Simulating an Ecosystem

Implementing natural selection using a genetic algorithm in a predator-prey environment

Bachelor's Thesis in Computer Science and Engineering

Aron Sjöberg
Emil Wingårdh
Erik Söderpalm
Theo Wiik
Viktor Fredholm
Yvonne Hansson

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

BACHELOR'S THESIS 2021

Simulating an Ecosystem

Implementing natural selection using a genetic algorithm in a predator-prey environment

Aron Sjöberg
Emil Wingårdh
Erik Söderpalm
Theo Wiik
Viktor Fredholm
Yvonne Hansson

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

Simulating an Ecosystem
Implementing natural selection using a genetic algorithm in a predator-prey environment
Aron Sjöberg, Emil Wingårdh, Erik Söderpalm,
Theo Wiik, Viktor Fredholm, Yvonne Hansson

Supervisor: Marco Fratarcangeli, Department of Computer Science and Engineering
Examiner: 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: A snapshot from the simulation.

Typeset in LaTeX
Gothenburg, Sweden 2021

Simulating an Ecosystem
Implementing natural selection using a genetic algorithm in a predator-prey environment
Aron Sjöberg, Emil Wingårdh, Erik Söderpalm,
Theo Wiik, Viktor Fredholm, Yvonne Hansson

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Sammandrag

Att simulera ett ekosystem kan vara ett krävande arbete, inte minst på grund av dess komplexitet. Ekosystemet är modellerat för att vara en förenkling av dess verkliga motsvarighet och skildrar en näringskedja bestående av kaniner, vargar, plantor, träd och vatten. Syftet med arbetet är att studera hur mängden resurser, individuella egenskaper och beteenden påverkar det naturliga urvalet hos de simulerade djuren. För att simulera naturligt urval, och på lång sikt evolution, introducerades genetiska algoritmer för djuren. För att kunna värdera resultatet av simulationen samlades data in. Den insamlade datan visar hur antalet djur av varje art fluktuerar över tid, samt hur djurens hörsel- och synförmåga, hastighet och storlek utvecklar sig över tid. I rapporten presenteras också teorin över de viktigaste komponenterna för skapandet av djurens beteende och evolution, samt för verktygen och funktionaliteten i Unity som använts för att bygga upp simulationen.

Resultatet visar att det är möjligt att nå jämvikt i simulationen. Vidare visar resultaten att antal djur beror på antalet plantor som finns i världen och att djurens egenskaper förändras över tid.

Slutsatserna som kan dras av resultatet är att instanserna av simulationen utförda med genetisk algoritm ger ett resultat där jämvikten håller längre jämfört med instanserna utan genetisk algoritm, givet samma startvärden. Samt att simulationen av ekosystemet skulle kunna förbättras ytterligare för att kunna möjliggöra att simulationen utförs med fler resurser och större antal djur.

Abstract

Simulations are computer programs based on models, used to imitate real-life scenarios. Simulating an ecosystem is a great challenge given the various factors that interact and animals that coexist. The thesis work presented in this paper aims to model an ecosystem and study what effect the number of resources, animal characteristics, and animal behaviors have on the natural selection of the simulated animals. The animals evolve through natural selection, implemented using genetic algorithms. Specifically, the project aims to investigate how the population and the traits of the animals change, based on interactions with the environment and with other species. This is done by gathering data while running the simulation.

The ecosystem, with all its components, was built using the Unity game engine. The rabbits and wolves, that constitute the food chain, were modeled to be accurate but simplified versions of their real-life counterparts. The results show that reaching equilibrium is possible, albeit difficult if the right start values and correct world size are not chosen. Furthermore, the number of animals depends on the number of plants, and the traits of the animals change over time.

The conclusions to be drawn from the results are that the instances of the simulation with the genetic algorithm implemented were able to run for longer compared to the instances without the genetic algorithm. The results also suggest even though it is greatly simplified, that the simulation proved to be stable for certain initial numbers of rabbits, wolves, and plants. This stability can be greatly improved in order for the simulation to handle greater variations in resources and the number of animals.

Keywords: ecosystem, simulation, Unity, genetic algorithm, evolution, natural selection.

Acknowledgments

We would like to thank the following creators for the assets and tools which we utilized in our project:

- Chris Nolet for "Quick Outline" [1], which we used for highlighting animals.
- Valady Team for "Low Poly Nature Pack (Lite)" [2], which we used for the flowers.
- Yasirkula for "Text to TextMesh Pro Upgrade Tool" [3], which we used for the menu text.

We would also like to thank the following Youtube creators and blogs for creating tutorials and guides which were really helpful for learning the Unity engine:

- The Unity channel [4] for overall tutorials for Unity, as well as for the wind shader for the trees.
- The Brackeys Team from the 'Brackeys' channel [5] for overall tutorials for Unity.
- Ned from the 'Ned Makes Games' channel [6] for the post-process outline effect.
- James from the 'gamesplusjames' channel [7] for the toon cel-shader.
- Daniel Ilett from the 'Daniel Ilett' channel [8] for the dithering transparency effect on the trees.
- Protokoll Studio for the water shader tutorial [9], parts of which were used for the water shading.

We also want to thank Michael Heron for giving us valuable feedback on the report.

Lastly we want to give special thanks to our supervisor Marco Fratarcangeli for the guidance and help, and for providing the assets used in the simulation.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

An ecosystem is defined as a biological community of interacting organisms and their physical environment [10]. It contains biotic factors that are multicellular (animals, plants) and unicellular (bacteria), as well as abiotic factors such as temperature and humidity. These factors interact with each other either directly or indirectly creating relations between the organisms and the environment.

Different types of ecosystems have developed over millions of years, resulting in several millions of different species. Just by observing nature, Charles Darwin in his book *On the Origin of Species*, presented his ideas that animals and plants adapted to their surroundings by natural selection [11]. This book was groundbreaking at the time and it is now the foundation for the modern evolutionary theory [12].

Simulating an ecosystem is interesting from a biological and technical point of view. From a biological point of view, simulating an accurate ecosystem could potentially be used to model how species might develop in the future. It could also predict how changes in the environment could affect different species. For example, it could showcase effects caused by humans, such as the destruction of natural habitats through deforestation [13], and how the global rise in temperature affects animals and plants [14]. However, only the effect of resources and characteristics will be explored in this paper.

Meanwhile from a technical point of view, simulating an ecosystem could be both challenging but interesting for several reasons: can something as intricate as DNA be accurately translated into a computer model? Can current hardware support the number of entities needed to simulate an ecosystem in real-time?

To investigate the viewpoints (mainly) from the technical aspects, this project's aim is to build a small ecosystem in Unity, a popular game engine, including resources (water and plants), and animals (predators and herbivores). Each individual will have different traits, for example, speed of motion, range of view, hearing, hunger, thirst, and gender. By simulating natural selection, implemented using a genetic algorithm, some of these traits will change over time.

1.2 Purpose

The purpose of the project is to study what effect the number of resources and individual characteristics have on the natural selection of the prey and predators in the simulation. The genes in the animals will be implemented using a genetic algorithm. By doing so, the goal is to see how genetic changes can be seen in the animals' offspring over time, thus simulating natural selection.

Additionally, we want to investigate whether an equilibrium can be reached where two species can live in coexistence within the ecosystem. This is to examine whether the genetic algorithm and subsequent natural selection can aid in keeping an ecosystem stable. Answering these questions will give a better understanding of how evolution and ecosystems work.

Finally, the created program used for running the simulation should be visually appealing and easy to manage. This will make it more user-friendly and allow for future use of the program.

1.3 Boundaries

Considering the complexity of a real-life ecosystem, it is necessary to include a number of boundaries in order to limit the scope of the project. The goal of the project is to implement a single, simplified ecosystem. However, the project will be developed to make it as extensible as possible to allow the addition of more animals and plants. Because of the intricacy of their real-life counterparts, limitations regarding the complexity of the simulation's animals and plants are needed. The animals in the simulation have senses such as vision and hearing. There was discussion about including a sense of smell. This idea was quickly discarded, since that in turn would have introduced other types of complexities such as wind, the direction of the wind, and animals and plants emitting smells.

The purpose of the plants in the simulation is simply to provide food for the rabbits. The plants do not reproduce, disappear or reappear in other parts of the world. Again, this would make the ecosystem more intricate thus making it more difficult to try to reach an equilibrium. Therefore, the plants were kept relatively simple. A real-life ecosystem normally consists of several tiers within a food chain. This simulation only contains three tiers of the food chain, of which only two consist of animals. The reason behind this is the complexity of reaching an equilibrium, which increases with the number of variables introduced to the ecosystem.

Furthermore, the number of animals that can be simulated at a time is around 1 000 entities, depending on computer hardware. A greater number than this can result in drops in FPS (frames per second), which may affect the results. The NavMesh is another limitation in this project. With too many entities in the world at a time, the NavMesh is limited to the number of paths it can calculate at the same time. This problem might negatively impact the results since the animals die while being stuck in the terrain. A greater number of animals and plants would probably lead to a stable ecosystem with more reliable data.

A majority of the boundaries mentioned exist due to the time constraints of the project. With more time given, perhaps a greater complexity to some of the functionality could have been introduced. However, given the relatively short amount of time for this project, almost all of the components with varying factors had to be kept simple.

1.4 Overview

Chapter 2 will describe the theory behind the most important components used in the simulation. This includes a brief introduction to Unity, the real-time development platform used for building the simulation. The different functionalities of Unity will be presented, as well as how they are incorporated into the simulation. Related works will be presented in chapter 3. Chapter 4 will describe the model of the ecosystem. This includes the functionality of the animals and plants, how the genetic algorithm is implemented, and what type of data is gathered from the simulation. The data gathered from the simulation will be presented in chapter 5 and discussed in chapter 6. Finally, chapter 7 will finalize this paper with a conclusion.

2

Theory

This section covers the relevant theory used to create and understand the simulation.

2.1 Finite State Machine

A finite state machine (FSM) is a design pattern used to manage different types of behaviors (states) and the transition between said behaviors [15]. As the name implies, an FSM consists of a finite amount of states. Transitions between the states depend on certain conditions. An example of a use case for an FSM is a traffic light. A traffic light has four states, red light (stop), red-yellow (prepare to go), green (go), and yellow (prepare to stop). An illustration of the traffic light's states and transitions can be seen in Fig. 2.1.

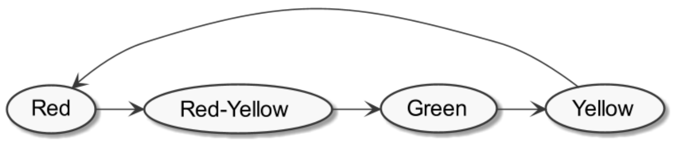


Figure 2.1: An example of an FSM for a traffic light. The nodes represent a state and the directional arrows represent a possible transition between states.

Finite State Machines allows flexible code [16]. An FSM allows for modular functionality and can reduce code complexity. The FSM was an effective way to structure the code for the animals in the simulation since they, similarly to a traffic light, have a finite amount of states in which they can be. For example, an animal is either hunting or eating, not both at the same time.

In real life, it is possible for animals to have several urges at the same time, for example feeling hungry and thirsty. While having the states strictly divided might not be analogous to real life, it gives a good enough approximation of real-life behavior. Furthermore, building the FSM state by state is relatively straightforward: specify what the state should do, how to reach the state, and how to move to the next state (if applicable).

With that said, an FSM also comes with some drawbacks. As mentioned, the animals can only be in one state at a time. For example, in the case that an animal is simultaneously hungry and thirsty, it can be necessary to predetermine what the animal should prioritize. Does it make sense for it to go after food or water first? These kinds of decisions can lead to strange behavior when running the simulation. If food is prioritized over water, the animal will go after food even though it is closer to a water source, or vice versa. So, an animal can be simultaneously hungry and thirsty, but they can only do the act of pursuing one of the resources.

Another drawback with FSMs is the decrease in code encapsulation. With FSMs, much of the logic is extracted to classes outside of the core class containing the state machine. The different states might depend on a specific property of a class, thus the shared fields must be accessible from the entity having a state machine to the different states (this can vary by methods of implementation). This could be solved by not extracting the logic of each state to its own class, but rather choose to implement it internally instead. However, this would result in the class being very large. In the end, the FSM was chosen for the sake of modularity and simplicity.

2.2 Genetic Algorithms

In the year 1859, Charles Darwin published *On the Origin of Species* [11]. In the book, Darwin presented the theory that in nature the individuals better suited for their environment will have a higher chance of reproducing. Genes that make the animal more suited to survive will have a greater probability to be passed on to the next generation. This is the fundamental force that drives evolution forward. Based on this idea of natural selection, genetic algorithms are used for optimizing and finding solutions to problems in computer programs [17].

There are multiple ways of creating a genetic algorithm, but all of them share the following components [18] [19]:

- One or more representations of the genes
- A generating function to generate new gene variations
- A fitness function that fits the genes by giving them a value to be able to judge which genes are better
- A selection function which describes the selection from the genetic variants based on the values from the fitness function
- A crossover function which describes how to combine the selected genetics
- A mutation function to change the result of the crossover function slightly, most often at random

By using the components above, a genetic algorithm generally works in the following way [18]: the genes are represented in binary encoding, i.e. a string consisting of ones and zeros. These are generated by a generating function. The ones and zeros have different meanings depending on implementation. Much like in real-life evolution, every iteration of a genetic algorithm is divided into generations. Generation 0 consists of a random selection of genes coming from a population (Fig. 2.2). Population size is dependent on implementation. The fitness function is used to determine how fit each gene in the given generation is. The fitness function is also dependent on implementation, but the general idea is that if a gene is fit enough, it is given a score based on *how* fit it is. If a gene is not fit at all, it will be given a score of 0. Based on the scores given by the fitness function, a new generation is to be selected using a selection function. Two genes with high fitness scores have a greater chance of being selected to be "parents", compared to genes with a low fitness score.

```
0 1 0 0 0 1 1 0     0 0 0 1 1 0 1 1

1 0 1 1 0 0 0 0     1 0 1 1 1 0 1 1

0 1 1 0 0 0 1 0     0 0 0 1 0 1 1 0
```

Figure 2.2: A population of six genes where each gene consists of eight bits.

As two genes are paired up, the crossover function is used to cut the genes in a random location and switch the bits of the genes (Fig. 2.3). These actions give two new genes, i.e. two new solutions, and are repeated for every gene in a generation. The idea of using a crossover function is to generate a better solution (a more fit one) for the next generation. In order to make sure that the best solutions are not destroyed in the process, the n top solutions from generation i is copied into generation i+*1*, this practice is called elitism and is optional [19].

As mentioned above, a crossover function is a regular implementation when pairing up the parents' genes. However, there are other such functions that can be used as well. For example uniform scanning or fitness scanning [20].

Uniform scanning works by bitwise operation of the parents' genes. The ith bit of the child has a 50 percent chance of coming from either parent's corresponding bit.

Similarly, fitness scanning works by bitwise operation. However, the fitness function is used here to determine what chance each parent has to pass on their bits to the child. If parent i has a fitness of $f(i)$ the probability $P(i)$ of it passing on its genes is $P(i) := \frac{f(i)}{\sum f(i)}$ [20].

In the case of this project, there are only two parents, resulting in a ratio between their fitness scores. This means that the expected number of bits $E(i)$ inherited from parent i is: $E(i) := P(i) * \text{String length}$

As evolution progresses with a certain degree of randomness, a mutation function is used to introduce random mutations to the genes. The mutation function goes through every gene and changes a random bit. This is done to create new variations of genes that otherwise would not have been created by only using the selection and crossover functions. The steps above are repeated for each generation. Eventually, the difference in the scores given by the fitness function will decrease between each generation. As this difference becomes negligible, the genetic algorithm has reached its goal by producing a result that could possibly be the best one.

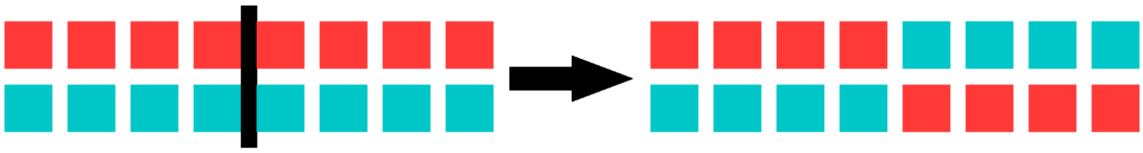


Figure 2.3: An arbitrary crossover of two genes where one of the genes to the right will belong to the child. The red and blue genes to the left originate from the parents. Here the cutoff is at four bits, resulting in two new mixed genes with four bits from each parent.

