Interactive Ecosystem Simulator

Applying Reinforcement Learning, Genetic Algorithms and Procedural Content Generation to Create a Graphical and Interactive Simulator

Bachelor of Science Thesis in Computer Science and Engineering

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Bachelor's thesis 2021

Interactive Ecosystem Simulator

Applying Reinforcement Learning, Genetic Algorithms and Procedural Content Generation to Create a Graphical and Interactive Simulator

JOHAN ATTERFORS
CARL HOLMBERG
ALEXANDER HUANG
ROBIN KARHU
BRAGE LAE
ALEXANDER LARSSON VAHLBERG

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Interactive Ecosystem Simulator
Applying Reinforcement Learning, Genetic Algorithms and Procedural Content Generation to Create a Graphical and Interactive Simulator
Johan Atterfors, Carl Holmberg, Alexander Huang, Robin Karhu, Brage Lae, Alexander Larsson-Vahlberg

Supervisor: Prof. Marco Fratarcangeli, Department of Computer Science and Engineering
Examiner: Prof. Gordana Dodig Crnkovic, Department of Computer Science and Engineering

Bachelor's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The image shows a screenshot of the simulation while running.

Typeset in LaTeX
Gothenburg, Sweden 2021

Abstract

As ecosystems are complex domains, both analytical and computer-aided models can aid in gaining insights about their dynamics. One such computer-aided model is the concept of ecosystem simulation. This project aims to build an interactive and visual ecosystem simulation in the Unity game engine. The purpose is to explore how modelling of animal behavior, trait evolution and dynamic terrain can be combined with a graphical representation to create an interactive ecosystem simulator. Implementation of these aspects includes exploration of machine learning and reactive behavior for animals, terrain generation, genetic reproductive algorithms as well as run-time visualization and collection of data. The effects of these aspects are evaluated using comparisons between animal behavior models, impact of terrain and outcomes of genetic evolution, in addition to software interactivity. The outcome of this project indicated that the machine learning prioritization animals performed nearly as well as reactive rule based animals in terms of survival, while the machine learning steered animals performed sub-par in comparison to the others. Furthermore, it showed that terrain changes seemingly has a greater impact on the predator populations compared to the prey populations in the simulator. Additionally, as a result of the proposed evolution model, genetic traits of animals indicated to be potentially adaptive to the environment. Finally, the graphical representation provided visual feedback and information to users. In total, the final product resulted in a working interactive ecosystem simulator. The implications of this thesis offers a baseline framework for modelling a visual interactive ecosystem simulator in regards to future research, academic and entertainment applications.

Keywords: Ecosystem, Simulation, Unity, AI, ML, RL, ABM, GA, PCG, Welford

Sammandrag

Då ekosystem är komplexa domäner, kan både analytiska och dator-drivna modeller av dessa verka som stöd för att få insikt i deras dynamik. En sådan dator-driven modell är konceptet av att simulera ekosystem. Det här projektet avser att bygga en interaktiv och visuell simulator av ekosystem i spelmotorn Unity. Syftet med projektet är att utforska hur modellering av djurbeteende, evolution och dynamisk terräng kan kombineras med en grafisk representation för att skapa en interaktiv simulator av ekosystem. I implementationen av dessa aspekter ingår en utforskning av maskininlärning och reaktivt beteende för djur, generation av terräng, algoritmer för genetisk fortplantning, samt visualisering och insamling av data under körning. Effekten av dessa aspekter evalueras genom jämförelser mellan djurens olika beteendemodeller, påverkan av terrängen, resultat av genetisk evolution, samt mjukvarans interaktivitet. Resultaten visar att djuren med maskininlärd prioritering presterade nästan lika väl som de reaktiva regelbaserade djuren gällande överlevnad, medan djuren med maskininlärd styrning presterade undermåligt i jämförelse med de andra metoderna. Vidare visar resultaten att terrängen verkar ha större påverkan på rovdjurspopulationerna än bytesdjurpopulationerna i simulatorn. Dessutom, som ett resultat av den givna evolutionära modellen, ger simulationer en indikation om att djuren potentiellt kan vara adaptiva till miljön. Slutligen, den grafiska representationen bidrog med visuell återkoppling och information till användaren. Sammanfattningsvis, den slutliga produkten resulterade i en fungerande interaktiv simulator av ekosystem. Tesens slutsatser ger ett grundläggande ramverk för modellering av en visuell simulator av ekosystem i avseende för vidare forskning, samt akademiska och underhållningsrelaterade tillämpningar.

Nyckelord: Ekosystem, Simulation, Unity, AI, ML, RL, ABM, GA, PCG, Welford

Acknowledgements

Special thanks to our supervisor Marco Fratarcangeli. We would also like to thank Michael Heron for voluntary help and advice during the end of the project.

Johan Atterfors, Carl Holmberg, Alexander Huang, Robin Karhu, Brage Lae, Alexander Larsson -Vahlberg, Gothenburg, June 2021

Glossary

AI Artificial Intelligence

FPS Frames Per Second

FSM Finite State Machine

GA Genetic Algorithm

MDP Markov Decision Process

ML Machine Learning

NavMesh Navigation Mesh, the area designating where agents can move

RL Reinforcement Learning

Contents

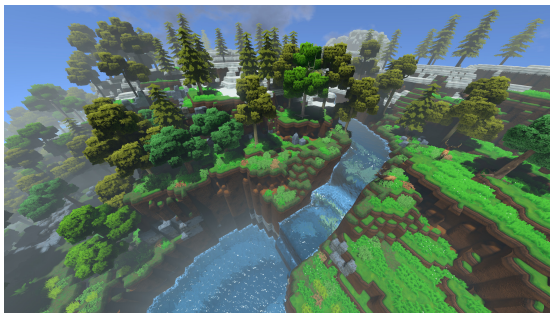
1
Introduction

Recent improvements in concurrent programming allow developers to create games with a large number of computer-controlled characters running simultaneously. A few examples are games such as “They are billions” and the mobile game “Bad North” [1][2]. Leveraging this technology allows for bigger and more complex games and simulations which can be used in both entertainment and scientific studies. One archetype in the field of scientific studies is ecosystem simulation where an ecosystem and its characteristics are modelled approximately. Using computers for the simulation is advantageous since real systems are chaotic and often too complex to be accurately modelled analytically. Additionally, graphical representation of the simulation allows for a clearer understanding of the results, particularly to people outside of computer science and ecology. Furthermore, allowing for user control of such a simulation increases its usefulness by allowing experimentation and promotes insight.

1.1 Background

There are two approaches to ecosystem modelling: analytical and computer-aided, where the first is mainly used for linear systems with defined mathematical expressions. Analytical ecosystem models can be represented as differential equations, such as the Lotka-Volterra equations which models a predator-prey system [3]. Computer-aided modelling is used when it is difficult to form a mathematical expression and when an approximate solution is sufficient. Regardless of the approach, ecosystem modelling allows for exploration of initial values and parameters and their corresponding results in terms of population changes, behavior and evolution. This allows scientists to perform studies that would otherwise be hard or even impossible due to lack of time or space.

In addition to providing a controlled environment for ecological studies, computational modelling can be combined with computer graphics to provide visual feedback. Since ecosystems can be difficult to comprehend, visual representation might aid understanding. As a result, multiple visual simulators have lately been popularized, for example the games “Eco” (figure 1.1a) and “Equilinox” (figure 1.1b) [4][5]. These are examples of how graphics can be used to portray ecosystems for entertainment.



(a) The Eco game[1].

(b) Equilinox game[2].

Figure 1.1: Two games that are examples of graphical ecosystem simulators. The games feature advanced terrain and life with graphical representations.

In addition to entertainment, the visual representation is valuable for simulations as it makes results tangible. Furthermore, with visual feedback both small scale interactions and emerging patterns on a larger time- and spatial scale can be observed. This can not only be used to provide entertainment value, but also to create a more immersive view of complex systems for other purposes.

Apart from visual technology, applications of traditional optimization techniques have emerged in the past decade. Specifically the application of *Genetic Algorithms (GA)*, which is a search heuristic inspired by Darwin's theory of natural selection, in continuous virtual ecosystems [6]. Simulating evolution in such environments has shown to be possible by the adaptive processes found in GAs as exemplified in "Bloop's world" [7, Chap. 9.13].

As the access to faster and cheaper computational power increases, so does the prospect to simulate more complex systems. However, most real ecosystems are still too complex to be simulated to their full extent. For natural systems, unmanageable complexity appears already at the level of visible interactions and as result of the large number of individual variables present in the system. Due to this, simplifications are required for both the living and inanimate components of ecosystems. Concretely, simplifications can be made in the characteristics and behaviors of the animals as well as the properties of the physical environment. These simplifications are also useful to improve performance in simulations involving larger amounts of entities. Examples of such simplified systems are the EcoSim research project, a software project on which multiple research papers have been published [8].

[1]Source: `https://wiki.play.eco/en/Eco_Wiki#/media/File:Cabin_Logs_Tomatoes_Corn.jpg`, used under Creative Commons: `https://creativecommons.org/licenses/by-nc-sa/4.0/`

[2]Source: `https://equilinox.com/presskit/equilinox/images/foxHunt.png`, from Equilinox press kit: `https://equilinox.com/presskit/sheet.php?p=equilinox#images`

1.2 Purpose

In this section, the purpose of the project is presented and motivated. The purpose is to explore how modelling of animal behavior, trait evolution and dynamic terrain can be combined with a graphical representation to create an immersive ecosystem simulator. The purpose is substantiated in the three goals below.

1. Model a plausible ecosystem with interacting animals influenced by evolution who perceive their own environment to make autonomous decisions.
2. Dynamically generate terrain from user input, which affects the ecosystem.
3. Create interactive software for use in other studies.

All decisions and implementations were made with the purpose of achieving one or more of these goals. The first goal of modelling a plausible ecosystem, in point one, requires studies and implementations of the interactions between entities in the simulated world. Furthermore, these interactions depend on their environment and can be categorized into different areas of responsibility. These areas include perception, which implies hearing and vision, decision making that prioritizes actions and the actions themselves. There is also a requirement of implementing genetic reproductive evolution for these areas to be affected by evolution. Together, these aspects make up the behavior of the animals. The second goal of dynamically creating terrain introduces the need for generating content procedurally. Additionally, to create a usable simulation software for others, results of the simulation need to be displayed in a clear way. This requires some form of data visualization, for instance by using graphs to show statistics about animals and plants over time. Aggregating statistics from a running stream of data requires specialized techniques. Finally, the third point also gives rise to a need of interacting with the simulator during run-time which requires a *Graphical User Interface (GUI)*.

1.3 Scope

As with many projects, there is a need to outline aspects that will not be explored, and how certain features will be limited. Below follows some points for clarity.

- Model a simplified ecological system without any intended specificity.
- Limit the food chain to only include select producers, herbivores and consumers.
- The animals' actions will function the same way throughout the duration of the simulation.

These limits mainly exists to prevent unnecessary precision in the modelling which might require time and effort better spent elsewhere. Simplifying the simulation is required to avoid details and studies we as a group are not well equipped for. The second limitation of not modelling a complete food chain originates from the extreme level of detail needed for such a simulation. These details include chemical interactions in the photosynthetic chain and microbial level in addition to longer

systematic changes such as circadian rhythms and seasonal changes are outside of the group's domain knowledge. Finally, limiting to only immutable actions contributes to system predictability, regulation and simplified design.

1.4 Thesis Outline

This thesis will present the implementation of an ecosystem simulator in Unity with C#. In chapter 2, relevant background theory for *Artificial Intelligence (AI)* behavior, GAs, environment generation, and statistical data collection is given. Chapter 3 presents the implementations and methodologies applied during the project. Chapter 4 is dedicated to presenting the outcomes of some ecosystem simulations, as well as determining the interactivity and performance of the software. Chapter 5 presents a discussion of whether the goals of the project are reached, some ethical aspects and future improvements. Finally, the conclusion in chapter 6 summarizes the project briefly.

2
Theory

In this chapter, the theory behind the projects technical implementations are covered. Firstly, an explanation of the theory behind the animals' AI behavior. Secondly, the theory of GAs are explained. Thereafter, a description of theory used in environment creation is presented, and finally a method for efficiently calculating the sum of squares for statistical collection is presented.

2.1 AI Behavior

Action taken in conjunction with the environment, in other words behavior, is required for simulating an ecosystem and the living entities within it, since living entities act. With the purpose of modelling entities that make autonomous decisions, this takes on an *Agent-based modeling* (ABM) approach where the entities are "modeled as a collection of autonomous decision-making entities called agents" [9]. The ABM approach implies that each agent acts based on its individual situation and set of rules. In an ecosystem simulation, the behavior exhibited by such agents contribute greatly to the survival of the animal and, by extension, the species. As they are governed by natural selection, animals that survive longer have a greater possibility of passing on their genes, thus behavior has a large impact on evolution as well. Therefore, it is clear that behavior is an integral part of the simulator.

For the purpose of this simulation one could consider 2 different branches of AI. These branches of AI and some of their respective techniques are:

- **Reactive**: Rule-based, actions taken based on current state.
  - Finite State Machines (FSM).
  - Behavior trees.

- **Machine Learning (ML)**: Learning from data, performing the most optimal action based on expected outcome.
  - Reinforcement Learning (RL).

Different approaches has contrasting outcomes and difficulties of implementation, which makes it important to choose approaches consciously. The following section therefore explains the branches of AI further and details some of the algorithms that might be applied in the respective branches. The advantages and disadvantages are also covered briefly to provide a basis for the decisions made in the project.

2.1.1 Reactive AI

Reactive AI consists of mainly deterministic behavior using rules, and is synonymous to a rule-based system [10]. As can be seen in figure 2.1, the reactive agent parses the current state into actions based on simple rules.

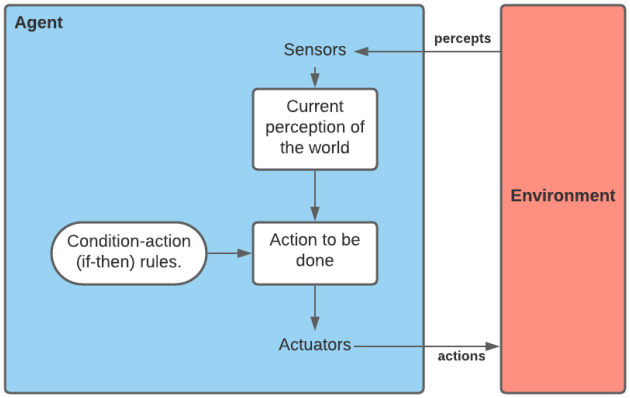


Figure 2.1: Reactive agent behavior. The agent (blue) receives percept from the environment (red), which are turned into actions that affect the environment.[1]

As shown in figure 2.1, a reactive agent's perceptions of the environment are parsed based on if-then rules into actions to be performed by the agent. A reactive agent acts according to pre-programmed decision logic and while such systems are often easy to implement, they also produce purely reactive results in terms of behavior, as implied by the name. Reactive systems are often used for their transparency and for ease of implementation in simple use cases. Common techniques used for reactive AI are behavior trees and *Finite State Machines (FSM)*.

An FSM is defined by a set of states and transitions between states. In the context of FSMs, a state is defined as a predefined subset of the agent's behavior. Only one state can be active at a time and the active state determines the current behavior of the agent. Conditions are also defined for when transitions between states occur, and in the traditional sense of FSM, each state determines these conditions and thus governs the transitions [11, Chapter 9]. An example of an FSM can be seen in figure 2.2, which shows the states, transitions and conditions.

[1]Heavily inspired by: `https://en.wikipedia.org/wiki/Multi-agent_system#/media/File:IntelligentAgent-SimpleReflex.png`

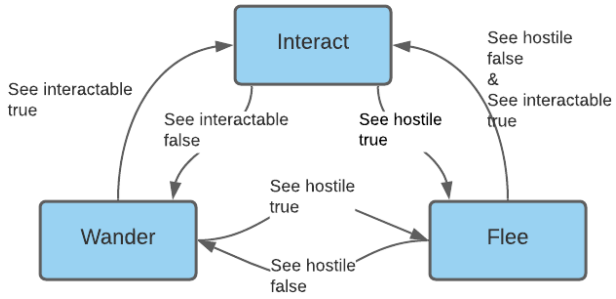


Figure 2.2: FSM example: the nodes correspond to states, edges are transitions and weights are conditions. Inspired by: [11, Chap. 9].

The figure displays how the actions transition into other actions based on conditions. The figure also shows that the system is easy to understand and manage. Nonetheless, imagining a similar FSM with 100 nodes illustrates a clear drawback to the approach, as it would be complex upon scaling. Since it is difficult to achieve more complex behavior with FSMs and reactive behavior in general, another more flexible approach is required.

2.1.2 Machine Learning AI

Machine Learning (ML) is a set of AI techniques in which an algorithm improves as it is exposed to more data. *Reinforcement learning (RL)* is in turn a subset of ML problems concerned with training an agent to interact with its environment and thus learn behavior. Conceptually, this is learning comparable to how humans learn, and is therefore strikingly different in procedure from the previously mentioned reactive approach. In combination with the rise of ML in recent years, it is therefore interesting to explore in which areas the ML approach is useful or not. For this project, the interest was to see what behavior it could produce in animals living in an ecosystem.

Mathematically, RL corresponds to maximizing reward by infinitely traversing a *Markov Decision Process (MDP)*. As seen in figure 2.3, an MDP is a directed graph with rewards and actions. In the context of an MDP, and continuing in this chapter, a state is defined as the agent's observations about the environment and itself. The nodes of the graph are states, from which multiple actions may be taken. As a result of an action, a transition to another state occurs based on probability. Rewards can then be given upon transition and the problem that needs solving is how to take actions that would optimize the reward.

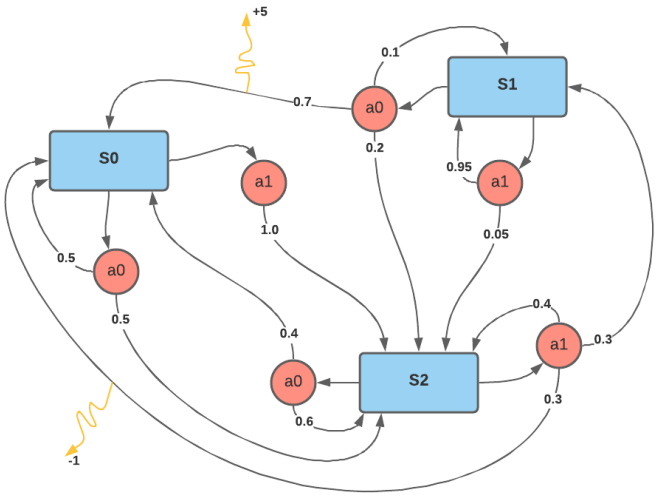


Figure 2.3: Machine Learning AI: MDP, represents the problem space of RL. A directed graph with rewards (yellow arrows) and actions (red circles).[2]

Training an agent to traverse an MDP means teaching it mappings between state, action and the expected *utility* of that action in that state. The expected utility is the reward that the action yields as well as the expected rewards from future actions. The rewards for future actions are discounted, which means that imminent rewards are valued higher.

The set of all mappings between state, action and their corresponding utility, is called a policy. A policy that maximizes reward, thus taking the best actions, is called an optimal policy. An optimal policy can be viewed as a map which tells the agent: "If you are in state S do action A since it is expected to give the highest total reward". The usage of such a policy to traverse an MDP is called *inference*. In other words, inference means to follow the relationships defined by the pre-gathered policy and by its guidance perform certain actions when in certain states.

In this project, inference corresponds to exhibiting animal behavior within an ecosystem which means that the problem is modeled as an MDP. The states in this case are made up of what the animal gathers from its senses in combination with its internal parameters. The actions are how the animal may interact with the environment, and the desired actions in certain situations can be rewarded or penalized.

There are many benefits of implementing RL over an FSM. With RL, dependencies between states become easier to maintain, and the behavior becomes easier to extend. Furthermore, since the RL agent defines its own behavior through training, it has the potential to be flexible and adaptable, which is covered further in section 3.2.3. These are good properties in a plausible ecosystem simulation making it interesting to explore.

[2]Heavily inspired by: `https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690`

However, one clear drawback of the ML approach and RL lies in the difficulty of implementation, as achieving precise behavior is a challenging task. This also means that the resulting behavior might be less successful at achieving its goal than if produced by reactive AI, as is covered in section 3.2.3.

2.2 Genetic Algorithms

To mimic evolution in software, *Genetic Algorithms* (GAs) are often utilized. Applying GAs provides the simulator with a plausible system by which animals can evolve and adapt their traits over time. While there are multiple alternative evolutionary algorithms to simulate evolution, a GA is notably applicable in this context due to the evident conceptual connection to an ecosystem.

