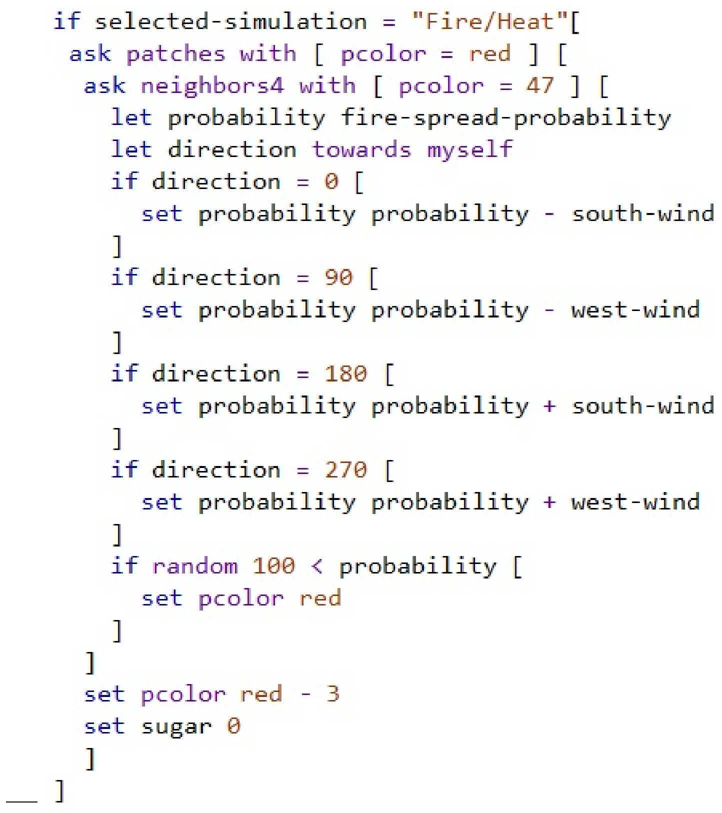


```
; Fire/Heat patches setup
if selected-simulation = "Fire/Heat" [
ask patches [
  setup-sugar
  if pxcor = min-pxcor
  [ set pcolor red ]
]
ask patches with [ pcolor = black ] [
 set sugar 0
  ]
]
```

Figure 7: Fire starting line

Inside the main function, we ask patches with red (fire) colour to look for neighbouring sugar patches. If the patches are available, perform a check on which direction the patch is facing (north, east, south and west). Furthermore, calculate the probability of fire spreading to those patches based on the following factors: initial spread probability, direction of the wind towards the current patch and influence of south/west wind direction. Roll a random number out of a 100 and if it is less than calculated probability, then the fire can spread to neighbouring patches. Finally, burnt patches have a brighter red colour and sugar level is set to zero.

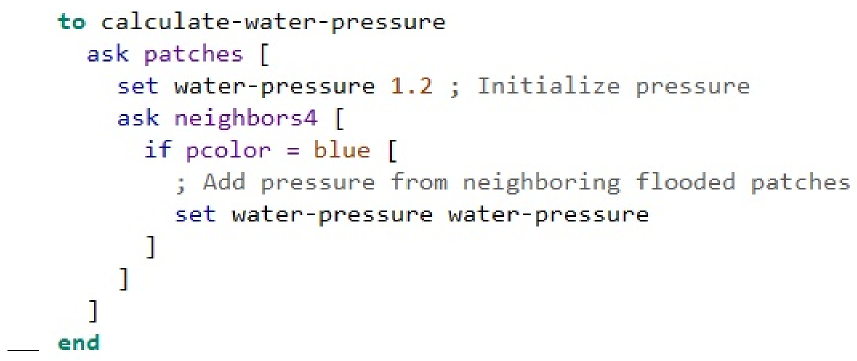
Flood/Water. A single UI element (a slider) is added to control the flood probability. The initialisation process is the same as with fire/wildfire, except the colour of a vertical patch line is changed to blue. The rest of the code is straightforward and simple. We ask neighbouring patches if it holds sugar and if so, roll a random number from 100 and if it is less than flood probability, these patches can turn blue. The flooding continues until there is no more water pressure coming from neighbouring (blue) patches. Each new blue patch will be assigned a fixed water pressure number (1.2 in this case). If all four neighbours are present, the centre patch will have a high pressure and flood will continue, otherwise there will be less pressure and the flood



```
if selected-simulation = "Fire/Heat"[
 ask patches with [ pcolor = red ] [
  ask neighbors4 with [ pcolor = 47 ] [
    let probability fire-spread-probability
    let direction towards myself
    if direction = 0 [
      set probability probability - south-wind
    ]
    if direction = 90 [
      set probability probability - west-wind
    ]
    if direction = 180 [
      set probability probability + south-wind
    ]
    if direction = 270 [
      set probability probability + west-wind
    ]
    if random 100 < probability [
      set pcolor red
    ]
  ]
  set pcolor red - 3
  set sugar 0
  ]
__ ]
```

Figure 8: Wind probability code snippet

will gradually come to a standstill. Lastly, a check should be performed to find any agents that are caught in the flooding; therefore, instantly kill them.



```
to calculate-water-pressure
  ask patches [
    set water-pressure 1.2 ; Initialize pressure
    ask neighbors4 [
      if pcolor = blue [
        ; Add pressure from neighboring flooded patches
        set water-pressure water-pressure
      ]
    ]
  ]
end
```

Figure 9: Water Pressure code snippet

By extending the simulation with these two environmental change scenarios, a real world encounters can be simulated. Furthermore, it creates a challenging environment for agents to live and survive. The next few sections will focus on neural network implementation that will evolve agents and make them smarter with each new generation.

3.5.5 Neural Network & Genetic Algorithm

A neural network (NN) is a computational model inspired by the biological neural networks of the human brain. It consists of neurons, which are organised into layers. The information flows from input layer (feed data in), to hidden layer (complex calculations), to output layer (decisions and predictions).

Genetic Algorithm (GA) is inspired by natural selection and genetics. It involves iteratively evolving a population of potential solutions (prey/predator) to a problem by applying evolutionary operators such as selection, crossover and mutation. Solutions are represented as chromosomes, which are encoded as string of values, which undergo genetic operations to produce offsprings with improved fitness.

By using genetic algorithm to evolve neural network parameters and structures within the simulation, the model can adapt and optimise predator and prey behaviours dynamically. Genetic algorithm enable the neural networks to learn effective strategies for both predators and prey by evolving their decision-making processes. This adaptive learning approach can lead to more realistic and sophisticated predator-prey interactions in the simulation. Neural network, with its ability to learn complex patterns and behaviours, can enhance the realism and intelligence of predator and prey agents. Furthermore, GA can optimize neural network configuration and weights based on the objectives, such as maximizing prey survival or predator hunting efficiency. By combining these two techniques, the predator and prey dynamics in the simulation can evolve over time, resulting in a more dynamic and engaging simulation environment that reflects real-world predator-prey relationships.

Neural Network

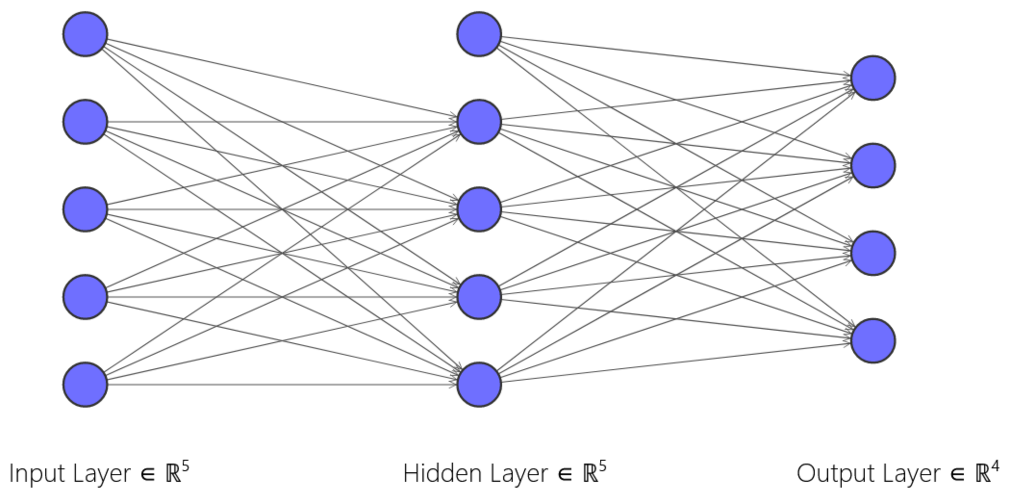
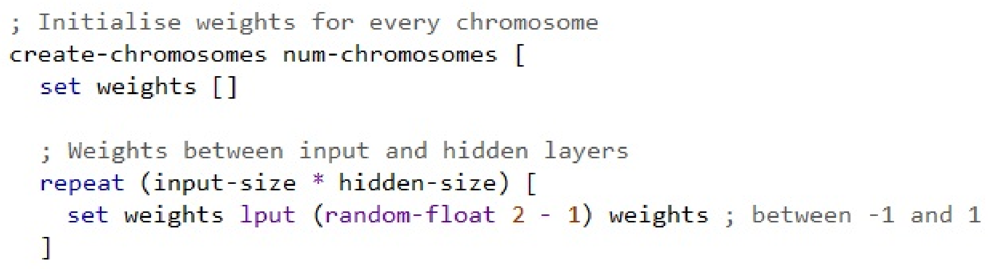


Figure 10: Predator/Prey Neural Network Diagram

The above diagram demonstrates neural network architecture for predator/prey agents in this project. The main functionality of this approach is for an agent to receive inputs, pass it on to hidden layer for calculations and therefore, produce the output that determines which moves the agent will take next. Each agent has its own neural network and receives inputs based on their action, location and vision.