The main downside of implementing a genetic algorithm is that the complexity of the components, mainly the fitness function, can contribute to the genetics becoming difficult to model. Another issue with the genetic algorithm is that the solutions tested are based on previous answers with some offset. If this offset is too small, the given result might not be the best result as the best solution could be missed.

Evolution strategy is another algorithm that is very similar to a genetic algorithm [21]. It could solve these issues in a better way, however, it is more complex. But for this project, it would have taken a lot more time to learn and implement it in a good way than the genetic algorithm.

2.3 Ecosystem Modeling

There exists many methods used to describe an ecosystem. Some of these methods are presented in this section.

2.3.1 Lotka-Volterra Model

The Lotka-Volterra model is a common model for describing the relationship between the number of animals in an ecosystem [22]. The model makes the following assumptions about the ecosystem: the ecosystem contains only two species, a predator and a prey (e.g. rabbits and wolves). The death rate of the rabbits is proportional to the number of wolves. The birthrate of the wolves is proportional to the number of rabbits.

The model is described by the following formula, where R is the size of the rabbit population and W the size of the wolf population.

$$dR/dt = \text{rabbit birth rate * R} - \text{rabbit death rate} * R * W$$

$$dW/dt = \text{wolf birth rate} * R * W - \text{wolf death rate} * W$$

This model is displayed in Fig. 2.4

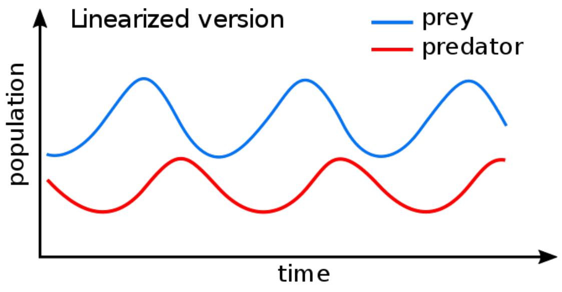


Figure 2.4: Example illustration of the Lotka-Volterra model [23]. License available at: https://creativecommons.org/licenses/by-sa/4.0/

The Lotka-Volterra model describes the dynamic equilibrium of an ecosystem that has the same properties as the simulation presented in this paper [22]. It is therefore interesting to see whether the resulting equilibrium from the simulation will follow a trend similar to a Lotka-Volterra model.

There are extensions of the Lotka-Volterra model which are more realistic for extreme values such as the nonlinear model worked on by G. F. Gause [22]. G. F. Gause's work is more realistic when the population of prey is high for example. Ultimately, a more realistic model is not needed to compare with the graphs produced by the simulation as the ecosystem is simplified.

2.3.2 Pearson Coefficient

To further investigate the relationships in the resulting equilibrium, the Pearson correlation coefficient will be used to evaluate if there is a correlation between entities in the ecosystem [24]. A correlation of -1 means that the two variables are linearly opposite, while a correlation of 1 means that the linear relationship is perfect. Lastly, a value of 0 means that they have no correlation.

2.4 The Unity Platform

The Unity real-time development platform is a game engine used for making a variety of 2D and 3D games, as well as a wide range of other types of applications in a variety of industries [25].

The Unity platform uses the Unity editor. The Unity editor gives the developer a diverse set of tools and applications, both built-in and downloadable, to be able to create anything the developer has in mind.

There are many reasons why Unity is a good tool to use for making different types of applications. Because of its popularity within the gaming industry (both for indie developers [26] as well as industry giants [27]), there is an enormous amount of material coming from tutorials, information, discussions on forums, Youtube videos, the Unity asset store, etc. This provides developers with a great source for answers to any problems that might arise during development. This has been helpful throughout the process of creating the simulation in Unity since it has been easy to find information on how to solve commonly occurring problems which have been encountered.

Other popular game engines are Unreal Engine and Godot. Unreal Engine is an advanced real-time 3D creation tool that allows scripting with C++ or visual scripting with "Blueprint" [28]. Since Unreal Engine uses C++, it can be more fitting for performance requiring tasks such as photo-realistic games [29]. However, as the visualization of the simulation in this project will be simple and stylized, the visual performance is not something that is of utmost concern. It is also very likely that performance issues will be caused by inefficient implementations rather than limitations of the programming language. Another reason Unreal Engine was not chosen was due to the inexperience of C++ within the group.

A popular and rapidly growing game engine is Godot [30]. Godot is a free and open-source game engine with 2D and 3D support, which allows scripting with GDscript (Godots own scripting language), C#, and Visual Scripting. It also has community-built bindings for other popular languages such as Python [31]. Since Godot has great 3D support and supports C#, it would be an equally good fit as Unity. However, Unity was chosen due to its maturity, popularity and because it was implied in the project description.

2.4.1 Terrain Tool

The terrain of the simulation was built using Unity's terrain tool, which is a built-in function within the Unity editor [32]. The tools can be used for customizing any terrain within any given project. For example, it is possible to lower/raise the ground surface, paint textures on the surface, and quickly add trees and other objects in bulk instead of adding them one by one.

The other options would have been either to piece together several different pre-made planes, procedurally generate a world, or to use a heightmap which is essentially a copy of a real-world place. The reason for the terrain tool being chosen was because, unlike the other options, this was the best way to make a unique but still realistic terrain. The other options could also result in almost the same results, however, with the piece-together option it would be hard to find good matches for the different planes while maintaining good edges. The results of eventual procedural generation would be difficult to predict since the environment could potentially negatively impact the simulation. Furthermore, the heightmap could end up being too steep at some places and cut off some parts of the world. These problems can all be avoided by using the terrain tool. The terrain tool also has built-in support for the NavMesh functionality which is described in the next section.

2.4.2 NavMesh

1.3 The NavMesh is a downloadable functionality for Unity that is used for making game objects move on walkable surface areas. The NavMesh consists of several components to provide the developer with additional control over the functionality of the NavMesh itself [33]. The NavMesh surface is one such component. It represents the surface area which a game object can move around on. The process of building the surface is called NavMesh baking. This process is performed by the Unity editor. During baking, the process gathers information about the meshes that make up the geometry of the terrain surface (incline, trees, lakes, etc.), and produces a walkable surface area from that information [34]. The walkable surface is the turquoise layer visible in Fig. 2.5. A NavMesh agent is attached to the game object itself. The NavMesh agent then uses the NavMesh surface to move the game object [35].

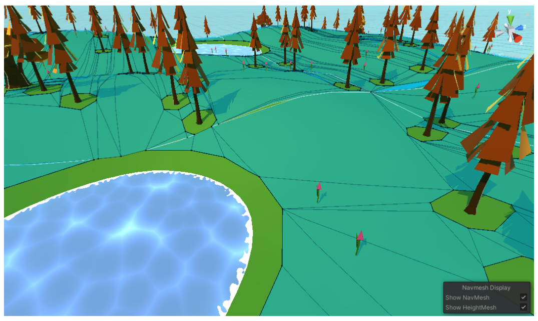
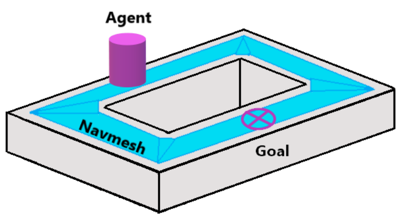

Navmesh Display
Show NavMesh ✓
Show HeightMesh ✓

Figure 2.5: The walkable turquoise surfaces built with the NavMesh baking functionality.

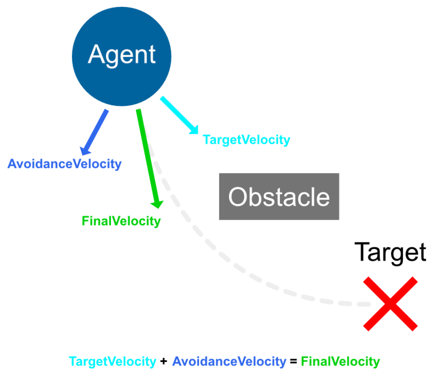
The reason why one might choose to use the NavMesh for pathfinding in a project like this is that it is easy to use since most of the work is performed by the editor. In order to make a NavMesh agent move to a point on the NavMesh it only needs to call the GoTo() method which comes with NavMesh components. The NavMesh does however come with some drawbacks such as on surfaces with a steep incline, the game objects have a risk of getting stuck in the terrain. For more complex surfaces, a 3D matrix might be better suited (further explained in Section 6.5). Additionally, the NavMesh comes with some performance constraints as it is not optimal for handling a large number of entities.

The Unity NavMesh consists of a set of connected walkable vertices (see Fig. 2.6a) [36]. These vertices form a convex polygon (a polygon that does not intersect itself).

With these polygons, the NavMesh can calculate paths between two walkable points. The NavMesh calculates its paths with the A* (A star) algorithm [36]. The A* algorithm is an efficient search algorithm that can both find the optimal solution (shortest path) and good estimates of the shortest path depending on the chosen heuristic [37]. The NavMesh calculates a path (sequence of polygons) which the NavMesh agent moves through. The agents move towards an edge of the next polygon.



(a) Visual representation of NavMesh and Agent.



(b) Simplified visualization of Unity's obstacle avoidance.

Figure 2.6: Visualisations of different aspects of the Unity NavMesh.

Unity's NavMesh also handles collisions and collision avoidance between obstacles and other agents. The collision avoidance is implemented using "reciprocal velocity obstacles" (RVO) [36]. RVO moves towards a desired position whilst steering away from possible future collisions (see Fig. 2.6b) [38].

2.4.3 Collision Handling

Unity provides components called colliders which can be attached to game objects [39]. These are used to check if different objects collide with each other. The animals in the simulation have vision and hearing senses (further discussed in Section 4.1.3). To accurately represent their reaction of seeing and hearing other objects, colliders are attached to the areas representing the senses so that a trigger can be created once a collision occurs. Additionally, colliders are attached to all food resources and animals. This ensures that when objects happen to enter within these sensory areas, it triggers a collision, and the information about the triggering item is sent to the appropriate method.

2.4.4 Particle Systems

Particle systems are commonly used to create a visual effect that represents non-static visual elements such as water or smoke. The particles can be modified to work in specific ways by using a special module connected to each particle system [40]. Using the particles module, it is possible to change the number of particles emitted and the behavior of these particles. In general, particle systems are used to make games feel less static and to give the player satisfying visual feedback.

In this simulation, the particle systems are used in order to visualize what the animals are doing and how they interact with the environment (further discussed in section 4.3.3).

2.4.5 Render Pipeline

A render pipeline is used to take the content within the scene and render them on the screen. Unity has three prebuilt render pipelines to choose from HDRP (High Definition Render Pipeline), URP (Universal Render Pipeline), and the already built-in render pipeline which functions as the standard pipeline. A developer also has the option to create a custom pipeline if preferred [41].

The simulation is rendered using URP. This was chosen because, unlike the standard pipeline, it allows for the use of shader graphs. It is also possible to add effects onto game objects, and building these effects is easier using graphs, rather than having to code it in scripts. Shader graphs give the developer a better overview of how the different components connect (Fig. 2.7), it is thus easier to use for developers not experienced in creating shader effects. Furthermore, URP was chosen over HDRP because HDRP is suitable for graphics-heavy games running on computers with appropriate hardware that can handle more complex calculations [42]. Also, the URP makes it possible to add post-processing effects to the project. The difference between shaders and post-processing effects is that shader effects are calculated at runtime, while post-processing effects are added to the rendered frame, to simulate camera and film effects [43] [44]. URP has several built-in post-processing effects, but custom effects can be added if needed [45].

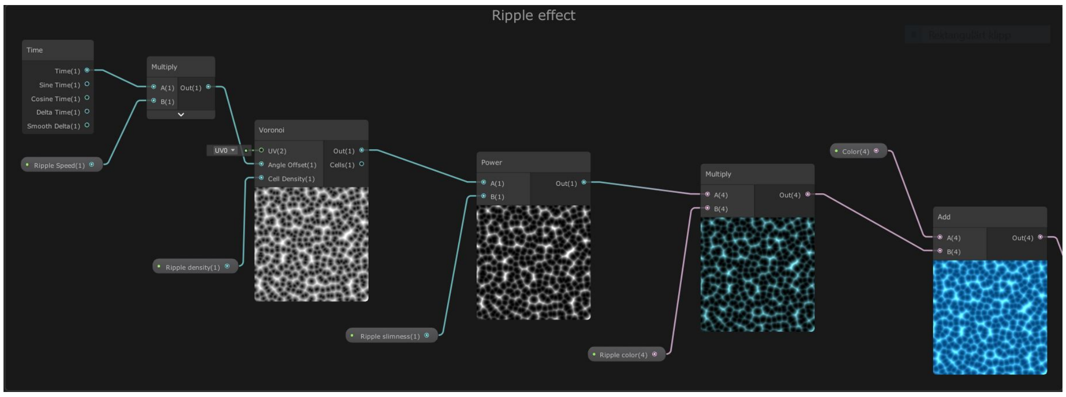


Figure 2.7: The part of the water shader that creates the ripple effect visible on every water source.

3 Related Work

During the course of the research for this project, it has become clear that the field of simulation and evolution is broad, and that there are many types of works in which simulation of creatures, ecosystems, and evolution are the primary topic. The reasons for this could be many. First, research can be conducted in various ways. For example, should an existing ecosystem be simulated? Or should new kinds of creatures be given the opportunity to freely evolve, based on well-defined criteria? Secondly, most of the research tends to focus on a single aspect of an ecosystem. For example by only simulating evolving creatures [46][47], focus exclusively on how to best model, simulate, and visualize trees [48], how to fastest render terrain or terrain objects [49][50], how to most accurately simulate water [51], focus on researching the implementation and application of genetic algorithms [19], etc. Notably, simulations are a very popular genre of games as well. This includes everything from car and flight simulators in varying degrees of "realness", life simulators, to simulations developed for new technology such as virtual reality (VR) and augmented reality (AR). Presented below are some work that has influenced the simulation presented in this paper, games that influenced the simulation genre and the rising popularity of VR games, as well as work that presents research in depicting simulated evolution.

3.1 Genetic Algorithms and Their Applications

The paper *A Study on Genetic Algorithm and its Applications* by Haldurai et al. explains genetic algorithms and the components of them [19]. This is where we got the idea about having bit strings as genes and how to do the crossover between parents. In this paper, the authors go through the pros and cons and usages of a genetic algorithm. The different parts of a genetic algorithm are also covered and explained and with these there were a few alternatives covered as well. The alternatives for selection and crossover functions have been a great help in understanding genetic algorithms. The examples use bit strings which were easy to implement and adapt to the simulation.

3.2 Simulating Low-poly Ecosystems

Early on in the project, a resource that helped the simulation immensely was a Youtube-video *Coding adventure: Simulating an Ecosystem* made by Sebastian Lague [52]. It is a video that depicts a simulation of an ecosystem. Similar to our work, this is a graphically simple (low-poly) simulation containing rabbits and foxes. This work helped with conceptualizing the simulation presented in this paper: the animals in the video have their hunger and thirst represented as bars hovering above them that tick down as time passes. Below the bars, the current state of the animal is visible. In the early stages of this project, bars and current state were implemented to be visible in the same way but were later changed due to a large number of animals being present in the simulation. Furthermore, the overall minimalist look of the ecosystem in the video was a look thought to be suitable for our simulation.

3.3 Simulations in Games

In the field of video games, simulations are a popular genre. A well-known game within this genre is *The Sims, 2000.* The Sims is a game where players control their own created person [53]. This concept of designing your own human would later evolve into creating your own creature, in the 2008 game Spore [54]. Here, the user would actively be a part of the evolution from a simple cell to a complex creature by designing and redesigning a creature between different stages of evolution.

Equilinox is a game in which the user controls an environment containing animals, plants, and other abiotic factors [55]. The user can control the placement of plants, trees, rocks, and different kinds of animals. The plants and animals will then evolve based on the environment they live in. It is unclear if genetic algorithms are included in this game for the evolution of plants and animals. It might be that the "evolution" is modeled to be deterministic; that combining certain entities is pre-determined to give a certain result. The look and feel of *Equilinox* have been an inspiration when creating the visuals for our simulation.