GA is an abstraction of a procedure regarded as a *metaheuristic* procedure, which by definition provides a solution to an optimization problem without depending on the problem at hand, unlike a heuristic procedure which is designed specifically for the problem [12]. The theory of GAs became popular through the works of J. Holland. In particular *Adaptation in Natural and Artificial Systems*, in which the professor introduced a formalized framework regarded as "Holland's schema theorem", also labeled as "The fundamental theorem of genetic algorithms" [13][14].

One of the applications emerging from Holland's theorem is the notion of GAs, also based on Charles Darwin's theory of "evolution by natural selection", which was presented in *On the Origin of Species* [15]. The literature proposes what is considered to be the foundation of evolutionary biology. It outlines the process where animals in an environment, each containing a set of genes, survives and passes down their genes by reproduction. Thus, the animals with sufficiently adapted genes survive and those not well adapted are unable to reproduce and pass their genes to successive generations. Same concept is utilized in GAs where the algorithm process is comprised of a sequence of steps visualized in the flow chart diagram in figure 2.4. The idea is to continuously generate better solutions by utilizing good solutions relative to the population for each new generation until a convergence is obtained.

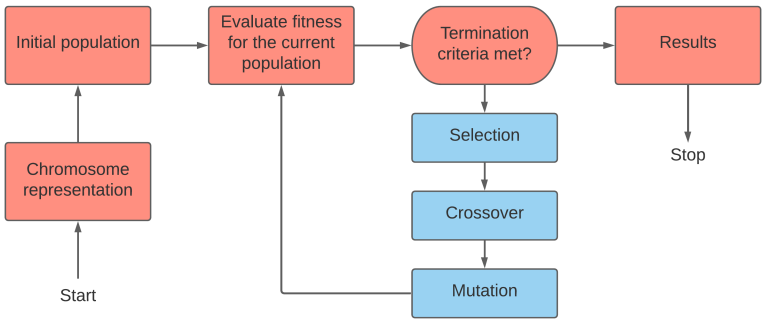


Figure 2.4: A flowchart diagram representation of a GA. Each container represents a step in the algorithmic sequence, ranging from chromosome representation, evaluating fitness function and checking for termination criteria. Until termination criteria is met, iterations of selection, crossover and mutation is processed between the fitness evaluation step to improve the solution.

In the context of GAs, the term *chromosome* is what evolution acts on, and is synonymous with the term *individual* [16]. As displayed in figure 2.5, a chromosome contains a set of genes that can be represented in numerous ways depending on the problem. A gene can for instance be encoded as a bit, an integer or a floating point value. Consequently, a chromosome is usually conveyed as a data structure or solely a string of bits in the case of a bit gene encoding. Additionally, a *population* is comprised by multiple chromosomes.

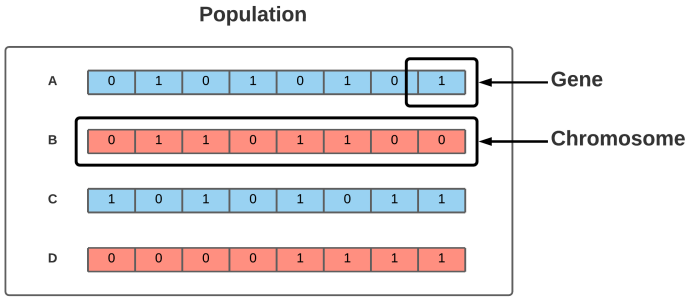


Figure 2.5: The figure shows a population, which has a set of chromosomes. The letters A, B, C and D refers to chromosomes (or individuals). A chromosome is in turn a set of genes, where these genes are encoded as bits.

GA is a subclass of Evolutionary Algorithms and refers to algorithms inspired by the core principles of Darwinian natural selection, notably *heredity*, *variation* and *selection* [7]. Heredity describes the process of children inheriting the properties of its parents for each new generation. Variation involves the concept of comprising a population with varying genes. Excluding variety in a population would prompt the population to pass down the same set of genes for each new generation, henceforth new combinations of genes would never occur and nothing would evolve. Selection is a mechanism where a subset of the population passes down its genes, particularly the fittest of individuals. For example, if rabbits are being hunted by wolves, the

fastest of rabbits would most likely be the ones surviving and passing their genes to the next generation. The three Darwinian principles are encompassed by distinct evolutionary operators such as *parent selection*, *crossover* and *mutation*.

2.2.1 Parent Selection

Parent selection is an operator where each individual is evaluated against a fitness function $f(x)$ and selected to pass down its genes depending on whether or not it is a good solution relative to the population [7]. J. S. Arora defines fitness function as "the relative importance of a design" [17]. In essence, it is an objective function used to indicate how close a given solution is to achieving a termination criteria. Such functions are usually implemented as algorithms or mathematical functions.

One of the traditional methods for selecting parents in a GA is *fitness proportionate selection*, which is a stochastic selection method that uses a probability ratio proportional to an individual's fitness score for selecting parents [12]. As an individual carry a higher fitness, its probability to be selected as a parent is higher. Assuming f to be the fitness function, a probability p for a given individual $x \in Population = \{x_1, x_2, ..., x_n\}$, is calculated as

$$p(x) = \frac{f(x)}{\sum_{i=1}^{n} f(i)}.$$

Selection for Ecosystem Simulations

While the vast majority of selection methods in GAs assume a discrete flow of the algorithm, where each generation is carried out by resetting the environment and subsequently proceeding in discrete steps, a realistic model of an ecological system entails a continuous approach. Furthermore, a realistic model of an ecosystem would not assume that all children are born at the same time step, nor that parents are selected by abstract methods which runs independently from the domain. Fitness proportionate selection is an example of a method that is independent from the domain, since it will allow for two animals to be selected as parents to a child independently of their physical positions as long as their fitness is high enough.

A selection method which is domain dependant for a continuous ecosystem simulation is demonstrated by D. Shiffman in *The Nature Of Code* [7], in which the professor claims that "things that happen to live longer, for whatever reason, have a greater chance of reproducing". In the proposed method, a fitness function is found in the form of survival and not a mathematical function. Thus, individuals that are well fit for their environment will survive for longer and consequently have more time for reproduction. The objective is to model natural selection such that the fitness of the GA is representative of some aspects of Darwinian fitness. Darwinian fitness is proportional to the ability to produce offspring and passing genes further to successive generations [18].

2.2.2 Crossover

During the crossover phase, which follows the selection phase, the genes that are to be received by each child from its parents are determined. This operation can be seen as creating a new chromosome with a set of genes constructed as a combination of each parent's genes. There are various methods for selecting genes in the crossover phase. Among those are the more notably *single point crossover* and *uniform crossover*. Uniform crossover works by treating each gene separately, and for each gene to be assigned in the child's chromosome, flip a coin to determine which parent's respective gene to choose, as can be seen in figure 2.6 [14]. The coin can either be unbiased, thus, each gene assignment has an equal probability of originating from either parent, or biased, such that the resulting chromosome has a higher probability of consisting of more genetic material from the favored parent [19]. Conversely, the idea of single point crossover is to randomly select a cutoff point in the child's chromosome, such that genes on the left of the cutoff point originates from one parent and genes on the right of the cutoff point originates from the other, as figure 2.7 shows.

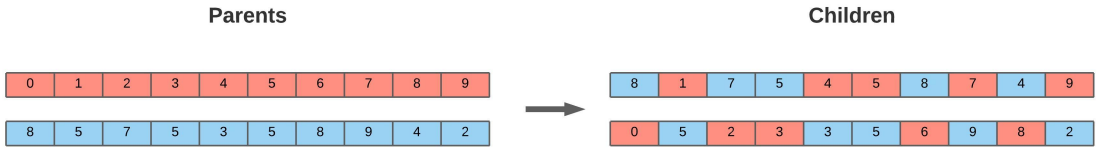


Figure 2.6: Uniform crossover operation on two parent's chromosomes resulting in two children's chromosomes.

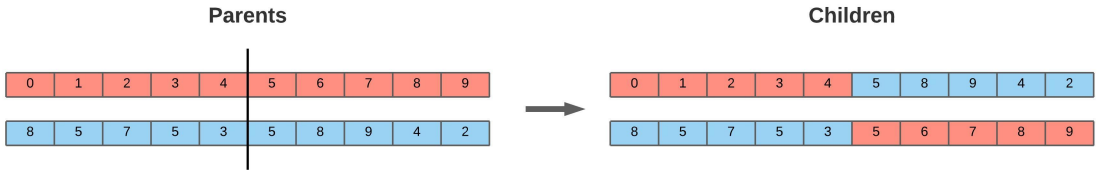


Figure 2.7: Single point crossover operation on two parent's chromosomes resulting in two children's chromosomes.

2.2.3 Mutation

While crossover solely outputs chromosomes based on already existing genes, mutation is applied afterwards to maintain genetic diversity in the population [6]. In the context of GAs, mutation is used to alter genes in chromosomes in order to avoid stagnating in a local maximum in the solution space. Consequently, diversity and exploration for potentially better solutions are obtained through mutation. A parameter that has to be considered in order for the algorithm to work properly is the probability of a mutation occurrence on a gene, namely the *mutation probability* [7]. In essence, for each gene in the chromosome there is a small probability of altering

the gene. This works by iterating over all the genes in the chromosome and appointing them for mutation with the same probability as the mutation probability. If they are not appointed for mutation, they are simply ignored. If mutation probability is set too high, the algorithm will correspond to a random primitive search [6]. In this case, the GA will mutate the chromosome too often, resulting in the optimal chromosome having to be found through randomness rather than incrementally going closer and closer to the optimal chromosome.

Since mutation is heavily dependant on chromosome representation, there exists various ways of mutating a gene once it has been appointed for mutation. If the chromosome is a binary encoding, a simple method like *bit flipping* is usually sufficient [6]. Bit flipping simply takes the current gene, which is represented as a bit, and inverts it as shown in figure 2.8.

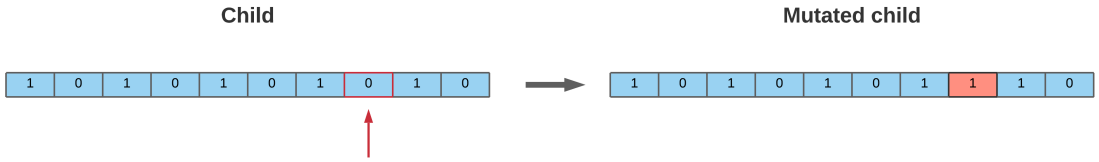


Figure 2.8: Bit flipping mutation. The red arrow points to a gene appointed for mutation.

However, in the case of a floating point number representation, a *uniform mutation* or *nonuniform mutation* is to be used [6]. Uniform mutation is an operation equivalent to bit flipping, where the current gene is assigned a value randomly from a range $[L_u, U_u]$ bounded by a user selected lower and upper limit. Nonuniform mutation for floating point values works by adding a small positive or negative value Δx_i to the gene in focus. In the case of floating point representation, the amount to be added is usually randomly drawn from a Gaussian distribution with mean zero and user selected standard deviation,

$$p(\Delta x_i) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(\Delta x_i - \epsilon)^2}{2\sigma^2}},$$

in which approximately two thirds of the drawn samples will lie within a standard deviation around the mean. The amount is usually clamped by the bounds of user selected lower and upper bound $[L_u, U_u]$.

A common way of drawing samples from a Gaussian distribution is to use the Box-Muller transform. Box-Muller transform is originally meant to sample two independent random variables from a normal distribution using two separate equations [20]. However, in the case of sampling a single number, one of the equations is sufficient. The procedure starts with sampling two independent values U_1 and U_2 from a uniform distribution on the unit interval $[0, 1]$ and then inputting these values to

$$X_{norm} = \sqrt{-2\log U_1}\cos(2\pi U_2),$$

which outputs a normally distributed sample. Lastly, the sample is used as input in

$$X_{gaus} = X_{norm}\sigma + \mu$$

to convert to a value derived from a Gaussian distribution with mean μ and standard deviation σ [20].

2.3 Environment Generation

To understand how the environment affects the ecosystems, terrain can be generated from input to provide different environments in the simulations. Presented in this section are the techniques in terrain generation which were applied within the project.

2.3.1 Noise-Based Terrain Generation

Computers are good at creating flat and smooth surfaces, but to imitate the randomness and imperfections of real environments, noise can be added. Using noise to generate a height map is common practice in the field of procedural landscape generation. Without a noise function, all valleys and mountains would need to be handcrafted which would not be optimal when the aim is to generate multiple varied environments [21].

In this implementation, the primary method used for generating the terrain mesh is based on *Perlin noise*, which is structured and organic random noise. As can be seen in figure 2.9, Perlin noise is not as random as truly random noise, and is more organic and cohesive.

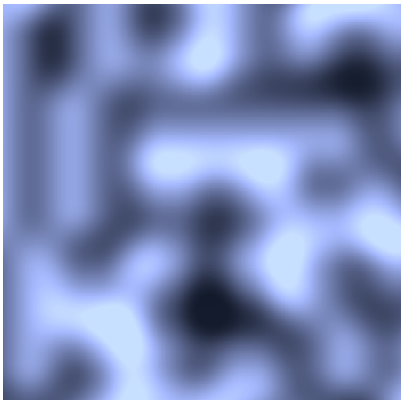
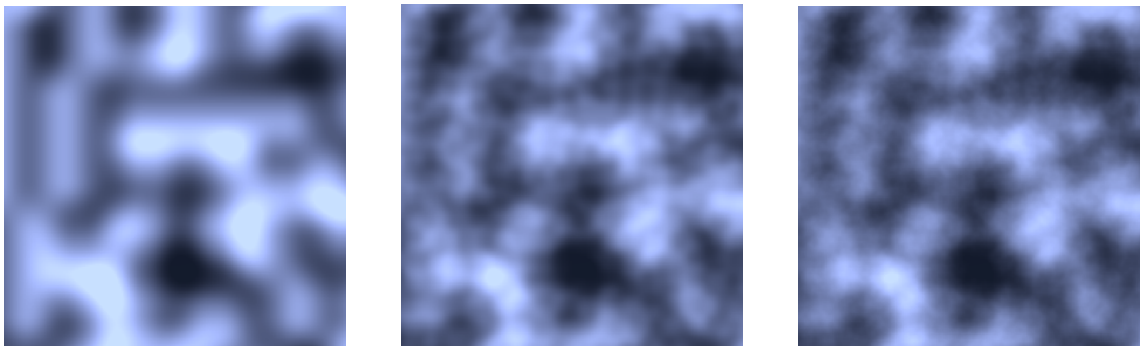


Figure 2.9: Example of *Perlin Noise* where values are represented as a gradient between black and white depending on the value at the point.

This figure shows generated noise, which could resemble a top down birds-eye view of a landscape, however, real world terrain is not as smooth as an image that Perlin noise generates. To make the terrain more realistic, a common technique is stacking different runs of the Perlin noise algorithms into multiple layers, called *octaves* [22].

This layering is done by creating a new noise map where every point consists of adding the same point from every layer, with each layer having less importance depending on a set persistence value. These layers also increase in detail, or frequency, the less persistence they have. Each succeeding octave has a diminishing effect but an increasing detail level, which helps create details and ruggedness on the terrain. Applying different numbers of octaves results in a more realistic height map that generates results that are more organic, as can be seen in figure 2.10 below.

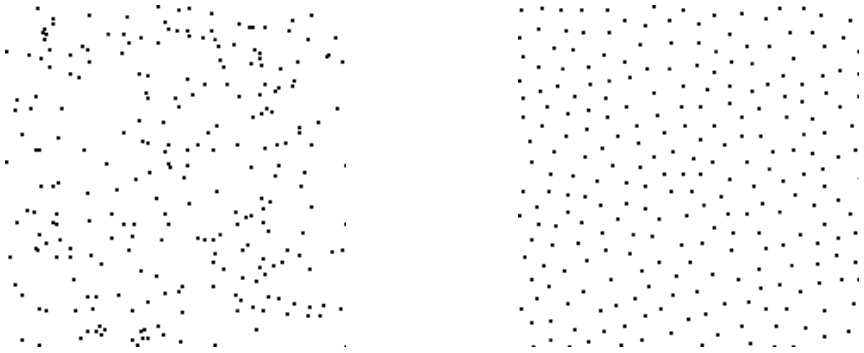


(a) One octave **(b)** Two octaves **(c)** Five octaves

Figure 2.10: The difference of using octaves and layering with increasing number of octaves from left to right. The only variable adjusted between the three sub-figures are the number of octaves. Due to diminishing returns for each new octave, the only visual difference between the results of figure 2.10b and 2.10c is the sharpness.

2.3.2 Poisson Sampling for Object Placement

There are many ways to generate positions for objects, with varying results. The main criteria of the object placement system in this project is the ability to modify the density of the placed objects, and that the objects are uniformly spread. The uniform distribution minimizes the chances that animals gets placed in groups and allows the animals to have a higher probability of surviving the first few seconds. This also makes sure that no impassable walls of trees are created. Some examples and comparisons of different methods are shown in figure 2.11.



(a) Random placement **(b)** Poisson placement

Figure 2.11: Example of two point generation methods[3]. Selecting points with completely random coordinates results in some clustering and some clear parts of the field, as can be seen in 2.11a. The method presented in figure 2.11b visualizes points generated with an algorithm used to create a *Poisson Disk Sample Set* [23].

Poisson Disk Sampling algorithms are very useful for placing objects in an environment, such that the objects are packed together but not closer to each other than the selected minimum distance [23]. Poisson Disk Sampling can be combined with the aforementioned noise algorithms, as described in section 2.3.1, to dynamically modify the minimum distance between the points used in the algorithm. As each point is generated using a radius from another point based on a set minimum distance, modifying this distance with a noise function creates a placement algorithm that coincides with the noise function. This results in interesting object placement that can look more natural than pure Poisson disk sample set points where each point is a sample point from the Poisson disk set, as can be seen in figure 2.12.

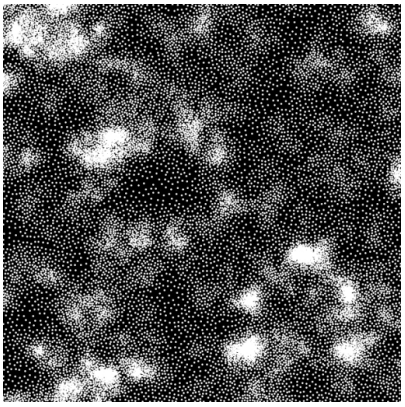


Figure 2.12: Poisson Disk Sample modified with Perlin Noise. Where each white point is a chosen location to place an object on the black background[4].

The clustering comes from setting the minimum distance between the points with the noise function. This is more natural clustering than the completely random placement, as the clustering is based on more organic Perlin noise.

[3]Source: `http://devmag.org.za/2009/05/03/poisson-disk-sampling/`
[4]Source: `http://devmag.org.za/2009/05/03/poisson-disk-sampling/`

2.3.3 Mesh Construction

To transfer the aforementioned techniques of generating a height map to a 3D environment, a mesh needs to be created. A mesh, or polygon mesh, consists of vertices connected with edges. A vertex consists of information about for example position, color and normal vector, and the edges that connect these vertices creating faces. These faces can usually either be triangles (three vertices) or quads (four vertices) depending on the number of edges for the shape.

2.3.4 Navigation

A navigation system is a vital part in the brains of the simulated animals. Without being able to find a path in the environment, the animal would not be able to move. There exist many solutions to the standard problem of finding the shortest path, one among them is A* (A star), which is used in Unity. A* is a standard path finding algorithm that works for node networks, but a 3D environment is not a node network. To adapt A* to a 3D environment, a method that can be used is the creation of a navigation mesh, or navmesh. With a navmesh the environment is divided into polygonal segments where each segment is fully walkable. An example of a navmesh can be found in figure 2.13.

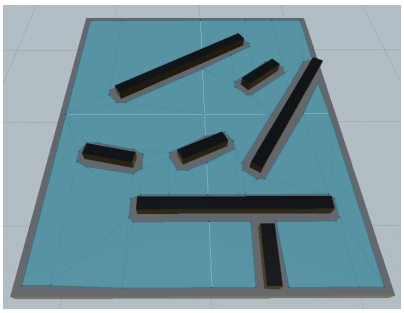


Figure 2.13: Visualization of a navmesh in a small environment where the gray is the environment, the black objects are impassable walls, and the blue layer atop the environment is the navmesh [5].

2.4 Calculating the Corrected Sum of Squares

To improve understanding of ecosystem simulations, it is useful to look at visualized data. To collect and display such data in run-time, efficient calculations are required. One algorithm that performs efficient statistical calculations is called *Welford's method* and is covered in this section.