The architecture has one input layer with five nodes. These nodes receive information such as, distance to predator or prey, distance to sugar patch, distance to environmental hazard, current energy level and movement speed. This is passed on to the hidden layer which has four nodes and a bias; therefore, calculates inputs with its weights and biases. Furthermore, after the calculation is finished, hidden layer will pass the outcome to the output layer, which has four nodes. Here, more calculations occur (weighted sum with biases). Finally, with the use of sigmoid activation function, we introduce non-linearity and output a number between zero and one. This decides if the final neuron should be activated or not; therefore, making an agent perform a new action or keep the same.

Initialisation of NN & GA
To create a neural network, we require a few global variables and a new breed. Global variables (input/hidden/output layers) define our neural network structure and will be used throughout the code. In addition, output values variable will hold the information once all the calculations are done in all layers. Lastly, a new breed of chromosomes is created and a shared property that holds chromosome weights. In addition, another shared property is created for preys and predators that will hold a personal chromosome, once it is assigned to each agent. We can do this action inside prey/predator setup function. Neural network will need access to its weights at a later stage. There is a possibility that a few agents will get assigned the same chromosome. We have chosen not to implement all unique chromosomes and this can be added to the future improvements.



```
; Initialise weights for every chromosome
create-chromosomes num-chromosomes [
  set weights []

  ; Weights between input and hidden layers
  repeat (input-size * hidden-size) [
    set weights lput (random-float 2 - 1) weights ; between -1 and 1
  ]
```

Figure 11: Weights Initialisation

Next, a new function is created to setup the neural network and it takes in the size of each layer as argument. Hidden layer is a list and other two layers are n-values. Inside the same function, we initialise weights for every chromosome. This is achieved by creating a user defined (slider) number of chromosomes and initialising a list of random float weights between -1 and 1. This operation is performed twice, one for input-hidden layer connection and another one for hidden-output layer connection. Finally, this function is called from the main setup function.

Input Layer
This is the starting point where raw data is fed into the neural network. In this simulation, we have predators and preys that are using similarly structured neural networks. Firstly, let's create a new function called "feed-forward" (one-way information flow: input - hidden - output) that takes in chromosome as an arguments. This function contains the main calculations for all layers.

For the prey agents: inside the function, two new variables are created that hold information about a single predator and a single sugar patch in the environment. With this, we can calculate the distance from current prey agent to "target". In order to do that, a new function is required, which does all the distance calculation. NetLogo has a simple keyword that calculates Euclidean distance between two objects; it is called "distance".

$$sqrt((x2 - x1)^2 + (y2 - y1)^2) \tag{7}$$

We pass the target object to the new function and simply report distance of it. In addition, to ensure that no error occurs when passing the target object to the function, enclose the code inside ifelse statement that checks if the target is nobody. If it is nobody then the function should report "nobody".

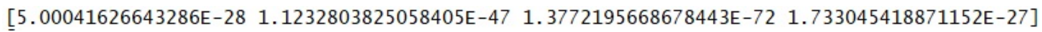
Back to "feedforward" function; create three new variables that will hold distance values for predator, sugar patch and water hazard patch. Furthermore, call the distance function; pass the predator, sugar and water turtles as a target. Next, create another variable and call it "total-inputs". This will hold all the values of calculated inputs inside a list. In this simulation, for prey agents, we are feeding in five inputs (distance to predator, distance to sugar patch, distance to water, energy and speed). The latter two do not need to be calculated. To ensure that the values are correct and stored in a list, create a debug statement that prints total-inputs variable, and it should show two float distance numbers and two integers for energy and speed.

Hidden Layer
In this layer, complex calculations occur. It contains a set of neurons that take a weighted sum of the values from input layer, add bias to it, and apply the activation function (sigmoid) to squash numbers between 0 and 1. This way, we introduce non-linearity and allow the neural network to learn complex patterns in the data.

Firstly, we have to create a few variables that will be used in this layer. Weighted sum will hold the calculated value of each input, access weights

variable should hold information of the weights in the chromosome, a bias variable that has a random value (-1 to 1) and lastly, two counters for incrementing inner and outer loops. Next, create two foreach loops, with the outer loop being hidden-layer and the inner loop being total-inputs. This is where the main calculation occurs.
Inner Loop: a new variable is created that holds index of a current weight used in calculation. Next, we calculate the weighted sum by adding input-value to weight value at index zero; therefore, store this in the already created weighted sum variable. Increment inner loop.
Outer Loop: we use already calculated weighted sum value and add bias to it. Furthermore, this new total sum is passed to sigmoid activation function and the output stored in a new variable. Finally, update hidden-layer values by replacing its current item with the final value, and increment outer loop counter.
This procedure is repeated a number of times, until all values are calculated. In order to make sure that the values are calculated and placed in the hidden layer, we can use "print" primitive to print the values of a hidden layer to the console.

—



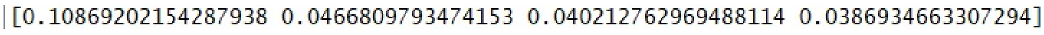
```
[5.00041626643286E-28 1.1232803825058405E-47 1.3772195668678443E-72 1.733045418871152E-27]
```

Figure 12: Hidden layer large values

Output Layer
This is the final layer in the neural network structure; therefore, it produces the final output values that are used in other functions (such as movement in this simulation).

The output layer is calculated exactly the same way as the hidden layer; however, with few minor changes. We have to make sure that variables used are not the same, to prevent overriding variables in the hidden layer and causing errors. Simply add the word "output" for every variable; therefore, this will give a clear indication that it is used in the output layer.
Finally, hidden layer will produce massive numbers that are very long and most likely above one (but no negative values). This is why we use sigmoid activation function again in the output layer; therefore, squashing massive numbers into a correct value that is between zero and one. This neural

—



```
[0.10869202154287938 0.0466809793474153 0.040212762969488114 0.0386934663307294]
```

Figure 13: Final output layer values - 4 inputs

network process should be repeated for predator agents as well. The only changes are input layer size (4), and instead of feeding in predator location, we feed in preys location.

Turtle movement
To make the turtles move, we use calculated outputs that influence certain behaviours for both preys and predators. Firstly, create a movement function for prey and store calculated output values into a new variable. Next, use "item" primitive in NetLogo to access a list with outputs and assign each index to a new variable that will represent: left turn, right turn, acceleration, deceleration (4 outputs). Furthermore, create another two new variables and store calculated distance to any predator and one of predators that are within preys vision cone.

Next, we create a simple evasive manouver that prey will use to evade any predator once it is inside the vision cone. First, use "towards" primitive to check where exactly a predator is. Then the prey can make a left or right turn and add 180 degrees to it, making it spin around and moving away from the enemy. In addition, increase the speed variable for up to 1.5 times. Each move should cost two energy; however, this could be slightly increased. Finally, in the end we perform a death check to see if the prey has more than 0 energy left. This movement could potentially cause an issue with the neural network controlled output and therefore, should be adjusted accordingly or removed if there is an issue.

Another movement command uses the outputs from neural network. We create an if else statement and use earlier created variables for the output values. At first, a check is performed to see if any predator is close to prey (using vision cone). If so, then we check if left turn output value is more than right turn output value, and call right turn. If its the other way around, we call a left turn. Ultimately, this indicates that if predator is danger close, prey should make a quick move to any side depending on output values. Next, a regular movement can occur. If no danger in sight, use random left or right move. In addition, add another two if statements to check if acceleration/deceleration output values are higher or lower than the other. This way we can make the prey move forward and increase or decrease it's speed. Finally, every turn cost two energy, but this can be adjusted depending on the simulation.

Conclusion
To conclude this section, we have successfully create a neural network that

is used by prey and predator agents. It takes in various inputs, calculates its values in the hidden layer and outputs final values that are used to move the agent and evade predator/kill prey. Furthermore, after adding genetic algorithm (next section), the offsprings neural network should improve with each generation. Lastly, a lot of neural network calculations can be computationally expensive for NetLogo software and therefore, slow it down. In case the simulation runs slow, the number of turtles and carrying capacity should be lowered.