3.4 Explaining Evolution

Natural selection and evolution is a complex subject. If one is not well-versed in it, it is difficult to accurately describe how it works. A Youtube video series that succeeds in explaining how natural selection works is the *Evolution* series made by Primer [56]. The series does not present new ideas or findings regarding natural selection or evolution. However, the series introduces experiments and scenarios to the viewer, and subsequently explains the results, and presents improvements to reach a greater accuracy in modeling natural selection. The video *Simulating natural selection* (a part of the series mentioned above) was especially of great help at the beginning of this project, as it focuses on the animals' speed and size and would therefore be useful information to plot for the results given by the simulation.

3.5 Self-evolving Creatures

A paper that presents an ecosystem with further complexity is *Evolution of a Complex Predator-Prey Ecosystem on Large-Scale Multi-Agent Deep Reinforcement Learning* by Yamada et al. [47]. The ecosystem presented is relatively simple in its presentation, with 2D squares representing the animals and larger squares surrounding the predators to represent their predation area (the area where they can catch their prey). However, by using reinforcement learning, a type of machine learning, the researchers have developed a complex system capable of replicating dynamic properties of predator-prey dynamics. Their results indicate that reinforcement learning is particularly effective in improving the survivability of the preys. If an evolutionary algorithm is used in conjunction with reinforcement learning, the survivability is passed on to following generations.

Machine learning can be used to make programs or objects gradually improve their accuracy or knowledge. A correctly implemented machine learning algorithm can even exceed human knowledge [57]. With that said, there exist multiple machine learning algorithms, and finding the right one for a certain problem may be difficult. Furthermore, implementing a machine learning algorithm can be both difficult and time-consuming if one does not have the proper experience in implementing and debugging such algorithms [58]. Because of our limited knowledge in machine learning, as well as because of time constraints, machine learning was deemed to be out of scope for the project presented in this thesis.

4
Model

The ecosystem consists of several components which include animals and other entities in the form of water and plants. These actors and entities form a simple food chain, similar to real life. The actors of the ecosystem have their own goals and functionality and affect each other. They search for their specific resources needed to survive and procreate. This chapter will describe the different components of the ecosystem in depth.

4.1 Animals

The simulation includes two species of animals: rabbits and wolves. The reason why these species were chosen was that their interaction is easy to understand as the wolves want to eat the rabbits and the rabbits want to stay away from the wolves. The animals were created to be a simplified version of their real-life counterparts, while still maintaining some accuracy in traits and behaviors.

4.1.1 Health, Nutrition, and Energy

The animals have the properties health, saturation, hydration, and stamina. These properties are visualized in Fig. 4.1. The saturation and hydration decrease at a fixed rate as the animal grows hungry and thirsty. At the beginning of the project, these were the only properties the animal had. However, the animals died instantly if their nutritional values reached zero, therefore a property was created that was modeled as health. The final property added to the simulation was stamina. This was a way of giving the animals different perseverance. The stamina decreases whenever a wolf hunts a rabbit or when a rabbit flees from a wolf (Subsection 4.1.2), and it increases whenever the animals do something else.

As time progresses, the animals become dehydrated and their saturation decreases. This will cause the animals to seek out food and water. When the animals reach a water or food source, they begin to eat or drink, thus increasing their hydration and saturation. If either of the properties reaches zero, it negatively impacts their health due to malnutrition. If an upper threshold (50% of animal's max hydration and saturation values) is met by both hydration and saturation, the animal will regenerate lost health. This threshold also needs to be met in order for the animals

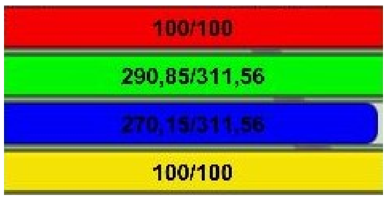


Figure 4.1: The progress bars belonging to an animal. Starting from the top: health, saturation, hydration, stamina.

to be able to mate. The rates which these properties increase/decrease will be discussed further in Section 4.1.4.

4.1.2 Behavior

To create life-like behaviors, the animals have an internal finite state machine (Section 2.1) with species-specific states and transitions (Fig. 4.2 for the wolves' states and Fig. B.2 in appendix for the rabbits' states). The purpose of each state is to represent the real-life behavior of the animals. At the beginning of the project, the state machine only consisted of the wander state. This functioned not only as a means for the animals to move around, but also to make sure the state machine worked as intended. As the project progressed and food and water were added, so was the pursue food, pursue water, hunt, eat and drink states. The dead state and flee state were added in conjunction with the hunt state, to allow the rabbits to flee from an incoming attack, and to make it so the animals could die. The pursue mate and birth state was added in order to facilitate reproduction. Search world state was added to allow for longer traversal across the world, making it easier for the animals to find food and water. The introduction of the idle state was for cosmetic reasons. Without it, the animals moved quite erratically, which did not look good.

Wander and **Idle** states are the two default states that the animals will be in if they are neither hungry nor thirsty, have not seen a mate or a predator to escape from. As the animals get progressively hungry, they will react to food and water sources and enter **Pursue Food**, **Pursue Water**, or **Hunt** states depending on their species. These states will in turn transition over to **Eat** and **Drink** states, where the animals will eat and drink until they are full or until the nourishment given from their respective food sources is exhausted. If an animal is well-fed and discovers a potential mate, the animal will transition to the **Pursue Mate** state. The females will get pregnant from such an encounter and will eventually enter the **Birth** state. If the animals are either hungry or thirsty and do not know the location of any food or water source, they will transition to the **Search World** state. In this state, the animals will traverse greater distances in order to find the necessary resources to survive. If the rabbits at any time encounter a wolf, they will transition to the **Flee** state. This state will cause the rabbits to run in the opposite direction of their enemy. Eventually, the animals' health will reach zero, either by old age, by getting killed, or simply by starving to death. When this occurs the animals will transition to the **Dead** state. Once the animals have reached this state, no other transitions can be made.

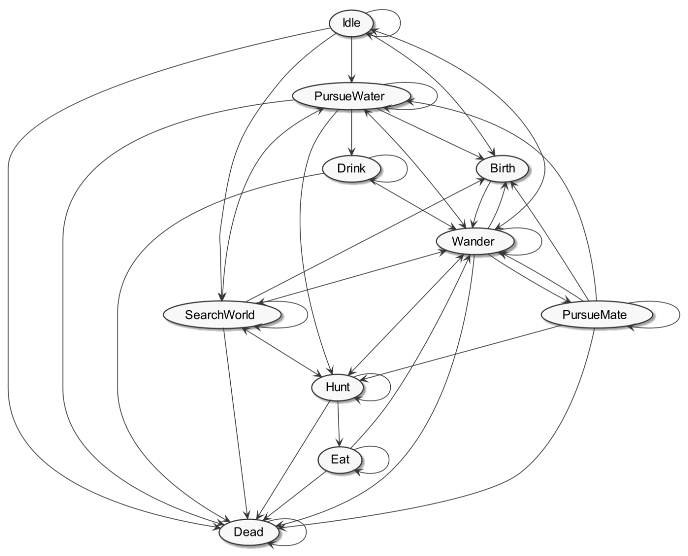


Figure 4.2: The state diagram for a wolf.

To a large degree, the animals have similar states and behaviors. However, there are a few distinct differences:

<u>Rabbits:</u> Since rabbits are (predominantly) herbivores [59], they survive by living off the plants in the simulation. When a plant is detected it is stored in the rabbits' memory and once it is hungry it will go to one of the plants which it has previously seen. They are also able to enter the flee state in the FSM, in order to have a chance of escaping an incoming attack from a wolf. Fleeing will cause the stamina bar to decrease (Section 4.1.1). When the stamina bar reaches zero, the rabbits are more vulnerable to attacks since they cannot run as fast anymore. The rabbits are slower than the wolves in general but they have better endurance. Furthermore, rabbits are notorious for their reproduction [60]. This is represented by a short gestation time as well as a short time between pregnancies (see Section 4.1.4 for details) and a litter of up to 12 rabbits, allowing them to propagate at an exceedingly high rate. Their lifespan in the simulation is 15 days.

<u>Wolves:</u> The wolves are the apex predators of the simulation and their only source of food is the rabbits. If they are hungry they will start chasing any rabbit within their field of view/hearing. If the rabbits are within physical range they will attack the rabbit, hurting the rabbit with a set attack damage, until the rabbit is dead and can be eaten. Whenever the wolves are in the hunt state their stamina bar will decrease. When the stamina has reached zero, they are unable to continue pursuing their prey and will be forced to leave the hunt state until their stamina has been recharged. The wolves have a gestation time approximately double that of the rabbit, as well as a longer time between pregnancies (see Section 4.1.4 for details) and a litter of up to 4 wolves. This gives the wolves a lower speed of reproduction although they have a longer lifespan of 30 days in the simulation.

4.1.3 Senses

The animals use auditory and visual senses to find food and water while traversing the world. They can be seen in Fig. 4.3. The senses, which are implemented using Unity's collision handling (Subsection 2.4.3), function such that when an object enters the area representing a sense, the Unity method `OnTriggerEnter()` is invoked [61]. Inside this method, several checks are performed in order to make sure the animals react correctly to the triggering object. For example, rabbits react when their *visual areas* collide with plants, water sources, other rabbits (for mating), and wolves. They also react to other rabbits and wolves when their *hearing areas* collide with such objects. Plants and water sources (in the simulation) cannot be heard and thus cannot be reacted upon with the auditory sense. Pseudocode representing the behavior can be seen in Fig. 4.4.

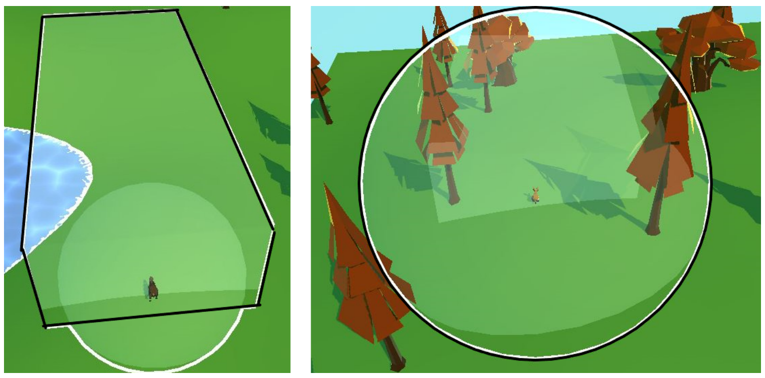


Figure 4.3: Visual representation of the animals' eyesight (left) and hearing (right). Unity handles collisions with the areas to represent animals seeing or hearing something.

```
object_entered_sensor_area(object):
    if object is plant:
        eat_plant(object)
    else if object is animal:
        notice_animal(object)
```

Figure 4.4: Pseudocode for the behavior of the senses.

4.1.4 Traits

The animals have a set of traits that differ between the species. There are two different types of traits, dynamic traits which change over time through evolution and which vary between individuals of the same species. The implementation of the evolution for the dynamic traits will be presented in section 4.4. The second type is the static traits which only differ between species and do not change over time. The reason why some traits are static is to reduce the complexity of the simulation. The traits are shown in the table 4.1. The values of the traits were originally set to replicate the real-life traits of the animals. They have been tweaked during development to try to reach an equilibrium between the species. One example of a tweak that does not correspond to real-life animals is that the wolves get hungry much faster than the rabbits. This is to make sure that the population of the wolves starts to decrease quickly once there are few rabbits in the simulation and thus preventing the rabbits from going extinct.

Table 4.1: This table shows the trait values for the wolves and the rabbits.

Animal	Rabbit	Wolf
Static traits		
Thirst decrease factor	0.29	0.25
Hunger decrease factor	0.33	0.56
Stamina decrease factor	4	7
Stamina increase factor	5	3
Fertility time (hours)	78	190
Max animal litter size	12	4
Pregnancy time (hours)	11	27
Hours between pregnancies	11	190
Old age (days)	15	30
Running speed factor	3	3.8
Dynamic traits		
Speed average	1.2	1.2
Size average	1.2	1.2
Hearing percent	50	50
Vision percent	50	50

Speed and size are both independent of each other. These traits affect how fast and big the animals are. In addition to the normal speed, there is a running speed factor that the animals use whenever they are in the hunt state, flee state, search world state, pursue food state, or pursue water state. In these states, the animals run faster than usual. As mentioned in (4.1.1) the animals have different values for their nourishment. The hydration and saturation depend on the animal species as well as the size and the speed of the animal.

$$\text{Max saturation} = \text{size}^3 * 100$$

$$\text{Max hydration} = \text{size}^3 * 100$$

$$\text{Hydration decrease} = (\text{size}^3 + \text{speed}^2) * \text{Thirst decrease factor}$$

$$\text{Saturation decrease} = (\text{size}^3 + \text{speed}^2) * \text{Hunger decrease factor}$$

For the max values, a base multiplier of 100 was used to get a readable scale. Furthermore, it seemed logical that if an animal grew in size then the stomach would grow in three dimensions, and that is why the size cubed factor is used to set the nourishment max values.

The nourishment decrease rates were inspired by the law for kinetic energy $E = mv^2/2$ [62]. In the decrease formulas, it is assumed that m is equivalent to size cubed, v is equivalent to speed and then a new constant is added in order to tweak the performance of the species to get a good balance in the ecosystem.

When the animals enter the flee and hunt state respectively, their stamina will decrease with a set stamina decrease factor. This factor is slightly higher for wolves, to prevent them from catching up to the rabbits too often. To compensate for having a higher decrease factor, they have a higher running speed factor which makes them a bit faster than the rabbits. The stamina increase factor is the rate at which stamina is recharged once the animals leave the flee and hunt state.

The animals' hearing and vision senses are dependent on each other. The animals are modeled to have a fixed amount of brainpower. The distribution of this brainpower to either get a better vision or better hearing is dependent on the genetic algorithm. The values for the distribution range from 0 to 100 percent. This is because it was difficult to define the energy cost for having a good vision/hearing. The data from the simulation will show which of these traits is more favorable (see Section 5).

The number of children an animal can birth at a time is set by a random function that ranges between one and the max number of children trait. The intervals for gestation and time between pregnancies are not set to replicate the accurate gestation and pregnancy time interval of the animals in real life but are rather set in such a way to increase the chances of reaching an equilibrium in the ecosystem. All traits connected to the reproduction of the animals were initially set to replicate those of the real animals. However as the project progressed, it was necessary to tweak these in order to make an equilibrium between the two species more likely to occur.

The old-age trait is a threshold for when the performance of the animals starts to decline. After the animal has passed the threshold, its speed will decrease by 20% each day. The animal will die automatically when its speed has dropped below 0.1 if it has not already died from starvation or thirst. This trait has also been tweaked in order to fit the ecosystem in the simulation.

4.2 The Environment

This section covers the environment of the simulation.

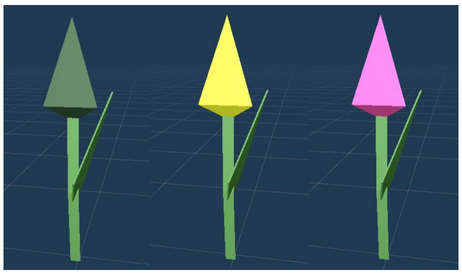
4.2.1 Plants

Plants are a vital part of the ecosystem since they make up the basis of the food chain. Similar to the animals, the plants have their own internal finite state machine (Section 2.1). However, the state machine belonging to the plants is simple compared to the animals' state machine, as it only has three possible states (Fig. 4.5a).

Plants begin in a seed state which is followed by a growing state after a certain number of days. Plants that are in the growing state can be noticed by the rabbits, but are not yet eatable. After some time, the plants have increased in nutritional value and enters the mature state. Plants can be consumed while in the mature state, where they continuously increase in nutritional value until they reach their maximum capacity. Plants that have lost their nutrition due to being fully eaten by an animal will transition back to the seed state. The different states of the plant can be seen by the color of the flower where the seed, growing, and mature states are represented using different colors (Fig. 4.5b).

$$\begin{tikzcd}
\boxed{\text{Seed}} \arrow[r] & \boxed{\text{Growing}} \arrow[r] & \boxed{\text{Mature}} \arrow[ll, bend right]
\end{tikzcd}$$

(a) The state transition diagram for the plants.