[4]Source: `https://upload.wikimedia.org/wikipedia/commons/6/62/D3D_Shading_Modes.png`

[5]Source: `https://docs.unity3d.com/uploads/Main/NavMeshAgentSetup.svg`

In 1962 B. P. Welford presented a short paper in which he described a one-pass algorithm for calculating the variance and mean of a data set [24]. A one-pass algorithm is an algorithm for processing data streams which reads its input data once. This is particularly useful in simulations or signal processing, where new data is continuously arriving. Since most other solutions requires two passes over the data, the entire collected data set need to be stored. Two-pass solutions also presents a problem if the mean and variance is needed before the end of the data stream, since the entire two-pass calculation would need to be executed each time a new value arrives. Furthermore, the method presented by Welford is numerically stable, which implies fewer errors even if the data set is large or the data points themselves differ by some magnitude. The following section will first explain the method to compute the mean, and the latter part will explain the recursion formula for variance in Welford's method.

The mean $\overline{x}$ of a set with n elements, where x_N is the Nth element, is defined as

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{2.1}$$

Thus, to find the Nth mean ($\overline{x}_N$) given the previous mean ($\overline{x}_{N-1}$) we can use the recursion

$$\overline{x}_N = \frac{1}{N}((N-1)\overline{x}_{N-1} + x_N).$$

This is, however, unstable numerically since the term $\frac{1}{N}$ could be very small in relation to the sum, which could cause instability when using floating point formats. Rewriting the equation by distributing $\frac{1}{N}$ and simplifying, results in the recursion

$$\overline{x}_n = \overline{x}_{N-1} + \frac{x_N - \overline{x}_{N-1}}{N}. \tag{2.2}$$

This can been shown to be numerically stable due to not using disproportionately sized numbers in division or multiplication.

In order to demonstrate the recursion for calculating the variance, it is best to start with the difference between two consecutive variances. The current variance is s_N^2 and the previous s_{N-1}^2 but for simplicity the derivation will instead use $(N-1)s_N^2$ and $(N-2)s_{N-1}^2$ since it allows for simpler summations. The result will be divided by $(N-1)$ for the correct variance. The difference between the consecutive variances is

$$(N-1)s_N^2 - (N-2)s_{N-1}^2. \tag{2.3}$$

Using the definition of the variance in (2.3) and summation rules, this can be shown to be equal to

$$s_N^2 = \frac{1}{N-1}((N-2)s_{N-1}^2 + (x_N - \overline{x}_N)(x_N - \overline{x}_{N-1})). \tag{2.4}$$

To conclude, (2.4) is a recursive formula requiring only the last known value of the variance S_{N-1}, the last known average $\overline{x}_{N-1}$ and the value to add to update the

variance. To avoid numerical instability, $(N-1) * s_N^2$ may be stored and restored to s_N^2 when needed. This method allows statistical data to be computed efficiently. It also simplifies implementation since no data from deceased animals need to be saved to memory during the simulation. Furthermore, it avoids calculating the statistical data by using regular methods, some of which are not as stable. Ultimately, this would have considerable impact on performance to serve live-data representations.

3
Method

Presented in this chapter are the design choices and implementations required to fulfill the purpose of the project. The following sections describe the chosen animals and their features, their AI behavior, the evolutionary system, the generation of the ecosystem's environment and the method of data collection.

3.1 Modelling the Animals

This section covers the internal models of animals, namely how they are affected by the environment via parameters, how they behave via traits and how they perceive the environment through senses. The animals included in the simulation were chosen based only on familiarity to the team and to achieve a predator-prey relationship.

This project has taken inspiration from other ecosystem simulations, which had limited the amount of animals to just one prey type and one predator type, specifically rabbits and foxes [25] [26]. On account of the modular approach to the animal system in this project, it was possible to expand the selection of animals without much additional work. The four chosen animals were rabbits, wolves, bears and deer, as seen in figure 3.1.

Figure 3.1: The animals featured in the simulation. Left: Deer and wolf, Right: Bear and Rabbit

3.1.1 Parameters

In the context of the simulation, the various values of the parameters and traits defined each individual animal (see appendix B). The parameters were simple abstractions/representations of an animal's need to eat, drink and procreate, and main-

taining them would serve as the animal's goals. The parameters would vary in value during an animal's lifetime. Additionally, the parameters were affected by the traits (explained in section 3.1.2), both in terms of limits and in the rate of change.

The behaviors of the animals were affected by *energy* and *hydration*, as animals needed to search for water and food to stay alive. The behavior outcomes of energy and hydration were different, despite similarities in implementation and execution. Notably, lack of energy became a reason for animals to leave the water sources to explore.

Although not using any scientific formulas for the rate of depletion, both energy and hydration were designed to deplete faster for larger animals and slower for smaller animals [27]. Despite a lack of knowledge on the exact correlation between the rate of depletion of water and body size, it was included for balance between the parameters. Like size, the speed of the animal would affect both energy and hydration depletion at a linear rate, which is realistic in the sense that higher effort spends more resources. With some basis in reality, vision affected the rate of depletion of energy [28]. To keep the senses balanced, hearing also affected energy.

Contrary to energy and hydration, *reproductive urge* and *age* were not affected by any traits in their rate of change. Furthermore, reproductive urge increased in value only when energy and hydration levels were high, thus only thriving animals could mate. This is loosely related to the fact that healthy animals have better capabilities of reproducing in the natural world. Two animals with maximum reproductive urge were able to mate and reproductive urge was depleted when the animals had mated.

3.1.2 Traits

The traits, unlike the parameters, were set only once at birth. Since the value of the parameters defined the current objective of the animals, the traits affected their ability to fulfill those goals. For the sake of clarity, the traits could be categorized into three sets. Parameter limits, traits relating to the maximum speed (maxSpeed) and the senses. See appendix B for a complete list of the traits.

The first set of traits, the parameter limits, were the simplest and relate to the parameters, which were elaborated above in section 3.1.1. These were aptly named *maxEnergy*, *maxHydration*, *maxReproductiveUrge* and *ageLimit*. The first three limited the otherwise infinite potential of regenerating the parameters. In the case of ageLimit, when an animal's age reached ageLimit it would die.

MaxSpeed relied on two other traits, *size* and *acceleration*. Unlike the other traits, maxSpeed was the only one derived from a formula (see 3.1) using other traits, and was consequently not directly inherited. In the evolutionary system, acceleration was inherited in lieu of maxSpeed. The reasoning for this was merely to simplify the method of how maxSpeed would otherwise be inherited, due to its reliance on the formula. Size, like acceleration, was factored into the calculation of speed, but

also affected the rate of change in the parameters. Size was the only trait that constrained maxSpeed. The possibility of adding other constraints, such as age or energy, was rejected due to the already complex nature of the simulation that such constraints may have further muddied the trait and parameter relations. Due to maxSpeed and size having the most visually noticeable changes, more focus was put in developing their interactions within the simulation, compared to the other traits. The formula used for maxSpeed was based on a formula in the article "A general scaling law reveals why the largest animals are not the fastest" [29], which was based on empirical data collected from 474 animal species. Using this formula increased the realism of the simulation. The formula was declared

$$v_{Max} = aM^b \cdot (1 - e^{(-hM^i)}),$$

where

$$\begin{aligned} v_{Max} &= max\ potential\ velocity \\ M &= body\ mass \\ b &= power\ law\ increase\ in\ speed \\ i &= d - 1 + g \\ d &= muscle\ force \\ g &= muscle\ mass \\ h &= cf. \end{aligned}$$

The constants a, c and f had little to no description in the article of what they represented, and were therefore not defined further. Due to the linear effect of a on the formula, and a desire to have a trait that could effectively modify v_{Max}, it was dubbed as acceleration, despite that being a misnomer. Because the animals did not have any defined mass, M semantically referred to their size instead. However, with the constraints given by the article, the curve plotted from v_{Max} never had the hump-shape as shown in the figures based on their collected empirical data. Since it was desired for the simulation to include a noticeable impact of size on the speed of animals, both positive and negative, some modification of the formula was necessary. Furthermore, the constants, which were derived after some testing to find suitable values, were applied and renamed to correspond to their respective traits as following: $v_{Max} = maxSpeed$, $a = acceleration$ and $M = size$, and gave rise to the final equation

$$maxSpeed = acceleration \cdot size^{0.4} \cdot (1 - e^{(-1.5 \cdot size^{1.34})}) - size^{1.34}. \tag{3.1}$$

The last category of traits, where those used for the senses, named *viewAngle*, *viewRadius* and *hearingRadius*. ViewRadius and hearingRadius both affected the range of sight and hearing, whereas viewAngle defined the angle of the animal's field of view.

3.1.3 Senses

Animals were designed to perceive their environment through *vision* and *hearing*. In terms of implementation, both vision and hearing worked similarly through scanning

for targets within a sphere, then iterating through the resultant list to decide each target's type, be it another animal, a plant or a water source. To differentiate the senses, and to keep them realistic, distinct constraints were outlined for each sense. Vision was limited by the viewAngle, and to not see through objects or terrain. Animals were able to notice not only other animals, but also plants and water. Hearing, on the other hand, was not limited by any angles, nor by terrain or objects, but could not detect plants. To detect plants, animals therefore had to use their vision. Hearing thus functioned primarily as an aid for the predator-prey interactions, and for animals to find potential mates.

3.2 Behavior

To quickly achieve basic behavior of the animals and in the interest of prototyping, it was initially decided that a reactive AI was sufficient for behavior. Within this category there exists a handful of solutions (described in section 2.1.1), one of them is an FSM and another is *behavior tree*. There exists both free and paid (visual) tools for behavior trees, and for FSM there is even a built-in tool in Unity called the Animator. While behavior trees could host more complex behavior, state machines were easier to implement in code, which was a priority. The built-in tool for state machines was found to not allow for a suitable degree of control, thus a code-based FSM implementation was chosen.

3.2.1 FSM Implementation

The implemented FSM was altered from standard practice to suit the needs of the project. According to common practice, transitioning from one state to the next should be handled in the current state. For this project however, an alteration of this approach was needed, which included separating most of the decision-logic (which state to switch to when) from the FSM to create modular and exchangeable decision making (see figure 3.2). The main reason for interchangeability was to allow exploration of different decision models (Decision Makers), while minimizing the development time needed for such exploration.

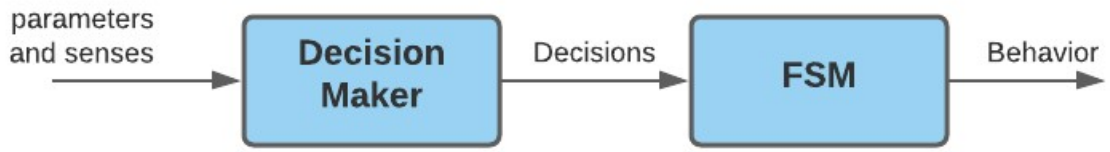


Figure 3.2: Modular decision making. The Decision Maker parses parameters and senses into decisions which are passed to the FSM. The FSM then acts based on the decisions, producing behavior.

While the decision making had been extracted from the states of the FSM, a requirement function was introduced to each of the states. This requirement function governed whether the state could be entered and was dependent on the parameters

and observations of the animal. Once entered, the states would then carry out its associated action. Despite the decoupling of transition control from the states, some transitions were still being made directly from states. Sometimes the animal would forcibly be put in the next state bypassing the decision making module. This was only due to somewhat poor definitions as two states could be joined together.

Initially, the decision maker was rule-based as it parsed the parameters of the animal, the current FSM state and its perception of the environment gathered from its senses. The parsing meant manual discretization of these values into simple yes/no statements that were then asked in order of prioritization. When an animal was deemed as both hungry and thirsty it would prioritize finding water over finding food due to the manual specification of such.

As the decision-maker had the added effect of decoupling states, the behavior model used could hardly be called an FSM at this point since one of the main characteristics of an FSM is that the states are tightly coupled (see chapter 2.1.1). Furthermore, to explore a more advanced solution, it was decided that another decision model was to be used altogether. The implementation of ML behavior was opted for as the prospect of applying ML behavior to animal behavior was deemed interesting to the team. There also existed sufficient tools and similar efforts using these tools, which produced behavior that could fit within a plausible ecosystem[1]. The usage of ML to simulate the behavior of animals in a graphical ecosystem simulation was to our knowledge also novel.

States

To better distinguish between the various actions that the animals could take, they had to enter a corresponding state before making the respective action and each state represented one distinguished action. The states encapsulated the essence of the actions, which would provide clearly defined boundaries, in terms of both code and simulation.

Firstly, there were three states that defined animal movement toward a specific target, *GoToFood*, *GoToWater* and *GoToMate*. These states depended on the target type, which could be either water, food (prey or plant), or a mate. The states were considered different enough in implementation to be kept separate, however, from a software perspective these could with benefit be joined together. Additionally, there were three states that represented what would happen when the animal reached its target: *DrinkingState*, *EatingState*, or *MatingState*. The motivation behind separating these from the GoTo states was also lacking and these could likely have been joined together with their respective GoTo state.

Two other states were the *Fleeing* and *Dead* states. The fleeing state would be

[1]Example ML behavior: `https://www.youtube.com/watch?v=la9ZjD3YXdA` and `https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Examples.md#food-collector`

entered when an animal saw a predator. For the sake of simplicity, no predator would flee from or hunt its own species, and all prey would flee from all predators. Due to their size, bears were able to hunt wolves, thus the wolves would flee from bears. The Dead state represented death and was an absorbing state, thus it was not possible to enter another state once the Dead state had been entered.

Finally, *Wandering* and *Waiting* were the two states that an animal could enter when no other state would be possible or allowed. An animal would usually enter the Wandering state when it was currently satisfied enough to not want to eat, drink or mate. It was meant to allow for the animals to spread out somewhat, rather than staying in one area. If an animal was not satisfied and there was no food, water, or mates available, the Wandering state would also function as a pseudo searching state. The likelihood of the animal finding what it wanted could therefore improve if it explored further away. For simplicity, it was decided that none of the animals would have a specified gender. This way the animals only needed to find another of their species to be able to mate. To prevent cases where the animals would impregnate each other, only one animal was permitted to be in the mating state when they were mating. Thus, to keep the non-mating animal from otherwise wandering off, it would instead enter the Waiting state. It would wait for the duration of the mating before it returned to the Wandering state.

3.2.2 ML-Agents Toolkit

To explore other solutions and introduce a level of flexibility into the behavior, an ML model was implemented. To implement ML in Unity, the open-source package *ML-Agents*[2] was used [30]. The ML-Agents toolkit made it possible to use Unity as an environment for the training of RL models in Python. Additionally, the toolkit provided state-of-the-art RL algorithms and an easy way to interface between the algorithms and Unity. The reasons for adopting this toolkit in this project therefore included:

- Faster setup time than implementing a custom solution. Alternatives would have been to write our own package directly in C# or our own communicator between Unity and Python. These approaches would have put the focus more on the implementation of such functionality rather than applied training of ML agents. The results in terms of functioning behavior was prioritized, so the ML-Agents toolkit was favored.
- Good documentation with references to scientific articles. This meant being introduced to RL in a top-down manner.
- Possibility of diving deeper. The toolkit provides the possibility to use custom-written RL algorithms in Python, which allows for customization later down the line once the learning process works in general. This meant that the toolkit would not inhibit progress towards more custom RL solutions.

[2]The toolkit's repository can be found here: `https://github.com/Unity-Technologies/ml-agents`

Despite many advantages, there was also some limitations of the toolkit. Examples of disadvantages include: exclusively run-time training, which resulted in less flexible behavior, and general communication with the toolkit interface.

ML Algorithm

As mentioned above in section 3.2.2, the main focus of implementing ML behavior was practical application of RL to achieve acceptable results. The specific algorithms used behind the scenes were therefore less prioritized and were subsequently explored only at a shallow level. The ML algorithm used was treated as a *black box* (see figure 3.3).

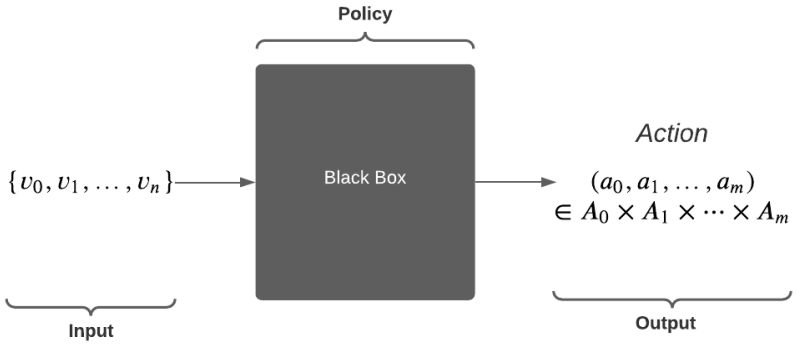


Figure 3.3: The black box perspective was applied during ML implementation. Input was fed into the black box that produced actions as output based on the policy. Heavily inspired by [31, p.5].

Figure 3.3 shows how the algorithm was omitted from close analysis, but was instead simply applied. The black box accepted inputs in the form of numbers or vectors ($\mathbf{v_n}$) concatenated into one vector, after which the policy parsed the input into actions. An action (a_m) was a value and each value was bounded by a set of possible values declared in a set of actions (A_m). Subsequently, the tuple $(a_0, a_1, \ldots, a_m)$ represented a combination of actions drawn from all possible combinations described as the Cartesian product of all sets of actions $A_0 \times A_1 \times \cdots \times A_m$.

The black box perspective was applied during inference as well as the training of the policy. This shifted the focus of the implementation more toward practical aspects of RL, such as the reward strategy, observations and training environment. Conclusively, this meant that more time could be allocated towards achieving acceptable behavior which is covered in chapter 3.2.3.

3.2.3 Machine Learning - Prioritization Behavior

The first clear step to introducing ML behavior was to exchange the reactive decision making module with an ML decision maker. The goal of this ML model was to teach it to prioritize certain actions, depending on its parameters as well as its observations about the environment. The actions would still be static and predetermined which included predetermined movement patterns. In the context of ML and

RL a state concerns the agent's parameters and observations at a point in time, and an action is what the RL model produces from the state. The goal was to still rely on the states of the FSM to represent actions, but transitions between them would be handled by the ML implementation.

Implementing an ML behavior model that performed better than, or on par with, a well-planned reactive model proved to be a difficult task. In addition to learning and working with the toolkit, there were many RL related hurdles to overcome to eventually arrive at a desirable behavior such as: proper reward strategy, providing proper observations, training environment definition and bugs in the simulation.

Rewards

The reward strategy was crucial to the training in order to steer the agent towards desired behavior. Later, this gave rise to a problem, since the model would only learn based on what it experienced. For instance, not fleeing upon the presence of a predator had to be penalized directly instead of relying on the implication that nearby wolves meant imminent death for the rabbit. The magnitude of the rewards also played an important role to the outcome of the learning. If the rewards or penalties were set too low or high, undesirable behaviors would occur. The final reward strategy that produced a desirable behavior was designed as follows:

- Each decision tick (continually occurring process of observation, decision and action):
  - Not fleeing while perceiving a predator was penalized by *-1*.
  - Being alive was penalized as described in (3.2) below.
- Successfully mating was rewarded once by *+2*.

$$-\frac{(hunger + thirst)}{maxAge} \tag{3.2}$$

The penalization of not fleeing when in danger was intentionally very high to dissuade any attempt at doing anything but fleeing in dangerous scenarios. The penalization of being alive is a function of hunger and thirst. Therefore, lower hunger and thirst would lower the penalty, which would encourage being satiated in terms of hunger and thirst. However, being alive was still penalized to induce urgency of completing the task of mating. If being alive was rewarded instead, the rabbit could bypass the goal of mating and just try to stay alive as long as possible.

Observations

The observations provided to the model were also critical to the learning outcome. For instance, providing unnecessary information would impact the learning speed as it would clutter the input with noise, and it could even prevent learning altogether. Another important aspect of the observations was to ensure that enough information was provided such that the agent would not get surprised by for example occasionally receiving fewer rewards without adequate reasoning from the observations. This

too could confuse the model leading to unsatisfactory results.