Genetic Algorithm

—

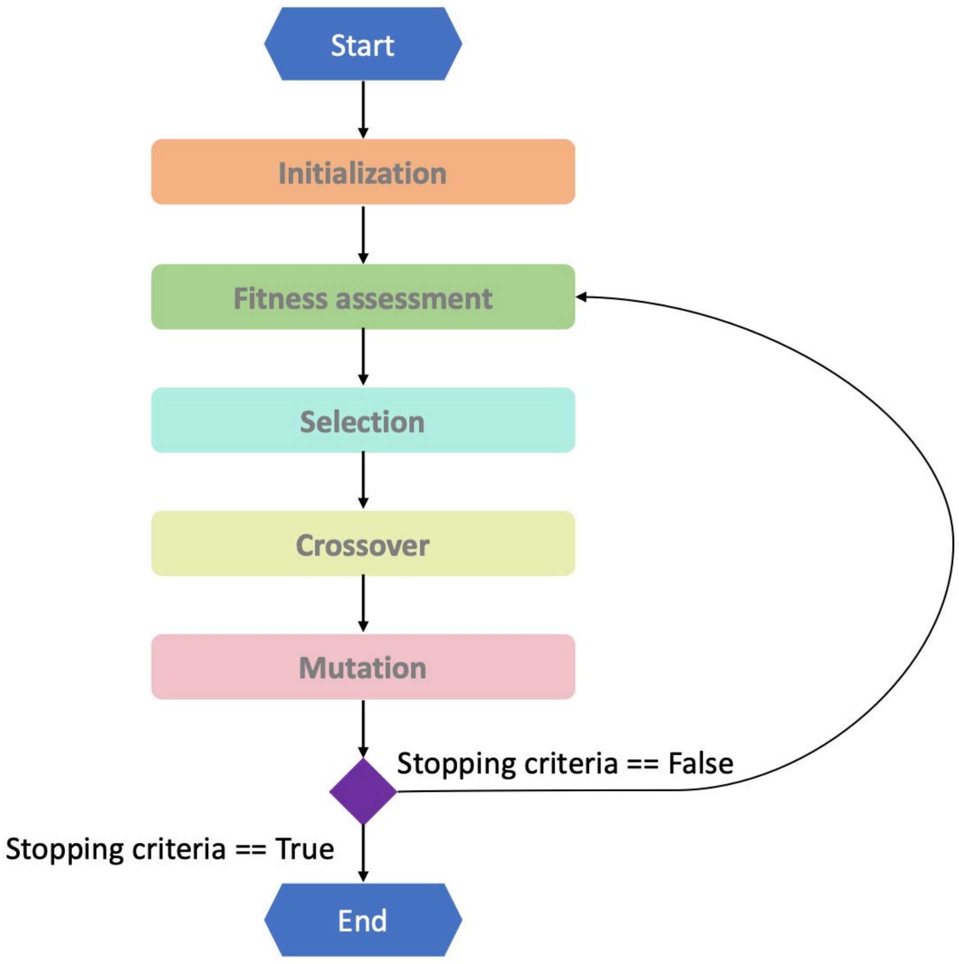


Figure 14: How GA works

Fitness

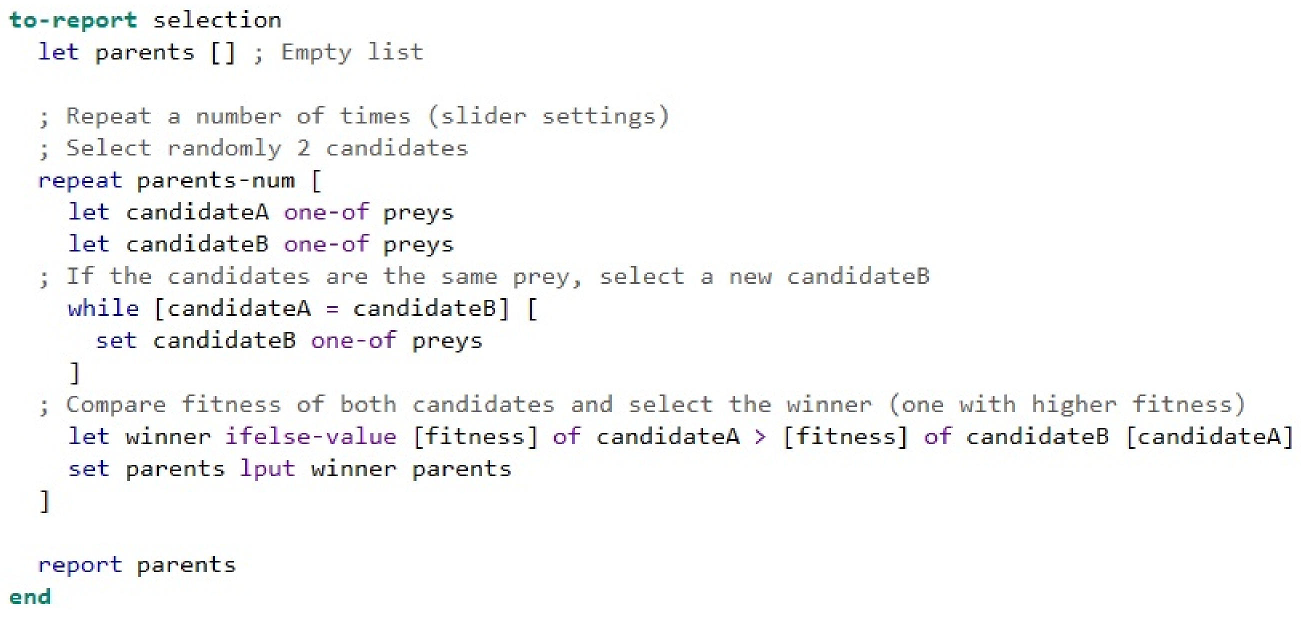
Fitness measures the performance of individuals, and it uses various factors that are calculated in the end and overall score is presented. In this simulation, we have preys and predators that will have a fitness function. Firstly, we create a new function to calculate fitness that takes in “current tick” as an argument. Current tick represents time that a turtle survived. Then, use an if else statement to check if current tick is zero. Because on start, the first generation will start with zero ticks, it cannot be divided; therefore, we set fitness to current energy level. Next, for other generations, we calculate current energy divided by current tick value. This value is then assigned to final fitness variable. In addition, the fitness value will likely be a decimal and it can be rounded if needed.

With this simple function, turtles fitness is calculated and it can be checked inside turtle inspection window. We chose only two factors for calculation; however, it could use other factors too. Such as, distance to predator/prey, speed and more if needed.

Selection

Selection determines which individuals from the current generation will take part in sharing their chromosomes towards the next generation. Agents with higher fitness scores have a higher probability of being selected. There are a few selection methods that help to choose the right candidates. Roulette wheel, Tournament selection and Rank Selection. Each method has its own way of finding best candidates. For this simulation, a tournament selection is used, which randomly selects two agents and compares their fitness; therefore, the winner moves on to crossover phase.

This is achieved by initialising an empty list where winners (parents) will be stored. Each tournament has two randomly selected agents. We create a global variable (a slider) that is used to specify how many times a loop should “repeat” (NetLogo in-built primitive). Inside the repeat loop, two random agents are taken and assigned to candidateA, candidateB variables. In order to have two different candidates, perform a check to validate if the selected candidates are not the same, and if it is, then select a new candidateB. Furthermore, by using an ifelse statement, their fitness is compared and whichever has a higher fitness value, becomes the winner. This winner is then put in a list of winners. By completing these steps, a list of parents is reported and used in the following sections, where crossover and mutation takes place.



```
to-report selection
  let parents [] ; Empty list

  ; Repeat a number of times (slider settings)
  ; Select randomly 2 candidates
  repeat parents-num [
    let candidateA one-of preys
    let candidateB one-of preys
  ; If the candidates are the same prey, select a new candidateB
    while [candidateA = candidateB] [
      set candidateB one-of preys
    ]
  ; Compare fitness of both candidates and select the winner (one with higher fitness)
    let winner ifelse-value [fitness] of candidateA > [fitness] of candidateB [candidateA]
    set parents lput winner parents
  ]

  report parents
end
```

Figure 15: Tournament Selection

Crossover

Through crossover, the fittest chromosomes are combined and a new chromosome created. It takes two chromosomes from two parents (from selection), randomly splits the first chromosome and then attaches a second half of the second chromosome. There are three approaches to this. One point crossover is when we cut a random point in the first parents chromosome and then attach it to the other half of second parents chromosome. Then there is a two point crossover that makes cuts at two random places in the chromosome, therefore, only the middle part is swapped. Finally, we return newly created chromosomes in a list Lastly, a uniform crossover swaps genes based on chance (usually 50%). By using one of these approaches, we create a brand new chromosome that offpsrings will use.