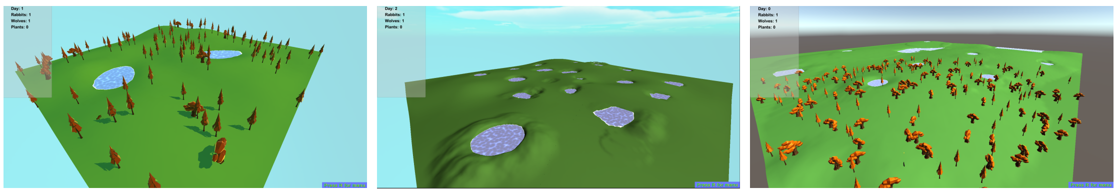
(b) The seed, growing, and mature state is represented in green, yellow, and violet color.

Figure 4.5: A visualization of the different states which a plant can be in.

4.2.2 Terrain

The terrain in the simulation was made to represent a real-life forest with trees, changes in amplitude, and waterholes. A forest is a natural habitat for both rabbits and wolves [63] [64] and was therefore deemed appropriate to have as an environment. The terrains were created using the Terrain-tool in Unity (Subsection 2.4.1).

There are three sizes of terrains to choose from: a small, a medium, and a large-sized terrain. The simulation started off with only a small world with the size of 150x150 game engine units. It has two big lakes which are easy to find, see (Fig. 4.6a). The small world had some performance issues because animals were in close proximity to each other, causing a lot of calculations to be done for the collisions between the animals.



(a) The small world **(b)** The medium world. **(c)** The large world.

Figure 4.6: The three different worlds which were created throughout the project. They all have different sizes and amounts of lakes.

A larger world was subsequently created to solve the performance issues from the small world. The large world has a size of 500x500 game units and has 11 lakes of different sizes scattered across the world, see Fig. 4.6c. Unfortunately, the large world also proved a bit troublesome as the size of the world required a greater number of animals in order for them to be able to find mates. This was especially noticeable for the wolves. The distance between the lakes caused the animals to constantly search for water sources. It also introduced a higher cost for calculating paths for the animals that overloaded the NavMesh (Further discussed in section 4.5.1).

Finally, a medium-sized world was created in order to fix the different problems the other two worlds had. The medium world has the size of 300x300 game units and has 16 lakes positioned across the world, see Fig. 4.6b. The higher density of lakes makes it easier for the animals to find water sources. The medium-sized world is also big enough to avoid the high amount of collisions from the small world. Another new aspect of the medium-sized world was that the trees were removed. This was an attempt to reduce the complexity of the path calculations, as the trees make the walkable Navmesh more complicated. However, it did not have a significant impact on the performance.

4.2.3 Terrain Effect on Animal Behavior

The original plan was to make it so that the terrain itself does not impact the behavior of either animals or plants. Instead, it works as a limiting factor where the smaller terrain is too small to sustain a large population of animals and plants. This is mostly true but there is one exception. Whenever the wolves enter the search world state and the rabbits enter flee state (Section 4.1.2) they forget any remembered water source location for a period of time, or until they see a new water source. For the small world, the wolves forget the lakes for 66 hours in the simulation and for the medium-sized world, they forget them for 20 hours. Rabbits forget the water source for 1 hour on both the small and the medium world. The difference in time between the two worlds depends on the number of water sources and the distance between them. The large world currently does not make the animals forget any water location, since the animals have a difficult time surviving in that world. The fact that the animals forget the lakes makes them able to explore the world more which enables them to find new lakes. It also prevents the wolves from always staying next to a lake all the time which gives the rabbits a chance to reach the water without being chased away. This is especially notable in the small world where there are only two water sources. It also prevents the rabbits from continuously running back and forth from a water source if there are wolves gathered near it. The rabbits forget the water source for a much shorter time compared to the wolves because the rabbits enter flee state every time they encounter a wolf. Thus, the time they forget a water location accumulates faster for the rabbits.

4.2.4 Time in the Simulation

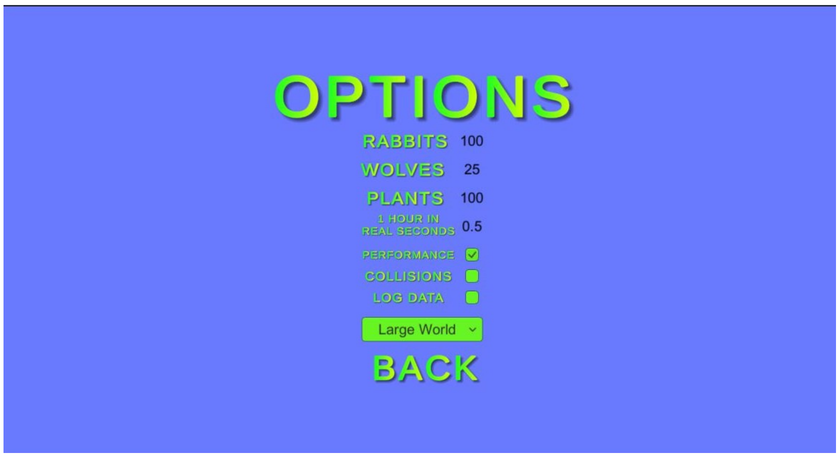
There are two different units of time used in the simulation: hours and days. One hour in the simulation is equivalent to 0.5 real-life seconds. One day is 24 hours in the simulation and is thus equivalent to 12 real-life seconds. The relation of in-simulation hours to real-life seconds is configurable to allow running the simulation faster or slower. However, this is not changed in any simulations in the report.

4.3 User Interface & Visual Effects

This section covers the user interface and visual effects of the simulation.

4.3.1 Menu

The simulation is configured and launched via a start screen. The screen has three buttons, "Start", "Options" and "Quit". Under the "Options" menu, the user can configure the number of species and the number of plants to be included in the simulation. The timescale of the simulation can also be changed. Moreover, users can enable/disable a few extra functionalities such as running the simulation in performance mode (no animation or visual cues will be played), enable collisions between animals (collisions require more computations and thus has a bigger impact on performance. Note: this does not turn off collisions that trigger on food and water), or log the results from the simulation (used to gather data). Finally, the user can specify which world to run the simulation in.


OPTIONS

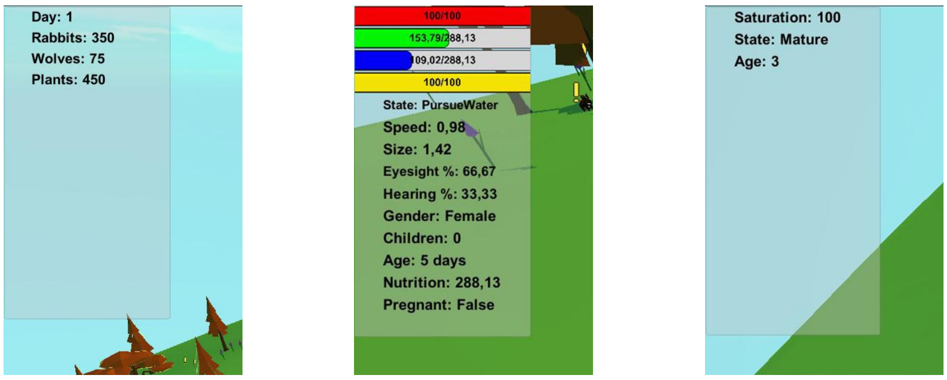
RABBITS 100
WOLVES 25
PLANTS 100
1 HOUR IN REAL SECONDS 0.5
PERFORMANCE ✓
COLLISIONS ☐
LOG DATA ☐
Large World

BACK

Figure 4.7: The options-menu of the user interface.

4.3.2 Overlay

An overlay was added in order to display the statistics of the ecosystem. It displays how many entities there are of each species, as well as how many (simulated) days have passed since the simulation was started, see Fig. 4.8a. When an animal or plant is clicked, the statistics panel will instead show the stats of the clicked entity.



Day: 1
Rabbits: 350
Wolves: 75
Plants: 450

100/100
153,79/288,13
109,02/288,13
100/100
State: PursueWater
Speed: 0,98
Size: 1,42
Eyesight %: 66,67
Hearing %: 33,33
Gender: Female
Children: 0
Age: 5 days
Nutrition: 288,13
Pregnant: False

Saturation: 100
State: Mature
Age: 3

(a) Nothing selected. **(b)** Selected wolf. **(c)** Selected plant.

Figure 4.8: Three different overlays for when inspecting different parts of the ecosystem.

The statistics panel was added to provide context to the different aspects of the ecosystem. Both an overarching overview concerning the whole ecosystem, as well as detailed information about animals and plants (Fig. 4.8b, 4.8c). The information on the panel is connected directly to the ecosystem and to the entities within it. The script (Subsection 3.2) that handles the statistics panel continuously listens to changes concerning the entities it displays. This ensures instantaneous information regarding the ecosystem or the selected entity being presented. The panel for the animals can be viewed in Fig. 4.8b The four progress bars in the top left corner is displaying health, saturation, hydration, and stamina, further described in Section 4.1.1.

4.3.3 Visual Cues

Another approach to visualizing information regarding the animals is the implementation of a number of visual cues that trigger during certain events. This way, information about the animals can be communicated to the user without having to actively click on an entity. The visual cues were created using the particle system in Unity (Subsection 2.4.4). The cues help the user see and interpret what is happening in the simulation. Furthermore, they can aid in understanding why the animals decide to behave as they do without having to actively follow an animal.

4.3.4 Shader Effects

As mentioned in Section 2.4.5, the URP made it possible to include shader graphs in the project. Shader graphs are used to add additional visual effects to a surface or object. There are a total of three shader graphs included in the simulation, giving the following effects:

- A toon cel-shader on the animals that is used to give a cartoon look to them.

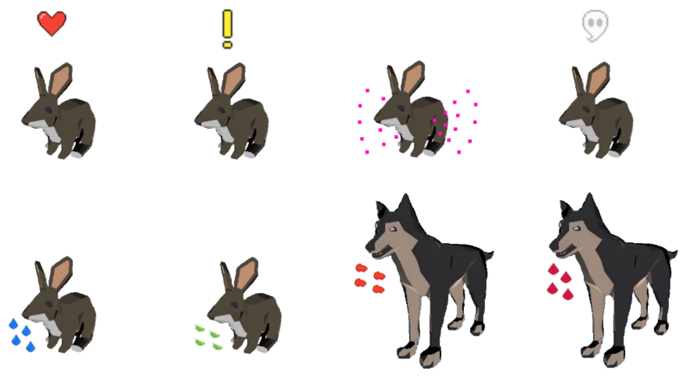


Figure 4.9: Visual cues, starting from top left: mating, flee (rabbits only), pregnancy, death, drinking, eating plant (rabbits only), eating flesh (wolves only), attacking (wolves only).

- A shader on the trees to give them the appearance of swaying in the wind. The shader also adds a dithering transparency effect to the trees, in order to not block the view of the camera.
- A shader on the water, used to give the appearance of a moving body of water, complete with foam on the edge of the shore.

Moreover, an outline effect has been added to the animals and plants, to make them easier to discover from far distances. This effect was not added as a shader, but instead as a post-processing effect. The effect works in the following way: For every animal and plant mesh, a second, larger mesh is rendered. The front face of the second mesh is subsequently culled, making the first mesh visible again. The difference between the first and the second mesh is colored black, thus giving the appearance of a thick line around the object.

All shaders and the outline effect were made using tutorials found online (see Acknowledgments).

4.4 Implementation of a Genetic Algorithm

The reason for using a genetic algorithm for reproduction is because the implementation is inspired by the process of natural selection. Thus, by implementing it in the animals, the idea is that the genes will trend towards the fittest point. This is particularly interesting since the question is what the best values of the traits, for each species, actually are. Moreover, the genetic algorithm allows several different ways of defining how genes are passed on, and how they can be mutated.

Since the best values for the traits are unknown it is a complicated task to decide a fitness function for the genes (Section 2.2), and thus it does not make much sense to use fitness scanning in the simulation. Instead, uniform scanning is being used. Both uniform and crossover scanning introduce randomness in their logic, but uniform scanning is more easily understood.

Furthermore, the mutations of the genes only occur at birth. This is simply for the sake of making conclusions about the data. If mutations could occur anytime during a lifespan, the resulting data could potentially be more difficult to interpret. The mutation is implemented as a bit flip mutation, where each bit in the gene string has a certain chance to be flipped.

Each gene represents some trait, such as speed. Thus, an evaluation method is required to calculate the value of a gene. The bit string of a gene is eight bits long, making a total of $2^8 = 256$ possible combinations. The value of a child's gene is determined by the value of the parents' genes and the number of set bits in the child's bit string. The number of set bits will determine in what interval the resulting value will be. All children's values will thus together make something similar to a distribution curve. If the child receives a bit string with zero or one set bits (9 combinations total), its value will fall below the lowest value of the parents. Similarly, for seven and eight set bits it will get a value above the highest value of the parents. Otherwise, it will get a value between the parents' values. This description is illustrated in Fig. 4.10. The intention with this is for the genes to eventually end up in a state with a distribution similar to the real world. However, if the child has an equal amount of set bits as both parents, the child's value will fall between the parents' values to stop it from increasing/decreasing indefinitely.

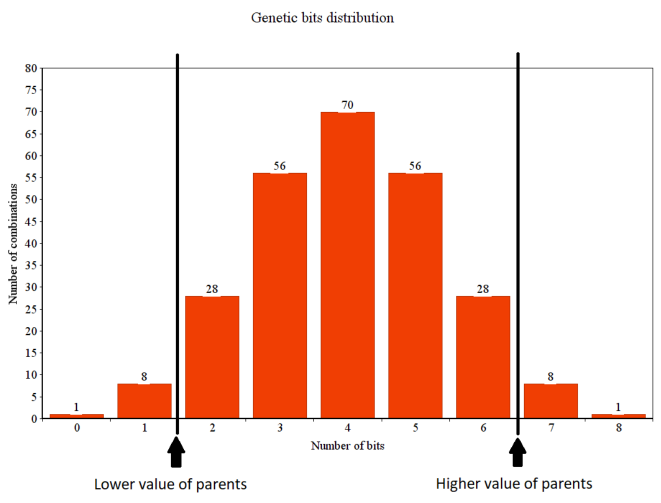


Figure 4.10: The distribution of the genes' values depending on the number of set bits. Each bar represents a value interval.

4.5 Performance Optimization

In order to make the simulation run as smoothly as possible, some optimizations had to be made.

4.5.1 NavMesh Optimization

In order for the animals to survive in the simulation, they need to be able to move to different parts of the world. For example, if there are a lot of wolves in one part of the world, the rabbits will die there. In order for the wolves to survive, they need to search the world for new rabbit hot spots. Likewise, the rabbits need to find spots free from wolves in order to survive. The simulation uses the Unity NavMesh in order to calculate paths for the animals. As the simulation progressed and the animals got a larger urge to explore the world, the NavMesh became a bottleneck as it became clear that it was not able to handle the increased amount of calculation of longer paths. To improve the performance of the navigation through the NavMesh it was necessary to reduce the number of long path calculations across the world while still allowing entities to be able to explore new parts of the world. The solution was to create a 2D matrix that represents the x and z coordinates of the world. Each animal exists within a box of this matrix and when it goes into the search world state it randomly chooses a point in one of the adjacent boxes of the matrix (Fig. 4.11). This prevents excessively long pathfinding to be done by the NavMesh and enables the simulation to run with around 1 000 entities.

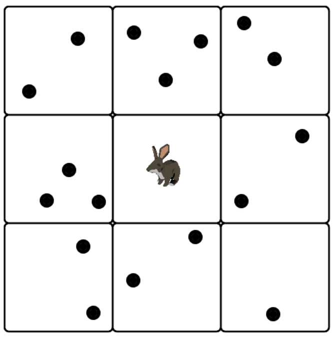


Figure 4.11: A rabbit in the search world state. The black dots represent the points the animal can choose to walk to.

4.5.2 Reduce Method Calls

The method `GoTo()` makes the animals move to a selected position on the Navmesh. While not being an expensive call if used sparingly, the number of animals that called on it made it into one of the most expensive methods when running the simulation. As such, some logic had to be rewritten in order to call the method only when necessary.