Another aspect of the observations to consider was normalization. Normalization means to scale the input to within the range of $[0, 1]$, which ensures each input value will be of the same magnitude. Normalization reportedly speeds up the training of neural networks and by extension the model used in the ML-agents toolkit [32].The final observations that produced a desirable behavior was as follows:

- Energy percentage (E for energy) calculated as $\frac{currentE}{maxE}$.
- Hydration percentage (H for Hydration) calculated as $\frac{currentH}{maxH}$.
- Boolean values (One for each) corresponding to all requirements for entering the following states being met or not:
  - goToFoodState
  - goToWaterState
  - goToMateState
  - fleeingState

Environment Efficacy

Closely related to the above points was the efficacy of the simulation itself. Minor bugs in the behavior of the animals or the environment such as faulty navigation, perception etc. could critically affect the learning process. For instance, if the wolves were non-hostile due to them not perceiving the rabbits, it would result in the rabbits not being afraid of the wolves. Furthermore, if the rabbits themselves were unable to eat food at certain times due to some bug in the simulator, this would send the message that eating might not have been worthwhile. Such bugs can be viewed as observations not appearing to the agent, which was as mentioned detrimental (or outright fatal) to the performance of the model.

Training Environment

The training environment was crucial, as successful training relied on the presence of a controlled training environment that was defined in a way that facilitated learning of the wanted behavior. For instance, the training environment was initially very similar to the simulation environment. This required multiple rabbit and wolf agents with reactive behavior that interacted within a small and flat environment. This environment failed to train the rabbits likely due to many reasons. One such reason was the aforementioned problem of wolves not acting as expected, which might have introduced uncertainty into the training. Furthermore, it proved difficult to train multiple agents that interacted with the same environment and each other. This might be explained with the rabbits competing for food as well as unpredictable behavior in the agents' mating partners, thus introducing uncertainty of the value of food and mating respectively.

Training an agent on a completely static environment would likely result in desired behavior in precisely that environment, however, the agent would fail to generalize to other environments. This issue is generally known as *overfitting*. Parameter randomization is the process of introducing variability to the agent's environment during training which is based on the concept of "domain randomization" [33]. This way, the agent learns not only how to handle a specific type of environment, but becomes adaptable and flexible enough to survive in many types of environment, thus reduces overfitting. Throughout the project, multiple forms and levels of parameter randomization were tested. The earlier attempts randomized amounts and positions of food, water, wolves and rabbits as well as environment size. Although, the final successful training environments randomized much less (see figure 3.4).

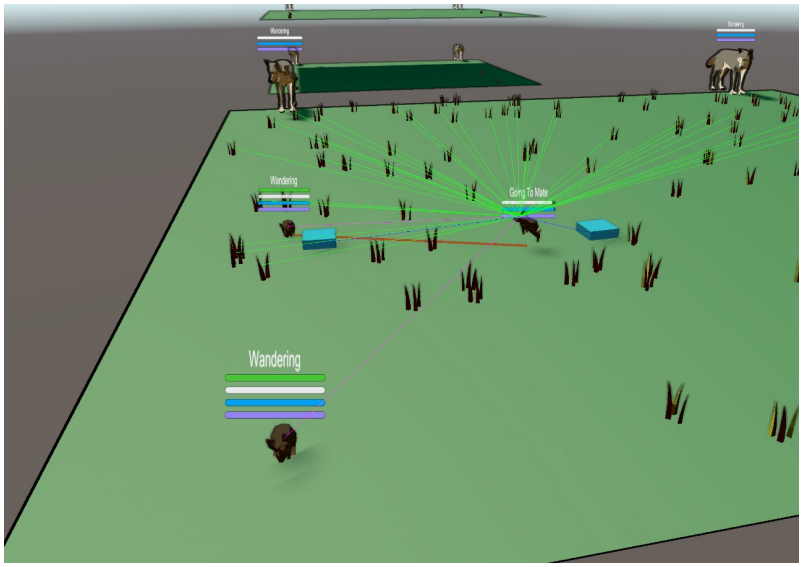


Figure 3.4: The final training environment for the ML prioritization. Two wolves and four rabbits (one outside the image) have fixed positions and only one rabbit agent is training in each environment. Multiple environments are running simultaneously. Lines represent what the rabbit perceives: food (green), water (blue), mate (purple) and the red line shows the path that the animal is following.

The final environment included one rabbit agent to be trained with randomized starting position, a fixed size environment, two stationary non-agent wolves, two stationary non-agent rabbits as well as randomized food and water positions. Furthermore, multiple environments used to train at the same time sped up training. The final environment was in conclusion very controlled which might prove, similar to the reward system, that simplicity is a winning concept when training ML agents.

An *episode* is the period from which an agent enters the training environment until it achieves its goal or fails. In this context, this refers to either mating or dying, respectively. Every episode began with populating the environment with food and water as well as moving the agent to a random position. At the end of each episode food and water was removed to be re-spawned next episode, and the agent's parameters were reset.

The ML model that was trained on a rabbit agent was subsequently applied to all of the animals. This was possible since the wanted behavior only differed in details, for instance, a rabbit walks towards its food since mushrooms do not run away, while a wolf has to chase its food. Such dissimilarities were instead handled in the actions and prioritization is handled the same between animals.

3.2.4 Machine Learning - Steering Behavior

An alternative approach to ML behavior was also explored, taking the prioritization approach one step further. This replaced the predetermined movement patterns for most of the actions in the previous approach. The idea was to train an agent to not only prioritize actions but also choose where to move. Subsequently the agent would interact with objects and other agents upon collision. The objective of the specified approach was to have an agent that would move to food, water and potential mates depending on internal observations such as energy, hydration and reproductive urge. Additionally, all external observations such as perceived entities from senses were considered. The only exception was fleeing, which still used predefined movement due to difficulties in training the agents to flee properly.

The steering approach to ML proved a great challenge. While the animals could learn the desired behavior in the training environment, they failed to generalize to the final simulation. The implementation of the approach is nevertheless covered in appendix C for the interested.

3.3 Evolutionary System

In order to model an evolutionary system in the simulator, animals needed to apply genetic operations upon reproduction. The fundamental principles of GAs were utilized to accommodate the need for this. Henceforth, selection was not implemented as a concrete algorithmic step, but rather represented animals viability to be selected by a partner within the observable area, with the constraint of both animals having maximum reproductive urge. Reproductive urge could only be gained while having high energy and high hydration as explained in 3.1.1. As a consequence, fitness would be associated with how well the animal fared in the environment. An animal that would eat, drink and survive effectively would also live longer, thus having more survival time leading to a higher chance of reproduction and in turn higher fitness. In other words, it corresponds to the simulated biological fitness, making the evolutionary system model mimicking natural evolution.

Crossover was implemented such that it genetically favors the parent with highest age at the time of mating. The motivation for introducing this bias was that if parent 1 had lived longer than parent 2 it would be an indication that the child would benefit more from parent 1's genes. As this differed from traditional ways of implementing crossover such as basic uniform crossover and one point crossover (see section 2.2.2), a uniform implementation with bias towards the parent with

longest survival time would potentially speed up convergence towards an optimal set of genes. The implementation of crossover is shown in listing 3.1 below.

```
Traits Crossover(Traits p1Traits, Traits p2Traits,
    float p1Age, float p2Age)
{
    Random rng = new Random();
    Traits childTraits = p1Traits.DeepCopy();
    float totalAge = p1Age + p2Age;
    float threshold = p2Age / totalAge;

    Type type = childTraits.GetType();
    foreach (PropertyInfo info in type.GetProperties()){
        double rnd = rng.NextDouble();
        if(rnd < threshold){
            info.setValue(childTraits, info.GetValue(p2Traits));
        }

    }
    return childTraits;
}
```

Listing 3.1: Implementation of crossover algorithm with bias towards the oldest parent. Note that Traits is a chromosome encoded as a class type with each instance variable corresponding to a gene.

Mutation was also implemented to maintain variety in traits. The implementation is shown in listing 3.2 and is inspired by the non-uniform mutation approach for floating point values described in section 2.2.3.

```
void Mutation(float mutationProbability, Traits childTraits){
    Random rng = new Random();
    Type type = childTratis.GetType();
    foreach (PropertyInfo in type.GetProperties()){
        double rnd = rng.NextDouble();
        if(rnd <= mutationProbability){
            float currentGene = (float) info.GetValue(
   childTraits);
            float mutationPercentage = Mathf.Clamp((float)
                SampleGaussian(rng, 0, 5)/100f, -0.1f, 0.1f);
            float mutatedGene = mutationPercentage * currentGene
                + currentGene;
            info.SetValue(childTraits, mutatedGene);
        }
    }
}

double SampleGaussian(Random random, double mean, double stddev)
   {
```

```
18      double u1 = 1 - random.NextDouble();
19      double u2 = 1 - random.NextDouble();
20      double xNorm = Math.Sqrt(-2 * Math.Log(u1)) *
21          Math.Cos(2 * Math.PI * u2);
22      return xNorm * stddev + mean;
23  }
```

Listing 3.2: Implementation of mutation using Gaussian sampling (with mean 0 and standard deviation 5) for mutating the gene's value.

3.4 Environment

This section explains two essential factors of the environment for animals. This includes the landscape with different features such as trees and other obstacles, as well as variable land height. In addition to this, there are also plants present, which are modeled according to simple rules.

3.4.1 Resources

To survive, the animals required a source of food and water. Water was distinguished between the purely visual and the invisible water sources, to provide the animals with places to drink. Water was an infinite resource, therefore, any number of animals could drink from it without depleting it. Since animals drinking from lakes has a negligible effect on water levels in real life, accordingly, infinite water approximates reality in this regard. The water was thus a very static aspect of the resources, while vegetation was more dynamic.

Since vegetation in nature can deplete when there are too many animals in one area, it was decided to be modeled more dynamically than water. There were two types of plants in the simulation, mushrooms and grass. Mushrooms had a nutritional value, age and a small chance to spread, while grass spawned continuously from water sources. Mushroom's nutritional value increased with age, and would only spread if above a certain age. If it were eaten, its nutritional value would reset to zero, and it would regrow some time later at the same spot. The mushrooms were also limited in density to avoid exponential spread. While the constraint put on mushroom expansion lowered the theoretical maximum for the plant availability, the never-ending supply of grass raised the practical minimum. This had the effect of decreasing herbivore population extinction rate.

3.4.2 Terrain and Landscape Generation

Understanding how the terrain affected the ecosystem was an important part of this project. The terrain generation in this report consisted of four separate parts, heightmap generation, mesh generation, water generation and object placement. During the early stages of the development of the heightmap generation and mesh

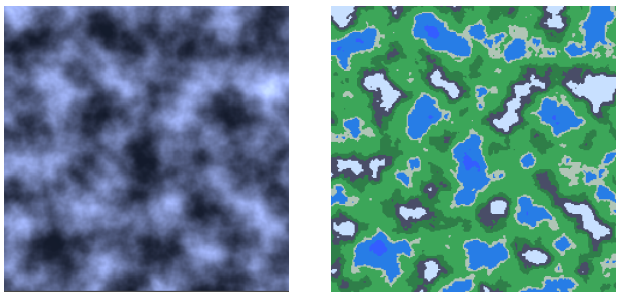
generation, the implementations were based on Sebastian Lague's tutorial series *Procedural Terrain Generation*[3].

3.4.3 Height-Map Generation

The basis of the terrain generation system was the heightmap generator. Heightmaps were generated with the built-in Perlin noise function in Unity, which generated a value based on x and y coordinates. The heightmap could be modified with the following parameters:

Scale: Applies a zoom or modifies the distance between the probed coordinates.
Octaves: The number of octaves that will be layered into the heightmap.
Persistence: The factor of the value that is carried over from the lower octave, adjusts amplitude.
Lacunarity: Adjusts the amount of detail that is added or removed each octave, effects frequency.
Seed: A seed for the Perlin noise function.

A visualization mode for a colored texture based on the heightmap was created. The mode consisted of a list of colors and thresholds for these, which allowed for editing the cutoffs for the different colors. Figure 3.5 displays a comparison between the value map that comes directly from the noise function and the colorized and visualized map.



(a) Value map **(b)** Colorized Map

Figure 3.5: This figure shows the visualization of the heightmap in two different ways. Figure 3.5a is where every height is represented by a value between 0 and 1, where 1 is white and 0 is black. Figure 3.5b visualizes the heightmap by colors to present a more appealing and understandable image.

3.4.4 Creating a Mesh

The first step in creating the environment from the height map, was to create a mesh that matched the selected settings. This mesh consisted of multiple vertices connected into triangles, which was calculated by first creating a grid of points, or vertices, that would then be connected into triangles. Furthermore, a normal vector

[3]Sebastian Lague's tutorial series:
`https://www.youtube.com/playlist?list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3`

was calculated for each vertex depending on the neighboring vertices. All this data was then used to apply a mesh to a Unity game object.

The necessity of the following step in the mesh generation process arose from a limitation in Unity where meshes could only contain roughly 64000 vertices. To create maps with larger sizes a technique called chunking was used. With chunking, the map was split into manageable segments called *chunks*, each with their own mesh. Connecting these chunks allowed circumvention Unity's vertex limit and permitted the creation of larger environments. Dividing the terrain into chunks created some issues with both the height-map generation and the mesh generation, where the edges of the meshes generated did not connect and the normal vectors of the border vertices were calculated incorrectly. The first issue was solved by adding global normalization that normalized over all chunks instead of in each chunk, and the second issue by adding another set of border vertices that were only used for normal calculations.

A stylistic choice was made to have the terrain be flat shaded. This provided the restriction that flat shaded chunks needed to be smaller than non-flat shaded chunks as flat shaded chunks needed more vertices while still limited to 64000 vertices.

3.4.5 Coloring the Mesh

Without colors or textures, the environment would look very bleak and unnatural. A simple method used for coloring the mesh was to color based on the height of the environment, similar to the heightmap visualized in figure 3.5. Another method more difficult to implement was a slope based coloring algorithm, that would set the color of the mesh based on the slope of the terrain. The first method was chosen as other parts of the project needed a higher level of prioritization than coloring the map. With the height mesh method, blending between the color layers created more realistic transitions between the layers. Texturing and coloring the map preceded through many iterations. The methods were divided into two sections, texture-based coloring and shader-based coloring.

Texture-Based Coloring

The first method implemented was built on techniques for visualizing noise- and height-maps, see figure 3.5. This simple method consisted of rendering the color map, see figure 3.5b, straight on the mesh. Coloring the mesh in this way created results where the height levels would look a bit jagged, and the pixels from the visualization texture would be clearly distinguishable. An example of this type of color rendering can be seen in figure 3.6.

Figure 3.6: Early version of the environment generation with the simple coloring based method on the height-map visualization texture.

The main problem with this method of coloring, was the visibility of the pixels and the lack of smoothness, which resulted in an unnatural look. Blending between the different height levels was difficult, as the coloring worked with large pixels. Therefore, this method was scrapped, and a shader-based implementation was researched instead.

Shader-Based Coloring

Unlike the height-map visualization, the shader-based coloring method did not depend on the heightmap data. The shader-based approach depended on the environment height of each point on the mesh, instead of the generated height value in the heightmap. This resulted in coloring that was not pixelated based on the height-map, and instead contained straight horizontal dividers between the different terrain types, as can be seen in figure 3.7.

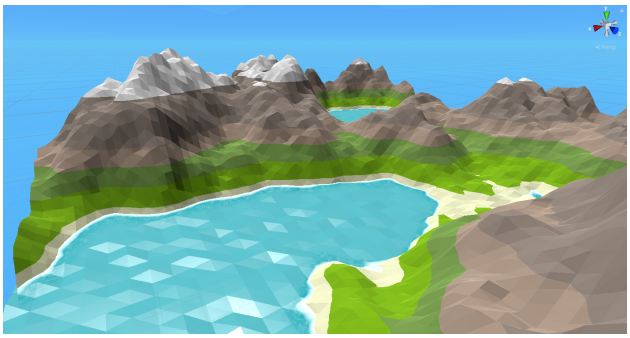


Figure 3.7: Early environment-generation with the first version of the shader colorization.

A shader-based approach solved both the smoothness and the pixelation problems of the texture-based colorization. With blending of the layers an even, natural look was achieved, which was implemented in the final version of the simulator.

3.4.6 Water System

Water was an essential part of an ecosystem, and there were many different ways water could be implemented. The chosen method for this simulation consisted of

two parts: the water plane, and the water source blocks. The water plane consisted of a visual water cube, where the bottom was always set to the bottom of the environment and the top set to the selected water level. This part was mostly visual, except for its interaction with object placement and navigation calculations. A more technical part of the water system was the water source block placement. The animals in the simulation were dependent on drinking water, and they looked for a game object with the tag *Target*. Placing this tag on the visual water would have lead to the animals looking for the center of this block instead of finding the edge of the water to drink. The solution was to place small invisible cubes, with the correct tag, on vertices that were within a selected distance of the water level.

The water plane described in the previous paragraph was shaded with a very simple opaque shader with a turquoise color. Shading the more advanced water cube required a more advanced shader. This shader was implemented based on a tutorial by *Binary Lunar*[4] and provided transparency, refraction, coloring and foam shading. The result of this shading can be seen in figure 3.8.

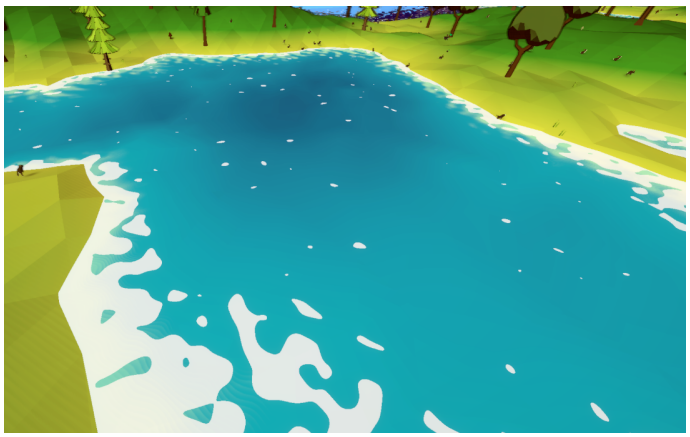


Figure 3.8: The water shader.

3.4.7 Object Placement

When designing a solution to the problem of placing trees and animals on the terrain, a generalized solution was conceptualized. This generalized solution would be able to place every object that needed to be placed in the game. The method was built on the Poisson set distribution, explained in section 2.3.2, and allowed placement of all different types of game objects.

Firstly, the algorithm described by Tulleken was implemented to generate a set of plausible points [23]. On these points, the objects were instantiated around ten units above the max height of the environment. A ray was emitted from each placed object and if the hit surface was tagged as ground, not water. If the hit surface was not ground, the placed object would get destroyed. The objects that were over ground got their y-coordinate set so the object was placed on the ground. An interesting

[4]Based on source: `https://www.youtube.com/watch?v=MHdDUqJHJxM`

observation was that when placing an animal with a navmesh agent component, instead of just placing the object, one had to call the warp function on the agent to make sure that the agent knew it was on the navmesh.

The distortion of the placement with noise mentioned in section 2.3.2 was not implemented, but could be an interesting improvement to the object placement implementation. One limitation of this approach was the fact that it was not possible to set the number of animals, only the distance between the spawned animals. Another possible improvement would be to make sure that objects were unable be placed on top of each other.

3.4.8 Navigation

The navigation system chosen for this project was Unity's built-in navigation system in combination with the NavMeshComponents[5] package from Unity's GitHub page. Navigation was implemented by adding a game object with the NavMeshSurface component attached. When the environment had finished generating, before the object placement, the navigation mesh construction function would be executed. The navmesh needed to be generated before the object placement, as the object placement needed to warp agents to attach them to the navmesh. An improvement to code design would be to place all the objects and subsequently generate the navmesh, and finally before the simulation start, warp on all the agents in the scene. This was not implemented due to time constraints in the later parts of the project.

When implementing the navigation system, many obstacles were discovered that limited the possible size of the environment. It was noticed that when increasing the environment size, or adding more chunks, the simulation would take too long to start. Using profiling, the cause was found to be the build time of the navmesh. One method tested was to modify the agent settings. Increasing the radius of the agent decreases the resolution of the navmesh, which possibly could increase performance, but no significant increase was noticed. The second and more plausible method of allowing larger environment sizes was to implement a system to pre-generate the terrain. This would result in that only the animals would be placed at run-time. This solution was implemented and researched but not finalized, although the solution was plausible. The last, untested, method was to exchange the built-in Unity system for some other navigation system. This could have been the best solution but due to time constraints this was not possible.