In this simulation, we have used a one point crossover to create chromosomes for two offsprings. Our function takes in two parents as arguments and takes their chromosomes. Then, using the same logic as before, we access chromosome weights and store it in a new variable. Next, we create two lists where new chromosomes will be stored after crossover. Create a new variable and store a random cut point in the first parents chromosome using "random length" primitive in NetLogo. Now we can use this cut point to combine both chromosomes. The first offspring will use a chromosome combination

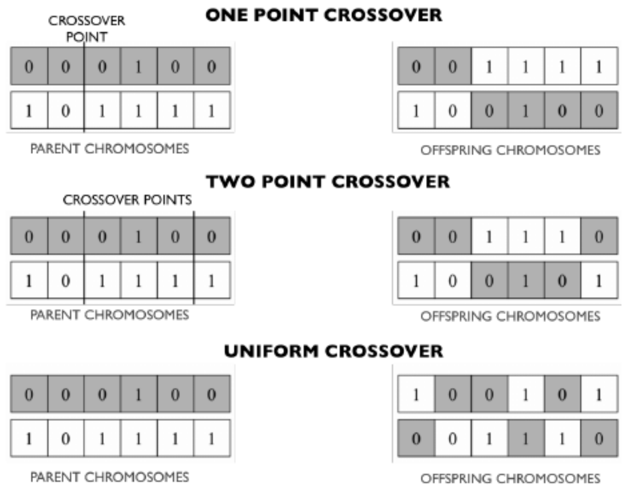
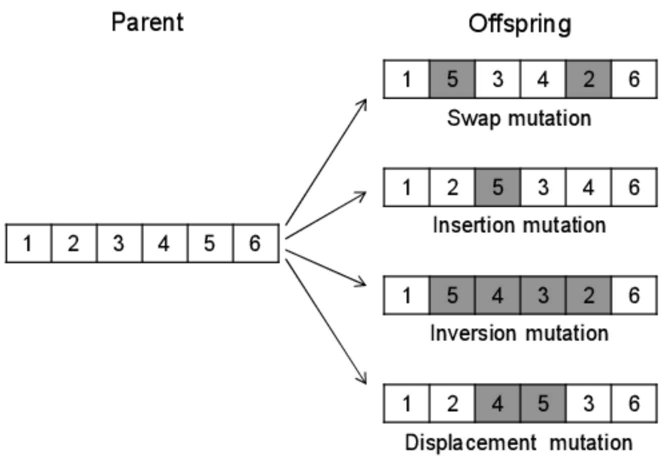


Figure 16: Crossover Methods

from parent one (start to cut point) and from parent two (cut point to end). Also, the second offspring will do the same but in the opposite way (cut point to end plus start to cut point). This way we create two chromosomes that has a combination of both parents. Finally, we report both offsprings that will be used for mutation.

Mutation
Mutation is an operator that increases diversity in the population. It also helps to avoid being stuck in local optima (sub-optimal solutions). There are a few approaches to mutation such as, bit flipping, insertion, gaussian, inversion and more.

In this simulation, we have chromosome weights that should be mutated; therefore, we use a swap method (bit flipping or weight flipping in this case). Create a local variable that will represent an index number in the chromosome and a global variable for mutation rate. Next, loop through the chromosome and roll a random float number from 0 to 1. If this number is less than mutation rate, then we can flip the selected number. Each weight value has an index assigned and so it can be flipped based on probability. When flipping, we use "replace-item" primitive in NetLogo and select the current index number. At this index, the weight should be replaced with a random float number from -1 to 1. This way we create diversity and introduce possibly new behaviours in the simulation. Lastly, we increment the index counter each time and in the end report the final chromosome.



—

Figure 17: Approaches to Mutation

Evolution
In this function, we execute all genetic algorithm functions in order and call it inside our "go" function. Firstly, call selection function and pass prey and predator turtles to it. Store it in separate variables. Next, we use "item" primitive to get the parent one and parent two from selected winners and store them in separate variables. We do this for both prey and predator and so there should be four new variables with four parents, two from each breed. This is done because we need to pass individual parents to crossover function.

Call the crossover function and pass both parents from prey and then from predator to it. This way, we will perform a one point crossover and return a new list with crossed over weights for one offspring. Furthermore, we call mutate function and pass our newly created chromosome (or list of weights). Do this for both preys and predators. After this, we should have a chromosome that can be assigned to an offspring.

Finally, we hatch one prey and one predator offspring, assign the new chromosome and set any other settings required (speed, vision, colour etc.). To finish the function, kill off the old generation of turtles, so it was replaced by a better version offsprings.

Conclusion
In conclusion, we have implemented a working genetic algorithm that further extends neural network by adding various operators that help agents to evolve. We have used fitness operator to calculate how "fit for purpose" are

turtles and therefore, used tournament selection operator to create small size competitions between agents, with the winner (the one with higher fitness) progressing to the next stage. Then we have applied crossover and mutation operators to create a new chromosome and make it more diverse. Finally, every new generation would hatch a new offspring and place it in general population.

Each new generation should now be more diverse and produce interesting results in how they manage to survive the world that has multiple threats such as, predators, flooding and wildfire destroying sugar patches. Lastly, our simulation is now complete and ready for testing.

3.6 Conclusion

In this chapter, we have described the main details of the approach taken to accomplish this project. We introduced NetLogo as the main development environment and the main programming language. Furthermore, we have covered some theory about agent based modelling and its components.

A detailed description of the structure of this project helped us to understand how to design user interface and use it through programming language. Next, we have implemented the base of the simulation, with a combination of population models such as, Lotka Voltera, Logistic Growth and Sugarscape. By providing formulas used in these models, we further improved our understanding in how this approach works together and can introduce interesting patterns and behaviours (although they are random, but useful for fundamental understanding).

Furthermore, we extended the simulation by adding environmental change effect (wildfire and flooding). This lowered the survival rate of our turtles; therefore, making them more vulnerable not only to enemies but the environment as well.

In order to increase learning and survival rate, we have introduced a neural network which takes in specific inputs and calculates specific outputs that are then used to control the movement of a turtle. We further extended neural network by adding genetic algorithm, which introduced the creation of smarter agents, by calculating their current fitness and selecting the fittest individuals to become parents. Therefore, combining their chromosomes to create new offsprings that go through mutation and introduce diversity in general population. Finally, with the new generation being alive, we kill the old generation, making sure that only the fittest turtles are present in the simulation.

4 Testing

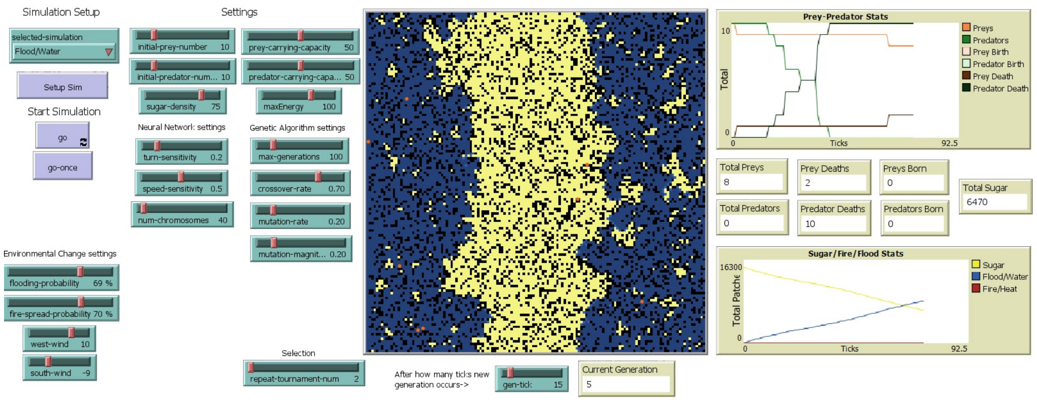
Multiple tests were carried out in order to gather the results of predator/prey behaviour as well as analyse different scenarios and environmental change impact on agents. For testing, we created four different simulations that would prompt any interesting/different behaviours from predators and preys. The list of simulations is as follows:

- Test Simulation #1 - Full. This is a full simulation with all agents and environmental change.
- Test Simulation #2 - No environmental change. Only preys and predators.
- Test Simulation #3 - Only prey. No predators or environmental change.
- Test Simulation #4 - No neural network or genetic algorithm. Plain simulation with preys and predators only.

The following will describe what was observed in each simulation and what behaviours occurred or any unusual activity. Then, we will compare how these simulations are different from each other and write an analysis

4.1 Test Results

Full Simulation



selected-simulation: Flood/Water
initial-prey-number 10
initial-predator-num... 10
sugar-density 75
prey-carrying-capacity 50
predator-carrying-capa... 50
maxEnergy 100
turn-sensitivity 0.2
speed-sensitivity 0.5
num-chromosomes 40
max-generations 100
crossover-rate 0.70
mutation-rate 0.20
mutation-magnit... 0.20
flooding-probability 69 %
fire-spread-probability 70 %
west-wind 10
south-wind -9
repeat-tournament-num 2
After how many ticks new generation occurs-> gen-tick 15
Current Generation 5
Total Preys 8
Prey Deaths 2
Preys Born 0
Total Predators 0
Predator Deaths 10
Predators Born 0
Total Sugar 6470

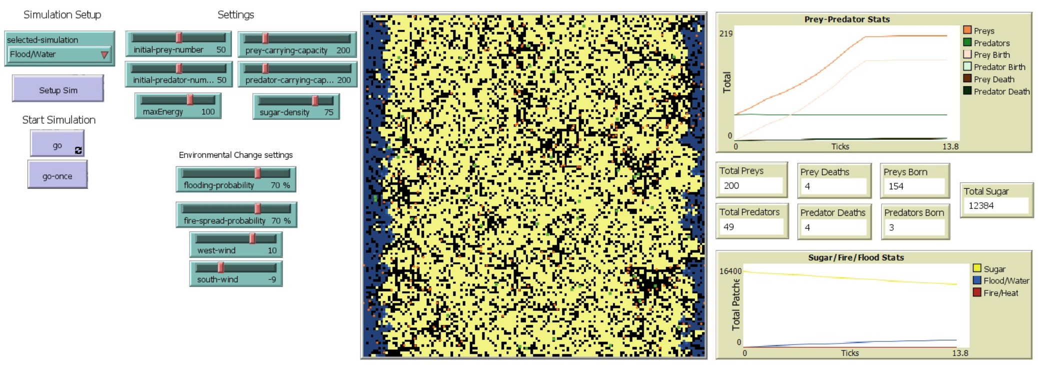
Figure 18: Full Simulation

This is the main simulation that has all the components such as, predators/preys and environmental change. Other tested simulations had similar results. This was tested multiple times and there was no significant change in behaviour in the first few generations. With each new generation, it was expected for offpsrings to develop better senses and therefore, become smarter. The agents were still walking into water and fire patches (it should learn not to). However, the testing was not done thouroughly because of NetLogo performance issues. It is assumed that further generations will adapt better behaviour, and learn how to avoid dangerous areas by using its vision.