4.6 Data

To be able to analyze the simulation, data about the simulation must be collected. The animals have a wide variety of properties that generates data. The collected data includes movement speed, size, age, species, percentage of brainpower used for vision and hearing, cause of death, amount of rabbits and wolves.

The simulation gathers data via logger classes. To allow an easy and extensible way of collecting data, a generic way of gathering data was created. This allowed multiple types of data to be collected with ease. This was done through an interface for logging with two methods, `Snapshot(EntityManager)` which saves the current state of the simulation with data relevant to the specific logger in memory. This is saved for every day of the simulation. Then, the data can be persisted with `Persist()` which saves the data on the disk and removes the saved data from memory. This was done as writing and reading files can be expensive operations, so with this setup, the data could be saved at a regular interval of every day but persisted locally less often to prevent performance issues.

The different loggers include DeathCauseLogger which logs the cause of deaths for the animals, DetailedIndividualLogger which logs the current state of every individual animal, FpsLogger which logs the current average frames per second and OverviewLogger which logs the total amount of animals, plants, and the average values of the animals' traits.

The collection of data is important for analysis. The data allows discussion of whether the results gathered are analogous to real-life scenarios. The data is visualized with Plotly.py [65]. Plotly.py allows easy visualizations of data from .json and .csv files.

4.7 Additional Information

Source Code - github.com/AronSeamountain/eco-simulation

5
Results

As mentioned in the purpose of the report, it is of interest to study what effect the number of resources, the individual characteristics, and behaviors have on the natural selection of the animals. Therefore data has been gathered at selected instances of the simulation 4.6. The simulations have been selected by running several instances of the simulation with the same settings. These instances are all plotted in the graphs, and all but one of the plotted lines are faded slightly. This is done in order to give a sense of the average distribution while still displaying one instance to be able to refer to that instance specifically. The instance selected is the one that was able to persist the most amount of days without any of the species going extinct. It is important to note that the highlighted values are the same for all graphs for the same simulation settings. For example Fig. 5.1a, 5.3a, and 5.2a highlight a statistical plot from the same instance.

An in-depth discussion about the results is conducted in Section 6.3.

The reason for plotting several instances with the same settings of the simulation is because the simulation is non-deterministic. It varies by every instance even if the world and the number of entities are the same at the initiation. Furthermore, the genetic algorithm introduces random mutations and therefore the result can vary a lot between each instance.

Additionally, the simulations in the result section were run on different computers. The reason for this was because of time constraints. As such, the information in the graphs in the following sections consists of data gathered from multiple computers. This could be another cause for the variations in the result, further discussed in Section 6.1.

Every instance of the simulation were run with the following configurations on the "Options"-screen:

- 1 hour in real seconds 0.5
- Performance ON
- Collisions OFF

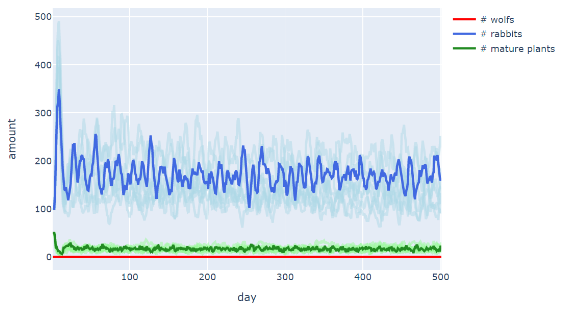
- Log data ON

5.1 Simulation of Only Rabbits

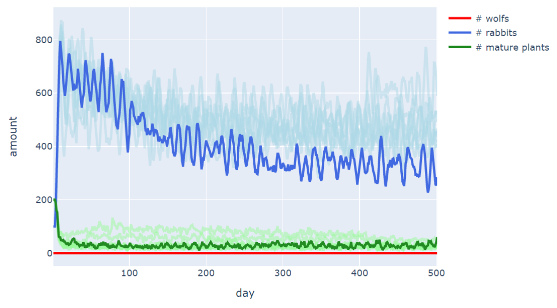
Three different types of instances of the simulation were executed with only rabbits. This was done in order to see how the number of plants affects the traits of the rabbits, as well as the number of rabbits. The simulations were run with 50, 200, and 400 plants, all in the small world and all starting with a population of 100 rabbits. The simulations were run for 500 days as this was proved to be enough time for the traits to stabilize.

5.1.1 Amount of Rabbits

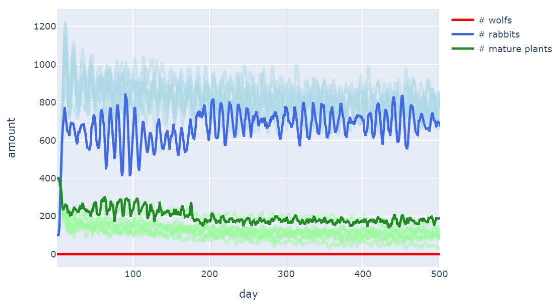
As can be seen in Fig. 5.1 a higher amount of plants can support a larger population of rabbits. 50 plants can support around 100-250 rabbits, 200 plants can support 300-600 rabbits and 400 plants can support 700-900 rabbits. The amount of mature plants stays very low in the 50/200 plants simulations which means that the rabbits find almost all plants and are constantly eating them to survive. Whereas in the 400 plant simulation there are between 150-200 mature plants at all times, indicating that there are more than enough plants available for the rabbits, see Fig 5.1c.



(a) 50 initial plants.



(b) 200 initial plants.

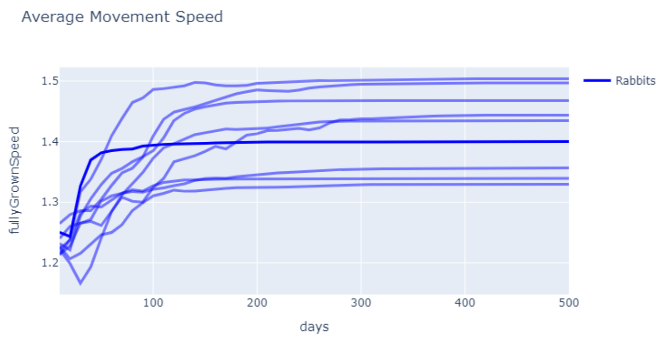


(c) 400 initial plants.

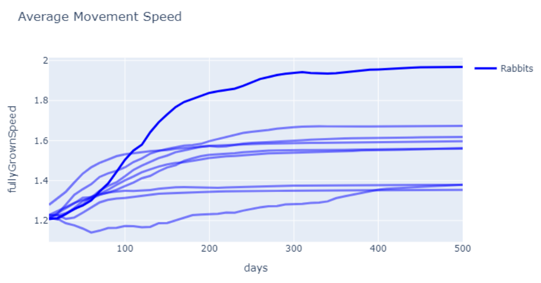
Figure 5.1: The number of animals and eatable plants with the initial number of rabbits being 100. Green indicates the number of eatable plants, blue indicates the number of rabbits and red indicates the amount of wolves (always zero).

5.1.2 Speed of the Rabbits

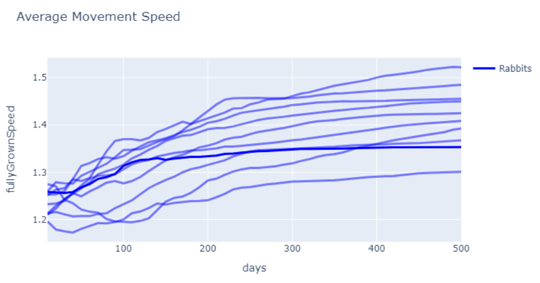
All simulations show an increasing trend in the speed of the rabbits, see Fig. 5.2. The average speed for the rabbits in the simulations with 50 and 400 plants are almost the same, see Fig. 5.2a 5.2c. They end up between 1.30-1.52. Whereas the simulations with 200 plants end up between 1.37-1.66 except for an outlier which ended up at 1.97, see Fig. 5.2b.



(a) 50 initial plants.



(b) 200 initial plants.

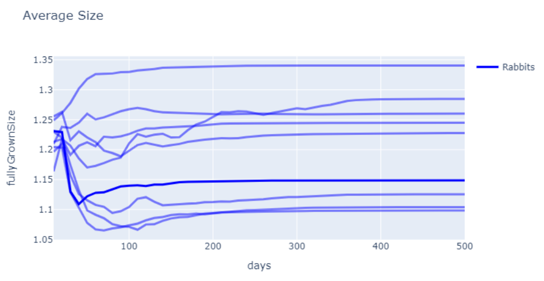


(c) 400 initial plants.

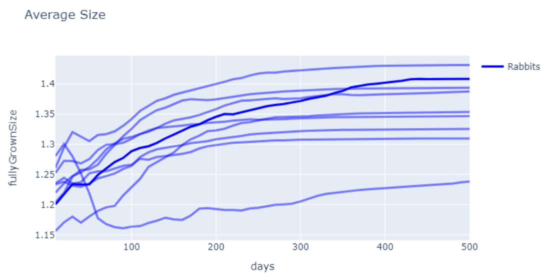
Figure 5.2: The average speed of the rabbits with the initial number of rabbits being 100.

5.1.3 Size of the Rabbits

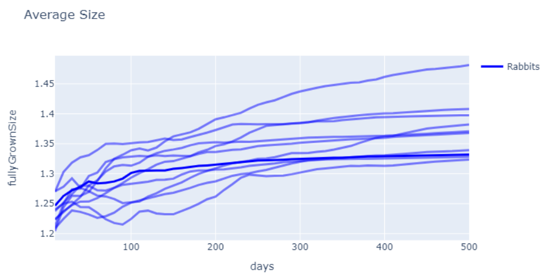
The rabbits' size distribution from the simulations can be seen in Fig. 5.3. For the simulations with 50 plants Fig. 5.3a shows some similarities with a normal distribution around the initial size of 1.2. It does not appear to be a trend that increases or decreases. In the simulations with 200 plants, an increasing trend can be seen for the size, which stabilizes between 1.24-1.46, Fig. 5.3b. Finally, the simulations with 400 plants show the most consistent increasing trend for the size which ends up at around 1.32-1.48.



(a) 50 initial plants.



(b) 200 initial plants.

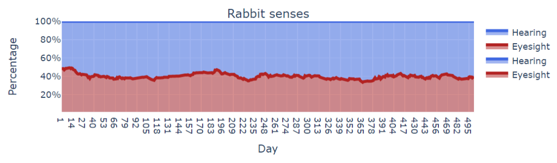


(c) 400 initial plants.

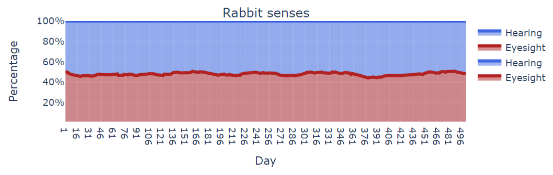
Figure 5.3: The average size of the rabbits with the initial number of rabbits being 100.

5.1.4 Vision and Hearing of the Rabbits

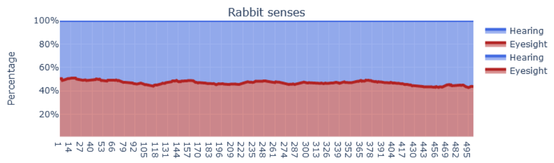
Based on the results it seems that hearing is favored over vision for the rabbits. This can be seen in Fig. 5.4. The senses do not seem to be affected significantly by the number of plants in the simulation and the percentage for hearing lands somewhere between 55-60% after having run the simulations for 500 days.



(a) 50 initial plants.



(b) 200 initial plants.



(c) 400 initial plants.

Figure 5.4: The percentage of brainpower utilized for the vision and hearing with the initial number of rabbits being 100.

5.2 Equilibrium

This section covers different simulations with both rabbits and wolves. As well as searching for equilibriums in the different worlds, a comparison is made between the genetic algorithm and no genetic algorithm by running simulations with the same initial values in the medium-sized world. This comparison can be found in Section 5.3.

The initial values were chosen as they produced the most stable equilibriums that were found during testing.

5.2.1 Equilibrium with 200 Plants in the Small World

Running the simulation with these values was proven not to be very stable. In most of the instances, either the wolves or rabbits died out within 50 days. Fig. 5.5 shows that there is no cyclic dependency between the rabbits and the wolves. Thus the graph has few similarities with a Lotka-Volterra model. All of the simulations in this subsection were run with 25 wolves, 100 rabbits, and 200 plants. The Pearson correlations of the highlighted curve are:

- Wolves and rabbits: -0.66.
- Plants and rabbits: -0.89.
- Plants and wolves: 0.71.

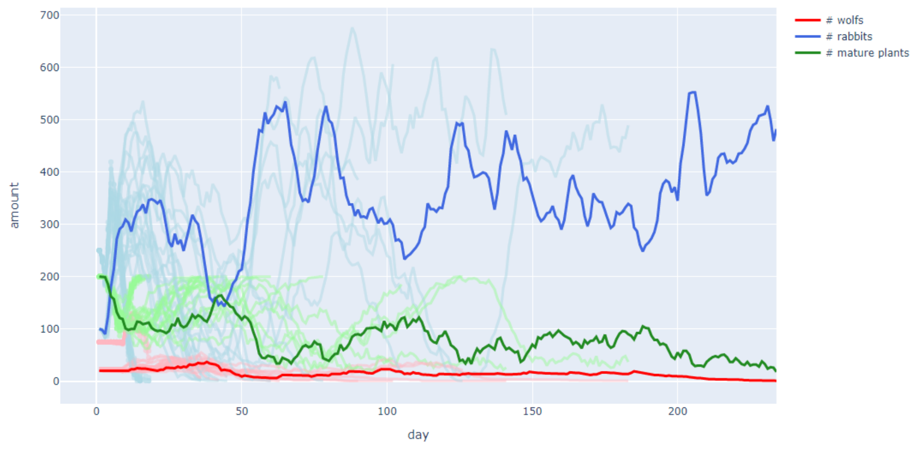
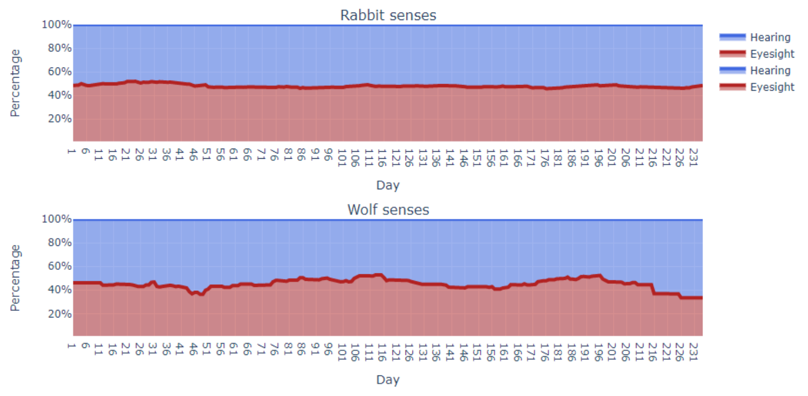
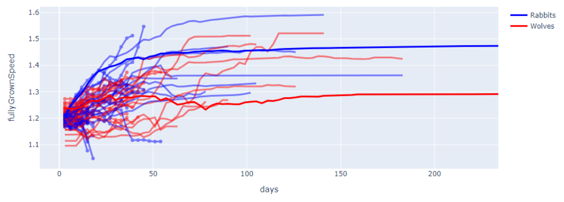


Figure 5.5: The number of animals and plants in the small world with the initial values of rabbits, plants, and wolves being 100, 200, and 25.

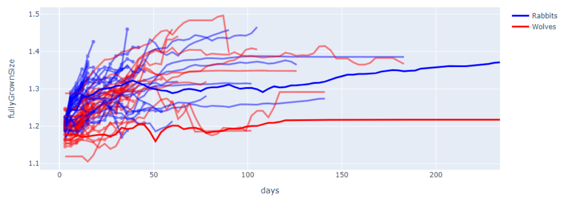
The traits of the animals are displayed in 5.6.



(a) The percentage of brainpower utilized for the vision and hearing.



(b) The average speed of the animals.



(c) The average size of the animals.

Figure 5.6: The traits of the animals with the initial amount of rabbits, plants, and wolves being 100, 200, and 25 respectively.