3.5 Statistical Calculations

Since interactive software was emphasized in the project purpose, accurate measures of statistics were needed. This implied data structures to collect, store and present

[5]Github page for NavMeshComponents: `https://github.com/Unity-Technologies/NavMeshComponents`

the data. This was done by recording data on the births, deaths of animals, in addition to sampling data each minute. Animal traits, generation and species birthrate were reported on birth. Age, distance travelled and cause of death were reported on death. Birthrates and population amounts per minutes were sampled from this data each minute. When data was recorded, the corresponding structure for each species and generation of the entity updated its mean and variance values using Welford's method, which is explained in section 2.4. The algorithm was implemented in C# as shown in listing 3.3.

```
1 NewMeanVaraiance(float m, float s, float n, float valueToAdd)
2 {
3     if (n <= 1) return (valueToAdd, 0)
4     float oldM = m;
5     s = s * (n - 2);
6     m += (valueToAdd - m) / n;
7     s += (valueToAdd - m) * (valueToAdd - oldM);
8     return m, s / (n - 1)
9 }
```

Listing 3.3: Implementation of the Welford algorithm to calculate the mean and variance as each measurement arrived from the simulation. The variables m, s and n are the old mean, the old variance and the current population size, respectively. The method returns a tuple of the new mean and the new variance.

The algorithm in listing 3.3 was useful, since only the relevant measurements needed to be stored. This simplified implementation and increased performance as the statistics were calculated with constant complexity.

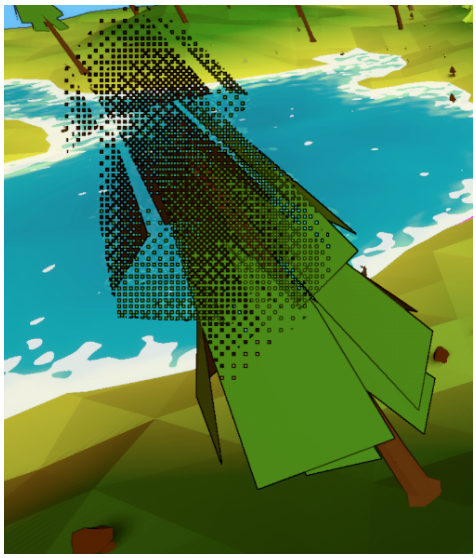
The structure for storing statistics was mostly implemented as C# lists, where each species was assigned a list for mean and a list for variance, and each index in these lists corresponded to a generation. This implementation was useful since the presentation-class for data used C# lists for input. Furthermore, methods were implemented to export the collected data to structured JSON text files. This was practical if the user would want to analyze the data to a greater extent than what is possible in the built-in functionality of the graph.

3.6 Graphics and Visualization

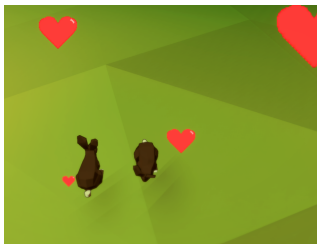
To improve the feel of the simulator, which was explained in section 1.1, some work was done to the graphical representation of the simulation. 3D models and animations were provided to vastly speed up development time, however, other graphical improvements and effects were made such as: creating a toon shader[6] (flat color shading and hard shadows) for the animals and trees, a shader for the water (figure 3.8), a transparency effect for the trees (figure 3.9a), a simulated wind sway for the trees[7] and some post processing effects. The post processing effects were

[6]Based on source: `https://www.youtube.com/watch?v=3SvyJrENsgc`
[7]Based on source: `https://www.youtube.com/watch?v=ZsoqrHHtg4I`

primarily used for a depth of field effect and to unify the tone of the colors in the scene. These effects were provided by Unity as a part of its post processing effects system.



(a) Transparency based on the camera's proximity.



(b) Particles indicating mating.

(c) Particles indicating pregnancy.

Figure 3.9: Examples of the graphical effects implemented for the simulation. 3.9b and 3.9c show the particles created by the team.

Some further work was also done using Unity's particle systems. These particle systems were used to indicate some of the animal states. The states that were given a particle system were: mating (figure 3.9b), fleeing and death. The particle systems were also used for situations that needed indication, specifically when an animal was pregnant (figure 3.9c), got attacked by a predator and when born. The particles in the particle systems for mating and pregnancy were custom made by the team using pixel art tools, and the pregnancy particle differed for each animal species. Aside from those particle systems, to avoid spending too much time on aesthetic work, the other particle systems used (fleeing, death, getting hit, birth) were third-party assets[8].

3.7 User Interface

As the purpose of the project is to create a simulator usable to others, a UI was necessary. This was achieved with a graphical UI, GUI, to make interaction for almost anyone. When starting the software a simple main menu appears hosting 3 options: Play, settings and quit. Pressing quit exits the application while pressing settings will take the user to a settings menu. In the settings menu the user can adjust the sound volume as well as select whether animal parameters should always

[8]Source: `https://assetstore.unity.com/packages/vfx/particles/cartoon-fx-free-109565`

be shown or not. Pressing play navigates the user to the environment creation scene where the user can adjust parameters of the environment to be created. The environment creation menu allows for control over the physical environment as well as the living and non living entities of the simulation (see figure 4.2).

4
Results

The results of the project is summarized in this chapter. As the goals in section 1.2 states, the outcome will partly be presented based on the ability of the simulator to model ecosystem characteristics. Moreover, the interactivity and performance of the software is described.

4.1 Ecosystem Simulation Outcomes

The ecosystem consists of terrain generation and interacting animals, establishing the relations within the simulation. Thus, the results in this section covers how the ecosystem reacts to different environments and animal populations.

The project has produced a considerable amount of possible configurations for an ecosystem. It was thus important to limit experimentation to a few representative tests to evaluate the software. It was deemed that the most interesting concepts to analyze were:

1. A comparison of ML-controlled and reactive rabbits in the same environment.
2. A comparison of terrain impact on survival.
3. A comparison of developed traits between rabbits living with and without predators.

To clarify, in the following sections all tests were limited to 2 hours of run-time. As an ecosystem reached this limit, it was declared to be a stable ecosystem if all the species included in the simulation were alive at the point of termination. Furthermore, to focus the results, the only animals represented in the tests are rabbits and wolves. All references to reactive, ML prioritization and ML steering behavior only refers to rabbits. Wolves were always reactive in these simulation runs. There was also randomness present in the simulation concerning initial placement of animals and plants as well as the movement of the animals. This gave rise to some level of uncertainty in the simulations, which was controlled for by running multiple tests for each configuration and subsequently comparing means. Besides the factor of randomness, the tests were executed as controlled experiments, where only one factor was changed and studied at a time. The plots for the simulations are found in appendix D.

4.1.1 Behavior Performance Results

The survival of the behavior models, which are described in section 3.2, are compared in in this section. Survival is defined as how long, and in what quantity, the species are able to survive. The tests were made to evaluate how well the simulation models animal behavior, as is the first goal of the project (see section 1.2). The parameters for the environment were kept constant between runs, which included: terrain shape, environment size, initial plant availability and initial water availability.

Rabbits only

The first comparison investigated the relative survival achieved with different behavior models for rabbits living without predators present, of which all results can be seen in section D.1.1. The behavior models compared were reactive, ML prioritization and ML steering. All tests were executed separately, with only one behavior model on each run, and with equal seeds. The seed corresponds to the terrain factor for which the terrain shape will be equal on all runs with the same seed. The ML steering rabbits went extinct around the 20 minute mark. The two other models, reactive and prioritization ML, survived until the end. The reactive rabbits appeared to be slightly more populous than the ML prioritization rabbits. The plant population stabilized after 20 to 40 minutes, and appeared to be slightly more populous with ML prioritization rabbits, than reactive rabbits.

Rabbits and Wolves

The second comparison investigated the same configurations as above, but with the addition of wolves. All plots can be seen in appendix D.1.2. Similarly to the tests with rabbits only, the ML steering rabbits went extinct before 20 minutes and the other two models survived until the end of the simulation. The wolves living with the ML steering rabbits also went extinct before 20 minutes. The wolves in the other two tests went extinct around the 60 minute mark. Again, the reactive rabbits appeared to be slightly more populous than the ML prioritization rabbits. The plant populations increased at the beginning, before stabilizing around similar values for all tests.

4.1.2 Trait Evolution Results

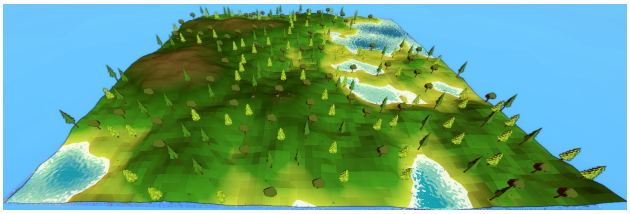
The evolution of traits were tested to evaluate the first goal of the project (see section 1.2). Trait evolution of the rabbits in simulations on seed 0 with only rabbits, and rabbits with wolves are displayed in appendix D.3. The effect that predators may have had on the evolution of the traits of rabbits is compared to when rabbits could develop without the threat of predators. As the first goal in section 1.2 states, finding the perfect combination of traits lies outside of the scope of the project. Hence, merely a select subset of all traits are showcased to display change and tendencies of genes as a result of evolution.

Both max speed and age limit had increasing mean value for both simulations with

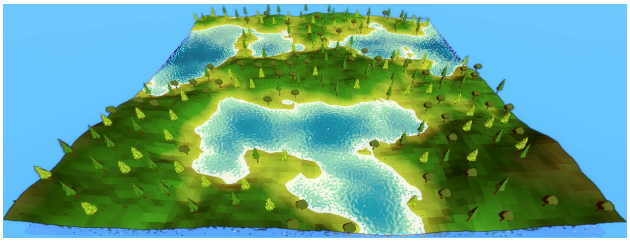
only rabbits, and simulations with rabbits and wolves. The max speed comparison had a few instances of simulations with contradicting values in respect to the mean value. In contrast, the mean max energy remained unchanged, although a few simulations with rabbits and wolves deviated notably from the mean both positively and negatively for greater generations. The mean value of max reproductive urge declined for both types of simulations. Finally, the mean view radius deviated between the two simulation types. The view radius for ecosystems with both rabbits and wolves did not change, whereas the view radius for ecosystems containing only rabbits declined. In essence, dynamic gene alteration between generations was a result of the GA implementation described in section 3.3.

4.1.3 Terrain Impact Results

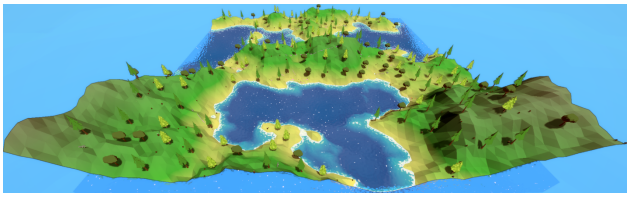
The impact on the ecosystem that different terrain shapes brings, were also investigated. Plots of the results can be found in appendix D.2. Only the seeds and height multiplier was changed between runs, and all other parameters were kept constant. These tests were made to evaluate how the ecosystem was affected by the generated terrain. In other words, whether the second goal of the project was fulfilled (see section 1.2). The examined terrain types were seed 0 (figure 4.1a), seed 32536 (figure 4.1b) and seed 32536 with increased height difference (with height multiplier set to 35, in figure 4.1c). Comparisons were split between differences in seed and height.



(a) Seed 0 terrain.



(b) Seed 32536 terrain.



(c) Seed 32536 terrain with different height settings.

Figure 4.1: Two terrains generated by specified seed in height map settings and a third with more hilly terrain.

The rabbit populations were similar when comparing seed 32536 and 0 in size. The wolf population was more populous and survived on average longer in the environment with seed 32536 than in seed 0. Lastly, the plants in the environment with seed 0 became more abundant compared to plants in the environment with seed 32536. When comparing between height differenced maps, the rabbits population living in the flatter map appear to vary more compared to the population in the HDIFF map. The wolves where more populous, on average, in flatter map compared the HDIFF map where no population made it past 40 minutes. The plant population appeared to decrease after the initial increase. Plants in the HDIFF map was on average more populous.

4.2 Interactivity

The simulation aims to allow the user to create their own ecosystem that requires interaction, which was covered in the third goal in section 1.2. The focus was on giving control, and less effort has been put into achieving high *Usability* as defined in [34].

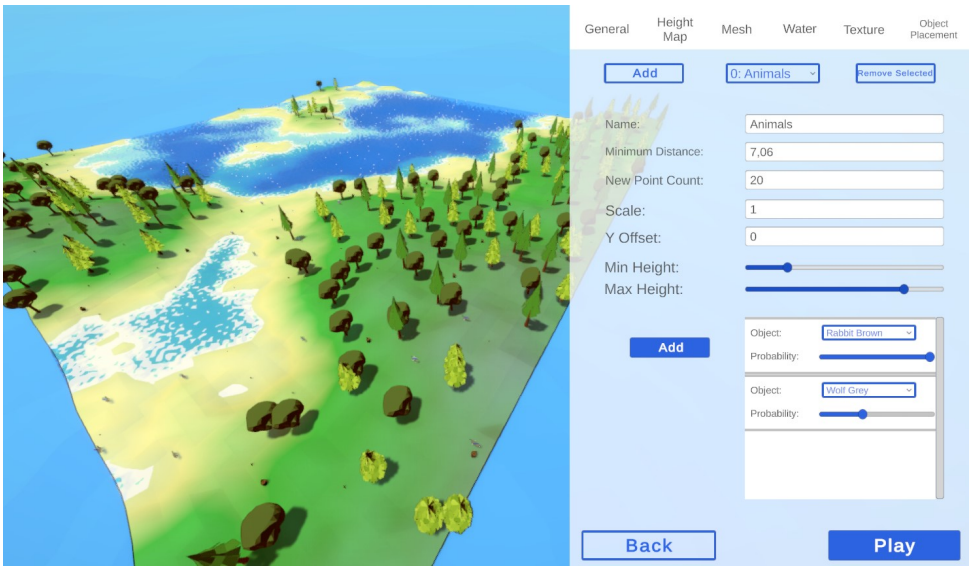


Figure 4.2: The environment creation menu. High level of control is offered although the usability may be lacking.

As can be seen in figure 4.2, there are multiple tabs that contain settings for different aspects of the environment. High level of control is offered, allowing the user to change most aspects of the simulation. Once the user has settled on a configuration for the environment, the simulation can be started by pressing play.

There are a few ways that the user can interact with the simulation at run-time: the simulation speed can be adjusted, the animals' parameters can be viewed as well as the current state of the animal. There is also an in-game graph view, seen in figure 4.3, which allows the user to view how different aspects of the animals in the simulation change through time.

Figure 4.3: The graph view. The user can view graphs based on a selection of statistics.

As seen in figure 4.3, the buttons to the right of the graph allows the user to switch between viewing different statistics. The available statistics are population, animal traits, birth rate and active plants (available plants). Run-time visualization of statistics has also been useful during development, providing a convenient method for both verifying the simulation and adjusting parameters to achieve a stable ecosystem. Nonetheless, the python library Matplotlib was used for displaying results in this report, as it is more advanced.

Performance of Software

Software needs to exhibit, to the user, satisfactory performance for it to be considered interactive. The performance was therefore also tested to measure how well this aspect of interactivity was met.

The performance was evaluated in two categories, compared to the number of active rabbits. These categories are the Frames per Second (FPS) in combination with CPU load and secondly the memory usage. The metrics were measured during the simulation of the stable rabbit- and plant-system. Machine used for the testing used a GTX 2060 with 6 GB VRAM GPU, AMD Ryzen 5 5600X CPU, 16 GB RAM. The results in appendix A show FPS and memory statistics at minute 11 in figure A.1, when slightly less than 700 rabbits were alive. In figure A.2 the test machine provided 60 FPS with some short, periodical, drops to as low as 15 FPS. The memory usage with slightly less than 700 rabbits remained below 1 GB of utilized RAM.

5
Discussion

This section interprets the results as well as discusses other approaches and future improvements.

5.1 Ecosystem Simulation Outcomes Evaluation

This section interprets the results from the tests covered in the result. Firstly, the behavior models are discussed, where the three different models of rabbits were tested with and without wolves in the environment. Secondly, the terrain impact on animal population in environments with rabbits and wolves are discussed. Finally, the evolution of traits are discussed.

5.1.1 Behavior Performance Evaluation

From the results in section 4.1.1 and plots in appendix D.1, the vector steered rabbits performed worst of the three models. This was true both in the case of environments with and without wolves. Furthermore, the reactive model seemed to fare slightly better than the prioritization model. There is however much variation around the mean and from the plots in section D.1 there are multiple simulation in which the reverse is true. It is thus said with caution that the reactive and prioritization models performed somewhat equally.

The wolves, all using the same model (reactive), went extinct before the end of the simulations as seen in figure D.4. Rabbit populations did not seem notably affected by wolves as time progressed. One exception is the first 20 minutes where the rabbit population size was slightly lower in those simulations with wolves, indicating a weak negative relationship between the two populations. However, the goal of the project was never to fully create an ecosystem in equilibrium so this is not a fundamental problem in the simulator. There are probably environment parameters which generate stable ecosystems over long time, however, this was outside the scope so no results explicitly expected this.

The goal of modeling interacting animals was achieved in the sense that at least two of the approaches produced rabbits that could interact and survive reasonably well. While the reactive behavior produced the highest survival in rabbits, the ML prioritization approach was comparable. Further exploration of this approach or the finalization of the ML-Steering approach might give interesting results in the future,

contributing to a more interesting or plausible simulation.

5.1.2 Trait Evolution Evaluation

The dynamics of traits highlighted by section 4.1.2 entailed some interesting observations. The result of maxSpeed indicated a general importance of higher max speeds relative to the starting value, as it increased in both rabbit simulations with and without wolves. This seemed to be the case despite the parameter currentSpeed having an inverse effect on energy consumption as described in 3.1.1. A possible cause being that quick rabbits arrive at food earlier and thus survive longer than its competitors. The same seemed to be indicated by the ageLimit trait change. Although ageLimit didn't have any direct or indirect theoretical disadvantages for higher values, which could potentially be the reason to why almost no simulation showed a decreasing trend regarding ageLimit. An additional interesting observation was the discrepancy of viewRadius between the two types of simulations. With the selective pressure of wolves, rabbits could potentially be required to retain their visual radius value to avoid stumbling across a predator, while the same selective pressure could be absent in an environment with solely rabbits. The effect of increased energy consumption could potentially be the reason for why the visual radius mean didn't increase uncontrollably over generations. As visual radius becomes higher in value, more energy is drawn, which is outlined in 3.1.1.

Despite results showing what is supposed to be trends of evolving traits, it should be noted that the number generation should be greatly considered before making any assumptions or conclusions about the evolving traits. Traditional runs of GAs usually iterates through magnitudes of greater number of generations before converging in a definitive solution. This process also usually involve visiting multiple local optimum points before convergence, which arguably none of the results presented seems to be doing. Given this detail, the goal of the project was not to find an optimal set of trait values, but simply to showcase a model for evolution which the results satisfies.

5.1.3 Terrain Impact Evaluation

Following the behavior model comparisons, different terrain was studied. In the results section D.2 a comparison between seed 0 and 32536 was studied. In figure D.6 comparing the rabbit populations in these two environments, little difference can be seen. It is different however when comparing wolf populations in figure D.7. In the environment with seed 32536, an environment with more water, the wolf population increased and more than doubled for some time compared to wolf population in the environment with seed 0. This big difference in wolf population did however, have negligible impact on the rabbit population, with only a slight reduction in rabbits between 20 and 40 minutes. Looking at plant availability in figure D.8, plants in the environment with seed 0 does seem more abundant compared to the other environment. This is probably due to larger landmass plants can spread to in this environment, in contrast to greater water abundance in the environment with seed

32536.

Comparing between the two environments with different height multiplier settings (HDIFF) in figure D.9 plotting the rabbits and figure D.4 plotting the wolves the rabbit population living in the HDIFF environment appears more stable. This could be explained by studying the wolf population, where it was significantly more populous in the flatter environment. This could indicate difficulty for wolves hunting in environments with higher height multiplier, one cause being reduced vision range as this is limited by the ground. Furthermore, weak relationships between rabbit and wolf populations could be established when comparing the environment height setting, since the increase of wolf population in figure D.10 is accompanied by a reduction of rabbit population in figure D.9. There is however no proof of this with causation, only correlation. Plant differences in figure D.11 appear to favor the HDIFF-environment slightly, which could indicate either easier spread in the HDIFF environment or less pressure from rabbits. The latter case could be due to rabbits' restrained vision or constrained navigation in environments with higher height multiplier.

5.1.4 General Comments

One of the main difficulties encountered was to obtain a stable ecosystem with predators surviving throughout the whole simulation run. As the simulation runs were heavily influenced by randomness in for instance wandering direction, vegetation and many other mechanisms, it proved to be difficult to encompass the desired outcome beyond what was influenced by deterministic variables such as animal traits. This difficulty imposed high volatility simulation results and usually led to not obtaining a fully stable ecosystem for all species involved. This was as mentioned, somewhat controlled for in the results by using multiple simulation with equal input parameters.