The simulations with neural network and genetic algorithm were not observed thouroughly and therefore, no valuable data was gathered. Although, the final simulation below, was thouroughly tested and valuable data gathered.

No Neural Network or Genetic Algorithm Simulation

—


Simulation Setup
selected-simulation: Flood/Water
Start Simulation

Settings
initial-prey-number 50
prey-carrying-capacity 200
initial-predator-num... 50
predator-carrying-cap... 200
maxEnergy 100
sugar-density 75

Environmental Change settings
flooding-probability 70 %
fire-spread-probability 70 %
west-wind 10
south-wind -9

Total Preys 200
Prey Deaths 4
Preys Born 154
Total Predators 49
Predator Deaths 4
Predators Born 3
Total Sugar 12384

Figure 19: No Neural Network or Genetic Algorithm Simulation

In this simulation, we have observed how prey and predator are behaving in the environment with two environmental changes - wildfire and flooding. The simulation was tested ten times using different settings. These settings varied from population of 20 to 100 and fire spread/flooding percentage adjusted between 30% and 80%. Carrying capacity was fixed at 200. In both scenarios, the environmental change was quite high (above 70%) and the fire/flood should be spreading all over the world; however, in each of the tests, prey would prevent it from spreading by eating sugar and this way cutting off fire/flood spread probability (because it only spreads on sugar patches).
This is an interesting approach to stopping the environmental hazard from spreading; however, the simulation does not have any neural network implemented and therefore, this behaviour is very random.

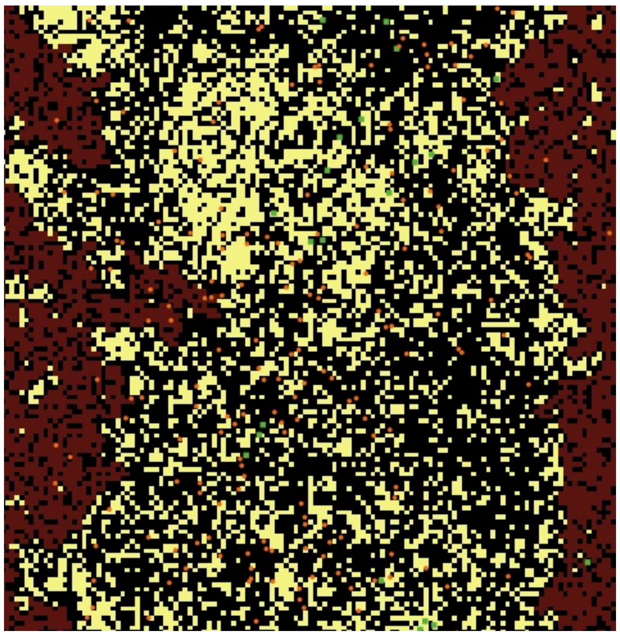
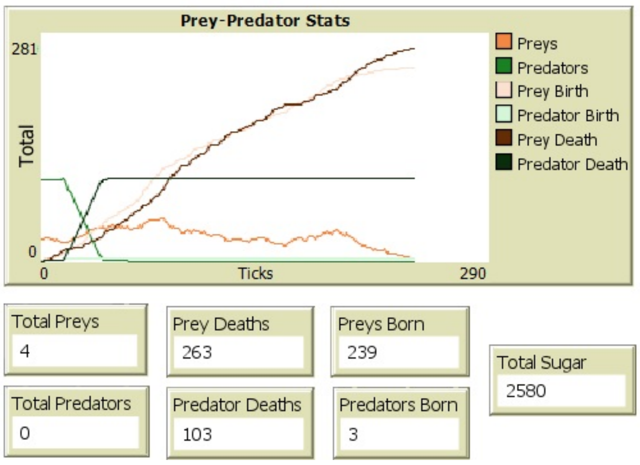


Figure 20: Empty sugar patches prevent fire from spreading

After observing test runs with various population number, it could be seen no big difference in predator/prey death/birth rate. If there was 100 preys and 20 predators, we could see predator birth rate slightly increase; however, they all die eventually. And if we have 100 predators and 20 preys, predators would die out faster, but prey would still reproduce fast enough and top the carrying capacity quickly. It appears that no matter what prey population size is, if there is plenty of sugar at the start, they will reproduce fast while predators will struggle.

Further tests revealed that if we want to reduce fast prey birth rate, sugar density in the world should be reduced. We tested the world with only 18% sugar density, which means the world will have little food available. In this case, we observed prey still reproducing; however, average prey population at a time was 34 for some period of time.

In conclusion, we can determine that this random behaviour can lead to interesting discoveries, which can be used in further implementations and extensions. However, this simulation shows predators never managing to successfully stay alive and they die out eventually, whereas prey can maintain stable population size and reproduce fast. Finally, environmental change (wildfire/flood) did not affect the population too much, mainly because prey



Total Preys: 4
Prey Deaths: 263
Preys Born: 239
Total Sugar: 2580
Total Predators: 0
Predator Deaths: 103
Predators Born: 3

Figure 21: Observation with low sugar density

would eat sugar in places where fire/flood would spread, this way maintaining further spread.

5 Conclusion

5.1 Discussion of Results

The results gathered show potential of using evolutionary algorithms and neural network together in order to create a more dynamic and adaptive NPCs. However, more testing is required in order to understand how computationally expensive this can be. We have used NetLogo simulation environment in this project and there could be seen a significant difference in performance with NN and GA and without it. The simulation runs smoothly without any issues and we observed agents performing random actions that has no meaningful output. On the other hand, a simulation with NN and GA would be more heavy on NetLogo performance as it majorly slowed down; however, it still could be seen from few generations that “learning” agents inherit their parents chromosome and increase their fitness value which leads to more interesting behaviour.

5.2 Fulfilments of Aims

The final outcome of the project was successful and we have created a simulation with prey and predator agents that can move around and explore the environment. Furthermore, we extended the simulation by implementing a neural network and genetic algorithm, which resulted in agents learning and evolving, with each new generation being a better version of its predecessors.

Through research, we have identified evolutionary algorithms that would work best in answering our research question. NEAT, rtNEAT and genetic algorithm was selected for further investigation and possible implementation. With NetLogo, it was possible to implement a 3rd party machine learning framework that use NEAT and hyperNEAT; however, the final decision was to use genetic algorithm and recreate a similar behaviour that NEAT/rt-NEAT would have.

Our second aim was to implement population models that would simulate a living environment. Through research, we have idetified three models that would create prey/predator agents and make them behave accordingly. The three models are: Lotka Volterra - enables a competitive environment, Sugarscape - creates prey that survives by consuming sugar or spice, Logistic Growth - enables carrying capacity. All these models were implemented and tested to make sure the algorithm works as expected. This built the foundation of the simulation.

Next aim was to implement an environmental change scenario that would pose threat to living organisms. There was a bunch of options to choose from; however, after further research, it was clear that wildfire and flooding is one of the most common scenarios in video game world. Therefore, we added these to the simulation, making it harder for prey/predator agents to navigate and survive by destroying sugar patches.

Our fourth aim was to implement a neural network. We have accomplished this successfully within NetLogo environment, making agents have multiple NN layers and weights that would be evolved at a later stage. A neural network was used because we want to feed in data as inputs to prey/predator agents and therefore, it should make complex calculations that would determine the next move in the output layer. This would result in creating smarter agents that introduce new behaviours and solutions.

Another aim was to find an evolutionary algorithm that could evolve neural network. Through research, we have identified genetic algorithm and therefore, chose this for implementation. It uses fitness calculation, tournament selection, crossover and mutation. With these operators, we were able to evolve neural network weights, create new chromosomes and increase the diversity of the population. This ensures that each generation has offsprings that are fitter and smarter; therefore, produce new information about their behaviour and how well they can adapt to challenging environments.

Our final aim was to experiment and run tests on multiple simulations while observing the behaviour. Four simulations were created that had different settings. One with no GA or NN, another with no environmental change, third with no predators and final, full simulation with everything. Tests were executed multiple times and all simulations compared in the end. The individual test results allowed us to improve our understanding in the behaviour of prey and predator species, while the overall comparison showed us how their behaviour differs in different scenarios. By running these tests and experiments, we gained a lot of new information that can be used in identifying future improvements.

In conclusion, all the aims in section 1.3 were accomplished; therefore, helping to answer our research question and making this project a success.

5.3 Future Work

When looking into the future of this project, a number of improvements should be considered. Firstly, NetLogo software requires further investigation in order to understand how to create faster simulations. Currently, the simulation with neural network and genetic algorithm are causing performance issues and it can't handle large amounts of agents being active at the same time. This prevents us from doing in-depth testing and therefore, gather better statistics that could be compared to other simulations.