5.2.2 Equilibrium with 400 Plants in the Medium World

The simulations presented in this subsection were stable, compared to the instances run on the small and large world, with several that ran for more than 400 days. As seen in Fig. 5.7, in a run of 1000 days, the system ended up in an equilibrium. The figure also displays similarities with a Lotka-Volterra model, in that there is a cyclic trend in the relation between the number of plants, rabbits, and wolves. As the amount of rabbits increase, the number of plants decreases. When the rabbit population is large, the population of wolves starts to increase until the rabbit population sharply decreases. Over the whole duration of the simulation, the Pearson correlations are the following:

- Wolves and rabbits: -0.49.
- Plants and rabbits: -0.92.
- Plants and wolves: 0.50.

The animals' traits were logged in the simulation which is displayed in Fig. 5.8. It can be seen from the figures that it is favorable for both species to be both larger and faster. Both species reach their full speed factor potential at 1.55. The rabbits did not get as big as the wolves did with a consistently larger size in most of the runs.

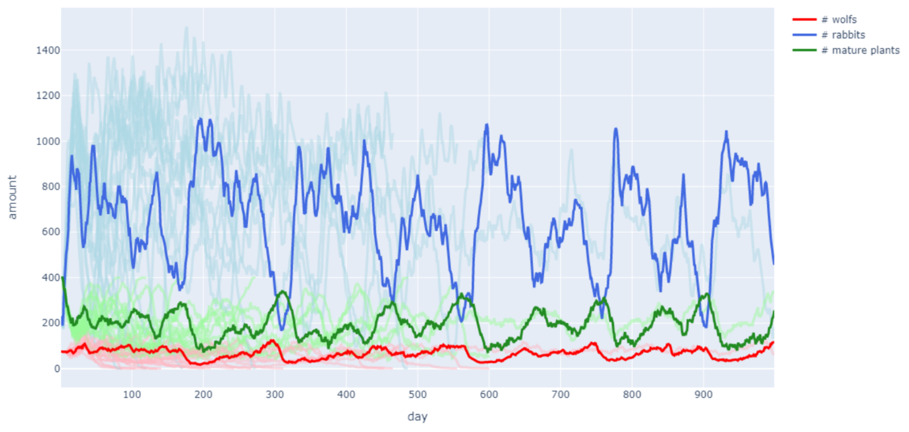
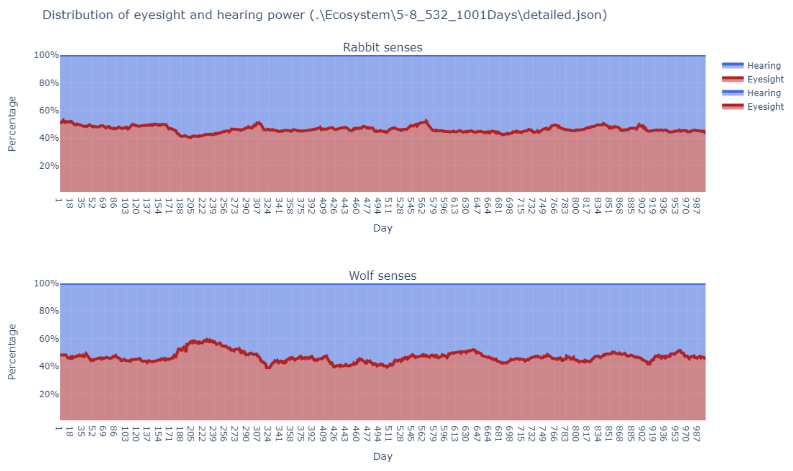
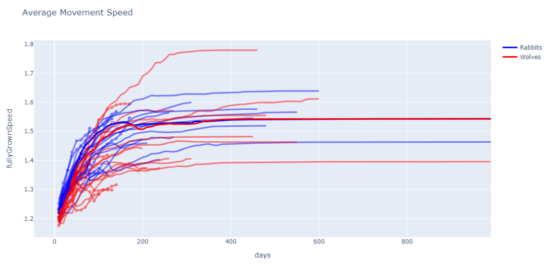


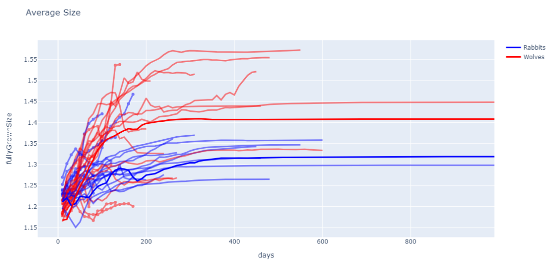
Figure 5.7: The number of animals in the medium world with 400 plants. The number of rabbits alternates roughly between 200 and 1000 while the amount of wolves alternates between 30 and 115. The initial number of rabbits, plants, and wolves are 230, 400, and 75 respectively.



(a) The percentage of brainpower utilized for the vision and hearing. The gap in the upper graph is caused by a plotting error.



(b) The average speed of the animals.



(c) The average size of the animals

Figure 5.8: The traits of the animals for the instances with the initial number of rabbits, plants, and wolves being 230, 400, and 75 respectively.

5.2.3 Equilibrium with 800 Plants in the Large World

In this equilibrium, the Pearson correlations are the following:

- Wolves and rabbits: -0.56.
- Plants and rabbits: -0.79.
- Plants and wolves: 0.83.

The instances executed in the large world ended quite fast compared to the instances in the other worlds. It is therefore difficult to draw any type of conclusions from these results. A short discussion can be found in Subsection 6.3.3.

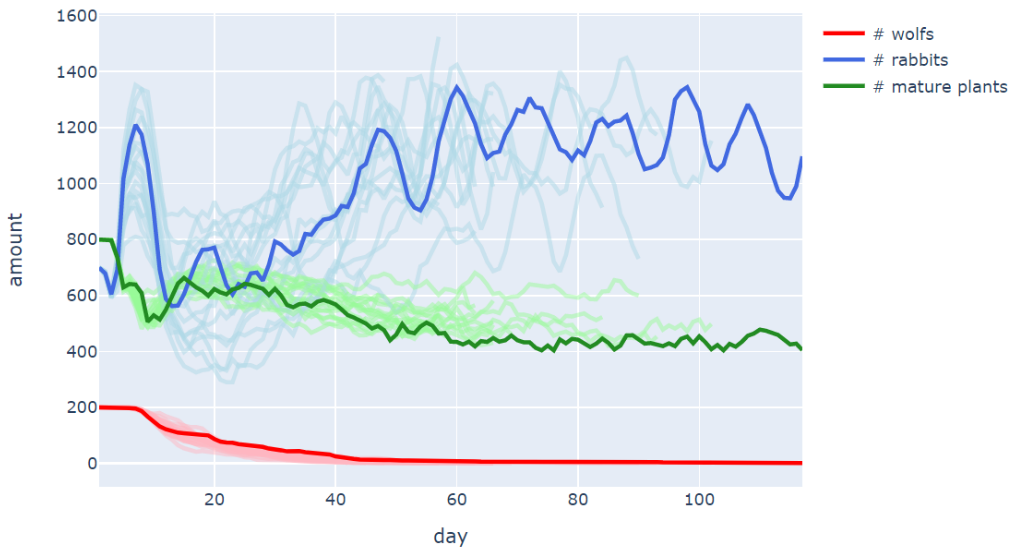
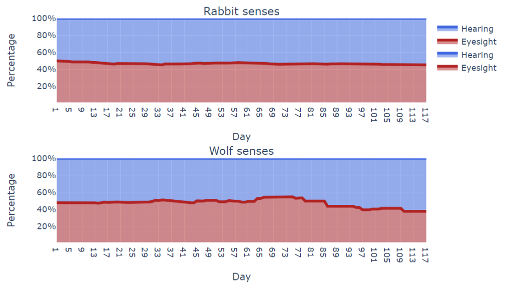


Figure 5.9: The number of animals in the large world with 800 plants. The initial number of rabbits, plants and, wolves are 700, 800, and 200 respectively.

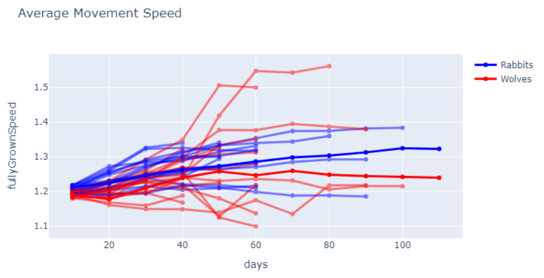
5.3 Genetic Comparison

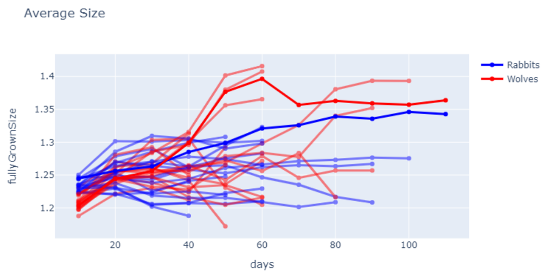
This part will cover medium world instances with 400 plants, where there was no genetic algorithm implemented. Thus the wolves and rabbits had equal speed and size, consequently making the only difference in desire for food their specific factors (as explained in Section 4.1.4). Unlike with the genetic algorithm implemented, the majority of instances with no genetic algorithm ended quickly with the wolves going extinct, as can be seen in Fig. 5.11. The highlighted curve is the longest simulation, which went on for 363 days with the following correlations:

- Wolves and rabbits: -0.81
- Plants and rabbits: -0.97
- Plants and wolves: 0.78



(a) The percentage of brainpower utilized for the vision and hearing.





(b) The average speed of the animals. **(c)** The average size of the animals.

Figure 5.10: The traits of the animals with the initial number of rabbits, plants, and wolves being 700, 800 and, 200 respectively.

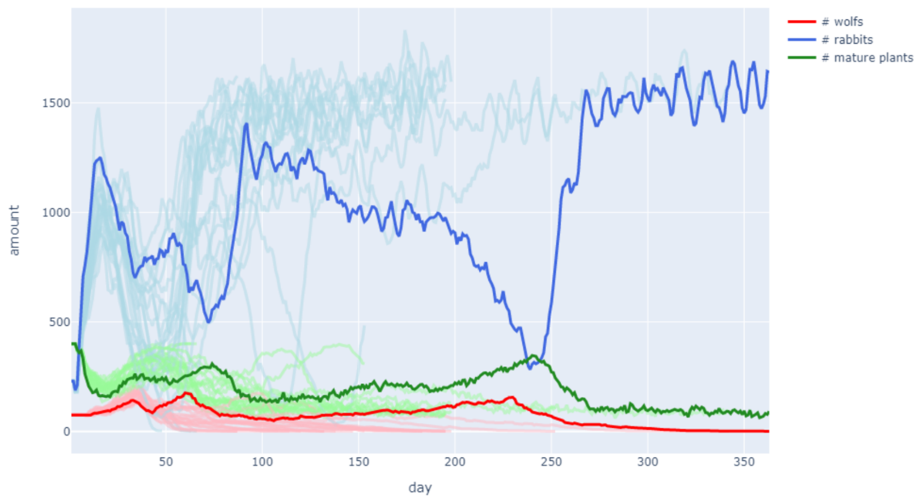


Figure 5.11: The number of animals in an iteration without genetic algorithms implemented.

6

Discussion

This section covers the discussion of the results, a discussion about performance, societal and ethical aspects regarding the project, as well as a discussion about potential improvements that can be made to the project.

6.1 Randomness Effects the Results

Every instance of the simulation produces different results, even on the same computer, since the mutations that occur in the animals are random. It could also be the case that the simulation runs differently depending on hardware but it is difficult to measure the magnitude of this factor because of the inherent randomness in the simulation. Nevertheless, because of time constraints, it was deemed necessary to gather as much data as possible from several computers, rather than presenting a low amount of data coming from one computer.

This randomness is why it is important to run the simulation several times when trying out values. Multiple instances with the same starting values give more certainty to the results. This enables us to look closer at certain simulations that behave interestingly and place them in a context.

6.2 Performance

When there is a large number of animals in the simulation, it is common for them to get stuck in place. This is because the NavMesh cannot handle the path calculations for all the entities concurrently. This causes the animals to stop in place while running and as a result, they starve to death. This causes the animals to get worse at surviving when there is a large amount of entities in the simulation. This is one of the reasons why it is difficult to reach an equilibrium in the large world. This problem was not discovered until it was too late to change the navigation system. If this information would have been known at an earlier stage of development, another approach would have been selected for the navigation which is further discussed in 6.5.

Another issue is related to when the FPS is low in the simulation is that the collision triggers (hearing and vision) seem to work inadequately. This problem has most likely negatively affected the results from the simulation since animals have been observed to die when they potentially could have reached the resources needed to survive. A discussion about potential fixes to this problem can be found in 6.5.

6.3 Analyzing the Results

The compiled data retrieved from the simulation gave some results that were expected and some that were not. The following subsections will present a discussion about these results.

6.3.1 Trends for Hearing and Vision

For all simulations, there did not seem to be much preference between either hearing or vision. This could be because the changes reflect small changes. For most instances of the simulation, the plants are not that scarce and the animals move around a lot combined with the animals' memory of food and water. Even a small hearing range or small vision area is enough to find mates and plants just as easily as they would otherwise because of their memory. There might have been more of a preference if the resources were rarer. This result was somewhat surprising since it was presumed that it would be more advantageous for the rabbits to have a larger hearing range. As for the wolves, it was presumed that their vision and hearing ranges would be somewhat evenly divided.

This could also be the cause for why the percentage of hearing and vision for the rabbits is close to 50%. Since there does not seem to be much of an advantage for hearing nor vision, the distribution is even. And since rabbits often exist in a larger amount, the sample size is higher and that is why it is closer to the 50% distribution which is expected if it was completely randomly distributed. The wolves are, in the instances of the simulation, often fewer and thus have a smaller sample size which could explain the greater variation of the vision and hearing graphs.

6.3.2 Simulation of Only Rabbits

Based on the results, it is not surprising that the rabbits thrive in instances where there are no predators. Fig. 5.1 indicates that for every start value, whether it be 50, 200, or 400 plants, the rabbits do not die out a single time over all the instances being run. From these graphs, it is clear that there is a correlation between the number of plants and the number of rabbits in the ecosystem. This was expected since a larger amount of plants should be able to support a greater amount of rabbits.

The size of the rabbits had an increasing trend in the simulations with 200 and 400 plants, unlike the simulations with 50 plants where there was no apparent trend. The reason for this could be the the amount of rabbits in the 50 plants simulations were too low to produce any apparent trend. Another reason could be that the amount of food is so low that it is not able to support the hunger for the larger rabbits which need more food.

In general, the speed of the rabbits (Fig. 5.2) is higher in the simulations with fewer plants. This is most likely due to more competition for plants, where the faster rabbits manage to get to them quicker.

As seen in Fig. 5.3a, with 50 plants it does not seem to be favorable to be big or small but the size seems to increase when there are more plants available. The reason for this could be because a larger size means that the animals do not get hungry as fast, but they need to eat more to be saturated. This is expected to be better as the amount of rabbits increases. The reason why the rabbits tend to get larger in the instances with more plants is that an increase in plants leads to an increase in rabbits. Thus the competition for the food increases, leading to larger rabbits being able to hold out longer without food.

In Fig. 5.1b the number of sustained rabbits goes down. This happened in all the instances but was most prominent in the highlighted run shown in the graph. Looking at the speed of the rabbits in Fig. 5.2b, it was much higher compared to the other instances with 200 plants. This is likely because when a majority of rabbits increased their speed, it made the rabbits race for food more and in turn wanting to get even faster. This selfish behavior made the rabbits having to eat more often and compete with each other which caused the total number of rabbits that the environment can support lower. Hence the more significant drop in amount of rabbits in Fig. 5.1b.

6.3.3 Equilibrium

As seen in Fig. 5.7, an equilibrium that lasts over 1000 days with both wolves and rabbits could only be reached in the medium world. This could be because the start values used in these instances of the medium world were enough to support a number of entities small enough for the NavMesh to function reasonably well. The medium world also has more lakes to which the rabbits can migrate if the number of wolves increases, which enables them to survive better. The Pearson coefficient for plants and rabbits indicates a strong correlation between them, as was expected. However, a stronger correlation between the wolves and the rabbits was expected. This could explain why in Fig. 5.7, the amount of wolves does not perfectly follow the trend of the rabbits as well as the number of plants does.