Another contributing factor regarding unstable ecosystems with predators was the population size of the predator. Since majority of simulations involving a predator included a predator population of around 5 to 25 units, it was evident that the probability of wolves stumbling across each other induced a relatively volatile predator population. To enable a more stable ecosystem, magnitudes higher population sizes, especially predator population in combination with more environment space would be preferred. The limit in the animal numbers were due to lack of computational resources or lack of performance in the software. Due to this, simulations with larger initial populations took too long to finish since the time progression speed in unity was limited and the simulation could simply not finish computing in time between frames.

5.2 Interactivity Evaluation

From looking at the object placement menu (figure 4.2), it is apparent that the control offered to the user is not very intuitive. The environment creation menu is very

closely linked to the underlying implementation, meaning that much of the controls offered have dubious effects. This could likely be much improved by abstracting and combining parameters for increased usability. Not much work has been put into the layout of the menu either which means there is no clear visual hierarchy to guide the user. Although, the creation menu serves its purpose in the sense that control is offered to the user, potential improvements to the interface are much needed [34]. The first thing that might be done to improve the usability would be to carry out user-test to gain insights into the real usage of the software.

The simulation might also benefit from more methods of interaction. This could increase usability and in turn the *User Experience* as it might increase utility by allowing more control [35]. Examples of such potential interactions include allowing the user to: spawn or remove animals in run-time, control animals, get more information about animals and change the terrain in run-time. These features would also need testing to indicate at their real value.

Performance Evaluation

As presented in chapter 4.2 and in appendix D, the simulator could, on a modern desktop, simulate almost 700 rabbits with 60 FPS, with some drops in frame rate. This is acceptable performance for a simulator so heavily reliant on 3d-visuals, however, not great in comparison to real scientific ecosystem simulation studies. In the EcoSim project, prey entities number to as much as 150 000 [8]. In comparison, Ecosim is a 2d-visual simulator with colored dots, which naturally reduces rendering load, but it is a comparative mark of trusted scientific ecosystem studies. In its current implementation the simulator is not designed for optimal performance. There has been optimizations on aspects that clearly negatively impacted performance but the simulator was not created with this as a main goal. As already mentioned in section 5.5, one improvement would have been to create the simulator with Unity's *Entity Components System* (ECS). Beyond architectural changes there are however a few problems worth noting in the current form of the simulator.

The performance is also influenced by animals perception. The complexity when an animal does not perceive other animals is linear ($O(n)$, where n is the number of animals) to the amount of entities since each animal need one check of their surroundings. Further, objects can be hidden in view, and thus a raycast is calculated to each object found inside the animals field of vision. This can be problematic if the simulated environment is small with many entities since it causes a quadratic complexity ($O(n^2)$) when all animals need to check all seen entities. The worst case of quadratic complexity is rarely reached since most simulations are performed with sizeable environments but is a factor of concern. There were attempts to use a part of ECS for multi-threading purposes. However, since ECS supports primitive types exclusively, the implementation of sensing, which requires support for reference types, is not directly compatible.

5.3 Applications

The applications of this project are found mainly in its combination of ecosystem simulation and graphical representation. This lends itself for applications in pedagogy to introduce intuition of ecosystems for biology students or the general public. Furthermore, the simulation allows for highly approximate ecological experiments where the user can for instance compare the impact on an ecosystem caused by the terrain, water availability, forest density (affects sight) and presence of predators.

Adapting ML models to ecosystems has only recently been made practical due to the new frameworks of ML-tools made available. This particular application is interesting since most known simulators apply GAs to agents with reactive behavior in their systems. This is however not how real ecosystems function since most species apply some level of functional adaption between the brain and the body. The ML model used in the project are trained to adapt to their internal state and the environment. While we compared this with an FSM-based model comparing this with other behavioral models provides for interesting future experiments.

5.4 Social and Ethical Aspects

The main focus of the ecosystem simulation is not an undertaking of social and ethical aspects. Nonetheless, a critical area to discuss regarding the project's success, are the influences the project might have on society and science. Altogether, this can be summarized on whether the project causes harm, and whether it constitutes benefit for society and science.

A simulation tool will rarely depict reality with absolute accuracy. Therefore, a simulation that is interpreted as being more correct than it is, can result in distributing incorrect knowledge. As a result, it is of grave importance that this report carefully defines to which degree components of the simulation mimics reality. It is advised that users of the simulation has knowledge about the benefits and inadequacies of the implementation, when it is used as an educating tool.

For this project to be deemed an ethical success, it should provide value to society and science. The behavior of ecological systems, their intricacies and the hardships of creating a somewhat stable system, has surely been discovered by the participants of this project. This newfound knowledge has shed light on the importance of tending to the real world ecological systems. Hopefully, this realization will be achieved by future users of the project. The simulation, we feel, provides a good basis for future users to gain high-level insights into the dynamics of ecosystems. The simulation allows the user to explore ecosystems in different settings, with different preconditions, also enabling experimentation on the impact of different aspects on the environment. Although the ecosystem model is highly approximate, we deem it to be an ethical success since the value it provides is apparent.

5.5 Alternative Solutions

As in most software projects there are multiple components which could have been structured differently. As this is a multi agent simulation it would most likely have benefited from Unity's ECS which is especially designed to handle independent agents and systems at a larger scale than what was used in this project. The ECS is however very different in structure from regular object oriented programming and requires knowledge about Unity. This limited the project in scale of entities since the group did not have the expertise beforehand. Beyond this underlying architecture choice there are also alternatives in certain features.

Other than using outside libraries, there was also a possibility to use a 2-dimensional simulator. This approach was considered and discarded since one of the main goals is to create a simulator for use of people outside the project. A 3 dimensional representation is subjectively more immersive and was so valued above the 2 dimensional approach. The slightly simpler approach in 2 dimensions might however have resulted in more interesting results regarding evolution since the project could spend more time with the models rather than solving the complications that arise in 3 dimensions.

Another choice of implementation lays in time progression. There are arguments for a synchronized simulations where each time step is carried out simultaneously, mainly in the simplification of development since actions and reactions are easier to design. This would as in the section above result in more time to spend on other parts, such as the model. As aforementioned, the immersion was valued high and subjectively higher in a asynchronous world which acted as in real life.

5.6 Future Improvements

Since the project was limited in time there are still features to be implemented and potential improvements to the simulator. One core change would be to, with the help of other expertise, change the model by which animals are affected and which actions they can be taken. This project was solely carried out by computer science students but would have benefited from the help of ecologists or biologists. This would hopefully result in more interesting results which would provide more insight on actual ecosystems.

Beyond core model changes in the animals, a more dynamic or accurate environment could benefit the results of evolving and adaptive traits. There were many suggestions during the project with seasons, day and night cycles and more diverse food resources. This would in turn impact the animal model so that animals could react to these environments and could create interesting behaviors. In addition, this would create a better representation of a given ecosystem, increasing both immersion and educational value to users.

5.6.1 Machine Learning Model

Currently, a single ML agent is trained to survive and mate, and the same trained model is thereafter used for both predators and prey. As an alternative to this, multiple agents could be allowed to train in the same environment and work towards a common goal. Furthermore, the predators and prey could be divided into teams which would allow them to compete with each other for survival during training. The ML-Agents toolkit has tools that can be used for both of these concepts and is defined as *Multi-agent Scenarios*[1]. This approach would shift the classification of the system in general more towards *Multi-Agent* rather than Agent-based since the agents would work towards a common goal [36]. Exploring this approach would be interesting as it could provide a different result.

Another technique that could improve the behavior or training is Curriculum Learning[2]. With this technique, a complex task is broken down into a series of smaller tasks to be learned. This could improve learning speed and the resulting behavior.

One could potentially improve the performance of the ML model further by creating a custom RL algorithm for the ML-agents toolkit. The algorithm currently used, PPO, is not claimed to be the most performant and custom-made solutions tend to, in general, achieve higher performance than general ones.

Other improvements might entail a change in tooling altogether. The training is performed entirely before simulation run-time (As explained in 3.2.2), a side effect of the ML-agents toolkit. Alternatively, in-simulation training would have enabled the agents to adapt to run-time changes in the environment. Although, this would be performance heavy which likely would reduce the number of supported agents in the simulation. This approach would also be necessary if the project was to be extended to support evolving behavior and dynamic terrain. Presently, the only aspect of adaptability present in the agents comes from the parameter randomization, the extent of adaptability from this approach is likely not comparable. In-simulation training could be achieved by writing a C# ML library in Unity or using an existing one.

[1]Multi-Agent Scenarios: `https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Design-Agents.md#defining-multi-agent-scenarios`

[2]Curriculum Learning: `https://github.com/gzrjzcx/ML-agents/blob/master/docs/Training-Curriculum-Learning.md`

6
Conclusion

This project set out to create a usable and immersive simulator with a plausible model for the ecosystem with multiple animal species and a dynamically generated terrain. The implemented species included rabbits, wolves, bears and deer. There were three decision models for the rabbits, rule based (reactive) and two ML models. The two ML models had different approaches, where one controlled the changes of internal animal states (ML prioritization) and the other used pure vector steering (ML steering). All animal agents implement both reactive and ML prioritization models, but only the rabbits implement an ML steering model. The environment was generated at run-time and populated with selected samples of the four species to allow the user to customize the environment. The hope was for the simulator to show interesting results emerging from the evolved traits and some population relationships in the independent animal models which in combination resulted in a plausible visual ecosystem. The project also aimed to make the software usable to people outside the project to allow for more insight into ecosystems.

From the results (see chapter 4 and appendix D) it was evident that the ML steering model in the rabbits performed worse than reactive and ML prioritization models in rabbits, who performed similarly to each other. This was probably due to the subjectively easier implementation of ML prioritization and reactive rabbits, where the former used a library and the latter had lower complexity in implementation. The ML steering rabbits used the same library as the ML prioritization model, however, it was utilized differently, which resulted in unexpected difficulties. The simulation showed some relationships between the wolf and rabbit populations. This was seen when comparing terrain impact on ecosystems in section 4.1.3. Furthermore, evolution of animal genes (traits) were showcased when comparing rabbits living with and without wolves (see section 4.1.2). Finally, users can interact with the simulator through GUIs to set up a simulation, view live statistics of the simulations and output the result to JSON formatted text files for further study, which all contribute to a understanding and interactability in the simulator.

The simulator does not easily generate stable ecosystems over a longer period of time. Wolves tend to go extinct during the first hour. Furthermore, since no tests with deer or bears were conducted, few conclusions may be drawn regarding these species and their successful implementation. However, it is possible that these systems would struggle to reach equilibrium between species, since simpler rabbit and wolf systems did so. Stable ecosystems was, however, not a goal of the project and is thus not critical. Nonetheless, it may be possible to find stable ecosystems over

longer periods of time by investigating different parameter settings

This project shows the possible applications of ML models in visual ecosystem simulations. This is mainly due to the new ML libraries (see section 3.2.2) which combine ML APIs with game engines. This allows for easier exploration of trained agents in visual systems, which can be interpreted more easily by a wider audience. Furthermore, the study of ecosystem simulations is useful in order to understand ecosystems in the real world, both for scientists, but also for the public. It is in this intersection this project has been focused and is subsequently relatively successful.

Finally, the project goals have been sufficiently reached as the simulation provides results from evolved animals in different environments with subjectively immersive graphics. The software is reasonably simple, yet powerful enough to provide results for most users. There are some improvements to be made, some in animal models with more accurate modelling and an increased dynamic environment but also in its user experience and performance. Increased usability would make the simulator more accessible to others, which fulfills the project's purpose. Better models with higher performance would increase the scientific value provided when exploring the simulator.

Bibliography

[1] "They Are Billions," Jul 2020. [Online]. Available: http://www.numantiangames.com/theyarebillions (Accessed 2021-04-23).

[2] "BadNorth," 2018. [Online]. Available: https://www.badnorth.com (Accessed 2021-04-23).

[3] N. Bacaër, *Lotka, Volterra and the predator–prey system (1920–1926)*, 1st ed. London, United Kingdom: Springer, 2011, ch. 13, pp. 71–76. [Online]. Available: https://doi.org/10.1007/978-0-85729-115-8_13 (Accessed 2021-04-06).

[4] "Eco," Strange Loop Team, Feb 2018. [Online]. Available: https://play.eco (Accessed 2021-04-23).

[5] "Equilinox," ThinMatrix, November 2015. [Online]. Available: https://equilinox.com (Accessed 2021-04-23).

[6] G. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Springer, Berlin, Heidelberg, 2015, vol. 2. [Online]. Available: https://doi.org/10.1007/978-3-662-44874-8 (Accessed 2021-04-09).

[7] D. Shiffman, *The Nature of Code*, 1st ed. D. Shiffman, New York, USA, Dec 2012, vol. 1. [Online]. Available: https://natureofcode.com/book/index (Accessed 2021-04-05).

[8] R. Gras, "Ecosim: An ecosystem simulation," Jan 2018. [Online]. Available: https://sites.google.com/site/ecosimgroup/research/ecosystem-simulation (Accessed 2021-04-06).

[9] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," *Proceedings of the National Academy of Sciences*, vol. 99, no. suppl 3, pp. 7280–7287, 2002. [Online]. Available: https://doi.org/10.1073/pnas.082080899

[10] C. Grosan and A. Abraham, *Intelligent Systems: A Modern Approach*, ser. Intelligent Systems Reference Library. Springer Berlin Heidelberg, 2011. [Online]. Available: https://books.google.se/books?id=c1fzgQj5lhkC

[11] D. Bourg and G. Seemann, *AI for Game Developers*, ser. O'Reilly Series. O'Reilly, 2004. [Online]. Available: https://books.google.se/books?id=Sz-Sqvm-hSYC

[12] D. Bani-Hani, "Genetic algorithm (ga): A simple and intuitive guide," Jun 2020. [Online]. Available: https://towardsdatascience.com/genetic-algorithm-a-simple-and-intuitive-guide-51c04cc1f9ed (Accessed 2021-04-08).

[13] J. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, May 1992.

[14] D. Whitley, "A genetic algorithm tutorial," pp. 15–16. [Online]. Available: http://www.cs.jhu.edu/~ayuille/courses/Stat202C-Spring11/ga_tutorial.pdf (Accessed 2021-04-08).

[15] C. R. Darwin, *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*, 1st ed. John Murray, 1859.

[16] S. Sivanandam and S. Deepa, *Introduction to Genetic Algorithms*. Springer, Berlin, Heidelberg, 2008.

[17] J. Arora, *Introduction to Optimum Design*. Elsevier, 2017, vol. 4. [Online]. Available: https://doi.org/10.1016/B978-0-12-800806-5.00017-2

[18] M. Z. L. Demetrius, "The measurement of darwinian fitness in human populations," *Biological Sciences*, 1984. [Online]. Available: https://doi.org/10.1098/rspb.1984.0048

[19] "Genetic algorithms - crossover." [Online]. Available: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm (Accessed 2021-04-08).

[20] "Box-muller transformation." [Online]. Available: https://mathworld.wolfram.com/Box-MullerTransformation.html (Accessed 2021-04-09).

[21] T. Archer, "Procedurally generating terrain," in *44th annual midwest instruction and computing symposium, Duluth*, 2011, pp. 378–393.

[22] R. Touti. [Online]. Available: https://rtouti.github.io/graphics/perlin-noise-algorithm (Accessed 2021-05-08).

[23] H. Tulleken, "Poisson disk sampling," May 2009. [Online]. Available: http://devmag.org.za/2009/05/03/poisson-disk-sampling/ (Accessed 2021-04-02).

[24] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, Aug. 1962. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/00401706.1962.10490022 (Accessed 2021-04-12).

[25] B. Hagabil, J. Köre, I. Schwartz, A. Solberg, A. Sölveland, and R. Zetterlund, "Simulating an ecosystem - exploring the possibility of generating terrain-dependent nonplayer character behaviour by using an evolutionary-based fuzzy cognitive map," 2020. [Online]. Available: https://hdl.handle.net/20.500.12380/301962 (Accessed 2021-04-13).

[26] S. Lague, "Coding adventure: Simulating an ecosystem," Youtube, June 2019. [Online]. Available: https://www.youtube.com/watch?v=r_It_X7v-1E (Accessed 2021-04-13).

[27] M. Kleiber, "Body size and metabolic rate," *Physiological Reviews*, vol. 27, no. 4, pp. 511–541, 1947, pMID: 20267758. [Online]. Available: https://doi.org/10.1152/physrev.1947.27.4.511 (Accessed 2021-04-16).

[28] D. Moran, R. Softley, and E. J. Warrant, "The energetic cost of vision and the evolution of eyeless mexican cavefish," *Science Advances*, vol. 1, no. 8,

2015. [Online]. Available: https://doi.org/10.1126/sciadv.1500363 (Accessed 2021-04-16).

[29] M. R. Hirt, W. Jetz, B. C. Rall, and U. Brose, "A general scaling law reveals why the largest animals are not the fastest," *Nature Ecology & Evolution*, Aug. 2017. [Online]. Available: https://doi.org/10.1038/s41559-017-0241-4 (Accessed 2021-04-18).

[30] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018. [Online]. Available: https://github.com/Unity-Technologies/ml-agents (Accessed 2021-03-22).

[31] C. Souza and L. Velho, "Deep reinforcement learning for high level character control," 2020.

[32] T. Stöttner, "Why Data should be Normalized before Training a Neural Network," *Medium*, May 2019. [Online]. Available: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d (Accessed 2021-04-23).

[33] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," 2017.

[34] "Usability 101: Introduction to Usability," Dec 2017. [Online]. Available: https://www.nngroup.com/articles/usability-101-introduction-to-usability (Accessed 2021-04-29).

[35] "The Definition of User Experience (UX)," Dec 2017. [Online]. Available: https://www.nngroup.com/articles/definition-user-experience (Accessed 2021-04-29).

[36] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387–434, 11 2005. [Online]. Available: https://doi.org/10.1007/s10458-005-2631-2

[37] "Learning environment - design agents." [Online]. Available: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Design-Agents.md (Accessed 2021-04-14).

A

Performance Profiling Results

Performance Profiling with the Unity Profiler with finished build Performance measurements were done with only rabbits and food. Test machine used a RTX 2060 with 6 GB VRAM GPU, AMD Ryzen 5 5600X CPU, 16 GB RAM.

Figure A.1: Rabbits per minute when profiling

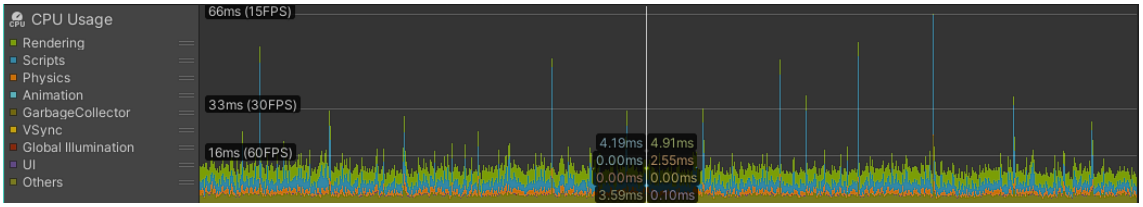


Figure A.2: Frame rate over the last 2000 frames at the end of the simulation in figure A.1. The y-axis supplies the time to render a frame, converted to FPS in parentheses. Lower means higher frame rate.

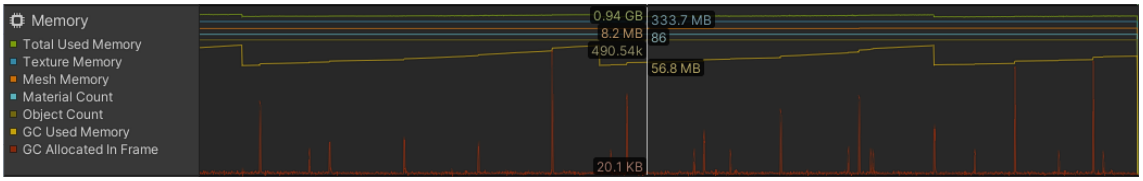


Figure A.3: Memory usage over the last 2000 frames at the end of the simulation in figure A.1, lower means less memory usage.

B

Animal Parameters and Traits

Parameters

- Energy
- Hydration
- ReproductiveUrge
- Age

Traits

- maxEnergy
- maxHydration
- maxReproductiveUrge
- ageLimit
- maxSpeed
- acceleration
- size
- viewAngle
- viewRadius
- hearingRadius

C

Machine Learning - Steering Implementation

While the Steering approach to ML was not used in the final product, the effort put into implementing it and the insights gained from this deserves to be mentioned here. The implementation in general followed the same procedure of the ML prioritization (see section 3.2.3), although it differed in details such as in which observations were used.