Furthermore, improvements should be considered when designing neural network. A more efficient architecture could be introduced in order to reduce the time required to calculate hidden and output layers. In addition, the use of output values for movement should be reconsidered, possibly introducing a more precise way to use it when making decisions that will affect prey/predator behaviour. For instance, adding more movement based on agents location (currently prey is only affected by the presence of a predator and a blue patch - water). Also introducing combined movement such as, turn and accelerate (currently all four movements are separate).

Next, genetic algorithm approach should be considered from multiple angles. For instance, it would benefit from experimenting with different operator methods such as, selection might produce different results if it was rank based or roulette wheel based. In addition, crossover could produce better chromosomes if uniform approach was used instead of one point crossover. By experimenting with these operators, we could potentially discover a more diverse generation that would lead to new discoveries in how agents approach dangers in the environment.

Finally, environmental change scenarios should affect prey/predator behaviour even further in order to get more interesting and unexpected results, possible new discoveries. Currently, we can choose one environmental change to initialise; however, it would be useful to experiment with multiple environmental hazards in a single simulation. By taking this approach, new behaviours and new solutions could emerge that would help answer our research question in more depth.

We have identified and discussed four improvements that would make this simulation more interesting and efficient with a possibility of discovering new behaviours that would enhance our experiment results. Moreover, the most important improvement is to make this simulation faster, where we could use large number of agents for experimenting.

5.4 Critical Evaluation of the Project

A few different approaches were considered when choosing which software to use in this project. The selected NetLogo software proved to be the right choice, which provided the necessary development environment as well as tools and easy to learn programming language. Furthermore, the use of neural network and genetic algorithm appeared to be the right decision in creating a living and learning agents. The implementation process was challenging and a few obstacles occurred along the way. Moreover, while it took some time, in the end, the obstacles were overcome and the implementation was successfully finished.

By using Trello board, Gantt chart and a diary, it was possible to track the progress of this project fast and easy, follow weekly deadlines and make notes in the diary that were used when something was not clear enough. These approaches proved to be an effective way to manage this project. Various experiments were carried out in the simulation, which provided us with better understanding about the behaviour of the agents and how neural network and genetic algorithm is beneficial when it comes to video games. In addition, analysing test results from four different simulations provided insights into understanding the algorithm better and what improvements are needed.

Overall, the project went well and we managed to stay on track with all major deadlines. The first half of the project was all about research and a lot of new information was learned that helped to accomplish implementation at a later stage. The second half of the project was more challenging with a few obstacles along the way; however, we managed to solve the issues without delaying any major deadlines. There were some changes in the project during the research and implementation phase; however, it did not have any major impact on deadlines or project itself.

References

An, G. (2017). Optimization and control of agent-based models in biology: A perspective. *Bulletin of Mathematical Biology*, *79*, 63-87.

Anisiu, M.-C. (2014). Lotka, volterra and their model. *Communications in Nonlinear Science and Numerical Simulation*, *32*, 9-17.

Bastille-Rousseau, G. (2018). Climate change can alter predator–prey dynamics and population viability of prey. *Oecologia*, 141-150.

Bedau, M. A. (2003). Artificial life: organization, adaptation and complexity from the bottom up. *Trends in Cognitive Sciences*, *7*(11), 505-512.

Belk, R., Humayun, M., & Gopaldas, A. (2020). Artificial life. *Journal of Macromarketing*, *40*, 221-336.

Cardamon, L. (2010). Learning to drive in the open racing car simulator using online neuroevolution. *IEEE Transactions on computational intelligence and AI in games*, *2*(3), 176-190.

Chan, L., Hogaboam, L., & Cao, R. (2022). Artificial intelligence in video games and esports. , 335-352.

Chen, J., Gongjin, M., Prof, L., Eiben, G., & Visser, A. (2019). The predator-prey evolutionary robots system:from simulation to real world.

Cherniha, R. (2022). Construction and application of exact solutions of the diffusive lotka-volterra system: A review and new results. *Communications in Nonlinear Science and Numerical Simulation*, *113*.

Chigot, E., & Wilson, D. G. (2022). Coevolution of neural networks for agents and environments. *GECCO 2022 Companion - Proceedings of the 2022 Genetic and Evolutionary Computation Conference*, 2306-2309.

Dhiman, D. B. (2023). Games as tools for social change communication: A critical review. *Global Media Journal*.

Dziuk, A. (2011). Creating intelligent agents through shaping of coevolution. , 1077-1083.

Eldahshan, K. A., Farouk, H., & Mofreh, E. (2022). Deep reinforcement learning based video games: A review. *MIUCC*.

Elfeky, E. Z., Elsayed, S., Marsh, L., Essam, D., Cochrane, M., Sims, B., & Sarker, R. (2021). A systematic review of coevolution in real-time strategy games. *IEEE Access*, *9*, 136647-136665.

Gluck, E., & Wightman, E. (2015). Simant simulation using neat.

Ha, D. (2019). Reinforcement learning for improving agent design. *Artificial Life*, *25*(4), 352-365.

Hardy, J. T. (2003). Climate change: Causes, effects, and solutions.

Ibrahim, M. Y., Sridhar, R., Geetha, T. V., & Deepika, S. S. (2019). Advances in neuroevolution through augmenting topologies - a case study. *Proceedings of the 11th International Conference on Advanced Computing, ICoAC 2019*, 111-116.

Jakobsson, J. (n.d.). *Rain world.* Retrieved 2023, from `https://rainworldgame.com/`

Justesen, N., Bontrager, P., Togelius, J., & Risi, S. (2020). Deep learning for video game playing. *IEEE Transactions on Games*, *12*(1), 1-20.

Kehoe, J. (2016). The specification of sugarscape.

Kucharavy, D. (2015). Application of logistic growth curve. *Procedia Engineering*, *131*, 280-290.

Langton, C. (2018). Artificial life: Proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems. , *6*.

Laws, A. N. (2017). Climate change effects on predator–prey interactions. *Current Opinion in Insect Science*, *23*, 28-34.

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature 2015 521:7553*, *521*(7553), 436-444.

Lehman, J., & Miikkulainen, R. (2013). Neuroevolution. *Scholarpedia*, *8*(6).

Liang, J., Shahrzad, H., & Miikkulainen, R. (2023). Asynchronous evolution of deep neural network architectures.

Macal, C. M. (n.d.). Agent-based modeling and simulation. *Proceedings of the 2009 Winter Simulation Conference.*

Matsuo, Y., LeCun, Y., Sahani, M., Precup, D., Silver, D., Sugiyama, M., . . . Morimoto, J. (2022). Deep learning, reinforcement learning, and world models. *Neural Networks*, *152*, 267-275.

Miikkulainen, R. (2006). Creating intelligent agents in games. *Frontiers of Engineering*.

Mohabeer, H. (2013). Improving the performance of neat related algorithm via complexity reduction in search space.

Mohammad. (2022). Introduction to evolutionary algorithms: Genetic algorithm, neuro-evolution.

Mohanty, H. (2023). Digital life: An advent of transhumanism. , 1-24.

Moshayedi, A. J., Roy, A. S., Kolahdooz, A., & Shuxin, Y. (2022). Deep learning application pros and cons over algorithm.

Nolfi, S. (1998). Coevolving predator and prey robots: Do arms races aruse in artificial evolution? *Artificial Life*, *4*(4), 311-335.

Olesen, J. K., Yannakakis, G. N., & Hallam, J. (2008). Real-time challenge balance in an rts game using rtneat. *IEEE Symposium on Computational Intelligence and Games*, 87-94.

Pénisson, S., Sniegowski, P. D., Colato, A., Stanovov, V., Akhmedova, S., & Semenkin, E. (2020). Neuroevolution of augmented topologies with difference-based mutation lineage dynamics and mutation-selection balance in non-adapting asexual populations neuroevolution of augmented topologies with difference-based mutation.

Rahman, A. (2007). An analysis to wealth distribution based on sugarscape model in an artificial society. , *20*(3).

Raihan, A. (2023). A review of the global climate change impacts, adaptation strategies, and mitigation options in the socio-economic and environmental sectors. *Journal of Environmental Science and Economics*, *2*(3), 36-58.

Reddemann, K. (2015). Evolving neural networks in npcs in video games.

Risi, S., & Togelius, J. (2015). Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, *9*(1), 25-41.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85-117.

Schrum, J. (2008). Constructing complex npc behavior via multi-objective neuroevolution. , 108-113.

Stanley, K. O. (2002). Evolving neural networks through augmenting topologies. *The MIT Press Journals*, *100*(2).

Stanley, K. O., Bryant, B. D., & Miikkulainen, R. (2005). Real-time neuroevolution in the nero video game. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, *9*(6).

Sun, Q. (2015). Artificial neural network using the training set of dts for pacman game. *IEEE Xplore*.

Tisue, S. (2004). Netlogo: A simple environment for modeling complexity. *International conference on complex systems*.

Triambak, S. (2021). A new logistic growth model applied to covid-19 fatality data. *Epidemics*, *37*, 1755-4365.

Tsoularis, A. (2002). Analysis of logistic growth models. *Mathematical Biosciences*, *179*(1), 21-55.

Yamada, J., Shawe-Taylor, J., & Fountas, Z. (2020). Evolution of a complex predator-prey ecosystem on large-scale multi-agent deep reinforcement learning. *IEEE*.

Yannakakis, G., & Hallam, J. (2005). A generic approach for obtaining higher entertainment in predator/prey games.