In order for the large world to reach an equilibrium, it is required that there are many more entities present than in the other worlds. This in turn will enable the wolves to find both prey and mates to create offspring to populate the vast world. This causes the performance issues related to the NavMesh, which is discussed in 6.2, which subsequently reduces the survivability of the wolves. This is most likely the reason why the shortest equilibriums come from the large world, see Fig. 5.9. The poor performance can also be deduced in the Pearson coefficients. The correlation between the plants and the rabbits, as well as the correlation between wolves and rabbits, is noticeably lower compared to the instances run on the medium world and the instances without genetics.

The small world, on the other hand, has the issue that the wolves make the rabbits go extinct if they are too many at any time. The reason for this is likely because there is nowhere for the rabbits to run due to the limited space. There are only two water sources in the small world. Because of this, the rabbits do not have as many options to find a water source free from wolves. Similarly, if there are too few wolves there is a low probability for the wolves to find any mates. The conclusion is that the small world seems to have a narrow span for which the number of wolves needs to be in for the equilibrium to persist. Out of the three equilibriums, the small world indicates the strongest correlation between wolves and rabbits. The correlation is quite unexpected as the resulting graph does not seem to indicate a strong correlation. The relatively high correlation between wolves and plants is likely due to the fact that an increased number of plants leads to an increased number of rabbits, subsequently leading to an increase in the number of wolves. Similarly to the other equilibriums, a high correlation between plants and rabbits is expected.

It might seem like the simulations could have started out with more wolves since it usually is the wolves that die out. However, all simulations have been tried with different starting values and increasing the number of wolves, drastically reduces the ability of the rabbits to survive. This is partly because of the wolves eating the rabbits but also because of the rabbits not being able to reach their water or food because of nearby wolves that force them into the flee state.

6.3.4 Trends for Size and Speed

The animals tend to increase in speed and size in most of the instances of the simulation. The speed increase is likely because the rabbits all race to reach the food and to escape the wolves. To catch the rabbits, the wolves also need to increase their speed. The reason why the animals' speed does not reach incredulous heights is presumably due to the fact that the cost of being fast increases and outweighs the benefits of being fast. The increase in size is probably to mitigate the negative effects of the speed increase, with the cost that the animals need to eat more to reach full saturation.

It was noted that the rabbits evolve faster than wolves since their speed and size tend to increase at a faster rate. This is believed to be a result of the rabbits having a shorter reproduction cycle and more offspring with each litter, as well as the rabbits existing in greater numbers compared to the wolves.

6.3.5 Genetics

The difference between instances of the simulation with genetics and the ones without genetics is easily distinguishable. Normally, in the no-genetics instances, the wolves almost or completely eradicated the rabbits. This was most likely due to the species having the same speed. Because of this, the wolves often went extinct just after roughly 150 days due to the lack of food. Meanwhile, in the instances with genetics, faster rabbits were fitter to survive. Consequently, the wolves did not kill the rabbits to the same extent, which lead to the equilibrium lasting for longer. As one can see, the simulations with genetics generally held on for 400 days or longer.

The Pearson coefficient for the instances without genetics shows that the correlation between plants and rabbits is somewhat similar to the instances with genetics, and this is expected. The coefficient for wolves and rabbits is stronger than the instances with genetics. This result is also expected since the wolves are able to keep pace with the rabbits as the rabbits do not evolve an increase in speed. The increase in correlation between plants and wolves follows as the correlation between both wolves and plants, and plants and rabbits, have increased.

In short, the results given by the simulation indicate that it is possible to reach an equilibrium, particularly with the start values used in the medium world. The reason for this is most likely due to two reasons. First, the medium world contains more water sources and is thus large enough for the animals to seek out new resources if needed. Second, the values used were large enough to sustain the number of rabbits and wolves, without them being so many that the NavMesh got negatively impacted. Furthermore, the differences in the results between the instances with and without the genetics included, indicate that some simulations without genetics were not stable but managed to be with genetics.

6.4 Societal and Ethical Aspects

Depending on the scope of a project in Unity, developers may or may not choose to create all necessary assets by themselves. As mentioned in subsection 2.4, when creating a product in Unity, the developer has access to the Unity asset store. Here, the developer can purchase the license to download assets to incorporate into the product[66]. Furthermore, code repositories such as GitHub and Bitbucket are two other alternatives where developers have the opportunity to use scripts written by other developers. When incorporating such material, it is important to know the formalities regarding the rights to distribute the final product. This is important because creators spend a considerable amount of time creating assets, thus, their work should be accredited appropriately.

Regarding simulations in general, they are of great use in different types of academic and scientific fields. Simulations can be used for educational purposes and aid researchers in their work by simulating a wide variety of different systems.

The simulation presented in this report is too simplified and inaccurate to be used for any sort of real-life approximation of an ecosystem. As mentioned in chapter 3, simulations, in general, tend to focus on a small section of a greater subject: instead of simulating an entire ecosystem, researchers tend to focus on simulating weather or the population of only one species and do so accurately.

However, it is important to remember that simulations are approximations of real life. An ecosystem depends on many factors that may be very difficult to foresee. Simulations depicting ecosystems should therefore not be seen as an exact representation of their real-life counterparts, but instead as good enough representations.

Thus, decisions based on conclusions drawn from results coming from a simulation should be taken with this into account.

6.5 Improvements

If time would not have been a limit, there are a number of improvements and additional content that could be added.

The first improvement would be to implement a 3D matrix to be used for animal movement across the terrain. The 3D matrix would encompass the terrain, with positions that animals can go to mapped out within the matrix. These positions would correspond to a position in the terrain. By switching the navigation of the animals from the NavMesh to a 3D matrix, it would allow for more hilly terrain, something that is not possible when using the NavMesh since the animals get stuck in the ground on steep inclines. However, the current implementation is more life-like as animals are not stuck to a grid.

Furthermore, the matrix implementation would potentially increase performance for several reasons. Firstly, all logic that is currently performed by the Unity colliders could instead be transcribed to 2D-matrix logic. Although the logic would not be as precise as with the colliders, it would most likely not make a noticeable difference in the outcome of the simulation. An example of how this logic would work in 2D is pictured in Fig. 6.1. Secondly, since the matrix would be very simple compared to the Navmesh terrain, the pathfinding would have to calculate far fewer possible paths.

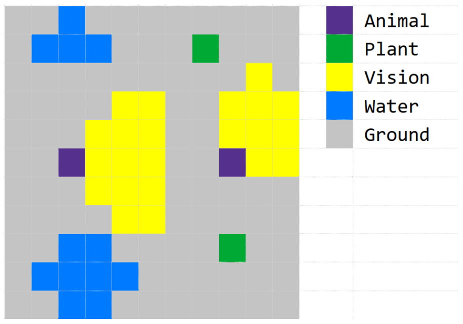


Figure 6.1: An example illustration of the ecosystem modeled as a 2D matrix.

Another possible improvement would be to fine-tune the behaviors and traits of the animals. This could include adding dens and burrows for the animals to seek shelter in, include pack behavior for both rabbits and wolves, and improve the vision areas for the animals, to further emulate real-life behavior. Furthermore, another improvement that could be implemented is that all animals should have a different amount of stamina instead of just 0 to 100 percent where all the animals of the same species have the same decrease and increase amount in their different states.

Several possible improvements can be made to the plants. One such improvement would be to add genes to the plants. This could potentially have an effect on where plants choose to grow, how fast they grow or die, how many offspring each plant can have, if the plants become poisonous or not, etc. Other types of plants could also be introduced. To further build on this idea, making it possible to crossbreed between different species of plants could possibly give way to new kinds of plant species. Lastly, another minor improvement could be to make the plants disappear when fully consumed. Then having new plants immediately reappear in the vicinity of the old plant. This would not have a significant impact on the simulation but would give the simulation a more life-like appearance.

Using the start screen (see Fig. 4.7) a user can set their own parameters of the ecosystem. The idea of users impacting the ecosystem can be further built upon. For example, by further gamifying the ecosystem, it would be possible to give the user more control: being able to change the outcome of the ecosystem in real-time by adding and removing resources and animals on a whim.

7 Conclusion

The aim of the project was to study what effect the number of resources, individual characteristics, and behaviors have on the natural selection of simulated animals. Working with these questions has given us more insight into the intricate balance that is required for an ecosystem to stay stable. This has highlighted the importance for a species to be able to adapt over generations through the phenomenon of natural selection and evolution. The results indicate that the simulated ecosystem managed to stay stable with evolution while it did not without evolution, given the same starting values. Moreover, the results show that it is possible to reach an equilibrium with correct start values and in a reasonably sized world. The limitation to reach an equilibrium in a larger world is caused by the NavMesh used to move the animals around. Meanwhile, the limitation in the small world is likely due to a lack of water sources and space for the animals to move around.

A more stable relationship between the species in the simulation is something that could be useful to further develop given more time. An improved equilibrium would give the animals a longer time to evolve. This in turn would give greater accuracy to the collected data, thus increasing the accuracy of the results that show what traits and characteristics are more favorable.

The genetic algorithm implemented made it possible for the animals to change size, speed, hearing, and eyesight. This resulted in the animals mostly becoming both larger and faster to be able to survive better. In most cases, there was no clear advantage for the animals having better hearing or eyesight.

To conclude this project, an ecosystem has been created where the number of resources and the characteristics and behaviors of the animals affects the traits of the animals over time. The model also includes an implemented genetic algorithm that can affect the traits of the animals' offspring. This was proven to keep the ecosystem stable by running comparative instances without the genetic algorithm active. Further improvements need to be implemented in order to support a larger amount of resources and animals.

Bibliography

[1] C. Nolet, *Quick outline | particles/effects | unity asset store*, assetstore.unity.com. [Online]. Available: `https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488` (visited on 04/23/2021).

[2] V. Team, *Low poly nature pack (lite) | 3d landscapes | unity asset store*, assetstore.unity.com. [Online]. Available: `https://assetstore.unity.com/packages/3d/environments/landscapes/low-poly-nature-pack-lite-40444` (visited on 04/23/2021).

[3] Yasirkula, *Text to textmesh pro upgrade tool | utilities tools | unity asset store*, assetstore.unity.com. [Online]. Available: `https://assetstore.unity.com/packages/tools/utilities/text-to-textmesh-pro-upgrade-tool-176732` (visited on 04/23/2021).

[4] Unity, *Simulating wind in urp (shader graph tutorial)*, www.youtube.com, Jul. 2020. [Online]. Available: `https://www.youtube.com/watch?v=ZsoqrHHtg4I` (visited on 03/24/2021).

[5] *Brackeys*, YouTube. [Online]. Available: `https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA`.

[6] N. M. Games, *Hull outline shader in unity urp using renderer features and culling! 2020.3 | game dev tutorial*, www.youtube.com, Nov. 2020. [Online]. Available: `https://www.youtube.com/watch?v=1QPA3s0S3Oo` (visited on 03/30/2021).

[7] gamesplusjames, *Creating a cel-shading toon shader*, www.youtube.com, Oct. 2019. [Online]. Available: `https://www.youtube.com/watch?v=3SvyJrENsgc` (visited on 03/22/2021).

[8] D. Ilett, *Transparency dithering in unity shader graph [tutorial]*, www.youtube.com, Apr. 2020. [Online]. Available: `https://www.youtube.com/watch?v=VG-Ux8RHMoA` (visited on 04/03/2021).

[9] *Unity | water with shader graph tutorial – protokoll studio*, area51.protokoll-studio.com. [Online]. Available: `https://area51.protokoll-studio.com/unity-water-with-shader-graph-tutorial` (visited on 03/23/2021).

[10] N. G. Society, *Ecosystem*, National Geographic Society, Oct. 2012. [Online]. Available: `https://www.nationalgeographic.org/encyclopedia/ecosystem/`.

[11] C. Darwin, *On the Origin of Species*. Hurst Company, 1912, pp. 54–68. [Online]. Available: `http://darwin-online.org.uk/converted/pdf/1912_Origin_F518.pdf` (visited on 05/04/2021).

[12] W. at Naturhistoriska riksmuseet, *Darwin och evolutionsteorin*.

[13] C. Nunez, *Deforestation and its effect on the planet*, Environment, Feb. 2019. [Online]. Available: `https://www.nationalgeographic.com/environment/article/deforestation` (visited on 05/13/2021).

[14] R. Lindsey and L. Dahlman, *Climate change: Global temperature | noaa climate.gov*, Climate.gov, Aug. 2020. [Online]. Available: `https://www.climate.gov/news-features/understanding-climate/climate-change-global-temperature` (visited on 05/11/2021).

[15] M. Mignano, *Finite state machine for game developers*, Game Developer Tips, Jul. 2016. [Online]. Available: `https://gamedevelopertips.com/finite-state-machine-game-developers/` (visited on 04/13/2021).

[16] ElProCus, *Finite state machine (fsm) : Types, properties, diagram, design and applications*, ElProCus - Electronic Projects for Engineering Students, Mar. 2019. [Online]. Available: `https://www.elprocus.com/finite-state-machine-mealy-state-machine-and-moore-state-machine/` (visited on 06/01/2021).

[17] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, p. 66, 1992, ISSN: 00368733, 19467087. [Online]. Available: `http://www.jstor.org/stable/24939139`.

[18] K. Codes, *Genetic algorithms explained by example*, www.youtube.com. [Online]. Available: `https://www.youtube.com/watch?v=uQj5UNhCPuo&t=434s` (visited on 05/05/2021).

[19] L. Haldurai, T. Madhubala, and R. Rajalakshmi, "A study on genetic algorithm and its applications," *International Journal of Computer Sciences and Engineering*, vol. 4, no. 10, pp. 139–140, 2016.

[20] *Genetic Algorithms with Multi-parent Recombination*, Oct. 1994, pp. 78, 79, 80, 81, ISBN: 978-3-540-58484-1. DOI: `10.1007/3-540-58484-6_252`. [Online]. Available: `https://www.researchgate.net/publication/220701605_Genetic_algorithms_with_multi-parent_recombination` (visited on 04/18/2021).

[21] M. Emmerich, O. M. Shir, and H. Wang, "Evolution strategies," in *Handbook of Heuristics*, R. Martí, P. Panos, and M. G. C. Resende, Eds. Cham: Springer International Publishing, 2018, pp. 1–31, ISBN: 978-3-319-07153-4. DOI: `10.1007/978-3-319-07153-4_13-1`. [Online]. Available: `https://doi.org/10.1007/978-3-319-07153-4_13-1`.

[22] T. E. of ScienceDirect, *Lotka-volterra model*, ScienceDirect, May 2021. [Online]. Available: `https://www.sciencedirect.com/topics/earth-and-planetary-sciences/lotka-volterra-model` (visited on 05/07/2021).

[23] AspidistraK, *Lotka volterra dynamics*, Feb. 2017. [Online]. Available: `https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations#/media/File:Lotka_Volterra_dynamics.svg` (visited on 05/18/2021).

[24] J. S. Milton and J. C. Arnold, *Introduction to probability and statistics : principles and applications for engineering and the computing sciences*. Boston, Mass. Mcgraw-Hill [20]08, 2003.

[25] U. Technologies, *Solutions | unity*, unity.com. [Online]. Available: `https://Unity.com/solutions` (visited on 04/08/2021).

[26] *Most used engines*, itch.io. [Online]. Available: `https://itch.io/game-development/engines/most-projects` (visited on 04/16/2021).

[27] ——, *Games made with unity*, unity.com. [Online]. Available: `https://unity.com/madewith` (visited on 04/14/2021).

[28] E. Games, *Unreal engine*, @UnrealEngine. [Online]. Available: `https://www.unrealengine.com/en-US/` (visited on 05/06/2021).

[29] *Unreal engine | what is unreal engine 4*, @UnrealEngine, 2019. [Online]. Available: `https://www.unrealengine.com/en-US/` (visited on 05/2020).

[30] G. Engine, *Godot engine - free and open source 2d and 3d game engine*, Godotengine.org. [Online]. Available: `https://godotengine.org/` (visited on 05/06/2021).

[31] E. Leblond, *Touilleman/godot-python*, GitHub, May 2021. [Online]. Available: `https://github.com/touilleMan/godot-python` (visited on 05/06/2021).

[32] U. Technologies, *Unity - manual: Terrain tools*, docs.unity3d.com. [Online]. Available: `https://docs.unity3d.com/Manual/terrain-Tools.html` (visited on 04/19/2021).