The observations for the steering agent included the following:

- Energy percentage (E for energy) calculated as $\frac{currentE}{maxE}$.
- Hydration percentage (H for Hydration) calculated as $\frac{currentH}{maxH}$.
- Reproductive viability (RU for Reproductive Urge), shown in figure (C.1).
- Direction of Agent's relative velocity (x and z value).
- Position (x and z value) of nearest food relative to agent's position.
- Distance to nearest food relative to agent's position.
- Position (x and z value) of nearest water relative to agent's position.
- Distance to nearest water relative to agent's position.
- Position (x and z value) of any perceived potential mate relative to agent's position.
- Distance to any perceived potential mate relative to agent's position.

$$\frac{currentRU}{maxRU} = 1 \tag{C.1}$$

All metrics in the observation space were normalized to avoid skewed policy updates. Unity's ML agent documentation specifies that "the greater the variation in ranges between the components of your observation, the more likely that training will be affected" [37].

The action space was kept as minimal as possible. The model would therefore not require as much training, since there would be fewer actions to choose from in each RL state. Actions included two continuous actions $CA_1 and CA_2$ which were clamped in the range of $[-1, 1]$. CA_1 would dictate the speed of the movement. The value range had to be limited within $[0, 1]$ using a linear equation $y = 0.5 - 0.5x$ where $y = 0$ represented no speed and $y = 1$ represented maximal speed. CA_2 would

dictate rotation of the moving direction around the y-axis, which was derived from the animal's forward direction. The value range was mapped to $[-110°, 110°]$.

The reward system was implemented in a straightforward manner. Reward for eating was given as the minimum of nutrition gained from the food and the difference of maximal energy and current energy as:

$$R_{eating} = \frac{\min(foodNutrition, maxE - currentE)}{maxE} 0.1.$$

Similarly, reward for drinking water was given as the difference between max hydration and current hydration according to:

$$R_{drinking} = \frac{maxH - currentH}{maxH} \cdot 0.1.$$

A negative reward would be given depending on the speed of the agent, such that a maximal speed would minimize the value of negative reward given, hence the equation:

$$R_{locomotion} = (0.5 \cdot CA_1 + 0.5) \cdot 0.0025 - 0.0025.$$

According to Unity's documentation on ML Agents it is common practice to give a small reward for forward movement in locomotion tasks [37]. Additionally, a small negative reward was given proportional to the amount of rotation for each action step, $R_{rotation} = (-0.0025) \cdot |CA_2|$ to prevent the agent from exploiting oscillating movement. Furthermore, a negative reward was given for dying $R_{dying} = (-1)$, and a positive reward was given for achieving the goal of reproducing $R_{reproducing} = 2$.

Training of each steering agent was carried out similarly to the prioritization approach (see section 3.2.3), in an environment with a specified length and width. However, in this environment the dummy rabbits were not statically placed and there were no dummy wolves. Subsequently, on episode start, the environment was populated with food, water blocks and a dummy mate for the agent, all with a random position on the platform. At the end of each episode everything in the environment except the agent was removed, whose parameters instead were reset.

D

Population and Trait Results

This appendix intends to highlight results derived from running the simulator with different settings. The results are arranged to compare the outcome of running the simulator with predefined settings from chapter 4. A mean value line is plotted for each type of simulation which is discontinued at the point of the first ending simulation of respective type. Continuing the mean for the remaining simulations would defeat the purpose of a mean line plot. In addition to the mean curve, individual runs are plotted in a similar color, albeit in a thinner line. The notion of simulation type is used to describe a predefined configuration for a simulation, for example `RW-P-32536-HDIFF` which corresponds to a simulation type; rabbit and wolves (RW), with rabbits having the machine learning prioritization behavior type (P) on seed 32536 (32536) with an abnormal height difference compared to standard configuration (HDIFF).

D.1 Behavior Results

The following results displays how the three different behavior models for rabbits (reactive, prioritization ML and steering ML) perform in equal environments. Section D.1.1 compares an ecosystem with just rabbits and food. Section D.1.2 uses identical environments, but wolves are added to the ecosystem in addition to the rabbits and food. Stable simulations ran for 2 hours in simulation time unless the rabbits went extinct.

D.1.1 Rabbits and Plants

In figure D.1 the compared models fared differently. The ML steered rabbits went extinct and did not survive while the two other models resulted in persisting populations. Note that the reactive model resulted in more rabbits than the prioritization model after the simulations had stabilized by minute 20. Figure D.2 shows the plant population over time in the different models. `R-P-0` seems to have produced slightly more plants than `R-R-0`.

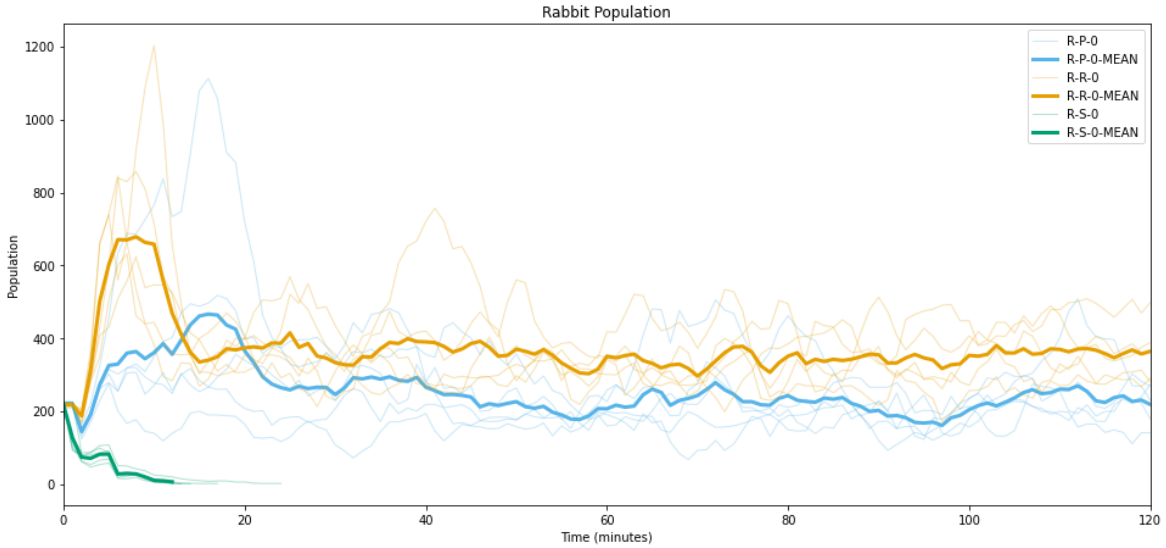


Figure D.1: This plot shows how the three different behavior models resulted in different populations over time. The ML steered rabbits went extinct before 20 minutes, whereas the other models survived for 2 hours.

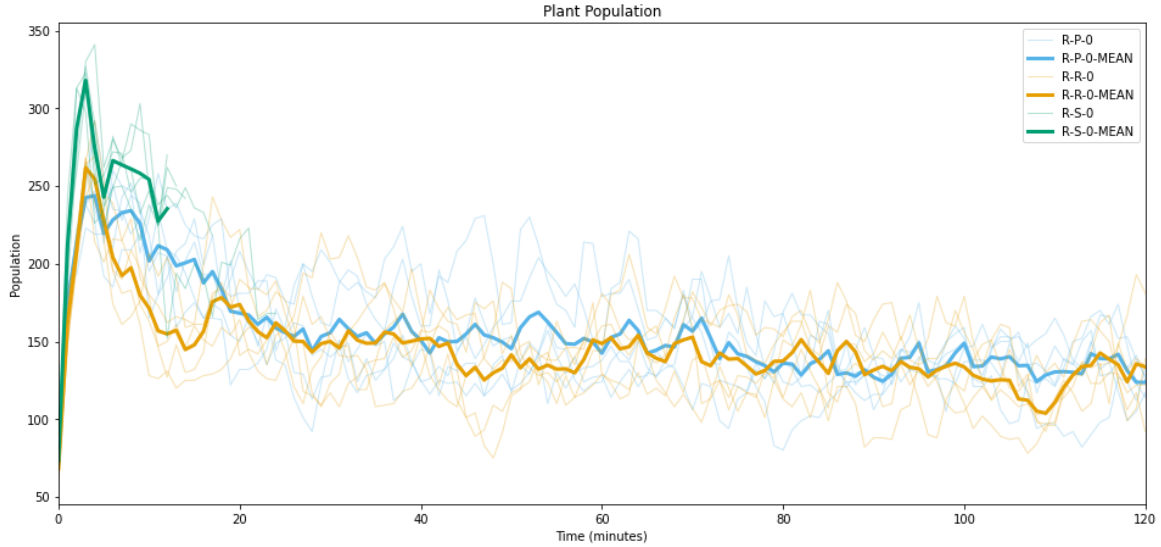


Figure D.2: This plot shows the population of the plants in the simulations comparing the three behavior models. All three simulations invoked an initially growing plant population, but both `R-P-0` and `R-R-0` had less volatility after the 20 minute mark. All the `R-S-0` simulations ended before the 20 minute mark, thus these simulations were not investigated further.

D.1.2 Rabbits, Wolves and Plants

Figure D.3 shows that the three models fared differently again. Much like in figure D.1 above, ML steered rabbits went extinct before 20 minutes, but the other two

models survived. Note how the overall rabbit populations are smaller compared to the previous result. The reactive model also generated slightly bigger populations overall as before, however, this seems less certain given the final minutes where the prioritization model yields increases in population.

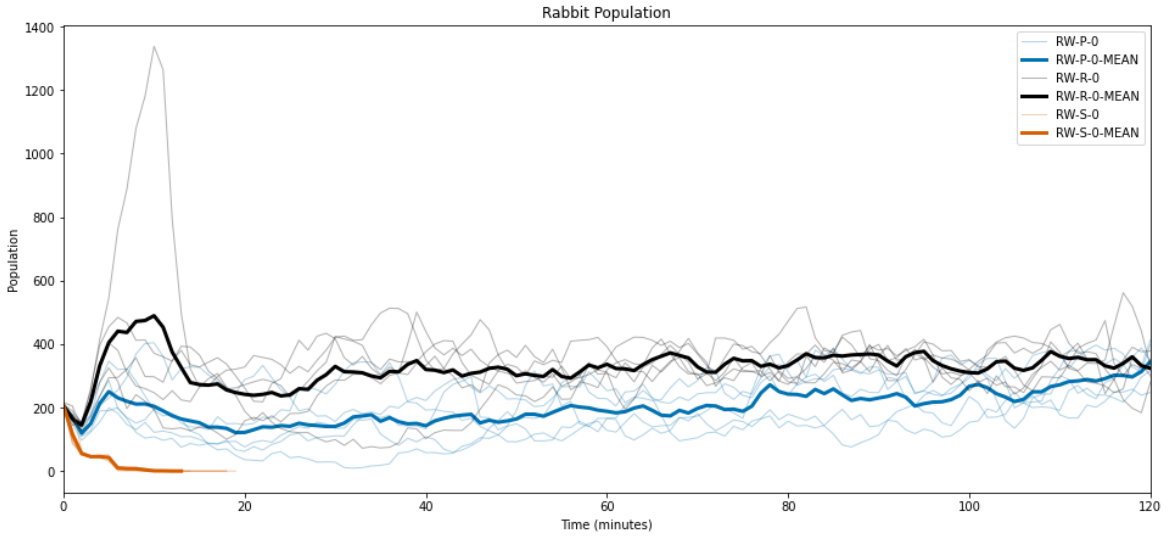


Figure D.3: This plot shows how the three different models for rabbits living with wolves resulted in different rabbit population numbers. The ML steered rabbits went extinct before 20 minutes, whereas the other models continued to 2 hours.

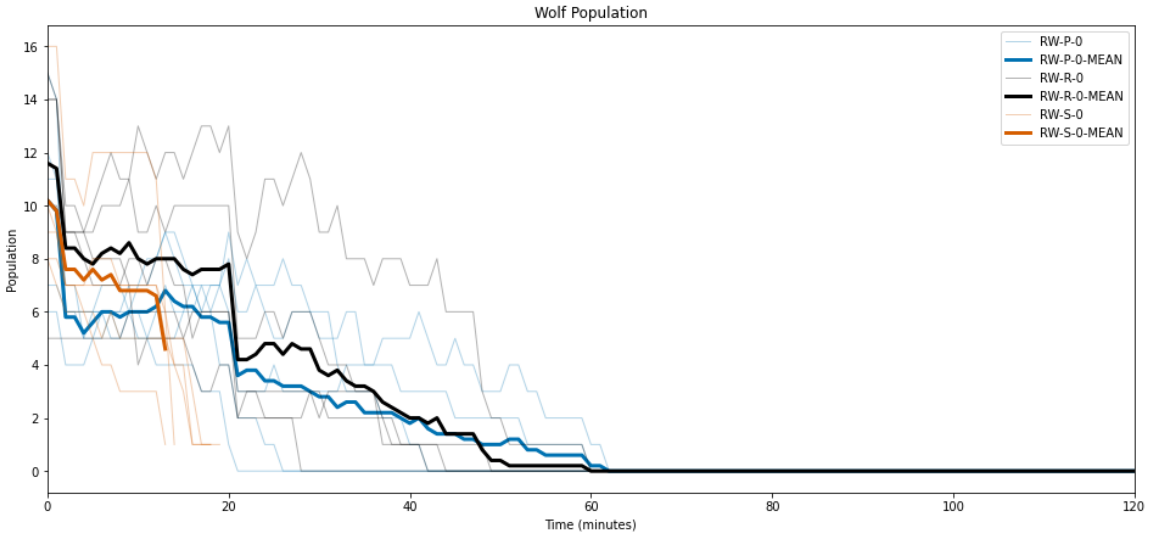


Figure D.4: This plot shows the wolf population for the three different simulation types with wolves. `RW-R-0` and `RW-P-0` ran for 2 hours and concluded in extinction usually around the 1 hour mark, while `RW-S-0` ended before the 20 minute mark.

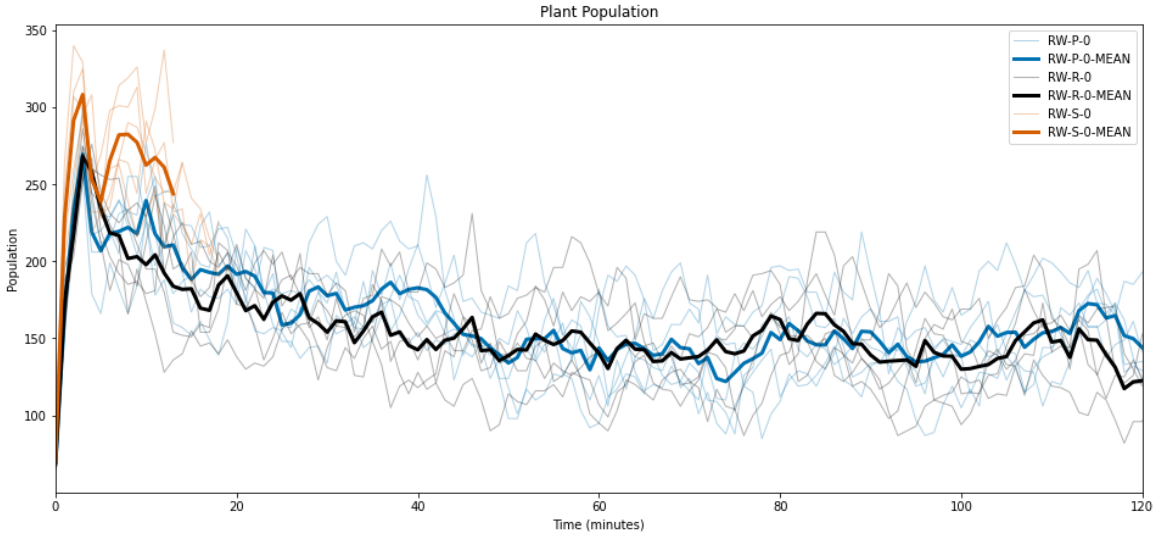


Figure D.5: This plot shows the plant population for each simulation type. The plant population had an initial growth for all the simulation types. `RW-R-0` and `RW-P-0` ran for 2 hours and had a less volatile plant population after the 20 minute mark. `RW-S-0` was ended before the 20 minute mark.

D.2 Terrain Impact Results

The following results displays the impact of terrain changes. These simulations are carried out in seed 32536, with more accessible water throughout the map, as compared to seed 0 where most water sources are found along one edge of the map. Additionally, a comparison between two types of simulations where the height setting is different is presented.

D.2.1 Seed Difference

Figure D.6, D.7 and D.8 shows rabbit population, wolf population and amount of food, respectively. These comparisons are conducted on an environment with reactive rabbits, wolves and seed 0 and 32536.

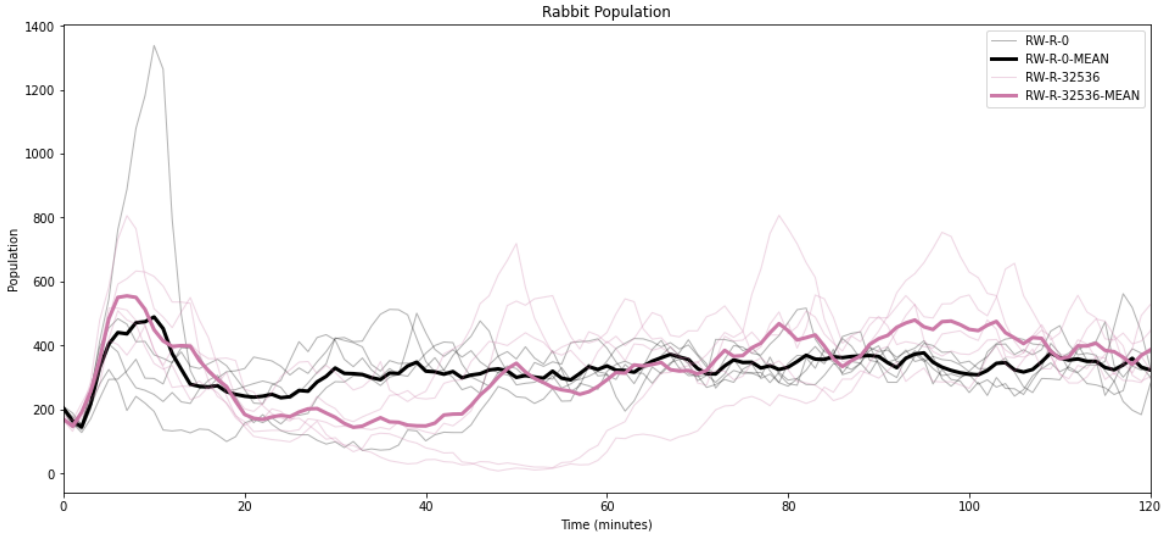


Figure D.6: This plot shows the rabbit populations in seed 0 and 32536 with the reactive rabbit model. The average populations on both seeds are similar.

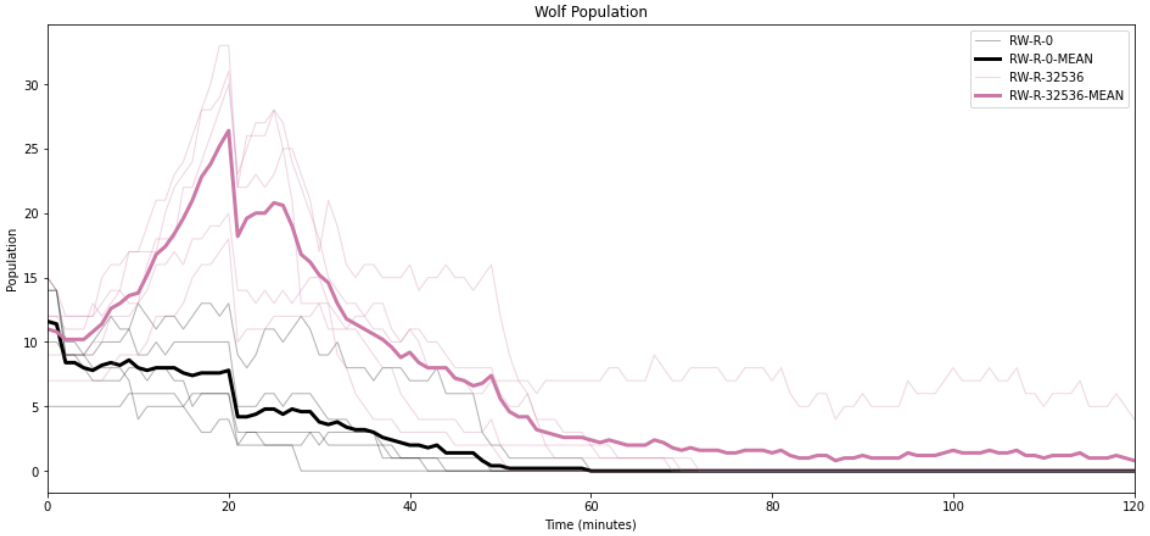


Figure D.7: This plot shows the wolf population over time for the two simulations with `RW-R-0` and `RW-R-32536`. The simulation on seed 32536 saw greater wolf populations on average.