Yin, G. (2009). On hybrid competitive lotka-volterra ecosystems. *Nonlinear Analysis*, *71*, 1370-1379.

Zhu, C. (2009). On competitive lotka-volterra model in random environments. *Journal of Mathematical Analysis and Applications*, *357*, 154-170.

Zou, J., Han, Y., & So, S. S. (2008). Overview of artificial neural networks. *Methods in Molecular Biology2*, *458*, 15-23.

Appendices

A Initial Project Overview

A.A Title of Project

The effect of climate change on predator/prey NPC behaviour in video games

A.B Overview of Project Content and Milestones

Overview: An Artificial Life project with the focus on how climate change affects predator/prey NPC behaviour models in video games.

The key stages are:

- To visualise how climate change can change the game world and how predator/prey NPC characters can adapt and survive under new conditions.
- To research current technology, predator/prey models and understand what research is already available and what is available for use in the next stage of this project.
- To develop knowledge and experience through research about artificial life models and climate change impact on it, thus it will help to develop a prototype.
- To create a prototype/simulation that would represent the results (more realistic and dynamic NPC) found in the research and provide a scenario that shows improvements made to an existing NPC model.

Projected Milestones:

2023

- Project management tools set up (Trello, Blog, GitHub, Gantt Chart) – ready for 1st September
- Literature review research concludes – draft ready for 31st October
- Interim meeting report – due week 9
- Explore and implement NPC model algorithms – (1st to 30th) November

- Explore and implement environmental changes to simulate climate change – (1st – 31st) December

2024

- Implementing improvements and new discoveries according to research results – 1st January – 29th February
- Running tests and comparing the prototype to another existing game/prototype – (1st – 15th) March
- Analysing & Evaluating test/comparison results – (15th – 25th) March
- Finishing dissertation – 25th March – 10th April
- Create honours poster presentation outlining the summary of the project, results and conclusion – (10th – 14th) April
- Poster & Dissertation – due 17th April
- Viva Voce – Week 13-15

A.C Main Deliverables(s)

- A dissertation paper with literature review, research, findings, test results, analysis and improvements made to the technology.
- A working prototype/simulation that shows an improved version of an existing predator/prey NPC model in a game world.
- An honours poster with the summary of the project, results and conclusion.

A.D Target Audience for the Deliverable(s)

The main target audience for this project are the people who develop games whether as an indie or a gaming company. They can benefit by using the research provided in this project to create a more realistic and efficient game that could be useful for educational purposes.

In addition, researchers at educational institutions would benefit by incorporating something new from this project into their own research whether it is climate change or artificial life related research.

Finally, anyone with an interest in artificial life and animal behaviour in video games would find this research interesting and useful in their own work.

A.E Work to be Undertaken

- An investigation into climate change use in video games.
- An investigation into artificial life in video games.
- An investigation into current predator/prey NPC models in video games.
- Collect data and statistics from similar existing video games that use artificial life.
- Design diagrams to show statistics/summary of all gathered/researched video games.
- An investigation into how current predator/prey model can be improved in a more dynamic, realistic way.
- Create diagrams/charts outlining results of the research.
- Research existing predator/prey model algorithms.
- Use the existing model as a base and implement new improvements according to research results.
- Test the prototype thoroughly. Record the results.
- Compare the prototype against other games that used same game engine. Record results.
- Analyse and evaluate the results.
- Create a poster with summary of the project, results and conclusion.

A.F Additional Information / Knowledge Required

- New knowledge will be required such as ML/RL and AI in video games.
- New knowledge will be required to understand how climate change affects animals and their survival.
- An extension of current knowledge about artificial life.
- An extension of current skills such as C++ and AI.
- An extension of technology skills of existing gaming engine UE/Unity or similar (game engine or simulation will depend on research results).

A.G Information Sources that Provide a Context for the Project

- A video showing predator/prey interaction. A simulation similar to this could be used to showcase the solution at the end of the project: https://www.youtube.com/watch?v=qwrp3lB-jkQ

- Video games often populate their in-game world with numerous ambient non-playable characters. Manually crafting interesting behaviours for such characters can be prohibitively expensive. As scripted AI gets re-used across multiple characters, they can appear overly similar, shallow and generally uninteresting for the player to interact with. In this paper we propose to evolve interesting behaviours in a simulated evolutionary environment. (Bulitko V, 2017)
- Prey can help fellow prey when the predator catches them (cooperation). Using a video game as a study system provided access to a large behavioural data set that would otherwise be difficult to obtain in nature. Using path analysis, we found evidence that coordinated resource acquisition and helping behaviours increased with the number of familiar individuals in the group. However, more cooperative actions increased their encounters with the predator and thus reduced their survival. (Er J, 2021)
- Simulation of population dynamics is a central research theme in computational biology, which contributes to understanding the interactions between predators and preys. Conventional mathematical tools of this theme, however, are incapable of accounting for several important attributes of such systems, such as the intelligent and adaptive behaviour exhibited by individual agents. This unrealistic setting is often insufficient to simulate properties of population dynamics found in the real-world. In this work, we leverage multi-agent deep reinforcement learning, and we propose a new model of large-scale predator-prey ecosystems. (Yamada J, 2020)
- Recently, there has been an increase in the development of online climate change games—that is, online video games specifically themed around climate change and/or related topics. This type of game has grown in popularity and more and more teachers have become increasingly interested in harnessing the potential of gaming strategies to advance climate change education. Well-designed games foster specific skills and abilities, contribute to the development of curricular content, and provide experiential learning, simulating unfamiliar circumstances that are impossible in real life. (Ouariachi T, 2019)

A.H Importance of the Project

Artificial Life is a broad topic and some areas are more researched than others. In this project, the focus goes to animal NPCs in video games. To add more complexity, climate change is taken into consideration and how animals behave and survive under changing conditions.

Both climate change and animal behaviour is researched well individually however, research is still in early stages when it comes to video games and a combination of both. Current research suggests that there is still a lot of unknowns in this field and that extensive research is required in order to understand animal behaviour during climate change.

This project will be using already researched data that can help to discover new ways of improving Alife (predator/prey) behaviour in video game world.

A.I Key Challenge(s) to be Overcome

The main key challenges come from development of the prototype:

- Understanding the complexity and implementation of the algorithm.
- Understanding how to implement Neural Networks and how it works with video game AI.
- Finding an existing algorithm that can be used as a base.

B Week 9 Report

Student Name: Edgar Park
Supervisor: Simon Powers
Second Marker: Andreas Steyven
Date: 17/11/2023

- Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **Yes**

Please comment on the progress made so far:
Need to learn new language, NetLogo
How can equations be implemented in NetLogo
Already found sources for integrating NEAT with NetLogo

- Is the progress satisfactory? **Yes**
- Can the student articulate their aims and objectives? **Yes No**

If yes then please comment on them, otherwise write down your suggestions.
Population based model from biology for predator prey model in video games in response to environmental change.
Population level as well as individual behaviour, a two level simulation.

- Does the student have a plan of work? **Yes**

If yes then please comment on that plan otherwise write down your suggestions.
Trello Board to keep track of task on a monthly basis.
Using the GANTT chart to keep track overall.

- Does the student know how they are going to evaluate their work? **Yes No**

If yes then please comment otherwise write down your suggestions.
Compare to other project (yet to be determined).

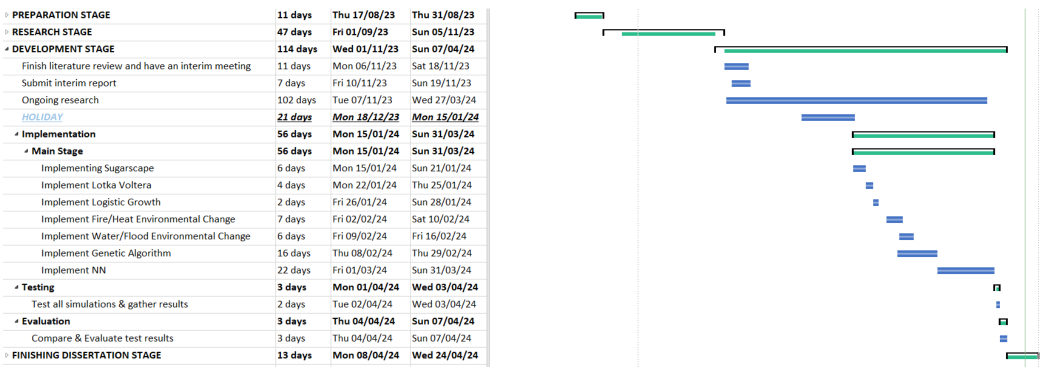
Any other recommendations as to the future direction of the project:
Background knowledge is now there to focus the scope and revise the RQs

and aims and objective in the light of this.