[33] ——, *Unity - manual: Navmesh building components*, docs.unity3d.com. [Online]. Available: `https://docs.unity3d.com/Manual/NavMesh-BuildingComponents.html` (visited on 04/14/2021).

[34] ——, *Unity - manual: Building a navmesh*, docs.unity3d.com. [Online]. Available: `https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html` (visited on 04/14/2021).

[35] ——, *Unity - manual: Navmesh surface*, docs.unity3d.com. [Online]. Available: `https://docs.unity3d.com/Manual/class-NavMeshSurface.html` (visited on 04/14/2021).

[36] ——, *Unity - manual: Inner workings of the navigation system*, docs.unity3d.com, Apr. 2021. [Online]. Available: `https://docs.unity3d.com/Manual/nav-InnerWorkings.html` (visited on 04/29/2021).

[37] R. B. Games, *Red blob games: Introduction to a**, Redblobgames.com, Jun. 2020. [Online]. Available: `https://www.redblobgames.com/pathfinding/a-star/introduction.html` (visited on 04/29/2021).

[38] J. van den Berg, M. Lin, and D. Manocha, *Reciprocal velocity obstacles for real-time multi-agent navigation*, gamma.cs.unc.edu, 2008. [Online]. Available: `https://gamma.cs.unc.edu/RVO/` (visited on 04/29/2021).

[39] U. Technologies, *Unity - manual: Colliders*, Unity3d.com, 2019. [Online]. Available: `https://docs.unity3d.com/Manual/CollidersOverview.html` (visited on 05/13/2021).

[40] ——, *Unity - scripting api: Particlesystem*, docs.unity3d.com. [Online]. Available: `https://docs.unity3d.com/ScriptReference/ParticleSystem.html` (visited on 04/19/2021).

[41] ——, *Unity - manual: Render pipelines*, Unity3d.com, 2020. [Online]. Available: `https://docs.unity3d.com/Manual/render-pipelines.html` (visited on 04/21/2021).

[42] ——, *Unity - manual: Using the high definition render pipeline*, docs.unity3d.com. [Online]. Available: `https://docs.unity3d.com/Manual/high-definition-render-pipeline.html` (visited on 04/19/2021).

[43] V. M. Cirne, *Everything you need to know about post-processing in unity*, Medium, Sep. 2019. [Online]. Available: `https://medium.com/@vinicius.cirne/everything-you-need-to-know-about-post-processing-in-unity-72b09534ac38` (visited on 05/14/2021).

[44] U. Technologies, *Unity - manual: Post-processing*, docs.unity3d.com, May 2021. [Online]. Available: `https://docs.unity3d.com/Manual/PostProcessingOverview.html` (visited on 05/14/2021).

[45] ——, *Unity - manual: Post processing*, Unity3d.com, 2021. [Online]. Available: `https://docs.unity3d.com/Manual/PostProcessingOverview.html` (visited on 04/29/2021).

[46] N. Lassabe, H. Luga, and Y. Duthen, "Evolving creatures in virtual ecosystems," vol. 4282, Jan. 2006, pp. 11–20, ISBN: 978-3-540-49776-9. DOI: `10.1007/11941354_2`.

[47] J. Yamada, J. Shawe-Taylor, and Z. Fountas, "Evolution of a complex predator-prey ecosystem on large-scale multi-agent deep reinforcement learning," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8. DOI: `10.1109/IJCNN48605.2020.9206765`.

[48] A. Zamuda, J. Brest, N. Guid, and V. Zumer, "Modelling, simulation, and visualization of forest ecosystems," in *EUROCON 2007 - The International Conference on "Computer as a Tool"*, 2007, pp. 2600–2606. DOI: `10.1109/EURCON.2007.4400683`.

[49] H. Zhao, X. Jin, and J. Shen, "Simple and fast terrain rendering using graphics hardware," in *Advances in Artificial Reality and Tele-Existence*, Z. Pan, A. Cheok, M. Haller, R. W. H. Lau, H. Saito, and R. Liang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 715–723, ISBN: 978-3-540-49779-0.

[50] L. Yao, L. Tang, C. Chen, and J. Sun, "Animating grass in real-time," in *Advances in Artificial Reality and Tele-Existence*, Z. Pan, A. Cheok, M. Haller, R. W. H. Lau, H. Saito, and R. Liang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 724–731, ISBN: 978-3-540-49779-0.

[51] Z. Xin, l. Fengxia, and Z. ShouYi, "Real-time and realistic simulation of large-scale deep ocean surface," Jan. 2006, pp. 686–694, ISBN: 978-3-540-49776-9. DOI: `10.1007/11941354_71`.

[52] S. Lague, *Coding adventure: Simulating an ecosystem*, www.youtube.com, Jul. 2019. [Online]. Available: `https://www.youtube.com/watch?v=r_It_X7v-1E`.

[53] E. Arts, *The sims-spel – officiell ea-webbplats*, Electronic Arts Inc., Oct. 2016. [Online]. Available: `https://www.ea.com/sv-se/games/the-sims` (visited on 04/26/2021).

[54] ——, *Spore™*, www.spore.com, 2009. [Online]. Available: `https://www.spore.com/` (visited on 04/26/2021).

[55] ThinMatrix, *Equilinox*, equilinox.com, 2019. [Online]. Available: `https://equilinox.com/` (visited on 04/26/2021).

[56] J. Helps, *Evolution - youtube*, www.youtube.com. [Online]. Available: `https://www.youtube.com/playlist?list=PLKortajF2dPBWMIS6KF4RLtQiG6KQrTdB` (visited on 04/26/2021).

[57] I. C. Education, *What is machine learning?* www.ibm.com, Jul. 2020. [Online]. Available: `https://www.ibm.com/cloud/learn/machine-learning` (visited on 05/10/2021).

[58] S. Z. Enam, *Why is machine learning 'hard'?* ai.stanford.edu, Nov. 2016. [Online]. Available: `https://ai.stanford.edu/~zayd/why-is-machine-learning-hard.html` (visited on 05/10/2021).

[59] M. L. Jeunesse, *10 cute animals you didn't know were cannibalistic*, Business Insider, Dec. 2018. [Online]. Available: `https://www.businessinsider.com/animals-that-are-cannibals-2018-12?r=US&IR=T#occasionally-rabbits-engage-in-cannibalistic-behavior-5` (visited on 05/11/2021).

[60] *Breeding like rabbits: A primer on rabbit reproduction*, Pets In Stitches, Jun. 2018. [Online]. Available: `https://petsinstitches.com/blog/rabbit-reproduction/` (visited on 05/07/2021).

[61] U. Technologies, *Unity - scripting api: Collider.ontriggerenter(collider)*, Unity3d.com, 2021. [Online]. Available: `https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html` (visited on 04/21/2021).

[62] T. E. of Encyclopaedia, *Kinetic energy*, Britannica, Nov. 2019. [Online]. Available: `https://www.britannica.com/science/kinetic-energy` (visited on 05/07/2021).

[63] A. Bradford, *Rabbits: Habits, diet other facts*, Live Science, Mar. 2017. [Online]. Available: `https://www.livescience.com/28162-rabbits.html` (visited on 05/10/2021).

[64] A. Expedition, *Gray wolf facts information*, American Expedition, 2015. [Online]. Available: `https://forum.americanexpedition.us/gray-wolf-information-facts-photos-and-artwork` (visited on 05/10/2021).

[65] *Plotly python graphing library*, plotly.com. [Online]. Available: `https://plotly.com/python/` (visited on 04/29/2021).

[66] *Asset store terms of service and eula*, Unity, Jul. 2020. [Online]. Available: `https://unity3d.com/legal/as_terms` (visited on 04/19/2021).

A

Performance

A simulation with a wide range of amount of animals was picked. The simulation was run on the medium sized world and the data is displayed in Fig A.1. From there we can extract that the simulation can run with an interactive fps over 30 for around 1200 animals simultaneously.

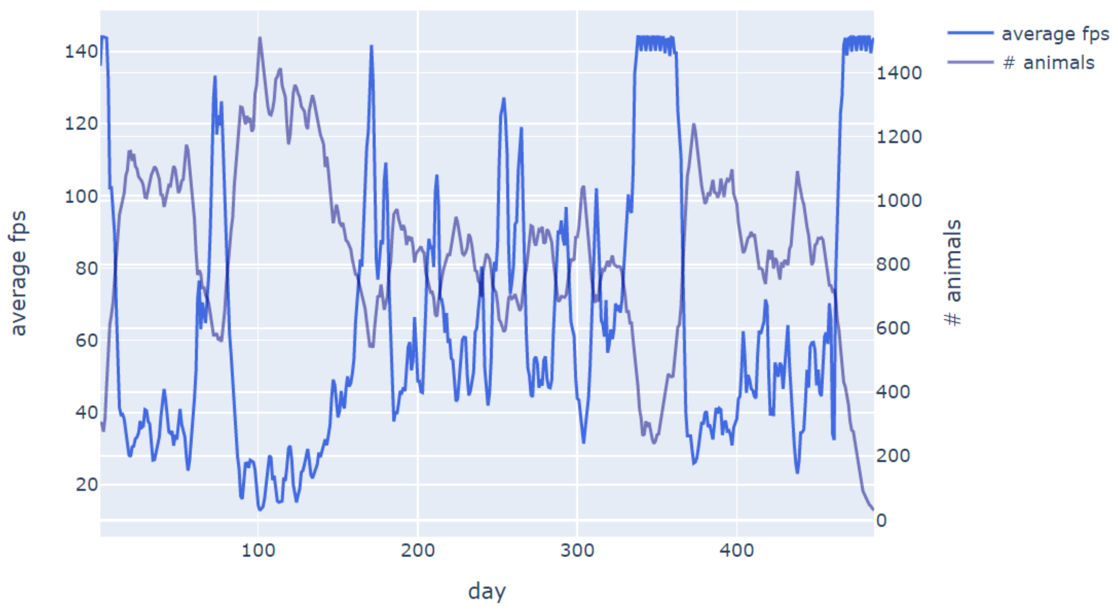


Figure A.1: The average frames per second for every day.

B

Finite State Machine

The state machine for the wolves and rabbits can be seen in Fig. B.1 and B.2

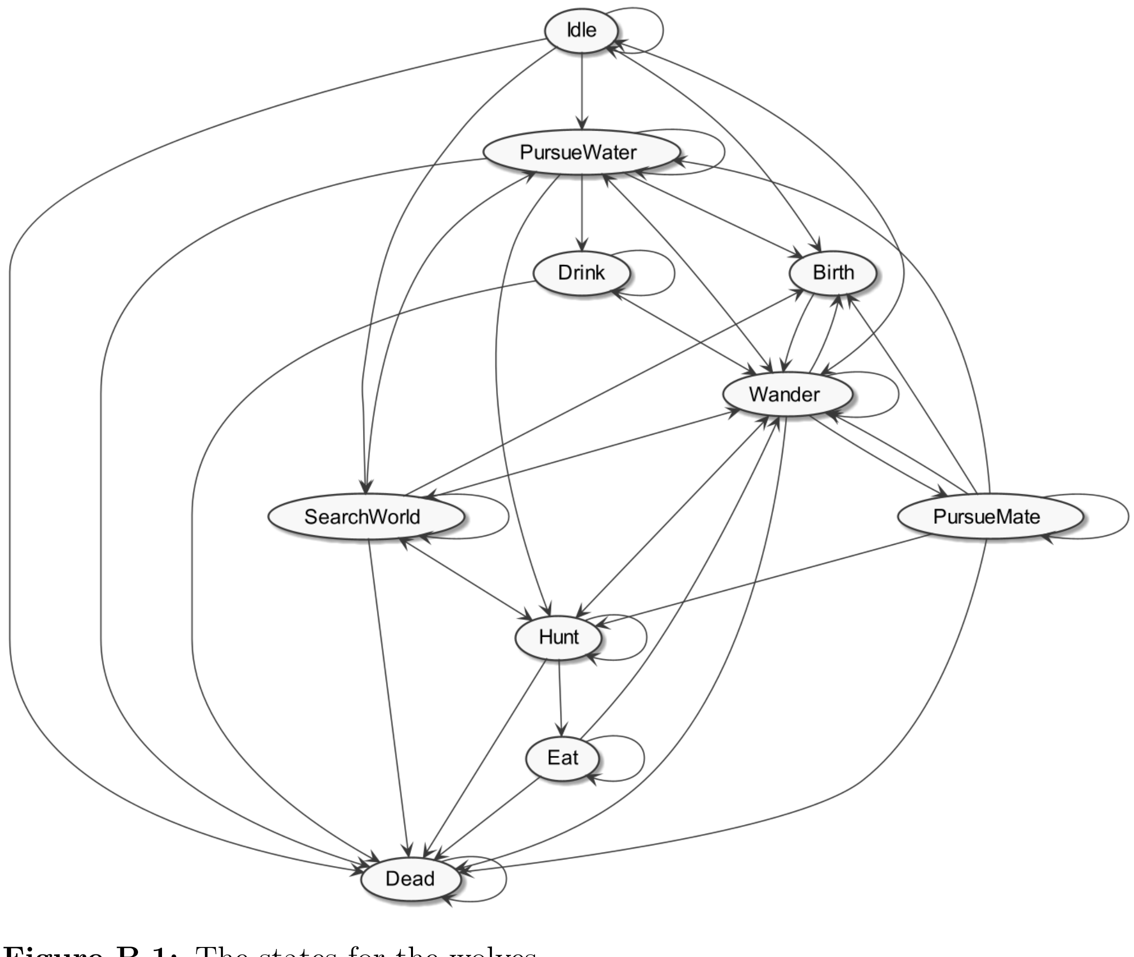


Figure B.1: The states for the wolves.

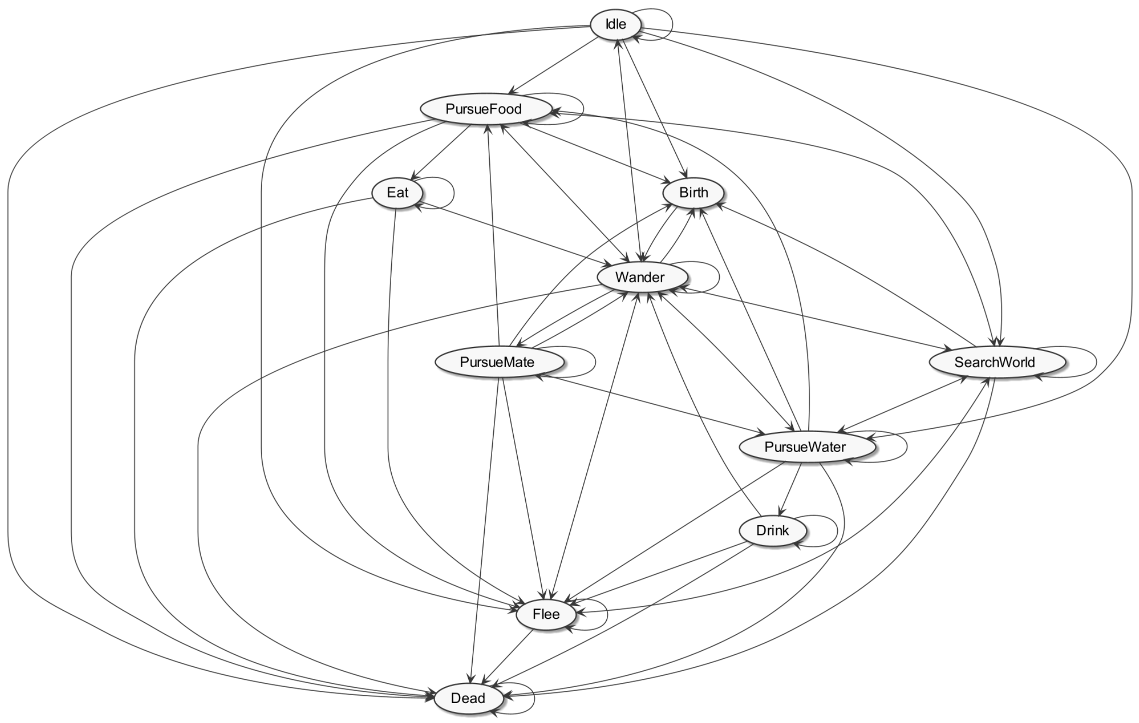


Figure B.2: The states for the rabbits.