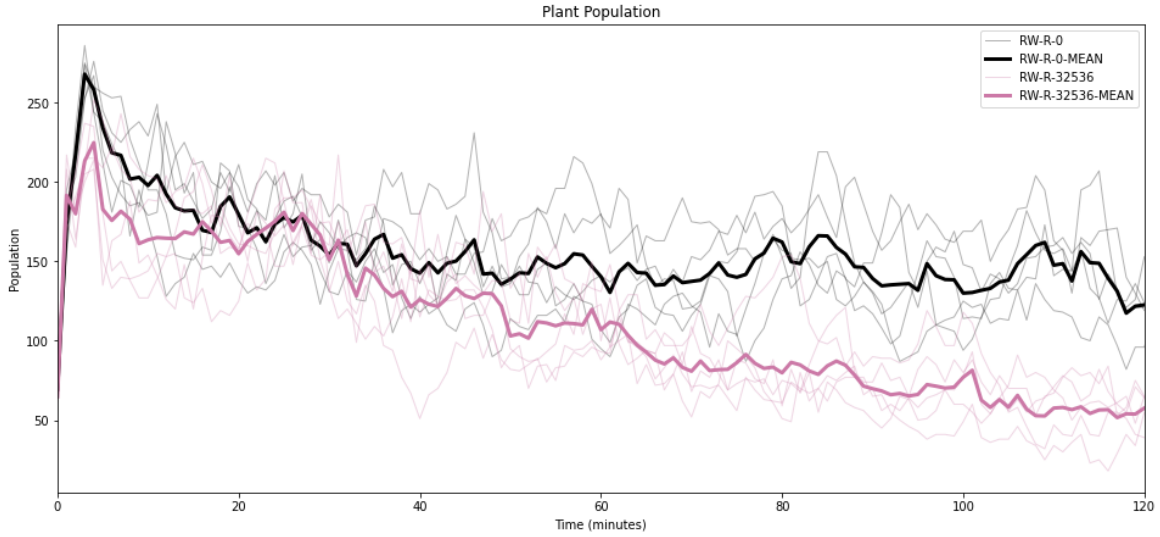


Figure D.8: This plot shows the available food over time for the two simulations with `RW-R-0` and `RW-R-32536`, where the food available to the rabbits in the former is more abundant.

D.2.2 Height Difference

This section displays the differences in rabbit, wolf and plant populations over time when using two different height settings for the environment. The simulations compared are `RW-R-32536` and `RW-R-32536-HDIFF`. The worlds where identical except for the height multiplier, set to 35 in the HDIFF map. The standard setting is 10.85. Thus, the HDIFF map is more hilly than the other.

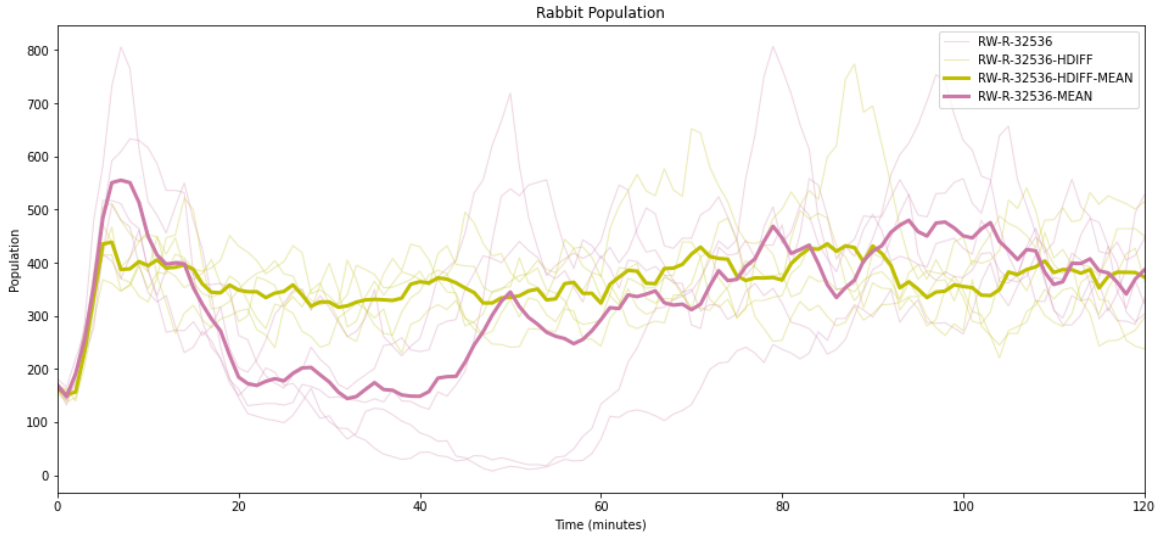


Figure D.9: This plot shows the rabbit population over time for the `RW-R-32536` and `RW-R-32536-HDIFF` worlds. Notably, the simulations with higher height multiplier appear more stable.

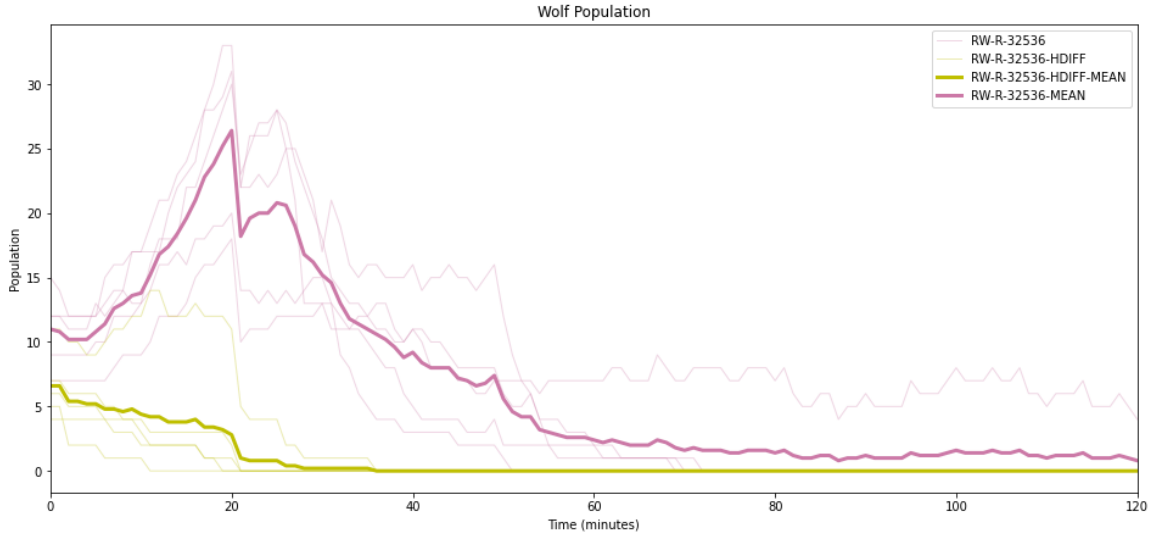


Figure D.10: This plot shows the population of wolves over time. The simulation on the standard height multiplier (`RW-R-32536`) had larger wolf populations.

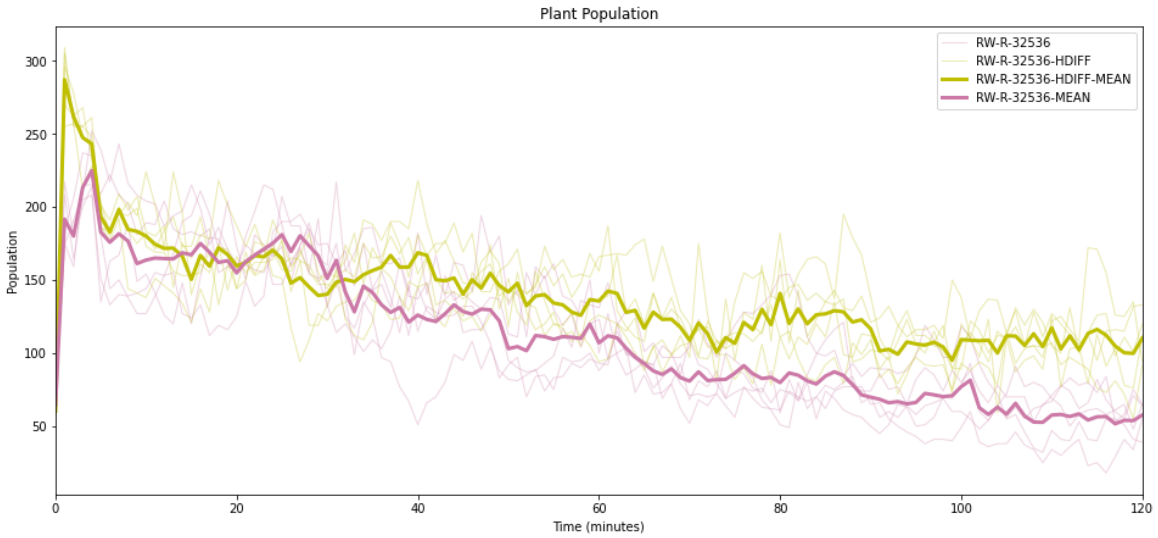


Figure D.11: This plot shows the available plants during the simulations `RW-R-32536` and `RW-R-32536-HDIFF`. Notably, the mean of HDIFF contains slightly more plants from 40 minutes onwards.

D.3 Evolution of Traits Results

This section presents evolution of select traits in rabbits in some simulations. The traits presented are the maxSpeed, ageLimit, maxEnergy, maxReproductiveUrge and viewRadius. These are not all traits, but a subset that are representative to the purpose of the project. Two simulation setups are compared, namely R-R-0 and RW-R-0. These are selected to show any difference of rabbits traits when living under

no selective pressure from wolves versus living with wolves. Note that each trait is a floating point value with no unit or real metric representation. Also note that the plots in this section are focused to give a more detailed view of the mean value line for each simulation type. Given this detail, it should be noted that simulations did not end where the x-axis limit of the plots lies.

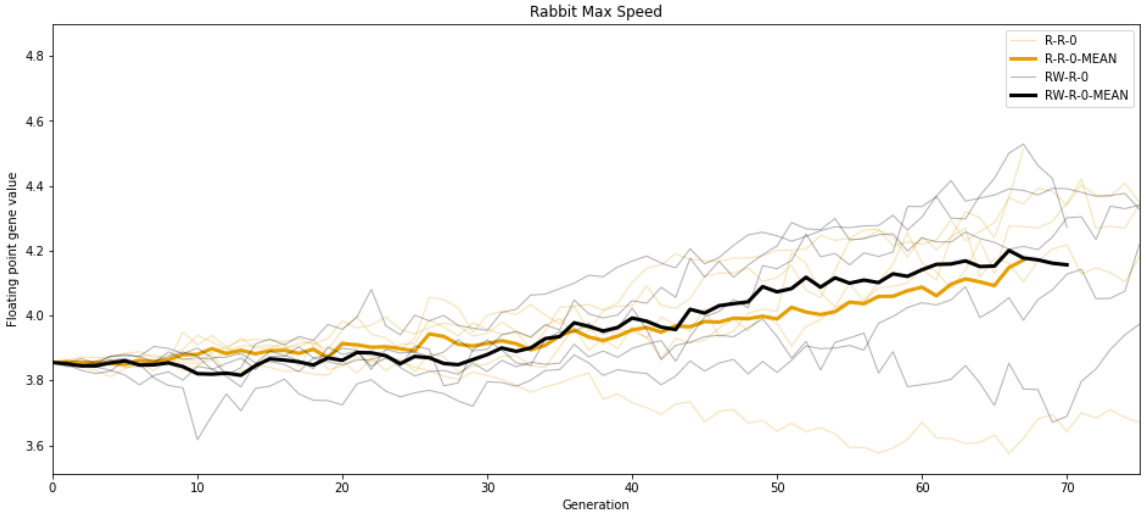


Figure D.12: This plot shows the maxSpeed trait of rabbits changing over generations. The trait increases in both simulations.

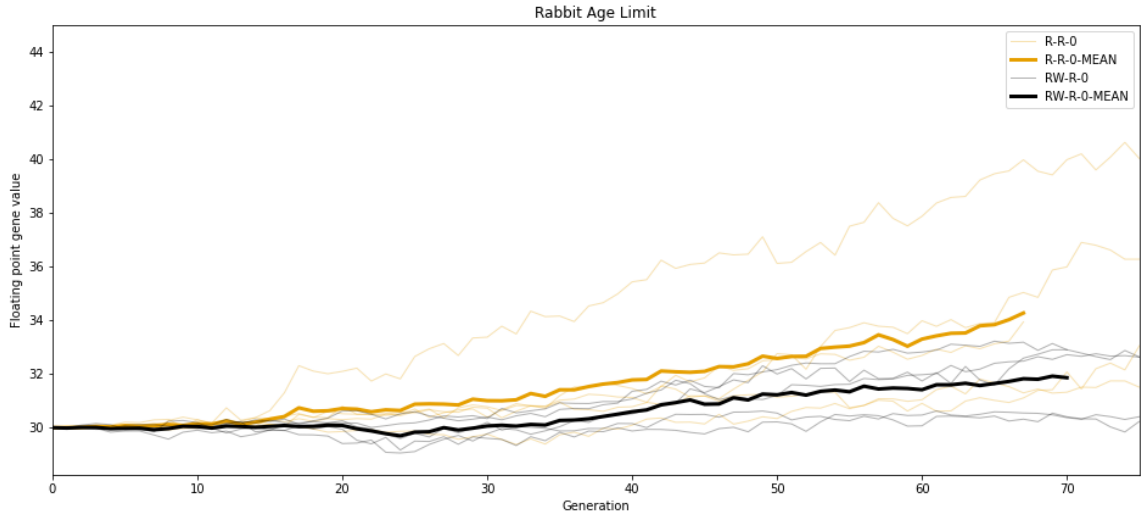


Figure D.13: This plot shows the ageLimit trait of rabbits changing over generations. The simulation with only rabbits (`R-R-0`) display a larger increase compared to the simulation with wolves `RW-R-0`.

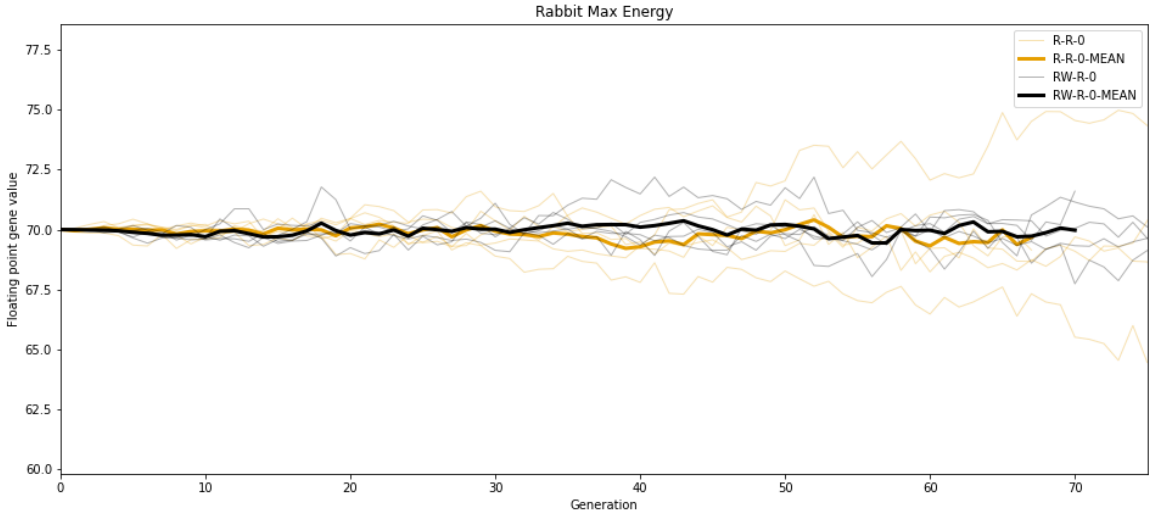


Figure D.14: This plot shows the maxEnergy trait of rabbits changing over generations. In both simulations the floating point value is roughly unchanged.

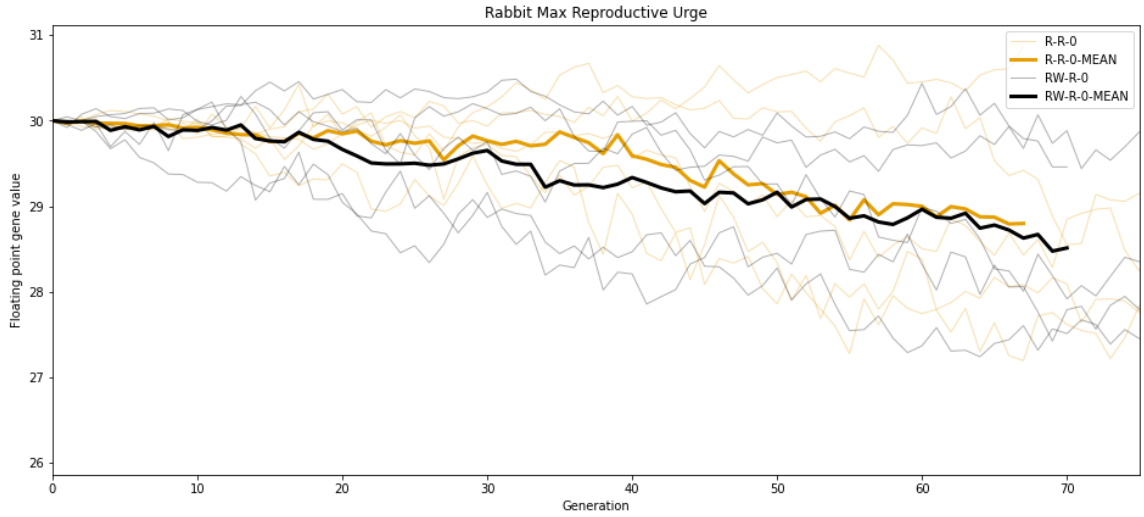


Figure D.15: This plot shows the maxReproductiveUrge trait of rabbits changing over generations. In both simulation the floating point value decreased over generation.

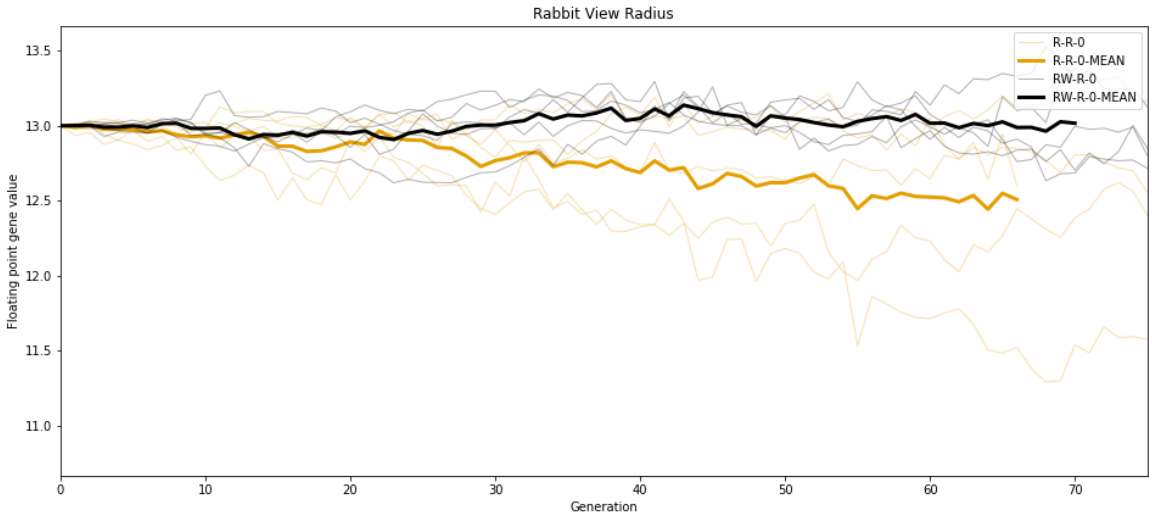


Figure D.16: This plot shows the viewRadius trait of rabbits changing over generations. In `RW-R-0` the trait remain mostly unchanged during the simulation, while the mean value decreased in the simulation type `R-R-0`.