Signatures:
Supervisor: *Simon Powers*,
Second Marker: *Andreas Steyven*,
Student: *Edgar Park*

C Project Management and Diary

C.A Gantt Chart



PREPARATION STAGE	**11 days**	**Thu 17/08/23**	**Thu 31/08/23**
RESEARCH STAGE	**47 days**	**Fri 01/09/23**	**Sun 05/11/23**
DEVELOPMENT STAGE	**114 days**	**Wed 01/11/23**	**Sun 07/04/24**
Finish literature review and have an interim meeting	11 days	Mon 06/11/23	Sat 18/11/23
Submit interim report	7 days	Fri 10/11/23	Sun 19/11/23
Ongoing research	102 days	Tue 07/11/23	Wed 27/03/24
HOLIDAY	*21 days*	*Mon 18/12/23*	*Mon 15/01/24*
Implementation	**56 days**	**Mon 15/01/24**	**Sun 31/03/24**
Main Stage	**56 days**	**Mon 15/01/24**	**Sun 31/03/24**
Implementing Sugarscape	6 days	Mon 15/01/24	Sun 21/01/24
Implement Lotka Voltera	4 days	Mon 22/01/24	Thu 25/01/24
Implement Logistic Growth	2 days	Fri 26/01/24	Sun 28/01/24
Implement Fire/Heat Environmental Change	7 days	Fri 02/02/24	Sat 10/02/24
Implement Water/Flood Environmental Change	6 days	Fri 09/02/24	Fri 16/02/24
Implement Genetic Algorithm	16 days	Thu 08/02/24	Thu 29/02/24
Implement NN	22 days	Fri 01/03/24	Sun 31/03/24
Testing	**3 days**	**Mon 01/04/24**	**Wed 03/04/24**
Test all simulations & gather results	2 days	Tue 02/04/24	Wed 03/04/24
Evaluation	**3 days**	**Thu 04/04/24**	**Sun 07/04/24**
Compare & Evaluate test results	3 days	Thu 04/04/24	Sun 07/04/24
FINISHING DISSERTATION STAGE	**13 days**	**Mon 08/04/24**	**Wed 24/04/24**

Figure 22: Gantt Chart Showing Project Timeline

Break down:

- Preparation Stage - Project Set Up (17/08/23 - 31/08/23) - 11 days.
- Research Stage - Literature Review (01/09/23 - 05/11/23) - 47 days.
- Development Stage - Implementation (15/01/24 - 31/03/24) - 56 days.
- Development Stage - Testing (01/04/24 - 03/04/24) - 3 days.
- Development Stage - Evaluation (04/04/24 - 07/04/24) - 3 days.
- Final Stage - Finish Write Up (08/04/24 - 21/04/24) - 13 days.
- Final Stage - Create Poster (21/04/24 - 23/04/24) - 2 days.

Furthermore, GitHub repository was setup to keep the project for public access whilst regularly committing updates. Lastly, Trello board was created where Kanban board was used to follow project milestones, on a month to month basis. This include having a backlog with all necessary tasks that would contribute towards a milestone in the end.

C.B Project Diary

C.B.1 11th October 2023

Created and adjusted Latex document (dissertation). Included IPO and wrote a section of literature review about neural networks. Added 14 references.

C.B.2 13th October 2023

Wrote about NEAT and rtNEAT in literature review. Started research on NEAT implementation.

C.B.3 16th-17th October 2023

Researched video games, predator and prey NPC functionality. Discovered a game (NERO) that could potentially be used as a base in this project. Added new sections to literature review about NEAT and coevolution. Found a package with rtNEAT, should be investigated further and checked if code is compatible and manageable.

C.B.4 18th October 2023

Continued literature review. Wrote about "shaping" and "evolution".

C.B.5 20th October 2023

Finished literature review and sent it to supervisor.

C.B.6 2nd November 2023

It seems that my knowledge of advanced Java is not present; therefore, alternatives were needed. Downloaded NetLogo and looked into how it can be used. Investigated Lotka Volterra approaches and formulas. Also looked into Sugarscape and Logistic Growth models. This will potentially be used for creating a simulation.

C.B.7 10th November 2023

Finished and sent literature review to the second marker and arranged a meeting with him. Started methodology chapter (Introduction).

C.B.8 17th November 2023

Held interim meeting. Submitted interim report and added it to the dissertation paper. The meeting went well; it was mainly myself talking about the project. Also received some general advice about approach to studying and handling multiple projects at university.

C.B.9 7th December 2023

NetLogo approach was chosen and a simulation will be created to answer the research question. Started learning and making a start in NetLogo.

C.B.10 9th December 2023

Started Sugarscape implementation in NetLogo, with preys and sugar.

C.B.11 12th December 2023

Came up with the second research question. Next to NEAT, I added rtNEAT, so basically one offline, one online learning.

C.B.12 13th December 2023

Completed conceptualization section in methodology.

C.B.13 10th-18th January 2024

Finished implementing Sugarscape model. Preys can only interact with sugar patches, collect sugar, convert to energy, reproduce. Every move costs energy and prey has a vision cone of 50 degrees.

C.B.14 22nd January 2024

Was researching for libraries that could potentially be connected to NetLogo. All Java based: Encog, Neuroph, Deeplearning4j. Encog seems to be closest to possibly being implemented, but not researched further yet. Gained more knowledge about various evolutionary algorithms: NEAT, rtNEAT, Hyper-NEAT.

C.B.15 27th January 2024

Started implementing predator agents in NetLogo. Also, started planning and researching some environmental changes or natural disasters that could be simulated inside NetLogo. These few ideas are wildfire/fire and flooding/water.

C.B.16 29th January 2024

Attempted to implement NEAT with R only to realise that NEAT actually stands for another framework and its not what I thought it is. Wasted time. Decided to go for my suggested environmental changes. Started the implementation in NetLogo

C.B.17 1st-8th February 2024

Finished wildfire and flood scenarios inside NetLogo. Did some minor prey and sugar patch improvements, also partially implemented predator agents. Downloaded "Encog" machine learning framework that has NEAT and hyperNEAT. According to research, a bridge can be created to connect Encog and NetLogo together and use it.

C.B.18 12th February 2024

NetLogo performance seems a little slow, implemented some debugging to check what is wrong. It appears that prey agent movements cause slower performance, but it could be NetLogo issue as well. Nothing I can do if that is the case. Finished implementing predator agents. Now the simulation has Competitive Lotka Volterra model as well as Sugarscape and Logistic Growth.

C.B.19 13th February 2024

Implemented a switch button that allows a user to switch between different simulations.

C.B.20 15th February 2024

Minor improvements to environmental changes. Added wind to wildfire simulation and water pressure to flood simulation.

C.B.21 16th February 2024

Refactoring code and fixed small behaviour issues that predator agents had.

C.B.22 21st February 2024

Since testing and comparing this simulation against a similar simulation is not possible, I have split my simulation into 4 different simulations that will be tested and compared against each other. The 4 simulations are: 1. Full Sim 2. No neural network 3. Only prey sim 4. No environmental change sim.

C.B.23 1st March 2024

Attempted to create a bridge between Encog ML framework and NetLogo; however, Java appears to be on advanced level. My knowledge of Java is not as good as C and therefore, it is difficult to figure out how to use Encog in this situation. In addition, because there is not much time left, alternatives are required.

C.B.24 2nd March 2024

After research and getting advice from the supervisor, the decision was made to recreate NEAT as close as possible by using neural network and genetic algorithm hybrid inside NetLogo.

C.B.25 4th March 2024

Attempted to implement genetic algorithm for prey agents. Looked into different operators, such as selection, crossover, mutation.

C.B.26 8th March 2024

Decided to drop one of the research questions (NEAT) and only focus on rtNEAT question. In addition, I have used the wrong approach in creating NN/GA hybrid. Since I partially implemented GA, I will need to redo it, because the code will look slightly different once NN is implemented.

C.B.27 9th-11th March 2024

Focus on writing dissertation and methodology chapter. Mainly writing about what I have done so far and some theory.

C.B.28 12th March 2024

Further researched papers about sugarscape and neural network architecture.

C.B.29 14th-17th March 2024

Made attempts to implement neural network, but NetLogo appears to be a little complicated to use and I run into various errors; therefore, it takes some time to fix it.

C.B.30 25th-31st March 2024

Somewhat implemented neural network, also amended genetic algorithm code and created a hybrid. However, i still run into errors and the simulation does not work as expected.

C.B.31 1st-6th April 2024

Made further attempts to fix the errors, but still running into them. This time NetLogo cannot access chromosome weights that are assigned to prey agents. Further investigating this.

C.B.32 7th-8th April 2024

Writing implementation chapter, specifically about neural network and input layer, also adding appendices to the dissertation. Lastly, sent a draft version to supervisor for review.

C.B.33 11th April 2024

Received dissertation feedback, made adjustments accordingly. Just a few minor changes were required.

C.B.34 16th April 2024

Updated GitHub with latest project files. Finished implementation of one of the simulations that will be used for testing (no NA or GA). Worked on the issue with accessing neural network weights in the main simulation. Issue fixed.

C.B.35 17th April 2024

Worked on finishing neural network implementation. Hidden layer calculation fixed and working. Output layer finished. Added movement that works with neural network. Will need some tuning, but atleast neural network works as expected now for both prey and predators. Updated GitHub files. Also adjusted fitness calculation for GA and updates selection operator to take in predators as well as prey agents.

C.B.36 18th April 2024

Started testing chapter. Tested a few simulations and written feedback from observations. Updated dissertation paper in general, with all the necessary information that is required to finish it (appendices etc.).

C.B.37 19th April 2024

Had a final meeting with the supervisor. Received final feedback on the simulation.

C.B.38 21st April 2024

Created a poster and finished implementing genetic algorithm in the main simulation. Did testing, but with minor NetLogo performance issues, was not able to test the simulation as intended.

C.B.39 22nd-23rd April 2024

Finished writing dissertation. Project Complete.